

ESP32-C5

技术参考手册 预发布 v0.1

PRELIMINARY



ESPRESSIF

关于本文档

ESP32-C5 技术参考手册面向使用 ESP32-C5 系列产品进行底层软件开发的人员，介绍了 ESP32-C5 系列产品中内置的硬件模块，包括概述、功能列表、硬件架构、编程指南、寄存器列表等信息。

本文档中的跳转

在本文档中实现跳转，请参考以下建议：

- [发布进度速览](#)（下一页）罗列了本文档中的所有章节，您可以从这里快速跳转至某个具体章节。
- 您还可以通过文档左侧的**书签**，从文中的任何位置直接跳转至另一个章节。注意，本文档已设置默认打开**书签**功能，但一些 PDF 阅读器或浏览器会忽略此设置。因此，如果您无法找到**书签**功能，请尝试以下方法：
 - 在您的浏览器中安装 PDF 阅读器拓展；
 - 下载本文档，使用本地 PDF 阅读器进行浏览；
 - 配置您的 PDF 阅读器，使其默认打开**书签**功能。
- 大多数 PDF 阅读器均支持跳转功能，允许您借助按钮、菜单选项或快捷键进行跳转（**向上、向下、向前、向后、后退、前进及前往页面**）等。
- 此外，您还可以使用本文档内置的 **GoBack** 按钮（每页右上角）快速后退至跳转之前的位置。注意，本功能仅适用于 Acrobat 系列的 PDF 阅读器（比如 Acrobat Reader 和 Adobe DC）以及内置 Acrobat 系列 PDF 阅读器或拓展的浏览器（比如 Firefox）。

发布进度速览

注意本文档尚未全部完成，具体发布进度见下表：

No.	章节	最新进度
第 I 卷：微处理器和主机		
1	高性能处理器 [to be added later]	0%
2	RISC-V 追踪编码器 (TRACE)	已发布
3	低功耗处理器	已发布
4	通用 DMA 控制器 (GDMA)	已发布
第 II 卷：存储器组织结构		
5	系统和存储器	已发布
6	eFuse 控制器 (eFuse) [to be added later]	80%
第 III 卷：系统组件		
7	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)	已发布
8	复位和时钟	已发布
9	芯片 Boot 控制	已发布
10	中断矩阵 [to be added later]	80%
11	事件任务矩阵 (ETM)	已发布
12	低功耗管理 [to be added later]	9%
13	系统定时器	已发布
14	定时器组 (TIMG)	已发布
15	看门狗定时器 (WDT)	已发布
16	实时时钟定时器	已发布
17	权限控制 (PMS)	已发布
18	系统寄存器 [to be added later]	9%
19	辅助调试	已发布
20	电源检测器	已发布
第 IV 卷：加密和安全组件		
21	AES 加速器 (AES)	已发布
22	ECC 加速器 (ECC)	已发布
23	HMAC 加速器 (HMAC)	已发布
24	RSA 加速器 (RSA)	已发布
25	SHA 加速器 (SHA)	已发布
26	数字签名算法 (DSA)	已发布
27	椭圆曲线数字签名算法 (ECDSA)	已发布
28	片外存储器加密与解密 (XTS_AES)	已发布
29	随机数发生器 (RNG)	已发布
第 V 卷：通讯接口		
30	UART 控制器 (UART)	已发布
31	SPI 控制器 [to be added later]	73%
32	I2C 控制器 (I2C)	已发布
33	I2S 控制器 (I2S)	已发布
34	脉冲计数控制器 (PCNT) [to be added later]	80%
35	USB 串口/JTAG 控制器	已发布

No.	章节	最新进度
36	控制器局域网灵活数据速率 [to be added later]	39%
37	SDIO 从机控制器 (SDIO)	已发布
38	LED PWM 控制器 (LEDC)	已发布
39	电机控制脉宽调制器 (MCPWM)	已发布
40	红外遥控 (RMT)	已发布
41	并行 IO 控制器 (PARLIO)	已发布
42	比特调节器	已发布
第 VI 卷：模拟信号处理		
43	温度传感器	已发布
44	ADC 控制器	已发布
45	模拟电压比较器	已发布

说明：

点击链接或扫描二维码确保您使用的是最新版本的文档：

https://www.espressif.com/documentation/esp32-c5_technical_reference_manual_cn.pdf



目录

I	微处理器和主机	33
1	RISC-V 追踪编码器 (TRACE)	34
1.1	术语	34
1.2	介绍	34
1.3	特性	35
1.4	架构概览	37
1.5	功能描述	37
1.5.1	同步	37
1.5.2	地址模式	37
1.5.3	可选边带信号	38
1.5.4	过滤	38
1.5.5	锚定	39
1.5.6	写存储器模式	39
1.5.7	自动重启	39
1.6	编码器输出数据包	39
1.6.1	头部	39
1.6.2	索引	40
1.6.3	有效载荷	40
1.6.3.1	格式 3	40
1.6.3.2	格式 2	42
1.6.3.3	格式 1	42
1.7	中断	44
1.8	编程流程	44
1.8.1	配置编码器选项	44
1.8.2	配置过滤器	45
1.8.3	使能编码器	46
1.8.4	关闭编码器	47
1.8.5	通知	47
1.8.6	解码数据包	48
1.8.7	AHB 配置	48
1.8.8	软件保留	48
1.9	寄存器列表	49
1.10	寄存器	50
2	低功耗处理器	61
2.1	特性	61
2.2	配置与状态寄存器 (CSR)	62
2.2.1	寄存器列表	62
2.2.2	寄存器	63
2.3	中断和异常	69
2.3.1	中断	69

2.3.2	中断处理	69
2.3.3	异常	70
2.4	调试	70
2.4.1	特性	70
2.4.2	功能描述	71
2.4.3	寄存器列表	71
2.4.4	寄存器	71
2.5	硬件触发器	73
2.5.1	特性	73
2.5.2	功能描述	73
2.5.3	触发执行流程	74
2.5.4	寄存器列表	74
2.5.5	寄存器	74
2.6	性能计数器	77
2.7	系统访问	78
2.7.1	存储器	78
2.7.2	外设	78
2.8	事件任务矩阵功能	78
2.9	睡眠和唤醒流程	78
2.9.1	特性	78
2.9.2	流程	79
2.9.3	唤醒源	80
2.10	寄存器列表	80
2.11	寄存器	81
3	通用 DMA 控制器 (GDMA)	82
3.1	概述	82
3.2	特性	82
3.3	架构	83
3.4	功能描述	83
3.4.1	链表	84
3.4.2	外设到存储器及存储器到外设的数据传输	85
3.4.3	存储器到存储器数据传输	85
3.4.4	启动 GDMA	86
3.4.5	读链表	86
3.4.6	数据传输结束标志	87
3.4.7	访问存储器	87
3.4.8	仲裁	87
3.5	事件任务矩阵功能	88
3.6	中断	89
3.7	编程流程	90
3.7.1	GDMA TX 通道配置流程	90
3.7.2	GDMA RX 通道配置流程	90
3.7.3	GDMA 存储器到存储器配置流程	90
3.7.4	通道优先级和权重配置流程	91
3.8	寄存器列表	92

3.9	寄存器	98
II	存储器组织结构	126
4	系统和存储器	127
4.1	概述	127
4.2	主要特性	127
4.3	功能描述	128
4.3.1	地址映射	128
4.3.2	内部存储器	130
4.3.2.1	ROM	130
4.3.2.2	HP SRAM	130
4.3.2.3	LP SRAM	131
4.3.3	外部存储器	131
4.3.3.1	外部存储器地址映射	131
4.3.3.2	Cache	131
4.3.3.3	Cache 操作	132
4.3.4	GDMA 地址空间	133
4.3.5	模块/外设地址空间映射	134
III	系统组件	137
5	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)	138
5.1	概述	138
5.2	主要特性	138
5.2.1	HP GPIO 交换矩阵和 HP IO MUX	138
5.2.2	LP GPIO 交换矩阵和 LP IO MUX	139
5.3	结构概览	139
5.4	通过 GPIO 交换矩阵的外设输入	141
5.4.1	概述	141
5.4.2	信号同步	142
5.4.3	GPIO 滤波器	142
5.4.4	毛刺滤波器	143
5.4.5	简单 GPIO 输入	143
5.4.6	GPIO 唤醒	143
5.4.6.1	HP GPIO 唤醒	143
5.4.6.2	LP GPIO 唤醒	143
5.4.7	编程流程	144
5.4.7.1	HP GPIO 交换矩阵	144
5.4.7.2	LP GPIO 交换矩阵	145
5.5	通过 GPIO 交换矩阵的外设输出	146
5.5.1	概述	146
5.5.2	简单 GPIO 输出	146
5.5.3	Sigma Delta 调制输出 (SDM)	146
5.5.3.1	功能描述	146
5.5.3.2	配置方法	147

5.5.4	编程流程	147
5.5.4.1	HP GPIO 交换矩阵	147
5.5.4.2	LP GPIO 交换矩阵	148
5.6	IO MUX 的直接输入输出功能	148
5.6.1	概述	148
5.6.2	功能描述	149
5.6.2.1	HP IO MUX	149
5.6.2.2	LP IO MUX	149
5.7	模拟功能描述	149
5.7.1	概述	149
5.7.2	模拟功能描述	149
5.8	Light-sleep 模式管脚功能	150
5.9	GPIO 管脚的 Hold 特性	150
5.10	GPIO 管脚的迟滞特性	151
5.11	GPIO 管脚供电和电源管理	152
5.11.1	GPIO 管脚供电	152
5.11.2	电源管理	152
5.12	HP 外设信号列表	152
5.13	HP IO MUX 管脚功能列表	158
5.14	LP IO MUX 管脚功能列表	160
5.15	GPIO 管脚模拟功能列表	161
5.16	事件任务矩阵 (ETM) 功能	161
5.17	中断	162
5.18	寄存器列表	163
5.18.1	HP GPIO 交换矩阵寄存器列表	163
5.18.2	HP IO MUX 寄存器列表	167
5.18.3	GPIO EXT 寄存器列表	167
5.18.4	LP GPIO 交换矩阵寄存器列表	169
5.18.5	LP IO MUX 寄存器列表	170
5.19	寄存器	170
5.19.1	HP GPIO 交换矩阵寄存器	170
5.19.2	HP IO MUX 寄存器	186
5.19.3	GPIO EXT 寄存器	189
5.19.4	LP GPIO 矩阵寄存器	209
5.19.5	LP IO MUX 寄存器	217
6	复位和时钟	221
6.1	复位	221
6.1.1	概述	221
6.1.2	结构图	221
6.1.3	特性	221
6.1.4	功能描述	222
6.1.5	外设复位	223
6.2	时钟	223
6.2.1	概述	223
6.2.2	结构图	224

6.2.3	特性	224
6.2.4	功能描述	225
6.2.4.1	高性能系统时钟	225
6.2.4.2	低功耗系统时钟	226
6.2.4.3	外设时钟	226
6.2.4.4	PMU 控制高性能系统时钟门控	230
6.3	配置流程	232
6.3.1	高性能系统时钟配置	232
6.3.2	低功耗系统时钟配置	233
6.3.3	外设时钟复位配置	233
6.4	寄存器列表	235
6.4.1	PCR 模块寄存器列表	235
6.4.2	低功耗系统时钟寄存器列表	237
6.5	寄存器	238
6.5.1	PCR 模块寄存器	238
6.5.2	低功耗系统时钟复位寄存器	289
7	芯片 Boot 控制	301
7.1	概述	301
7.2	功能描述	301
7.2.1	默认配置	302
7.2.2	芯片启动模式控制	302
7.2.3	SDIO 输入采样沿和输出驱动沿控制	304
7.2.4	ROM 代码日志打印控制	304
7.2.5	JTAG 信号源控制	305
7.2.6	晶振频率选择	305
8	事件任务矩阵 (ETM)	307
8.1	概述	307
8.2	特性	307
8.3	功能描述	307
8.3.1	架构	307
8.3.2	事件	308
8.3.3	任务	311
8.3.4	事件任务触发状态	315
8.3.5	时序考虑因素	315
8.3.6	通道控制	316
8.4	寄存器列表	317
8.5	寄存器	321
9	系统定时器	411
9.1	概述	411
9.2	主要特性	411
9.3	系统定时器架构	412
9.4	时钟源选择	412
9.5	功能描述	413

9.5.1	计数器	413
9.5.2	比较器和报警	414
9.5.3	事件任务矩阵	415
9.5.4	同步操作	415
9.6	中断	415
9.7	编程示例	416
9.7.1	读取当前计数器的值	416
9.7.2	在单次报警模式下配置一次性报警	416
9.7.3	在周期报警模式下配置周期性报警	417
9.7.4	唤醒后时间补偿	417
9.8	寄存器列表	418
9.9	寄存器	420
10	定时器组 (TIMG)	435
10.1	概述	435
10.2	主要特性	435
10.3	结构概览	436
10.4	功能描述	436
10.4.1	16 位预分频器与时钟选择器	436
10.4.2	54 位时基计数器	437
10.4.3	警报生成	437
10.4.4	定时器重新加载	438
10.4.5	TIMGO 定时器组慢速时钟频率计算	438
10.5	事件任务矩阵功能	438
10.6	中断	440
10.7	配置流程	440
10.7.1	定时器用作简单时钟	440
10.7.2	定时器用于单次报警	440
10.7.3	通过 APB 设置定时器用于周期性报警	441
10.7.4	通过 ETM 设置定时器用于周期性报警	441
10.7.5	慢速时钟频率计算	442
10.8	寄存器列表	444
10.9	寄存器	445
11	看门狗定时器 (WDT)	457
11.1	概述	457
11.2	数字看门狗定时器	458
11.2.1	主要特性	458
11.2.2	功能描述	459
11.2.2.1	时钟源与 32 位计数器	459
11.2.2.2	阶段与超时动作	460
11.2.2.3	写保护	460
11.2.2.4	Flash 引导保护	461
11.3	超级看门狗定时器	461
11.3.1	主要特性	461
11.3.2	SWD 控制器	461

11.3.2.1	结构	461
11.3.2.2	工作流程	462
11.4	中断	462
11.5	寄存器列表	462
11.6	寄存器	463
12	实时时钟定时器	471
12.1	概述	471
12.2	主要特性	471
12.3	功能描述	471
12.4	事件任务矩阵功能	473
12.5	中断	473
12.6	寄存器列表	474
12.7	寄存器	475
13	权限控制 (PMS)	483
13.1	概述	483
13.2	APM 简介	484
13.3	主要特性	485
13.4	TEE 与 REE 术语	486
13.5	功能描述	486
13.5.1	TEE 控制器功能描述	486
13.5.2	APM 控制器功能描述	487
13.5.2.1	结构概览	487
13.5.2.2	SYS_APM 控制器	488
13.5.2.2.1	地址范围	489
13.5.2.2.2	地址范围的访问权限	489
13.5.2.3	PERI_APM 控制器	490
13.6	非法访问与中断	491
13.6.1	SYS_APM 控制器	491
13.6.2	PERI_APM 控制器	492
13.7	配置流程	492
13.8	寄存器列表	493
13.8.1	HP_APM_REG	493
13.8.2	LP_APM_REG	495
13.8.3	LP_APMO_REG	496
13.8.4	CPU_APM_REG	497
13.8.5	TEE_REG	498
13.8.6	LP_TEE_REG	500
13.9	寄存器	502
13.9.1	HP_APM_REG	502
13.9.2	LP_APM_REG	513
13.9.3	LP_APMO_REG	520
13.9.4	CPU_APM_REG	524
13.9.5	TEE_REG	532
13.9.6	LP_TEE_REG	534

14	辅助调试	538
14.1	概述	538
14.2	主要特性	538
14.3	功能描述	538
14.3.1	区域读写监测	538
14.3.2	栈指针监测	539
14.3.3	PC 记录	539
14.3.4	CPU/DMA 总线访问记录	539
14.4	中断	539
14.5	配置流程	540
14.5.1	区域监测和栈指针监测配置	540
14.5.2	PC 记录配置	541
14.5.3	CPU/DMA 总线访问记录配置	541
14.5.3.1	总线访问记录 1 配置	541
14.5.3.2	总线访问记录 2 配置	545
14.6	寄存器列表	548
14.6.1	总线记录配置寄存器列表	548
14.6.2	其他寄存器列表	549
14.7	寄存器	551
14.7.1	总线记录配置寄存器	551
14.7.2	其他寄存器	560
15	电源检测器	574
15.1	概述	574
15.2	主要特性	574
15.3	功能描述	574
15.3.1	架构图	574
15.3.2	欠压检测器	575
15.3.3	电压毛刺检测器	576
15.4	中断	577
15.5	寄存器列表	579
15.6	寄存器	580
IV	加密和安全组件	587
16	AES 加速器 (AES)	588
16.1	概述	588
16.2	主要特性	588
16.3	时钟和复位	588
16.4	工作模式简介	588
16.5	Typical AES 工作模式	590
16.5.1	密钥、明文、密文	590
16.5.2	字节序	590
16.5.3	Typical AES 工作模式的流程	593
16.6	DMA-AES 工作模式	593
16.6.1	密钥、明文、密文	594

16.6.2	字节序	594
16.6.3	标准增量函数	595
16.6.4	块个数	595
16.6.5	初始向量	595
16.6.6	DMA-AES 工作模式的流程	595
16.7	抗攻击伪轮函数	596
16.8	存储器列表	597
16.9	寄存器列表	598
16.10	寄存器	599
17	ECC 加速器 (ECC)	605
17.1	概述	605
17.2	主要特性	605
17.3	ECC 背景知识	605
17.3.1	椭圆曲线与曲线上的点	605
17.3.2	仿射坐标系与 Jacobian 坐标系	605
17.3.3	内存块	606
17.3.4	数据与数据块	606
17.3.5	数据存储	607
17.3.6	数据读取	607
17.3.7	标准运算与 Jacobian 运算	607
17.4	功能描述	607
17.4.1	曲线模式	607
17.4.2	工作模式	608
17.4.2.1	标准点乘模式	608
17.4.2.2	标准点验证模式	609
17.4.2.3	标准点验证 + 标准点乘模式	609
17.4.2.4	Jacobian 点乘模式	609
17.4.2.5	点加模式	609
17.4.2.6	Jacobian 点验证模式	610
17.4.2.7	标准点验证 + Jacobian 点乘模式	610
17.4.2.8	模加模式	610
17.4.2.9	模减模式	611
17.4.2.10	模乘模式	611
17.4.2.11	模除模式	611
17.4.3	提升抗攻击性能	612
17.4.4	时钟	612
17.4.5	复位	613
17.5	中断	613
17.6	配置流程	614
17.7	寄存器列表	615
17.8	寄存器	616
18	HMAC 加速器 (HMAC)	621
18.1	概述	621
18.2	主要特性	621

18.3	功能描述	621
18.3.1	上行模式	621
18.3.2	下行模式 - JTAG 启动功能	622
18.3.3	下行模式 - 数字签名算法密钥派生功能	622
18.4	HMAC eFuse 配置	622
18.5	配置流程	624
18.5.1	启动 HMAC 模块	624
18.5.2	配置 HMAC 密钥和密钥功能	624
18.5.3	下行模式流程	624
18.5.4	上行模式流程	624
18.6	HMAC 算法细节	625
18.6.1	附加填充位	625
18.6.2	HMAC 算法结构	626
18.7	寄存器列表	628
18.8	寄存器	629
19	RSA 加速器 (RSA)	635
19.1	概述	635
19.2	主要特性	635
19.3	功能描述	635
19.3.1	大数模幂运算	636
19.3.2	大数模乘运算	637
19.3.3	大数乘法运算	637
19.3.4	控制加速	638
19.4	中断	639
19.5	存储器列表	641
19.6	寄存器列表	641
19.7	寄存器	642
20	SHA 加速器 (SHA)	647
20.1	概述	647
20.2	主要特性	647
20.3	工作模式简介	647
20.4	功能描述	648
20.4.1	信息预处理	648
20.4.1.1	附加填充比特	648
20.4.1.2	信息解析	648
20.4.1.3	哈希初始值 (Initial Hash Value)	649
20.4.2	哈希运算流程	649
20.4.2.1	Typical SHA 模式下的运算流程	649
20.4.2.2	DMA-SHA 模式下的运算流程	650
20.4.3	信息摘要存储	651
20.4.4	中断	651
20.5	寄存器列表	652
20.6	寄存器	653

21	数字签名算法 (DSA)	657
21.1	概述	657
21.2	主要特性	657
21.3	功能描述	657
21.3.1	概述	657
21.3.2	私钥运算子	658
21.3.3	软件准备工作	658
21.3.4	硬件工作流程	659
21.3.5	软件工作流程	660
21.4	存储器列表	662
21.5	寄存器列表	663
21.6	寄存器	664
22	椭圆曲线数字签名算法 (ECDSA)	667
22.1	概述	667
22.2	主要特性	667
22.3	ECDSA 背景知识	667
22.3.1	域参数	667
22.3.2	密钥生成	668
22.3.3	签名生成	668
22.3.4	签名验证	668
22.4	功能描述	669
22.4.1	ECDSA 工作模式	669
22.4.2	数据和数据块	670
22.4.2.1	数据写入	670
22.4.2.2	数据读取	671
22.4.2.3	消息填充	671
22.4.2.4	解析消息	671
22.4.3	安全功能	671
22.4.3.1	抗攻击性能	672
22.4.3.2	动态访问权限	672
22.4.3.3	硬件占用	672
22.5	配置流程	672
22.5.1	ECDSA 流程	672
22.5.1.1	IDLE 阶段	673
22.5.1.2	PREP 阶段	674
22.5.1.3	LOAD 阶段	674
22.5.1.4	PROC 阶段	674
22.5.1.5	GAIN 阶段	674
22.5.1.6	POST 阶段	675
22.5.1.7	ECDSA SHA 接口	675
22.5.2	时钟	675
22.5.3	复位	676
22.6	中断	677
22.7	存储器块	678
22.8	寄存器列表	679

22.9	寄存器	680
23	片外存储器加密与解密 (XTS_AES)	687
23.1	概述	687
23.2	主要特性	687
23.3	模块结构	687
23.4	功能描述	688
23.4.1	XTS 算法	688
23.4.2	密钥	689
23.4.3	目标空间	689
23.4.4	数据写入	690
23.4.5	手动加密模块	690
23.4.6	自动加密模块	691
23.4.7	自动解密模块	691
23.5	软件流程	692
23.6	抗 DPA 攻击	692
23.6.1	时钟抗 DPA 攻击	693
23.6.2	伪轮次抗 DPA 功能	693
23.7	寄存器列表	695
23.8	寄存器	696
24	随机数发生器 (RNG)	702
24.1	概述	702
24.2	主要特性	702
24.3	功能描述	702
24.4	编程指南	703
24.5	寄存器列表	705
24.6	寄存器	706
V	通讯接口	707
25	UART 控制器 (UART)	708
25.1	概述	708
25.2	主要特性	708
25.3	UART 架构	709
25.4	功能描述	710
25.4.1	时钟与复位	710
25.4.2	UART FIFO	710
25.4.3	波特率产生与检测	711
25.4.3.1	波特率产生	711
25.4.3.2	波特率检测	711
25.4.4	UART 数据帧	712
25.4.5	AT_CMD 字符格式	713
25.4.6	RS485	713
25.4.6.1	驱动控制	714
25.4.6.2	转换延时	714

25.4.6.3	总线侦听	714
25.4.7	IrDA	714
25.4.8	唤醒	715
25.4.9	流控	716
25.4.9.1	硬件流控	716
25.4.9.2	软件流控	717
25.4.10	GDMA 模式	718
25.5	中断	719
25.6	编程流程	720
25.6.1	寄存器类型	720
25.6.2	具体步骤	721
25.6.2.1	UART n 模块初始化	721
25.6.2.2	UART n 通信配置	721
25.6.2.3	启动 UART n	722
25.7	寄存器列表	723
25.7.1	UART 寄存器列表	723
25.7.2	LP UART 寄存器列表	724
25.7.3	UHCI 寄存器列表	725
25.8	寄存器	727
25.8.1	UART 寄存器	727
25.8.2	LP UART 寄存器	749
25.8.3	UHCI 寄存器	768
26	I2C 控制器 (I2C)	787
26.1	概述	787
26.2	主要特性	787
26.3	结构概览	788
26.4	功能描述	790
26.4.1	时钟配置	790
26.4.2	滤除 SCL 和 SDA 噪声	790
26.4.3	SCL 时钟拉伸	791
26.4.4	SCL 空闲时产生 SCL 脉冲	791
26.4.5	同步	791
26.4.6	漏级开路输出	792
26.4.7	时序参数配置	793
26.4.8	超时控制	794
26.4.9	指令配置	795
26.4.10	TX/RX RAM 数据存储	796
26.4.11	数据转换	797
26.4.12	寻址模式	797
26.4.13	10 位寻址的读写标志位检查	797
26.4.14	启动控制器	797
26.5	LP_I2C 和 I2C 的功能差异	798
26.6	中断	798
26.7	配置流程	799
26.7.1	I2C 主机写入从机, 7 位寻址, 单次命令序列	799

26.7.1.1	场景介绍	800
26.7.1.2	配置示例	800
26.7.2	I2C 主机写入从机, 10 位寻址, 单次命令序列	801
26.7.2.1	场景介绍	801
26.7.2.2	配置示例	802
26.7.3	I2C 主机写入从机, 7 位双地址寻址, 单次命令序列	802
26.7.3.1	场景介绍	803
26.7.3.2	配置示例	803
26.7.4	I2C 主机写入从机, 7 位寻址, 多次命令序列	804
26.7.4.1	场景介绍	805
26.7.4.2	配置示例	806
26.7.5	I2C 主机读取从机, 7 位寻址, 单次命令序列	807
26.7.5.1	场景介绍	807
26.7.5.2	配置示例	807
26.7.6	I2C 主机读取从机, 10 位寻址, 单次命令序列	808
26.7.6.1	场景介绍	809
26.7.6.2	配置示例	809
26.7.7	I2C 主机读取从机, 7 位双寻址, 单次命令序列	810
26.7.7.1	场景介绍	811
26.7.7.2	配置示例	811
26.7.8	I2C 主机读取从机, 7 位寻址, 多次命令序列	812
26.7.8.1	场景介绍	813
26.7.8.2	配置示例	814
26.8	寄存器列表	816
26.8.1	I2C 寄存器列表	816
26.8.2	LP_I2C 寄存器列表	817
26.9	寄存器	819
26.9.1	I2C 寄存器	819
26.9.2	LP_I2C 寄存器	841
27	I2S 控制器 (I2S)	862
27.1	概述	862
27.2	术语	862
27.3	特性	863
27.4	系统架构	864
27.5	I2S 模块支持的音频协议	865
27.5.1	TDM Philips 标准模式	866
27.5.2	TDM MSB 对齐标准模式	866
27.5.3	TDM PCM 标准模式	867
27.5.4	PDM 标准模式	867
27.6	I2S TX/RX 模块时钟	868
27.7	I2S 模块复位	870
27.8	I2S 主/从机模式	870
27.8.1	主/从机发送模式	870
27.8.2	主/从机接收模式	871
27.9	发送数据	871

27.9.1	数据格式控制	871
27.9.1.1	通道有效数据位宽	872
27.9.1.2	通道有效数据字节序	872
27.9.1.3	A 律/ μ 律压缩/解压缩	872
27.9.1.4	通道发送数据位宽	873
27.9.1.5	通道数据比特顺序	873
27.9.2	通道模式控制	874
27.9.2.1	TDM 输出模式	874
27.9.2.2	PDM 输出模式	875
27.9.3	同步计数器	877
27.10	接收数据	878
27.10.1	通道模式控制	878
27.10.1.1	TDM 输入模式	878
27.10.1.2	PDM 输入模式	879
27.10.2	数据格式控制	879
27.10.2.1	通道数据比特顺序	879
27.10.2.2	通道储存数据位宽	879
27.10.2.3	通道接收数据位宽	880
27.10.2.4	通道储存数据字节序	880
27.10.2.5	A 律/ μ 律压缩/解压缩	880
27.11	事件任务矩阵功能	880
27.12	I2S 中断	881
27.13	软件配置流程	882
27.13.1	软件配置 I2S 发送流程	882
27.13.2	软件配置 I2S 接收流程	882
27.14	寄存器列表	884
27.15	寄存器	885
28	USB 串口/JTAG 控制器	903
28.1	概述	903
28.2	主要特性	903
28.3	功能描述	905
28.3.1	CDC-ACM USB 接口描述	905
28.3.2	CDC-ACM 固件接口描述	906
28.3.3	USB-JTAG 接口：JTAG 命令处理器	907
28.3.4	USB-JTAG 接口：CMD_REP 使用示例	908
28.3.5	USB-JTAG 接口：响应捕捉单元	908
28.3.6	USB-JTAG 接口：控制传输请求	909
28.4	中断	909
28.5	配置流程	910
28.6	寄存器列表	912
28.7	寄存器	914
29	SDIO 从机控制器 (SDIO)	939
29.1	概述	939
29.2	主要特性	939

29.3	结构概览	939
29.4	协议标准	940
29.5	功能描述	940
29.5.1	物理总线	940
29.5.2	支持命令	941
29.5.3	I/O function 0 地址空间	941
29.5.4	I/O function 1/2 地址空间	943
29.5.4.1	访问 SLC HOST 寄存器空间	944
29.5.4.2	使用递增地址传输包	944
29.5.4.3	使用固定地址传输包	944
29.5.5	DMA	944
29.5.5.1	链表	945
29.5.5.2	链表回写	946
29.5.5.3	数据填充与丢弃	947
29.5.6	SDIO 总线时序	948
29.6	中断	949
29.6.1	主机侧中断	949
29.6.2	从机侧中断	949
29.7	包的发送和接收流程	949
29.7.1	从机向主机发送包	950
29.7.2	从机从主机接收包	951
29.8	寄存器列表	954
29.8.1	HINF 寄存器列表	954
29.8.2	SLC 寄存器列表	954
29.8.3	SLC Host 寄存器列表	955
29.9	寄存器	956
29.9.1	HINF 寄存器	956
29.9.2	SLC 寄存器	961
29.9.3	SLC Host 寄存器	983
30	LED PWM 控制器 (LEDC)	993
30.1	概述	993
30.2	主要特性	993
30.3	结构概览	993
30.4	功能描述	994
30.4.1	定时器	994
30.4.1.1	时钟源	995
30.4.1.2	时钟分频器配置	995
30.4.1.3	20 位计数器	996
30.4.2	PWM 生成器	997
30.4.3	占空比渐变	998
30.4.3.1	线性占空比渐变	998
30.4.3.2	伽马曲线渐变	999
30.4.3.3	占空比渐变暂停和恢复	1000
30.5	事件任务矩阵功能	1000
30.6	中断	1002

30.7	配置流程	1002
30.8	存储空间	1004
30.9	寄存器列表	1005
30.10	寄存器	1007
31	电机控制脉宽调制器 (MCPWM)	1019
31.1	概述	1019
31.2	主要特性	1019
31.3	模块	1022
31.3.1	概述	1022
31.3.1.1	定时器模块	1022
31.3.1.2	操作器模块	1022
31.3.1.3	捕获模块	1023
31.3.1.4	ETM 模块	1023
31.3.2	PWM 定时器模块	1023
31.3.2.1	PWM 定时器模块的配置	1023
31.3.2.2	PWM 定时器工作模式和定时事件生成	1024
31.3.2.3	PWM 定时器影子寄存器	1028
31.3.2.4	PWM 定时器同步和锁相	1028
31.3.3	PWM 操作器模块	1028
31.3.3.1	PWM 生成器模块	1029
31.3.3.2	死区生成器模块	1040
31.3.3.3	PWM 载波模块	1044
31.3.3.4	故障检测模块	1047
31.3.4	捕获模块	1048
31.3.4.1	介绍	1048
31.3.4.2	捕获定时器	1048
31.3.4.3	捕获通道	1049
31.3.5	ETM 模块	1049
31.3.5.1	介绍	1049
31.3.5.2	MCPWM 可产生的 ETM 事件	1049
31.3.5.3	MCPWM 可接收的 ETM 任务	1050
31.4	中断	1050
31.5	寄存器列表	1052
31.6	寄存器	1055
32	红外遥控 (RMT)	1101
32.1	概述	1101
32.2	主要特性	1101
32.3	功能描述	1102
32.3.1	RMT 架构	1102
32.3.2	RAM	1103
32.3.2.1	RAM 结构	1103
32.3.2.2	RAM 使用说明	1103
32.3.2.3	RAM 访问方式	1104
32.3.3	时钟	1104

32.3.4	发射器	1104
32.3.4.1	普通发送模式	1104
32.3.4.2	乒乓发送模式	1105
32.3.4.3	发送加载波	1105
32.3.4.4	持续发送模式	1105
32.3.4.5	多通道同时发送	1106
32.3.5	接收器	1106
32.3.5.1	普通接收模式	1106
32.3.5.2	乒乓接收模式	1106
32.3.5.3	接收滤波	1106
32.3.5.4	接收去载波	1106
32.4	配置参数更新	1107
32.5	中断	1107
32.6	寄存器列表	1109
32.7	寄存器	1111
33	并行 IO 控制器 (PARLIO)	1124
33.1	概况	1124
33.2	术语	1124
33.3	特性	1124
33.4	系统架构	1125
33.5	功能描述	1126
33.5.1	时钟生成器	1126
33.5.2	时钟复位使用限制	1126
33.5.3	主从机模式	1127
33.5.4	RX 模块接收模式	1128
33.5.4.1	电平使能模式	1129
33.5.4.2	脉冲使能模式	1129
33.5.4.3	软件使能模式	1130
33.5.5	RX 模块 GDMA SUC EOF 信号生成	1130
33.5.6	RX 模块超时	1130
33.5.7	TX 模块无限制长度数据传输	1130
33.5.8	TX 片选功能	1131
33.5.9	TX 模块输出时钟门控	1131
33.5.10	TX 模块总线空闲值	1131
33.5.11	单帧数据传输	1131
33.5.12	字节范围内数据顺序翻转	1131
33.6	中断	1132
33.7	配置流程	1132
33.7.1	数据接收操作流程	1132
33.7.2	数据发送操作流程	1133
33.8	应用示例	1133
33.8.1	与 SPI 进行数据传输	1134
33.8.2	与 I2S 进行数据传输	1135
33.9	寄存器列表	1137
33.10	寄存器	1138

34 比特调节器	1149
34.1 概述	1149
34.2 特性	1150
34.3 应用示例	1150
34.4 架构	1150
34.4.1 数据路径	1150
34.4.2 控制路径	1152
34.5 功能描述	1152
34.5.1 指令	1152
34.5.2 配置寄存器	1159
34.6 配置流程	1160
34.7 寄存器列表	1162
34.8 寄存器	1163
 VI 模拟信号处理	 1175
 35 温度传感器	 1176
35.1 介绍	1176
35.2 特性	1176
35.3 结构概览	1176
35.4 功能描述	1177
35.4.1 温度传感器上电	1177
35.4.2 时钟管理	1177
35.4.3 自动监测模式	1177
35.4.4 温度测量范围与温度偏移	1178
35.4.5 数据转换	1178
35.5 配置流程	1178
35.6 中断	1179
35.7 事件任务矩阵功能	1179
35.8 寄存器列表	1180
35.9 寄存器	1181
 36 ADC 控制器	 1185
36.1 概述	1185
36.2 术语表	1185
36.3 特性	1185
36.4 结构概览	1186
36.5 功能描述	1187
36.5.1 ADC 上电	1187
36.5.2 ADC 通道	1187
36.5.3 ADC 时钟	1187
36.5.4 单次采样模式	1188
36.5.5 多通道采样模式	1188
36.5.6 ADC 转换和衰减	1188
36.5.7 DIG ADC FSM	1188
36.5.8 样式表	1189

36.5.9	ADC 滤波器	1191
36.5.10	阈值监控器	1192
36.5.11	GDMA 支持	1192
36.6	事件任务矩阵功能	1192
36.7	中断	1193
36.8	配置流程	1193
36.8.1	单次采样配置	1193
36.8.2	多通道采样配置	1194
36.9	寄存器列表	1195
36.10	寄存器	1196
37	模拟电压比较器	1207
37.1	概述	1207
37.2	主要特性	1207
37.3	系统架构	1207
37.4	功能描述	1208
37.5	事件任务矩阵	1209
37.6	中断	1209
37.7	配置流程	1209
VII	附录	1211
	相关文档和资源	1212
	词汇列表	1213
	外设相关词汇	1213
	寄存器相关缩写	1213
	寄存器的访问类型	1215
	如何配置寄存器的保留域	1217
	概述	1217
	如何配置保留域	1217
	中断配置寄存器	1218
	修订历史	1219

表格

1.3-1	追踪编码器参数	36
1.5-1	可选边带信号	38
1.6-1	头部格式	39
1.6-2	索引格式	40
1.6-3	格式 3 子格式 0	40
1.6-4	格式 3 子格式 1	41
1.6-5	格式 3 子格式 3	41
1.6-6	格式 2	42
1.6-7	格式 1, 有地址	43
1.6-8	格式 1, 无地址	44
1.7-1	TRACE 的内部中断源	44
1.8-1	调试模块的触发支持 (mcontrol 寄存器的 action 字段)	45
1.8-2	追踪状态	47
2.3-1	LP CPU 异常原因	70
2.6-1	性能计数器	77
2.9-1	唤醒源	80
3.4-1	GDMA 配置寄存器与外设选择关系表	85
4.3-1	地址映射	129
4.3-2	模块/外设地址空间映射表	134
5.4-1	HP GPIO 唤醒信号触发条件和清除条件	143
5.4-2	LP GPIO 唤醒信号触发条件和清除条件	144
5.8-1	IO MUX Light-sleep 管脚功能控制寄存器	150
5.12-1	GPIO 交换矩阵外设信号	154
5.13-1	HP IO MUX 管脚功能	159
5.14-1	LP IO MUX 管脚功能	160
5.15-1	GPIO 管脚模拟功能	161
5.17-1	HP IO MUX 和 HP GPIO 交换矩阵内部中断源	162
5.17-2	LP IO MUX 和 LP GPIO 交换矩阵内部中断源	163
6.1-1	复位源	222
6.2-1	晶振频率选择	225
6.2-2	CPU_CLK 时钟源选择	225
6.2-3	CPU_CLK、AHB_CLK 和 HP_ROOT_CLK 的时钟频率	225
6.2-4	衍生高性能时钟源	227
6.2-5	高性能系统外设时钟	228
6.2-6	衍生低功耗时钟源	228
6.2-7	低功耗系统外设时钟	229
6.2-8	控制模块的读写寄存器时钟门控与 PMU 相关寄存器映射关系	231
6.2-9	控制模块的工作时钟门控与 PMU 相关寄存器映射关系	232
7.2-1	Strapping 管脚默认配置	302

7.2-2	芯片启动模式控制	302
7.2-3	SDIO 输入采样沿/输出驱动沿控制	304
7.2-4	UART0 ROM 日志打印控制	304
7.2-5	USB 串口/JTAG ROM 日志打印控制	305
7.2-6	JTAG 信号源控制	305
7.2-7	晶振频率选择	306
8.3-1	事件任务矩阵通道 n 可选事件	308
8.3-2	事件任务矩阵通道 n 可映射任务	311
9.5-1	UNIT n 配置控制位	413
9.5-2	报警触发条件	414
9.5-3	同步操作	415
9.6-1	系统定时器的内部中断源	416
10.4-1	可逆计数器向上计数时的报警场景	437
10.4-2	可逆计数器向下计数时的报警场景	438
10.6-1	TIMG n 的内部中断源	440
11.2-1	超时动作	460
12.3-1	配置 RTC 定时器记录特定事件的发生时刻	472
12.3-2	RTC 定时器目标时刻配置	472
12.5-1	RTC 定时器的内部中断源	473
13.1-1	PMP 和 APM 管控的区域分布	483
13.4-2	TEE 与 REE 之间的比较	486
13.5-1	HP 系统主机访问来源	486
13.5-2	APM 控制器配置信息	488
14.5-1	HP CPU 包格式	543
14.5-2	LP CPU 包格式	543
14.5-3	DMA 包格式	543
14.5-4	LOST 包格式	543
14.5-5	DMA 访问来源	544
14.5-6	DMA_0 包格式	546
14.5-7	DMA LOST 包格式	546
15.3-1	电源管脚检测的配置	577
15.4-1	电源检测器的内部中断源	577
16.4-1	工作模式	589
16.4-2	密钥长度和解密方向	589
16.5-1	Typical AES 模式下状态返回值	590
16.5-2	Typical AES 文本字节序	590
16.5-3	AES-128 密钥字节序	591
16.5-4	AES-256 密钥字节序	592
16.6-1	块模式选择	593
16.6-2	DMA-AES 模式下状态返回值	594

16.6-3	TEXT-PADDING	594
16.6-4	DMA-AES 存储字节序	595
17.3-1	ECC 硬件加速器内存块	606
17.4-1	ECC 加速器曲线模式控制	608
17.4-2	ECC 硬件加速器工作模式控制	608
18.4-1	HMAC 功能及配置数值	623
19.3-1	加速效果	639
19.4-1	RSA 的内部中断源	639
20.3-1	工作模式选择	648
20.3-2	运算标准选择	648
20.4-1	不同运算标准信息摘要的寄存器占用情况	651
20.4-2	SHA 加速器的内部中断源	651
22.4-1	ECDSA 工作模式	669
22.4-2	ECDSA 椭圆曲线选择	669
22.4-3	ECDSA SHA 算法	670
22.4-4	ECDSA 工作状态	670
22.7-1	ECDSA 存储器块	678
23.4-1	根据 Key_A 生成的 Key 值	689
23.4-2	目标空间与寄存器堆的映射关系	690
23.6-1	XTS_AES 伪轮次功能配置	694
25.2-1	UART 和 LP UART 特性区分	708
25.4-1	UART_CHAR_WAKEUP 模式配置	716
26.3-1	I2C 时序参数 (引自 The I2C-bus specification Version 2.1 Table 5)	789
26.4-1	I2C 同步寄存器	792
27.4-1	模块信号描述	865
27.9-1	通道有效数据位宽控制	872
27.9-2	通道有效数据字节序控制	872
27.9-3	PDM 输出模式下 I2S 取数逻辑	875
27.9-4	原始 PDM 数据的 I2S 通道模式控制	876
27.9-5	I2S PCM-to-PDM 数据输出	876
27.10-1	通道储存数据位宽控制	879
27.10-2	通道储存数据字节序控制	880
28.3-1	标准 CDC-ACM 控制请求	905
28.3-2	CDC-ACM 中 RTS 和 DTR 的设置	906
28.3-3	半字节中的命令	907
28.3-4	USB-JTAG 控制请求	909
28.3-5	JTAG 功能描述符	909
28.5-1	复位芯片进入下载模式	911
28.5-2	复位 SoC 从 flash 启动	911

29.5-1	SDIO 从机的 CCCR 设置	942
29.5-2	SDIO 从机的 FBR 设置	943
30.4-1	常用配置频率及精度	997
30.8-1	LEDC 存储空间	1004
31.3-1	操作器模块包含的各个子模块功能	1023
31.3-2	PWM 生成器中的所有定时事件	1030
31.3-3	PWM 定时器递增计数时，定时事件的优先级	1031
31.3-4	PWM 定时器递减计数时，定时事件的优先级	1031
31.3-5	控制死区时间生成器开关的字段	1041
31.3-6	死区生成器的典型操作模式	1042
31.3-7	MCPWM 可产生的 ETM 事件	1049
31.3-8	MCPWM 可接收的 ETM 任务	1050
32.4-1	更新配置参数	1107
33.5-1	时钟限制下复位 AHB 时钟域操作步骤	1127
33.5-2	时钟限制下 TX 模块从机模式操作要求	1128
33.5-3	时钟限制下 RX 模块从机模式操作要求	1128
33.6-1	PARLIO 的内部中断源	1132
34.5-1	指令结构	1153
34.5-2	当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值	1154
34.5-3	CTL_MUX_SEL 的值	1155
34.5-4	指令操作码	1156
34.5-5	LOOP 操作码	1156
34.5-6	ADD 操作码	1157
34.5-7	IF 操作码	1157
34.5-8	IFN 操作码	1157
34.5-9	LDCTD 操作码	1158
34.5-10	LDCTI 操作码	1158
34.5-11	ADDCTI Opcode	1159
34.5-12	EOF 模式设置	1160
35.4-1	温度测量范围与温度偏移	1178
36.5-1	SAR ADC 通道	1187
37.4-1	模拟电压比较器 PAD 和 GPIO 之间的映射关系	1208
37.6-1	模拟电压比较器的内部中断源	1209
37.7-4	ENA/RAW/ST 寄存器的配置	1218

插图

1.0-1	追踪编码器概述图	34
1.2-1	指令追踪流程概览	35
2.0-1	LP CPU 概览	61
2.9-1	LP CPU 唤醒和睡眠流程	79
3.1-1	共用 GDMA 通道的模块	82
3.3-1	GDMA 控制器的架构	83
3.4-1	链表结构图	84
3.4-2	链表关系图	86
4.3-1	系统结构与地址映射结构	129
4.3-2	ROM-Cache 系统结构	130
4.3-3	Cache 系统结构	132
4.3-4	具有 GDMA 功能的外设/模块	134
5.3-1	HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 框图	140
5.3-2	焊盘内部结构	141
5.4-1	GPIO 输入经 HP IO MUX 运行时钟上升沿或下降沿同步	142
5.4-2	GPIO 输入信号滤波时序图	142
5.4-3	毛刺滤波器时序示例	145
5.10-1	未使能迟滞功能时 PAD 上的电平翻转示例	151
5.10-2	使能迟滞功能时 PAD 上的电平翻转示例	152
6.1-1	四种复位等级	221
6.2-1	系统时钟	224
6.3-1	时钟配置示例	233
7.2-1	芯片启动流程	303
8.3-1	事件任务矩阵通道 n 架构	308
8.3-2	事件任务矩阵时钟结构	315
9.3-1	系统定时器结构图	412
9.5-1	系统定时器生成报警	413
10.1-1	定时器组概览	435
10.3-1	定时器组架构	436
11.1-1	看门狗定时器概览	457
11.2-1	ESP32-C5 的数字看门狗定时器	459
11.3-1	SWD 控制器结构	461
13.1-1	PMP 和 APM 的管控关系	484
13.5-1	APM 控制器结构图	487
15.3-1	电源检测器架构图	575

15.3-2	欠压检测器处理方式	576
15.3-3	电压毛刺检测器	577
17.4-1	clk_ecc 时钟图	612
17.4-2	clk_apb_ecc 时钟图	613
18.6-1	HMAC 附加填充位示意图	626
18.6-2	HMAC 结构示意图	626
21.3-1	软件准备工作与硬件工作流程	658
22.5-1	ECDSA 流程	673
22.5-2	clk_ecdsa 时钟图	676
22.5-3	clk_apb_ecdsa 时钟图	676
23.3-1	片外存储器加解密结构	688
24.3-1	噪声源	703
25.3-1	UART 基本架构图	709
25.4-1	UART 控制器分频	711
25.4-2	UART 信号下降沿较差时序图	712
25.4-3	UART 数据帧结构	712
25.4-4	AT_CMD 字符格式	713
25.4-5	RS485 模式驱动控制结构图	714
25.4-6	SIR 模式编解码时序图	715
25.4-7	IrDA 编解码结构图	715
25.4-8	硬件流控图	716
25.4-9	硬件流控信号连接图	717
25.4-10	GDMA 模式数据传输	718
25.6-1	UART 编程流程	721
26.3-1	I2C 主机基本架构	788
26.3-2	I2C 从机基本架构	788
26.3-3	I2C 协议时序 (引自 The I2C-bus specification Version 2.1 Fig.31)	789
26.4-1	I2C 时序图	793
26.4-2	I2C 命令寄存器结构	795
26.7-1	I2C 主机写 7 位寻址的从机	800
26.7-2	I2C 主机写 10 位寻址的从机	801
26.7-3	I2C 主机写 7 位双地址寻址从机	803
26.7-4	I2C 主机分段写 7 位寻址的从机	805
26.7-5	I2C 主机读 7 位寻址的从机	807
26.7-6	I2C 主机读 10 位寻址的从机	809
26.7-7	I2C 主机从 7 位寻址从机的 M 地址读取 N 个数据	811
26.7-8	I2C 主机分段读 7 位寻址的从机	813
27.4-1	ESP32-C5 I2S 系统框图	864
27.5-1	时序图 – TDM Philips 标准	866
27.5-2	时序图 – TDM MSB 对齐标准	867

27.5-3	时序图 – TDM PCM 标准	867
27.5-4	时序图 – PDM 标准	868
27.6-1	I2S 时钟分频器	868
27.9-1	TX 数据格式控制	874
27.9-2	TDM 通道控制	875
27.9-3	PDM 通道控制	877
28.2-1	USB 串口/JTAG 控制器高层框图	904
28.2-2	USB 串口/JTAG 控制器框图	905
29.3-1	SDIO 从机功能结构图	940
29.5-1	CMD52 内容	941
29.5-2	CMD53 内容	941
29.5-3	function 0 地址空间	941
29.5-4	function 1/2 地址空间	944
29.5-5	SDIO 从机 DMA 链表描述符结构	945
29.5-6	SDIO 从机 DMA 链表	946
29.5-7	主机向从机发送递增地址包的数据流	947
29.5-8	采样时序图	948
29.5-9	输出时序图	948
29.7-1	从机向主机发送包的流程	950
29.7-2	从机从主机接收包的流程	952
29.7-3	从机 CPU 挂载 buffer 的流程	953
30.3-1	LED PWM 控制器架构	994
30.3-2	定时器和 PWM 生成器功能框图	994
30.4-1	LEDC_CLK_DIV 为非整数时的分频	996
30.4-2	计数器和 PWM 信号精度关系	996
30.4-3	LED PWM 输出信号图	998
30.4-4	输出信号占空比线性渐变图	999
30.4-5	输出信号占空比伽马曲线渐变图	1000
31.2-1	MCPWM 外设概览	1020
31.3-1	定时器模块	1022
31.3-2	操作器模块	1022
31.3-3	捕获模块	1023
31.3-4	ETM 模块	1023
31.3-5	递增计数模式波形	1024
31.3-6	递减计数模式波形	1025
31.3-7	递增递减循环模式波形，同步事件后递减	1025
31.3-8	递增递减循环模式波形，同步事件后递增	1025
31.3-9	递增模式中生成的 UTEP 和 UTEZ	1026
31.3-10	递减模式中生成的 DTEP 和 DTEZ	1027
31.3-11	递增递减模式中生成的 DTEP 和 UTEZ	1027
31.3-12	PWM 操作器的子模块	1029
31.3-13	递增递减模式下的对称波形	1032
31.3-14	递增计数模式，单边不对称波形，PWMxA 和 PWMxB 独立调制–高电平	1033

31.3-15	递增计数模式，脉冲位置不对称波形，PWMxA 独立调制	1034
31.3-16	递增递减循环计数模式，双沿对称波形，在 PWMxA 和 PWMxB 上独立调制-高电平有效	1035
31.3-17	递增递减循环计数模式，双沿对称波形，在 PWMxA 和 PWMxB 上独立调制-互补	1036
31.3-18	递增递减循环计数模式，错误或同步事件，在 PWMxA 和 PWMxB 上相同调制	1037
31.3-19	NCI 在 PWMxA 输出上软件强制事件示例	1038
31.3-20	CNTU 在 PWMxB 输出上软件强制事件示例	1039
31.3-21	死区模块的开关拓扑	1041
31.3-22	高电平有效互补 (AHC) 死区波形	1042
31.3-23	低电平有效互补 (ALC) 死区波形	1043
31.3-24	高电平有效 (AH) 死区波形	1043
31.3-25	低电平有效 (AL) 死区波形	1044
31.3-26	PWM 载波操作的波形示例	1045
31.3-27	载波模块的第一个脉冲和之后持续的脉冲示例	1046
31.3-28	PWM 载波模块中持续脉冲的 7 种占空比设置	1047
32.3-1	RMT 结构框图	1102
32.3-2	RAM 中脉冲编码结构	1103
33.4-1	PARLIO 系统框图	1125
33.5-1	PARLIO 时钟生成	1126
33.5-2	主机时钟正向波形	1128
33.5-3	主机时钟负向波形	1128
33.5-4	RX 模块电平使能子模式	1129
33.5-5	RX 模块脉冲使能子模式	1129
33.5-6	RX 模块软件使能子模式	1130
34.1-1	比特调节器系统架构图	1149
34.4-1	比特调节器数据路径图	1151
34.4-2	比特调节器控制路径图	1152
35.3-1	温度传感器结构框图	1177
36.4-1	SAR ADC 结构图	1186
36.5-1	SAR ADC 时钟结构图	1187
36.5-2	DIG ADC FSM 概况	1189
36.5-3	APB_SARADC_SAR_PATT_TAB1_REG 包含样式 0 - 3	1190
36.5-4	APB_SARADC_SAR_PATT_TAB2_REG 包含样式 4 - 7	1190
36.5-5	样式结构	1190
36.5-6	cmd0 配置示例	1191
36.5-7	cmd1 配置示例	1191
36.5-8	GDMA 数据格式	1192
37.3-1	模拟电压比较器系统框图	1207

第 I 卷

微处理器和主机

该部分介绍构成芯片系统的基本要素，包括高性能 CPU 和低功耗 CPU 的微处理器架构、用于调试的跟踪编码、直接内存访问 (DMA) 控制器等。

第 1 章

RISC-V 追踪编码器 (TRACE)

ESP32-C5 的高性能 CPU (HP CPU) 通过追踪编码器 (Trace Encoder) 实现了对指令追踪接口的支持。追踪编码器连接 CPU 的指令追踪接口，将信息压缩成更小的数据包，存入内部 SRAM。

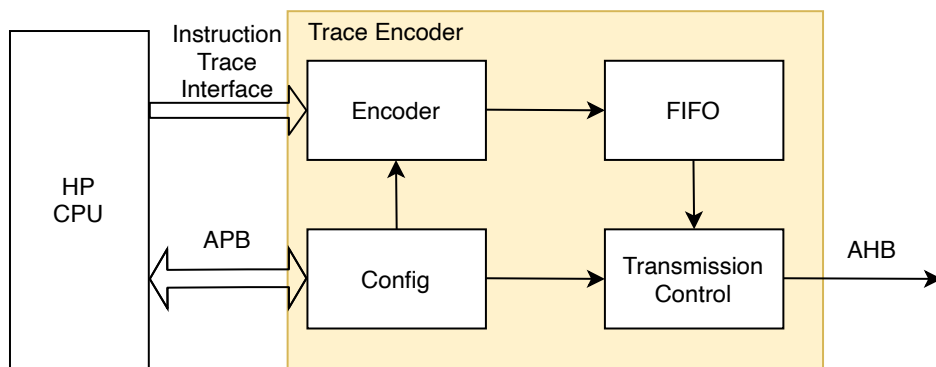


图 1.0-1. 追踪编码器概述图

1.1 术语

为了更好地说明 RISC-V 追踪编码器的功能，本章使用了以下术语。

hart	RISC-V 硬件线程
分支 (branch)	在一定条件下改变执行流程的指令
无法推测的不连续地址 (uninferable discontinuity, or updiscon)	无法仅通过程序二进制文件推测出的程序计数器地址偏移
增量 (delta)	指令在存储器中的存储地址不连续时，程序计数器的地址偏移
陷阱 (trap)	异常或中断，会触发陷阱处理程序
筛选 (qualification)	指令符合筛选条件，即称为通过筛选，才会被追踪
te_inst	编码器发送的数据包的名称
退役 (retire)	机器状态更新时，执行指令的最后一个阶段
EPC	异常程序计数器

1.2 介绍

对于复杂的系统来说，了解程序执行流程并非易事。造成软件不按预期工作可能有很多因素，比如与其他核、外设、实时事件的交互，执行不畅，或以上所有因素的结合。

调试器很难实时监测系统运行中的程序执行流程，因为调试器是侵入性的，会破坏系统的运行状态。但是，监测程序执行又很重要。

此时，可使用指令追踪来追踪程序执行。指令追踪的工作原理是从一个已知的起始地址追踪执行，并发送通常由跳转、调用、返回和分支指令产生的地址增量，中断和异常也会产生地址增量。

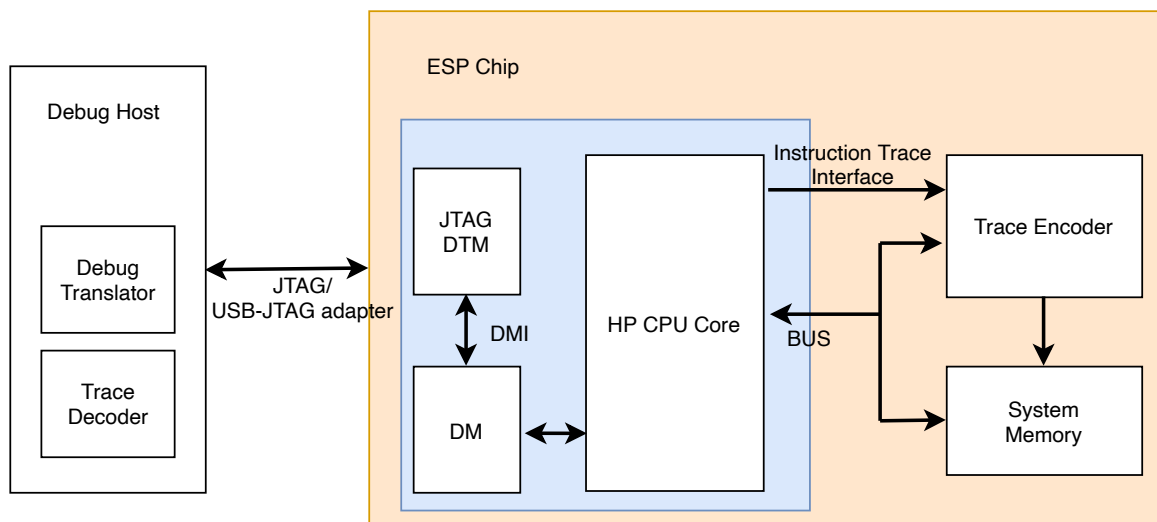


图 1.2-1. 指令追踪流程概览

图 1.2-1 为指令增量追踪的流程。

- HP CPU 核有一个指令追踪接口，可以输出 HP CPU 执行指令的信息，包括指令地址、指令类型等。更多关于 ESP32-C5 HP CPU 指令追踪接口的信息，可参考章节 1 高性能处理器 [\[to be added later\]](#)。
- 追踪编码器通过 HP CPU 的指令追踪接口获取相关指令追踪信息，将信息压缩成带宽更小的数据包，存入系统存储器中。
- 调试器（即图中的 Debug Host）可以将系统存储器中的追踪数据包通过 JTAG 或 USB Serial/JTAG 导出，然后用解码器解压缩、重建程序的执行流。追踪解码器 (Trace Decoder) 通常是外部 PC 上的软件，在获取追踪数据包后，根据在 hart 上运行的程序二进制数据，重建程序的指令流。解码可以离线进行，也可以在 hart 执行时实时进行。

1.3 特性

- 兼容 [Efficient Trace for RISC-V Version 2.0](#) (RISC-V 高效追踪规范 v2.0)，实现的参数详见表 1.3-1
- 支持增量地址模式和完整地址模式
- 支持过滤器
- 支持通过调试触发器或过滤器报告指令地址
- 支持下列边带信号 (sideband signals) 控制追踪数据流
 - 调试触发器启动或关闭编码器
 - hart 暂停时，编码器在报告最后一个数据包后停止工作

- hart 复位后，编码器在报告最后一个数据包后停止工作
- 追踪 FIFO 即将变满时暂停 hart
- 支持任意地址范围用作追踪存储器
- 可配置的同步模式：
 - 同步计数器按包计数
 - 同步计数器按周期计数
 - 关闭同步计数器
- 支持丢包状态标识
- 支持丢包后自动重启
- 写追踪存储器时支持循环和非循环模式
- 具有两个中断：
 - 包的大小超过配置的存储器空间时触发中断
 - 丢包时触发中断
- 具有 128×8 位 FIFO，用于缓存数据包
- 支持 AHB 突发传输，突发长度可配置

表 1.3-1. 追踪编码器参数

参数名称	值	描述
arch_p	0	追踪器符合 RISC-V 追踪规范初始版本
bpred_size_p	0	不支持分支预测模式
cache_size_p	0	不支持跳转目标缓存模式
call_counter_size_p	0	不支持隐式返回模式
ctype_width_p	0	ctype 总线宽度
context_width_p	0	context 总线宽度
ecause_width_p	6	异常原因宽度
ecause_choice_p	0	不支持多选
f0s_width_p	0	不支持格式 0 数据包
filter_context_p	0	不支持过滤上下文信息
filter_excint_p	1	支持过滤异常原因或中断
filter_privilege_p	1	支持过滤特权
filter_tval_p	1	支持过滤陷阱值
iaddress_lsb_p	1	支持压缩指令
iaddress_width_p	32	指令总线为 32 位
iretire_width_p	1	iretire 总线宽度
ilastsize_width_p	0	ilastsize 总线宽度
itype_width_p	3	itype 总线宽度
nocontext_p	1	在 te_inst 数据包中去掉上下文信息
notime_p	1	在 te_inst 数据包中去掉时间信息
privilege_width_p	1	仅支持机器和用户模式

参数名称	值	描述
retires_p	1	每个块退役指令的最大数量
return_stack_size_p	0	不支持隐式返回模式
sijump_p	0	不支持根据顺序推测跳转
taken_branches_p	1	每个时钟周期仅一条指令退役
impdef_width_p	0	未实现

上述参数的具体描述，详见 [Efficient Trace for RISC-V Version 2.0 > 章节 Parameters and Discovery](#)。

1.4 架构概览

本章节主要介绍 ESP32-C5 追踪编码器的具体实现。

如图 1.0-1 所示，追踪编码器包含一个编码器、一个 FIFO、一个寄存器配置模块和一个发送控制模块。

编码器通过指令追踪接口接收 HP CPU 的指令信息，将指令信息压缩成不同的数据包，写入内部 FIFO。

发送控制模块通过 AHB 总线将 FIFO 中的数据写入内部 SRAM。

FIFO 的深度为 128，宽度为 8 位。存储器写入带宽不足时，FIFO 可能会溢出，发生丢包。如果发生丢包，编码器会发送一个包通知丢包情况，在 FIFO 变空前一直停止工作，如果中断已使能还会触发中断。

1.5 功能描述

1.5.1 同步

为保障追踪的有效性，追踪过程中必须定期同步。同步可以通过发送完整的指令地址实现。同步计数器的值达到 `TRACE_RESYNC_PROLONGED_REG` 寄存器的 `TRACE_RESYNC_PROLONGED` 字段的值时，编码器会发送一个同步数据包（格式 3 子格式 0，见章节 1.6.3.1）。

追踪编码器支持两种同步模式，可通过 `TRACE_RESYNC_MODE` 配置：

- 0：关闭同步计数器
- 1：无效值，没有作用
- 2：同步计数器按数据包计数
- 3：同步计数器按周期计数

可通过增加 `TRACE_RESYNC_PROLONGED` 的值，降低发送同步数据包的频率，从而降低同步数据包对带宽的占用。

1.5.2 地址模式

ESP32-C5 支持两种地址模式：增量地址模式和完整地址模式。

- 增量地址模式（默认）：在增量地址模式下，地址会编码为当前指令的实际地址与上一个包含地址的数据包报告的指令实际地址的差值。这种差值编码所需的位比完整地址少，可以令追踪压缩更高效。
- 完整地址模式：在完整地址模式下，追踪的所有地址会编码为绝对地址，而不是差值。这种编码方式相对低效，但是可以辅助软件解码器开发者进行调试。该模式可以通过置位 `TRACE_FULL_ADDRESS` 使能。

1.5.3 可选边带信号

Efficient Trace for RISC-V Version 2.0 > 章节 Optional Sideband Signals 介绍了用于控制编码器的可选边带信号。ESP32-C5 实现了下列信号：

表 1.5-1. 可选边带信号

信号	功能
trigger[2:0]	- 位 0 的脉冲可以令编码器开始追踪，并持续追踪直到另有通知。追踪时还需要满足其他过滤条件。 - 位 1 的脉冲可以令编码器停止追踪，直到另有通知。 - 位 2 的脉冲可以令编码器报告特定地址。 该功能可通过置位 TRACE_DM_TRIGGER_ENA 使能。触发器地址匹配时，建立 trigger 信号。例如，若 mcontrol 寄存器的 action 字段为 2，则触发器地址匹配时 trigger 信号的位 0 变为高电平；若 action 字段为 3，则触发器地址匹配时 trigger 信号的位 1 变为高电平。更多信息，详见表 1.8-1。
halted	暂停 hart。halted 信号建立时，编码器先输出一个数据包报告 hart 暂停前退役的最后一条指令的地址，然后再输出一个支持数据包指示追踪已暂停。halted 信号失效时，编码器重新开始追踪，从同步数据包开始。该功能可通过置位 TRACE_HALT_ENA 使能。
reset	复位 hart。具体行为同上文 halted 的描述。
stall	请求 hart 暂停。如果某些应用需要无损追踪，便可以在追踪编码器不能输出追踪数据包（内部 FIFO 将满）时使用该信号暂停 hart。该功能通过置位 TRACE_STALL_ENA 使能。

1.5.4 过滤

过滤机制控制编码器追踪的标准。比如，过滤机制可以让编码器在出现下列情况时追踪：

- 指令地址在特定范围内时追踪
- 从一个指令地址开始追踪，到出现第二个指令地址时结束追踪
- 针对一个或多个特定特权等级追踪
- 出现异常及/或特定异常原因出发中断处理程序时追踪

每个过滤器单元包含一个过滤器和一个比较器单元。

每个比较器单元实际包含一对比较器（主比较器和次比较器，简称 P、S），可根据要求设置与信号单元匹配的范围，并支持：

- 从指令地址 (iaddress) 和异常陷阱值 (tval) 中选择输入
- 每个比较器可以独立选择运算方式 (<、>、=、!= 等)
- 次匹配值可以是主比较器的屏蔽值
- 两个比较器有多种一起使用的方式： P 、 $P \& \& S$ 、 $!(P \& \& S)$ 、锁存（P 匹配时置 1，S 匹配时清零）
- 每个比较器可以报告特定指令地址

每个过滤器可以指定要过滤的 hart 指令，并具有以下功能：

- 选择比较器单元进行匹配
- 可以选择特定的特权等级 (priv) 和异常原因 (ecause) 作为输入进行匹配
- 选择匹配中断

1.5.5 锚定

由于数据包的长度不定，为在写入存储器时分清数据包之间的边界，ESP32-C5 编码器在数据包之间插入 0：

- 数据包的长度最大为 13 字节，因此正常数据包中不能出现连续 14 个值为 0 的字节。连续 14 个及以上值为 0 的字节后，第一个不为 0 的字节肯定是数据包的第一个字节。
- 每发送 128 个数据包，编码器会向存储器写入 14 个值为 0 的字节作为锚定符。

1.5.6 写存储器模式

写追踪存储器时，追踪数据包的大小可能会超过追踪存储器的容量。此时可配置写存储器的模式，选择是否循环写入数据包：

- 循环模式：追踪数据包的大小超过追踪存储器的容量（即 `TRACE_MEM_CURRENT_ADDR_REG` 的值达到 `TRACE_MEM_END_ADDR_REG` 的值）时，存储器循环写入数据，回到存储器的起始地址 `TRACE_MEM_START_ADDR_REG`，此前写入的数据会被新数据覆盖。
- 非循环模式：追踪数据包的大小超过追踪存储器的容量时，存储器不循环，停在 `TRACE_MEM_END_ADDR_REG`，此前写入的数据保留。

1.5.7 自动重启

因 FIFO 溢出而丢包时，编码器将停止工作，需要软件手动恢复或硬件自动重启。如果 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_RESTART_ENA` 位为 1，FIFO 清空后编码器会自动重启。

如果使能自动重启功能，编码器无论出现什么情况都会自动重启。因此，要关闭编码器，必须先清除 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_RESTART_ENA` 位从而关闭自动重启。

1.6 编码器输出数据包

本节主要介绍 ESP32-C5 追踪编码器输出的数据包格式。

追踪数据包包括头部、索引和有效载荷，依次按照头部、索引和有效载荷的顺序以比特流形式传输。头部、索引和有效载荷中的字段按照下文表格的罗列顺序依次传输，多位字段先传输最低有效位。

1.6.1 头部

头部为 1 个字节，格式如表 1.6-1 所示。

表 1.6-1. 头部格式

字段	位宽	描述	值
length	5	数据包的总长度	4~13

字段	位宽	描述	值
flow	2	保留	0
timestamp	1	保留	0

1.6.2 索引

索引为 2 个字节，格式如表 1.6-2 所示。

表 1.6-2. 索引格式

字段	位宽	描述	值
index	16	每个数据包的索引	0~65536

1.6.3 有效载荷

有效载荷的宽度为 1 至 10 字节。

1.6.3.1 格式 3

格式 3 的数据包用于同步和发送支持信息。规范中定义了四种子格式，ESP32-C5 仅支持 3 种。

格式 3 子格式 0 - 同步

该格式的数据包包含解码器完全识别指令所需的全部信息。开始追踪第一条指令时（除非指令恰好是异常处理程序中的第一条指令），或同步定时器超时触发同步时，编码器会发送该格式的数据包。该格式数据包有效载荷的长度为 5 个字节。

表 1.6-3. 格式 3 子格式 0

字段名	位宽	描述
format	2	11 (sync): 同步
subformat	2	00 (start): 开始追踪，或开始同步
branch	1	0: 地址指向分支指令，且分支跳转 1: 指令非分支指令，或分支没有跳转
privilege	1	所追踪指令的特权等级
address	31	指令的完整地址。字段值需左移 1 位，以复原字节地址
sign_extend	3	保留

格式 3 子格式 1 - 异常

该格式的数据包同样包含解码器完全识别指令所需的全部信息。发生异常或触发中断时，编码器会发送该格式的数据包。该格式的数据包中包含了异常原因、异常指令的“陷阱值”，陷阱处理程序的地址和异常指令的地址。该格式数据包有效载荷的长度不定：如果 interrupt 字段为 1，则 tvalepc 字段省略，长度为 6 个字节；否则长度为 10 个字节。

表 1.6-4. 格式 3 子格式 1

字段名	位宽	描述
format	2	11 (sync): 同步
subformat	2	01 (exception): 异常原因和陷阱处理程序的地址
branch	1	0: 地址指向分支指令, 且分支跳转 1: 指令非分支指令, 或分支没有跳转
privilege	1	所追踪指令的特权等级
ecause	6	异常原因
interrupt	1	中断
theaddr	1	该字段为 1 时, address 字段指向陷阱处理程序的地址。该字段为 0 时, 如果异常情况是无法推测出程序计数器的目标地址偏移量, 则 address 字段指向异常程序计数器的值, 如果是其他异常中断, 则 address 字段不定义。
address	31	指令的完整地址。地址必须向左偏移 1 位, 以复原字节地址
tvalepc	32	utval、stval 或 mtval 控制/状态寄存器的值。如果 interrupt 字段为 1, 该字段省略。
sign_extend	3	保留

格式 3 子格式 3 - 支持

该格式的数据包包含解码器需要的辅助信息, 在追踪结束时、过滤器匹配失败或丢包时发出。该格式数据包有效载荷的长度为 2 个字节。

表 1.6-5. 格式 3 子格式 3

字段名	位宽	描述
format	2	11 (sync): 同步
subformat	2	11 (support): 解码器需要的辅助信息
enable	1	表示编码器已开启
encoder_mode	1	永远为 0。仅分支追踪支持
qual_status	2	表示指令的筛选状态: 00 (no_change): 指令筛选状态没有变化 01 (ended_rep): 指令筛选已结束, 上一个指令为最后一个通过筛选的指令 10 (trace lost): 丢失了一个或多个包 11 (ended_upd): 指令筛选已结束, 即便上一个 te_inst 不是最后一个通过筛选的指令, 也会因指令地址不连续、无法推测而发送

字段名	位宽	描述
ioptions	6	表示以下可选模式： • 根据顺序推测跳转（位 5）：该位为 1 时，不会报告根据顺序推测出的跳转目标地址。该功能未实现，因此此位始终为 0 • 分支预测（位 4）：该位为 1 时，使能分支预测。该功能未实现，因此此位始终为 0 • 缓存跳转目标（位 3）：该位为 1 时，使能跳转目标缓存。该功能未实现，因此此位始终为 0 • 完整地址（位 2）： – 0：增量地址模式 – 1：完整地址模式 • 隐式异常（位 1）：如果可以通过 ecause 字段（异常原因）确定陷阱向量，则格式 3 子格式 1 数据包不包含地址。该功能未实现，因此此位始终为 0 • 隐式返回（位 0）：该位为 1 时，不报告函数的返回地址。该功能未实现，因此此位始终为 0
sign_extend	2	保留

1.6.3.2 格式 2

该格式的数据包仅包含指令地址，在必须报告指令地址时使用（比如指令目标地址为无法推测的不连续地址，或指令为发生异常之前的最后一个指令），包中没有分支信息。

增量地址模式使能时，该格式数据包有效载荷的长度不定，在 5 到 8 个字节之间。地址字段宽度可为 8、16、24 或 32 位，取决于数据包长度。例如，如果数据包头部为 n 字节，则地址字段宽度为 $n - 4$ 字节。地址字段需要通过有符号扩展到 32 位。

表 1.6-6. 格式 2

字段名	位宽	描述
format	2	10 (addr-only)：无分支信息
notify	1	若该位的值与地址字段的最高有效位不同，则表示 trigger[2] 或过滤器匹配触发了通知请求，数据包报告指令的目标地址不是无法推测的不连续地址
updiscon	1	若该位的值与 notify 的值不同，则表示指令的目标地址为无法推测的不连续地址，且该地址为异常、特权变更或同步之前的最后一条指令
zero_extend	4	保留
address	不定	指令的增量地址，位宽为 8、16、24 或 32 位

1.6.3.3 格式 1

该格式的数据包包含分支信息，在必须报告分支信息（比如分支映射已满时）或在上一个包后出现至少一个分支、必须报告指令地址时使用。

增量地址模式使能时，该格式数据包有效载荷的长度不定，在 5 到 12 个字节之间。地址字段宽度可为 8、16、24 或 32 位，取决于数据包长度。例如：

- 数据包头部为 n 字节
- format 字段的值为 1
- branch 字段的值为：
 - 0: 数据包为不包含地址的格式 1 数据包
 - 1: branch_map 宽度为 1 位, zero_ext 宽度为 6 位, address 宽度为 $n - 5$ 位
 - 2 ~ 3: branch_map 宽度为 3 位, zero_ext 宽度为 4 位, address 宽度为 $n - 5$ 位
 - 4 ~ 7: branch_map 宽度为 7 位, zero_ext 宽度为 0 位, address 宽度为 $n - 5$ 位
 - 8 ~ 15: branch_map 宽度为 15 位, zero_ext 宽度为 0 位, address 宽度为 $n - 6$ 位
 - 16 ~ 31: branch_map 宽度为 31 位, zero_ext 宽度为 0 位, address 宽度为 $n - 8$ 位

格式 1, 有地址时 branch_map 字段

有效载荷长度不定。

表 1.6-7. 格式 1, 有地址

字段名	位宽	描述
format	2	01: 包含分支信息
branches	5	该字段的值决定了 branch_map 的位宽 (branch_map 的有效位数量), 具体如下: • 0: (该格式中此值无效) • 1: 1 位 • 2 ~ 3: 3 位 • 4 ~ 7: 7 位 • 8 ~ 15: 15 位 • 16 ~ 31: 31 位 比如, 若 branches = 12, 则 branch_map 为 15 位, 低 12 位有效。
branch_map	不定	表示分支是否跳转的位数组。位 0 对应最早执行的分支指令。每个位的含义如下: • 0: 分支已跳转 • 1: 分支未跳转 该字段的位宽不定, 取决于 branches 字段的值
notify	1	若该位的值与地址字段的最高有效位不同, 则表示 trigger[2] 或过滤器匹配触发了通知请求, 数据包报告指令的目标地址不是无法推测的不连续地址
updiscon	1	若该位的值与 notify 的值不同, 则表示指令的目标地址为无法推测的不连续地址, 且该地址为异常、特权变更或同步之前的最后一条指令
zero_ext	不定	该字段的宽度取决于 branches 字段, 具体如下: • 1: 6 位 • 2 ~ 3: 4 位 • 4 ~ 31: 0 位
address	不定	指令的增量地址, 位宽为 8、16、24 或 32 位

格式 1, 无地址时 branch_map 字段

该格式数据包有效载荷的长度为 5 个字节。

表 1.6-8. 格式 1, 无地址

字段名	位宽	描述
format	2	01: 包含分支信息
branches	5	该字段的值决定了 branch_map 的位宽, 仅 0 有效, 对应 branch_map 的位宽为 31 位
branch_map	31	表示分支是否跳转的位数组。位 0 对应最早执行的分支指令。每个位的含义如下: 0: 分支已跳转 1: 分支未跳转
sign_extend	2	保留

1.7 中断

ESP32-C5 的 TRACE (即 HP CPU 的追踪编码器) 可以生成 TRACE_INTR 中断信号并发送给[中断矩阵](#)。

TRACE_INTR 中断信号由 TRACE 的内部中断源生成。表 1.7-1 列出了 TRACE 的内部中断源、中断触发条件、以及生成的中断信号。

表 1.7-1. TRACE 的内部中断源

内部中断源	触发条件	中断信号
TRACE_MEM_FULL_INTR	数据包大小超过追踪存储器容量 (即 TRACE_MEM_CURRENT_ADDR_REG 的值达到 TRACE_MEM_END_ADDR_REG 的值) 时触发。如有必要, 可使能中断通知 HP CPU 处理, 比如可再申请新的存储器空间	TRACE_INTR
TRACE_FIFO_OVERFLOW_INTR	内部 FIFO 溢出时触发, 表示一个或多个数据包丢失	TRACE_INTR

说明:
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 [中断矩阵 \[to be added later\]](#) > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器, 详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [1.9 寄存器列表](#)。

1.8 编程流程

1.8.1 配置编码器选项

默认情况下, 追踪数据包和增量地址会发送到存储器中, 直到追踪 FIFO 溢出或追踪存储器变满为止。可以通过一些可选的配置设置来改变追踪行为。编码器具有一些可选的配置选项, 可以改变追踪行为。

- 完整地址模式
 - 编码器默认使用增量地址模式。可以置位 `TRACE_FULL_ADDRESS` 以启用完整地址模式用来调试。
- 丢包后重启
 - 因 FIFO 溢出而丢失追踪数据包时，如果 `TRACE_RESTART_ENA` 为 1，编码器将自动停止。FIFO 为空时，编码器将重新开始报告数据包。
- FIFO 将满时暂停 CPU
 - 如果 `TRACE_STALL_ENA` 为 1，FIFO 将满时会向 CPU 发出暂停请求，CPU 会一直暂停，直到请求失效。
- CPU 暂停时停止追踪

如果 `TRACE_HALT_ENA` 为 1，编码器会在 CPU 暂停时先输出一个数据包，报告暂停前最后一条指令的地址，然后再输出一个支持数据包指示追踪已停止。halted 信号失效时，编码器重新开始追踪，从同步数据包开始。

- CPU 复位时停止追踪

如果 `TRACE_RESET_ENA` 为 1，编码器会单独复位。编码器会在 CPU 复位时先输出一个数据包，报告复位前最后一条指令的地址，然后再输出一个支持数据包指示追踪已停止。CPU 退出复位状态时，编码器重新开始追踪，从同步数据包开始。

- 使用调试模块的触发输出

HP CPU 的调试模块有一个触发单元，其匹配控制寄存器 (mcontrol) 有一个 4 位的 action 字段，该字段值为 2 ~ 4 时用于追踪。ESP32-C5 已实现此功能。action 字段的值详见表 1.8-1。

表 1.8-1. 调试模块的触发支持 (mcontrol 寄存器的 action 字段)

值	描述
2	开始追踪
3	结束追踪
4	通知编码器报告地址

有关调试模块触发单元的详细信息，请参考章节 1 高性能处理器 [to be added later]。

1.8.2 配置过滤器

章节 1.5.4 中描述的过滤单元可以按以下方式配置：

1. 选择以下任一匹配模式：
 - 置位 `TRACE_MATCH_PRIVILEGE` 启用特权匹配模式：
 - 通过 `TRACE_MATCH_CHOICE_PRIVILEGE` 配置特权级别
 - 置位 `TRACE_MATCH_ECAUSE` 启用 ecause 匹配模式：
 - 通过 `TRACE_MATCH_CHOICE_ECAUSE` 配置 ecause 值
 - 置位 `TRACE_MATCH_INTERRUPT` 启用中断匹配模式：
 - 通过 `TRACE_MATCH_VALUE_INTERRUPT` 配置中断级别

- 置位 `TRACE_MATCH_COMP` 启用比较器匹配模式：
 - 配置主比较器：
 - * 通过 `TRACE_P_INPUT` 选择输入
 - * 通过 `TRACE_P_FUNCTION` 选择比较器运算符
 - * 通过 `TRACE_FILTER_P_COMPARATOR_MATCH_REG` 配置匹配值
 - 必要时配置次比较器：
 - * 通过 `TRACE_S_INPUT` 选择输入
 - * 通过 `TRACE_S_FUNCTION` 选择比较器运算符
 - * 通过 `TRACE_FILTER_S_COMPARATOR_MATCH_REG` 配置匹配值
 - 通过 `TRACE_MATCH_MODE` 选择用于匹配的比较器：
 - * 0: 仅主比较器用于匹配，次比较器无效
 - * 1: 主比较器和次比较器都用于匹配 ($P \& \& S$)
 - * 2: 主比较器或次比较器都不用于匹配 ($\neg(P \& \& S)$)
 - * 3: 主比较器匹配时开始过滤，次比较器匹配时结束过滤

如果置位 `TRACE_P_NOTIFY` 或 `TRACE_S_NOTIFY`，比较器会以通知的形式报告匹配地址，指令不会过滤。例如：

- 如果 `TRACE_MATCH_MODE` 为 0，并且 `TRACE_P_NOTIFY` 为 1，则所有指令都会追踪，指令与主比较器匹配时会发出通知。关于通知的介绍，请查看章节 1.8.5。
- 如果 `TRACE_MATCH_MODE` 为 0，并且 `TRACE_S_NOTIFY` 为 1，则指令将通过主比较器过滤，指令与次比较器匹配时会发出通知。

2. 置位 `TRACE_FILTER_EN` 以启用过滤单元。

1.8.3 使能编码器

- 通过 `TRACE_MEM_START_ADDR_REG` 和 `TRACE_MEM_END_ADDR_REG` 配置追踪存储器的地址空间。编码器可以访问的地址范围为 `0x40800000 ~ 0x4085FFFF`
- 置位 `TRACE_MEM_CURRENT_ADDR_UPDATE`，将 `TRACE_MEM_CURRENT_ADDR_REG` 的值更新为 `TRACE_MEM_START_ADDR_REG` 的值
- (可选) 通过 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_MEM_LOOP` 位配置写存储器的模式
 - 0: 非循环模式
 - 1: 循环模式 (默认)
- 通过 `TRACE_RESYNC_PROLONGED_REG` 寄存器的 `TRACE_RESYNC_MODE` 位配置同步模式
 - 0: 关闭同步计数器 (默认)
 - 1: 无效值，没有作用
 - 2: 同步计数器按数据包计数
 - 3: 同步计数器按周期计数

- (可选) 通过 `TRACE_RESYNC_PROLONGED_REG` 配置同步计数器的阈值，默认为 128
- (可选) 使能中断
 - 向 `TRACE_INTR_ENA_REG` 寄存器的相应位写 1 开启对应中断
 - 向 `TRACE_INTR_CLR_REG` 寄存器的相应位写 1 清除对应中断
 - 读 `TRACE_INTR_RAW_REG` 寄存器的值，获取中断状态
- (可选) 向 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_RESTART_ENA` 位写 1，使能自动重启。该功能默认开启
- 向 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_TRIGGER_ON` 字段写 1，使能编码器
- 使能追踪编码器，有以下两种方式：
 - 向 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_TRIGGER_ON` 位写 1
 - 通过调试模块的 `action` 字段使能编码器。触发器地址匹配且 `action` 字段为 2 时，编码器使能。触发器的配置，请参考 RISC-V Debug Specification。

编码器使能后，会一直追踪 HP CPU 的指令接口，向追踪存储器写入数据包。

1.8.4 关闭编码器

停止输出追踪数据包，有以下两种方式：

1. 向 `TRACE_TRIGGER_REG` 寄存器的 `TRACE_TRIGGER_OFF` 位写 1 关闭编码器
2. 通过调试模块的 `action` 字段关闭编码器。触发器地址匹配且 `action` 字段为 3 时，编码器关闭。触发器的配置，请参考 RISC-V Debug Specification。

由于内部 FIFO 的存在，数据包在追踪结束后不会立即写入存储器。建议查询 `TRACE_FIFO_STATUS_REG` 确认数据是否全部写入存储器。

表 1.8-2. 追踪状态

字段	描述
<code>TRACE_FIFO_EMPTY</code>	该字段如果为 1，则表示 FIFO 为空，所有数据已写入内存
<code>TRACE_WORK_STATUS</code>	编码器的工作状态： • 0：空闲状态，编码器未启动 • 1：编码器正常工作，输出数据包 • 2：编码器未输出数据包，CPU 已暂停或复位 • 3：编码器未输出数据包，发生数据包丢失，等待 FIFO 为空时重新启动

1.8.5 通知

如果一个目标地址很特殊，即使不是无法推测的不连续地址，用户也可能想要报告。

通知和报告特殊地址有以下两种方式：

1. 调试模块的触发器地址匹配，并且 `action` 为 4

2. 过滤器比较器匹配

收到通知请求时，编码器会发出格式 2 或格式 1 的数据包报告地址，具体取决于 branch_map 字段。如果 branch_map 为 0，则发送格式 2 数据包，否则发送格式 1 数据包。notify 字段表示数据包是否报告的是通知地址。如果地址是无法推测的不连续地址，即使是通知用的数据包，notify 字段也不会变为高电平，否则解码器无法正确重建指令流。

1.8.6 解码数据包

- 找到解码的第一个地址
 - 读 [TRACE_INTR_RAW_REG](#) 寄存器的 [TRACE_MEM_FULL_INTR_RAW](#) 位，查看追踪存储器是否已满
 - * 如果值为 0，从 [TRACE_MEM_START_ADDR_REG](#) 寄存器读取追踪数据包
 - * 如果值为 1，且开启了存储器循环模式，则之前的数据包已被覆盖。此时，读取 [TRACE_MEM_CURRENT_ADDR_REG](#) 的值，获取最后一个写入的地址，该地址即为解码器处理的第一个地址
- 用解码器解码数据包
 - 解码器从第一个地址开始读取所有数据包，结合二进制文件重建数组
 - 如章节 1.6 所述，编码器向存储器分区边界写 14 个值为 0 的字节。因此出现 14 个值为 0 的字节时，则表示下一个非 0 字节为新数据包的头部

1.8.7 AHB 配置

ESP32-C5 编码器支持 AHB 总线，可选配置项可以控制编码器的传输带宽。

如果有多个 DMA 主机，可以通过 [TRACE_AHB_CONFIG_REG](#) 配置 AHB 突发传输，增加编码器的带宽。

- [TRACE_HBURST](#) 支持以下配置：

- 0x0: SINGLE
- 0x1: INCR (长度未定义)
- 0x2: INCR4
- 0x4: INCR8
- 其他值：保留

突发传输意味着仲裁成功后，可以连续传输多个字。SINGLE 传输为 1 拍，INCR 长度未定义，INCR4 为 4 拍，INCR 为 8 拍。突发长度越长，带宽越大。

- 传输配置为 INCR 时，由于传输长度不确定，为了避免编码器长时间占用总线，传输长度达到 [TRACE_MAX_INCR](#) 后 INCR 传输结束，重新开始总线仲裁。

该配置用于调整带宽。带宽足够时，不需要配置突发传输，可使用默认配置。

1.8.8 软件保留

[TRACE_WORK_STATUS](#) 表示编码器的状态。编码器不处于空闲状态时，必须备份配置，并在芯片进入睡眠状态时还原配置。

1.9 寄存器列表

本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

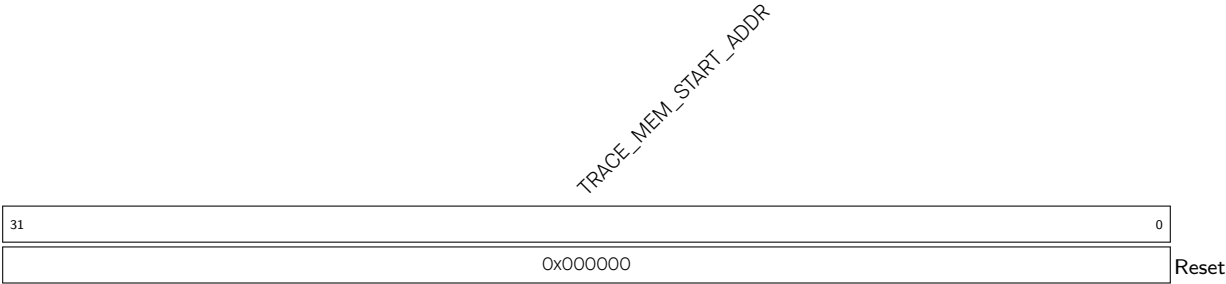
名称	描述	地址	访问
存储器配置寄存器			
TRACE_MEM_START_ADDR_REG	存储器起始地址	0x0000	R/W
TRACE_MEM_END_ADDR_REG	存储器结束地址	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	存储器当前地址	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	存储器地址更新	0x000C	WT
追踪编码器 FIFO 状态寄存器			
TRACE_FIFO_STATUS_REG	FIFO 装填寄存器	0x0010	RO
中断寄存器			
TRACE_INTR_ENA_REG	中断使能寄存器	0x0014	R/W
TRACE_INTR_RAW_REG	中断原始状态寄存器	0x0018	RO
TRACE_INTR_CLR_REG	中断清除寄存器	0x001C	WT
追踪编码器配置寄存器			
TRACE_TRIGGER_REG	追踪触发寄存器	0x0020	varies
TRACE_CONFIG_REG	追踪配置寄存器	0x0024	R/W
TRACE_FILTER_CONTROL_REG	过滤器控制寄存器	0x0028	R/W
TRACE_FILTER_MATCH_CONTROL_REG	过滤器匹配控制寄存器	0x002C	R/W
TRACE_FILTER_COMPARATOR_CONTROL_REG	过滤器比较器匹配控制寄存器	0x0030	R/W
TRACE_FILTER_P_COMPARATOR_MATCH_REG	主比较器匹配值寄存器	0x0034	R/W
TRACE_FILTER_S_COMPARATOR_MATCH_REG	次比较器匹配值寄存器	0x0038	R/W
TRACE_RESYNC_PROLONGED_REG	再同步配置寄存器	0x003C	R/W
TRACE_AHB_CONFIG_REG	AHB 配置寄存器	0x0040	R/W
时钟门控控制和配置寄存器			
TRACE_CLOCK_GATE_REG	时钟门控控制寄存器	0x0044	R/W
版本寄存器			
TRACE_DATE_REG	版本控制寄存器	0x03FC	R/W

1.10 寄存器

本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 [4 系统和存储器](#) 中的表 4.3-2。

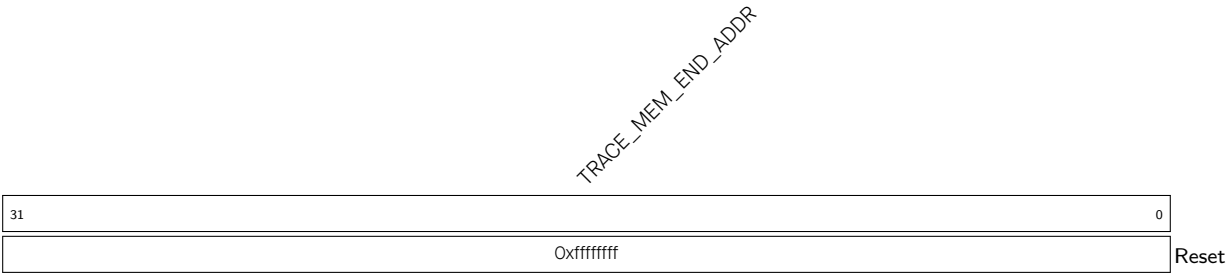
关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 1.1. TRACE_MEM_START_ADDR_REG (0x0000)



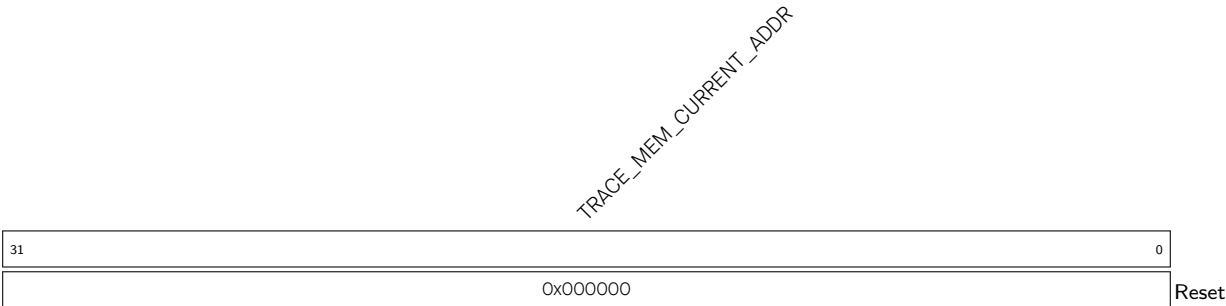
TRACE_MEM_START_ADDR 配置追踪存储器的起始地址。(R/W)

Register 1.2. TRACE_MEM_END_ADDR_REG (0x0004)



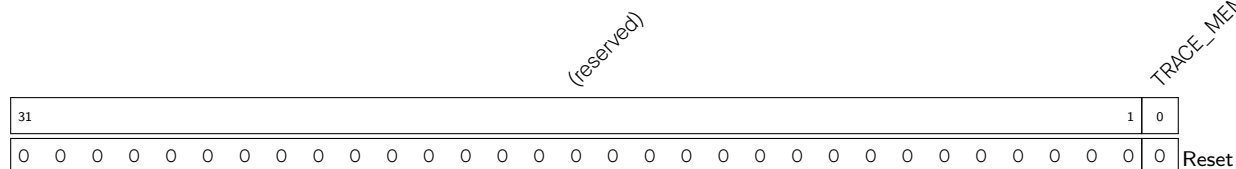
TRACE_MEM_END_ADDR 配置追踪存储器的结束地址。(R/W)

Register 1.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



TRACE_MEM_CURRENT_ADDR 表示当前要写入的存储器地址。(RO)

Register 1.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)



TRACE_MEM_CURRENT_ADDR_UPDATE 配置是否将 TRACE_MEM_CURRENT_ADDR 的值更新为 TRACE_MEM_START_ADDR 的值。

0: 不更新

1: 更新

(WT)

Register 1.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY 表示 FIFO 的状态。

0: 不为空

1: 为空

(RO)

TRACE_WORK_STATUS 表示编码器的状态。

0: 空闲状态

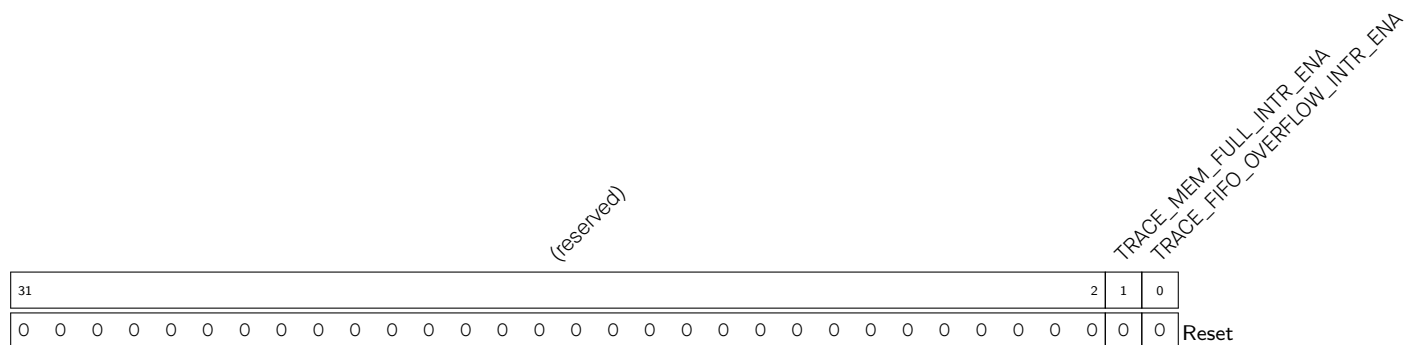
1: 工作状态

2: 因 hart 暂停或复位处于待机状态

3: 丢失状态

(RO)

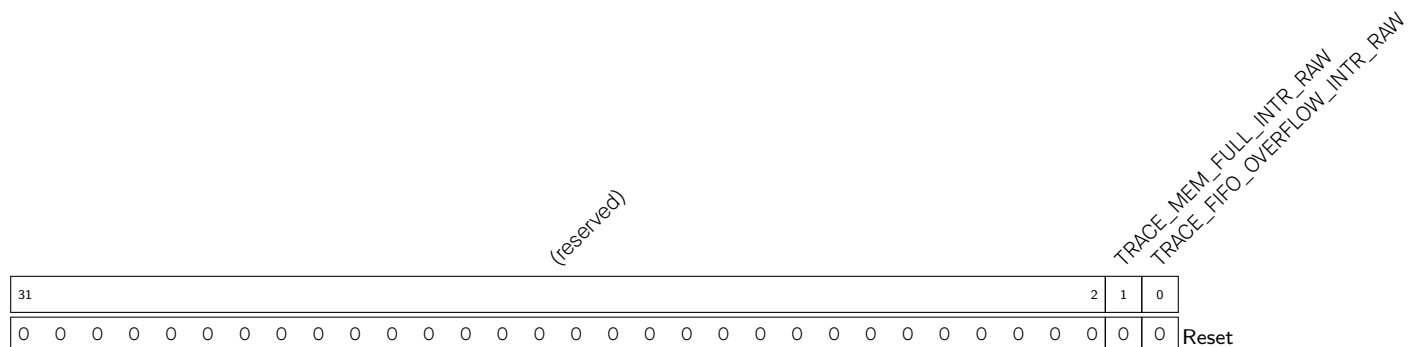
Register 1.6. TRACE_INTR_ENA_REG (0x0014)



TRACE_FIFO_OVERFLOW_INTR_ENA 写1使能 TRACE_FIFO_OVERFLOW_INTR。(R/W)

TRACE MEM FULL INTR ENA 写1使能 TRACE MEM FULL INTR。(R/W)

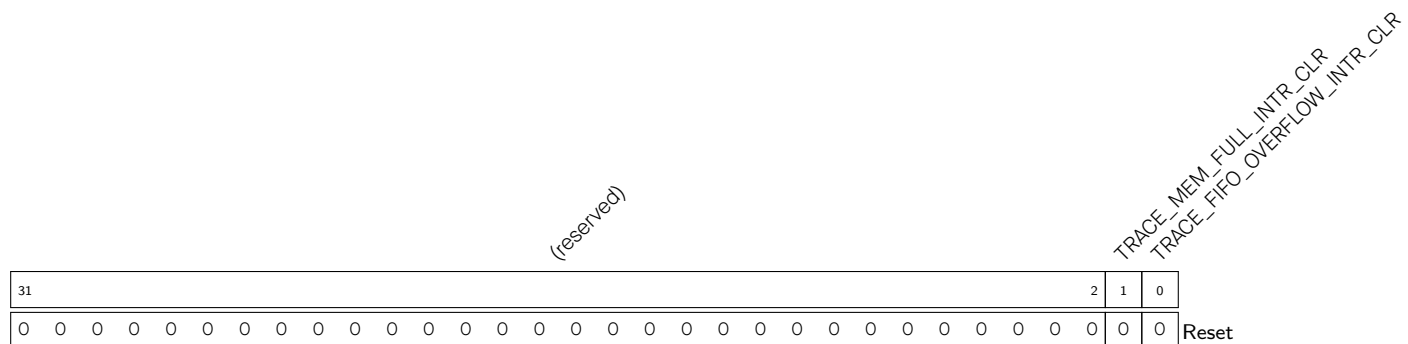
Register 1.7. TRACE_INTR_RAW_REG (0x0018)



TRACE_FIFO_OVERFLOW_INTR_RAW TRACE_FIFO_OVERFLOW_INTR 的原始中断状态。(RO)

TRACE_MEM_FULL_INTR_RAW TRACE_MEM_FULL_INTR 的原始中断状态。(RO)

Register 1.8. TRACE_INTR_CLR_REG (0x001C)



TRACE_FIFO_OVERFLOW_INTR_CLR 写1清除 TRACE_FIFO_OVERFLOW_INTR。(WT)

TRACE_MEM_FULL_INTR_CLR 写1清除TRACE_MEM_FULL_INTR。(WT)

Register 1.9. TRACE_TRIGGER_REG (0x0020)

(reserved)																												TRACE_RESTART_ENA TRACE_MEM_LOOP TRACE_TRIGGER_OFF TRACE_TRIGGER_ON				
31																												4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	Reset

- TRACE_TRIGGER_ON

配置开启追踪编码器。

0: 无效值，没有作用

1: 开启

(WT)
- TRACE_TRIGGER_OFF

配置关闭追踪编码器。

0: 无效值，没有作用

1: 关闭

(WT)
- TRACE_MEM_LOOP

配置存储器写入模式。

0: 非循环模式

1: 循环模式

(R/W)
- TRACE_RESTART_ENA

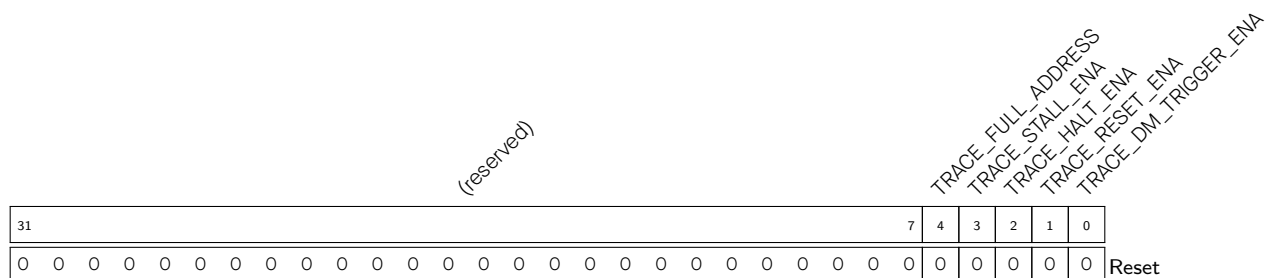
配置是否开启编码器自动重启功能。

0: 关闭

1: 开启

(R/W)

Register 1.10. TRACE_CONFIG_REG (0x0024)



TRACE_DM_TRIGGER_ENA 配置是否使能 trigger 信号。

0: 禁用

1: 使能

(R/W)

TRACE_RESET_ENA 配置是否使能 reset 信号。

0: 禁用

1: 使能

(R/W)

TRACE_HALT_ENA 配置是否使能 halted 信号。

0: 禁用

1: 使能

(R/W)

TRACE_STALL_ENA 配置是否使能 stall 信号。

0: 禁用

1: 使能

(R/W)

TRACE_FULL_ADDRESS 配置地址模式。

0: 增量地址模式

1: 完整地址模式

Register 1.11. TRACE_FILTER_CONTROL_REG (0x0028)

[illegible]

TRACE_FILTER_EN 配置是否使能过滤功能。

0: 禁用

1: 使能

(R/W)

TRACE_MATCH_COMP 配置是否使能比较器匹配模式。

0: 禁用

1: 使能

(R/W)

TRACE_MATCH_PRIVILEGE 配置是否使能特权匹配模式。

0: 禁用

1: 使能

(R/W)

TRACE_MATCH_ECAUSE 配置是否使能 ecause 匹配模式。

0: 禁用

1: 使能 (R/W)

TRACE_MATCH_INTERRUPT 配置是否使能中断匹配模式。

0: 禁用

1: 使能

(R/W)

Register 1.12. TRACE_FILTER_MATCH_CONTROL_REG (0x002C)

(reserved)																								TRACE_MATCH_CHOICE_ECAUSE								TRACE_MATCH_VALUE_INTERRUPT								TRACE_MATCH_CHOICE_PRIVILEGE																							
31																								8								7								2								1								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0								0								0								Reset															

TRACE_MATCH_CHOICE_PRIVILEGE 配置匹配的特权等级。

0: 用户模式

1: 机器模式

(R/W)

TRACE_MATCH_VALUE_INTERRUPT 配置匹配的中断。仅 TRACE_MATCH_INTERRUPT 为 1 时有效。

0: itype=1

1: itype=2

(R/W)

TRACE_MATCH_CHOICE_ECAUSE 配置匹配的 ecause 代碼。(R/W)

Register 1.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

(reserved)																		TRACE_MATCH_MODE		(reserved)		TRACE_S_NOTIFY		TRACE_S_FUNCTION		(reserved)		TRACE_S_INPUT		(reserved)		TRACE_P_NOTIFY		TRACE_P_FUNCTION		(reserved)		TRACE_P_INPUT	
31																		18	17	16	15	14	13	12	10	9	8	7	6	5	4	2	1	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset								

TRACE_P_INPUT 配置主比较器匹配的输入。

- 0: iaddr
 - 1: tval
- (R/W)

TRACE_P_FUNCTION 配置主比较器的运算符。

- 0: 等于
 - 1: 不等于
 - 2: 小于
 - 3: 小于或等于
 - 4: 大于
 - 5: 大于或等于
 - 其他值: 始终匹配
- (R/W)

TRACE_P_NOTIFY 配置是否明确报告主比较器匹配的指令地址。

- 0: 不报告
 - 1: 报告
- (R/W)

TRACE_S_INPUT 配置次比较器匹配的输入。

- 0: iaddr
 - 1: tval
- (R/W)

TRACE_S_FUNCTION 配置次比较器的运算符。

- 0: 等于
 - 1: 不等于
 - 2: 小于
 - 3: 小于或等于
 - 4: 大于
 - 5: 大于或等于
 - 6: 次匹配值是主比较器的屏蔽值
 - 其他值: 始终匹配
- (R/W)

见下页...

Register 1.13. TRACE_FILTER_COMPARATOR_CONTROL_REG (0x0030)

接上页...

TRACE_S_NOTIFY 配置是否明确报告次比较器匹配的指令地址。

- 0: 不报告
- 1: 报告

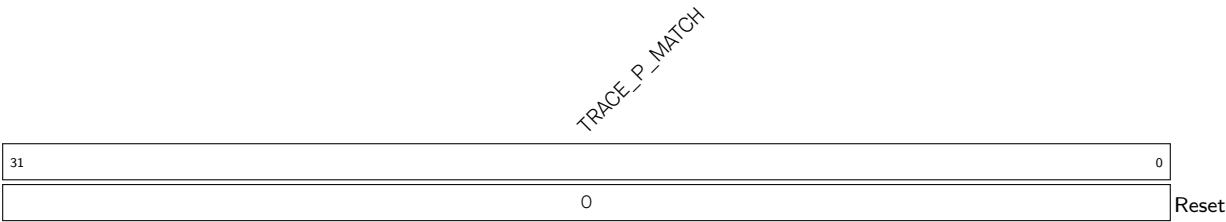
(R/W)

TRACE_MATCH_MODE 配置比较器的匹配条件。

- 0: 仅主比较器匹配
- 1: 主比较器和次比较器都用于匹配 (P&S)
- 2: 主比较器和次比较器都不用于匹配 (!P&S)
- 3: 主比较器匹配时开始过滤，次比较器匹配时结束过滤

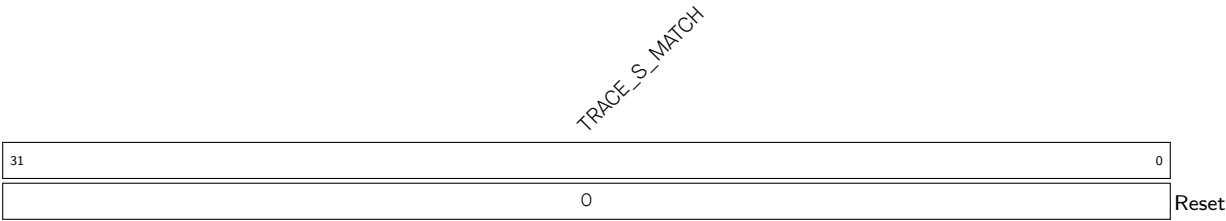
(R/W)

Register 1.14. TRACE_FILTER_P_COMPARATOR_MATCH_REG (0x0034)



TRACE_P_MATCH 配置主比较器的匹配值。(R/W)

Register 1.15. TRACE_FILTER_S_COMPARATOR_MATCH_REG (0x0038)



TRACE_S_MATCH 配置次比较器的匹配值。(R/W)

Register 1.16. TRACE_RESYNC_PROLONGED_REG (0x003C)

(reserved)						TRACE_RESYNC_MODE																		TRACE_RESYNC_PROLONGED															
31						26						25		24		23																				0			
0						0						0		0		0		0		128																		Reset	

TRACE_RESYNC_PROLONGED 配置同步计数器的閾值。(R/W)

TRACE_RESYNC_MODE 配置同步模式。

- 0: 关闭计数器
1: 无效值
2: 按数据包计数同步
3: 按周期计数同步
(R/W)

Register 1.17. TRACE_AHB_CONFIG_REG (0x0040)

																												(reserved)																																																								TRACE_MAX_INCR										TRACE_HBURST										
31																																																																																				6		5		3		2		0												
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																																																																																				0x0		0x0		Reset																

TRACE_HBURST 配置 AHB 突发传输模式。

- 0: SINGLE
1: INCR (不定义长度)
2: INCR4
4: INCR8
其他值: 无效
(R/W)

TRACE_MAX_INCR 配置 INCR 模式下突发传输最大的长度。(R/W)

Register 1.18. TRACE_CLOCK_GATE_REG (0x0044)

(reserved)																												TRACE_CLK_EN	
31																												1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

TRACE_CLK_EN 配置寄存器时钟门控。

0: 仅在应用写寄存器时开启时钟，从而降低功耗

1: 一直强制为寄存器开启时钟

该位不影响寄存器访问。

(R/W)

Register 1.19. TRACE_DATE_REG (0x03FC)

(reserved)				TRACE_DATE																									
31			28	27																									0
0	0	0	0		0x2211300																								Reset

TRACE_DATE 版本控制寄存器。(R/W)

第 2 章

低功耗处理器

ESP32-C5 低功耗处理器 (Low-Power CPU, LP CPU) 是基于 RISC-V ISA 的 32 位处理器，包括整数 (I)、乘法/除法 (M)、原子 (A) 和压缩 (C) 标准扩展。LP CPU 内核具有 2 级有序标量流水线，具有极低的功耗。CPU 内核架构包含中断控制器 (INTC)、调试模块 (DM) 和用于访问存储器和外设的系统总线 (SYS BUS) 接口。

LP CPU 默认处于睡眠状态（详见章节 2.9），在芯片进入 Deep-sleep 时可保持上电状态（详见章节 4 低功耗管理 [to be added later]），能够访问绝大部分外设和存储器，详情请参考章节 4 系统和存储器。LP CPU 有两种应用场景：

- 功耗不敏感场景：高性能处理器 (HP CPU) 处于工作状态，LP CPU 可协助 HP CPU 完成一些对速度和效率不敏感的控制或计算
- 功耗敏感场景：HP CPU 处于掉电状态来节省功耗时，可以唤醒 LP CPU 处理一些外部唤醒事件

HP CPU 和 LP CPU 可访问的资源如图 2.0-1 所示：

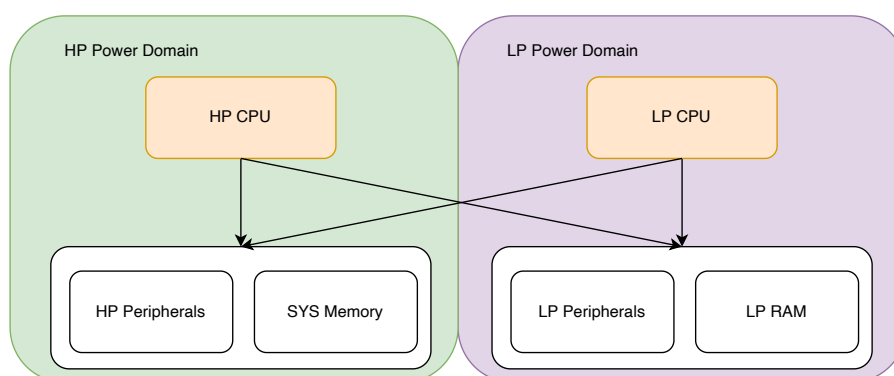


图 2.0-1. LP CPU 概览

在上图中，HP 电源域 (HP Power Domain) 和 LP 电源域 (LP Power Domain) 可分别上下电，当其中一个电源域处于掉电状态时，另一个电源域的 CPU 无法访问该域的资源，只有当该域处于上电状态时才可访问其资源，更多关于电源的信息，请参考章节 4 低功耗管理 [to be added later]。

2.1 特性

LP CPU 具有以下特性：

- 时钟工作频率高达 20 MHz
- 支持 1 个向量中断入口
- 调试模块 (DM) 符合 RISC-V 调试规范 v0.13，支持通过行业标准的 JTAG/USB 端口连接外部调试器
- 硬件触发器符合 RISC-V 调试规范 v0.13，具有 2 个断点/观察点
- 32 位 AHB 系统总线，用于访问存储器和外设

- 支持核心性能指标事件
- 可唤醒 HP CPU 或向 HP CPU 发送中断
- 可访问 HP 存储器和 LP 存储器
- 可访问所有外设空间

2.2 配置与状态寄存器 (CSR)

2.2.1 寄存器列表

下表为 CPU 可访问的 CSR 列表。除了自定义的性能计数器 CSR 外，所有已实现的 CSR 都遵循 RISC-V 指令集手册 V1.10 第二卷“特权架构” (RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10) 中所述的位域标准映射。必须注意的是，受 CPU 中实现的功能子集的限制，即使在标准 CSR 中也并非实现了所有位域。有关详细的 CSR 寄存器描述，请参阅下一小节。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
机器模式信息 CSR			
mhartid	机器模式 hart ID 寄存器	0xF14	RO
机器模式陷阱设置 CSR			
mstatus	机器模式状态寄存器	0x300	R/W
misa ¹	机器模式 ISA 寄存器	0x301	R/W
mie	机器模中断使能寄存器	0x304	R/W
mtvec ²	机器模式陷阱向量寄存器	0x305	R/W
机器模式陷阱处理 CSR			
mscratch	机器模式暂存寄存器	0x340	R/W
mepc	机器模式陷阱程序计数器	0x341	R/W
mcause ³	机器模式陷阱原因寄存器	0x342	R/W
mtval	机器模式陷阱值寄存器	0x343	R/W
mip	机器模中断等待寄存器	0x344	R/W
触发器模块 CSR (与调试模块共用)			
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
调试模式 CSR			
dcsr	调试模式控制与状态寄存器	0x7B0	R/W
dpc	调试模式 PC 寄存器	0x7B1	R/W
dscratch0	调试模式暂存寄存器 0	0x7B2	R/W
dscratch1	调试模式暂存寄存器 1	0x7B3	R/W
机器模式计数器/定时器 CSR			
mcycle	机器模式时钟周期计数器寄存器	0xB00	R/W

¹尽管 [misa](#) 具有 R/W 属性，但由于它的域是硬连线的，所以写操作无效。在 RISC-V 术语中称为 WARL（写入任意数值读取合法数值）。

²[mtvec](#) 仅支持在向量模式下对陷阱处理进行配置，基地址为 256 字节对齐。

³[mcause](#) 中反映的外部中断 ID 也包括 RISC-V 标准为处理器内部中断源预留的 ID。

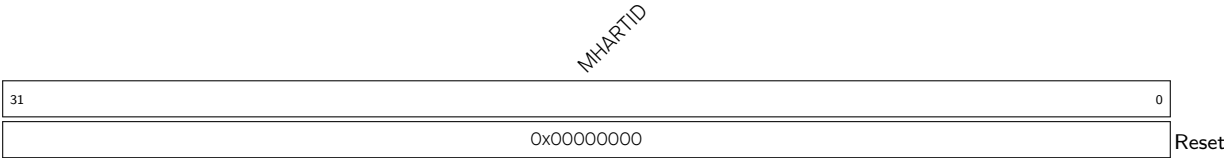
名称	描述	地址	访问
minstret	机器模式退役指令计数器寄存器	0xB02	R/W
mhpmcounter $n(n:3-12)$	机器模式性能监测计数器寄存器	0xB00+ n	R/W
mcycleh	mcycle 的高 32 位	0xB80	R/W
minstreth	minstret 的高 32 位	0xB82	R/W
mhpmcounter $nh(n:3-12)$	mhpmcounter $n(n:3-12)$ 的高 32 位	0xB80+ n	R/W
机器模式计数器设置 CSR			
mcountinhibit	机器模式计数器控制寄存器	0x320	R/W

请注意，如果对上表中只读属性的任何 CSR 尝试执行写入/置位/清除操作，CPU 将生成非法指令异常。

2.2.2 寄存器

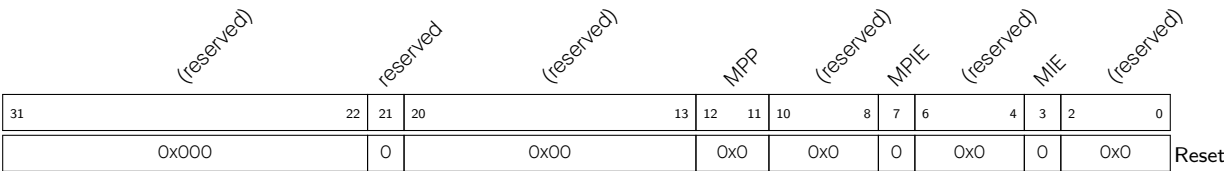
关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 2.1. mhartid (0xF14)



MHARTID 表示硬件线程编号。LP CPU 硬件线程编号为 0。(RO)

Register 2.2. mstatus (0x300)



MIE 写 1 使能全局机器模式中断。(R/W)

MPIE 写 1 使能（陷阱）之前的机器模式中断。(R/W)

MPP 配置（陷阱）之前的机器特权模式。

0x3: 机器模式

其他值：无效

注意：仅低位可写。由于高位直接绑定低位，写入高位将被忽略。

(R/W)

Register 2.3. misa (0x301)

MXL		(reserved)				Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x1		0x0				0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	1

Reset

MXL 机器 XLEN = 1 (32 位)。(RO)**Z** 保留 = 0。(RO)**Y** 保留 = 0。(RO)**X** 非标准扩展 = 0。(RO)**W** 保留 = 0。(RO)**V** 保留 = 0。(RO)**U** 实现用户模式 = 0。(RO)**T** 保留 = 0。(RO)**S** 实现监督模式 = 0。(RO)**R** 保留 = 0。(RO)**Q** 四精度浮点扩展 = 0。(RO)**P** 保留 = 0。(RO)**O** 保留 = 0。(RO)**N** 支持用户级别中断 = 0。(RO)**M** 整数乘除法标准扩展 = 1。(RO)**L** 保留 = 0。(RO)**K** 保留 = 0。(RO)**J** 保留 = 0。(RO)**I** RV32I 基本 ISA = 1。(RO)**H** 虚拟机管理程序扩展 = 0。(RO)**G** 其他标准扩展 = 0。(RO)**F** 单精度浮点扩展 = 0。(RO)**E** RV32E 基本 ISA = 0。(RO)**D** 双精度浮点扩展 = 0。(RO)**C** 压缩标准扩展 = 1。(RO)**B** 保留 = 0。(RO)**A** 原子标准扩展 = 1。(RO)

Register 2.4. mie (0x304)

(reserved)		(reserved)	
31	30	29	0
0x0	0x0	0x0	
Reset			

IE 写 1 使能中断。(R/W)

Register 2.5. mtvec (0x305)

BASE								(reserved)				MODE	
31							8	7			2	1	0
0x000000								0x00				0x1	
Reset													

MODE 表示机器模式中中断是否为向量模式。仅支持向量模式 **0x1**。(RO)

BASE 配置陷阱向量基址的高 24 位，地址为 256 字节对齐。(R/W)

Register 2.6. mscratch (0x340)

MSCRATCH																															
31																															0
0x00000000																															
Reset																															

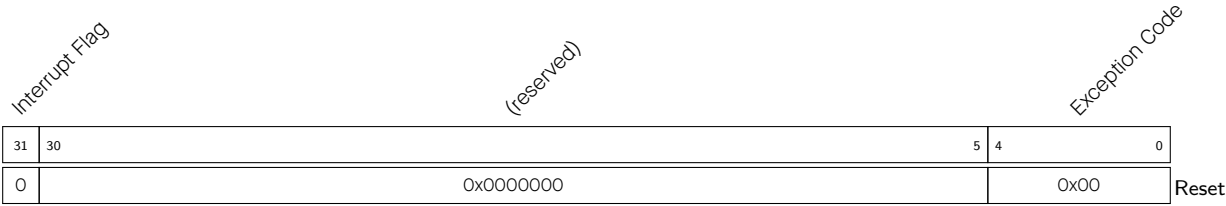
MSCRATCH 保存用户自定义的机器暂存信息。(R/W)

Register 2.7. mepc (0x341)

MEPC																															
31																															0
0x00000000																															
Reset																															

MEPC 配置机器陷阱/异常程序计数器。当 CPU 遇到陷阱时，此域将自动更新为 CPU 将要执行的指令的地址。(R/W)

Register 2.8. mcause (0x342)



Exception Code CPU 进入异常时，此域将自动更新为最近的陷阱或中断的唯一 ID。可能的异常 ID：

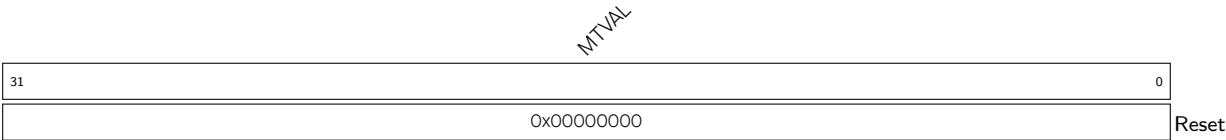
- 0x2：非法指令
- 0x3：硬件断点/观察点或 EBREAK
- 0x6：原子指令非对齐

注意：异常 ID 0x0（指令地址非对齐）不存在，因为 CPU 在取指时始终屏蔽地址的最低位。

(R/W)

Interrupt Flag CPU 进入陷阱时，此标志位将自动更新。如果被置位，则表示最近的陷阱是由中断引起。在异常情况下保持为 0。(R/W)

Register 2.9. mtval (0x343)



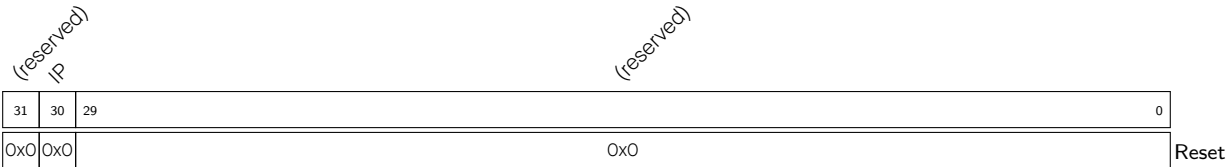
MTVAL 配置机器模式陷阱值。将自动更新为与异常有关的数据，该数据可能有助于处理该异常。根据异常编号有以下解读：

- 0x1：指令地址错误
- 0x2：指令 opcode 错误
- 0x5：存储器读操作的数据地址错误
- 0x7：存储器写操作的数据地址错误

注意：该寄存器不支持其他异常 ID 和中断

(R/W)

Register 2.10. mip (0x344)

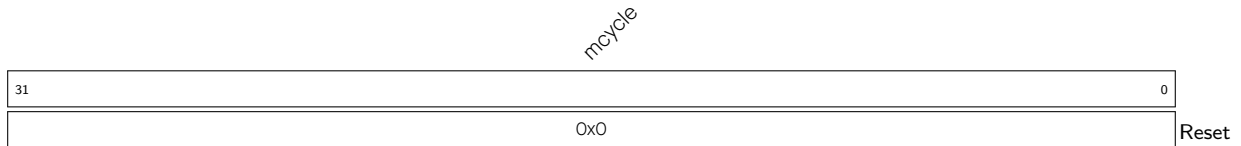


IP 配置中断的等待状态。

- 0：不等待
- 1：等待

(R/W)

Register 2.11. mcycle (0xB00)

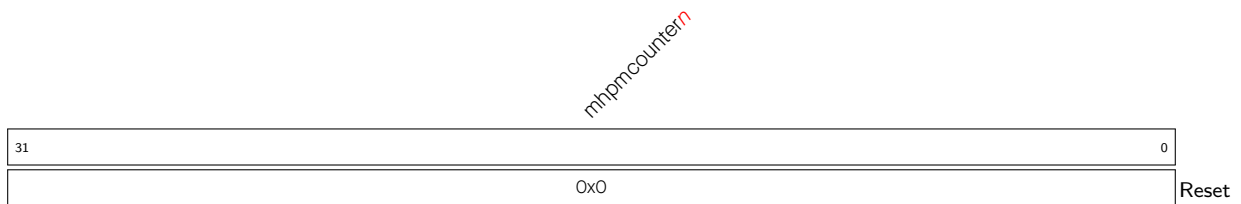


MCYCLE 配置时钟周期计数器的低 32 位。(R/W)

Register 2.12. minstret (0xB02)

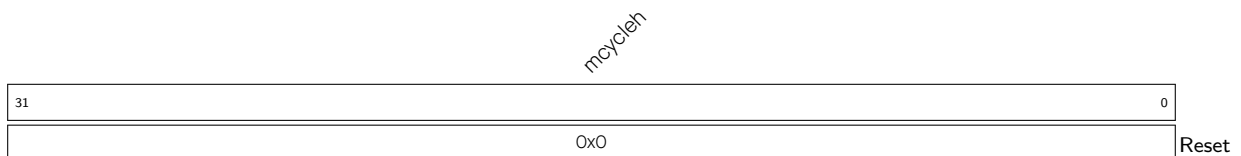


MINSTRET 配置指令计数器的低 32 位。(R/W)

Register 2.13. mhpmpcounter_{*n*} (*n*: 3-12) (0xB00+*n*)

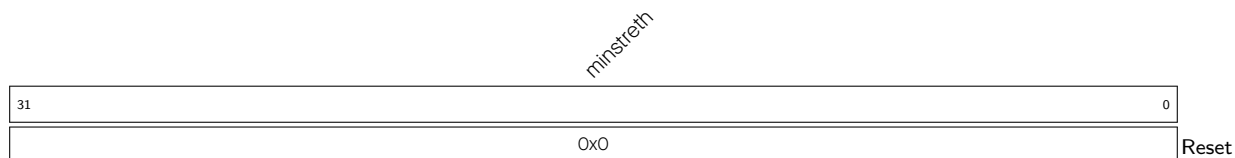
MHPMCOUNTER_{*n*} 配置性能计数器 *n* 的低 32 位。(R/W)

Register 2.14. mcycleh (0xB80)



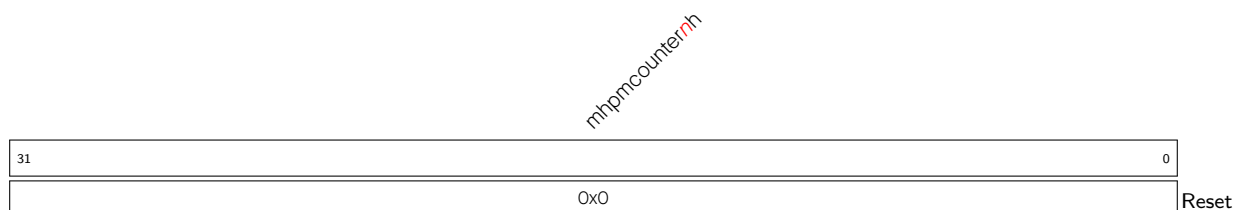
MCYCLEH 配置时钟周期计数器的高 32 位。(R/W)

Register 2.15. minstreth (0xB82)



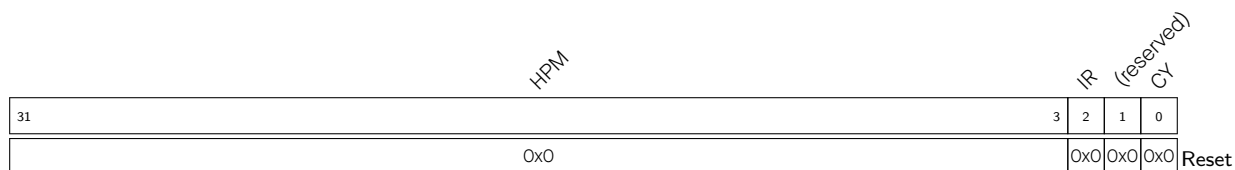
MINSTRETH 配置指令计数器的高 32 位。(R/W)

Register 2.16. mhpmmcounter n (n : 3-12)h (0xB80+ n)



MHPMCOUNTER n h 配置性能计数器 n 的高 32 位。(R/W)

Register 2.17. mcouninhibit (0x320)



CY 配置时钟周期计数器是否递增计数。

0: 不计数

1: 递增计数

(R/W)

IR 配置指令计数器是否递增计数。

0: 不计数

1: 递增计数

(R/W)

HPM 配置性能计数器 $n(n:3-12)$ 是否递增计数。

0: 不计数

1: 递增计数

(R/W)

2.3 中断和异常

低功耗 CPU 根据 RISC-V 指令集手册 V1.10 第二卷“特权架构”实现了中断和异常的处理。当 CPU 进入中断/异常处理程序时，内核将执行以下操作：

- 将当前 PC 保存到 `mepc` CSR 中
- 将 `mstatus` 的 `MIE` 保存到 `mstatus` 的 `MPIE`
- 将当前特权模式保存到 `mstatus` 的 `MPP`
- 将 `mstatus` 的 `MIE` 清 0
- 特权模式切换到机器模式
- 跳转到处理程序地址
 - 对于所有异常原因，跳转到 `mtvec` CSR 中向量表的基地址
 - 对于中断，跳转到 $mtvec + 4 * 30$
- 执行 `mret` 指令后，内核跳转到之前保存在 `mepc` CSR 中的程序计数器，并将 `mstatus` 的 `MIE` 恢复为 `mstatus` 的 `MPIE`，特权模式切换到 `MPP` 对应的特权模式

当内核启动时，向量表的基地址被初始化为启动地址 `0x50000000`。启动后可通过写入 `mtvec` CSR 更改基地址。有关配置和状态寄存器的详细信息，请参阅章节 2.2.1。

内核在复位后从 `0x50000080` 的地址开始取指。

2.3.1 中断

ESP32-C5 低功耗 CPU 仅支持 1 个中断入口，所有的中断事件都会跳转到该入口。ESP32-C5 低功耗 CPU 支持中断的外设源有：

- 电源管理单元 (PMU)
- 低功耗定时器 (RTC_TIMER)
- 低功耗 UART (LP_UART)
- 低功耗 I2C (LP_I2C)
- 低功耗 IO MUX (LP IO MUX)

各外设的中断请参考各外设章节。

2.3.2 中断处理

默认情况下，全局禁用中断，因为 `mstatus` 的 `MIE` 的复位值为 0。软件必须设置该位以启用全局中断。

1. 中断使能

- 置位 `mstatus` 的 `MIE` 域，启用全局中断使能
- 置位 `mie` CSR 的第 30 位，使能中断 30

2. 中断使能后，LP CPU 即可响应中断，同时去配置相应外设的中断，使其能够触发中断信号给 LP CPU。

3. 触发中断后 LP CPU 会跳转到 $mtvec + 4 * 30$ 。

4. 进入中断服务程序 (ISR) 后，用户需要查询 `LPPERI_INTERRUPT_SOURCE_REG` 获取触发中断的外设，并进行处理，注意如果有多个外设触发，依次处理每一个中断，直到所有的外设都处理完成，如果未处理完所有中断，会再次进入中断服务程序。
5. 中断清除，只需要清除各外设的中断信号即可。

2.3.3 异常

低功耗 CPU 支持 RISC-V 标准异常，能够触发以下异常：

表 2.3-1. LP CPU 异常原因

异常编号	描述
2	非法指令
3	断点 (EBREAK)
6	原子指令非对齐

2.4 调试

本节介绍如何调试和测试低功耗 CPU。调试功能由标准 JTAG 管脚提供，并符合 RISC-V 外部调试支持规范 v0.13 (RISC-V External Debug Support, Version 0.13)。

ESP32-C5 系统调试结构请参考章节 1 高性能处理器 *[to be added later]* 中的图 8。

用户与运行调试器 (Debugger，例如 GDB) 的调试主机 (DEBUG HOST，例如笔记本电脑) 进行交互。调试器通过调试转换器 (Debug Translator，可能包含硬件驱动，例如 OpenOCD) 与调试传输硬件 (DEBUG TRANSPORT，例如 ESP-Prog 适配器) 进行通信。调试传输硬件通过标准 JTAG 接口将调试主机连接到处理器内核的调试传输模块 (JTAG DTM)。JTAG DTM 使用调试模块接口 (DMI) 提供对调试模块 (DM) 的访问。

ESP32-C5 有两个 DTM 通过菊花链结构互联，其中 TAP0 用于连接 HP CPU，TAP1 用于连接 LP CPU。

HP 调试模块 (DM) 调试单核 HP CPU。LP 调试模块 (DM) 调试单核 LP CPU。

低功耗 CPU 实现了四个内核调试寄存器：`dcsr`、`dpc`、`dscratch0` 和 `dscratch1`。所有这些寄存器只能从调试模式访问，如果软件试图在内核未处于调试模式的情况下进行访问，则会触发非法指令异常。

2.4.1 特性

低功耗 CPU 支持以下调试特性：

- 向调试器提供有关实现的必要信息
- 支持暂停和恢复 CPU 内核
- CPU 内核寄存器（包括 CSR）可以由调试器读取/写入
- 可以通过调试器复位 CPU 内核
- 可以在软件断点（植入的断点指令）上暂停 CPU
- 硬件单步调试
- 支持两个硬件触发器（可用作断点/观察点），具体见章节 2.5

2.4.2 功能描述

调试机制遵守 RISC-V 外部调试支持规范 v0.13。有关调试功能的详细介绍，请参考 RISC-V 外部调试支持规范 v0.13 (RISC-V External Debug Support, Version 0.13)。

根据 RISC-V 外部调试支持规范，hart 存在 nonexistent、unavail、running、halted 等状态，低功耗 CPU 默认处于 unavail 状态，需要用户配置 [LPPERI_CPU_REG](#) 寄存器清除 unavail 状态，从而能够连接低功耗 CPU 进行调试。

2.4.3 寄存器列表

下表列出了低功耗 CPU 支持的调试 CSR。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
dcsr	调试控制和状态寄存器	0x7B0	R/W
dpc	调试 PC 寄存器	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W

所有调试模块寄存器的实现均符合 RISC-V 外部调试支持规范 v0.13。请参考 RISC-V 外部调试支持规范 v0.13 获取详细信息。

2.4.4 寄存器

以下是低功耗 CPU 支持的调试 CSR 的详细描述。

关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 2.18. dcsr (0x7B0)

xdebugver				(reserved)												ebreakm				(reserved)				ebreaku				(reserved)				cause				(reserved)				step		prv																																									
31				28				27								16				15				14				13				12				11				9				8				6				5				3				2				1				0															
0				1				0				0				0				0				0				0				0				0				0				0				0				0				0				0				0				0				1				1				Reset			

xdebugver 表示调试版本。

4: 存在外部调试支持
(RO)

ebreakm 配置机器模式下执行 EBREAK 指令的行为。

0: 触发异常，mcause 是 3
1: 进入调试模式
(R/W)

ebreaku 配置用户模式下执行 ebreak 指令的行为。

0: 如特权模式所述，触发异常，mcause 是 3
1: 进入调试模式
(R/W)

cause 表示进入调试模式的原因。当单个周期中有多个原因导致进入调试模式，会反映出具有最高优先级数值的那个原因。

1: 执行了一条 ebreak 指令（优先级 3）
2: 触发模块引起暂停（优先级 4）
3: haltreq 被置位（优先级 2）
4: step 被置位导致 CPU 单步执行（优先级 1）
其他值：保留，供日后使用
(RO)

step 当被置位且不处于调试模式时，内核将仅执行单个指令，然后进入调试模式。

如果指令由于异常而未能完成，则内核将在执行异常处理程序之前立即进入调试模式，并置位相应的异常寄存器。

设置该位不会屏蔽中断，这与 RISC-V 外部调试支持规范 v0.13 中的规定不同。
(R/W)

prv 保存 CPU 进入调试模式时候的特权级别。退出调试模式时，调试器可以更改此值以改变内核的特权级别。仅支持 **0x3**（机器模式）和 **0x0**（用户模式）。(RO)

Register 2.19. dpc (0x7B1)

dpc																																				Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
310																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

dpc 进入调试模式时，dpc 会更新为将要执行的下一条指令的地址。恢复时，PC 设置为存储在 dpc 中的地址。调试模式可能会修改 dpc。可在调试模式下访问。(R/W)

Register 2.20. dscratch0 (0x7B2)

dscratch0																															
31																															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

dscratch0 供调试模块内部使用。(R/W)

Register 2.21. dscratch1 (0x7B3)

dscratch1																															
31																															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

dscratch1 供调试模块内部使用。(R/W)

2.5 硬件触发器

2.5.1 特性

硬件触发器模块提供了断点和观察点功能，供调试使用。硬件触发器具有以下特性：

- 2 个独立触发单元
- 每个单元都可以配置为匹配程序计数器的地址
- 可以暂停执行并将控制权转移给调试器

2.5.2 功能描述

硬件触发器模块提供了 3 个 CSR，详情见章节 2.5.5。其中，**tdata1** 和 **tdata2** 是抽象 CSR，也就是说它们是用于访问某个触发单元中的内部寄存器的影子寄存器，一次访问一个触发单元。

要选择特定的触发单元，需要将相应的编号 (0-1) 写入 **tselect** CSR。当写入有效数值时，抽象 CSR **tdata1** 和 **tdata2** 将自动匹配该触发单元的内部寄存器。每个触发单元都有两个内部寄存器，即 **mcontrol** 和 **maddress**，它们分别与 **tdata1** 和 **tdata2** 匹配。

向 **tselect** 写入超过 1 的数值时会导致 **tselect** 写入 1。

由于软件或调试器可能需要知道所选触发器的类型以便正确解读 **tdata1** 和 **tdata2**，因此 **tdata1** 的 4 个位 (31-28) 对所选触发器的类型进行了编码。此域为只读访问属性，并且值始终为 0x2，代表触发器仅支持地址/数据匹配。因此，可以推断 **tdata1** 和 **tdata2** 会通过 **mcontrol** 和 **maddress** 被解读。RISC-V 外部调试支持规范 v0.13 提供了其他可能值的信息，但是该触发模块仅支持 0x2 类型。

一旦选定了触发单元，就可以通过置位 **mcontrol** CSR (**tdata1**) 中相应的域并将目标地址写入 **maddress** CSR (**tdata2**) 来对该触发单元进行配置。

2.5.3 触发执行流程

当触发器触发引起硬件线程暂停并进入调试模式时 (action = 1):

- dpc 被设置为当前 PC（在解码阶段）
- dcsr 的 cause 域被设置为 2，表示暂停是由于触发器触发引起

2.5.4 寄存器列表

下表列出了 CPU 可访问的的触发模块 CSR，只有在机器模式下才可以对它们进行读写。

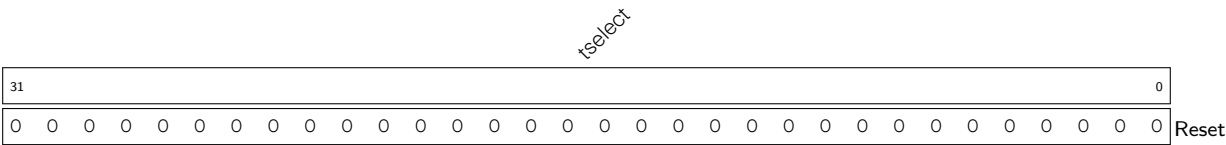
请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
mcontrol	tdata1 影子寄存器	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W

2.5.5 寄存器

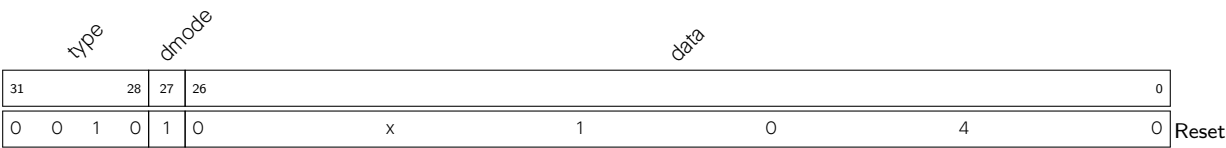
关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 2.22. tselect (0x7A0)



tselect 配置选择触发器单元。(R/W)

Register 2.23. tdata1 (0x7A1)



Register 2.24. tdata2 (0x7A2)



tdata2 配置抽象 tdata2 的内容。由于仅支持匹配类型 (0x2) 触发器, 此域将始终被解读为 **maddress**。
(R/W)

Register 2.25. mcontrol (0x7A1)

(reserved)				dmode		maskmax				hit		select		timing		sizelo		action		chain		match		m		(reserved)		s		u		execute		store		load									
31				28		27		26		21		20		19		18		17		16		15		12		11		10		7		6		5		4		3		2		1		0	
0x2				0				0x0		0		0		0		0		0001		0		0		0		0		0		0		0		0		0		0		0		Reset			

dmode 与 tdata1 的 dmode 一致。(RO)

maskmax 表示硬件支持的最大自然对齐的二次幂范围。

0: 一个字节的范围，仅支持精确匹配

其他值：不支持

(RO)

hit 硬件未实现，始终为 0。(RO)

select 选择地址匹配或数据匹配。

0: 匹配虚拟地址

1: 匹配读取或写入的数据值或执行的指令

注意：只实现了地址匹配，此域总为 0。

(RO)

timing 表示触发器何时被触发。

0: 当匹配后, 触发器会在指令执行前触发

1: 当匹配后, 触发器会在指令执行后触发

注意：此域总为 0。

(RO)

szelo 仅支持任意大小匹配，此域总为 0。(RO)

action 配置选定的触发器在触发时进行以下操作。

0x0: 引起断点异常

0x1: 进入调试模式（仅当 dmode = 1 时有效）

注意：仅支持进入调试模式，此域总为 1。

(RO)

CHAIN 硬件未实现，始终为 0。(RO)

match 配置触发器进行数据/指令地址的匹配操作。

0x0: 精确字节匹配, 即与访问中某个字节对应的地址必须精确匹配

`maddress` 的值。0x1: NAPOT 匹配, 即访问中至少有一个字节处于 `maddress` 中规定的 NAPOT 区域。

注意：仅支持精确字节匹配，此域总为 0。

(R/W)

m 置位使选定的触发器在机器模式下操作。(RO)

S 置位使选定的触发器在监督模式下操作。不支持在监督模式下操作，此域总为 0。(RO)

u 置位使选定的触发器在用户模式下操作。不支持在用户模式下操作，此域总为 0。(RO)

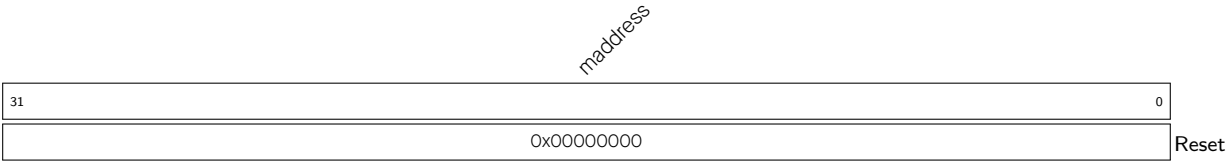
见下页……

Register 2.25. mcontrol (0x7A1)

[接上页...](#)

- execute** 置位使选定的触发器匹配指令的虚拟地址。(R/W)
- store** 置位使选定的触发器匹配存储器写操作的虚拟地址。硬件不支持，此域总为 0。(RO)
- load** 置位使选定的触发器匹配存储器读操作的虚拟地址。硬件不支持，此域总为 0。(RO)

Register 2.26. maddress (0x7A2)



maddress 配置选定的触发器执行匹配操作时使用的地址。(R/W)

2.6 性能计数器

低功耗 CPU 实现了时钟周期计数器 `mcycle(h)`、指令计数器 `minstret(h)` 以及 10 个事件计数器 `mhpmcounter(n:3-12)`。时钟周期计数器和指令计数器始终可用且为 64 位，其余性能计数器为 40 位。默认情况下，所有计数器在复位后启用。通过 `mcountinhibit` CSR 中的相应位来单独启用/禁用对应的计数器。

每个计数器的计数事件都是固定的，参考表 2.6-1。

表 2.6-1. 性能计数器

计数器	计数事件
mcycle	统计时钟周期数
minstret	统计指令数量
mhpmcounter3	统计内存访问的等待周期
mhpmcounter4	统计取指等待周期
mhpmcounter5	统计内存读操作的数量，非对齐读视作两次。
mhpmcounter6	统计内存写操作的数量，非对齐写视作两次。
mhpmcounter7	统计无条件跳转指令的数量 (jal, jr, jalr)
mhpmcounter8	统计分支指令的数量
mhpmcounter9	统计分支指令跳转的数量
mhpmcounter10	统计压缩指令的数量
mhpmcounter11	统计乘法指令的等待周期
mhpmcounter12	统计除法指令的等待周期

2.7 系统访问

2.7.1 存储器

ESP32-C5 低功耗 CPU 可以访问的存储器资源有 LP SRAM 和 HP SRAM，详情参考章节 4 [系统和存储器](#)。其中：

- LP SRAM：16 KB, 地址范围为 0x5000_0000 ~ 0x5000_3FFF，可进行取指、读数据、写数据等操作
- HP SRAM：384 KB, 地址范围为 0x4080_0000 ~ 0x4085_FFFF，可进行取指、读数据、写数据等操作

说明：

LP CPU 访问 HP SRAM 的延迟很大，但可以无延迟访问 LP SRAM。

低功耗 CPU 支持原子指令集，低功耗 CPU 和高性能 CPU 可通过原子指令访问存储器，从而实现内存访问的原子性。关于原子指令集的细节请参考 RISC-V 指令集手册 v2.2 第一卷“非特权架构” (RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 2.2)。

2.7.2 外设

章节 4 [系统和存储器](#) 表4.3-2 详细列出了低功耗 CPU 可访问的外设及基地址。

2.8 事件任务矩阵功能

在 ESP32-C5 中，LP CPU 支持事件任务矩阵 (ETM) 功能，即可以通过任意外设的 ETM 事件触发 LP CPU 的 ETM 任务，或者通过 LP CPU 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 [事件任务矩阵 \(ETM\)](#)。这里仅介绍与 LP CPU 相关的 ETM 任务和 ETM 事件。

LP CPU 可接收的 ETM 任务有：

- ULP_TASK_WAKEUP_CPU：唤醒 LP CPU

LP CPU 可产生的 ETM 事件有：

- ULP_EVT_ERR_INTR：表示 LP CPU 发生异常
- ULP_EVT_START_INTR：表示 LP CPU 时钟打开

2.9 睡眠和唤醒流程

2.9.1 特性

- 低功耗 CPU 工作在低功耗系统中，可以在高性能系统工作或进入睡眠的时候，独立睡眠、唤醒及工作
- 低功耗 CPU 可以根据软件运行状态，主动配置寄存器进入睡眠状态
- 低功耗 CPU 可以被如下事件唤醒：
 - 主核置位寄存器 [PMU_HP_TRIGGER_LP](#)
 - LP IO 的中断状态

- ETM 事件
- RTC 定时器超时
- LP UART 接收一定数量的 RX 脉冲，需要使能 `LPPERI_LP_UART_WAKEUP_EN`

2.9.2 流程

低功耗 CPU 默认处于睡眠状态，低功耗 CPU 的睡眠唤醒模块将按照如下流程唤醒开始工作以及进入睡眠。

LP CPU 的唤醒源配置如表 2.9-1 所示。

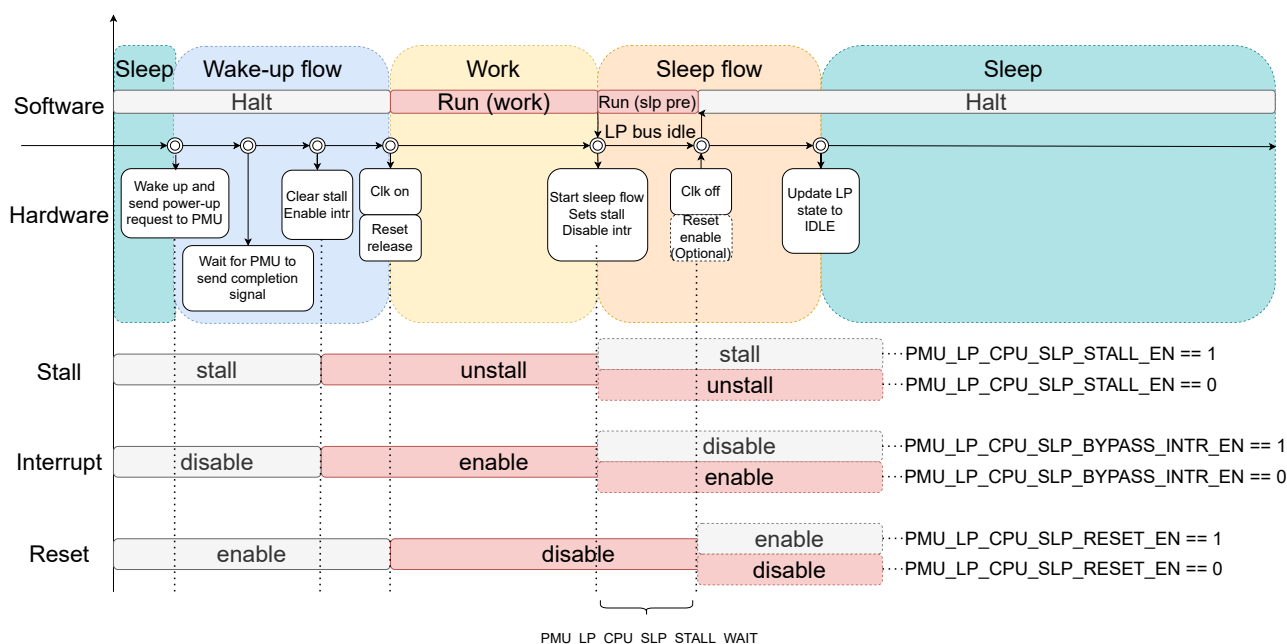


图 2.9-1. LP CPU 唤醒和睡眠流程

LP CPU 上电后的首次启动取决于 HP CPU 对能唤醒使能信号和唤醒源的配置。

• LP CPU 的初始化

- 初始化 LP 存储器。
- 启动 LP CPU。由于 LP CPU 的启动依赖于唤醒过程，所以推荐使用 `PMU_HP_TRIGGER_LP` 寄存器来开始 LP CPU 的初始化，方法如下：

- * 将 `PMU_LP_CPU_WAKEUP_EN` 设置为 0x1
- * 将 `PMU_HP_TRIGGER_LP` 设置为 0x1
- * LP CPU 被唤醒后开始运行

• 唤醒启动流程：

- 睡眠唤醒模块接收到唤醒信号向 PMU 发送上电请求。
- 如果当前功耗状态（时钟，电源等）已经满足低功耗 CPU 需求，则 PMU 立刻回复完成信号，否则将调整功耗状态再回复完成信号。
- 睡眠唤醒模块关闭低功耗 CPU 的 STALL 状态，并打开中断接收使能。

- 睡眠唤醒模块开启时钟并释放复位（如果睡眠没有使能复位则忽略该步骤），开始工作。
- 睡眠流程：
 - 低功耗 CPU 配置寄存器 `PMU_LP_CPU_SLEEP_REQ`，睡眠唤醒模块启动睡眠流程。
 - 如果 `PMU_LP_CPU_SLP_STALL_EN` 为 1，睡眠唤醒模块则使能低功耗 CPU 的 STALL 状态，如果为 0，则不使能低功耗 CPU 的 STALL 状态。
 - 睡眠唤醒模块等待 `PMU_LP_CPU_SLP_STALL_WAIT` 个低功耗 CPU 时钟周期后，关闭低功耗 CPU 的时钟，如果 `PMU_LP_CPU_SLP_RESET_EN` 为 1，则使能低功耗 CPU 的复位。

2.9.3 唤醒源

表 2.9-1. 唤醒源

寄存器值 ¹	唤醒源	描述
0x1	寄存器 <code>PMU_HP_TRIGGER_LP</code>	通过主核置位寄存器 <code>PMU_HP_TRIGGER_LP</code> 可唤醒低功耗 CPU，置位 <code>PMU_HP_SW_TRIGGER_INT_CLR</code> 可清除唤醒源
0x2	LP UART	LP UART 接收一定数量的 RX 脉冲，需要使能 <code>LPPERI_LP_UART_WAKEUP_EN</code> ，详情可见章节 25 UART 控制器 (UART)
0x4	LP IO	该唤醒源使用 LP IO 的中断状态寄存器信号，详情可见章节 5 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)
0x8	ETM	可接收来自 ETM 的唤醒源，详情可见章节 8 事件任务矩阵 (ETM)
0x10	RTC 定时器	使用 RTC 定时器的 target 1 超时中断控制，详情可见章节 4 低功耗管理 [to be added later]

¹ 寄存器 `PMU_LP_CPU_WAKEUP_EN` 的值

2.10 寄存器列表

本小节的所有地址均为相对于低功耗外设的基地址的地址偏移量（相对地址），具体基地址请见章节 [4 系统和存储器](#) 中的表 [4.3-2](#)。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

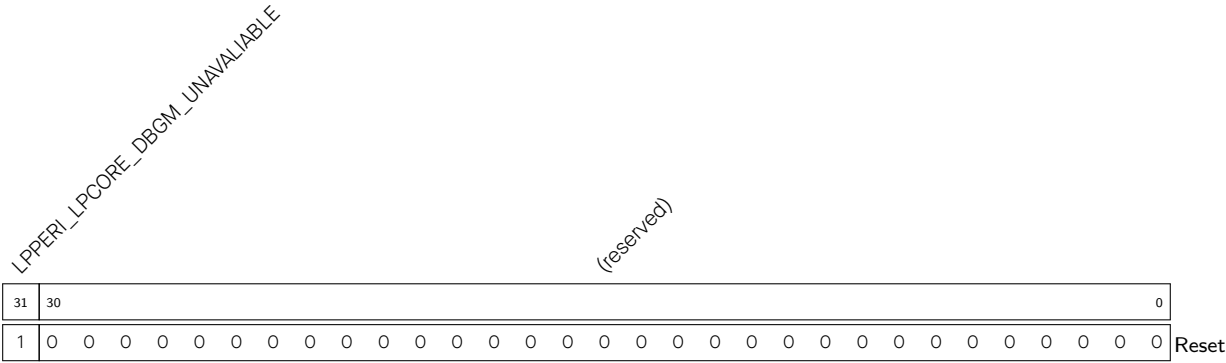
名称	描述	地址	访问
<code>LPPERI_CPU_REG</code>	LP CPU 控制寄存器	0x000C	R/W
<code>LPPERI_INTERRUPT_SOURCE_REG</code>	LP CPU 中断状态寄存器	0x0020	RO

2.11 寄存器

本小节的所有地址均为相对于低功耗外设的基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 2.27. LPPERI_CPU_REG (0x000C)



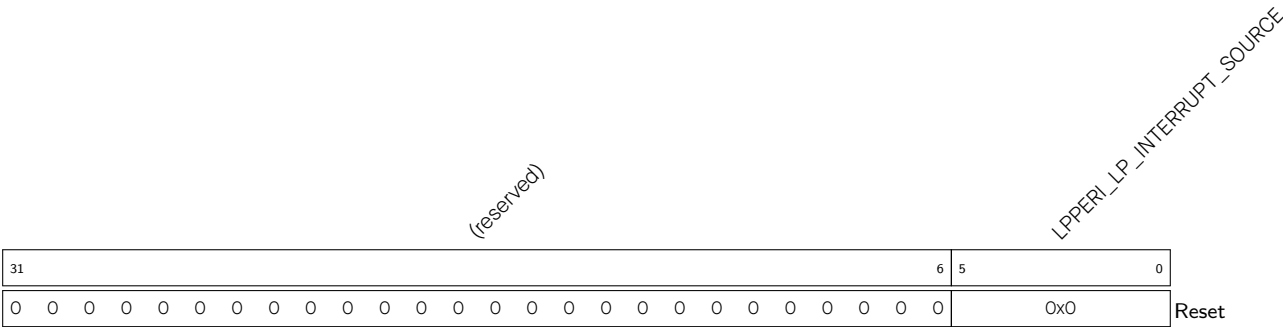
LPPERI_LPCORE_DBGM_UNAVALIABLE 配置 LP CPU 状态。

0: LP CPU 可以连接 JTAG

1: LP CPU 不可用，不能连接 JTAG

(R/W)

Register 2.28. LPPERI_INTERRUPT_SOURCE_REG (0x0020)



LPPERI_LP_INTERRUPT_SOURCE 表示 LP 中断源。

Bit 5: PMU_LP_INT

Bit 4: 保留

Bit 3: RTC_TIMER_LP_INT

Bit 2: LP_UART_INT

Bit 1: LP_I2C_INT

Bit 0: LP_IO_INT

(RO)

第 3 章

通用 DMA 控制器 (GDMA)

3.1 概述

通用直接存储访问 (General Direct Memory Access, GDMA) 用于在外设与存储器之间以及存储器与存储器之间提供高速数据传输。软件可以在无需 CPU 干预的情况下通过 GDMA 快速搬移数据，从而降低了 CPU 的工作负载，提高了效率。

GDMA 有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。这 6 个通道被支持 GDMA 功能的外设所共享，用户可以将通道分配给任何支持 GDMA 功能的外设。GDMA 支持的外设包括 PARLIO、ADC、UHCI、AES、SHA、I2S、通用 SPI (GP-SPI)。ESP32-C5 的 GDMA 可以访问片内存储器和片外存储器。

GDMA 控制器支持通道间可配置优先级及权重仲裁以管理外设不同的带宽需求。

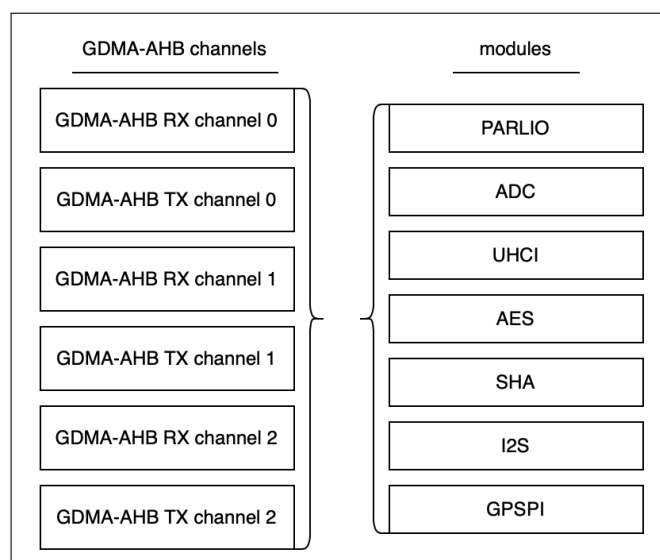


图 3.1-1. 共用 GDMA 通道的模块

3.2 特性

GDMA 具有以下特性：

- AHB 总线架构
- 数据传输以字节为单位，传输数据量可软件编程
- 支持任意地址和大小 (size) 访问
- 对齐要求：
 - 描述符存储地址：字对齐

- 数据地址和长度：无要求
- 支持链表
- 访问存储器时，支持 INCR 突发传输
- 三个传输通道和三个接收通道
- 可配置支持 GDMA 的外设连接任一通道
- 支持通道间优先级和权重仲裁配置
- 支持存储搬运功能

3.3 架构

ESP32-C5 中所有需要进行高速数据传输的模块都具有 GDMA 功能。GDMA 控制器与 CPU 的数据总线使用相同的地址空间访问存储器。图 3.3-1 为 GDMA 控制器基本架构图。

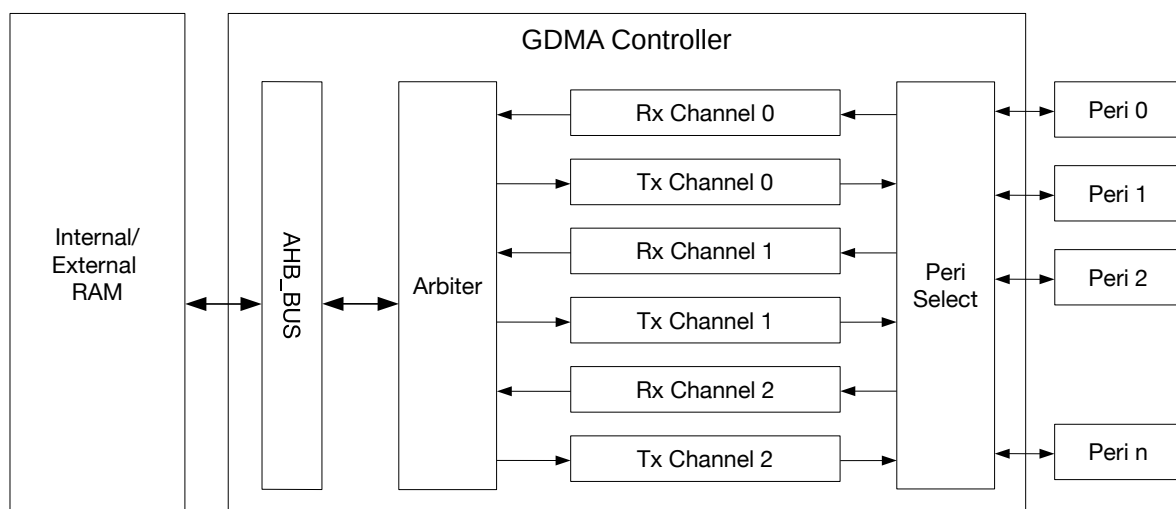


图 3.3-1. GDMA 控制器的架构

GDMA 控制器有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。每个通道可选择与不同的外设相连，从而实现通道资源由外设共享。

GDMA 通过 AHB_BUS 将数据存入存储器，或者将数据从存储器取出。在通过总线传输数据之前，GDMA 采用可配置的仲裁机制对每个通道的读写请求进行仲裁。存储器的具体使用范围详见章节 4 系统和存储器。

软件可以通过挂载链表的方式来使用 GDMA 控制器。GDMA 链表存储在存储器中。这些链表包括 `outlink $n$` 与 `inlink $n$` ，本文以 n 来表示通道号， n 为 0 ~ 2。GDMA 从存储器中取得链表，然后根据 `outlink $n$` 中的内容将相应存储器中的数据发送出去，也可根据 `inlink $n$` 中的内容将接收的数据存入指定存储器地址空间。

3.4 功能描述

3.4.1 链表



图 3.4-1. 链表结构图

图 3.4-1 所示为链表的结构图。发送链表与接收链表结构相同。每个链表由一个或者若干个描述符构成，一个描述符由 3 个字组成。链表应存放在存储器中，供 GDMA 控制器使用。描述符每一字段的意义如下：

- owner (DW0) [31]: 表示当前描述符对应的 buffer 允许的操作者。
0: 允许的操作者为 CPU。
1: 允许的操作者为 GDMA 控制器。
在 GDMA 使用完该描述符对应的 buffer 后，对于接收描述符，硬件默认会自动将该位清零；对于发送描述符，需要将 AHB_DMA_OUT_AUTO_WRBK_CHn 置 1，硬件才会自动将该位清零。软件也可通过置位 AHB_DMA_OUT_LOOP_TEST_CHn 或 AHB_DMA_IN_LOOP_TEST_CHn 来关闭硬件自动清零的功能。软件在挂载链表时需要将该位置 1。
- 注意：**本文以 AHB_DMA_OUT 开头的寄存器对应 TX 通道寄存器，以 AHB_DMA_IN 开头的寄存器对应 RX 通道寄存器。
- suc_eof (DW0) [30]: 表示一个描述符对应的数据成功传输后是否触发 AHB_DMA_IN_SUC_EOF_CHn_INT 或 AHB_DMA_OUT_EOF_CHn_INT 中断。
0: 当前描述符成功传输后不会触发中断；
1: 当前描述符成功传输后触发中断。
对于接收描述符，需要软件将该位写 0，硬件会在接收到包含 EOF 标志的数据后将该位置 1。
对于发送描述符，需要软件按需要将描述符中的该位置 1。如果软件在某个描述符中将该位配置为 1，则 GDMA 在处理完该描述符时，会在发送给外设的数据中加入 EOF 标志，告知外设该段数据是一个阶段性结束。
- Reserved (DW0) [29]: 保留。此位为无关项。
- err_eof (DW0) [28]: 表示接收结束错误标志。
0: 接收数据没有错误。
1: 接收数据有错误。
该位只用于 UHCI 或 PARLIO 利用 GDMA 接收数据。对于接收描述符，硬件在收完描述符对应的数据段并检测到接收数据错误会将该位置 1。
- Reserved (DW0) [27:24]: 保留。

- length (DW0) [23:12]: 表示当前描述符对应的 buffer 中的有效字节数。对于发送描述符, 该段由软件填写, 表示从 buffer 中读取数据时需要读取的字节数; 对于接收描述符, 该段由硬件使用完该 buffer 后或者接收到最后一个数据时自动填写, 表示 buffer 中存储的有效字节数。
- size (DW0) [11:0]: 表示当前描述符对应的 buffer 容量的字节数。
- buffer address pointer (DW1): buffer 的地址。
- next descriptor address (DW2): 下一个描述符的地址。如果当前描述符为链表中最后一个描述符时, 该值可以为 0。该地址必须指向存储器的地址空间。

说明:

需要注意, GDMA 的链表描述符本身存储地址要求 4 字节对齐。

用 GDMA 接收数据时, 如果接收数据的长度小于当前描述符指定的 buffer size, 那么下一个描述符对应的接收数据不会占用该 buffer 的剩余空间。如果接收数据的长度大于当前描述符指定的 buffer size, 那么 GDMA 完成当前描述符后继续获取下一个描述符来完成数据传输。

3.4.2 外设到存储器及存储器到外设的数据传输

GDMA 支持存储器到外设及外设到存储器的数据传输, 分别对应 TX 及 RX 功能。TX 通道通过 `outlinkn` 实现将指定存储器中的数据搬运到外设的发送端; RX 通道通过 `inlinkn` 实现将外设接收到的数据搬运到存储器中。

每个 RX/TX 通道均可以被配置连接到任意一个支持 GDMA 功能的外设, 表 3.4-1 所示为通过寄存器配置通道与其对应外设的关系。其中“Dummy-*n*”是 GDMA 在实现存储器到存储器数据搬运功能时可选配的寄存器值。当其中一个通道已经与某一个外设连接时, 其他通道将不能配置为与该外设连接。

表 3.4-1. GDMA 配置寄存器与外设选择关系表

<code>AHB_DMA_PERI_IN_SEL_CH_n</code> <code>AHB_DMA_PERI_OUT_SEL_CH_n</code>	外设
0	Dummy-0
1	GP-SPI
2	UHCI
3	I2S
4 ~ 5	Dummy-4 ~ 5
6	AES
7	SHA
8	ADC
9	PARLIO
10 ~ 15	Dummy-10 ~ 15
16 ~ 63	无效值

3.4.3 存储器到存储器数据传输

GDMA 支持存储器到存储器的数据传输。置位 `AHB_DMA_MEM_TRANS_EN_CHn`, TX 通道 *n* 的输出将与 RX 通道 *n* 的输入相连, 从而使能存储器到存储器的数据传输功能。需要注意的是, 一个 TX 通道只与其编号对应的 RX 通道相连而实现存储器到存储器的数据传输, 并且需要将 `AHB_DMA_PERI_IN_SEL_CHn` 和 `AHB_DMA_PERI_OUT_SEL_CHn` 配置成相同且对应 Dummy 的值。

3.4.4 启动 GDMA

软件通过挂载链表的方式来使用 GDMA。对于接收数据，软件挂载好接收链表并准备好接收数据，配置 `AHB_DMA_INLINK_ADDR_CH $n$` 字段指向第一个接收链表描述符，置位 `AHB_DMA_INLINK_START_CH $n$` 位启动 GDMA。对于发送数据，软件挂载好发送链表并准备好发送数据，配置 `AHB_DMA_OUTLINK_ADDR_CH $n$` 字段指向第一个发送链表描述符，置位 `AHB_DMA_OUTLINK_START_CH $n$` 位启动 GDMA。
`AHB_DMA_INLINK_START_CH $n$` 与 `AHB_DMA_OUTLINK_START_CH $n$` 位由硬件自动清零。

有时您可能想要在 DMA 数据传输已经开始后追加更多描述符。要挂载更多描述符，原本看似只需清空已挂载链表最后一个描述符的 EOF 位，并将该描述符的 next descriptor address (DW2) 字段配置为新链表第一个描述符的地址。但如果 DMA 数据传输已经或即将结束，这个方法便无法奏效。GDMA 控制器有专门的逻辑来确保数据传输继续或重启：如果数据传输仍在进行，GDMA 控制器会确保顾及到新追加的描述符；如果数据传输已经结束，GDMA 控制器会重启数据传输，传输新追加的描述符。这个逻辑由 Restart 功能实现。

软件使用 Restart 功能时，需要重写已挂载链表的最后一个描述符，使其第三个字中的内容（即 DW2）指向新链表的首地址；然后置位 `AHB_DMA_INLINK_RESTART_CH $n$` 或者 `AHB_DMA_OUTLINK_RESTART_CH $n$` （这两个位由硬件自动清零），如图 3.4-2 所示，硬件会在读取已挂载链表的最后一个描述符时，获取新挂载链表的地址，从而继续处理新挂载的链表。

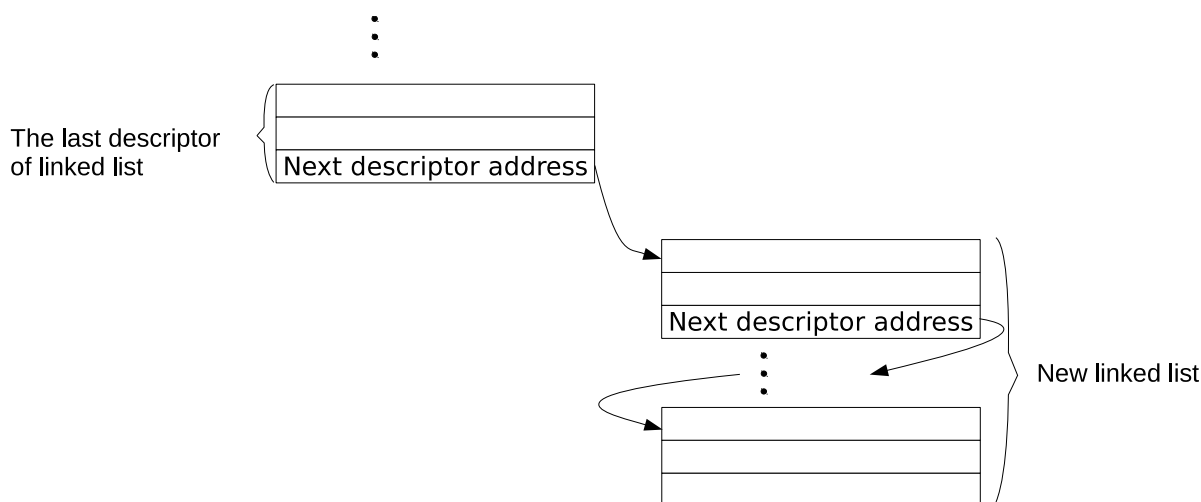


图 3.4-2. 链表关系图

3.4.5 读链表

软件在配置并启动 GDMA 后，GDMA 会从存储器中读取链表。GDMA 会检查读入的链表描述符是否正确。只有当链表描述符通过检查时，GDMA 对应的通道才会开始搬运数据。当链表描述符没有通过检查，硬件将触发描述符错误中断（`AHB_DMA_IN_DSCR_ERR_CH $n$ _INT` 或者 `AHB_OUT_DSCR_ERR_CH $n$ _INT`），同时该通道将会处于阻塞状态，停止工作。

描述符检查项包括：

- `AHB_DMA_IN_CHECK_OWNER_CH $n$` 或者 `AHB_DMA_OUT_CHECK_OWNER_CH $n$` 置 1 时，检查描述符的 owner 位。如果该位为 0，表示当前操作者为 CPU，则不通过检查。`AHB_DMA_IN_CHECK_OWNER_CH $n$` 或者 `AHB_DMA_OUT_CHECK_OWNER_CH $n$` 置 0 时，不检查描述符的 owner 位。
- 检查描述符本身的存储地址是否在配置的存储器地址空间内。如果 GDMA 的描述符地址在 `AHB_DMA_ACCESS_INTR_MEM_START_ADDR ~ AHB_DMA_ACCESS_INTR_MEM_END_ADDR` 值的范围内，则通过检查。请参见章节 3.4.7)。

软件在检查到通道描述符错误中断后，需要准备新描述符，并置位 `AHB_DMA_OUTLINK_START_CHn` 或者 `AHB_DMA_INLINK_START_CHn` 位启动 GDMA。

3.4.6 数据传输结束标志

注意：本章中提到发送链表描述符的 EOF 为描述符中的第一个字中的 bit 30 指示的 `suc_eof`，接收链表描述符的 EOF 可以为 bit 30 指示的 `suc_eof` 和 bit 28 指示的 `err_eof`。

GDMA 通过 EOF 来指示对应描述符所需传输的数据段传输结束。

发送数据时，产生 EOF 和 TOTAL EOF 两种中断：

- `AHB_DMA_OUT_EOF_CHn_INT`：链表中的任意一个描述符有配置 `suc_eof` 标志位，则当该描述符对应的数据传输完成后，GDMA 都会产生 EOF 中断。置位 `AHB_DMA_OUT_EOF_CHn_INT_ENA` 位使能该中断。
- `AHB_DMA_OUT_TOTAL_EOF_CHn_INT`：如果该链表中的最后一个描述符配置了 `suc_eof` 标志位，则当该链表的最后一个描述符对应的数据传输完成后，GDMA 会产生 TOTAL EOF 中断。置位 `AHB_DMA_OUT_TOTAL_EOF_CHn_INT_ENA` 位使能该中断。

接收数据时，可以产生以下中断：

- `AHB_DMA_IN_SUC_EOF_CHn_INT`：表示带有 EOF 标志的数据段接收完成。置位 `AHB_DMA_IN_SUC_EOF_CHn_INT_ENA` 位使能该中断。
- (仅 UHCI 和 PARLIO) `AHB_DMA_IN_ERR_EOF_CHn_INT`：表示对应描述符所需传输的数据段接收完成但接收数据有错误。置位 `AHB_DMA_IN_ERR_EOF_CHn_INT_ENA` 使能该中断。需要注意的是，只有当通道连接的外设为 UHCI 或 PARLIO 时，才支持该中断。

软件在检测到 `AHB_DMA_OUT_TOTAL_EOF_CHn_INT` 或 `AHB_DMA_IN_SUC_EOF_CHn_INT` 中断时，可以通过获取 `AHB_DMA_OUT_EOF_DES_ADDR_CHn` 或 `AHB_DMA_IN_SUC_EOF_DES_ADDR_CHn` 字段中的值，获取当前已经完成数据传输的描述符的存储地址。这样，软件可以知道哪些描述符已经被使用并根据需要回收描述符。RX 方向还支持记录 `AHB_DMA_IN_ERR_EOF_CHn_INT` 中断触发时描述符的存储地址，可通过 `AHB_DMA_IN_ERR_EOF_DES_ADDR_CHn` 字段获取。

3.4.7 访问存储器

GDMA 任意 RX/TX 通道均可以访问存储器，其可访问的地址空间为寄存器

`AHB_DMA_ACCESS_INTR_MEM_START_ADDR` ~ `AHB_DMA_ACCESS_INTR_MEM_END_ADDR` 的配置值区间。

为加速数据传输速率，支持突发传输模式。GDMA 默认不使能突发传输模式，用户可以通过配置 `AHB_DMA_IN_DATA_BURST_MODE_SEL_CHn` 和 `AHB_DMA_OUT_DATA_BURST_MODE_SEL_CHn` 为 0、1、2 将 AHB 总线突发传输长度分别配置为 SINGLE、INCR4 或 INCR8。

GDMA 访问配置的存储器地址空间时，链表描述符参数无对齐要求。

3.4.8 仲裁

为了确保及时响应通用 SPI 等高速低延迟外设的请求，GDMA 引入通道优先级仲裁和通道权重仲裁两种仲裁机制。

- 优先级仲裁方式：GDMA 支持 6 (0 ~ 5) 个等级的优先级，数字越大优先级越高。若多个通道在同一时刻存在数据传输需求，GDMA 会按照通道优先级从高到低顺序响应每个通道的数据传输请求；其数值越大，

对应的优先级越高，请求响应越及时

- 权重仲裁方式：

- 每个通道的权重（即令牌数）可以配置为 0 ~ 15 之间的任意值。GDMA 将时间按照一定的总线 (AHB) 时钟周期长度划分成片，在每个时间片内，各个通道的数据传输请求次数应该按照通道所配置的令牌数进行分配，在任意一个时间片内，某通道每请求一次数据传输消耗一次令牌，若某个通道在某个时间片内消耗尽令牌数，则该通道请求将不再被响应，直到仲裁时间达到时间片规定长度或者所有通道的令牌数均已经消耗尽，此时将刷新令牌数，进入下一个时间片周期内。
- 若通道令牌数配置为非零值，即使无数据传输请求，也会在每个时间片内等待所有通道的令牌数耗尽，不会提前退出当前仲裁时间片。通道无数据传输请求且令牌值非 0 时，会导致一些带宽浪费，基于此，GDMA 提供额外的权重仲裁优化开关。权重仲裁优化打开时，如果通道无数据传输需求，则该通道令牌数会被认为是 0，不再参与仲裁，当所有通道均无数据传输请求时，仲裁器提前退出当前时间片，以此提高仲裁响应时间，提高传输速率。

说明：

- 若同时配置优先级仲裁和通道权重仲裁，且优先级不同，权重相同，则在各个通道传输相同的数据量，且数据量较少的前提下，优先级高的通道优先传输完成。
- 若优先级不同且权重也不同，各个通道传输数据量相同且较大，则权重较大的通道是能优先分配到大带宽的，会优先完成传输。

3.5 事件任务矩阵功能

在 ESP32-C5 中，GDMA 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GDMA 的 ETM 任务，或者通过 GDMA 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 [事件任务矩阵 \(ETM\)](#)。这里仅介绍与 GDMA 相关的 ETM 任务和 ETM 事件。

GDMA 可接收的 ETM 任务有：

- GDMA_TASK_IN_START_CH n ：触发时开启 GDMA 的接收通道 n 进行数据传输。
- GDMA_TASK_OUT_START_CH n ：触发时开启 GDMA 的发送通道 n 进行数据传输。

说明：

以上两个 ETM 任务与 CPU 配置 [AHB_DMA_INLINK_START_CH \$n\$](#) 、[AHB_DMA_OUTLINK_START_CH \$n\$](#) 实现的功能相同。当 [AHB_DMA_IN_ETM_EN_CH \$n\$](#) 或 [AHB_DMA_OUT_ETM_EN_CH \$n\$](#) 为 1 时，只有 ETM 可以配置对应方向及对应通道 GDMA 的启动，当 [AHB_DMA_IN_ETM_EN_CH \$n\$](#) 或 [AHB_DMA_OUT_ETM_EN_CH \$n\$](#) 为 0 时，仅可通过 CPU 使能对应的 GDMA。

GDMA 可产生的 ETM 事件有：

- GDMA_EVT_IN_DONE_CH n ：表示通道 n 接收方向链表描述符指向的数据接收完成。
- GDMA_EVT_IN_SUC_EOF_CH n ：通道 n 接收方向 EOF 位为 1 的链表描述符指向的数据接收完成指示。
- GDMA_EVT_IN_FIFO_EMPTY_CH n ：通道 n 接收方向 FIFO 非空到空状态改变指示。
- GDMA_EVT_IN_FIFO_FULL_CH n ：通道 n 接收方向 FIFO 非满到满状态改变指示。
- GDMA_EVT_OUT_DONE_CH n ：通道 n 表示通道 n 发送方向链表描述符指向的数据发送完成。

- GDMA_EVT_OUT_EOF_CH n : 通道 n 发送方向 EOF 位为 1 的链表描述符指向的数据发送完成指示。
- GDMA_EVT_OUT_TOTAL_EOF_CH n : 通道 n 发送方向最后一个并且 EOF 位为 1 的链表描述符指向的数据发送完成指示。
- GDMA_EVT_OUT_FIFO_EMPTY_CH n : 通道 n 发送方向 FIFO 非空到空状态改变指示。
- GDMA_EVT_OUT_FIFO_FULL_CH n : 通道 n 发送方向 FIFO 非满到满状态改变指示。

在具体应用中, GDMA 的 ETM 事件可以用来触发 GDMA 的 ETM 任务, 例如, GDMA_EVT_OUT_TOTAL_EOF_CH0 事件可以触发 GDMA_TASK_IN_START_CH1 任务, 从而触发新一轮的 GDMA 操作。

3.6 中断

ESP32-C5 的 GDMA 可以生成以下中断信号并发送给[中断矩阵](#)。

- GDMA_IN_CHO_INTR
- GDMA_IN_CH1_INTR
- GDMA_IN_CH2_INTR
- GDMA_OUT_CHO_INTR
- GDMA_OUT_CH1_INTR
- GDMA_OUT_CH2_INTR

这些中断信号由 GDMA 的内部中断源生成。

以下中断源生成 GDMA_IN_CH n _INTR (n 为 0 ~ 2) 中断信号:

- AHB_DMA_IN_DONE_CH n _INT: 对于接收通道 n , 当一个接收链表描述符对应的数据接收完成时触发此中断。
- AHB_DMA_IN_SUC_EOF_CH n _INT: 对于接收通道 n , 当一个接收链表描述符对应的数据接收完成, 并且描述符的 suc_eof 为 1 时触发此中断。
- AHB_DMA_IN_ERR_EOF_CH n _INT: 对于接收通道 n , 当接收的描述符对应数据段中有错误发生时触发此中断。(该中断只用于外设选择 UHCI (UART0/UART1) 或 PARLIO 时)。
- AHB_DMA_IN_DSCR_EMPTY_CH n _INT: 对于接收通道 n , 当接收链表描述符指向的 buffer size 小于待接收数据长度时触发此中断。
- AHB_DMA_IN_DSCR_ERR_CH n _INT: 对于接收通道 n , 当接收链表描述符未通过任一描述符检查时触发此中断。
- AHB_DMA_INFIFO_OVF_CH n _INT: 当 GDMA 接收数据方向缓存溢出时触发此中断。
- AHB_DMA_INFIFO_UDF_CH n _INT: 当 GDMA 接收数据方向缓存无数据时触发此中断。

以下中断源生成 GDMA_OUT_CH n _INTR (n 为 0 ~ 2) 中断信号:

- AHB_DMA_OUT_DONE_CH n _INT: 对于发送通道 n , 当一个发送链表描述符对应的数据发送完成时触发此中断。
- AHB_DMA_OUT_EOF_CH n _INT: 对于发送通道 n , 当发送描述符的 EOF 位为 1, 且 [AHB_DMA_OUT_EOF_MODE_CH \$n\$](#) 为 0 时, 该描述符对应的最后一个数据由总线传入 GDMA TX 通道时,

该中断触发；当发送描述符的 EOF 位为 1，且 `AHB_DMA_OUT_EOF_MODE_CHn` 为 1 时，该描述符对应的最后一个数据从 GDMA TX 通道发出时，该中断触发。

- `AHB_DMA_OUT_DSCR_ERR_CHn_INT`：对于发送通道 `n`，当发送链表描述符未通过任一描述符检查时会触发此中断。
- `AHB_DMA_OUT_TOTAL_EOF_CHn_INT`：对于发送通道 `n`，当一个链表（可包含多个链表描述符）对应的所有数据都已发送完成时触发此中断。
- `AHB_DMA_OUTFIFO_OVF_CHn_INT`：当 GDMA 发送数据方向缓存溢出时触发此中断。
- `AHB_DMA_OUTFIFO_UDF_CHn_INT`：当 GDMA 发送数据方向缓存无数据时触发此中断。

3.7 编程流程

可通过配置 `PCR_GDMA_CLK_EN` 寄存器管理 GDMA 时钟门控，默认时钟门控打开。可通过配置 `PCR_GDMA_RST_EN` 对 GDMA 模块全局复位。

3.7.1 GDMA TX 通道配置流程

利用 GDMA 发送数据时，GDMA TX 通道的软件配置流程如下：

1. 对寄存器 `AHB_DMA_OUT_RST_CHn` 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针；
2. 挂载好发送链表，配置寄存器 `AHB_DMA_OUTLINK_ADDR_CHn` 指向第一个发送链表描述符；
3. 配置 `AHB_DMA_PERI_OUT_SEL_CHn` 为对应的外设号，见表 3.4-1；
4. 置位 `AHB_DMA_OUTLINK_START_CHn` 启动 GDMA TX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；
6. 等待 `AHB_DMA_OUT_TOTAL_EOF_CHn_INT` 中断，即数据传输完成。

3.7.2 GDMA RX 通道配置流程

利用 GDMA 接收数据时，GDMA RX 通道的软件配置流程如下：

1. 对寄存器 `AHB_DMA_IN_RST_CHn` 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针；
2. 挂载好接收链表，配置寄存器 `AHB_DMA_INLINK_ADDR_CHn` 指向第一个接收链表描述符；
3. 配置 `AHB_DMA_PERI_IN_SEL_CHn` 为对应的外设号，见表 3.4-1；
4. 置位 `AHB_DMA_INLINK_START_CHn` 启动 GDMA RX 通道发送数据；
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节；

3.7.3 GDMA 存储器到存储器配置流程

利用 GDMA 从存储器到存储器搬运数据时配置流程如下：

1. 对寄存器 `AHB_DMA_OUT_RST_CHn` 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针；
2. 对寄存器 `AHB_DMA_IN_RST_CHn` 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针；
3. 挂载好发送链表，配置寄存器 `AHB_DMA_OUTLINK_ADDR_CHn` 指向第一个发送链表描述符；

4. 挂载好接收链表，配置寄存器 `AHB_DMA_INLINK_ADDR_CHn` 指向第一个接收链表描述符；
5. 置位 `AHB_DMA_MEM_TRANS_EN_CHn` 使能存储器到存储器传输功能；
6. 置位 `AHB_DMA_OUTLINK_START_CHn` 启动 GDMA TX 通道发送数据；
7. 置位 `AHB_DMA_INLINK_START_CHn` 启动 GDMA RX 通道发送数据；
8. 如果某个发送链表描述符的 `suc_eof` 位配置为 1，则该描述符对应的数据段传输结束后会触发 `AHB_DMA_IN_SUC_EOF_CHn_INT` 中断。

3.7.4 通道优先级和权重配置流程

优先级仲裁配置方法如下：

1. 配置 `AHB_DMA_TX_PRI_CHn` 和 `AHB_DMA_RX_PRI_CHn` 为 TX 和 RX 通道的优先级。

权重仲裁配置方法如下：

1. 配置 `AHB_DMA_ARB_TIMEOUT` 为权重仲裁的时间片长度；
2. 配置 `AHB_DMA_TX_CH_ARB_WEIGH_CHn` 和 `AHB_DMA_RX_CH_ARB_WEIGH_CHn` 为各个通道的令牌值；
3. 清除 `AHB_DMA_TX_ARB_WEIGH_OPT_DIR_CHn`，使能 TX 和 RX 方向的权重仲裁优化；
4. 置位 `AHB_DMA_WEIGHT_EN`，使能权重仲裁。

3.8 寄存器列表

本小节的所有地址均为相对于 GDMA 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
AHB_DMA_IN_INT_RAW_CHO_REG	RX 通道 0 原始中断状态寄存器	0x0000	R/ WTC/ SS
AHB_DMA_IN_INT_ST_CHO_REG	RX 通道 0 屏蔽中断状态寄存器	0x0004	RO
AHB_DMA_IN_INT_ENA_CHO_REG	RX 通道 0 中断使能寄存器	0x0008	R/W
AHB_DMA_IN_INT_CLR_CHO_REG	RX 通道 0 中断清除寄存器	0x000C	WT
AHB_DMA_IN_INT_RAW_CH1_REG	RX 通道 1 原始中断状态寄存器	0x0010	R/ WTC/ SS
AHB_DMA_IN_INT_ST_CH1_REG	RX 通道 1 屏蔽中断状态寄存器	0x0014	RO
AHB_DMA_IN_INT_ENA_CH1_REG	RX 通道 1 中断使能寄存器	0x0018	R/W
AHB_DMA_IN_INT_CLR_CH1_REG	RX 通道 1 中断清除寄存器	0x001C	WT
AHB_DMA_IN_INT_RAW_CH2_REG	RX 通道 2 原始中断状态寄存器	0x0020	R/ WTC/ SS
AHB_DMA_IN_INT_ST_CH2_REG	RX 通道 2 屏蔽中断状态寄存器	0x0024	RO
AHB_DMA_IN_INT_ENA_CH2_REG	RX 通道 2 中断使能寄存器	0x0028	R/W
AHB_DMA_IN_INT_CLR_CH2_REG	RX 通道 2 中断清除寄存器	0x002C	WT
AHB_DMA_OUT_INT_RAW_CHO_REG	TX 通道 0 原始中断状态寄存器	0x0030	R/ WTC/ SS
AHB_DMA_OUT_INT_ST_CHO_REG	TX 通道 0 屏蔽中断状态寄存器	0x0034	RO
AHB_DMA_OUT_INT_ENA_CHO_REG	TX 通道 0 中断使能寄存器	0x0038	R/W
AHB_DMA_OUT_INT_CLR_CHO_REG	TX 通道 0 中断清除寄存器	0x003C	WT
AHB_DMA_OUT_INT_RAW_CH1_REG	TX 通道 1 原始中断状态寄存器	0x0040	R/ WTC/ SS
AHB_DMA_OUT_INT_ST_CH1_REG	TX 通道 1 屏蔽中断状态寄存器	0x0044	RO

名称	描述	地址	访问
AHB_DMA_OUT_INT_ENA_CH1_REG	TX 通道 1 中断使能寄存器	0x0048	R/W
AHB_DMA_OUT_INT_CLR_CH1_REG	TX 通道 1 中断清除寄存器	0x004C	WT
AHB_DMA_OUT_INT_RAW_CH2_REG	TX 通道 2 原始中断状态寄存器	0x0050	R/ WTC/ SS
AHB_DMA_OUT_INT_ST_CH2_REG	TX 通道 2 屏蔽中断状态寄存器	0x0054	RO
AHB_DMA_OUT_INT_ENA_CH2_REG	TX 通道 2 中断使能寄存器	0x0058	R/W
AHB_DMA_OUT_INT_CLR_CH2_REG	TX 通道 2 中断清除寄存器	0x005C	WT
调试寄存器			
AHB_DMA_AHB_TEST_REG	保留	0x0060	R/W
配置寄存器			
AHB_DMA_MISC_CONF_REG	杂项寄存器	0x0064	R/W
AHB_DMA_IN_CONFO_CHO_REG	RX 通道 0 的配置寄存器 0	0x0070	R/W
AHB_DMA_IN_CONF1_CHO_REG	RX 通道 0 的配置寄存器 1	0x0074	R/W
AHB_DMA_IN_POP_CHO_REG	RX 通道 0 的弹出控制寄存器	0x007C	varies
AHB_DMA_IN_LINK_CHO_REG	RX 通道 0 的链表描述符配置和控制寄存器	0x0080	varies
AHB_DMA_OUT_CONFO_CHO_REG	TX 通道 0 的配置寄存器 0	0x00D0	R/W
AHB_DMA_OUT_CONF1_CHO_REG	TX 通道 0 的配置寄存器 1	0x00D4	R/W
AHB_DMA_OUT_PUSH_CHO_REG	TX 通道 0 的推入控制寄存器	0x00DC	varies
AHB_DMA_OUT_LINK_CHO_REG	TX 通道 0 的链表描述符配置和控制寄存器	0x00E0	varies
AHB_DMA_IN_CONFO_CH1_REG	RX 通道 1 的配置寄存器 0	0x0130	R/W
AHB_DMA_IN_CONF1_CH1_REG	RX 通道 1 的配置寄存器 1	0x0134	R/W
AHB_DMA_IN_POP_CH1_REG	RX 通道 1 的弹出控制寄存器	0x013C	varies
AHB_DMA_IN_LINK_CH1_REG	RX 通道 1 的链表描述符配置和控制寄存器	0x0140	varies
AHB_DMA_OUT_CONFO_CH1_REG	TX 通道 1 的配置寄存器 0	0x0190	R/W
AHB_DMA_OUT_CONF1_CH1_REG	TX 通道 1 的配置寄存器 1	0x0194	R/W
AHB_DMA_OUT_PUSH_CH1_REG	TX 通道 1 的推入控制寄存器	0x019C	varies
AHB_DMA_OUT_LINK_CH1_REG	TX 通道 1 的链表描述符配置和控制寄存器	0x01A0	varies
AHB_DMA_IN_CONFO_CH2_REG	RX 通道 2 的配置寄存器 0	0x01F0	R/W
AHB_DMA_IN_CONF1_CH2_REG	RX 通道 2 的配置寄存器 1	0x01F4	R/W

名称	描述	地址	访问
AHB_DMA_IN_POP_CH2_REG	RX 通道 2 的弹出控制寄存器	0x01FC	varies
AHB_DMA_IN_LINK_CH2_REG	RX 通道 2 的链表描述符配置和控制寄存器	0x0200	varies
AHB_DMA_OUT_CONFO_CH2_REG	TX 通道 2 的配置寄存器 0	0x0250	R/W
AHB_DMA_OUT_CONF1_CH2_REG	TX 通道 2 的配置寄存器 1	0x0254	R/W
AHB_DMA_OUT_PUSH_CH2_REG	TX 通道 2 的推入控制寄存器	0x025C	varies
AHB_DMA_OUT_LINK_CH2_REG	TX 通道 2 的链表描述符配置和控制寄存器	0x0260	varies
AHB_DMA_TX_CH_ARB_WEIGHT_CHO_REG	TX 通道 0 仲裁权重配置寄存器	0x02DC	R/W
AHB_DMA_TX_ARB_WEIGHT_OPT_DIR_CHO_REG	TX 通道 0 权重仲裁优化使能寄存器	0x02E0	R/W
AHB_DMA_TX_CH_ARB_WEIGHT_CH1_REG	TX 通道 1 仲裁权重配置寄存器	0x0304	R/W
AHB_DMA_TX_ARB_WEIGHT_OPT_DIR_CH1_REG	TX 通道 1 权重仲裁优化使能寄存器	0x0308	R/W
AHB_DMA_TX_CH_ARB_WEIGHT_CH2_REG	TX 通道 2 仲裁权重配置寄存器	0x032C	R/W
AHB_DMA_TX_ARB_WEIGHT_OPT_DIR_CH2_REG	TX 通道 2 权重仲裁优化使能寄存器	0x0330	R/W
AHB_DMA_RX_CH_ARB_WEIGHT_CHO_REG	RX 通道 0 仲裁权重配置寄存器	0x0354	R/W
AHB_DMA_RX_ARB_WEIGHT_OPT_DIR_CHO_REG	RX 通道 0 权重仲裁优化使能寄存器	0x0358	R/W
AHB_DMA_RX_CH_ARB_WEIGHT_CH1_REG	RX 通道 1 仲裁权重配置寄存器	0x037C	R/W
AHB_DMA_RX_ARB_WEIGHT_OPT_DIR_CH1_REG	RX 通道 1 权重仲裁优化使能寄存器	0x0380	R/W
AHB_DMA_RX_CH_ARB_WEIGHT_CH2_REG	RX 通道 2 仲裁权重配置寄存器	0x03A4	R/W
AHB_DMA_RX_ARB_WEIGHT_OPT_DIR_CH2_REG	RX 通道 2 权重仲裁优化使能寄存器	0x03A8	R/W
AHB_DMA_IN_LINK_ADDR_CHO_REG	RX 通道 0 链表描述符配置寄存器	0x03AC	R/W
AHB_DMA_IN_LINK_ADDR_CH1_REG	RX 通道 1 链表描述符配置寄存器	0x03B0	R/W
AHB_DMA_IN_LINK_ADDR_CH2_REG	RX 通道 2 链表描述符配置寄存器	0x03B4	R/W
AHB_DMA_OUT_LINK_ADDR_CHO_REG	TX 通道 0 链表描述符配置寄存器	0x03B8	R/W
AHB_DMA_OUT_LINK_ADDR_CH1_REG	TX 通道 1 链表描述符配置寄存器	0x03BC	R/W
AHB_DMA_OUT_LINK_ADDR_CH2_REG	TX 通道 2 链表描述符配置寄存器	0x03C0	R/W
AHB_DMA_INTR_MEM_START_ADDR_REG	可访问地址空间起始地址配置寄存器	0x03C4	R/W
AHB_DMA_INTR_MEM_END_ADDR_REG	可访问地址空间结束地址配置寄存器	0x03C8	R/W
AHB_DMA_ARB_TIMEOUT_REG	权重仲裁超时配置寄存器	0x03DC	R/W
AHB_DMA_WEIGHT_EN_REG	权重仲裁使能寄存器	0x0400	R/W
版本寄存器			

名称	描述	地址	访问
AHB_DMA_DATE_REG	版本控制寄存器	0x0068	R/W
状态寄存器			
AHB_DMA_INFIFO_STATUS_CHO_REG	RX 通道 0 FIFO 状态	0x0078	RO
AHB_DMA_IN_STATE_CHO_REG	RX 通道 0 状态	0x0084	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CHO_REG	RX 通道 0 传输结束 (EOF) 时的接收链表描述符地址	0x0088	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CHO_REG	RX 通道 0 上发生错误时的接收链表描述符地址	0x008C	RO
AHB_DMA_IN_DSCR_CHO_REG	RX 通道 0 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0090	RO
AHB_DMA_IN_DSCR_BFO_CHO_REG	RX 通道 0 当前已预读取的接收链表描述符地址	0x0094	RO
AHB_DMA_IN_DSCR_BF1_CHO_REG	RX 通道 0 上一个已预读取的接收链表描述符地址	0x0098	RO
AHB_DMA_OUTFIFO_STATUS_CHO_REG	TX 通道 0 FIFO 状态	0x00D8	RO
AHB_DMA_OUT_STATE_CHO_REG	TX 通道 0 状态	0x00E4	RO
AHB_DMA_OUT_EOF_DES_ADDR_CHO_REG	TX 通道 0 传输结束 (EOF) 时的发送链表描述符地址	0x00E8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CHO_REG	TX 通道 0 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x00EC	RO
AHB_DMA_OUT_DSCR_CHO_REG	TX 通道 0 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x00F0	RO
AHB_DMA_OUT_DSCR_BFO_CHO_REG	TX 通道 0 当前已预读取的发送链表描述符地址	0x00F4	RO
AHB_DMA_OUT_DSCR_BF1_CHO_REG	TX 通道 0 上一个已预读取的发送链表描述符地址	0x00F8	RO
AHB_DMA_INFIFO_STATUS_CH1_REG	RX 通道 1 FIFO 状态	0x0138	RO
AHB_DMA_IN_STATE_CH1_REG	RX 通道 1 状态	0x0144	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH1_REG	RX 通道 1 传输结束 (EOF) 时的接收链表描述符地址	0x0148	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH1_REG	RX 通道 1 上发生错误时的接收链表描述符地址	0x014C	RO
AHB_DMA_IN_DSCR_CH1_REG	RX 通道 1 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0150	RO
AHB_DMA_IN_DSCR_BFO_CH1_REG	RX 通道 1 当前已预读取的接收链表描述符地址	0x0154	RO
AHB_DMA_IN_DSCR_BF1_CH1_REG	RX 通道 1 上一个已预读取的接收链表描述符地址	0x0158	RO
AHB_DMA_OUTFIFO_STATUS_CH1_REG	TX 通道 1 FIFO 状态	0x0198	RO
AHB_DMA_OUT_STATE_CH1_REG	TX 通道 1 状态	0x01A4	RO

名称	描述	地址	访问
AHB_DMA_OUT_EOF_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的发送链表描述符地址	0x01A8	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	TX 通道 1 传输结束 (EOF) 时的最后一个发送链表描述符地址	0x01AC	RO
AHB_DMA_OUT_DSCR_CH1_REG	TX 通道 1 已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x01B0	RO
AHB_DMA_OUT_DSCR_BFO_CH1_REG	TX 通道 1 当前已预读取的发送链表描述符地址	0x01B4	RO
AHB_DMA_OUT_DSCR_BF1_CH1_REG	TX 通道 1 上一个已预读取的发送链表描述符地址	0x01B8	RO
AHB_DMA_INFIFO_STATUS_CH2_REG	RX 通道 2 FIFO 状态	0x01F8	RO
AHB_DMA_IN_STATE_CH2_REG	RX 通道 2 状态	0x0204	RO
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH2_REG	RX 通道 2 传输结束 (EOF) 时接收链表描述符地址	0x0208	RO
AHB_DMA_IN_ERR_EOF_DES_ADDR_CH2_REG	当 RX 通道 2 上发生错误时接收链表描述符地址	0x020C	RO
AHB_DMA_IN_DSCR_CH2_REG	当前预读取接收链表描述符指向的下一个接收链表描述符的地址在 RX 通道 2 上	0x0210	RO
AHB_DMA_IN_DSCR_BFO_CH2_REG	当前预读取接收链表描述符的地址在 RX 通道 2 上	0x0214	RO
AHB_DMA_IN_DSCR_BF1_CH2_REG	上一个已预读取的取接收链表描述符的地址在 RX 通道 2 上	0x0218	RO
AHB_DMA_OUTFIFO_STATUS_CH2_REG	TX 通道 2 FIFO 状态	0x0258	RO
AHB_DMA_OUT_STATE_CH2_REG	TX 通道 2 的状态	0x0264	RO
AHB_DMA_OUT_EOF_DES_ADDR_CH2_REG	当 TX 通道 2 传输结束 (EOF) 时发送链表描述符地址	0x0268	RO
AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	TX 通道 2 传输结束 (EOF) 时最后一个发送链表描述符地址	0x026C	RO
AHB_DMA_OUT_DSCR_CH2_REG	TX 通道 2 当前预读取发送链表描述符指向的下一个发送链表描述符的地址	0x0270	RO
AHB_DMA_OUT_DSCR_BFO_CH2_REG	TX 通道 2 当前预读取发送链表描述符的地址	0x0274	RO
AHB_DMA_OUT_DSCR_BF1_CH2_REG	TX 通道 2 上一个已预读取的取发送链表描述符的地址在	0x0278	RO
优先级寄存器			
AHB_DMA_IN_PRI_CHO_REG	RX 通道 0 的优先级寄存器	0x009C	R/W
AHB_DMA_OUT_PRI_CHO_REG	TX 通道 0 的优先级寄存器	0x00FC	R/W

名称	描述	地址	访问
AHB_DMA_IN_PRI_CH1_REG	RX 通道 1 的优先级寄存器	0x015C	R/W
AHB_DMA_OUT_PRI_CH1_REG	TX 通道 1 的优先级寄存器	0x01BC	R/W
AHB_DMA_IN_PRI_CH2_REG	RX 通道 2 的优先级寄存器	0x021C	R/W
AHB_DMA_OUT_PRI_CH2_REG	TX 通道 2 的优先级寄存器	0x027C	R/W
外设选择寄存器			
AHB_DMA_IN_PERI_SEL_CHO_REG	RX 通道 0 的外设选择寄存器	0x00A0	R/W
AHB_DMA_OUT_PERI_SEL_CHO_REG	TX 通道 0 的外设选择寄存器	0x0100	R/W
AHB_DMA_IN_PERI_SEL_CH1_REG	RX 通道 1 的外设选择寄存器	0x0160	R/W
AHB_DMA_OUT_PERI_SEL_CH1_REG	TX 通道 1 的外设选择寄存器	0x01C0	R/W
AHB_DMA_IN_PERI_SEL_CH2_REG	RX 通道 2 的外设选择寄存器	0x0220	R/W
AHB_DMA_OUT_PERI_SEL_CH2_REG	TX 通道 2 的外设选择寄存器	0x0280	R/W

3.9 寄存器

本小节的所有地址均为相对于 GDMA 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 3.1. AHB_DMA_IN_INT_RAW_CH n _REG (n : 0-2) (0x0000+0x10* n)

(reserved)																								AHB_DMA_IN_FIFO_UDF_CH _n _INT_RAW AHB_DMA_IN_FIFO_OVF_CH _n _INT_RAW AHB_DMA_IN_DSCR_EMPTY_CH _n _INT_RAW AHB_DMA_IN_DSCR_ERR_CH _n _INT_RAW AHB_DMA_IN_SUC_EOF_CH _n _INT_RAW AHB_DMA_IN_ERR_EOF_CH _n _INT_RAW AHB_DMA_IN_DONE_CH _n _INT_RAW																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																								7	6	5	4	3	2	1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

AHB_DMA_IN_DONE_CH n _INT_RAW AHB_DMA_IN_DONE_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_SUC_EOF_CH n _INT_RAW AHB_DMA_IN_SUC_EOF_CH n _INT 的原始中断状态。对于 UHCI，接收链表描述符指向的最后一个字节数据接收成功、且接收通道 n 没有检测到数据错误时，该位变为 1。(R/WTC/SS)

AHB_DMA_IN_ERR_EOF_CH n _INT_RAW AHB_DMA_IN_ERR_EOF_CH n _INT 的原始中断状态。仅对 UHCI 有效。(R/WTC/SS)

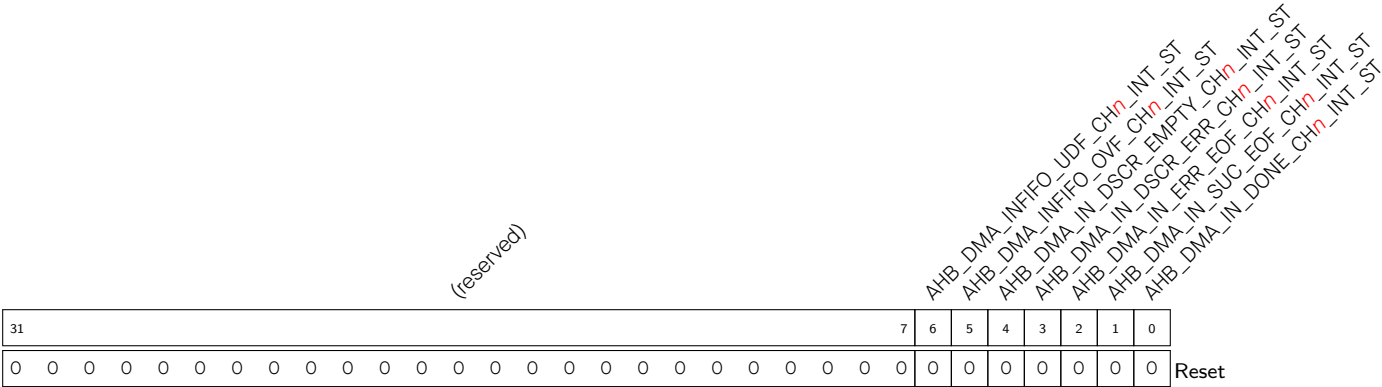
AHB_DMA_IN_DSCR_ERR_CH n _INT_RAW AHB_DMA_IN_DSCR_ERR_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_DSCR_EMPTY_CH n _INT_RAW AHB_DMA_IN_DSCR_EMPTY_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_FIFO_OVF_CH n _INT_RAW AHB_DMA_IN_FIFO_OVF_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_IN_FIFO_UDF_CH n _INT_RAW AHB_DMA_IN_FIFO_UDF_CH n _INT 的原始中断状态。(R/WTC/SS)

Register 3.2. AHB_DMA_IN_INT_ST_CHn_REG (n: 0-2) (0x0004+0x10*n)



- AHB_DMA_IN_DONE_CHn_INT_ST AHB_DMA_IN_DONE_CHn_INT 的屏蔽中断状态。(RO)
- AHB_DMA_IN_SUC_EOF_CHn_INT_ST AHB_DMA_IN_SUC_EOF_CHn_INT 的屏蔽中断状态。(RO)
- AHB_DMA_IN_ERR_EOF_CHn_INT_ST AHB_DMA_IN_ERR_EOF_CHn_INT 的屏蔽中断状态。(RO)
- AHB_DMA_IN_DSCR_ERR_CHn_INT_ST AHB_DMA_IN_DSCR_ERR_CHn_INT 的屏蔽中断状态。
(RO)
- AHB_DMA_IN_DSCR_EMPTY_CHn_INT_ST AHB_DMA_IN_DSCR_EMPTY_CHn_INT 的屏蔽中断
状态。(RO)
- AHB_DMA_IN_FIFO_OVF_CHn_INT_ST AHB_DMA_IN_FIFO_OVF_CHn_INT 的屏蔽中断状态。(RO)
- AHB_DMA_IN_FIFO_UDF_CHn_INT_ST AHB_DMA_IN_FIFO_UDF_CHn_INT 的屏蔽中断状态。(RO)

Register 3.3. AHB_DMA_IN_INT_ENA_CH n _REG (n : 0-2) (0x0008+0x10* n)

Diagram of the 32-bit Reset register. The register is divided into two main sections. The upper 24 bits are labeled "(reserved)" and are currently set to 0. The lower 8 bits are labeled "Reset" and are currently set to 0. The 8 bits of the Reset section are: bit 7 (0), bit 6 (0), bit 5 (0), bit 4 (0), bit 3 (0), bit 2 (0), bit 1 (0), and bit 0 (0). The labels for the Reset section are: AHB_DMA_IN_DONE_CHn_INT_ENA, AHB_DMA_IN_SUC_EOF_CHn_INT_ENA, AHB_DMA_IN_ERR_EOF_CHn_INT_ENA, AHB_DMA_IN_DSCR_ERR_CHn_INT_ENA, AHB_DMA_IN_DSCR_EMPTY_CHn_INT_ENA, AHB_DMA_IN_OVF_CHn_INT_ENA, AHB_DMA_INFIFO_UDF_CHn_INT_ENA, and AHB_DMA_INFIFO_UDF_CHn_INT_ENA.

AHB_DMA_IN_DONE_CH_n_INT_ENA 写1使能AHB_DMA_IN_DONE_CH_n_INT。(R/W)

AHB_DMA_IN_SUC_EOF_CH_n_INT_ENA 写1使能AHB_DMA_IN_SUC_EOF_CH_n_INT。(R/W)

AHB_DMA_IN_ERR_EOF_CH_n_INT_ENA 写1使能 AHB_DMA_IN_ERR_EOF_CH_n_INT。(R/W)

AHB_DMA_IN_DSCR_ERR_CH n _INT_ENA 写 1 使能 AHB_DMA_IN_DSCR_ERR_CH n _INT。(R/W)

AHB_DMA_IN_DSCR_EMPTY_CH n _INT_ENA 写 1 使能 AHB_DMA_IN_DSCR_EMPTY_CH n _INT。
(R/W)

AHB_DMA_INFIFO_OVF_CH_n_INT_ENA 写1使能AHB_DMA_INFIFO_OVF_CH_n_INT。(R/W)

AHB_DMA_INFIFO_UDF_CH n _INT_ENA 写 1 使能 AHB_DMA_INFIFO_UDF_CH n _INT。(R/W)

Register 3.4. AHB_DMA_IN_INT_CLR_CH_{*n*}_REG (*n*: 0-2) (0x000C+0x10n*)**

Diagram of the 32-bit Reset register. The register is divided into two main sections. The upper 24 bits are labeled "(reserved)". The lower 8 bits are individual control bits, each with a label above it. The labels are: AHB_DMA_INFIFO_UDF_CHn_INT_CLR, AHB_DMA_INFIFO_OVF_CHn_INT_CLR, AHB_DMA_IN_DSCR_EMPTY_CHn_INT_CLR, AHB_DMA_IN_DSCR_ERR_EOF_CHn_INT_CLR, AHB_DMA_IN_SUC_EOF_CHn_INT_CLR, AHB_DMA_IN_DONE_CHn_INT_CLR, AHB_DMA_IN_DSCR_ERR_EOF_CHn_INT_CLR, and AHB_DMA_IN_DSCR_ERR_EOF_CHn_INT_CLR. The bits are numbered 31 down to 0 from left to right. The bit values are: 31: 0, 30: 0, 29: 0, 28: 0, 27: 0, 26: 0, 25: 0, 24: 0, 23: 0, 22: 0, 21: 0, 20: 0, 19: 0, 18: 0, 17: 0, 16: 0, 15: 0, 14: 0, 13: 0, 12: 0, 11: 0, 10: 0, 9: 0, 8: 0, 7: 0, 6: 0, 5: 0, 4: 0, 3: 0, 2: 0, 1: 0, 0: 0.

AHB_DMA_IN_DONE_CH_n_INT_CLR 写1清除AHB_DMA_IN_DONE_CH_n_INT。(WT)

AHB_DMA_IN_SUC_EOF_CH_n_INT_CLR 写1清除 AHB_DMA_IN_SUC_EOF_CH_n_INT。(WT)

AHB_DMA_IN_ERR_EOF_CH_n INT CLR 写1清除 AHB_DMA_IN_ERR_EOF_CH_n INT。(WT)

AHB_DMA_IN_DSCR_ERR_CH_n_INT_CLR 写1清除 AHB_DMA_IN_DSCR_ERR_CH_n_INT。(WT)

AHB_DMA_IN_DSCR_EMPTY_CH n _INT_CLR 写 1 清除 AHB_DMA_IN_DSCR_EMPTY_CH n _INT。
(WT)

AHB_DMA_INFIFO_OVF_CH n _INT_CLR 写1清除AHB_DMA_INFIFO_OVF_CH n _INT。(WT)

AHB_DMA_INFIFO_UDF_CH n _INT_CLR 写1清除 AHB_DMA_INFIFO_UDF_CH n _INT。(WT)

Register 3.5. AHB_DMA_OUT_INT_RAW_CH n _REG (n : 0-2) (0x0030+0x10* n)

Register 0x00000000: AHB_DMA_OUT_DONE

31	6	5	4	3	2	1	0
(reserved)		Reset	0	0	0	0	0

AHB_DMA_OUT_DONE_CHn_INT_RAW
 AHB_DMA_OUT_DONE_CHn_INT_RAW
 AHB_DMA_OUT_DONE_CHn_INT_RAW
 AHB_DMA_OUT_DONE_CHn_INT_RAW
 AHB_DMA_OUT_DONE_CHn_INT_RAW

AHB_DMA_OUT_DONE_CH n _INT_RAW AHB_DMA_OUT_DONE_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_EOF_CH n _INT_RAW AHB_DMA_OUT_EOF_CH n _INT 的原始中断状态。(R/WTC/SS)

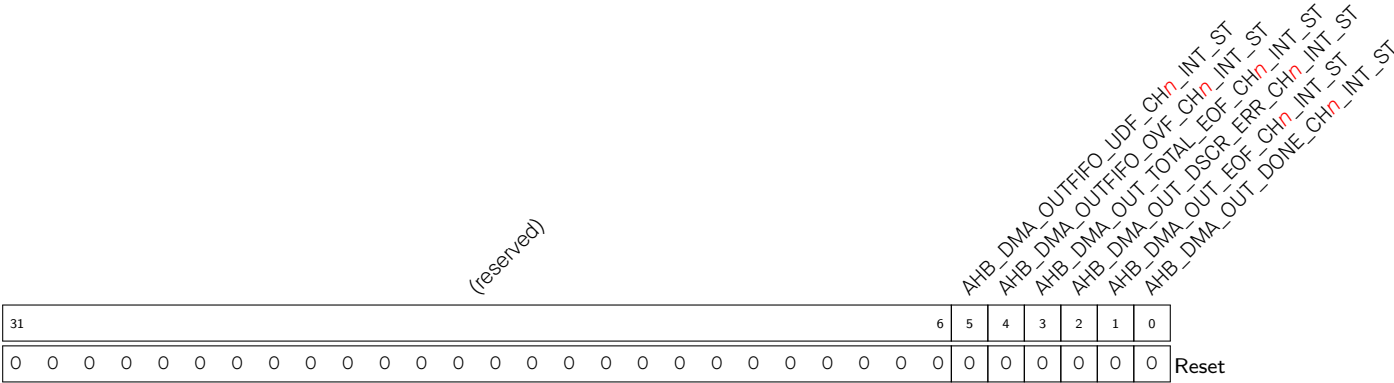
AHB_DMA_OUT_DSCR_ERR_CH n _INT_RAW AHB_DMA_OUT_DSCR_ERR_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUT_TOTAL_EOF_CH n _INT_RAW AHB_DMA_OUT_TOTAL_EOF_CH n _INT 的原始中断状态。(R/WTC/SS)

AHB_DMA_OUTFIFO_OVF_CH n _INT_RAW AHB_DMA_OUTFIFO_OVF_CH n _INT 的原始中断状态。
(R/WTC/SS)

AHB_DMA_OUTFIFO_UDF_CH n _INT_RAW AHB_DMA_OUTFIFO_UDF_CH n _INT 的原始中断状态。
(R/WTC/SS)

Register 3.6. AHB_DMA_OUT_INT_ST_CHn_REG (n: 0-2) (0x0034+0x10*n)



AHB_DMA_OUT_DONE_CHn_INT_ST AHB_DMA_OUT_DONE_CHn_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_EOF_CHn_INT_ST AHB_DMA_OUT_EOF_CHn_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_DSCR_ERR_CHn_INT_ST AHB_DMA_OUT_DSCR_ERR_CHn_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUT_TOTAL_EOF_CHn_INT_ST AHB_DMA_OUT_TOTAL_EOF_CHn_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUTFIFO_OVF_CHn_INT_ST AHB_DMA_OUTFIFO_OVF_CHn_INT 的屏蔽中断状态。(RO)

AHB_DMA_OUTFIFO_UDF_CHn_INT_ST AHB_DMA_OUTFIFO_UDF_CHn_INT 的屏蔽中断状态。(RO)

Register 3.7. AHB_DMA_OUT_INT_ENA_CHn_REG (n: 0-2) (0x0038+0x10*n)

(reserved)																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT_INT_ENA						
																								AHB_DMA_OUT						

Register 3.8. AHB_DMA_OUT_INT_CLR_CH n _REG (n : 0-2) (0x003C+0x10* n)

AHB_DMA_OUT_DONE_CH_n_INT_CLR 写1清除 AHB_DMA_OUT_DONE_CH_n_INT。(WT)

AHB DMA OUT EOF CH_n INT CLR 写1清除 AHB DMA OUT EOF CH_n INT。(WT)

AHB_DMA_OUT_DSCR_ERR_CH n _INT_CLR 写 1 清除 AHB_DMA_OUT_DSCR_ERR_CH n _INT。
(WT)

AHB_DMA_OUT_TOTAL_EOF_CH n _INT_CLR 写 1 清除 AHB_DMA_OUT_TOTAL_EOF_CH n _INT。
(WT)

AHB_DMA_OUTFIFO_OVF_CH_n_INT_CLR 写1清除 AHB_DMA_OUTFIFO_OVF_CH_n_INT。(WT)

AHB_DMA_OUTFIFO_UDF_CH_n_INT_CLR 写1清除 AHB_DMA_OUTFIFO_UDF_CH_n_INT。(WT)

Register 3.9. AHB_DMA_AHB_TEST_REG (0x0060)

[illegible]

AHB_DMA_AHB_TESTMODE 保留。(R/W)

AHB_DMA_AHB_TESTADDR 保留。(R/W)

Register 3.10. AHB_DMA_MISC_CONF_REG (0x0064)

[illegible]

AHB_DMA_AHB_M_RST_INTER 先写1再写0复位内部AHB FSM。(R/W)

AHB_DMA_ARB_PRI_DIS 配置关闭优先级仲裁。

0: 开肩

1: 关闭

(R/W)

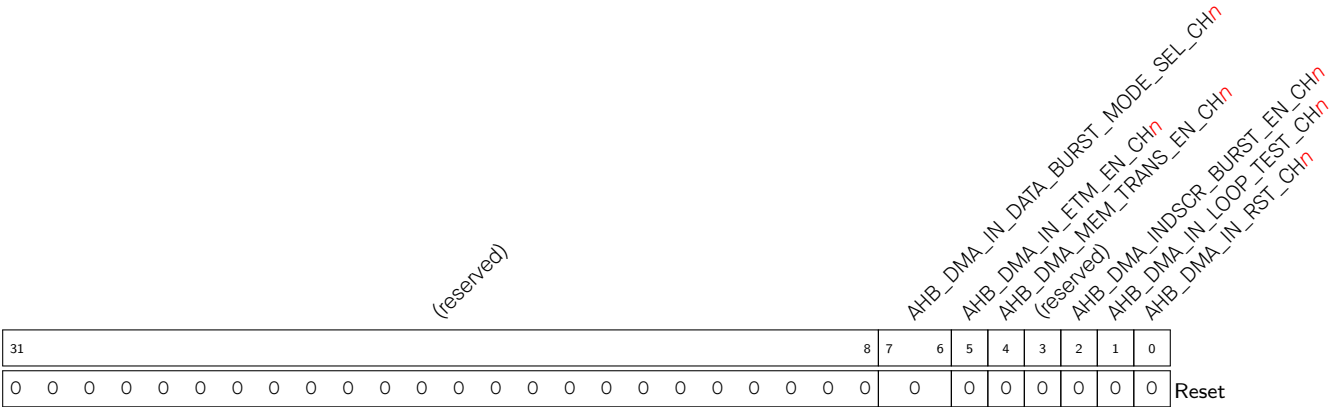
AHB_DMA_CLK_EN 配置 AHB DMA 时钟门控。

0: 仅在应用写寄存器时开启时钟

1: 一直强制为寄存器开启时钟

(R/W)

Register 3.11. AHB_DMA_IN_CONFO_CH n _REG (n : 0-2) (0x0070+0xC0* n)



AHB_DMA_IN_RST_CH n 先写 1 再写 0 复位接收通道 n 的 FSM 和 RX FIFO 指针。(R/W)

AHB_DMA_IN_LOOP_TEST_CH n 配置接收链表回写时 owner 位的值。(R/W)

AHB_DMA_INDSR_BURST_EN_CH n 配置开启 INCR 突发传输让接收通道 n 读取描述符。

0: 关闭

1: 开启

(R/W)

AHB_DMA_MEM_TRANS_EN_CH n 配置开启存储器到存储器数据传输。

0: 关闭

1: 开启

(R/W)

AHB_DMA_IN_ETM_EN_CH n 配置开启接收通道 n 的 ETM 控制。

0: 关闭

1: 开启

(R/W)

AHB_DMA_IN_DATA_BURST_MODE_SEL_CH n 配置 RX 通道 n 最大突发长度。

0: SINGLE

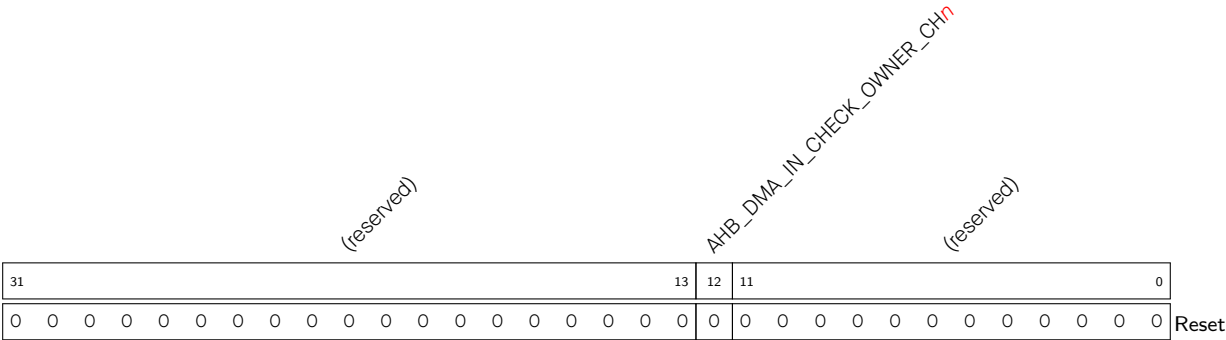
1: INCR4

2: INCR8

3: 保留

(R/W)

Register 3.12. AHB_DMA_IN_CONF1_CH n _REG (n : 0-2) (0x0074+0xC0* n)

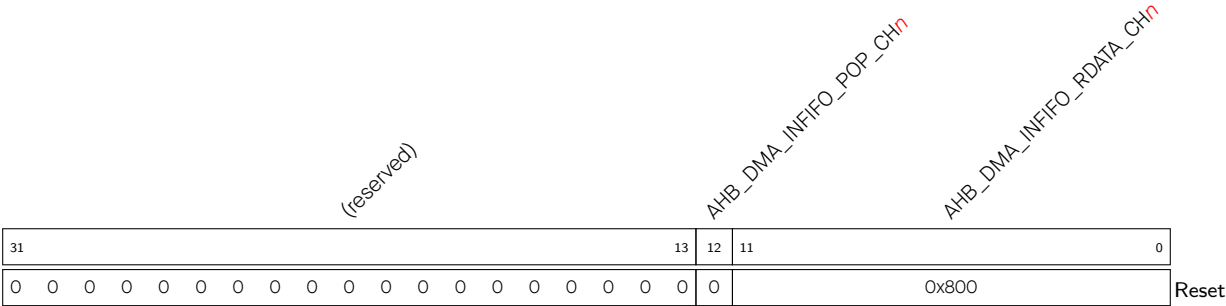


AHB_DMA_IN_CHECK_OWNER_CH n 配置开启接收通道 n 的 owner 位检查。

- 0: 关闭
- 1: 开启

(R/W)

Register 3.13. AHB_DMA_IN_POP_CH n _REG (n : 0-2) (0x007C+0xC0* n)



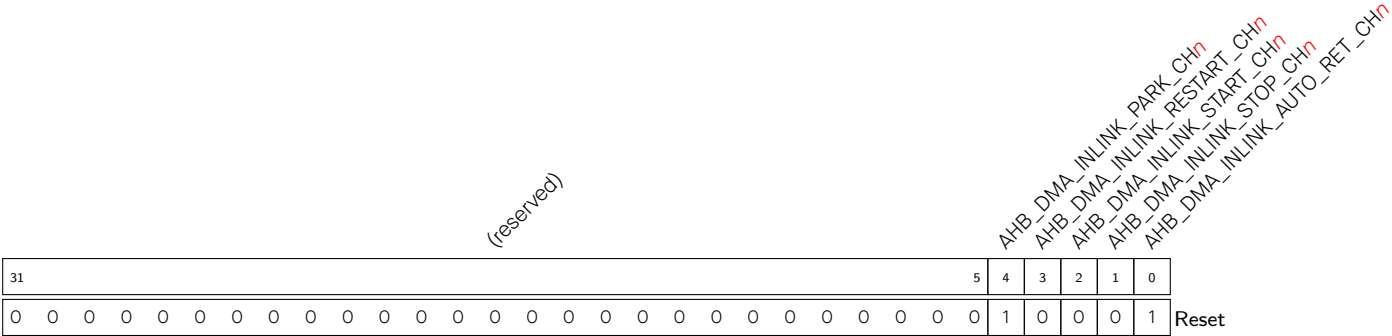
AHB_DMA_INFIFO_RDATA_CH n 表示从 AHB DMA RX FIFO 弹出的数据。(RO)

AHB_DMA_INFIFO_POP_CH n 配置从 AHB DMA RX FIFO 中弹出数据。

- 0: 无效值。没有作用
- 1: 弹出

(WT)

Register 3.14. AHB_DMA_IN_LINK_CHn_REG (n: 0-2) (0x0080+0xC0*n)



AHB_DMA_INLINK_AUTO_RET_CHn 配置在接收数据有错时返回到当前接收链表描述符的地址。

- 0: 不返回
- 1: 返回
- (R/W)

AHB_DMA_INLINK_STOP_CHn 配置接收通道 n 停止接收数据。

- 0: 无效值。没有作用
- 1: 停止
- (WT)

AHB_DMA_INLINK_START_CHn 配置开启接收通道 n 进行数据传输。

- 0: 关闭
- 1: 开启
- (WT)

AHB_DMA_INLINK_RESTART_CHn 配置重启接收通道 n 进行 AHB DMA 传输。

- 0: 无效值。没有作用
- 1: 重启
- (WT)

AHB_DMA_INLINK_PARK_CHn 表示接收链表描述符 FSM 的状态。

- 0: 运行
- 1: 空闲
- (RO)

Register 3.15. AHB_DMA_OUT_CONFO_CH n _REG (n : 0-2) (0x00D0+0xC0*(n -1))

(reserved)																						AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH ⁿ																			
(reserved)																						(reserved)																			
(reserved)																						AHB_DMA_OUT_ETM_EN_CH ⁿ																			
(reserved)																						AHB_DMA_OUTDSCR_BURST_EN_CH ⁿ																			
(reserved)																						AHB_DMA_OUT_EOF_MODE_CH ⁿ																			
(reserved)																						AHB_DMA_OUT_AUTO_WRRBACK_CH ⁿ																			
(reserved)																						AHB_DMA_OUT_LOOP_TEST_CH ⁿ																			
(reserved)																						AHB_DMA_OUT_RST_CH ⁿ																			
31																						10										9	8	7	6	5	4	3	2	1	0
0																						0										0	0	0	0	0	1	0	0	0	Reset

Reset

AHB_DMA_OUT_RST_CH n 配置发送通道 n FSM 和 TX FIFO 指针的复位状态。

0: 复位释放

1: 复位

(R/W)

AHB_DMA_OUT_LOOP_TEST_CH n 配置发送链表回写时 owner 位的值。(R/W)

AHB_DMA_OUT_AUTO_WRRBACK_CH n 配置在发送完 TX FIFO 中的所有数据时是否开启发送链表自动回写。

0: 关闭

1: 开启

(R/W)

AHB_DMA_OUT_EOF_MODE_CH n 配置生成 EOF 标志的时间。

0: 待发送数据推入 AHB DMA FIFO 时生成发送通道 n 的 EOF 标志。

1: 待发送数据从 AHB DMA FIFO 中弹出时生成发送通道 n 的 EOF 标志。

(R/W)

AHB_DMA_OUTDSCR_BURST_EN_CH n 配置开启 INCR 突发传输，让发送通道 n 读取描述符。

0: 关闭

1: 开启

(R/W)

AHB_DMA_OUT_ETM_EN_CH n 配置开启发送通道 n 的 ETM 控制。

0: 关闭

1: 开启

(R/W)

AHB_DMA_OUT_DATA_BURST_MODE_SEL_CH n 配置 TX 通道 n 最大突发长度。

0: SINGLE

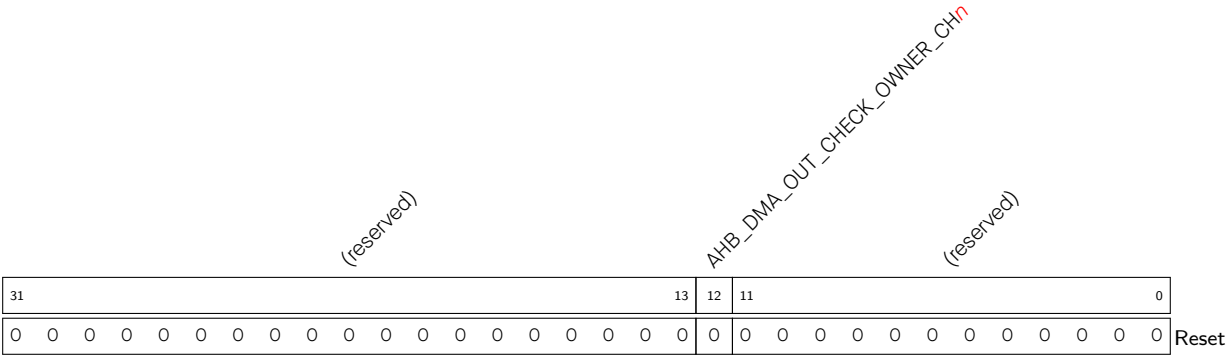
1: INCR4

2: INCR8

3: 保留

(R/W)

Register 3.16. AHB_DMA_OUT_CONF1_CH n _REG (n : 0-2) (0x00D4+0xC0* n)



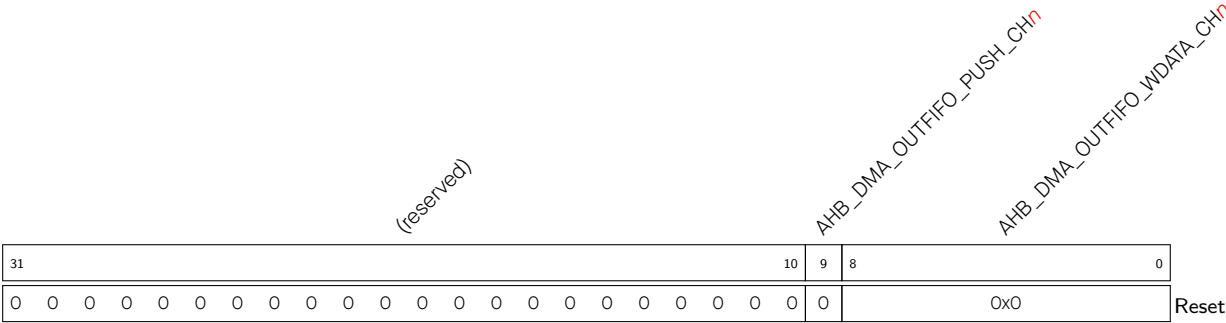
AHB_DMA_OUT_CHECK_OWNER_CH n 配置开启发送通道 n 的 owner 位检查。

0: 关闭

1: 开启

(R/W)

Register 3.17. AHB_DMA_OUT_PUSH_CH n _REG (n : 0-2) (0x00DC+0xC0* n)



AHB_DMA_OUTFIFO_WDATA_CH n 表示需要推入 AHB DMA TX FIFO 中的数据。(R/W)

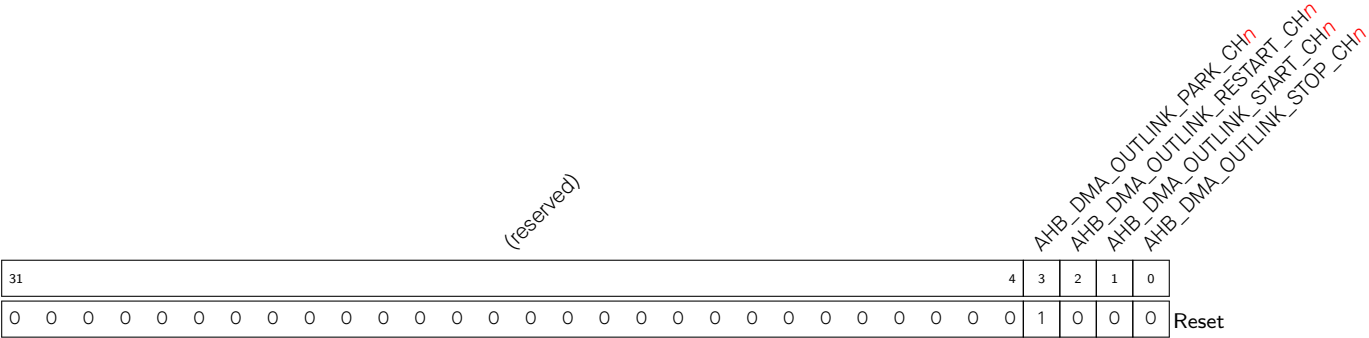
AHB_DMA_OUTFIFO_PUSH_CH n 配置将数据推入 AHB DMA TX FIFO。

0: 无效值。没有作用

1: 推入

(WT)

Register 3.18. AHB_DMA_OUT_LINK_CH n _REG (n : 0-2) (0x00E0+0xC0* n)



AHB_DMA_OUTLINK_STOP_CH n 配置发送通道 n 停止发送数据。

0: 无效值。没有作用

1: 停止

(WT)

AHB_DMA_OUTLINK_START_CH n 配置开启发送通道 n 进行数据传输。

0: 关闭

1: 开启

(WT)

AHB_DMA_OUTLINK_RESTART_CH n 配置重启发送通道 n 进行 AHB DMA 传输。

0: 无效值。没有作用

1: 重启

(WT)

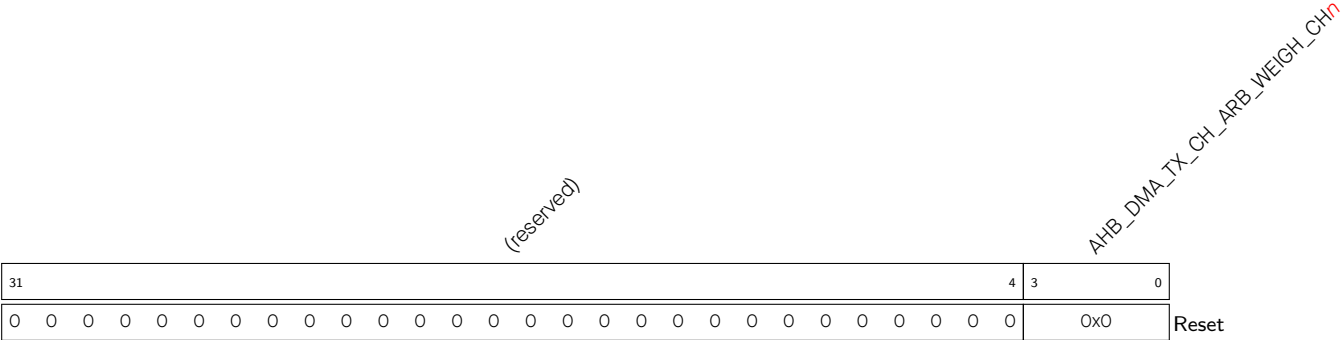
AHB_DMA_OUTLINK_PARK_CH n 表示发送链表描述符 FSM 的状态。

0: 运行

1: 空闲

(RO)

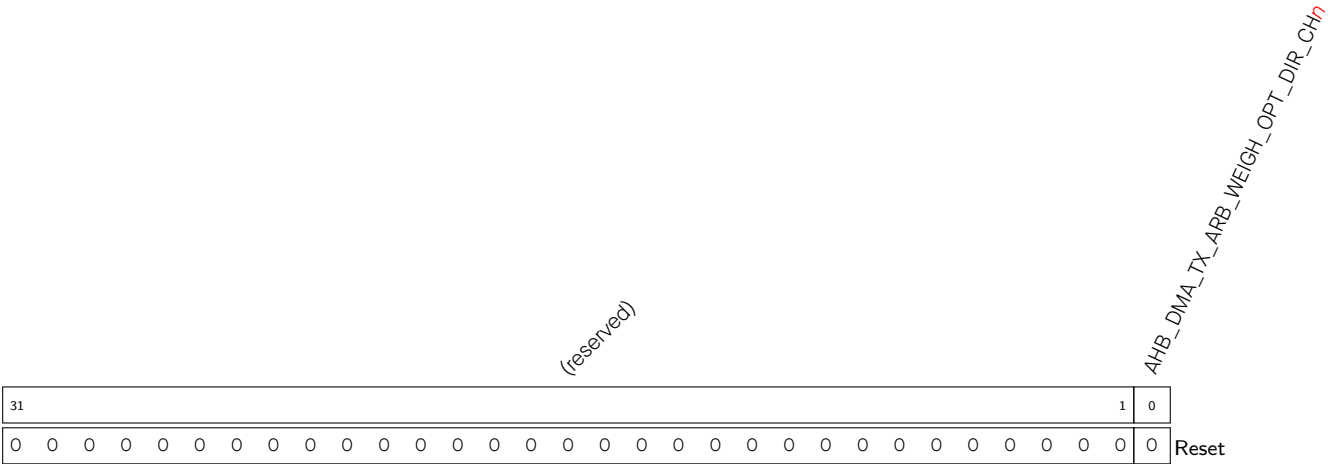
Register 3.19. AHB_DMA_TX_CH_ARB_WEIGHT_CH n _REG (n : 0-2) (0x02DC+0x28* n)



AHB_DMA_TX_CH_ARB_WEIGHT_CH n 配置 TX 通道 n 的权重（即令牌数）。

取值范围：0 ~ 15。(R/W)

Register 3.20. AHB_DMA_TX_ARB_WEIGHT_OPT_DIR_CH n _REG (n : 0-2) (0x02E0+0x28* n)



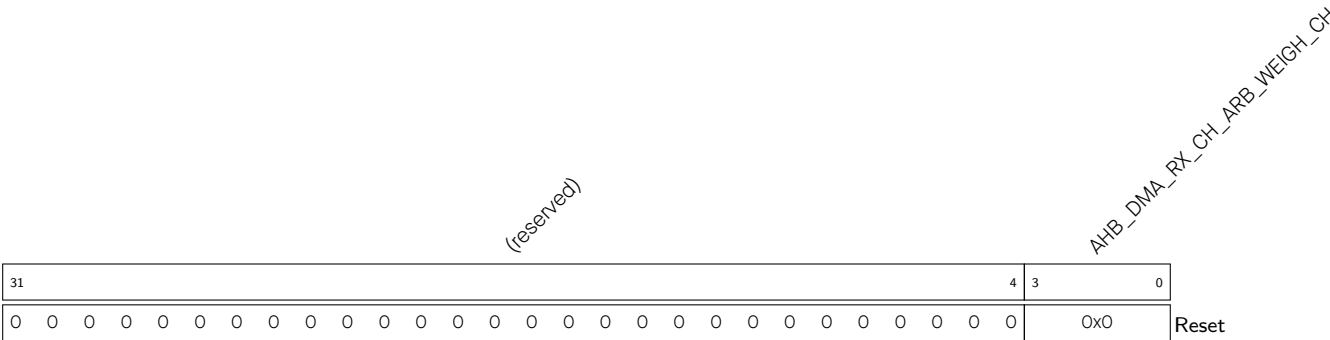
AHB_DMA_TX_ARB_WEIGHT_OPT_DIR_CH n 配置发送通道 n 是否开启权重仲裁优化。

0: 关闭

1: 开启

(R/W)

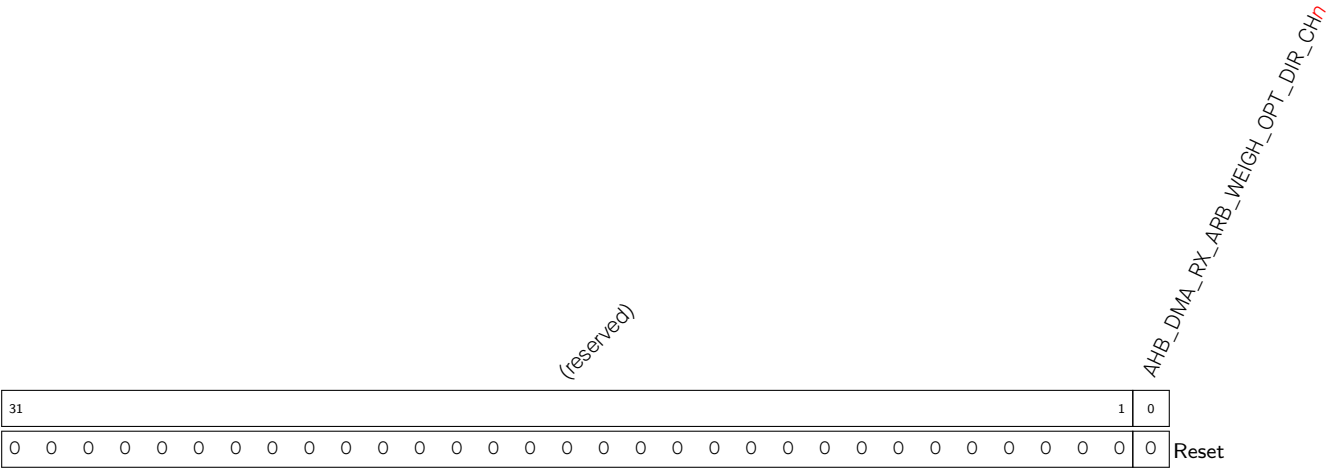
Register 3.21. AHB_DMA_RX_CH_ARB_WEIGHT_CH n _REG (n : 0-2) (0x0354+0x28* n)



AHB_DMA_RX_CH_ARB_WEIGHT_CH n 配置 RX 通道 n 的权重（即令牌数）。

取值范围：0 ~ 15。（R/W）

Register 3.22. AHB_DMA_RX_ARB_WEIGHT_OPT_DIR_CH n _REG (n : 0-2) (0x0358+0x28* n)

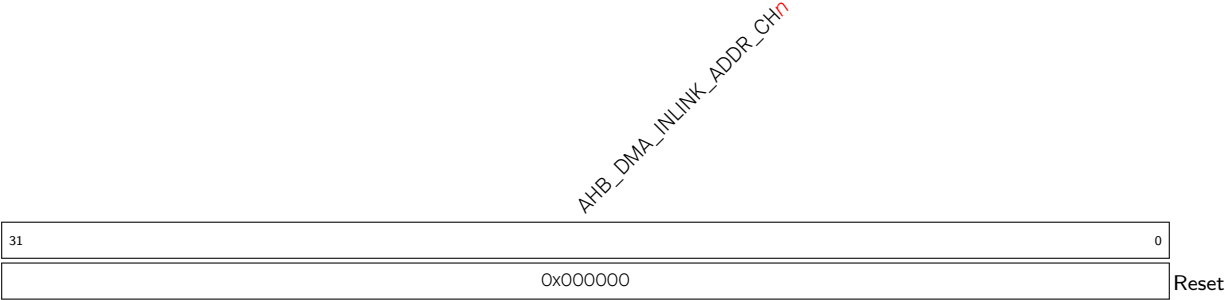


AHB_DMA_RX_ARB_WEIGHT_OPT_DIR_CH n 配置接收通道 n 是否开启权重仲裁优化。

- 0: 关闭
- 1: 开启

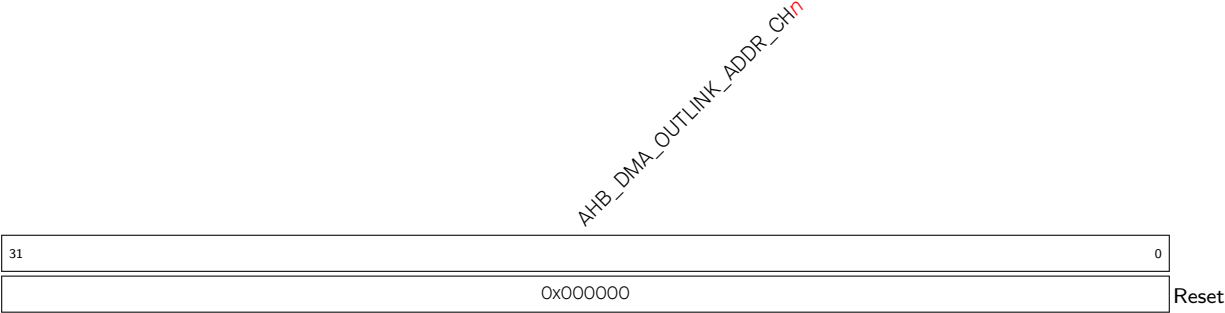
(R/W)

Register 3.23. AHB_DMA_IN_LINK_ADDR_CH n _REG (n : 0-2) (0x03AC+0x4* n)



AHB_DMA_INLINK_ADDR_CH n 表示第一个接收链表描述符的地址。(R/W)

Register 3.24. AHB_DMA_OUT_LINK_ADDR_CH n _REG (n : 0-2) (0x03B8+0x4* n)



AHB_DMA_OUTLINK_ADDR_CH n 表示第一个发送链表描述符的地址。(R/W)

Register 3.25. AHB_DMA_INTR_MEM_START_ADDR_REG (0x03C4)

AHB_DMA_ACCESS_INTR_MEM_START_ADDR	
31	0
0x000000	
Reset	

AHB_DMA_ACCESS_INTR_MEM_START_ADDR 配置可访问地址空间的起始地址。(注意：只可配置内部存储地址空间，不可配置片外地址范围，可能会发生意外问题)。(R/W)

Register 3.26. AHB_DMA_INTR_MEM_END_ADDR_REG (0x03C8)

AHB_DMA_ACCESS_INTR_MEM_END_ADDR	
31	0
0xffffffff	
Reset	

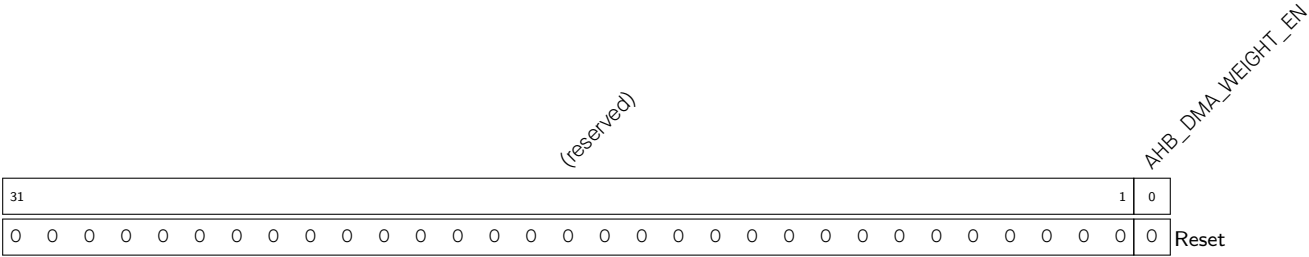
AHB_DMA_ACCESS_INTR_MEM_END_ADDR 配置可访问地址空间的结束地址。(R/W)

Register 3.27. AHB_DMA_ARB_TIMEOUT_REG (0x03DC)

(reserved)																AHB_DMA_ARB_TIMEOUT																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x00																Reset															

AHB_DMA_ARB_TIMEOUT 配置权重仲裁时间片。单位：AHB 总线时钟周期。(R/W)

Register 3.28. AHB_DMA_WEIGHT_EN_REG (0x0400)



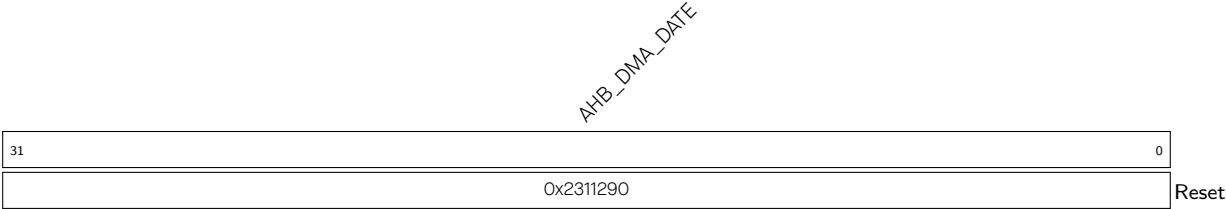
AHB_DMA_WEIGHT_EN 配置开启权重仲裁。

0: 关闭

1: 开启

(R/W)

Register 3.29. AHB_DMA_DATE_REG (0x0068)



AHB_DMA_DATE 版本控制寄存器。(R/W)

Register 3.30. AHB_DMA_INFIFO_STATUS_CHn_REG (n: 0-2) (0x0078+0xC0*n)

(reserved)				AHB_DMA_IN_BUF_HUNGRY_CHn				AHB_DMA_IN_REMAIN_UNDER_4B_CHn				AHB_DMA_IN_REMAIN_UNDER_3B_CHn				AHB_DMA_IN_REMAIN_UNDER_2B_CHn				AHB_DMA_IN_REMAIN_UNDER_1B_CHn				(reserved)				AHB_DMA_INFIFO_CNT_CHn				(reserved)				AHB_DMA_INFIFO_EMPTY_CHn				AHB_DMA_INFIFO_FULL_CHn			
31	28	27	26	25	24	23	22	15	14	8	7	2	1	0	Reset																												
0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1																

AHB_DMA_INFIFO_FULL_CHn 表示 L1 RX FIFO 是否为满。

0: 未
1: 已
(RO)

AHB_DMA_INFIFO_EMPTY_CHn 表示 L1 RX FIFO 是否为空。

0: 未
1: 已
(RO)

AHB_DMA_INFIFO_CNT_CHn 表示接收通道 n L1 RX FIFO 中的字节数。(RO)

AHB_DMA_IN_REMAIN_UNDER_1B_CHn 保留。(RO)

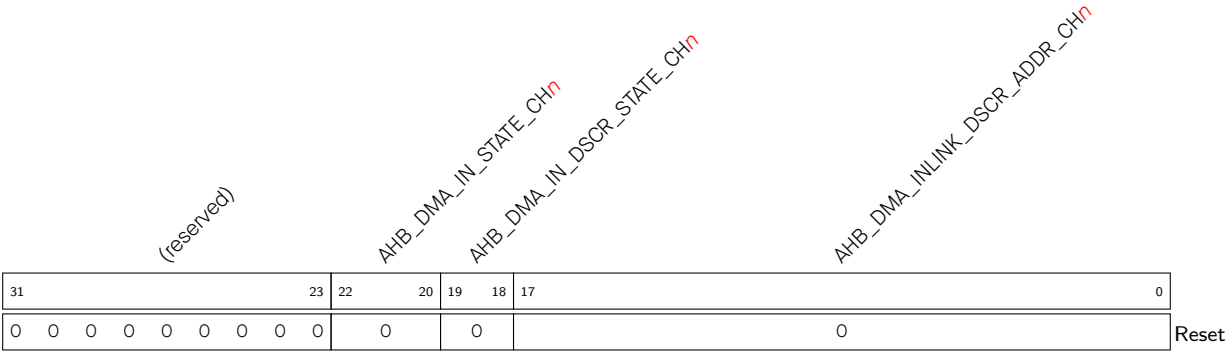
AHB_DMA_IN_REMAIN_UNDER_2B_CHn 保留。(RO)

AHB_DMA_IN_REMAIN_UNDER_3B_CHn 保留。(RO)

AHB_DMA_IN_REMAIN_UNDER_4B_CHn 保留。(RO)

AHB_DMA_IN_BUF_HUNGRY_CHn 保留。(RO)

Register 3.31. AHB_DMA_IN_STATE_CH n _REG (n : 0-2) (0x0084+0xC0* n)



AHB_DMA_INLINK_DSCR_ADDR_CH n 表示下一个预读取（但未处理）的接收链表描述符所在地址的低 18 位。如果当前处理的接收链表描述符是最后一个描述符，则该字段表示当前处理的接收链表描述符地址。(RO)

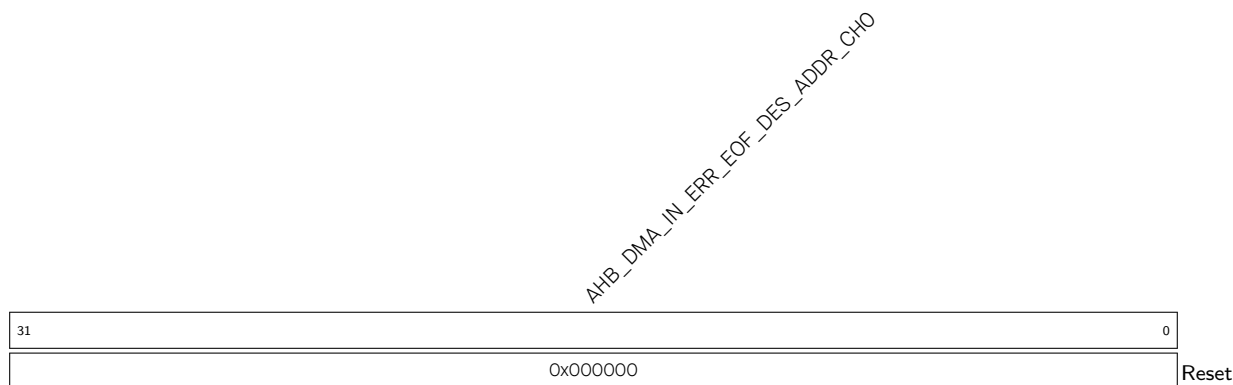
AHB_DMA_IN_DSCR_STATE_CH n 保留。(RO)

AHB_DMA_IN_STATE_CH n 保留。(RO)

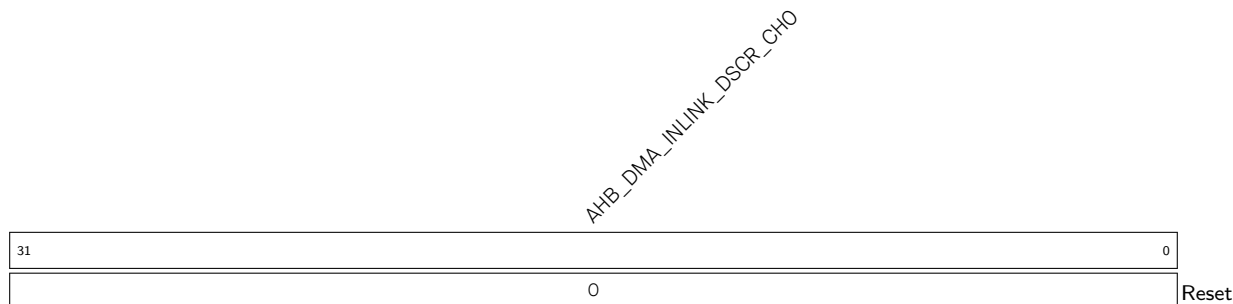
Register 3.32. AHB_DMA_IN_SUC_EOF_DES_ADDR_CH n _REG (n : 0-2) (0x0088+0xC0* n)



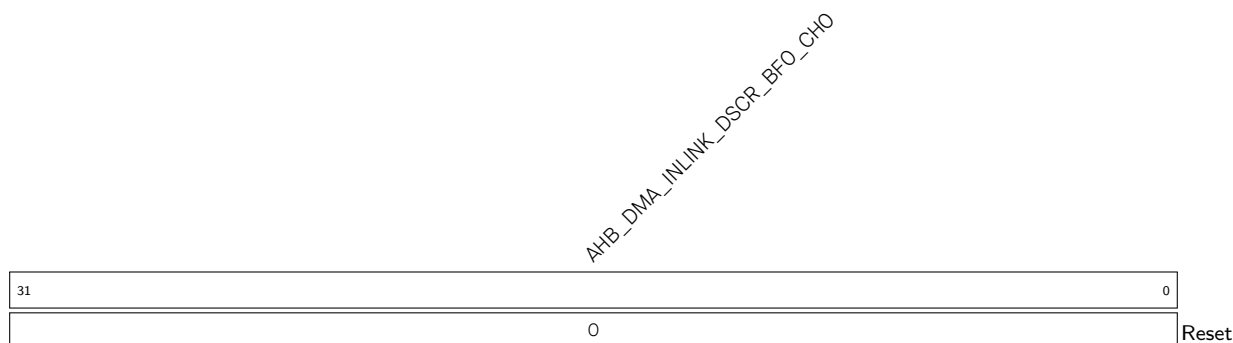
AHB_DMA_IN_SUC_EOF_DES_ADDR_CH n 表示 EOF 位为 1 的接收链表描述符的地址。(RO)

Register 3.33. AHB_DMA_IN_ERR_EOF_DES_ADDR_CH n _REG (n : 0-2) (0x008C+0xC0* n)

AHB_DMA_IN_ERR_EOF_DES_ADDR_CH n 表示当前接收数据有错时,相应接收链表描述符的位置。
仅对 UHCI 有效。(RO)

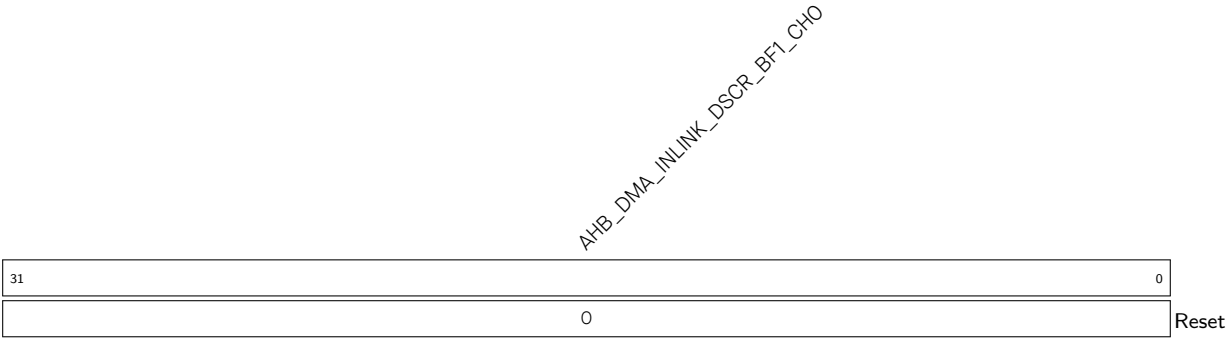
Register 3.34. AHB_DMA_IN_DSCR_CH n _REG (n : 0-2) (0x0090+0xC0* n)

AHB_DMA_INLINK_DSCR_CH n 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址 x+1。(RO)

Register 3.35. AHB_DMA_IN_DSCR_BFO_CH n _REG (n : 0-2) (0x0094+0xC0* n)

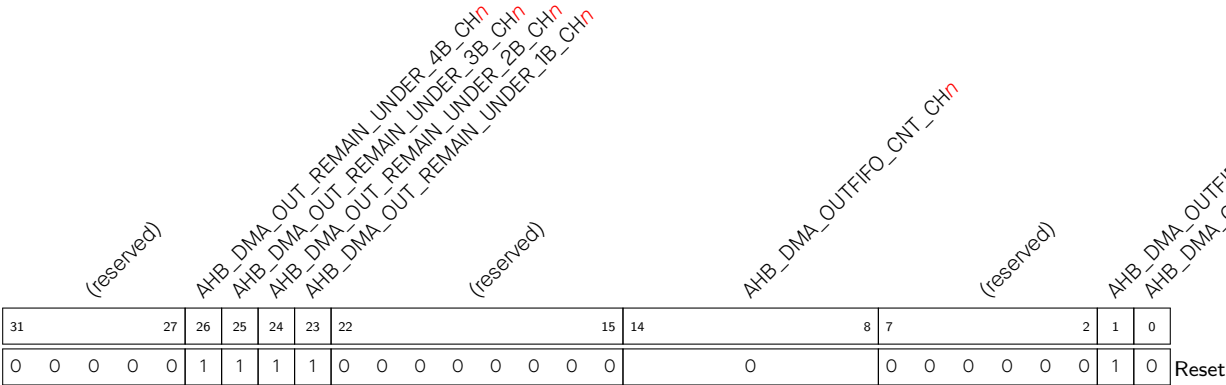
AHB_DMA_INLINK_DSCR_BFO_CH n 表示当前已预读取的接收链表描述符所在地址 x。(RO)

Register 3.36. AHB_DMA_IN_DSCR_BF1_CH n _REG (n : 0-2) (0x0098+0xC0* n)



AHB_DMA_INLINK_DSCR_BF1_CH n 表示前一个已预读取的接收链表描述符所在地址 x-1。(RO)

Register 3.37. AHB_DMA_OUTFIFO_STATUS_CH n _REG (n : 0-2) (0x00D8+0xC0* n)



AHB_DMA_OUTFIFO_FULL_CH n 表示 L1 TX FIFO 是否为满。

0: 未满
1: 已满
(RO)

AHB_DMA_OUTFIFO_EMPTY_CH n 表示 L1 TX FIFO 是否为空。

0: 未空
1: 已空
(RO)

AHB_DMA_OUTFIFO_CNT_CH n 表示发送通道 n L1 TX FIFO 中的字节数。(RO)

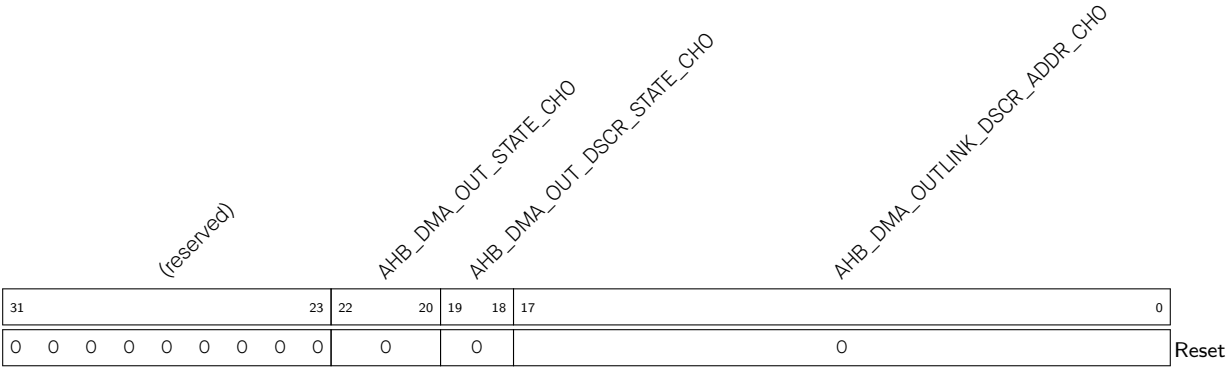
AHB_DMA_OUT_REMAIN_UNDER_1B_CH n 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_2B_CH n 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_3B_CH n 保留。(RO)

AHB_DMA_OUT_REMAIN_UNDER_4B_CH n 保留。(RO)

Register 3.38. AHB_DMA_OUT_STATE_CHn_REG (n: 0-2) (0x00E4+0xC0*n)

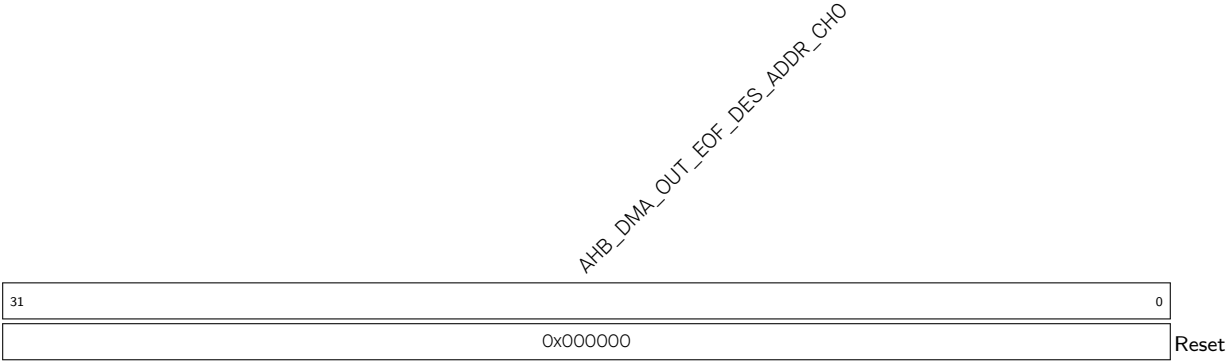


AHB_DMA_OUTLINK_DSCR_ADDR_CHn 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

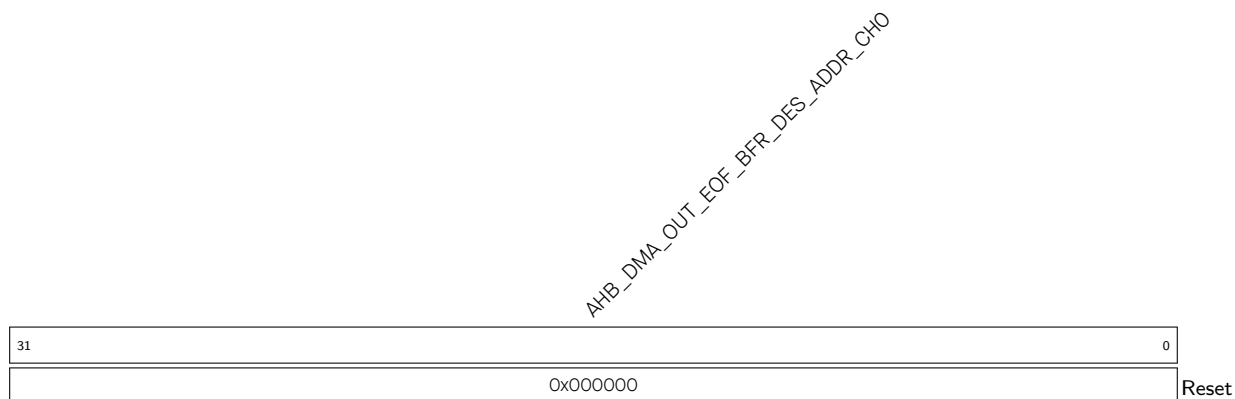
AHB_DMA_OUT_DSCR_STATE_CHn 保留。(RO)

AHB_DMA_OUT_STATE_CHn 保留。(RO)

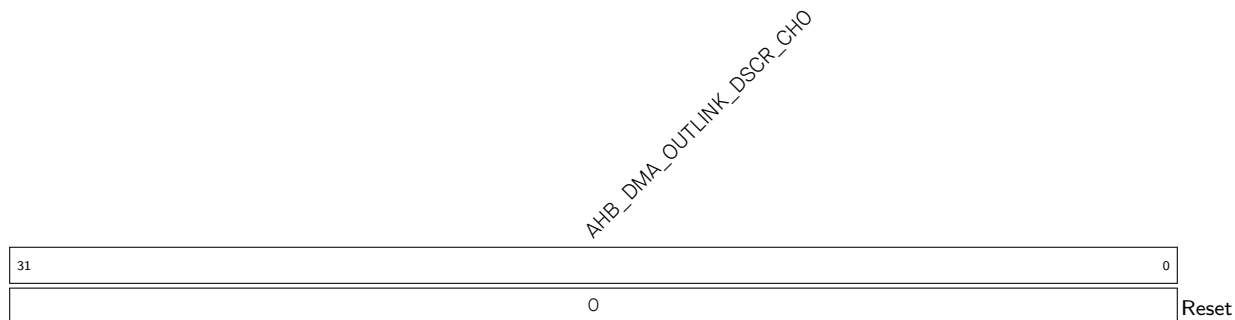
Register 3.39. AHB_DMA_OUT_EOF_DES_ADDR_CHn_REG (n: 0-2) (0x00E8+0xC0*n)



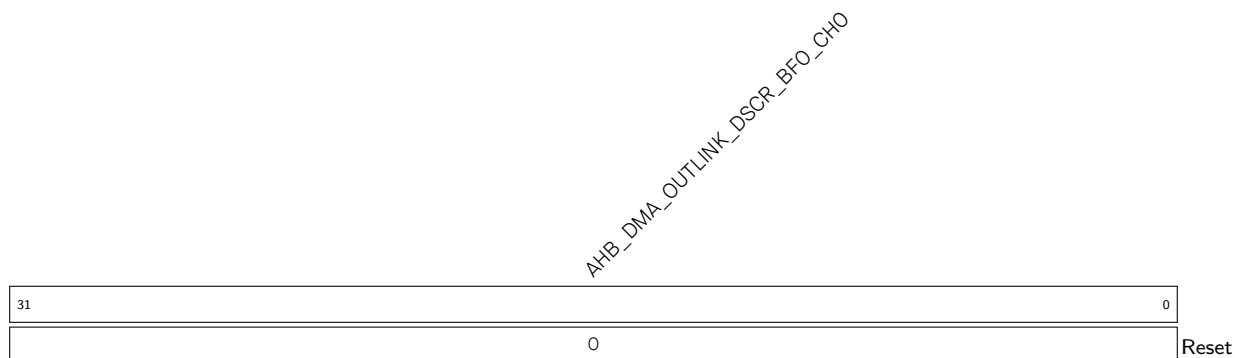
AHB_DMA_OUT_EOF_DES_ADDR_CHn 表示 EOF 位为 1 的发送链表描述符的地址。(RO)

Register 3.40. AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH n _REG (n : 0-2) (0x00EC+0xC0* n)

AHB_DMA_OUT_EOF_BFR_DES_ADDR_CH n 表示倒数第二个发送链表描述符的地址。(RO)

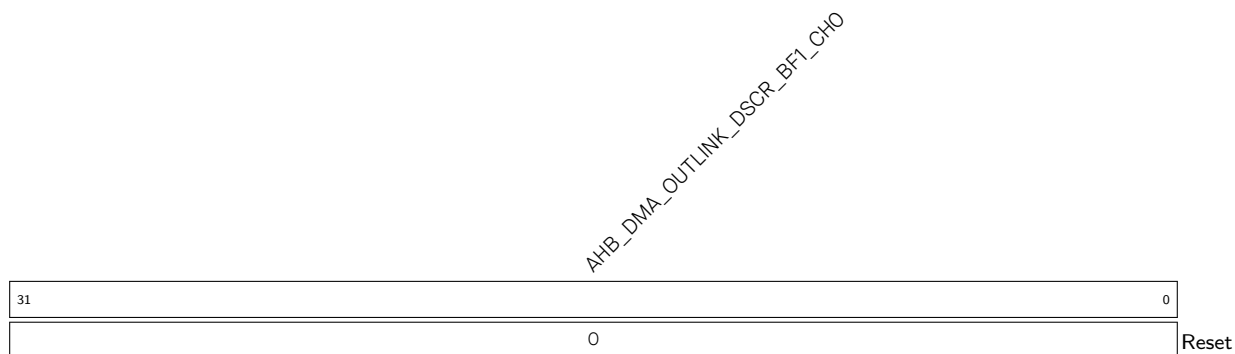
Register 3.41. AHB_DMA_OUT_DSCR_CH n _REG (n : 0-2) (0x00F0+0xC0* n)

AHB_DMA_OUTLINK_DSCR_CH n 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址 y+1。(RO)

Register 3.42. AHB_DMA_OUT_DSCR_BFO_CH n _REG (n : 0-2) (0x00F4+0xC0* n)

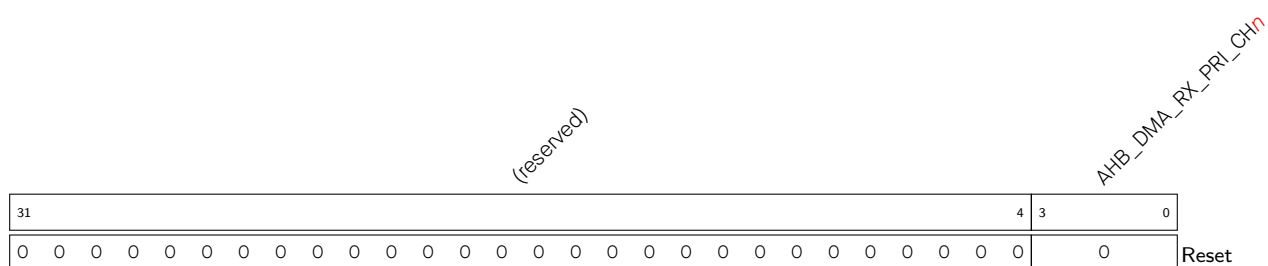
AHB_DMA_OUTLINK_DSCR_BFO_CH n 表示当前已预读取的发送链表描述符所在地址 y。(RO)

Register 3.43. AHB_DMA_OUT_DSCR_BF1_CH n _REG (n : 0-2) (0x00F8+0xC0* n)



AHB_DMA_OUTLINK_DSCR_BF1_CH_n 表示前一个已预读取的发送链表描述符所在地址 y-1。(RO)

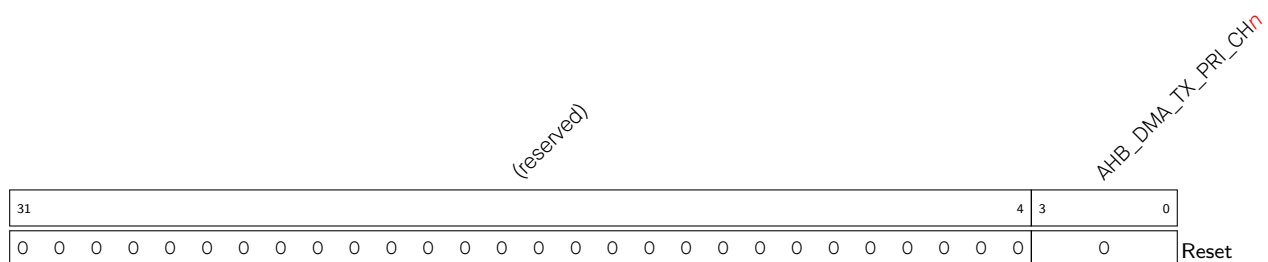
Register 3.44. AHB_DMA_IN_PRI_CH_{*n*}_REG (*n*: 0-2) (0x009C+0xC0n*)**



AHB_DMA_RX_PRI_CH n 配置接收通道 n 的优先级。值越大，优先级越高。

取值范围：0 ~ 5 (R/W)

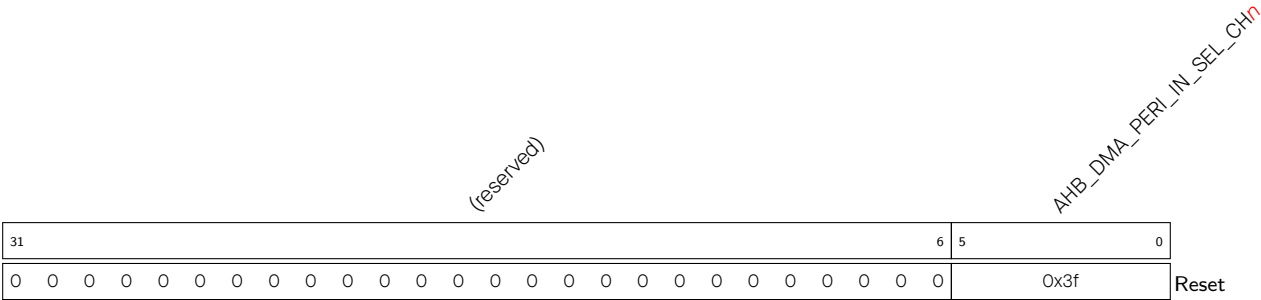
Register 3.45. AHB_DMA_OUT_PRI_CH n _REG (n : 0-2) (0x00FC+0xC0* n)



AHB_DMA_TX_PRI_CH n 配置发送通道 n 的优先级。值越大，优先级越高。

取值范围：0 ~ 5 (R/W)

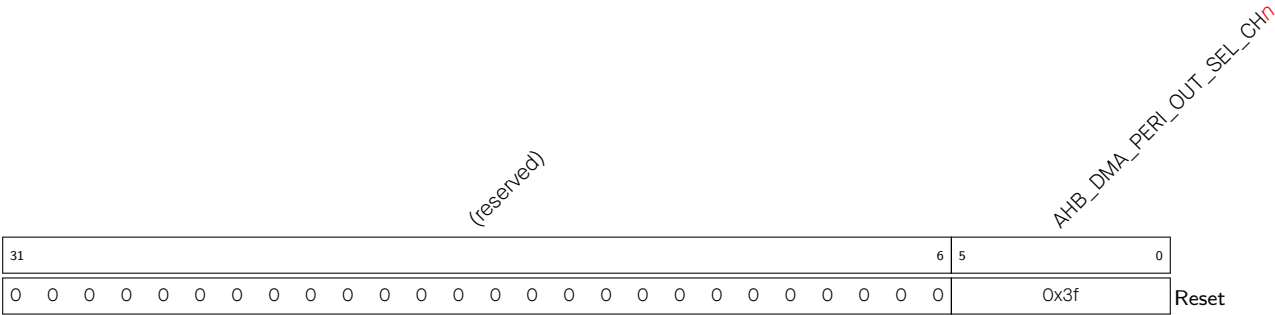
Register 3.46. AHB_DMA_IN_PERI_SEL_CHn_REG (n: 0-2) (0x00A0+0xC0*n)



AHB_DMA_PERI_IN_SEL_CHn 配置连接接收通道 n 的外设。

- 0: Dummy
 - 1: GP-SPI
 - 2: UHCI
 - 3: I2S
 - 4: Dummy
 - 5: Dummy
 - 6: AES
 - 7: SHA
 - 8: ADC
 - 9: PARLIO
 - 10: Dummy
 - 11 ~ 15: Dummy
 - 16 ~ 63: 无效值
- (R/W)

Register 3.47. AHB_DMA_OUT_PERI_SEL_CH n _REG (n : 0-2) (0x0100+0xC0* n)



AHB_DMA_OUT_PERI_SEL_CH n 配置连接发送通道 n 的外设。

- 0: Dummy
 - 1: GP-SPI
 - 2: UHCI
 - 3: I2S
 - 4: Dummy
 - 5: Dummy
 - 6: AES
 - 7: SHA
 - 8: ADC
 - 9: PARLIO
 - 10: Dummy
 - 11 ~ 15: Dummy
 - 16 ~ 63: 无效值
- (R/W)

第 II 卷

存储器组织结构

该部分详细描述系统的存储器组织结构，例如 RAM、ROM、eFuse 和外部存储器的组织和映射，为理解与存储器相关的子系统提供了框架。

第 4 章

系统和存储器

4.1 概述

ESP32-C5 是一个超低功耗和高度集成的系统，集成了：

- 一颗高性能 RISC-V 32 位处理器 (HP CPU)，五级流水线架构，主频高达 240 MHz
- 一颗低功耗 RISC-V 32 位处理器 (LP CPU)，二级流水线架构，主频高达 40 MHz 或 48 MHz（见下方说明）

说明：

ESP32-C5 支持晶振频率选择，详情请参考章节 6 复位和时钟。

所有的内部存储器、外部存储器以及外设都分布在 HP CPU 和 LP CPU 的总线上。

4.2 主要特性

- **地址空间**
 - 720 KB 内部存储器空间，可被指令总线或数据总线访问
 - 832 KB 外设地址空间
 - 32 MB 外部存储器虚拟地址空间，可被指令总线或数据总线访问
 - 384 KB 内部 DMA 地址空间
 - 32 MB 外部 DMA 地址空间
- **内部存储器**
 - 320 KB ROM
 - 384 KB HP SRAM
 - 16 KB LP SRAM
- **外部存储器**
 - 最大支持 32 MB 片外 flash
 - 最大支持 32 MB 片外 RAM
- **外设空间**
 - 总计 61 个模块/外设
- **GDMA**

- 7 个模块/外设支持 GDMA

4.3 功能描述

4.3.1 地址映射

图 4.3-1 展示了系统结构与地址映射结构。所有非保留地址均可通过指令总线 and 数据总线进行访问，即指令总线 and 数据总线访问的地址空间一样。HP CPU 和 LP CPU 的数据总线与指令总线均为小端序。所有总线均为 32 位数据宽度。HP CPU 和 LP CPU 均可以通过数据总线进行单字节、双字节、四字节的访问。

HP CPU 支持：

- 直接通过数据总线和指令总线访问内部存储器。
 - 384 KB HP SRAM (0x4080_0000 ~ 0x4085_FFFF)
 - 16 KB LP SRAM (0x5000_0000 ~ 0x5000_3FFF)
- 通过 ROM-Cache 访问内部存储器。
 - 320 KB ROM (0x4000_0000 ~ 0x4004_FFFF)

说明：

软件可以以 cache-able 或 noncache-able 的方式访问 ROM 的地址。其中 cache-able 访问方式是指通过 cache 进行访问；noncache-able 访问方式是不通过 cache 进行直接访问。cache-able 访问还是 noncache-able 访问则是由 CPU PMA (Physical Memory Attribution) 控制。

- 通过 cache 访问外部存储器，虚地址经过 MMU 映射到外部存储器的实地址空间。
 - 多达 32 MB 外部 flash (0x4200_0000 ~ 0x43FF_FFFF)
 - 多达 32 MB 外部 RAM (0x4200_0000 ~ 0x43FF_FFFF)
- 直接通过数据总线访问模块/外设。
 - HP CPU 外设
 - HP 外设
 - LP 外设

LP CPU 支持：

- 直接通过数据总线和指令总线访问 HP SRAM 和 LP SRAM。
 - 384 KB HP SRAM (0x4080_0000 ~ 0x4085_FFFF)
 - 16 KB LP SRAM (0x5000_0000 ~ 0x5000_3FFF)
- 直接通过数据总线访问模块/外设。
 - HP 外设
 - LP 外设

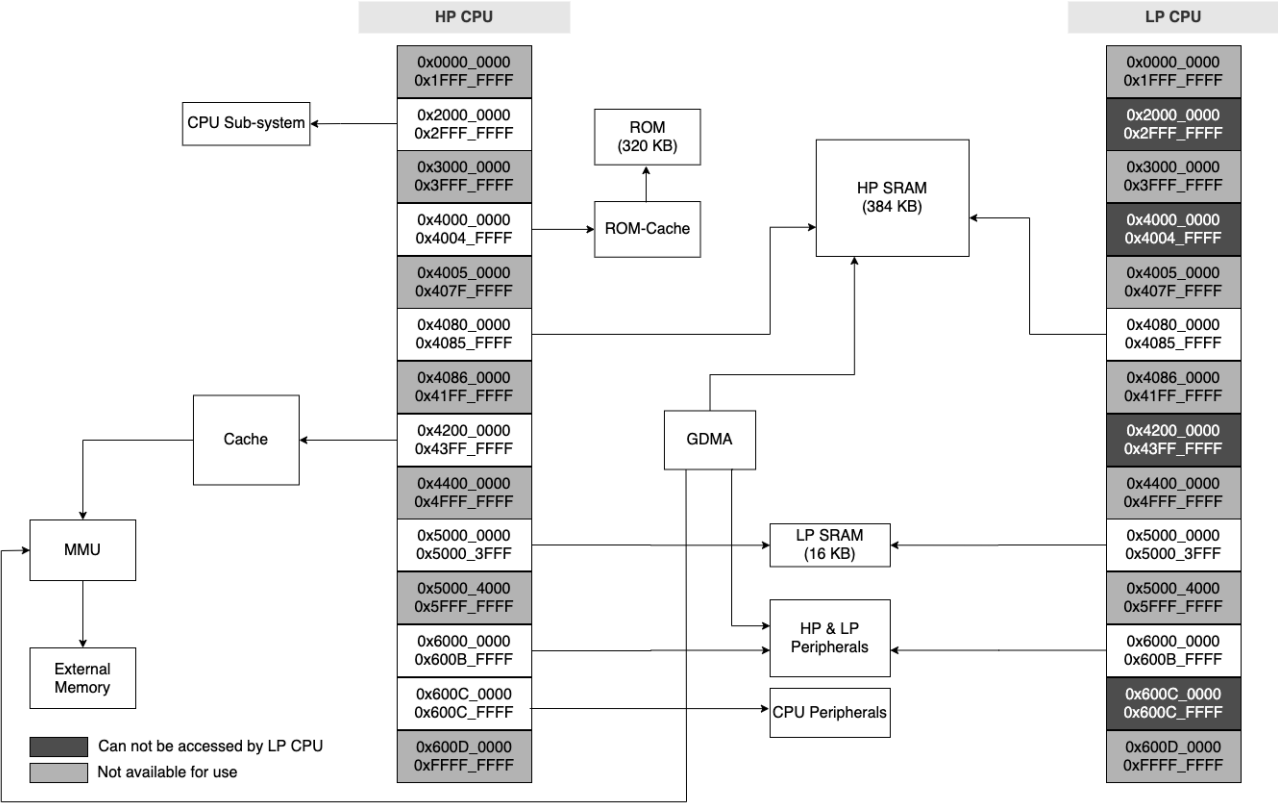


图 4.3-1. 系统结构与地址映射结构

说明：

- 地址空间中可用的地址范围可能大于实际可用的内存。
- 关于 CPU Sub-system，详情请参考章节 1 高性能处理器 [to be added later]。
- 部分地址空间只可被 HP CPU 访问，不能被 LP CPU 访问，已在图 4.3-1 中注明。后文描述中将不再对这部分地址空间进行区分。

表 4.3-1 描述了数据总线与指令总线中的各段地址所能访问的目标。

表 4.3-1. 地址映射

总线类型	边界地址		容量	目标
	低位地址	高位地址		
	0x0000_0000	0x3FFF_FFFF		保留
数据总线/指令总线	0x4000_0000	0x4004_FFFF	320 KB	ROM*
	0x4005_0000	0x407F_FFFF		保留
数据总线/指令总线	0x4080_0000	0x4085_FFFF	384 KB	HP SRAM*
	0x4086_0000	0x41FF_FFFF		保留
数据总线/指令总线	0x4200_0000	0x43FF_FFFF	32 MB	外部存储器
	0x4400_0000	0x4FFF_FFFF		保留
数据总线/指令总线	0x5000_0000	0x5000_3FFF	16 KB	LP SRAM*
	0x5000_4000	0x5FFF_FFFF		保留

见下页

表 4.3-1 – 接上页

总线类型	边界地址		容量	目标
	低位地址	高位地址		
数据总线/指令总线	0x6000_0000	0x600C_FFFF	832 KB	外设
	0x600D_0000	0xFFFF_FFFF		保留

* 所有内部存储器均接受权限管理。只有获取到访问内部存储器的访问权限，才可以执行正常的访问操作，HP CPU 和 LP CPU 访问内部存储器时才可以被响应。关于权限管理的更多信息，请参考章节 13 权限控制 (PMS)。

4.3.2 内部存储器

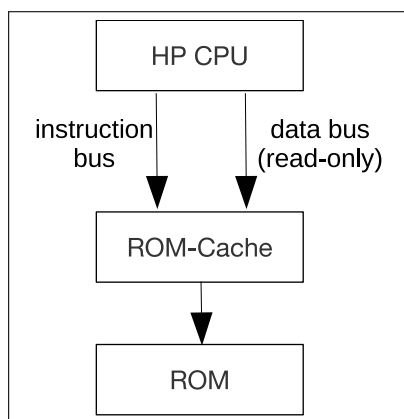
ESP32-C5 的内部存储器包含如下三种类型：

- ROM (320 KB)：只读存储器，不可编程，仅可由 HP CPU 访问。用于存放系统底层软件的 ROM 代码和只读数据。
- HP SRAM (384 KB)：易失性存储器，可快速响应 HP CPU 和 LP CPU 的访问请求。HP CPU 的访问延迟通常是一个 HP CPU 时钟周期。
- LP SRAM (16 KB)：以静态 RAM (SRAM) 方式实现，因此也是易失性存储器。但在 Deep-sleep 模式下，存放在 LP SRAM 中的数据不会丢失。LP SRAM 可被 HP CPU 和 LP CPU 访问，通常用来存放一些在睡眠模式下仍需保持的程序指令和数据。

4.3.2.1 ROM

ROM 的容量为 320 KB，只读。如表 4.3-1 所示，HP CPU 可以通过指令总线或数据总线地址段 0x4000_0000 ~ 0x4004_FFFF 直接或借助 ROM-Cache 访问这部分存储器（见图 4.3-2）。

ESP32-C5 ROM-Cache 为 2 路组相联，容量为 4 KB，块大小为 64 字节。指令总线和数据总线可以同时访问该 ROM-Cache，ROM-Cache 经过仲裁对其中一个做出响应。当 ROM-Cache 缺失时，ROM-Cache 控制器会向 ROM 发起请求。



注：图中 → 表示控制流方向

图 4.3-2. ROM-Cache 系统结构

4.3.2.2 HP SRAM

HP SRAM 的容量为 384 KB，可读可写。如表 4.3-1 所示，HP CPU 和 LP CPU 可以通过数据总线或指令总线访问这部分存储器。

4.3.2.3 LP SRAM

LP SRAM 容量为 16 KB，为可读可写的 SRAM。如表 4.3-1 所示，HP CPU 和 LP CPU 可以通过数据总线和指令总线的共用地址段 0x5000_0000 ~ 0x5000_3FFF 访问这部分存储器。

LP SRAM 存在两种访问模式：

- 高速访问模式：以 HP CPU 时钟频率进行访问，此时 HP CPU 能够无延迟访问 LP SRAM，但是 LP CPU 访问 LP SRAM 的延迟在十几个到几十个 LP CPU 周期不等。
- 低速访问模式：以 LP CPU 时钟频率进行访问，此时 LP CPU 能够无延迟访问 LP SRAM，但是 HP CPU 访问 LP SRAM 的延迟在十几个到几十个 HP CPU 周期不等。

用户可根据应用场景进行模式的切换，比如：

- 如果 LP CPU 未启动，可以切换到高速访问模式，从而提高 HP CPU 访问速度。
- 如果 LP CPU 正在 LP SRAM 执行代码，可以切换到低速访问模式。
- 如果 HP CPU 处于休眠时，必须切换到低速模式。

模式切换配置流程

- 配置 LP_AON_FAST_MEM_MUX_SEL 选择模式：
 - 0：低速访问模式
 - 1：高速访问模式
- 置位 LP_AON_FAST_MEM_MUX_SEL_UPDATE 开始切换模式
- 查询 LP_AON_FAST_MEM_MUX_SEL_STATUS 确认是否切换完成：
 - 0：表示切换完成
 - 1：表示切换未完成

4.3.3 外部存储器

ESP32-C5 支持以 SPI、Dual SPI、Quad SPI、QPI 等接口形式连接片外 flash 和片外 RAM。ESP32-C5 还支持基于 XTS-AES 算法的硬件手动加密和自动解密功能，从而保护开发者片外 flash 和片外 RAM 中的程序和数

据。

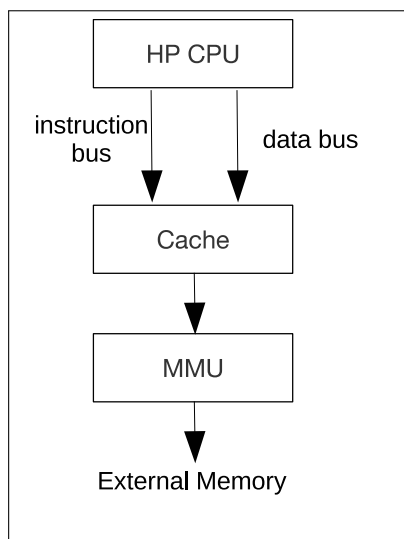
4.3.3.1 外部存储器地址映射

HP CPU 借助高速缓存 (cache) 来访问外部存储器，GDMA 也可以访问外部存储器。内存管理单元 (Memory Management Unit, MMU) 把 HP CPU 和 GDMA 的地址 (0x4200_0000 ~ 0x43FF_FFFF) 映射为访问片外存储器的实地址。经过地址映射，ESP32-C5 最大支持 32 MB 的片外 flash 和 32 MB 的片外 RAM。

请注意，指令总线地址空间 (32 MB) 和数据总线地址空间 (32 MB) 是共用的。

4.3.3.2 Cache

如图 4.3-3 所示，ESP32-C5 采用一个共享 cache，为 4 路组相联，容量为 32 KB，块大小为 32 字节。指令总线和数据总线可以同时访问该 cache，cache 经过仲裁对其中一个做出响应。当 cache 缺失时，cache 控制器会向外部存储器发起请求。



注：图中 → 表示控制流方向

图 4.3-3. Cache 系统结构

4.3.3.3 Cache 操作

ESP32-C5 cache 支持如下几种操作：

1. 回写 (Write-back)

- 清除标记存储器中的脏位，并将新数据更新到外部存储器。
- 回写操作完成后，新数据将被更新到外部存储器中。
- 可以选择是否对 cache 中的数据作失效处理。
- 如果没有选择失效处理，那么 HP CPU 可以直接从 cache 读取/写入数据，而不会发生 cache 缺失。

2. 清除 (Clean)

- 清除标记存储器中的脏位，但不把新数据更新到外部存储器。
- 清理操作完成后，外部存储器中存储的仍然是旧数据，而 cache 会保留新的数据，但 cache 并不知道其存储的为新数据。
- 然后 HP CPU 可以直接从 cache 读取/写入数据，而不会发生 cache 缺失。

3. 失效 (Invalidate)

- 清除标记存储器中的有效位，用于删除 cache 中的有效数据。
- 该操作完成后，删除的数据将仅存于外部存储器中。
- 如果 HP CPU 接着去访问该数据，那么需要访问外部存储器。
- 该操作包括两种类型：
 - 手动失效 (Manual-Invalidate)：仅对 cache 中落入指定区域的地址对应的数据做失效处理。
 - 整体失效 (Invalidate-All)：会对 cache 中的所有数据做失效处理。

4. 预取 (Preload)

- 将指令和数据提前加载到 cache 中。

- 预取操作的最小单位为 1 个 cache 块。
- 该操作包括两种类型：
 - 手动预取 (Manual-Preload)：硬件按软件指定的虚地址预取一段连续的数据。
 - 自动预取 (Auto-Preload)：硬件根据当前命中/缺失（取决于配置）的地址，自动预取一段连续的数据。

5. 锁定/解锁 (Lock/Unlock)

- 保护 cache 中的数据不被替换掉。
- 该锁定操作包括两种类型：
 - 预锁定：在填充缺失数据到 cache 时，如果该数据落在指定区域，cache 则将该数据锁定，未落入指定区域的数据不会被锁定。
 - 手动锁定：检查 cache 中的数据，并将落在指定区域的数据锁定，未落入指定区域的数据不会被锁定。
- 当缺失发生时，cache 会优先替换掉未被锁定的那一路的数据，因此锁定区域的数据会一直保存在 cache 中。
- 但当所有路都被锁定时，cache 将进行正常替换，就像所有路都没有被锁定一样。
- 解锁是锁定的逆操作，但解锁只有手动解锁。
- 请注意，手动失效操作只对未被锁定的数据起作用。如需对已锁定的数据执行手动失效操作，请先解锁这些数据。

4.3.4 GDMA 地址空间

ESP32-C5 中的 GDMA 外设包含三个 TX 通道和三个 RX 通道，可提供 DMA 服务，包括：

- 内部存储器中不同位置的数据搬运
- 模块/外设和内部存储器之间的数据搬运
- 外部存储器中不同位置的数据搬运
- 模块/外设和外部存储器之间的数据搬运
- 内部存储器和外部存储器之间的数据搬运

GDMA 可以通过与数据总线完全相同的地址读写 HP SRAM 和片外 RAM，即 GDMA 通过地址 0x4080_0000 ~ 0x4085_FFFF 访问 HP SRAM，通过地址 0x4200_0000 ~ 0x43FF_FFFF 访问片外 RAM。ESP32-C5 中共有七个外设/模块可以和 GDMA 联合工作。如图 4.3-4 所示，其中的七根竖线依次对应这七个具有 GDMA 功能的外设/模块，横线表示 GDMA 的某一通道（可为任意通道），竖线与横线的交点表示对应外设/模块可以访问 GDMA 的某一通道。

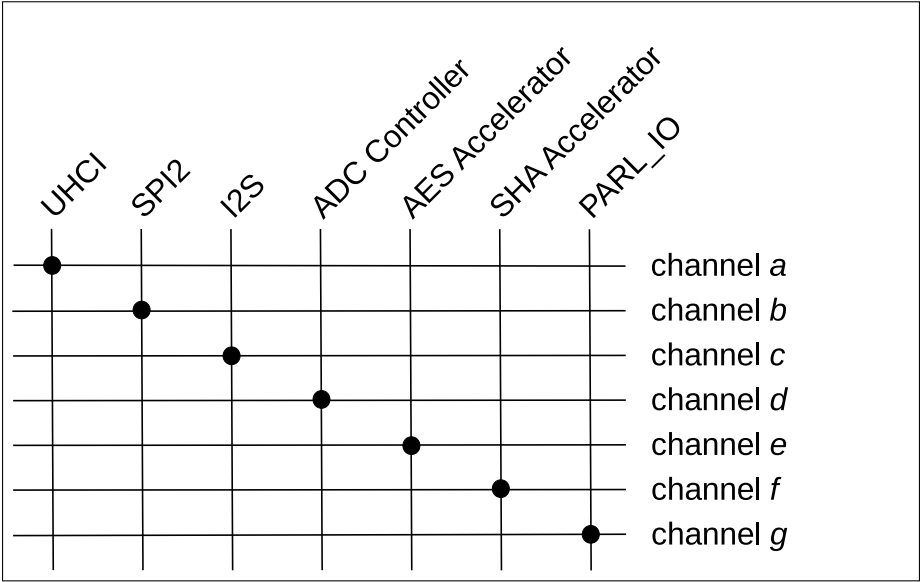


图 4.3-4. 具有 GDMA 功能的外设/模块

具有 GDMA 功能的模块/外设通过 GDMA 可以访问任何 GDMA 可以访问到的存储器。更多关于 GDMA 的信息，请参考章节 3 通用 DMA 控制器 (GDMA)。

说明：
当使用 GDMA 访问任何存储器时，都需要获取对应的访问权限，否则访问将会失败。关于权限管理的更多信息，请参考章节 13 权限控制 (PMS)。

4.3.5 模块/外设地址空间映射

表 4.3-2 详细列出了模块/外设地址空间的各段地址与其能访问到的模块/外设的映射关系。其中，“边界地址”（包括低位地址和高位地址）栏中的两列数值共同决定了对应模块/外设的地址空间。

表 4.3-2. 模块/外设地址空间映射表

目标	边界地址		容量 (KB)
	低位地址	高位地址	
UART 控制器 0 (UART0)	0x6000_0000	0x6000_0FFF	4
UART 控制器 1 (UART1)	0x6000_1000	0x6000_1FFF	4
片外存储器加密与解密 (XTS_AES) ¹	0x6000_2000	0x6000_2FFF	4
SPI 控制器 0 (SPiO) ¹	0x6000_2000	0x6000_2FFF	4
SPI 控制器 1 (SPi1)	0x6000_3000	0x6000_3FFF	4
I2C 控制器 (I2C)	0x6000_4000	0x6000_4FFF	4
UHCI 控制器 (UHCI)	0x6000_5000	0x6000_5FFF	4
红外遥控 (RMT)	0x6000_6000	0x6000_6FFF	4
LED PWM 控制器 (LEDC)	0x6000_7000	0x6000_7FFF	4
定时器组 0 (TIMGO)	0x6000_8000	0x6000_8FFF	4
定时器组 1 (TIMG1)	0x6000_9000	0x6000_9FFF	4

见下页

表 4.3-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
系统定时器 (SYSTIMER)	0x6000_A000	0x6000_AFFF	4
双线汽车接口 0 (TWAIO)	0x6000_B000	0x6000_BFFF	4
I2S 控制器 (I2S)	0x6000_C000	0x6000_CFFF	4
双线汽车接口 1 (TWA1)	0x6000_D000	0x6000_DFFF	4
逐次逼近型模数转换器 (SAR ADC)	0x6000_E000	0x6000_EFFF	4
温度传感器	0x6000_E000	0x6000_EFFF	4
USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)	0x6000_F000	0x6000_FFFF	4
中断矩阵 (INTMTX)	0x6001_0000	0x6001_0FFF	4
保留	0x6001_1000	0x6001_1FFF	
脉冲计数控制器 (PCNT)	0x6001_2000	0x6001_2FFF	4
事件任务矩阵 (SOC_ETM)	0x6001_3000	0x6001_3FFF	4
电机控制器 (MCPWM)	0x6001_4000	0x6001_4FFF	4
并行 IO 控制器 (PARL_IO)	0x6001_5000	0x6001_5FFF	4
保留	0x6001_6000	0x6001_9FFF	
存储器访问监控 2 (PSARM_MEM_MONITOR) 2	0x6001_A000	0x6001_AFFF	4
保留	0x6001_B000	0x6007_FFFF	
通用 DMA 控制器 (GDMA)	0x6008_0000	0x6008_0FFF	4
通用 SPI2 控制器 (GP-SPI2)	0x6008_1000	0x6008_1FFF	4
比特传输控制器 (BITSCRAMBLER)	0x6008_2000	0x6008_2FFF	4
保留	0x6008_3000	0x6008_6FFF	
AES 加速器 (AES)	0x6008_8000	0x6008_8FFF	4
SHA 加速器 (SHA)	0x6008_9000	0x6008_9FFF	4
RSA 加速器 (RSA)	0x6008_A000	0x6008_AFFF	4
ECC 加速器 (ECC)	0x6008_B000	0x6008_BFFF	4
数字签名 (DS)	0x6008_C000	0x6008_CFFF	4
HMAC 加速器 (HMAC)	0x6008_D000	0x6008_DFFF	4
ECDSA 加速器 (ECDSA)	0x6008_E000	0x6008_EFFF	4
保留	0x6008_F000	0x6008_FFFF	
IO MUX	0x6009_0000	0x6009_0FFF	4
GPIO 交换矩阵	0x6009_1000	0x6009_1FFF	4
存储器访问监控 1 (TCM_MEM_MONITOR) 2	0x6009_2000	0x6009_2FFF	4
保留	0x6009_3000	0x6009_4FFF	
高性能系统寄存器 (HP_SYSREG)	0x6009_5000	0x6009_5FFF	4
电源/时钟/复位寄存器 (PCR)	0x6009_6000	0x6009_6FFF	4
保留	0x6009_7000	0x6009_7FFF	
可信执行环境寄存器 (TEE) 2	0x6009_8000	0x6009_8FFF	4
高性能访问权限管理 (HP_APM)	0x6009_9000	0x6009_97FF	2
低功耗访问权限管理 (LP_APM0)	0x6009_9800	0x6009_9FFF	2
CPU 访问权限管理寄存器 (CPU_APM_REG)	0x6009_A000	0x6009_AFFF	4
电源管理单元 (PMU)	0x600B_0000	0x600B_03FF	1

见下页

表 4.3-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
低功耗时钟复位寄存器 (LP_CLKRST)	0x600B_0400	0x600B_07FF	1
保留	0x600B_0800	0x600B_0BFF	
低功耗定时器 (LP_TIMER)	0x600B_0C00	0x600B_0FFF	1
低功耗常开寄存器 (LP_AON)	0x600B_1000	0x600B_13FF	1
低功耗 UART (LP_UART)	0x600B_1400	0x600B_17FF	1
低功耗 I2C (LP_I2C)	0x600B_1800	0x600B_1BFF	1
低功耗看门狗定时器 (LP_WDT)	0x600B_1C00	0x600B_1FFF	1
保留	0x600B_2000	0x600B_23FF	
I2C 模拟主机 (I2C_ANA_MST)	0x600B_2400	0x600B_27FF	1
低功耗外设 (LPPERI)	0x600B_2800	0x600B_2BFF	1
低功耗模拟外设 (LP_ANA_PERI)	0x600B_2C00	0x600B_2FFF	1
低功耗可信执行环境寄存器 (LP_TEE) ²	0x600B_3400	0x600B_37FF	1
低功耗访问权限管理 (LP_APM) ²	0x600B_3800	0x600B_3BFF	1
保留	0x600B_3C00	0x600B_3FFF	
低功耗 IO MUX (LP_IO_MUX)	0x600B_4000	0x600B_43FF	1
低功耗 GPIO 交换矩阵 (LP_GPIO)	0x600B_4400	0x600B_47FF	1
eFuse 控制器 (eFuse)	0x600B_4800	0x600B_4FFF	2
保留	0x600B_5000	0x600B_5FFF	
RISC-V Trace 编码器 (TRACE)	0x600C_0000	0x600C_0FFF	4
保留	0x600C_1000	0x600C_1FFF	
总线访问监控 (BUS_MONITOR) ²	0x600C_2000	0x600C_2FFF	4
保留	0x600C_3000	0x600C_3FFF	
中断优先级寄存器 (INTPRI)	0x600C_5000	0x600C_5FFF	4
保留	0x600C_6000	0x600C_6FFF	
Cache	0x600C_8000	0x600C_8FFF	4
保留	0x600C_9000	0x600C_9FFF	

¹ 片外存储器加密与解密 (XTS_AES) 模块的寄存器空间与 SPI 控制器 0 (SPI0) 模块的寄存器空间共用地址 0x6000_2000 ~ 0x6000_2FFF。

² 该模块/外设的地址空间不连续。

说明:

如图 4.3-1 所示:

- HP CPU 可访问表 4.3-2 列出的所有外设。
- LP CPU 不能访问表 4.3-2 中的 RISC-V Trace 编码器 (TRACE)、辅助调试 (ASSIST_DEBUG)、中断优先级寄存器 (INTPRI) 和 cache，其余外设均可访问。

第 III 卷

系统组件

该部分介绍芯片的系统级功能，包括系统启动、时钟、GPIO、定时器、看门狗、调试辅助、事件和中断处理、低功耗管理以及各种系统寄存器。

第 5 章

GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)

5.1 概述

GPIO 交换矩阵是一种硬件结构，允许芯片上的 GPIO 管脚与外设输入/输出信号之间进行动态的重新映射和交换，以提供更大的灵活性。IO MUX（输入/输出多路复用）是一种技术，允许同一管脚在不同时间承担不同的功能，通过配置实现对管脚功能的动态切换。

ESP32-C5 提供两组 GPIO 交换矩阵和 IO MUX：

- HP GPIO 交换矩阵和 HP IO MUX
- LP GPIO 交换矩阵和 LP IO MUX

ESP32-C5 共有 29 个 GPIO 管脚，编号为 GPIO0~GPIO28，其中包括：

- 7 个 LP GPIO 管脚 (GPIO0~GPIO6)，可由 HP 或 LP 外设使用。
- 14 个 HP GPIO 管脚 (GPIO7~GPIO14 和 GPIO23~GPIO28)，只能由 HP 外设使用。
- 8 个外接 flash 或 PSRAM 专用的 GPIO 管脚 (GPIO15~GPIO22)。这些专用管脚不可用于其他用途，因此不在后续章节进行描述。如需了解专用管脚的更多信息，请参考 [《ESP32-C5 技术规格书》](#) > 章节芯片与 flash 的管脚对应关系。

每个管脚都可用作一个通用 IO，或连接一个内部的外设信号。这些模块共同组成了芯片的 GPIO 控制。

- 利用 HP GPIO 交换矩阵和 HP IO MUX，可配置 HP 外设模块的输入信号来源于**任何的 GPIO 管脚**，并且 HP 外设模块的输出信号也可连接到**任意 GPIO 管脚**。
- 利用 LP GPIO 交换矩阵和 LP IO MUX，可配置 LP 外设模块的输入信号来源于**任何的 LP GPIO 管脚**，并且 LP 外设模块的输出信号也可连接到**任意 LP GPIO 管脚**。

除非另有说明，本章节所述的 GPIO 交换矩阵同时包括 HP GPIO 交换矩阵和 LP GPIO 交换矩阵；IO MUX 同时包括 HP IO MUX 和 LP IO MUX。

5.2 主要特性

5.2.1 HP GPIO 交换矩阵和 HP IO MUX

HP GPIO 交换矩阵具有以下特性：

- HP 外设输入输出信号和 GPIO 管脚之间的全交换矩阵
- 76 个 HP 外设输入信号可以选择任意一个 GPIO 管脚的输入信号
- 每个 GPIO 管脚的输出信号可以来自 77 个 HP 外设输出信号的任意一个
- HP 外设输入信号经 GPIO SYNC 模块同步至 HP IO MUX 运行时钟

- 支持 GPIO 滤波器对输入信号进行滤波
- 支持毛刺滤波器对输入信号进行二次滤波
- 支持 Sigma Delta 调制输出 (SDM)
- 支持 GPIO 简单输入输出
- 支持 HP GPIO 唤醒

HP IO MUX 具有以下特性：

- 控制 21 个 GPIO (GPIO0~GPIO14、GPIO23~GPIO28) 供 HP 外设使用
- 为每个 GPIO 管脚提供一个寄存器 `IO_MUX_GPIO $n$ _REG`，用于控制管脚的输入/输出、上拉/下拉、驱动强度、功能选择等配置
- 支持高频信号如 SPI 直接通过 HP IO MUX 输入和输出外设，实现更好的高频数字特性

5.2.2 LP GPIO 交换矩阵和 LP IO MUX

LP GPIO 交换矩阵具有如下特性：

- 支持 GPIO 简单输入输出
- 支持 LP GPIO 唤醒

LP IO MUX 具有如下特性：

- 7 个 LP GPIO 管脚 (GPIO0~GPIO6) 供 LP 外设使用
- 为每个 LP GPIO 管脚提供一个 `LP_IO_MUX_GPIO $n$ _REG` 寄存器，用于控制管脚的输入/输出、上拉/下拉、驱动强度、功能选择等配置
- 支持使用 `LP_AON_GPIO_MUX_SEL` 来选择管脚控制来自 HP IO MUX 还是 LP IO MUX

5.3 结构概览

图 5.3-1 所示为 HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 将信号引入外设和引出至管脚的具体过程。

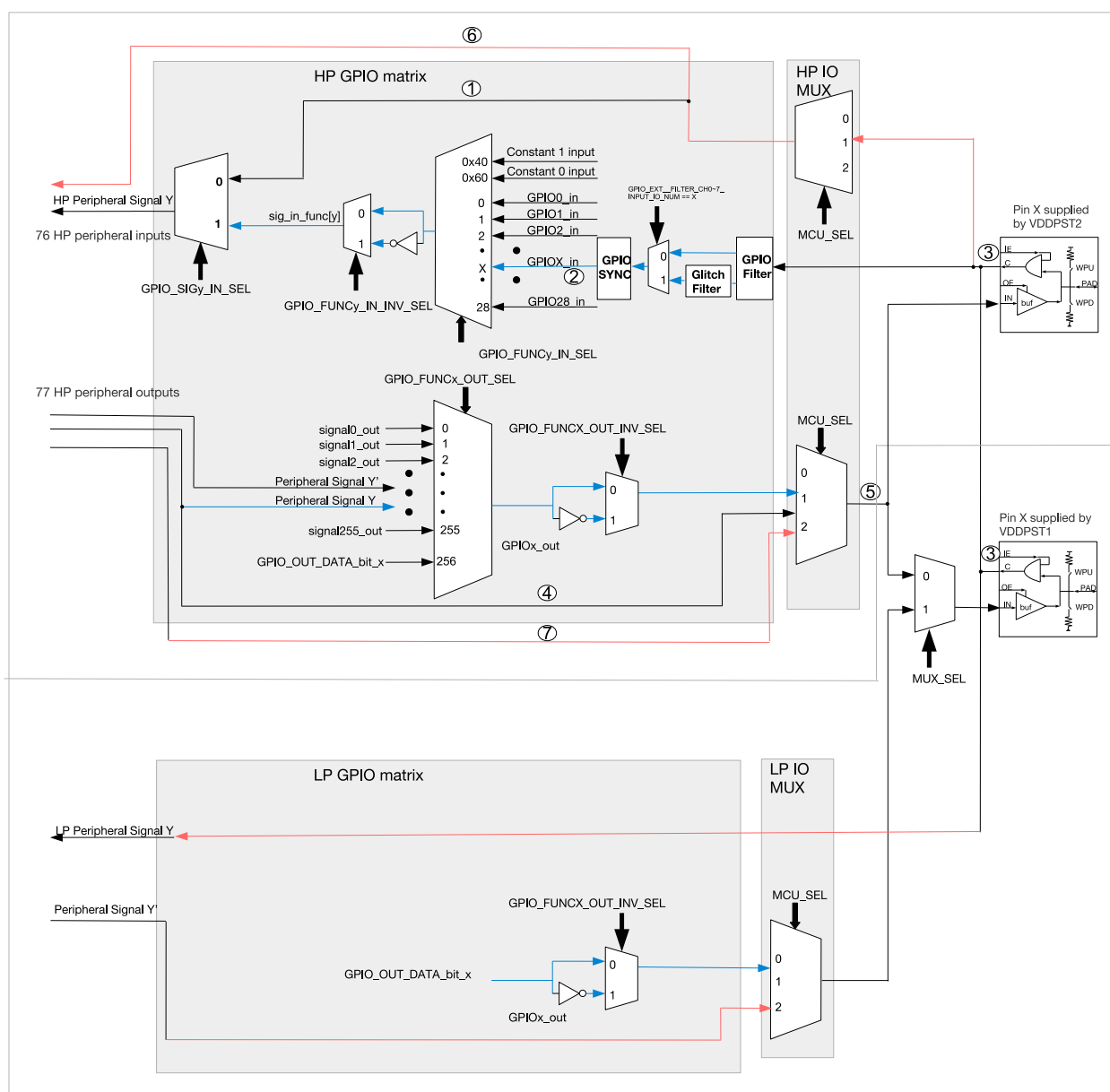


图 5.3-1. HP GPIO 交换矩阵、HP IO MUX、LP GPIO 交换矩阵和 LP IO MUX 框图

以下各点内容是对上图数字标注区域的解释说明，以 HP 系统为例：

① 部分输入信号既可以通过 HP GPIO 交换矩阵连接外设又可以直接通过 HP IO MUX 直连外设，这些输入信号在表 5.12-1 “信号可经由 IO MUX 直接输入”一栏中被标为“yes”。剩余其他信号只能通过 HP GPIO 交换矩阵连接至外设。

② ESP32-C5 共有 21 个用户可用的 GPIO 管脚，因此从 GPIO SYNC 进入到 HP GPIO 交换矩阵的输入共有 21 个。

③ 管脚由 IE、OE、WPU 和 WPD 信号控制。

④ 部分输出信号既可以通过 HP GPIO 交换矩阵连接外设又可通过 HP IO MUX 直连管脚，这些输出信号在表 5.12-1 “信号可经由 IO MUX 直接输出”一栏中被标为“yes”。剩余其他信号只能通过 HP GPIO 交换矩阵连接至管脚。

⑤ 从 HP GPIO 交换矩阵到 HP IO MUX 的输出共有 21 个，对应 GPIO X: 0~14、23~28。

⑥ 仅通过 HP IO MUX 的外设输入。

⑦ 仅通过 HP IO MUX 的外设输出。

图 5.3-2 展示了芯片焊盘 (PAD) 的内部结构，即芯片逻辑与 GPIO 管脚之间的电气接口。ESP32-C5 所有的 GPIO 管脚均采用这一结构，且由 IE、OE、WPU 和 WPD 信号控制。信号配置请参考 `IO_MUX_GPIOx_REG` 寄存器或 `LP_IO_MUX_GPIOx_REG` 寄存器。

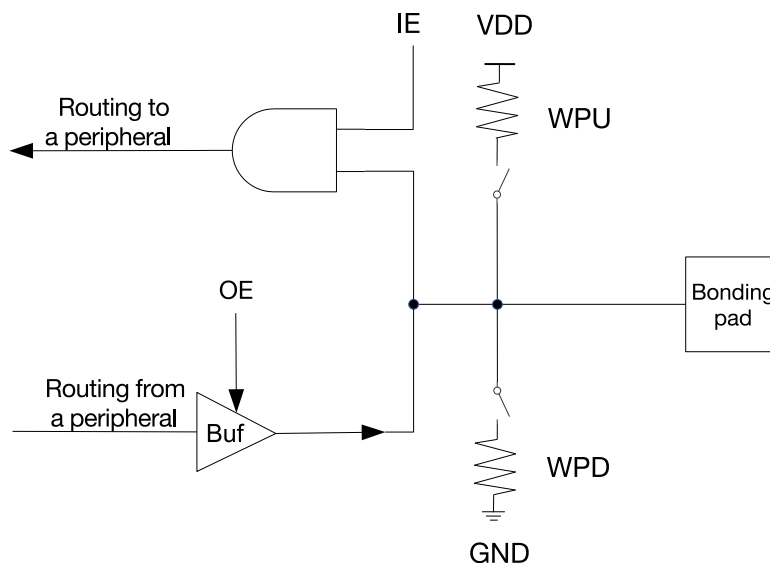


图 5.3-2. 焊盘内部结构

- IE：输入使能
- OE：输出使能
- WPU：内部弱上拉电阻
- WPD：内部弱下拉电阻
- Bonding pad：接合焊盘，芯片逻辑的结点，实现芯片封装内晶片与 GPIO 管脚之间的物理连接。

5.4 通过 GPIO 交换矩阵的外设输入

5.4.1 概述

通过 HP GPIO 交换矩阵接收外部输入信号，一般步骤如下：

- 配置 HP GPIO 交换矩阵从 21 个 GPIO (0~14、23~28) 中获取外部输入信号，见表 5.12-1。
- 配置外设输入选择通过 HP GPIO 交换矩阵接收输入信号。
- 配置 GPIO 管脚由 HP IO MUX 控制。

具体配置流程，可参考图 5.3-1 和章节 5.4.7。

如图 5.3-1 所示，使用 GPIO 交换矩阵将信号从管脚输入时，所有外部输入信号均来源于 GPIO 管脚，然后经 GPIO 滤波器进行滤波，见章节 5.4.7 中的步骤 2。

毛刺滤波器仅在 HP GPIO 交换矩阵中可用。毛刺滤波器硬件可从 GPIO 滤波器输出的信号中选择八个信号进行滤波，其他没被选中的信号则直接进入 GPIO SYNC 硬件，见章节 5.4.7 中的步骤 3。

所有信号经 GPIO 滤波器硬件或毛刺滤波器硬件滤波后，均将被 GPIO SYNC 硬件同步至 IO MUX 运行时钟，然后进入 GPIO 交换矩阵，见章节 5.4.2。外部输入信号也可经 IO MUX 直接输入至外设，但信号无法进行滤波及时钟同步操作。

5.4.2 信号同步

HP GPIO 交换矩阵支持信号同步功能。GPIO SYNC 模块的功能如图 5.4-1 所示。其中，negative sync 为 GPIO 输入经过 HP IO MUX 运行时钟的下降沿同步，positive sync 为 GPIO 输入经过 HP IO MUX 运行时钟的上升沿同步。

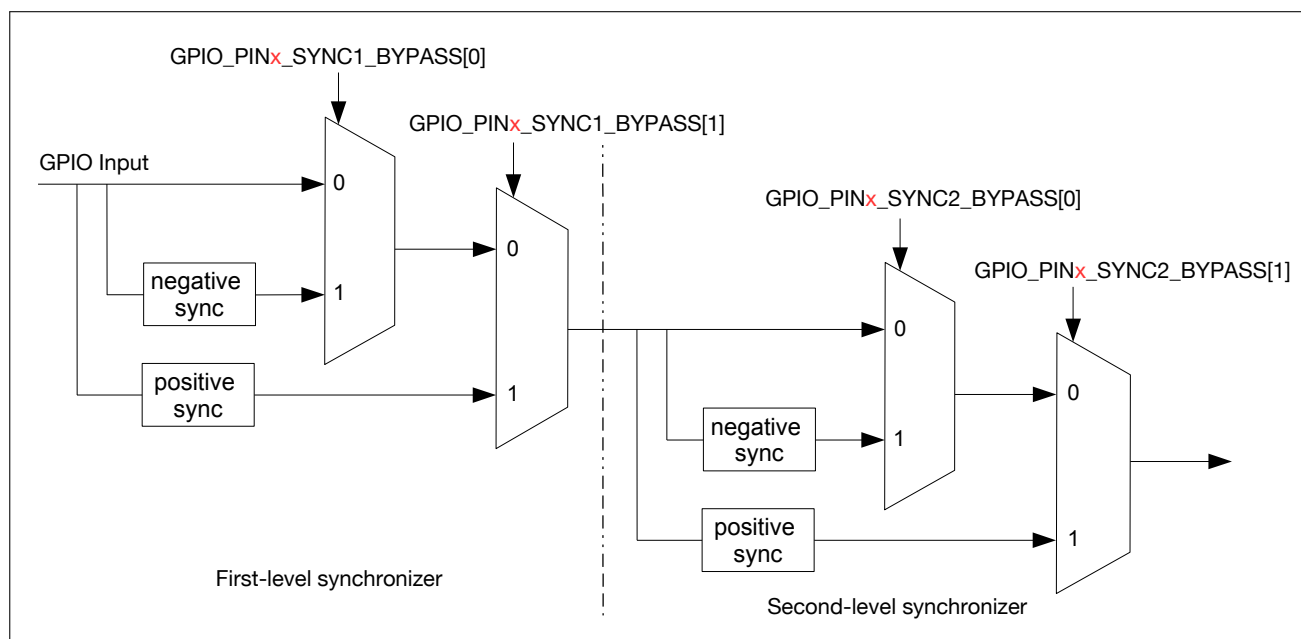


图 5.4-1. GPIO 输入经 HP IO MUX 运行时钟上升沿或下降沿同步

同步器 (synchronizer) 默认关闭同步功能。但如果一个异步外设信号连接到管脚时，该信号应通过两级同步器（即图 5.4-1 中的 first-level synchronizer 和 second-level synchronizer）进行同步，以减小亚稳态产生的概率。

5.4.3 GPIO 滤波器

GPIO 滤波功能仅在 HP GPIO 交换矩阵中可用。使能 GPIO 滤波器功能后，只有当输入信号的有效宽度大于两个时钟周期时，输入信号才会被采样。否则，输入信号将会被滤掉。见图 5.4-2。

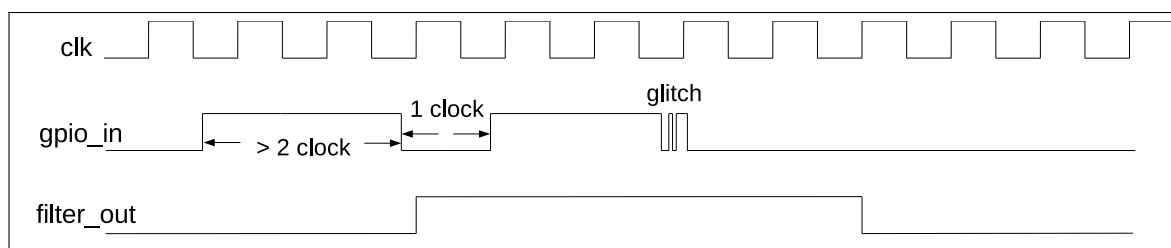


图 5.4-2. GPIO 输入信号滤波时序图

5.4.4 毛刺滤波器

毛刺滤波功能仅在 HP GPIO 交换矩阵中可用。毛刺滤波器硬件支持八个通道，每个通道可从 GPIO 滤波器硬件输出的 21 个信号 (0~14、23~28) 中选择一个信号，进行二次滤波。该毛刺滤波器硬件可用于对慢速信号进行滤波。见章节 5.4.7.1 中的步骤 3。

5.4.5 简单 GPIO 输入

HP GPIO 交换矩阵和 LP GPIO 交换矩阵均支持简单 GPIO 输入。使用简单 GPIO 输入模式时，任意 GPIO 管脚的值均可随时读取，而无需将 GPIO 管脚输入绑定到某个外设信号。

配置 HP GPIO 交换矩阵实现简单 GPIO 输入，具体过程如下：

- 置位 `IO_MUX_GPIOx_REG` 的 `IO_MUX_GPIOx_MCU_IE` 位，使能管脚输入。
- 读取 `GPIO_IN_REG[x]`，即可实现简单 GPIO 输入。

配置 LP GPIO 交换矩阵实现简单 GPIO 输入，具体过程如下：

- 置位 `LP_IO_MUX_GPIOx_REG` 的 `LP_IO_MUX_GPIOx_FUN_IE` 位，使能管脚输入。
- 读取 `LP_GPIO_IN_REG[x]`，即可实现简单 GPIO 输入。

5.4.6 GPIO 唤醒

5.4.6.1 HP GPIO 唤醒

当 HP 系统处于 Light-sleep 模式时，HP GPIO 唤醒功能可用于将系统从 Light-sleep 模式唤醒。当 HP 系统掉电时，HP GPIO 唤醒功能不可用。可通过配置 `GPIO_PINx_WAKEUP_ENABLE` (x : 0~14, 23~28) 使能 HP GPIO 唤醒功能。

HP GPIO 支持高电压敏感和低电压敏感两种唤醒方式，具体由 `GPIO_PINx_INT_TYPE` 配置决定，见下表。

表 5.4-1. HP GPIO 唤醒信号触发条件和清除条件

<code>GPIO_PINx_INT_TYPE</code> ^{1,2}	触发条件	清除条件
5	GPIO 输入为高电平	GPIO 输入为低电平
4	GPIO 输入为低电平	GPIO 输入为高电平

¹ x = 0~14、23~28。

² 配置为其他值时，HP GPIO 唤醒功能不可用。

5.4.6.2 LP GPIO 唤醒

LP GPIO 唤醒功能可用于将系统从 Deep-sleep 模式唤醒。当 LP 外设掉电时，LP GPIO 唤醒功能不可用。GPIO0~GPIO6 可通过配置 `LP_GPIO_PINx_WAKEUP_ENABLE` (x = 0~6) 使能 LP GPIO 唤醒功能。

LP GPIO 唤醒支持上升沿敏感、下降沿敏感、上升沿和下降沿敏感、高电压敏感以及低电压敏感唤醒。具体情况由寄存器配置决定，见下表。

表 5.4-2. LP GPIO 唤醒信号触发条件和清除条件

LP_GPIO_PIN x _INT_TYPE ^{1,2}	触发条件	清除条件
1	GPIO 输入从低电平跳变为高电平	置位 LP_GPIO_PIN x _EDGE_WAKEUP_CLR
2	GPIO 输入从高电平跳变为低电平	置位 LP_GPIO_PIN x _EDGE_WAKEUP_CLR
3	GPIO 输入从高电平跳变为低电平， 或从低电平跳变为高电平	置位 LP_GPIO_PIN x _EDGE_WAKEUP_CLR
4	GPIO 输入为低电平	GPIO 输入为高电平
5	GPIO 输入为高电平	GPIO 输入为低电平

¹ $x = 0 \sim 6$ 。
² 配置为其他值时，LP GPIO 唤醒功能不可用。

5.4.7 编程流程

5.4.7.1 HP GPIO 交换矩阵

把某个 HP 外设输入信号 Y 绑定到某个 GPIO 管脚 X 的配置过程如下：

- 在 HP GPIO 交换矩阵中配置外设信号 Y 的 GPIO_FUNC y _IN_SEL_CFG_REG 寄存器：
 - 置位 GPIO_SIG y _IN_SEL 选择通过 HP GPIO 交换矩阵接收外部输入信号。
 - 设置 GPIO_FUNC y _IN_SEL 为需要的 GPIO 管脚编号，此处应为 X 。

注意：并不是所有的 HP 外设信号都有有效的 GPIO_SIG y _IN_SEL 位，即有些外设信号只能通过 HP GPIO 交换矩阵接收外部输入信号。

- 可选：置位 IO_MUX_GPIO n _FILTER_EN 使能 GPIO 管脚的输入信号滤波功能。
- 启用毛刺滤波器功能，步骤如下：
 - 配置 GPIO_EXT_FILTER_CH n _INPUT_IO_NUM 为 m 。此处 n 为通道编号，取值范围为：0~7。 m 为 GPIO 管脚编号，取值范围为：0~14、23~28。
 - 配置 GPIO_EXT_FILTER_CH n _WINDOW_WIDTH 为 VALUE1，配置 GPIO_EXT_FILTER_CH n _WINDOW_THRES 为 VALUE2。在 VALUE1 + 1 个周期内，如果存在 VALUE2 + 1 个输入信号与当前输出信号值不匹配，毛刺滤波器硬件将反转输出信号。GPIO_EXT_FILTER_CH n _WINDOW_WIDTH 和 GPIO_EXT_FILTER_CH n _WINDOW_THRES 可以配置为相同的值 VALUE3，那么只有宽度大于 VALUE3 + 1 个时钟周期的信号才会被采样。
 - 置位 GPIO_EXT_FILTER_CH n _EN 使能通道 n 。

相关示例见图 5.4-3，其中 GPIO_EXT_FILTER_CH x _WINDOW_WIDTH 配置为 3
GPIO_EXT_FILTER_CH x _WINDOW_THRES 配置为 2。在 T1 之前的四个时钟周期内，输出信号值 (signal_out) 一直为 “0”。输入信号值 (signal_in) 有三个时钟周期为 “1”，则输出信号在 T1 之后反转为 “1”。

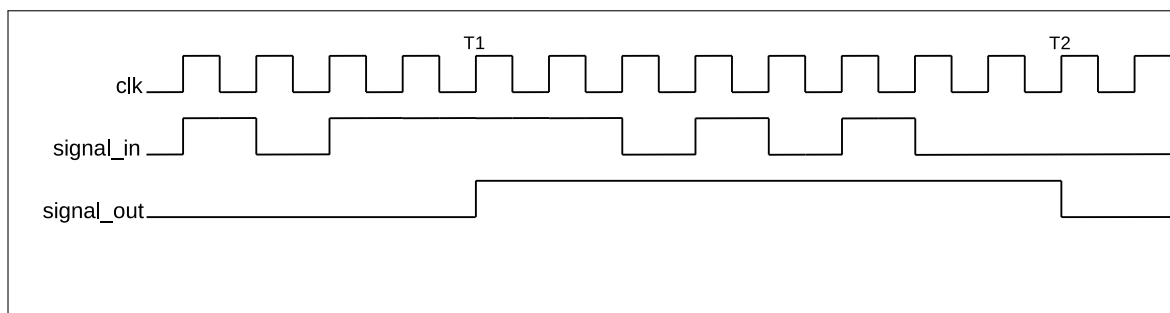


图 5.4-3. 毛刺滤波器时序示例

4. 同步 GPIO 输入信号。配置 GPIO 管脚 X 的 `GPIO_PIN $x$ _REG` 来同步 GPIO 输入信号，过程如下：

- 如图 5.4-1 所示，配置 `GPIO_PIN $x$ _SYNC1_BYPASS` 使能输入信号在第一级同步中为上升沿或下降沿同步。
- 如图 5.4-1 所示，配置 `GPIO_PIN $x$ _SYNC2_BYPASS` 使能输入信号在第二级同步中为上升沿或下降沿同步。

5. 配置 HP IO MUX 寄存器使能 GPIO 管脚的输入功能。配置 GPIO 管脚 X 的 `IO_MUX_GPIO $x$ _REG`，过程如下：

- 置位 `IO_MUX_GPIO $x$ _FUN_IE` 使能输入^{2,3}。
- 置位或清零 `IO_MUX_GPIO $x$ _FUN_WPU` 和 `IO_MUX_GPIO $x$ _FUN_WPD`，使能或关闭内部上拉/下拉电阻。

说明：

1. 同一个输入管脚可以同时绑定多个输入信号。
2. 置位 `GPIO_FUNC $y$ _IN_INV_SEL` 可以把输入信号取反。
3. 无需将输入信号绑定到一个 GPIO 管脚也可以使 HP 外设读取恒低或恒高电平的输入值。实现方式为选择特定的 `GPIO_FUNC $y$ _IN_SEL` 输入值而不是一个 GPIO 序号：
 - 设置 `GPIO_FUNC $y$ _IN_SEL` 为 0x40，则输入信号恒为 1。
 - 设置 `GPIO_FUNC $y$ _IN_SEL` 为 0x60，则输入信号恒为 0。

编程示例

要把 UART0 RXD 输入信号³ (UORXD_in，信号索引号 6) 绑定到 GPIO7，请按照以下步骤操作：

1. 置位 `GPIO_FUNC6_IN_SEL_CFG_REG` 中的 `GPIO_SIG6_IN_SEL` 位，使能通过 HP GPIO 交换矩阵接收外部输入信号。
2. 配置 `GPIO_FUNC6_IN_SEL_CFG_REG` 中的 `GPIO_FUNC6_IN_SEL` 为 7，即选择管脚 GPIO7。
3. 置位 `IO_MUX_GPIO7_REG` 中 `IO_MUX_GPIO7_FUN_IE` 位使能管脚输入。

5.4.7.2 LP GPIO 交换矩阵

LP GPIO 交换矩阵输入仅支持 GPIO 唤醒，配置 LP GPIO 输入唤醒可参考章节 5.4.6.2。

5.5 通过 GPIO 交换矩阵的外设输出

5.5.1 概述

通过 GPIO 交换矩阵输出外设信号，一般步骤如下：

- 配置 HP GPIO 交换矩阵将 HP 外设信号（即在表 5.12-1 中“输出信号”一栏所列出的信号）输出到 21 个 GPIO (0~14、23~28) 管脚
- 输出信号从外设输出到 GPIO 交换矩阵，然后到达 IO MUX。IO MUX 必须设置相应管脚为 GPIO 功能，这样输出 GPIO 信号就能连接到相应管脚。

具体的编程步骤见章节 5.5.4.1。

说明：

表 5.12-1 中输出索引号为 97~100 的外设信号，没有连接至外设，可配置为从一个 GPIO 管脚输出后，直接由另一个 GPIO 管脚输入（索引号：97~100）。

5.5.2 简单 GPIO 输出

GPIO 交换矩阵也可用于简单 GPIO 输出，即 GPIO 管脚可直接输出所期望的输出值，而无需将某个外设信号绑定到该 GPIO 管脚。

对于 HP GPIO 交换矩阵，具体配置如下：

- 设置 `GPIO_FUNCx_OUT_SEL` 为特定的外设索引值 256 (0x100)。
- 设置 `GPIO_OUT_REG` 中相应位的值为期望 GPIO 输出的值。

说明：

- `GPIO_OUT_REG[0~14 和 23~28]` 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。
- 推荐使用相应的 W1TS 和 W1TC 寄存器，例如 `GPIO_OUT_W1TS` 来置位/清除 `GPIO_OUT_REG`。

对于 LP GPIO 交换矩阵，仅支持简单 GPIO 输出，因此无需配置功能选择寄存器，仅需设置 `LP_GPIO_OUT_REG` 中相应位的值为期望 GPIO 输出的值。

说明：

推荐使用相应的 W1TS 和 W1TC 寄存器，例如 `LP_GPIO_OUT_W1TS` 或 `LP_GPIO_OUT_W1TC` 来置位/清除 `LP_GPIO_OUT_REG`。

5.5.3 Sigma Delta 调制输出 (SDM)

5.5.3.1 功能描述

HP 外设输出信号中有四个信号（在表 5.12-1 中索引为：76~79）支持 1-bit 二阶 SDM 调制输出。上述四个信号通道默认输出使能。Sigma Delta 调制器可实现输出可配占空比的 PDM（脉冲密度调制）信号。二阶 Sigma Delta 调制器函数为：

$$H(z) = X(z)z^{-1} + E(z)(1-z^{-1})^2$$

$E(z)$ 为量化误差, $X(z)$ 为输入。

Sigma Delta 调制器内部支持对 IO MUX 运行时钟的 1~256 倍分频:

- 置位 `GPIO_EXT_SIGMADELTA_CLK_EN` 使能调制器时钟。
- 配置 `GPIO_EXT_SD $n$ _PRESCALE` 实现分频。 n 取值范围为 0~3, 对应四个信号通道。

分频后的时钟周期为调制器输出单位脉冲的周期。

`GPIO_EXT_SD $n$ _IN` 为有符号数, 范围为 [-128, 127], 配置此寄存器控制输出 PDM 信号的占空比¹。

- `GPIO_EXT_SD $n$ _IN` = -128, 调制器输出信号占空比为 0%。
- `GPIO_EXT_SD $n$ _IN` = 0, 调制器输出信号占空比接近 50%。
- `GPIO_EXT_SD $n$ _IN` = 127, 调制器输出信号占空比接近 100%。

PDM 信号占空比计算公式为:

$$Duty\_Cycle = \frac{GPIO\_EXT\_SDn\_IN + 128}{256}$$

说明:

对 PDM 信号来说, 占空比是指在若干脉冲周期内 (比如 256 个脉冲周期), 高电平占整个统计周期的比值。

5.5.3.2 配置方法

SDM 的配置方法如下:

- 将 SDM 输出经 HP GPIO 交换矩阵连接至相应管脚, 见章节 5.5.4。
- 置位 `GPIO_EXT_SIGMADELTA_CLK_EN`, 使能 SDM 时钟
- 配置 `GPIO_EXT_SD $n$ _PRESCALE` 设置时钟分频系数。
- 配置 `GPIO_EXT_SD $n$ _IN` 设置 SDM 输出信号的占空比。

5.5.4 编程流程

5.5.4.1 HP GPIO 交换矩阵

如图 5.3-1 所示, 表 5.12-1 中“输出信号”列的某一个信号通过 HP GPIO 交换矩阵到达 HP IO MUX, 然后连接到某个 GPIO 管脚。

输出外设信号 Y 到某一 GPIO 管脚 X^1 的步骤如下:

1. 在 HP GPIO 交换矩阵中配置 GPIO 管脚 X 的 `GPIO_FUNC $x$ _OUT_SEL_CFG_REG` 和 `GPIO_ENABLE_REG[ $x$ ]`。推荐使用相应 WITS (写 1 置位) 和 WITC (写 1 清零) 来更新 `GPIO_ENABLE_REG` 中的值。
 - 设置 `GPIO_FUNC $x$ _OUT_SEL_CFG_REG` 中的 `GPIO_FUNC $x$ _OUT_SEL` 字段为外设输出信号 Y 的索引 (Y)。

- 要将信号强制使能输出模式，需要将 GPIO 管脚 X 对应的 `GPIO_FUNC $x$ _OUT_SEL_CFG_REG` 寄存器中 `GPIO_FUNC $x$ _OE_SEL` 字段置位；同时需要将 `GPIO_ENABLE_W1TS_REG` 中的相应位置位。或者，将 `GPIO_FUNC $x$ _OE_SEL` 清零，即选择采用外设的输出使能信号，此时输出使能信号由内部逻辑功能决定。比如，表 5.12-1 中“`GPIO_FUNC $n$ _OE_SEL = 0` 时输出信号的输出使能信号”一栏的 `SPIQ_oe` 信号。
 - 置位 `GPIO_ENABLE_W1TC_REG` 中相应位可以关闭 GPIO 管脚的输出。
2. 要选择以开漏方式输出，可以设置 GPIO 管脚 X 的 `GPIO_PIN $x$ _REG` 寄存器中 `GPIO_PIN $x$ _PAD_DRIVER` 位。
 3. 配置 HP IO MUX 寄存器来选择经由 HP GPIO 交换矩阵输出信号。配置 GPIO 管脚 X 的 `IO_MUX_GPIO $x$ _REG` 的过程如下：
 - 设置 GPIO 管脚 X 的 `IO_MUX_GPIO $x$ _MCU_SEL` 为所需的管脚功能。此处选择数值 1，即功能 1 (GPIO 功能)，适用于所有管脚。
 - 设置 `IO_MUX_GPIO $x$ _FUN_DRV` 字段为特定的输出强度值 (0~3)，值越大，输出驱动能力越强。
 - 在开漏模式下，通过置位/清零 `IO_MUX_GPIO $x$ _FUN_WPU` 和 `IO_MUX_GPIO $x$ _FUN_WPD` 使能或关闭上拉/下拉电阻。
 4. 使能迟滞功能：
 - 请参考章节 5.10。

说明：

- 某一个外设的输出信号可以同时从多个管脚输出。
- 置位 `GPIO_FUNC $x$ _OUT_INV_SEL` 可以把输出的信号取反。

5.5.4.2 LP GPIO 交换矩阵

LP GPIO 交换矩阵输出仅支持简单 GPIO 输出，配置 LP GPIO 简单输出可参考章节 5.5.2。

5.6 IO MUX 的直接输入输出功能

5.6.1 概述

SPI、JTAG 等数字信号可以旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以此类信号可以直接通过 IO MUX 输入和输出。这样比使用 GPIO 交换矩阵的灵活度要低，即每个 GPIO 管脚的 IO MUX 寄存器只有较少的功能选择，但可以实现更好的高频数字特性。

ESP32-C5 中有 7 个 GPIO 管脚 (GPIO0~GPIO6) 具有低功耗 (LP) 性能。这些管脚可由 HP IO MUX 或 LP IO MUX 控制。如果由 LP IO MUX 控制，这些管脚将旁路 HP IO MUX 和 HP GPIO 交换矩阵，直接供 LP 系统中的外设使用。

当这些管脚被配置为 LP GPIO 管脚，仍然能够在芯片处于 Deep-sleep 模式下由 LP 系统的外设控制，也可以作为唤醒源将芯片从 Deep-sleep 中唤醒。

5.6.2 功能描述

LP 管脚 (GPIO0~GPIO6) 由 LP_AON_GPIO_MUX_REG 中的 LP_AON_GPIO_MUX_SEL 的 bit[n] ($n = 0 \sim 6$) 位控制。此位默认为 0，通过 HP IO MUX 输入输出信号。

如果置位 LP_AON_GPIO_MUX_SEL 的 bit[n]，则控制权交由 LP IO MUX 接管。在这种模式下，LP_IO_MUX_GPIO n _REG 寄存器用于控制 LP GPIO 管脚。表 5.14-1 列出了 LP GPIO 管脚的 LP 功能。LP_IO_MUX_GPIO n _REG 寄存器使用的是 LP GPIO 管脚的序号，ESP32-C5 芯片中 LP GPIO 管脚的序号与 HP GPIO 管脚的序号相同。

5.6.2.1 HP IO MUX

对于 HP 外设输入信号，旁路 HP GPIO 交换矩阵必须配置以下两个字段：

1. GPIO 管脚的 IO_MUX_GPIO n _MCU_SEL 必须配置为相应的管脚功能。表 5.13-1 列出了管脚功能。
2. 清除 GPIO_SIG n _IN_SEL，直接将输入信号连接到外设。

对于 HP 外设输出信号，旁路 HP GPIO 交换矩阵只需将 GPIO 管脚的 IO_MUX_GPIO n _MCU_SEL 配置为相应的管脚功能即可。

5.6.2.2 LP IO MUX

对于 LP 外设输入信号，旁路 LP GPIO 交换矩阵必须配置 LP_IO_MUX_GPIO n _MCU_SEL 字段：GPIO 管脚的 LP_IO_MUX_GPIO n _MCU_SEL 必须配置为相应的管脚功能。表 5.14-1 列出了管脚功能。

对于 LP 外设输出信号，旁路 LP GPIO 交换矩阵只需将 LP GPIO 管脚的 LP_IO_MUX_GPIO n _MCU_SEL 配置为相应的管脚功能即可。

说明：

并非所有外设输入/输出信号均可直接通过 IO MUX 连接到外设，某些输入/输出信号只能通过 HP GPIO 交换矩阵连接到外设，详见表 5.12-1。

5.7 模拟功能描述

5.7.1 概述

ESP32-C5 共有 11 个 GPIO 管脚 (GPIO0~GPIO6, GPIO8~GPIO9, GPIO13~GPIO14) 具有模拟功能，其中包括 6 个 LP GPIO 管脚和 5 个 HP GPIO 管脚。见表 5.15-1。

5.7.2 模拟功能描述

当使用管脚的模拟功能时，需要确保该管脚处于浮空状态，此时外部模拟信号通过 GPIO 管脚直接与内部的模拟信号相连。相关配置如下：

- 清除 IO_MUX_GPIO n _FUN_IE、IO_MUX_GPIO n _FUN_WPU 和 IO_MUX_GPIO n _FUN_WPD 将管脚设置为浮空状态。
- 向 GPIO_ENABLE_W1TC 中对应位写 1，清除输出使能。

对于表 5.15-1 中列出的不同模拟功能的使用方法，请参考相关章节：

- XTAL_32K 相关功能见章节 4 低功耗管理 [to be added later]。
- ADC 相关功能见章节 36 ADC 控制器。
- 模拟电压比较器功能见章节 37 模拟电压比较器。

说明：
表 5.15-1 中 GPIO1 有两种模拟功能，可以同时使用。

5.8 Light-sleep 模式管脚功能

当 ESP32-C5 处于 Light-sleep 模式时管脚可以有不同的功能。如果某一 GPIO 管脚的 IO_MUX_GPIO n _REG 寄存器中 IO_MUX_GPIO n _SLP_SEL 置位时，芯片处于 Light-sleep 模式下将由另一组不同的寄存器控制管脚。

表 5.8-1. IO MUX Light-sleep 管脚功能控制寄存器

IO MUX 功能	正常工作模式 OR IO_MUX_GPIO n _SLP_SEL = 0	Light-sleep 模式 AND IO_MUX_GPIO n _SLP_SEL = 1
输出驱动强度	IO_MUX_GPIO n _FUN_DRV	IO_MUX_GPIO n _MCU_DRV
上拉电阻	IO_MUX_GPIO n _FUN_WPU	IO_MUX_GPIO n _MCU_WPU
下拉电阻	IO_MUX_GPIO n _FUN_WPD	IO_MUX_GPIO n _MCU_WPD
输入使能	IO_MUX_GPIO n _FUN_IE	IO_MUX_GPIO n _MCU_IE
输出使能	由 GPIO 交换矩阵的 OE_SEL*	IO_MUX_GPIO n _MCU_OE

说明：
如果 IO_MUX_GPIO n _SLP_SEL 置为 0，则芯片在正常工作和 Light-sleep 模式下，管脚的功能一样。此时，具体的输出使能配置请参考 5.5.4 章节。

5.9 GPIO 管脚的 Hold 特性

每个 GPIO 管脚都有单独的 Hold 功能，由电源管理单元 (PMU) 或寄存器控制。管脚的 Hold 功能被置上后，管脚在置上 Hold 那一刻的状态被强制保持，无论内部信号如何变化，修改 IO MUX 配置或者 GPIO 配置，都不会改变管脚的状态。应用如果希望在看门狗超时触发内核复位时或者 Deep-sleep 触发内核复位时管脚的状态不被改变，就需要提前把 Hold 置上。

每个 GPIO 管脚的 Hold 状态由各管脚的 Hold 使能信号和全局的 Hold 使能信号经过或运算后共同控制。

- 各个 GPIO 管脚的 Hold 信号
 - LP_AON_GPIO_HOLD0_REG[n] ($n = 0\sim14, 23\sim28$) 控制各个 GPIO 管脚的 Hold 信号。
- 全局 Hold 信号
 - HP GPIO 的全局 Hold 信号
 - * PMU_TIE_HIGH_HP_PAD_HOLD_ALL：用于使能所有 HP GPIO 管脚的全局 Hold 信号。

* PMU_TIE_LOW_HP_PAD_HOLD_ALL: 用于关闭所有 HP GPIO 管脚的全局 Hold 信号。

– LP GPIO 的全局 Hold 信号

* PMU_TIE_HIGH_LP_PAD_HOLD_ALL: 用于使能 LP GPIO 管脚的全局 Hold 信号。

* PMU_TIE_LOW_LP_PAD_HOLD_ALL: 用于关闭所有 LP GPIO 管脚的全局 Hold 信号。

可通过以下任一方式使能或关闭管脚的 Hold 功能。

• **方式 1: 使能或关闭某一个管脚的 Hold 功能**

如要在 Deep-sleep 中保持该管脚的输入输出的状态值, 需要置位 LP_AON_GPIO_HOLD0_REG[n] 中的相应位, 其中 $n = 0 \sim 14, 23 \sim 28$, 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。在芯片被唤醒后, 若要关闭 Hold 功能, 可将 LP_AON_GPIO_HOLD0_REG 中的相应位 bit[n] 清除。

• **方式 2: 使能或关闭所有 HP 管脚和 LP 管脚的 Hold 功能**

– HP 管脚

置位 PMU_TIE_HIGH_HP_PAD_HOLD_ALL 可保持所有 HP 管脚的输入输出状态; 置位 PMU_TIE_LOW_HP_PAD_HOLD_ALL 可关闭所有 HP 管脚的 Hold 功能。

– LP 管脚

置位 PMU_TIE_HIGH_LP_PAD_HOLD_ALL 可保持所有 LP 管脚的输入输出状态; 置位 PMU_TIE_LOW_LP_PAD_HOLD_ALL 可关闭所有 LP 管脚的 Hold 功能。

• **方式 3: 使用 PMU 任务**

在芯片进入 Deep-sleep 前, 可配置 PMU 任务, 用以保持所有 HP 管脚的状态。进入 Deep-sleep 后, PMU 可自动保持这些 HP 管脚的值。

5.10 GPIO 管脚的迟滞特性

每个 GPIO 管脚都具有迟滞功能。当未使能迟滞功能时, 如图 5.10-1 所示, 芯片焊盘 (PAD) 上输入给芯片内部的信号 (C) 的电平翻转只有一个阈值 (V_t , 约 1.7 V)。当 PAD 的电压高于 V_t 时, C 的电平为高, 否则为低。但如果 PAD 上的信号有噪声, 可能会影响 C 上的信号。

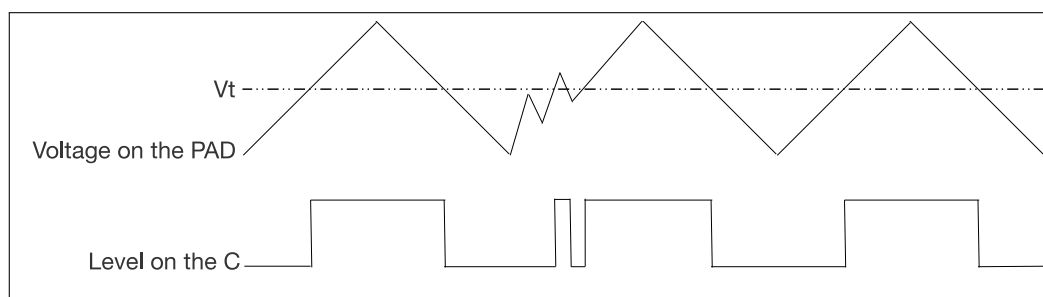


图 5.10-1. 未使能迟滞功能时 PAD 上的电平翻转示例

当使能迟滞功能时, 如图 5.10-2 所示, C 的电平翻转有两个阈值, 分别为高电平阈值 (V_{th} , 约 1.7 V) 和低电平阈值 (V_{tl} , 约 1.4 V)。当 PAD 的电压由低变高时, 如果电压高于 V_{th} , 则 C 的电平为高。当 PAD 的电压由高变低时, 若电压低于 V_{tl} , 则 C 的电平为低。当 PAD 的电压在 V_{th} 和 V_{tl} 之间时, C 的电平不变。迟滞功能可以减少噪声的影响, 起到抗干扰作用, 减少 C 的电平翻转次数。

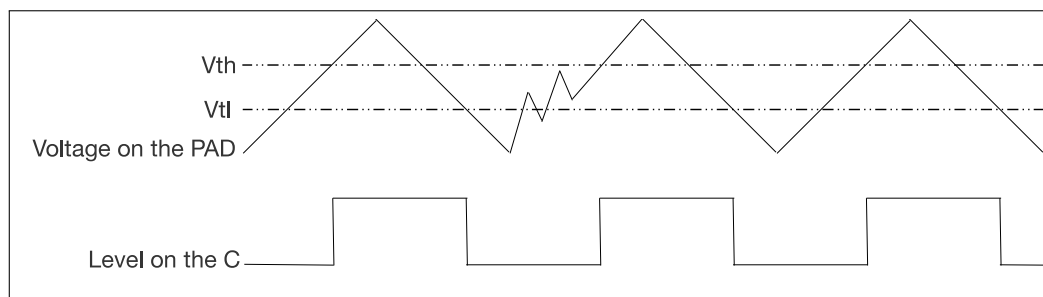


图 5.10-2. 使能迟滞功能时 PAD 上的电平翻转示例

要使能迟滞功能，请按照以下步骤操作：

- 当 `IO_MUX_GPIO $n$ _HYS_SEL` 为 0 时 (n 的取值范围是 0~14、23~28，分别对应 GPIO0~GPIO14、GPIO23~GPIO28)：
 - 当 `EFUSE_HYS_EN_PAD` 为 1 时，开启 GPIO0 ~GPIO14、GPIO23~GPIO28 的迟滞功能。
 - 当 `EFUSE_HYS_EN_PAD` 为 0 时，关闭 GPIO0 ~GPIO14、GPIO23~GPIO28 的迟滞功能。
- 当 `IO_MUX_GPIO $n$ _HYS_SEL` 为 1 时 (n 的取值范围是 0~14、23~28，分别对应 GPIO0~GPIO14、GPIO23~GPIO28)：
 - 设置 `IO_MUX_GPIO $n$ _HYS_EN` 为 1，可开启 GPIO n 的迟滞功能
 - 设置 `IO_MUX_GPIO $n$ _HYS_EN` 为 0，可关闭 GPIO n 的迟滞功能

建议设置 `IO_MUX_GPIO $n$ _HYS_SEL` 为 1，使用 `IO_MUX_GPIO $n$ _HYS_EN` 来开启或关闭 GPIO n 的迟滞功能。

5.11 GPIO 管脚供电和电源管理

5.11.1 GPIO 管脚供电

GPIO 管脚供电请参考《ESP32-C5 技术规格书》中管脚定义章节。所有管脚均可用于将芯片从 Light-sleep 中唤醒，但仅有 VDDPST1 域中的管脚 (GPIO0~GPIO6) 可用于将芯片从 Deep-sleep 唤醒。

5.11.2 电源管理

ESP32-C5 的管脚可由如下不同的电源域供电。

- VDDPST1: LP GPIO 管脚的输入电源
- VDDPST2: HP GPIO 管脚的输入电源

5.12 HP 外设信号列表

表 5.12-1 列出了所有经由 HP GPIO 交换矩阵的外设输入输出信号。

请注意 `GPIO_FUNC $n$ _OE_SEL` 位的配置：

- `GPIO_FUNC $n$ _OE_SEL` = 1，则寄存器 `GPIO_ENABLE_REG` 中的相应位 n 将用于控制信号输出使能。

- `GPIO_ENABLE_REG[n]` = 0: 输出关闭。
- `GPIO_ENABLE_REG[n]` = 1: 输出使能。
- `GPIO_FUNCn_OE_SEL` = 0, 则输出信号的使能由外设控制, 例如表 5.12-1 中 “`GPIO_FUNCn_OE_SEL` = 0 时输出信号的输出使能信号” 一栏的 `SPIQ_oe`。注意, 使能信号 `SPIQ_oe` 可设置为 1 (1'd1) 或 0 (1'd0), 具体由外设的配置决定。如果 “`GPIO_FUNCn_OE_SEL` = 0 时输出信号的输出使能信号” 一栏中为 1'd1, 则表示输出信号默认始终使能。

说明:

信号连续编号, 但并非所有信号均有效。

- 表 5.12-1 “输入信号” 一栏中有名字的信号均为有效输入信号。
- 表 5.12-1 “输出信号” 一栏中有名字的信号均为有效输出信号。

表 5.12-1. GPIO 交换矩阵外设信号

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC _n _OEN_SEL = 0 时 输出信号的输出使能信号	信号可经由 IO MUX 直接输出
0	—	-	-	ledc_ls_sig_out0	1'd1	no
1	—	—	—	ledc_ls_sig_out1	1'd1	no
2	—	—	—	ledc_ls_sig_out2	1'd1	no
3	—	—	—	ledc_ls_sig_out3	1'd1	no
4	—	—	—	ledc_ls_sig_out4	1'd1	no
5	—	—	—	ledc_ls_sig_out5	1'd1	no
6	UORXD_in	0	yes	UOTXD_out	1'd1	yes
7	UOCTS_in	0	no	UORTS_out	1'd1	no
8	UODSR_in	0	no	UODTR_out	1'd1	no
9	U1RXD_in	1	no	U1TXD_out	1'd1	no
10	U1CTS_in	0	no	U1RTS_out	1'd1	no
11	U1DSR_in	0	no	U1DTR_out	1'd1	no
12	I2S_MCLK_in	0	no	I2S_MCLK_out	1'd1	no
13	I2SO_BCK_in	0	no	I2SO_BCK_out	1'd1	no
14	I2SO_WS_in	0	no	I2SO_WS_out	1'd1	no
15	I2SI_SD_in	0	no	I2SO_SD_out	1'd1	no
16	I2SI_BCK_in	0	no	I2SI_BCK_out	1'd1	no
17	I2SI_WS_in	0	no	I2SI_WS_out	1'd1	no
18	—	—	—	I2SO_SD1_out	1'd1	no
19	—	—	—	—	—	—
20	—	—	—	—	—	—
21	—	—	—	—	—	—
22	—	—	—	—	—	—
23	—	—	—	—	—	—
24	—	—	—	—	—	—
25	—	—	—	—	—	—
26	—	—	—	—	—	—
27	cpu_gpio_in0	0	no	cpu_gpio_out0	cpu_gpio_out_oen0	no
28	cpu_gpio_in1	0	no	cpu_gpio_out1	cpu_gpio_out_oen1	no

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC n _OEN_SEL = 0 时 输出信号的输出使能信号	信号可经由 IO MUX 直接输出
29	cpu_gpio_in2	0	no	cpu_gpio_out2	cpu_gpio_out_oen2	no
30	cpu_gpio_in3	0	no	cpu_gpio_out3	cpu_gpio_out_oen3	no
31	cpu_gpio_in4	0	no	cpu_gpio_out4	cpu_gpio_out_oen4	no
32	cpu_gpio_in5	0	no	cpu_gpio_out5	cpu_gpio_out_oen5	no
33	cpu_gpio_in6	0	no	cpu_gpio_out6	cpu_gpio_out_oen6	no
34	cpu_gpio_in7	0	no	cpu_gpio_out7	cpu_gpio_out_oen7	no
35	usb_jtag_tdo_bridge	0	no	—	—	—
36	—	—	—	—	—	—
37	—	—	—	—	—	—
38	—	—	—	—	—	—
39	—	—	—	—	—	—
40	—	—	—	—	—	—
41	—	—	—	—	—	—
42	—	—	—	—	—	—
43	—	—	—	—	—	—
44	—	—	—	—	—	—
45	—	—	—	—	—	—
46	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
47	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
48	parl_rx_data0	0	no	parl_tx_data0	1'd1	no
49	parl_rx_data1	0	no	parl_tx_data1	1'd1	no
50	parl_rx_data2	0	no	parl_tx_data2	1'd1	no
51	parl_rx_data3	0	no	parl_tx_data3	1'd1	no
52	parl_rx_data4	0	no	parl_tx_data4	1'd1	no
53	parl_rx_data5	0	no	parl_tx_data5	1'd1	no
54	parl_rx_data6	0	no	parl_tx_data6	1'd1	no
55	parl_rx_data7	0	no	parl_tx_data7	1'd1	no
56	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe	yes
57	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe	yes
58	FSPID_in	0	yes	FSPID_out	FSPID_oe	yes

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC _n _OEN_SEL = 0 时 输出信号的输出使能信号	信号可经由 IO MUX 直接输出
59	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe	yes
60	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe	yes
61	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe	yes
62	parl_rx_clk_in	0	no	parl_rx_clk_out	1'd1	no
63	parl_tx_clk_in	0	no	parl_tx_clk_out	1'd1	no
64	rmt_sig_in0	0	no	rmt_sig_out0	1'd1	no
65	rmt_sig_in1	0	no	rmt_sig_out1	1'd1	no
66	twai0_rx	1	no	twai0_tx	1'd1	no
67	—	—	—	twai0_bus_off_on	1'd1	no
68	—	—	—	twai0_clkout	1'd1	no
69	—	—	—	twai0_standby	1'd1	no
70	twai1_rx	1	no	twai1_tx	1'd1	no
71	—	—	—	twai1_bus_off_on	1'd1	no
72	—	—	—	twai1_clkout	1'd1	no
73	—	—	—	twai1_standby	1'd1	no
74	—	—	—	—	—	—
75	—	—	—	—	—	—
76	pcnt_rst_in0	0	no	gpio_sd0_out	1'd1	no
77	pcnt_rst_in1	0	no	gpio_sd1_out	1'd1	no
78	pcnt_rst_in2	0	no	gpio_sd2_out	1'd1	no
79	pcnt_rst_in3	0	no	gpio_sd3_out	1'd1	no
80	pwm0_sync0_in	0	no	pwm0_out0a	1'd1	no
81	pwm0_sync1_in	0	no	pwm0_out0b	1'd1	no
82	pwm0_sync2_in	0	no	pwm0_out1a	1'd1	no
83	pwm0_f0_in	0	no	pwm0_out1b	1'd1	no
84	pwm0_f1_in	0	no	pwm0_out2a	1'd1	no
85	pwm0_f2_in	0	no	pwm0_out2b	1'd1	no
86	pwm0_cap0_in	0	no	parl_tx_cs_o	1'd1	no
87	pwm0_cap1_in	0	no	—	—	—
88	pwm0_cap2_in	0	no	—	—	—

信号索引	输入信号	默认值	信号可由 IO MUX 直接输入	输出信号	GPIO_FUNC _n _OEN_SEL = 0 时 输出信号的输出使能信号	信号可由 IO MUX 直接输出
89	—	—	—	—	—	—
90	—	—	—	—	—	—
91	—	—	—	—	—	—
92	—	—	—	—	—	—
93	—	—	—	—	—	—
94	—	—	—	—	—	—
95	—	—	—	—	—	—
96	—	—	—	—	—	—
97	sig_in_func_97	0	no	sig_in_func97	1'd1	no
98	sig_in_func_98	0	no	sig_in_func98	1'd1	no
99	sig_in_func_99	0	no	sig_in_func99	1'd1	no
100	sig_in_func_100	0	no	sig_in_func100	1'd1	no
101	pcnt_sig_ch0_in0	0	no	FSPICS1_out	FSPICS1_oe	yes
102	pcnt_sig_ch1_in0	0	no	FSPICS2_out	FSPICS2_oe	yes
103	pcnt_ctrl_ch0_in0	0	no	FSPICS3_out	FSPICS3_oe	yes
104	pcnt_ctrl_ch1_in0	0	no	FSPICS4_out	FSPICS4_oe	yes
105	pcnt_sig_ch0_in1	0	no	FSPICS5_out	FSPICS5_oe	yes
106	pcnt_sig_ch1_in1	0	no	—	—	—
107	pcnt_ctrl_ch0_in1	0	no	—	—	—
108	pcnt_ctrl_ch1_in1	0	no	—	—	—
109	pcnt_sig_ch0_in2	0	no	—	—	—
110	pcnt_sig_ch1_in2	0	no	—	—	—
111	pcnt_ctrl_ch0_in2	0	no	—	—	—
112	pcnt_ctrl_ch1_in2	0	no	—	—	—
113	pcnt_sig_ch0_in3	0	no	—	—	—
114	pcnt_sig_ch1_in3	0	no	—	—	—
115	pcnt_ctrl_ch0_in3	0	no	—	—	—
116	pcnt_ctrl_ch1_in3	0	no	—	—	—
117 ~ 255	—	—	—	—	—	—

5.13 HP IO MUX 管脚功能列表

表 5.13-1 列出了每个 GPIO 管脚的 HP IO MUX 功能及其默认状态。

表 5.13-1. HP IO MUX 管脚功能

GPIO No.	名称	功能 0	功能 1	功能 2	功能 3	驱动强度	复位	说明
0	XTAL_32K_P	GPIO0	GPIO0	—	—	2	0	R
1	XTAL_32K_N	GPIO1	GPIO1	—	—	2	0	R
2	MTMS	MTMS	GPIO2	FSPIQ	—	2	1	R
3	MTDI	MTDI	GPIO3	—	—	2	1	R
4	MTCK	MTCK	GPIO4	FSPIHD	—	2	1*	R
5	MTDO	MTDO	GPIO5	FSPIWP	—	2	1	R
6	GPIO6	GPIO6	GPIO6	FSPICLK	—	2	0	R
7	GPIO7	SDIO_DATA1	GPIO7	FSPID	—	2	1	
8	GPIO8	SDIO_DATA0	GPIO8	—	—	2	0	—
9	GPIO9	SDIO_CLK	GPIO9	—	—	2	0	—
10	GPIO10	SDIO_CMD	GPIO10	FSPICS0	—	2	0	—
11	UOTXD	UOTXD	GPIO11	—	—	2	4	—
12	UORXD	UORXD	GPIO12	—	—	2	1	—
13	GPIO13	SDIO_DATA3	GPIO13	—	—	3	1	USB
14	GPIO14	SDIO_DATA2	GPIO14	—	—	3	3*	USB
23	GPIO23	GPIO23	GPIO23	—	—	2	0	—
24	GPIO24	GPIO24	GPIO24	—	—	2	0	—
25	GPIO25	GPIO25	GPIO25	—	—	2	1	—
26	GPIO26	GPIO26	GPIO26	—	—	2	1	—
27	GPIO27	GPIO27	GPIO27	—	—	2	3	—
28	GPIO28	GPIO28	GPIO28	—	—	2	3	—

注意：
HP IO MUX 中，没有用到的管脚需要配置成 GPIO 功能。

驱动强度

“驱动强度”一栏所示为每个管脚复位后的默认驱动强度。

- 0 - 驱动电流 = ~5 mA
- 1 - 驱动电流 = ~10 mA
- 2 - 驱动电流 = ~20 mA
- 3 - 驱动电流 = ~40 mA

复位

“复位”一栏所示为每个管脚复位后的默认配置。

- 0 - IE = 0 (输入关闭)
- 1 - IE = 1 (输入使能)
- 3 - IE = 1, WPU = 1 (输入使能, 上拉电阻使能)
- 4 - IE = 1, OE = 1 (输入使能, 输出使能)
- 1* - 如果 EFUSE_DIS_PAD_JTAG = 1, 则 MTCK 管脚复位后浮空, 即 IE = 1。如果 EFUSE_DIS_PAD_JTAG = 0, 则 MTCK 管脚连接内部上拉电阻, 即 IE = 1, WPU = 1。
- 3* - IE = 1, WPU = 0, GPIO14 的 USB 上拉默认值为 1, 因此, 其上拉电阻使能, 具体见以下说明。

说明

- R - LP 管脚, 部分具有模拟功能, 见表 5.15-1。
- USB - USB 上拉电阻使能
 - USB 管脚 (GPIO13 和 GPIO14) 默认开启 USB 功能, 此时管脚是否上拉由 USB 是否上拉决定。USB 上拉由 USB_SERIAL_JTAG_DP/DM_PULLUP 控制, USB 上拉电阻的具体阻值可通过 USB_SERIAL_JTAG_PULLUP_VALUE 位控制, 详见 [《ESP32-C5 技术参考手册》](#) > 章节 USB 串口/JTAG 控制器。
 - USB 管脚关闭 USB 功能时, 用作普通 GPIO, 默认禁用管脚内部弱上/下拉电阻, 可通过 IO_MUX_GPIO_n_MCU_WPU/WPD 配置。

5.14 LP IO MUX 管脚功能列表

表 5.14-1 列出了每个 GPIO 管脚的 LP IO MUX 功能。GPIO 管脚默认由 HP IO MUX 控制, 因此表 5.14-1 仅列出了管脚的 LP IO MUX 功能。

表 5.14-1. LP IO MUX 管脚功能

LP IO 名称	功能 0	功能 1	功能 2	功能 3
GPIO0	LP_UART_DTRN	LP_GPIO0	—	—
GPIO1	LP_UART_DSRN	LP_GPIO1	—	—
GPIO2	LP_UART_RTSN	LP_GPIO2	—	LP_I2C_SDA
GPIO3	LP_UART_CTSN	LP_GPIO3	—	LP_I2C_SCL
GPIO4	LP_UART_RXD_PAD	LP_GPIO4	—	—
GPIO5	LP_UART_TXD_PAD	LP_GPIO5	—	—

LP IO 名称	功能 0	功能 1	功能 2	功能 3
GPIO6	—	LP_GPIO6	—	—

注意：

- LP_UART 的硬件流控功能无法和 LP_I2C 同时使用。
- LP IO MUX 中，没有用到的管脚需要配置成 LP_GPIO 功能。
- 由于 LP IO MUX 功能通过使用 LP GPIO 编号的 LP GPIO 寄存器配置，上表列出的是 LP GPIO 名称。

5.15 GPIO 管脚模拟功能列表

表 5.15-1 列出了 GPIO 管脚及其相应的模拟功能。

表 5.15-1. GPIO 管脚模拟功能

名称	模拟功能 0	模拟功能 1
XTAL_32K_P	XTAL_32K_P	
XTAL_32K_N	XTAL_32K_N	ADC1_CH0
MTMS	—	ADC1_CH1
MTDI	—	ADC1_CH2
MTCK	—	ADC1_CH3
MTDO	—	ADC1_CH4
GPIO6	—	ADC1_CH5
GPIO8	ZCD0*	—
GPIO9	ZCD1*	—
GPIO13	USB_D-	—
GPIO14	USB_D+	—

* ZCD0 和 ZCD1 是模拟 PAD 电压比较功能，见章节 37 模拟电压比较器。

5.16 事件任务矩阵 (ETM) 功能

在 ESP32-C5 中，GPIO 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GPIO 的 ETM 任务，或者通过 GPIO 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与 GPIO 相关的 ETM 任务和 ETM 事件。

GPIO ETM 有八个任务通道 x (0~7)。每个任务通道可接收的 ETM 任务有：

- GPIO_TASK_CH x _SET：触发时 GPIO 变为高电平。
- GPIO_TASK_CH x _CLEAR：触发时 GPIO 变为低电平。
- GPIO_TASK_CH x _TOGGLE：触发时 GPIO 电平翻转。

将任务通道 x 配置到 GPIO y 的示例如下：

- 配置 IO_MUX_GPIO y _MCU_SEL 的值为 1，使得 GPIO y 选择表 5.13-1 中的功能 1。

- 配置 `GPIO_ENABLE_REG[y]` 的值为 1。
- 配置 `GPIO_EXT_ETM_TASK_GPIOy_SEL` 为 `x`。
- 置位 `GPIO_EXT_ETM_TASK_GPIOy_EN`，使能 `GPIOy` 受到任务通道 `x` 的控制。

说明：

- 一个任务通道可被一个或多个 GPIO 选择。
- 如果 `GPIOy` 选择的任务通道 `x` 其信号 (`GPIO_TASK_CHx_SET`、`GPIO_TASK_CHx_CLEAR` 和 `GPIO_TASK_CHx_TOGGLE`) 中有两个或三个同时有效，则 `GPIO_TASK_CHx_SET` 优先级最高，`GPIO_TASK_CHx_CLEAR` 优先级次之，`GPIO_TASK_CHx_TOGGLE` 优先级最低。
- `GPIOy` 受到 ETM 任务通道控制时，`GPIO_OUT_REG`、`GPIO_FUNCn_OUT_INV_SEL` 和 `GPIO_FUNCn_OUT_SEL` 的值可能会被硬件修改。在 GPIO 退出 ETM 任务通道控制后，建议对上述寄存器重新进行配置。

GPIO 有八个事件通道，每个事件通道可以生成的 ETM 事件有：

- `GPIO_EVT_CHx_RISE_EDGE`：表示对应 GPIO 滤波器（见图 5.3-1）输出信号出现上升沿。
- `GPIO_EVT_CHx_FALL_EDGE`：表示对应 GPIO 滤波器（见图 5.3-1）输出信号出现下降沿。
- `GPIO_EVT_CHx_ANY_EDGE`：表示对应 GPIO 滤波器（见图 5.3-1）输出信号发生翻转。

事件通道具体配置如下：

- 置位 `GPIO_EXT_ETM_CHx_EVENT_EN` 使能事件通道 `x` (0~7)。
- 配置 `GPIO_EXT_ETM_CHx_EVENT_SEL` 为 `y` (0~14、23~28)，即选择 21 个 GPIO 中的一个 GPIO。

说明：

一个 GPIO 可被一个或多个事件通道选择。

GPIO 的 ETM 事件可以用来触发 GPIO 的 ETM 任务。例如，事件通道 0 选择 GPIO0，GPIO1 选择任务通道 0，并且 `GPIO_EVT_CH0_RISE_EDGE` 事件用于触发 `GPIO_TASK_CH0_TOGGLE` 任务。当通过 GPIO0 向芯片输入一个方波信号时，芯片通过 GPIO1 输出一个二分频的方波信号。

5.17 中断

ESP32-C5 的 HP IO MUX 和 HP GPIO 交换矩阵可以生成以下中断信号并发送给中断矩阵。

- `GPIO_EXT_REG_INT`
- `GPIO_PROCPU_INT`

HP IO MUX 和 HP GPIO 交换矩阵有多个中断源可生成以上中断信号。表 5.17-1 列出了 HP IO MUX 和 HP GPIO 交换矩阵内部的中断源及其触发条件。

表 5.17-1. HP IO MUX 和 HP GPIO 交换矩阵内部中断源

内部中断源	触发条件	中断信号
<code>GPIO_EXT_COMP_O_ALL_INT</code>	参考章节 37 模拟电压比较器	<code>GPIO_EXT_REG_INT</code>
<code>GPIO_EXT_COMP_O_NEG_INT</code>	参考章节 37 模拟电压比较器	<code>GPIO_EXT_REG_INT</code>

GPIO_EXT_COMP_O_POS_INT	参考章节 37 模拟电压比较器	GPIO_EXT_REG_INT
GPIO_REG_INTR	GPIO_STATUS_INTERRUPT[n] & GPIO_PINn_INT_ENA[0] (n: 0~28)	GPIO_PROCPU_INT

ESP32-C5 的 LP IO MUX 和 LP GPIO 交换矩阵可以生成以下中断信号，生成的中断信号只会发送给 LP_CORE。

- LP_GPIO_INTR

表 5.17-2 列出了 LP IO MUX 和 LP GPIO 交换矩阵内部的中断源及其触发条件。

表 5.17-2. LP IO MUX 和 LP GPIO 交换矩阵内部中断源

内部中断源	触发条件	中断信号
LP_GPIO_REG_INTR	LP_GPIO_STATUS_INTERRUPT[n] (n: 0~6)	LP_GPIO_INTR

5.18 寄存器列表

5.18.1 HP GPIO 交换矩阵寄存器列表

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。请查看章节寄存器的访问类型，了解访问列缩写的含义。

名称	描述	地址	访问
配置寄存器			
GPIO_STRAP_REG	Strapping 管脚寄存器	0x0000	RO
GPIO_OUT_REG	GPIO 输出寄存器	0x0004	R/ W/ SC/ WTC
GPIO_OUT_WITS_REG	GPIO 输出置位寄存器	0x0008	WT
GPIO_OUT_WITC_REG	GPIO 输出清除寄存器	0x000C	WT
GPIO_ENABLE_REG	GPIO 输出使能寄存器	0x0034	R/ W/ WTC
GPIO_ENABLE_WITS_REG	GPIO 输出使能置位寄存器	0x0038	WT
GPIO_ENABLE_WITC_REG	GPIO 输出使能清除寄存器	0x003C	WT
GPIO_IN_REG	GPIO 输入寄存器	0x0064	RO
中断状态寄存器			
GPIO_PROCPU_INT_REG	GPIO 的 CPU 中断状态寄存器	0x00A4	RO
GPIO_STATUS_REG	GPIO 中断状态寄存器	0x0074	R/ W/ WTC
GPIO_STATUS_WITS_REG	GPIO 中断状态置位寄存器	0x0078	WT
GPIO_STATUS_WITC_REG	GPIO 中断状态清除寄存器	0x007C	WT
GPIO_STATUS_NEXT_REG	GPIO 中断源寄存器	0x00B4	RO
管脚配置寄存器			
GPIO_PINO_REG	GPIO0 管脚配置寄存器	0x00C4	R/W

名称	描述	地址	访问
GPIO_PIN1_REG	GPIO1 管脚配置寄存器	0x00C8	R/W
GPIO_PIN2_REG	GPIO2 管脚配置寄存器	0x00CC	R/W
GPIO_PIN3_REG	GPIO3 管脚配置寄存器	0x00D0	R/W
GPIO_PIN4_REG	GPIO4 管脚配置寄存器	0x00D4	R/W
GPIO_PIN5_REG	GPIO5 管脚配置寄存器	0x00D8	R/W
GPIO_PIN6_REG	GPIO6 管脚配置寄存器	0x00DC	R/W
GPIO_PIN7_REG	GPIO7 管脚配置寄存器	0x00E0	R/W
GPIO_PIN8_REG	GPIO8 管脚配置寄存器	0x00E4	R/W
GPIO_PIN9_REG	GPIO9 管脚配置寄存器	0x00E8	R/W
GPIO_PIN10_REG	GPIO10 管脚配置寄存器	0x00EC	R/W
GPIO_PIN11_REG	GPIO11 管脚配置寄存器	0x00F0	R/W
GPIO_PIN12_REG	GPIO12 管脚配置寄存器	0x00F4	R/W
GPIO_PIN13_REG	GPIO13 管脚配置寄存器	0x00F8	R/W
GPIO_PIN14_REG	GPIO14 管脚配置寄存器	0x00FC	R/W
GPIO_PIN15_REG	GPIO15 管脚配置寄存器	0x0100	R/W
GPIO_PIN16_REG	GPIO16 管脚配置寄存器	0x0104	R/W
GPIO_PIN17_REG	GPIO17 管脚配置寄存器	0x0108	R/W
GPIO_PIN18_REG	GPIO18 管脚配置寄存器	0x010C	R/W
GPIO_PIN19_REG	GPIO19 管脚配置寄存器	0x0110	R/W
GPIO_PIN20_REG	GPIO20 管脚配置寄存器	0x0114	R/W
GPIO_PIN21_REG	GPIO21 管脚配置寄存器	0x0118	R/W
GPIO_PIN22_REG	GPIO22 管脚配置寄存器	0x011C	R/W
GPIO_PIN23_REG	GPIO23 管脚配置寄存器	0x0120	R/W
GPIO_PIN24_REG	GPIO24 管脚配置寄存器	0x0124	R/W
GPIO_PIN25_REG	GPIO25 管脚配置寄存器	0x0128	R/W
GPIO_PIN26_REG	GPIO26 管脚配置寄存器	0x012C	R/W
GPIO_PIN27_REG	GPIO27 管脚配置寄存器	0x0130	R/W
GPIO_PIN28_REG	GPIO28 管脚配置寄存器	0x0134	R/W
输入配置寄存器			
GPIO_FUNC6_IN_SEL_CFG_REG	输入信号 6 配置寄存器	0x02DC	R/W
GPIO_FUNC7_IN_SEL_CFG_REG	输入信号 7 配置寄存器	0x02E0	R/W
GPIO_FUNC8_IN_SEL_CFG_REG	输入信号 8 配置寄存器	0x02E4	R/W
GPIO_FUNC9_IN_SEL_CFG_REG	输入信号 9 配置寄存器	0x02E8	R/W
GPIO_FUNC10_IN_SEL_CFG_REG	输入信号 10 配置寄存器	0x02EC	R/W
GPIO_FUNC11_IN_SEL_CFG_REG	输入信号 11 配置寄存器	0x02F0	R/W
GPIO_FUNC12_IN_SEL_CFG_REG	输入信号 12 配置寄存器	0x02F4	R/W
GPIO_FUNC13_IN_SEL_CFG_REG	输入信号 13 配置寄存器	0x02F8	R/W
GPIO_FUNC14_IN_SEL_CFG_REG	输入信号 14 配置寄存器	0x02FC	R/W
GPIO_FUNC15_IN_SEL_CFG_REG	输入信号 15 配置寄存器	0x0300	R/W
GPIO_FUNC16_IN_SEL_CFG_REG	输入信号 16 配置寄存器	0x0304	R/W
GPIO_FUNC17_IN_SEL_CFG_REG	输入信号 17 配置寄存器	0x0308	R/W
GPIO_FUNC27_IN_SEL_CFG_REG	输入信号 27 配置寄存器	0x0330	R/W
GPIO_FUNC28_IN_SEL_CFG_REG	输入信号 28 配置寄存器	0x0334	R/W

名称	描述	地址	访问
GPIO_FUNC29_IN_SEL_CFG_REG	输入信号 29 配置寄存器	0x0338	R/W
GPIO_FUNC30_IN_SEL_CFG_REG	输入信号 30 配置寄存器	0x033C	R/W
GPIO_FUNC31_IN_SEL_CFG_REG	输入信号 31 配置寄存器	0x0340	R/W
GPIO_FUNC32_IN_SEL_CFG_REG	输入信号 32 配置寄存器	0x0344	R/W
GPIO_FUNC33_IN_SEL_CFG_REG	输入信号 33 配置寄存器	0x0348	R/W
GPIO_FUNC34_IN_SEL_CFG_REG	输入信号 34 配置寄存器	0x034C	R/W
GPIO_FUNC35_IN_SEL_CFG_REG	输入信号 35 配置寄存器	0x0350	R/W
GPIO_FUNC46_IN_SEL_CFG_REG	输入信号 46 配置寄存器	0x037C	R/W
GPIO_FUNC47_IN_SEL_CFG_REG	输入信号 47 配置寄存器	0x0380	R/W
GPIO_FUNC48_IN_SEL_CFG_REG	输入信号 48 配置寄存器	0x0384	R/W
GPIO_FUNC49_IN_SEL_CFG_REG	输入信号 49 配置寄存器	0x0388	R/W
GPIO_FUNC50_IN_SEL_CFG_REG	输入信号 50 配置寄存器	0x038C	R/W
GPIO_FUNC51_IN_SEL_CFG_REG	输入信号 51 配置寄存器	0x0390	R/W
GPIO_FUNC52_IN_SEL_CFG_REG	输入信号 52 配置寄存器	0x0394	R/W
GPIO_FUNC53_IN_SEL_CFG_REG	输入信号 53 配置寄存器	0x0398	R/W
GPIO_FUNC54_IN_SEL_CFG_REG	输入信号 54 配置寄存器	0x039C	R/W
GPIO_FUNC55_IN_SEL_CFG_REG	输入信号 55 配置寄存器	0x03A0	R/W
GPIO_FUNC56_IN_SEL_CFG_REG	输入信号 56 配置寄存器	0x03A4	R/W
GPIO_FUNC57_IN_SEL_CFG_REG	输入信号 57 配置寄存器	0x03A8	R/W
GPIO_FUNC58_IN_SEL_CFG_REG	输入信号 58 配置寄存器	0x03AC	R/W
GPIO_FUNC59_IN_SEL_CFG_REG	输入信号 59 配置寄存器	0x03B0	R/W
GPIO_FUNC60_IN_SEL_CFG_REG	输入信号 60 配置寄存器	0x03B4	R/W
GPIO_FUNC61_IN_SEL_CFG_REG	输入信号 61 配置寄存器	0x03B8	R/W
GPIO_FUNC62_IN_SEL_CFG_REG	输入信号 62 配置寄存器	0x03BC	R/W
GPIO_FUNC63_IN_SEL_CFG_REG	输入信号 63 配置寄存器	0x03C0	R/W
GPIO_FUNC64_IN_SEL_CFG_REG	输入信号 64 配置寄存器	0x03C4	R/W
GPIO_FUNC65_IN_SEL_CFG_REG	输入信号 65 配置寄存器	0x03C8	R/W
GPIO_FUNC66_IN_SEL_CFG_REG	输入信号 66 配置寄存器	0x03CC	R/W
GPIO_FUNC70_IN_SEL_CFG_REG	输入信号 70 配置寄存器	0x03DC	R/W
GPIO_FUNC74_IN_SEL_CFG_REG	输入信号 74 配置寄存器	0x03EC	R/W
GPIO_FUNC75_IN_SEL_CFG_REG	输入信号 75 配置寄存器	0x03F0	R/W
GPIO_FUNC76_IN_SEL_CFG_REG	输入信号 76 配置寄存器	0x03F4	R/W
GPIO_FUNC77_IN_SEL_CFG_REG	输入信号 77 配置寄存器	0x03F8	R/W
GPIO_FUNC78_IN_SEL_CFG_REG	输入信号 78 配置寄存器	0x03FC	R/W
GPIO_FUNC79_IN_SEL_CFG_REG	输入信号 79 配置寄存器	0x0400	R/W
GPIO_FUNC80_IN_SEL_CFG_REG	输入信号 80 配置寄存器	0x0404	R/W
GPIO_FUNC81_IN_SEL_CFG_REG	输入信号 81 配置寄存器	0x0408	R/W
GPIO_FUNC82_IN_SEL_CFG_REG	输入信号 82 配置寄存器	0x040C	R/W
GPIO_FUNC83_IN_SEL_CFG_REG	输入信号 83 配置寄存器	0x0410	R/W
GPIO_FUNC84_IN_SEL_CFG_REG	输入信号 84 配置寄存器	0x0414	R/W
GPIO_FUNC85_IN_SEL_CFG_REG	输入信号 85 配置寄存器	0x0418	R/W
GPIO_FUNC86_IN_SEL_CFG_REG	输入信号 86 配置寄存器	0x041C	R/W
GPIO_FUNC87_IN_SEL_CFG_REG	输入信号 87 配置寄存器	0x0420	R/W

名称	描述	地址	访问
GPIO_FUNC88_IN_SEL_CFG_REG	输入信号 88 配置寄存器	0x0424	R/W
GPIO_FUNC89_IN_SEL_CFG_REG	输入信号 89 配置寄存器	0x0428	R/W
GPIO_FUNC90_IN_SEL_CFG_REG	输入信号 90 配置寄存器	0x042C	R/W
GPIO_FUNC91_IN_SEL_CFG_REG	输入信号 91 配置寄存器	0x0430	R/W
GPIO_FUNC92_IN_SEL_CFG_REG	输入信号 92 配置寄存器	0x0434	R/W
GPIO_FUNC97_IN_SEL_CFG_REG	输入信号 97 配置寄存器	0x0448	R/W
GPIO_FUNC98_IN_SEL_CFG_REG	输入信号 98 配置寄存器	0x044C	R/W
GPIO_FUNC99_IN_SEL_CFG_REG	输入信号 99 配置寄存器	0x0450	R/W
GPIO_FUNC100_IN_SEL_CFG_REG	输入信号 100 配置寄存器	0x0454	R/W
GPIO_FUNC101_IN_SEL_CFG_REG	输入信号 101 配置寄存器	0x0458	R/W
GPIO_FUNC102_IN_SEL_CFG_REG	输入信号 102 配置寄存器	0x045C	R/W
GPIO_FUNC103_IN_SEL_CFG_REG	输入信号 103 配置寄存器	0x0460	R/W
GPIO_FUNC104_IN_SEL_CFG_REG	输入信号 104 配置寄存器	0x0464	R/W
GPIO_FUNC105_IN_SEL_CFG_REG	输入信号 105 配置寄存器	0x0468	R/W
GPIO_FUNC106_IN_SEL_CFG_REG	输入信号 106 配置寄存器	0x046C	R/W
GPIO_FUNC107_IN_SEL_CFG_REG	输入信号 107 配置寄存器	0x0470	R/W
GPIO_FUNC108_IN_SEL_CFG_REG	输入信号 108 配置寄存器	0x0474	R/W
GPIO_FUNC109_IN_SEL_CFG_REG	输入信号 109 配置寄存器	0x0478	R/W
GPIO_FUNC110_IN_SEL_CFG_REG	输入信号 110 配置寄存器	0x047C	R/W
GPIO_FUNC111_IN_SEL_CFG_REG	输入信号 111 配置寄存器	0x0480	R/W
GPIO_FUNC112_IN_SEL_CFG_REG	输入信号 112 配置寄存器	0x0484	R/W
GPIO_FUNC113_IN_SEL_CFG_REG	输入信号 113 配置寄存器	0x0488	R/W
GPIO_FUNC114_IN_SEL_CFG_REG	输入信号 114 配置寄存器	0x048C	R/W
GPIO_FUNC115_IN_SEL_CFG_REG	输入信号 115 配置寄存器	0x0490	R/W
GPIO_FUNC116_IN_SEL_CFG_REG	输入信号 116 配置寄存器	0x0494	R/W
输出配置寄存器			
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 输出配置寄存器	0x0AC4	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 输出配置寄存器	0x0AC8	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 输出配置寄存器	0x0ACC	varies
GPIO_FUNC3_OUT_SEL_CFG_REG	GPIO3 输出配置寄存器	0x0AD0	varies
GPIO_FUNC4_OUT_SEL_CFG_REG	GPIO4 输出配置寄存器	0x0AD4	varies
GPIO_FUNC5_OUT_SEL_CFG_REG	GPIO5 输出配置寄存器	0x0AD8	varies
GPIO_FUNC6_OUT_SEL_CFG_REG	GPIO6 输出配置寄存器	0x0ADC	varies
GPIO_FUNC7_OUT_SEL_CFG_REG	GPIO7 输出配置寄存器	0x0AE0	varies
GPIO_FUNC8_OUT_SEL_CFG_REG	GPIO8 输出配置寄存器	0x0AE4	varies
GPIO_FUNC9_OUT_SEL_CFG_REG	GPIO9 输出配置寄存器	0x0AE8	varies
GPIO_FUNC10_OUT_SEL_CFG_REG	GPIO10 输出配置寄存器	0x0AEC	varies
GPIO_FUNC11_OUT_SEL_CFG_REG	GPIO11 输出配置寄存器	0x0AF0	varies
GPIO_FUNC12_OUT_SEL_CFG_REG	GPIO12 输出配置寄存器	0x0AF4	varies
GPIO_FUNC13_OUT_SEL_CFG_REG	GPIO13 输出配置寄存器	0x0AF8	varies
GPIO_FUNC14_OUT_SEL_CFG_REG	GPIO14 输出配置寄存器	0x0AFC	varies
GPIO_FUNC23_OUT_SEL_CFG_REG	GPIO23 输出配置寄存器	0x0B20	varies
GPIO_FUNC24_OUT_SEL_CFG_REG	GPIO24 输出配置寄存器	0x0B24	varies

名称	描述	地址	访问
GPIO_FUNC25_OUT_SEL_CFG_REG	GPIO25 输出配置寄存器	0x0B28	varies
GPIO_FUNC26_OUT_SEL_CFG_REG	GPIO26 输出配置寄存器	0x0B2C	varies
GPIO_FUNC27_OUT_SEL_CFG_REG	GPIO27 输出配置寄存器	0x0B30	varies
GPIO_FUNC28_OUT_SEL_CFG_REG	GPIO28 输出配置寄存器	0x0B34	varies
时钟门控寄存器			
GPIO_CLOCK_GATE_REG	GPIO 时钟门控寄存器	0x0DF8	R/W
版本寄存器			
GPIO_DATE_REG	版本控制寄存器	0x0DFC	R/W

5.18.2 HP IO MUX 寄存器列表

本小节的所有地址均为相对于 HP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节寄存器的访问类型，了解访问列缩写的含义。

名称	描述	地址	访问
配置寄存器			
IO_MUX_GPIO0_REG	GPIO0 的 HP IO MUX 配置寄存器	0x0000	R/W
IO_MUX_GPIO1_REG	GPIO1 的 HP IO MUX 配置寄存器	0x0004	R/W
IO_MUX_GPIO2_REG	GPIO2 的 HP IO MUX 配置寄存器	0x0008	R/W
IO_MUX_GPIO3_REG	GPIO3 的 HP IO MUX 配置寄存器	0x000C	R/W
IO_MUX_GPIO4_REG	GPIO4 的 HP IO MUX 配置寄存器	0x0010	R/W
IO_MUX_GPIO5_REG	GPIO5 的 HP IO MUX 配置寄存器	0x0014	R/W
IO_MUX_GPIO6_REG	GPIO6 的 HP IO MUX 配置寄存器	0x0018	R/W
IO_MUX_GPIO7_REG	GPIO7 的 HP IO MUX 配置寄存器	0x001C	R/W
IO_MUX_GPIO8_REG	GPIO8 的 HP IO MUX 配置寄存器	0x0020	R/W
IO_MUX_GPIO9_REG	GPIO9 的 HP IO MUX 配置寄存器	0x0024	R/W
IO_MUX_GPIO10_REG	GPIO10 的 HP IO MUX 配置寄存器	0x0028	R/W
IO_MUX_GPIO11_REG	GPIO11 的 HP IO MUX 配置寄存器	0x002C	R/W
IO_MUX_GPIO12_REG	GPIO12 的 HP IO MUX 配置寄存器	0x0030	R/W
IO_MUX_GPIO13_REG	GPIO13 的 HP IO MUX 配置寄存器	0x0034	R/W
IO_MUX_GPIO14_REG	GPIO14 的 HP IO MUX 配置寄存器	0x0038	R/W
IO_MUX_GPIO23_REG	GPIO23 的 HP IO MUX 配置寄存器	0x005C	R/W
IO_MUX_GPIO24_REG	GPIO24 的 HP IO MUX 配置寄存器	0x0060	R/W
IO_MUX_GPIO25_REG	GPIO25 的 HP IO MUX 配置寄存器	0x0064	R/W
IO_MUX_GPIO26_REG	GPIO26 的 HP IO MUX 配置寄存器	0x0068	R/W
IO_MUX_GPIO27_REG	GPIO27 的 HP IO MUX 配置寄存器	0x006C	R/W
IO_MUX_GPIO28_REG	GPIO28 的 HP IO MUX 配置寄存器	0x0070	R/W
版本寄存器			
IO_MUX_DATE_REG	版本控制寄存器	0x01FC	R/W

5.18.3 GPIO EXT 寄存器列表

GPIO EXT 寄存器包括 SDM 寄存器、毛刺滤波器寄存器和 ETM 寄存器。

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址 + 0x0F00 的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节寄存器的访问类型，了解访问列缩写的含义。

名称	描述	地址	访问
SDM 配置寄存器			
GPIO_EXT_SIGMADELTA_MISC_REG	MISC 寄存器	0x0004	R/W
GPIO_EXT_SIGMADELTAO_REG	SDM 通道 0 的占空比配置寄存器	0x0008	R/W
GPIO_EXT_SIGMADELTA1_REG	SDM 通道 1 的占空比配置寄存器	0x000C	R/W
GPIO_EXT_SIGMADELTA2_REG	SDM 通道 2 的占空比配置寄存器	0x0010	R/W
GPIO_EXT_SIGMADELTA3_REG	SDM 通道 3 的占空比配置寄存器	0x0014	R/W
配置寄存器			
GPIO_EXT_PAD_COMP_CONFIG_O_REG	跨零检测配置寄存器	0x0058	R/W
GPIO_EXT_PAD_COMP_FILTER_O_REG	配置跨零检测的中断源屏蔽周期	0x005C	R/W
毛刺滤波器配置寄存器			
GPIO_EXT_GLITCH_FILTER_CHO_REG	毛刺滤波器通道 0 的配置寄存器	0x00D8	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	毛刺滤波器通道 1 的配置寄存器	0x00DC	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	毛刺滤波器通道 2 的配置寄存器	0x00E0	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	毛刺滤波器通道 3 的配置寄存器	0x00E4	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	毛刺滤波器通道 4 的配置寄存器	0x00E8	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	毛刺滤波器通道 5 的配置寄存器	0x00EC	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	毛刺滤波器通道 6 的配置寄存器	0x00F0	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	毛刺滤波器通道 7 的配置寄存器	0x00F4	R/W
ETM 配置寄存器			
GPIO_EXT_ETM_EVENT_CHO_CFG_REG	ETM 通道 0 的配置寄存器	0x0118	R/W
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM 通道 1 的配置寄存器	0x011C	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM 通道 2 的配置寄存器	0x0120	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM 通道 3 的配置寄存器	0x0124	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM 通道 4 的配置寄存器	0x0128	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM 通道 5 的配置寄存器	0x012C	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM 通道 6 的配置寄存器	0x0130	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM 通道 7 的配置寄存器	0x0134	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	ETM 的 GPIO 选择配置寄存器 0	0x0158	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	ETM 的 GPIO 选择配置寄存器 1	0x015C	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	ETM 的 GPIO 选择配置寄存器 2	0x0160	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	ETM 的 GPIO 选择配置寄存器 3	0x0164	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	ETM 的 GPIO 选择配置寄存器 4	0x0168	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	ETM 的 GPIO 选择配置寄存器 5	0x016C	R/W
中断寄存器			
GPIO_EXT_INT_RAW_REG	GPIO_EXT 的原始中断寄存器	0x01D0	RO/ WTC/ SS
GPIO_EXT_INT_ST_REG	GPIO_EXT 的屏蔽中断寄存器	0x01D4	RO
GPIO_EXT_INT_ENA_REG	GPIO_EXT 的中断使能寄存器	0x01D8	R/W
GPIO_EXT_INT_CLR_REG	GPIO_EXT 的中断清除寄存器	0x01DC	WT

名称	描述	地址	访问
版本寄存器			
GPIO_EXT_VERSION_REG	版本控制寄存器	0x01FC	R/W

5.18.4 LP GPIO 交换矩阵寄存器列表

本小节的所有地址均为相对于 LP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。请查看章节寄存器的访问类型，了解访问列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_GPIO_OUT_REG	LP GPIO 输出寄存器	0x0004	R/ W/ WTC
LP_GPIO_OUT_WITS_REG	LP GPIO 输出置位寄存器	0x0008	WT
LP_GPIO_OUT_WITC_REG	LP GPIO 输出清除寄存器	0x000C	WT
LP_GPIO_ENABLE_REG	LP GPIO 输出使能寄存器	0x0010	R/ W/ WTC
LP_GPIO_ENABLE_WITS_REG	LP GPIO 输出使能置位寄存器	0x0014	WT
LP_GPIO_ENABLE_WITC_REG	LP GPIO 输出使能清除寄存器	0x0018	WT
LP_GPIO_IN_REG	LP GPIO 输入寄存器	0x001C	RO
LP_GPIO_STATUS_REG	LP GPIO 中断状态寄存器	0x0020	R/ W/ WTC
LP_GPIO_STATUS_WITS_REG	LP GPIO 中断状态置位寄存器	0x0024	WT
LP_GPIO_STATUS_WITC_REG	LP GPIO 中断状态清除寄存器	0x0028	WT
LP_GPIO_STATUS_NEXT_REG	LP GPIO 中断源寄存器	0x002C	RO
LP_GPIO_PIN0_REG	LP GPIO0 配置寄存器	0x0030	varies
LP_GPIO_PIN1_REG	LP GPIO1 配置寄存器	0x0034	varies
LP_GPIO_PIN2_REG	LP GPIO2 配置寄存器	0x0038	varies
LP_GPIO_PIN3_REG	LP GPIO3 配置寄存器	0x003C	varies
LP_GPIO_PIN4_REG	LP GPIO4 配置寄存器	0x0040	varies
LP_GPIO_PIN5_REG	LP GPIO5 配置寄存器	0x0044	varies
LP_GPIO_PIN6_REG	LP GPIO6 配置寄存器	0x0048	varies
LP_GPIO_PIN7_REG	LP GPIO7 配置寄存器	0x004C	varies
LP_GPIO_FUNC0_OUT_SEL_CFG_REG	LP GPIO0 的输出配置寄存器	0x02B0	R/W
LP_GPIO_FUNC1_OUT_SEL_CFG_REG	LP GPIO1 的输出配置寄存器	0x02B4	R/W
LP_GPIO_FUNC2_OUT_SEL_CFG_REG	LP GPIO2 的输出配置寄存器	0x02B8	R/W
LP_GPIO_FUNC3_OUT_SEL_CFG_REG	LP GPIO3 的输出配置寄存器	0x02BC	R/W
LP_GPIO_FUNC4_OUT_SEL_CFG_REG	LP GPIO4 的输出配置寄存器	0x02C0	R/W
LP_GPIO_FUNC5_OUT_SEL_CFG_REG	LP GPIO5 的输出配置寄存器	0x02C4	R/W
LP_GPIO_FUNC6_OUT_SEL_CFG_REG	LP GPIO6 的输出配置寄存器	0x02C8	R/W
LP_GPIO_FUNC7_OUT_SEL_CFG_REG	LP GPIO7 的输出配置寄存器	0x02CC	R/W
时钟门控寄存器			
LP_GPIO_CLOCK_GATE_REG	LP GPIO 时钟门控寄存器	0x03F8	R/W
版本寄存器			
LP_GPIO_DATE_REG	版本控制寄存器	0x03FC	R/W

5.18.5 LP IO MUX 寄存器列表

本小节的所有地址均为相对于 LP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节寄存器的访问类型，了解访问列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_IO_MUX_GPIO0_REG	GPIO0 的 LP IO MUX 配置寄存器	0x0000	R/W
LP_IO_MUX_GPIO1_REG	GPIO1 的 LP IO MUX 配置寄存器	0x0004	R/W
LP_IO_MUX_GPIO2_REG	GPIO2 的 LP IO MUX 配置寄存器	0x0008	R/W
LP_IO_MUX_GPIO3_REG	GPIO3 的 LP IO MUX 配置寄存器	0x000C	R/W
LP_IO_MUX_GPIO4_REG	GPIO4 的 LP IO MUX 配置寄存器	0x0010	R/W
LP_IO_MUX_GPIO5_REG	GPIO5 的 LP IO MUX 配置寄存器	0x0014	R/W
LP_IO_MUX_GPIO6_REG	GPIO6 的 LP IO MUX 配置寄存器	0x0018	R/W
版本寄存器			
LP_IO_MUX_DATE_REG	版本控制寄存器	0x01FC	R/W

5.19 寄存器

5.19.1 HP GPIO 交换矩阵寄存器

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 5.1. GPIO_STRAP_REG (0x0000)

(reserved)																GPIO_STRAPPING															
31															16	15															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00														Reset

GPIO_STRAPPING 表示 strapping 管脚的值。

- Bit[0]: 无效
- Bit[1]: 无效
- Bit[2]: 无效
- Bit[3]: GPIO27
- Bit[4]: GPIO28
- Bit[5]: GPIO7
- Bit[6]: GPIO25
- Bit[7]~bit[15]: 无效

更多有关 strapping 管脚控制的功能，见章节 7 芯片 Boot 控制。

(RO)

Register 5.2. GPIO_OUT_REG (0x0004)

GPIO_OUT_DATA_ORIG																																
31																															0	
0x000000																																Reset

GPIO_OUT_DATA_ORIG 配置简单输出模式下 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出值。

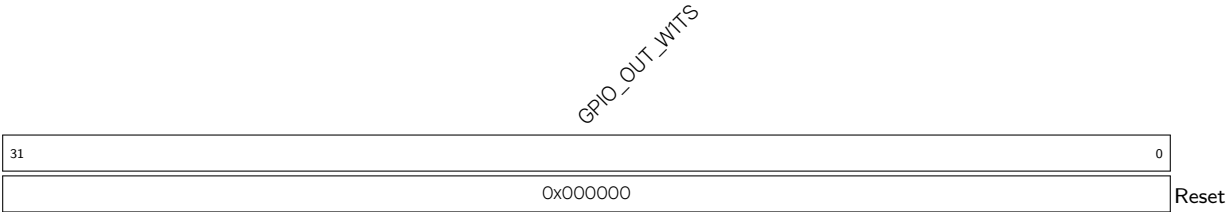
0: 低电平

1: 高电平

bit[0]~bit[14] 和 bit[23]~bit[28] 的值分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出值。

(R/W/SC/WTC)

Register 5.3. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS 配置是否置位 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出寄存器 [GPIO_OUT_REG](#)。

0: 不置位

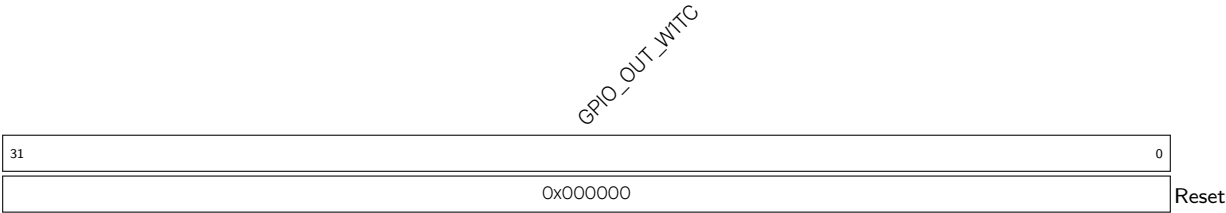
1: [GPIO_OUT_REG](#) 中相应位也将置 1

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来置位 [GPIO_OUT_REG](#)。

(WT)

Register 5.4. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC 配置是否清零 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出寄存器 [GPIO_OUT_REG](#)。

0: 不清零

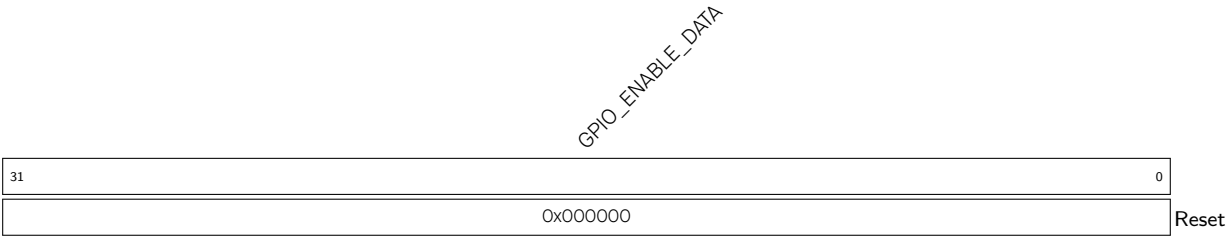
1: [GPIO_OUT_REG](#) 中相应位将被清零

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来清零 [GPIO_OUT_REG](#)。

(WT)

Register 5.5. GPIO_ENABLE_REG (0x0034)

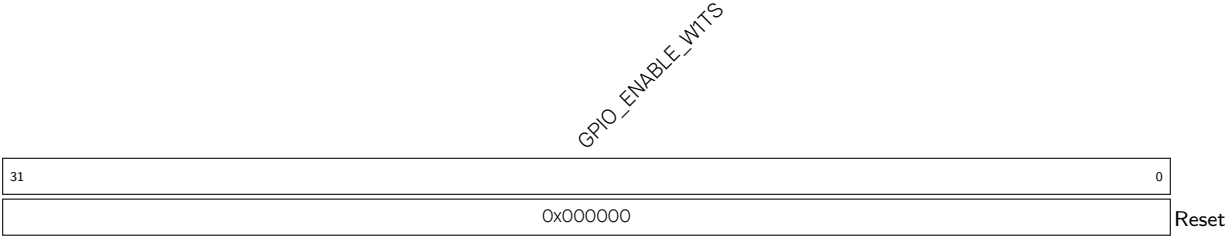


GPIO_ENABLE_DATA 配置是否使能 GPIO0~GPIO14 和 GPIO23~GPIO28 输出。

- 0: 不使能
- 1: 使能

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。
(R/W/WTC)

Register 5.6. GPIO_ENABLE_W1TS_REG (0x0038)

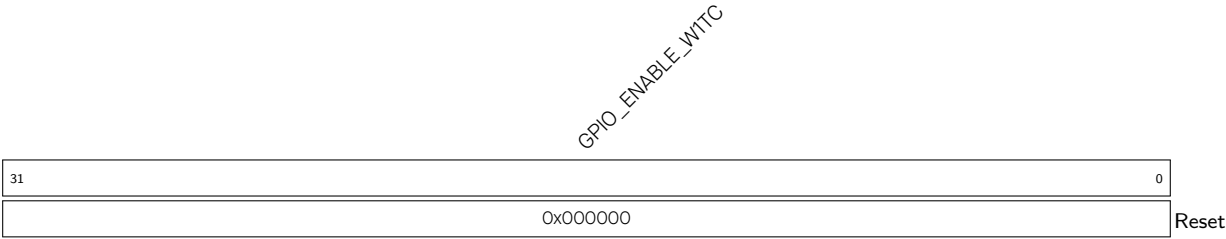


GPIO_ENABLE_W1TS 配置是否置位 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出使能寄存器 [GPIO_ENABLE_REG](#)。

- 0: 不置位
- 1: [GPIO_ENABLE_REG](#) 中相应位也将置 1

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。
推荐使用该寄存器来置位 [GPIO_ENABLE_REG](#)。
(WT)

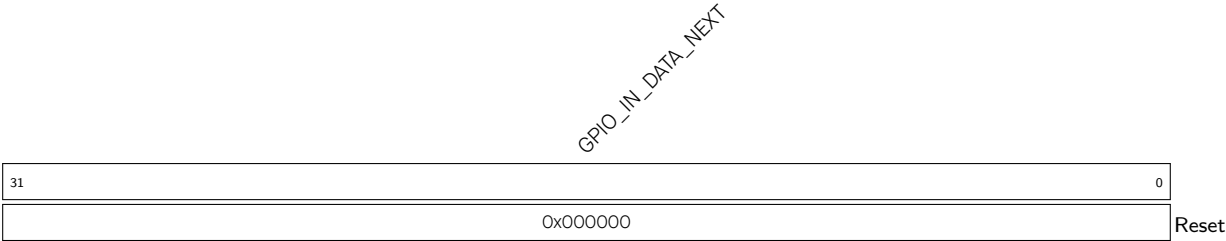
Register 5.7. GPIO_ENABLE_W1TC_REG (0x003C)



GPIO_ENABLE_W1TC 配置是否清零 GPIO0~GPIO14 和 GPIO23~GPIO28 的输出使能寄存器 [GPIO_ENABLE_REG](#)。

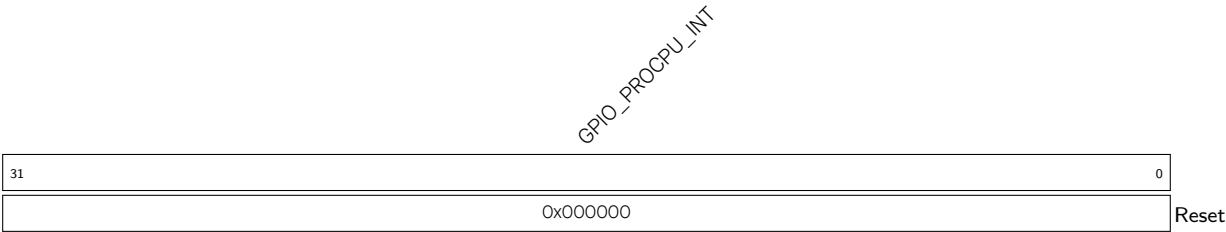
- 0: 不清零
 - 1: [GPIO_ENABLE_REG](#) 中相应位将被清零
- bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。
推荐使用该寄存器来清零 [GPIO_ENABLE_REG](#)。
(WT)

Register 5.8. GPIO_IN_REG (0x0064)



GPIO_IN_DATA_NEXT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的输入值。
每一位均代表一个管脚的输入值：
0: 低电平
1: 高电平
bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。
(RO)

Register 5.9. GPIO_PROCPU_INT_REG (0x00A4)



GPIO_PROCPU_INT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的 CPU 中断状态。

每位代表：

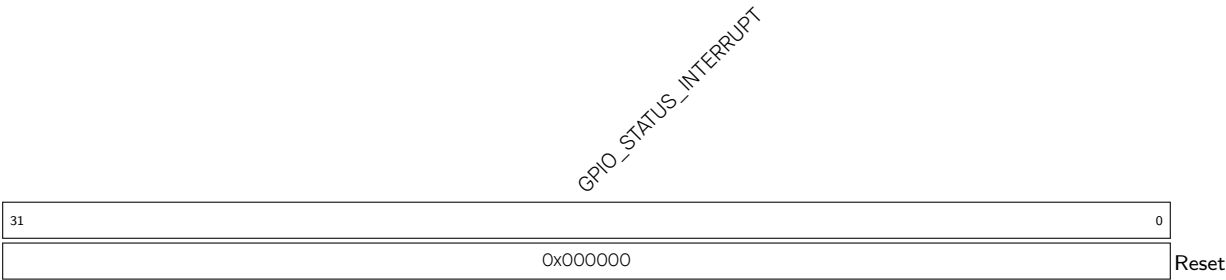
- 0：表示 CPU 中断未使能，或对应 GPIO 未产生 **GPIO_PINn_INT_TYPE** 所配置的中断。
- 1：表示 CPU 中断使能后，对应 GPIO 产生了 **GPIO_PINn_INT_TYPE** 所配置的中断。

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

如果 **GPIO_PINn_REG** 中 bit[13] 高电平有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 **GPIO_STATUS_REG** 中相应位的中断状态一致。

(RO)

Register 5.10. GPIO_STATUS_REG (0x0074)



GPIO_STATUS_INTERRUPT GPIO0~GPIO14 和 GPIO23~GPIO28 的中断状态寄存器，软件可配置该寄存器的值。

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

每一位均代表对应 GPIO 的中断状态：

- 0：表示对应 GPIO 未产生 **GPIO_PINn_INT_TYPE** 所配置的中断，或软件配置该位寄存器的值为 0。
- 1：表示对应 GPIO 产生了 **GPIO_PINn_INT_TYPE** 所配置的中断，或软件配置该位寄存器的值为 1。

(R/W/WTC)

Register 5.11. GPIO_STATUS_W1TS_REG (0x0078)

GPIO_STATUS_W1TS	
31	0
0x000000	
Reset	

GPIO_STATUS_W1TS 配置是否置位 GPIO0~GPIO14 和 GPIO23~GPIO28 的中断状态寄存器 [GPIO_STATUS_INTERRUPT](#)。

每一位的值可以为：

0：不置位

1： [GPIO_STATUS_INTERRUPT](#) 中的相应位也将置 1

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来置位 [GPIO_STATUS_INTERRUPT](#)。

(WT)

Register 5.12. GPIO_STATUS_W1TC_REG (0x007C)

GPIO_STATUS_W1TC	
31	0
0x000000	
Reset	

GPIO_STATUS_W1TC 配置是否清零 GPIO0~GPIO14 和 GPIO23~GPIO28 的中断状态寄存器 [GPIO_STATUS_INTERRUPT](#)。

每一位的值可以为：

0：不清零

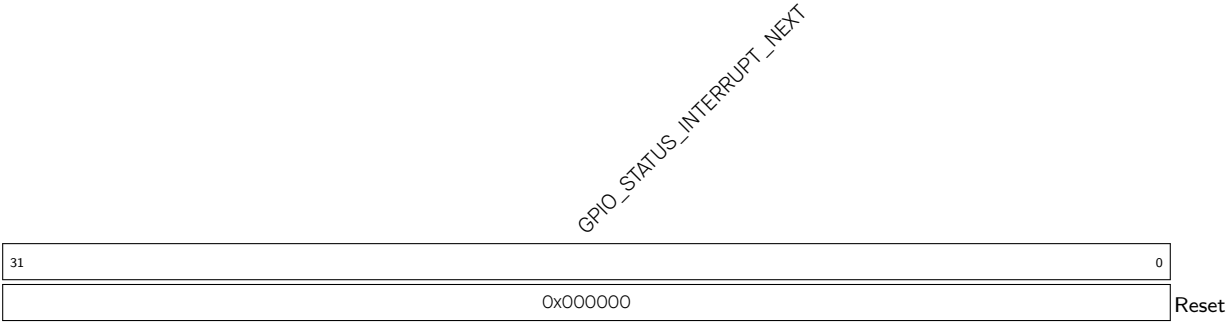
1： [GPIO_STATUS_INTERRUPT](#) 中的相应位将被清零

bit[0]~bit[14] 和 bit[23]~bit[28] 分别对应 GPIO0~GPIO14 和 GPIO23~GPIO28。

推荐使用此寄存器来清零 [GPIO_STATUS_INTERRUPT](#)。

(WT)

Register 5.13. GPIO_STATUS_NEXT_REG (0x00B4)



GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0~GPIO14 和 GPIO23~GPIO28 的中断源信号。

每位代表：

0：表示 GPIO 未产生 GPIO_PINn_INT_TYPE 所配置的中断

1：GPIO 产生了 GPIO_PINn_INT_TYPE 所配置的中断

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 5.14. GPIO_PIN n _REG (n : 0-14, 23-28) (0x00C4+0x4* n)

(reserved)																GPIO_PIN _n _INT_ENA				(reserved)			GPIO_PIN _n _WAKEUP_ENABLE				GPIO_PIN _n _INT_TYPE				(reserved)			GPIO_PIN _n _SYNC1_BYPASS				GPIO_PIN _n _PAD_DRIVER				GPIO_PIN _n _SYNC2_BYPASS									
31																18				17				13				12		11		10		9		7		6		5		4		3		2		1		0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x0				0		0		0		0x0		0		0		0x0		0		0x0		Reset													

GPIO_PIN_n_SYNC2_BYPASS 配置是否使能 GPIO 输入数据在第二级同步中 HP IO MUX 运行时钟上升沿同步或下降沿同步。

0: 不同步

1: 在下降沿进行同步

2: 在上升沿进行同步

3: 在上升沿进行同步

(R/W)

GPIO_PIN n _PAD_DRIVER 配置选择管脚的输出驱动模式。

0: 正常输出

1: 开漏输出

(R/W)

GPIO_PIN_n_SYNC1_BYPASS 配置是否使能 GPIO 输入数据在第一级同步中 HP IO MUX 运行时钟上升沿同步或下降沿同步。

0: 不同步

1: 在下降沿进行同步

2: 在上升沿进行同步

3: 在上升沿进行同步

(R/W)

GPIO PIN_{*n*} INT TYPE 配置 GPIO 中断类型。

0: 关闭 GPIO 中断

1: 上升沿触发

2: 下降沿触发

3: 任一沿触发

4: 低电平触发

5: 高电平触发

(R/W)

GPIO_PIN_n_WAKEUP_ENABLE 配置是否使能 GPIO 唤醒功能。

0: 不使能

1: 使能

该功能只能将 CPU 从 Light-sleep 中唤醒。

(R/W)

见下页

Register 5.14. GPIO_PIN n _REG (n : 0-14, 23-28) (0x00C4+0x4* n)

接上页

GPIO_PIN_n_INT_ENA 配置是否使能 CPU 中断或 CPU 非屏蔽中断。

- bit[13]: 配置是否使能 CPU 中断:

0: 不使能

1: 使能

- bit[14]~bit[17]: 无效

(R/W)

Register 5.15. GPIO_FUNC*m*_IN_SEL_CFG_REG (*m*: 6-17) (0x02DC+0x4*(*m*-6))

Diagram illustrating the structure of the 32-bit register:

- Bits 31 to 8: (reserved)
- Bit 7: GPIO_SIGM_IN_SEL
- Bit 6: GPIO_FUNCm_IN_SEL
- Bits 5 to 0: GPIO_FUNCm_IN_SEL (Reset value: 0x60)

GPIO_FUNC m _IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 m 。

0x0: 选择 GPIO0

0x1: 选择 GPIO1

• • • • •

0x1B: 选择 GPIO27

0x1C: 选择 GPIO28

Or

0x40: 选择恒高电平输入

0x60: 选择恒低电平输入

(R/W)

GPIO_FUNC_m_IN_INV_SEL 配置是否反转输入值。

0: 不反转

1: 反转

(R/W)

GPIO_SIG_m_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0: 旁路 HP GPIO 交换矩阵, 即直接通过 HP IO MUX 连接信号与外设

1: 通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 5.16. GPIO_FUNC_p IN_SEL_CFG_REG (*p*: 27-35) (0x0330+0x4*(*p*-27))

[illegible]

GPIO_FUNC*p*_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 *p*。

0x0: 选择 GPIO0

0x1: 选择 GPIO1

• • • • •

0x1B: 选择 GPIO27

0x1C: 选择 GPIO28

Or

0x40: 选择恒高电平输入

0x60: 选择恒低电平输入

(R/W)

GPIO_FUNC*p*_IN_INV_SEL 配置是否反转输入值。

0: 不反转

1: 反转

(R/W)

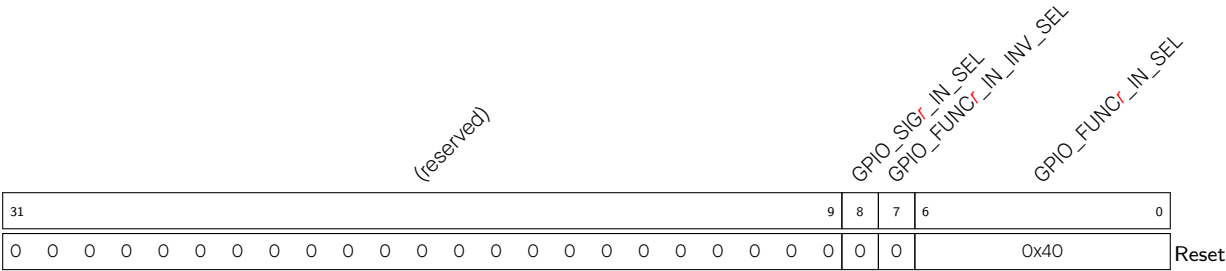
GPIO_SIG_p_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0: 旁路 HP GPIO 交换矩阵, 即直接通过 HP IO MUX 连接信号与外设

1: 通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 5.17. GPIO_FUNC*r*_IN_SEL_CFG_REG (*r*: 46-66) (0x037C+0x4*(*r*-46))



GPIO_FUNC*r*_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 *r*。

- 0x0: 选择 GPIO0
- 0x1: 选择 GPIO1
-
- 0x1B: 选择 GPIO27
- 0x1C: 选择 GPIO28
- Or
- 0x40: 选择恒高电平输入
- 0x60: 选择恒低电平输入
- (R/W)

GPIO_FUNC*r*_IN_INV_SEL 配置是否反转输入值。

- 0: 不反转
- 1: 反转
- (R/W)

GPIO_SIG*r*_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

- 0: 旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设
- 1: 通过 HP GPIO 交换矩阵连接信号与外设
- (R/W)

Register 5.18. GPIO_FUNC70_IN_SEL_CFG_REG (0x03DC)

(reserved)																								GPIO_SIG70_IN_SEL		GPIO_FUNC70_IN_INV_SEL		GPIO_FUNC70_IN_SEL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
31																							9	8	7	6			0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

GPIO_FUNC70_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 70。

- 0x0: 选择 GPIO0
 - 0x1: 选择 GPIO1
 -
 - 0x1B: 选择 GPIO27
 - 0x1C: 选择 GPIO28
 - Or
 - 0x40: 选择恒高电平输入
 - 0x60: 选择恒低电平输入
- (R/W)

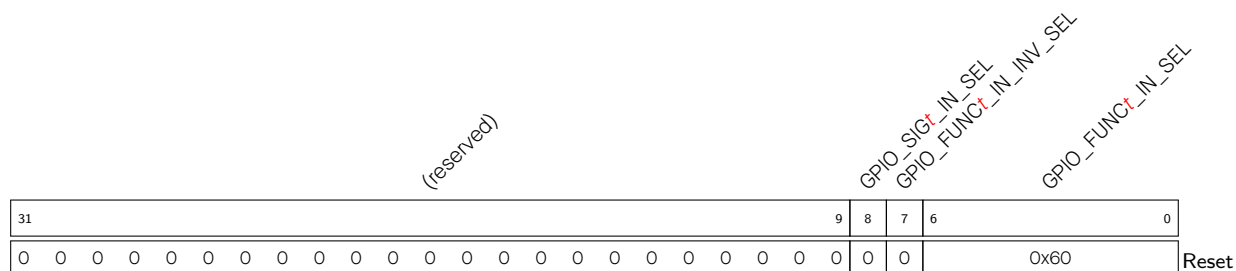
GPIO_FUNC70_IN_INV_SEL 配置是否反转输入值。

- 0: 不反转
 - 1: 反转
- (R/W)

GPIO_SIG70_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

- 0: 旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设
 - 1: 通过 HP GPIO 交换矩阵连接信号与外设
- (R/W)

Register 5.19. GPIO_FUNC*t*_IN_SEL_CFG_REG (*t*: 76-88) (0x03F4+0x4*(*t*-76))



GPIO_FUNC t _IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 t 。

0x0: 选择 GPIO0

0x1: 选择 GPIO1

• • • • •

0x1B: 选择 GPIO27

0x1C: 选择 GPIO28

Or

0x40: 选择恒高电平输入

0x60: 选择恒低电平输入

(R/W)

GPIO_FUNC_t_IN_INV_SEL 配置是否反转输入值。

0: 不反转

1: 反转

(R/W)

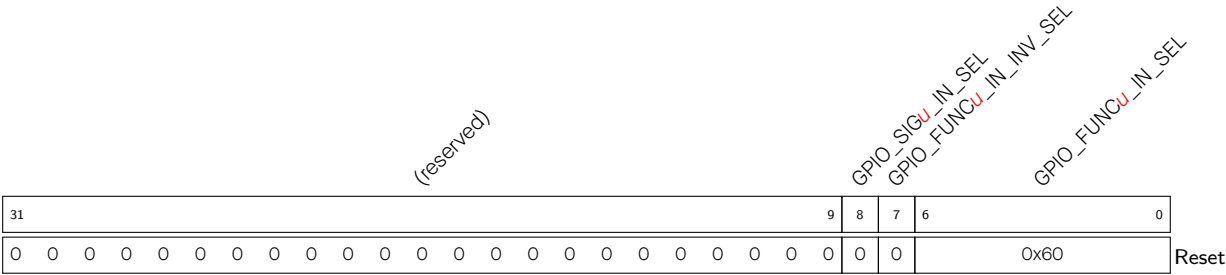
GPIO_SIG^t IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

0: 旁路 HP GPIO 交换矩阵, 即直接通过 HP IO MUX 连接信号与外设

1: 通过 HP GPIO 交换矩阵连接信号与外设

(R/W)

Register 5.20. GPIO_FUNCu_IN_SEL_CFG_REG (u: 97-116) (0x0448+0x4*(u-97))



GPIO_FUNCu_IN_SEL 配置从 21 个 GPIO 中选择一个管脚连接输入信号 u。

- 0x0: 选择 GPIO0
- 0x1: 选择 GPIO1
-
- 0x1B: 选择 GPIO27
- 0x1C: 选择 GPIO28
- Or
- 0x40: 选择恒高电平输入
- 0x60: 选择恒低电平输入
- (R/W)

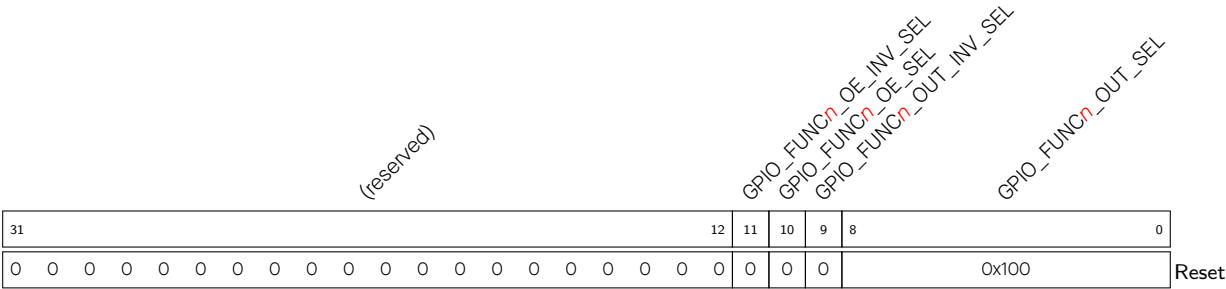
GPIO_FUNCu_IN_INV_SEL 配置是否反转输入值。

- 0: 不反转
- 1: 反转
- (R/W)

GPIO_SIGu_IN_SEL 配置是否通过 HP GPIO 交换矩阵连接信号。

- 0: 旁路 HP GPIO 交换矩阵，即直接通过 HP IO MUX 连接信号与外设
- 1: 通过 HP GPIO 交换矩阵连接信号与外设
- (R/W)

Register 5.21. GPIO_FUNCn_OUT_SEL_CFG_REG (n: 0-14, 23-28) (0x0AC4+0x4*n)



GPIO_FUNCn_OUT_SEL 配置从外设信号中选择一个信号 Y ($0 \leq Y < 256$) 输出至 GPIO n 。

- 0: 选择信号 0
 - 1: 选择信号 1
 -
 - 254: 选择信号 254
 - 255: 选择信号 255
 - 256: GPIO_OUT_REG 和 GPIO_ENABLE_REG 中的 bit[n] 将用作输出值和输出使能信号。
- 信号列表见表 5.12-1。
- (R/W/SC)

GPIO_FUNCn_OUT_INV_SEL 配置是否反转输出值。

- 0: 不反转
 - 1: 反转
- (R/W/SC)

GPIO_FUNCn_OE_SEL 配置选择输出使能信号。

- 0: 使用外设的输出使能信号
 - 1: 强制使用 GPIO_ENABLE_REG 中的 bit[n] 用作输出使能信号。
- (R/W)

GPIO_FUNCn_OE_INV_SEL 配置是否反转输出使能信号。

- 0: 不反转
 - 1: 反转
- (R/W)

Register 5.22. GPIO_CLOCK_GATE_REG (0x0DF8)

(reserved)																												GPIO_CLK_EN	
31																												1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

GPIO_CLK_EN 配置是否使能时钟门控。

0: 不使能

1: 此位置 1，则时钟自由运转。

(R/W)

Register 5.23. GPIO_DATE_REG (0x0DFC)

(reserved)				GPIO_DATE																										
31	28	27																											0	
0	0	0	0	0x2312140																										Reset

GPIO_DATE 版本控制寄存器。

(R/W)

5.19.2 HP IO MUX 寄存器

本小节的所有地址均为相对于 HP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 5.24. IO_MUX_GPIO n _REG (n : 0-14, 23-28) (0x0000+0x4* n)

(reserved)																IO_MUX_GPIO _n _HYS_SEL IO_MUX_GPIO _n _HYS_EN IO_MUX_GPIO _n _FILTER_EN IO_MUX_GPIO _n _MCU_SEL IO_MUX_GPIO _n _FUN_DRV IO_MUX_GPIO _n _FUN_IE IO_MUX_GPIO _n _FUN_WPU IO_MUX_GPIO _n _FUN_WPD IO_MUX_GPIO _n _MCU_DRV IO_MUX_GPIO _n _MCU_IE IO_MUX_GPIO _n _MCU_WPU IO_MUX_GPIO _n _MCU_WPD IO_MUX_GPIO _n _SLP_SEL IO_MUX_GPIO _n _MCU_OE															
31													18	17	16	15	14	12		11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x1	2		0	0	0	0	0	0	0	0	0	0	Reset				

IO_MUX_GPIO n _MCU_OE 配置是否使能睡眠模式下 GPIO n 的输出。
0: 不使能
1: 使能
(R/W)

IO_MUX_GPIO n _SLP_SEL 配置是否使能 GPIO n 进入睡眠模式。
0: 不进入
1: 进入
(R/W)

IO_MUX_GPIO n _MCU_WPD 配置是否使能睡眠模式下 GPIO n 的下拉电阻。
0: 不使能
1: 使能
(R/W)

IO_MUX_GPIO n _MCU_WPU 配置是否使能睡眠模式下 GPIO n 的上拉电阻。
0: 不使能
1: 使能
(R/W)

IO_MUX_GPIO n _MCU_IE 配置是否使能睡眠模式下 GPIO n 的输入。
0: 不使能
1: 使能
(R/W)

IO_MUX_GPIO n _MCU_DRV 配置睡眠模式下 GPIO n 的驱动强度。
0: ~5 mA
1: ~10 mA
2: ~20 mA
3: ~40 mA
(R/W)

IO_MUX_GPIO n _FUN_WPD 配置是否使能 GPIO n 的下拉电阻。
0: 不使能
1: 使能
(R/W)

见下页

Register 5.24. IO_MUX_GPIO n _REG (n : 0-14, 23-28) (0x0000+0x4* n)

接上页

IO_MUX_GPIO n _FUN_WPU 配置是否使能 GPIO n 的上拉电阻。

0: 不使能

1: 使能

(R/W)

IO_MUX_GPIO n _FUN_IE 配置是否使能 GPIO n 的输入。

0: 不使能

1: 使能

(R/W)

IO_MUX_GPIO n _FUN_DRV 配置 GPIO n 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO n _MCU_SEL 配置选择管脚的 HP IO MUX 功能。

0: 选择功能 0

1: 选择功能 1

.....

(R/W)

IO_MUX_GPIO n _FILTER_EN 配置是否使能管脚输入信号的滤波功能。

0: 不使能

1: 使能

(R/W)

IO_MUX_GPIO n _HYS_EN 当 IO_MUX_GPIO n _HYS_SEL 设置为 1 时，配置是否使能管脚的迟滞功能。

0: 不使能

1: 使能

(R/W)

IO_MUX_GPIO n _HYS_SEL 配置选择 GPIO n 迟滞功能的使能信号。

0: 使用 eFuse 的输出使能信号

1: 使用 IO_MUX_GPIO n _HYS_EN 用作输出使能信号

(R/W)

Register 5.25. IO_MUX_REG_DATE_REG (0x01FC)

(reserved)				IO_MUX_REG_DATE																									
31				28	27																								0
0	0	0	0	0x2311270																									Reset

IO_MUX_REG_DATE 版本控制寄存器。
(R/W)

5.19.3 GPIO EXT 寄存器

本小节的所有地址均为相对于 HP GPIO 交换矩阵基地址 + 0x0F00 的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节[如何配置寄存器的保留域](#)。

Register 5.26. GPIO_EXT_SIGMADELTA_MISC_REG (0x0004)

(reserved)																															GPIO_EXT_SIGMADELTA_CLK_EN				
31																															1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

GPIO_EXT_SIGMADELTA_CLK_EN 配置是否使能 SDM 时钟。

0：不使能

1：使能

(R/W)

Register 5.27. GPIO_EXT_SIGMADELTA n _REG (n : 0-3) (0x0008+0x4* n)

Diagram of the 32-bit GPIO_EXT_CONFIG register structure:

31	16	15	8	7	0
(reserved)		GPIO_EXT_SD _n _PRESCALE		GPIO_EXT_SD _n _IN	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		0xff		0x0	

GPIO_EXT_SD_n_IN 配置 SDM 输出信号的占空比。

(R/W)

GPIO_EXT_SDn_PRESCALE 配置 HP IO MUX 运行时钟的分频系数，分频系数有效值为 0~255。

0: 不分频

1~255: 2~256 分频

(R/W)

Register 5.28. GPIO_EXT_PAD_COMP_CONFIG_O_REG (0x0058)

(reserved)																GPIO_EXT_DREF_COMP_O			GPIO_EXT_MODE_COMP_O			GPIO_EXT_XPD_COMP_O		
31															5	4			2	1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0		0		0	Reset		

GPIO_EXT_XPD_COMP_O 配置是否使能模拟 PAD 电压比较器。

0: 不使能

1: 使能

(R/W)

GPIO_EXT_MODE_COMP_O 配置选择模拟 PAD 电压比较器的参考电压。

0: 参考电压为内部参考电压，同时 GPIO8 可用作常规 GPIO

1: 参考电压为 GPIO8 PAD 上的电压

(R/W)

GPIO_EXT_DREF_COMP_O 配置选择模拟 PAD 电压比较器的内部参考电压。

0: 内部参考电压为 0 * VDDPST1

1: 内部参考电压为 0.1 * VDDPST1

2: 内部参考电压为 0.2 * VDDPST1

3: 内部参考电压为 0.3 * VDDPST1

4: 内部参考电压为 0.4 * VDDPST1

5: 内部参考电压为 0.5 * VDDPST1

6: 内部参考电压为 0.6 * VDDPST1

7: 内部参考电压为 0.7 * VDDPST1

(R/W)

Register 5.29. GPIO_EXT_PAD_COMP_FILTER_O_REG (0x005C)

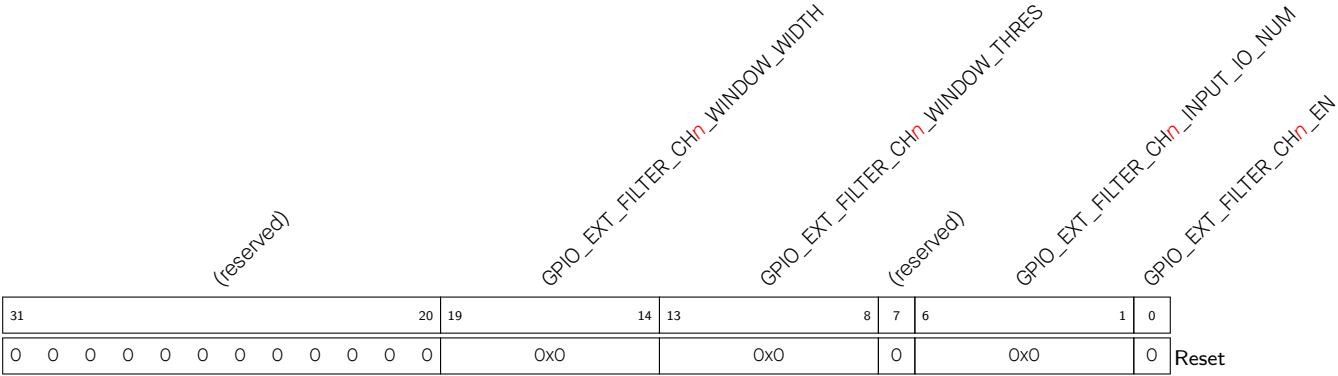
GPIO_EXT_ZERO_DET_FILTER_CNT_O																															
31																															0
0x000000																															
Reset																															

GPIO_EXT_ZERO_DET_FILTER_CNT_O 配置模拟 PAD 电压比较器的新中断源屏蔽周期。

单位: HP IO MUX 运行时钟周期

(R/W)

Register 5.30. GPIO_EXT_GLITCH_FILTER_CHn_REG (n: 0-7) (0x00D8+0x4*n)



GPIO_EXT_FILTER_CHn_EN 配置是否使能毛刺滤波器的通道 n。
0: 不使能
1: 使能
(R/W)

GPIO_EXT_FILTER_CHn_INPUT_IO_NUM 配置选择毛刺滤波器的 GPIO 输入。
0: 选择 GPIO0
1: 选择 GPIO1
.....
27: 选择 GPIO27
28: 选择 GPIO28
29~63: 保留
(R/W)

GPIO_EXT_FILTER_CHn_WINDOW_THRES 配置毛刺滤波器的窗口阈值。窗口阈值应小于或等于 [GPIO_EXT_FILTER_CHn_WINDOW_WIDTH](#)。
单位: HP IO MUX 运行时钟的一个周期
(R/W)

GPIO_EXT_FILTER_CHn_WINDOW_WIDTH 配置毛刺滤波器的窗口宽度。窗口宽度的有效值为 0~63。
单位: HP IO MUX 运行时钟的一个周期
(R/W)

Register 5.31. GPIO_EXT_ETM_EVENT_CH n _CFG_REG (n : 0-7) (0x0118+0x4* n)

Register diagram for `GPIO_EXT_ETM_CHn_EVENT_SEL`:

- Bits 31-8: (reserved)
- Bits 7-6: (reserved)
- Bit 5: `GPIO_EXT_ETM_CHn_EVENT_SEL`
- Bit 0: `Reset`

GPIO_EXT_ETM_CH n _EVENT_SEL 配置选择 ETM 事件通道使用的 GPIO。

0: 选择 GPIO0

1: 选择 GPIO1

• • • • •

27: 选择 GPIO27

28: 选择 GPIO28

29~63: 保留

(R/W)

GPIO_EXT_ETM_CH n _EVENT_EN 配置是否使能 ETM 事件发送。

0: 不使能

1: 使能

(R/W)

Register 5.32. GPIO_EXT_ETM_TASK_PO_CFG_REG (0x0158)

(reserved)		GPIO_EXT_ETM_TASK_GPIO4_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO4_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO3_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO3_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO2_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO2_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO1_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO1_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO0_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO0_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO0_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO0_EN	
31	30	29	28	27	26	24		23	22	21	20	18		17	16	15	14	12		11	10	9	8	6		5	4	3	2	0																	
0	0	0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0		0	0	0x0		Reset																

GPIO_EXT_ETM_TASK_GPIO0_SEL 配置选择 GPIO0 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO0_EN 配置是否使能 GPIO0 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO1_SEL 配置选择 GPIO1 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO1_EN 配置是否使能 GPIO1 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

见下页

Register 5.32. GPIO_EXT_ETM_TASK_PO_CFG_REG (0x0158)

接上页

GPIO_EXT_ETM_TASK_GPIO2_SEL 配置选择 GPIO2 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO2_EN 配置是否使能 GPIO2 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO3_SEL 配置选择 GPIO3 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO3_EN 配置是否使能 GPIO3 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO4_SEL 配置选择 GPIO4 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

见下页

Register 5.32. GPIO_EXT_ETM_TASK_PO_CFG_REG (0x0158)

[接上页](#)

GPIO_EXT_ETM_TASK_GPIO4_EN 配置是否使能 GPIO4 响应 ETM 任务。

0: 不使能

1: 使能

(R/W)

Register 5.33. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)

(reserved)		GPIO_EXT_ETM_TASK_GPIO9_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO9_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO8_EN		GPIO_EXT_ETM_TASK_GPIO8_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO7_EN		GPIO_EXT_ETM_TASK_GPIO7_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO6_EN		GPIO_EXT_ETM_TASK_GPIO6_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO5_EN		GPIO_EXT_ETM_TASK_GPIO5_SEL	
31	30	29	28	27	26	24	23	22	21	20	18	17	16	15	14	12	11	10	9	8	6	5	4	3	2	0					
0	0	0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		Reset				

GPIO_EXT_ETM_TASK_GPIO5_SEL 配置选择 GPIO5 的 ETM 任务通道。

- 0: 选择通道 0
1: 选择通道 1
2: 选择通道 2
3: 选择通道 3
4: 选择通道 4
5: 选择通道 5
6: 选择通道 6
7: 选择通道 7

(R/W)

GPIO_EXT_ETM_TASK_GPIO5_EN 配置是否使能 GPIO5 响应 ETM 任务。

- 0: 不使能
1: 使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO6_SEL 配置选择 GPIO6 的 ETM 任务通道。

- 0: 选择通道 0
1: 选择通道 1
2: 选择通道 2
3: 选择通道 3
4: 选择通道 4
5: 选择通道 5
6: 选择通道 6
7: 选择通道 7

(R/W)

GPIO_EXT_ETM_TASK_GPIO6_EN 配置是否使能 GPIO6 响应 ETM 任务。

- 0: 不使能
1: 使能

(R/W)

见下页

Register 5.33. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)

接上页

GPIO_EXT_ETM_TASK_GPIO7_SEL 配置选择 GPIO7 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO7_EN 配置是否使能 GPIO7 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO8_SEL 配置选择 GPIO8 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO8_EN 配置是否使能 GPIO8 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO9_SEL 配置选择 GPIO9 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

见下页

Register 5.33. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x015C)

接上页

GPIO_EXT_ETM_TASK_GPIO9_EN 配置是否使能 GPIO9 响应 ETM 任务。

0: 不使能

1: 使能

(R/W)

Register 5.34. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)

(reserved)		GPIO_EXT_ETM_TASK_GPIO14_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO14_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO13_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO13_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO12_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO12_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO11_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO11_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO10_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO10_SEL	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
0	0	0	0	0	0x0	0	0	0	0	0	0x0	0	0	0	0	0	0x0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0x0	Reset						

GPIO_EXT_ETM_TASK_GPIO10_SEL 配置选择 GPIO10 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO10_EN 配置是否使能 GPIO10 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO11_SEL 配置选择 GPIO11 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO11_EN 配置是否使能 GPIO11 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

见下页

Register 5.34. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)

接上页

GPIO_EXT_ETM_TASK_GPIO12_SEL 配置选择 GPIO12 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO12_EN 配置是否使能 GPIO12 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO13_SEL 配置选择 GPIO13 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO13_EN 配置是否使能 GPIO13 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

GPIO_EXT_ETM_TASK_GPIO14_SEL 配置选择 GPIO14 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

见下页

Register 5.34. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x0160)

[接上页](#)

GPIO_EXT_ETM_TASK_GPIO14_EN 配置是否使能 GPIO14 响应 ETM 任务。

0: 不使能

1: 使能

(R/W)

Register 5.35. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x0168)

(reserved)		GPIO_EXT_ETM_TASK_GPIO24_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO24_SEL		(reserved)		GPIO_EXT_ETM_TASK_GPIO23_EN		(reserved)		GPIO_EXT_ETM_TASK_GPIO23_SEL		(reserved)		(reserved)		(reserved)		(reserved)		(reserved)		(reserved)		(reserved)		(reserved)	
31	30	29	28	27	26	24	23	22	21	20	18	17	16	15	14	12	11	10	9	8	6	5	4	3	2	0					
0	0	0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0x0		0	0	0	0x0	

Reset

GPIO_EXT_ETM_TASK_GPIO23_SEL 配置选择 GPIO23 的 ETM 任务通道。

- 0: 选择通道 0
1: 选择通道 1
2: 选择通道 2
3: 选择通道 3
4: 选择通道 4
5: 选择通道 5
6: 选择通道 6
7: 选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO23_EN 配置是否使能 GPIO23 响应 ETM 任务。

- 0: 不使能
1: 使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO24_SEL 配置选择 GPIO24 的 ETM 任务通道。

- 0: 选择通道 0
1: 选择通道 1
2: 选择通道 2
3: 选择通道 3
4: 选择通道 4
5: 选择通道 5
6: 选择通道 6
7: 选择通道 7
(R/W)

见下页

Register 5.35. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x0168)

接上页

GPIO_EXT_ETM_TASK_GPIO24_EN 配置是否使能 GPIO24 响应 ETM 任务。

0: 不使能

1: 使能

(R/W)

Register 5.36. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x016C)

(reserved)								GPIO_EXT_ETM_TASK_GPIO28_EN								(reserved)								GPIO_EXT_ETM_TASK_GPIO28_SEL								(reserved)								GPIO_EXT_ETM_TASK_GPIO27_EN								GPIO_EXT_ETM_TASK_GPIO27_SEL								(reserved)								GPIO_EXT_ETM_TASK_GPIO26_EN								GPIO_EXT_ETM_TASK_GPIO26_SEL								(reserved)								GPIO_EXT_ETM_TASK_GPIO25_EN							
31								24								23	22	21	20	18								17	16	15	14	12								11	10	9	8	6								5	4	3	2	0																																							
0 0 0 0 0 0 0 0								0 0 0 0x0								0 0 0 0x0								0 0 0 0x0								0 0 0 0x0								0 0 0 0x0								0 0 0 0x0								0 0 0 0x0								0 0 0 0x0								Reset																							

GPIO_EXT_ETM_TASK_GPIO25_SEL 配置选择 GPIO25 的 ETM 任务通道。

- 0: 选择通道 0
1: 选择通道 1
2: 选择通道 2
3: 选择通道 3
4: 选择通道 4
5: 选择通道 5
6: 选择通道 6
7: 选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO25_EN 配置是否使能 GPIO25 响应 ETM 任务。

- 0: 不使能
1: 使能
(R/W)

GPIO_EXT_ETM_TASK_GPIO26_SEL 配置选择 GPIO26 的 ETM 任务通道。

- 0: 选择通道 0
1: 选择通道 1
2: 选择通道 2
3: 选择通道 3
4: 选择通道 4
5: 选择通道 5
6: 选择通道 6
7: 选择通道 7
(R/W)

GPIO_EXT_ETM_TASK_GPIO26_EN 配置是否使能 GPIO26 响应 ETM 任务。

- 0: 不使能
1: 使能
(R/W)

见下页

Register 5.36. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x016C)

接上页

GPIO_EXT_ETM_TASK_GPIO27_SEL 配置选择 GPIO27 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO27_EN 配置是否使能 GPIO27 响应 ETM 任务。

- 0: 不使能
 - 1: 使能
- (R/W)

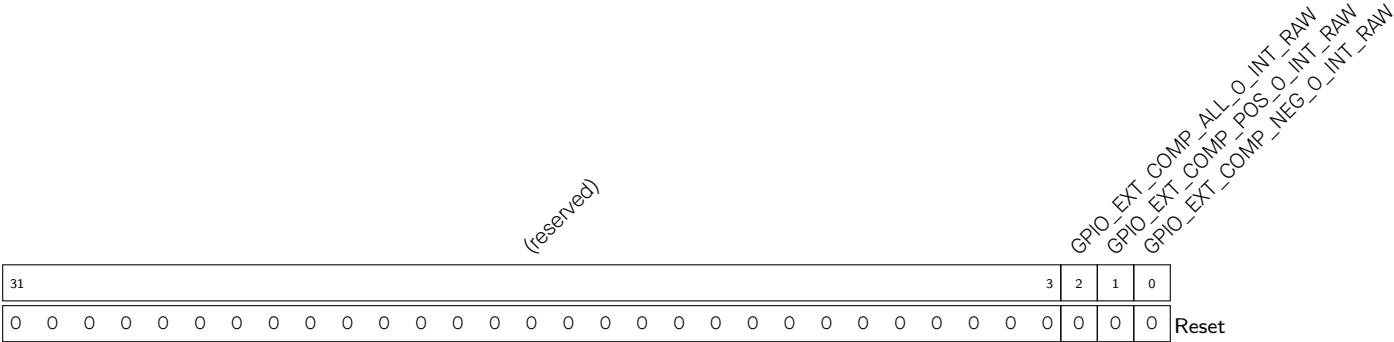
GPIO_EXT_ETM_TASK_GPIO28_SEL 配置选择 GPIO28 的 ETM 任务通道。

- 0: 选择通道 0
 - 1: 选择通道 1
 - 2: 选择通道 2
 - 3: 选择通道 3
 - 4: 选择通道 4
 - 5: 选择通道 5
 - 6: 选择通道 6
 - 7: 选择通道 7
- (R/W)

GPIO_EXT_ETM_TASK_GPIO28_EN 配置是否使能 GPIO28 响应 ETM 任务。

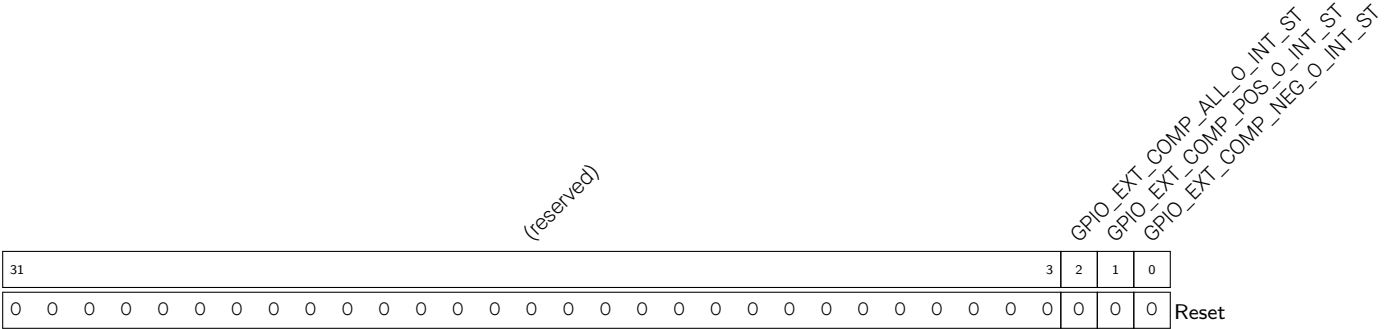
- 0: 不使能
 - 1: 使能
- (R/W)

Register 5.37. GPIO_EXT_INT_RAW_REG (0x01D0)



- GPIO_EXT_COMP_NEG_O_INT_RAW GPIO_EXT_COMP_NEG_O_INT 的原始中断状态。
(RO/WTC/SS)
- GPIO_EXT_COMP_POS_O_INT_RAW GPIO_EXT_COMP_POS_O_INT 的原始中断状态。
(RO/WTC/SS)
- GPIO_EXT_COMP_ALL_O_INT_RAW GPIO_EXT_COMP_ALL_O_INT 的原始中断状态。
(RO/WTC/SS)

Register 5.38. GPIO_EXT_INT_ST_REG (0x01D4)



- GPIO_EXT_COMP_NEG_O_INT_ST GPIO_EXT_COMP_NEG_O_INT 的屏蔽中断状态。
(RO)
- GPIO_EXT_COMP_POS_O_INT_ST GPIO_EXT_COMP_POS_O_INT 的屏蔽中断状态。
(RO)
- GPIO_EXT_COMP_ALL_O_INT_ST GPIO_EXT_COMP_ALL_O_INT 的屏蔽中断状态。
(RO)

Register 5.39. GPIO_EXT_INT_ENA_REG (0x01D8)

(reserved)																															GPIO_EXT_COMP_ALL_O_INT_ENA GPIO_EXT_COMP_POS_O_INT_ENA GPIO_EXT_COMP_NEG_O_INT_ENA				
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	Reset		

GPIO_EXT_COMP_NEG_O_INT_ENA 写1使能 GPIO_EXT_COMP_NEG_O_INT 中断。
(R/W)

GPIO_EXT_COMP_POS_0_INT_ENA 写 1 使能 GPIO_EXT_COMP_POS_0_INT 中断。
(R/W)

GPIO_EXT_COMP_ALL_O_INT_ENA 写1使能 GPIO_EXT_COMP_ALL_O_INT 中断。
(R/W)

Register 5.40. GPIO_EXT_INT_CLR_REG (0x01DC)

Diagram of the 32-bit GPIO_EXT_COMP register. The register is divided into four 8-bit fields. The top 24 bits are labeled "(reserved)". The bottom 8 bits are labeled "Reset" and contain the value 00000000. The four 8-bit fields are:

- Field 1 (bits 31-24): 00000000
- Field 2 (bits 23-16): 00000000
- Field 3 (bits 15-8): 00000000
- Field 4 (bits 7-0): 00000000

The labels for the fields are: GPIO_EXT_COMP_ALL_0_INT_CLR, GPIO_EXT_COMP_POS_0_INT_CLR, GPIO_EXT_COMP_NEG_0_INT_CLR, and GPIO_EXT_COMP_POS_0_INT_CLR.

GPIO_EXT_COMP_NEG_O_INT_CLR 写1清除 GPIO_EXT_COMP_NEG_O_INT 中断。
(WT)

GPIO_EXT_COMP_POS_0_INT_CLR 写 1 清除 GPIO_EXT_COMP_POS_0_INT 中断。
(WT)

GPIO_EXT_COMP_ALL_O_INT_CLR 写1清除 GPIO_EXT_COMP_ALL_O_INT 中断。
(WT)

Register 5.41. GPIO_EXT_VERSION_REG (0x01FC)

(reserved)				GPIO_EXT_DATE																									
31	28	27																											0
0	0	0	0																										0x2312140
																													Reset

GPIO_EXT_DATE 版本控制寄存器。
(R/W)

5.19.4 LP GPIO 矩阵寄存器

本小节的所有地址均为相对于 LP GPIO 交换矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

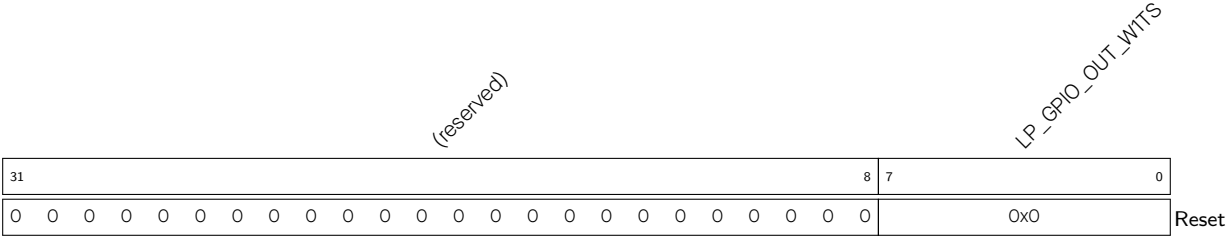
关于保留 (reserved) 域的处理，请查看章节如何配置寄存器的保留域。

Register 5.42. LP_GPIO_OUT_REG (0x0004)

(reserved)																												LP_GPIO_OUT_DATA_ORIG															
31																												7								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																												0x0								Reset							

LP_GPIO_OUT_DATA_ORIG 配置 GPIO0~GPIO6 的输出。
每一位的取值可以为：
0：低电平
1：高电平
Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。
(R/W/WTC)

Register 5.43. LP_GPIO_OUT_W1TS_REG (0x0008)



LP_GPIO_OUT_W1TS 配置是否置位 GPIO0~GPIO6 的输出寄存器 LP_GPIO_OUT_REG。

每一位的取值可以为：

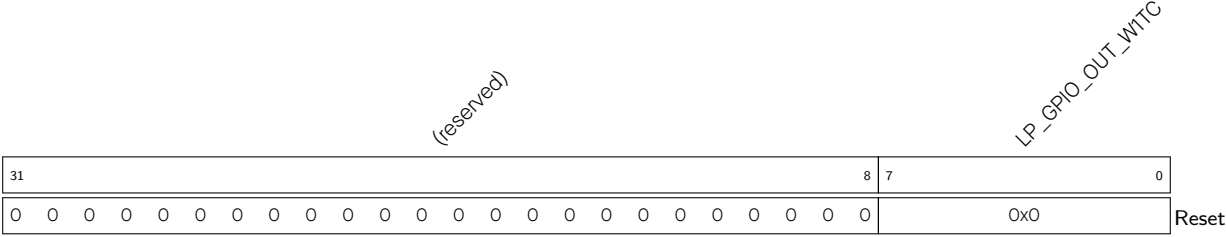
- 0：不置位
- 1：LP_GPIO_OUT_REG 中相应位也将置 1

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来置位 LP_GPIO_OUT_REG。

(WT)

Register 5.44. LP_GPIO_OUT_W1TC_REG (0x000C)



LP_GPIO_OUT_W1TC 配置是否清零 GPIO0~GPIO6 的输出寄存器 LP_GPIO_OUT_REG。

每一位的取值可以为：

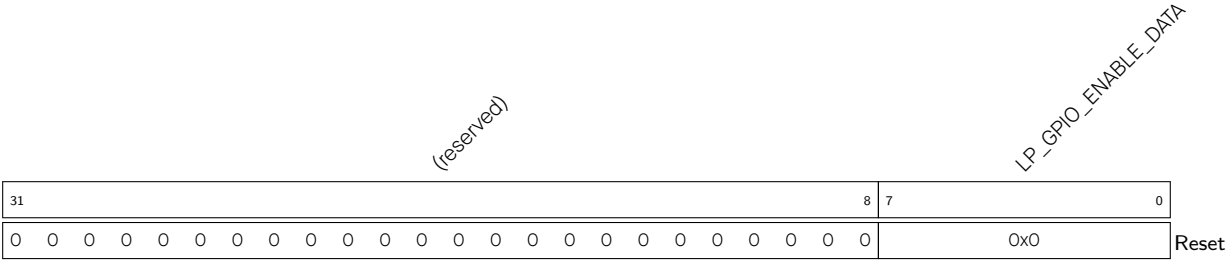
- 0：不清零
- 1：LP_GPIO_OUT_REG 中相应位将被清零。

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来清零 LP_GPIO_OUT_REG。

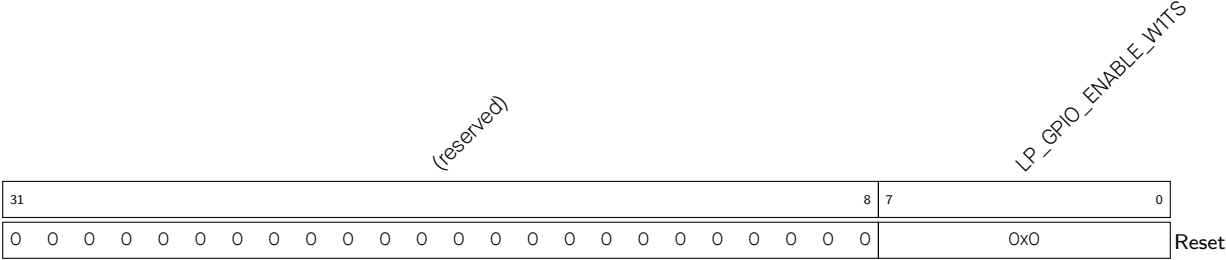
(WT)

Register 5.45. LP_GPIO_ENABLE_REG (0x0010)



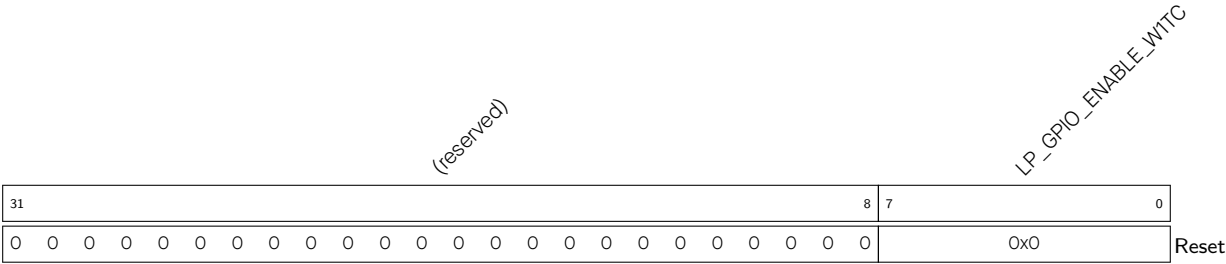
LP_GPIO_ENABLE_DATA 配置是否使能 GPIO0~GPIO6 的输出。
每一位的取值可以为：
0：不使能
1：使能
Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。
(R/W/WTC)

Register 5.46. LP_GPIO_ENABLE_W1TS_REG (0x0014)



LP_GPIO_ENABLE_W1TS 配置是否置位 GPIO0~GPIO6 的输出使能寄存器 [LP_GPIO_ENABLE_REG](#)。
每一位的取值可以为：
0：不置位
1：[LP_GPIO_ENABLE_REG](#) 中相应位也将置 1
Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。
推荐使用此寄存器来置位 [LP_GPIO_ENABLE_REG](#)。
(WT)

Register 5.47. LP_GPIO_ENABLE_W1TC_REG (0x0018)



LP_GPIO_ENABLE_W1TC 配置是否清零 GPIO0~GPIO6 的输出使能寄存器 **LP_GPIO_ENABLE_REG**。

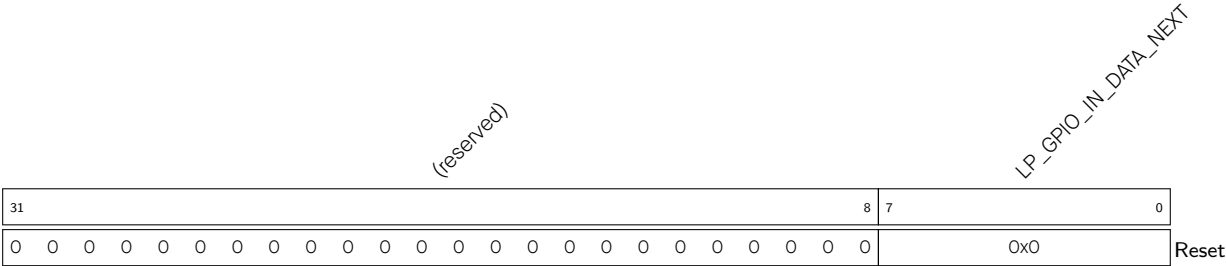
每一位的取值可以为：

- 0：不清零
- 1： **LP_GPIO_ENABLE_REG** 中相应位将被清零

推荐使用此寄存器来清零 **LP_GPIO_ENABLE_REG**。

(WT)

Register 5.48. LP_GPIO_IN_REG (0x001C)



LP_GPIO_IN_DATA_NEXT 表示 GPIO0~GPIO6 的输入值。

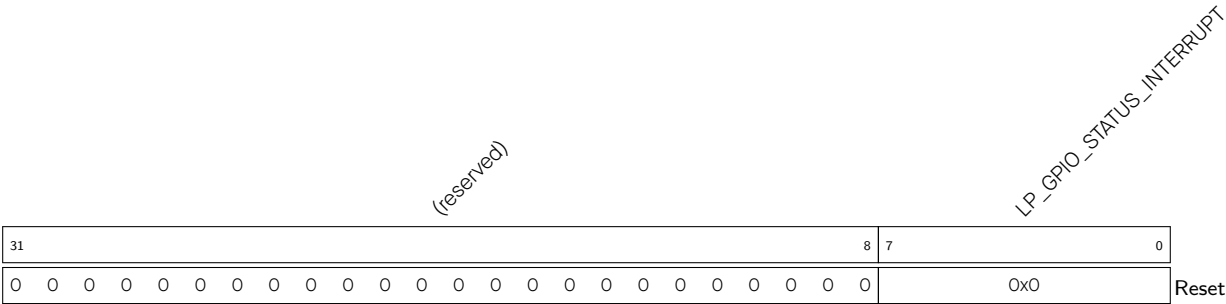
每一位均代表一个管脚的输入值：

- 0：低电平
- 1：高电平

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

(RO)

Register 5.49. LP_GPIO_STATUS_REG (0x0020)



LP_GPIO_STATUS_INTERRUPT 配置 GPIO0~GPIO6 的中断状态。

每一位的取值可以为：

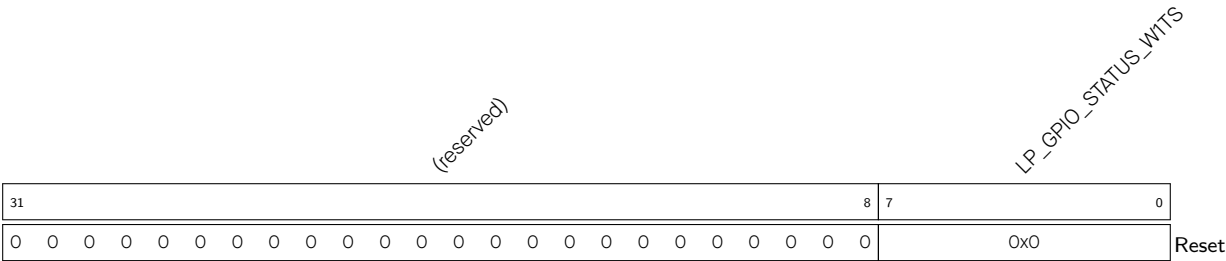
- 0：没有中断
- 1：已触发中断

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

该字段与 LP_GPIO_PIN_n_REG 中的 LP_GPIO_PIN_n_INT_TYPE 配合使用。

(R/W/WTC)

Register 5.50. LP_GPIO_STATUS_W1TS_REG (0x0024)



LP_GPIO_STATUS_W1TS 配置是否置位 GPIO0~GPIO6 的中断状态寄存器 [LP_GPIO_STATUS_INT](#)。

每一位的取值可以为：

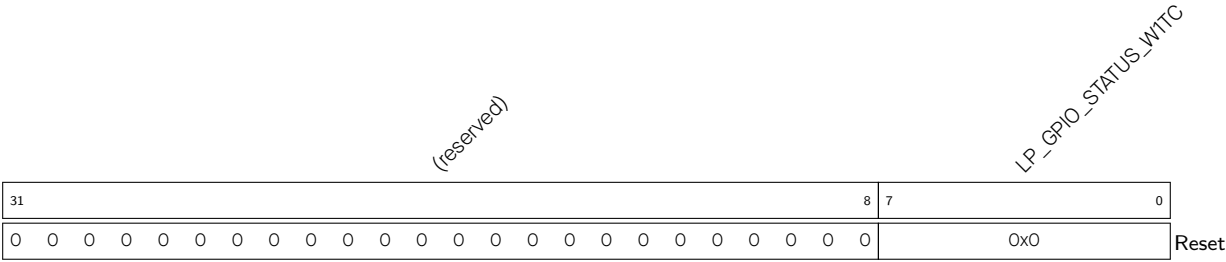
- 0：不置位
- 1：[LP_GPIO_STATUS_INT](#) 中的相应位也将置 1

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来置位 [LP_GPIO_STATUS_INT](#)。

(WT)

Register 5.51. LP_GPIO_STATUS_W1TC_REG (0x0028)



LP_GPIO_STATUS_W1TC 配置是否清零 GPIO0~GPIO6 的中断状态寄存器 [LP_GPIO_STATUS_INT](#)。

每一位的取值可以为：

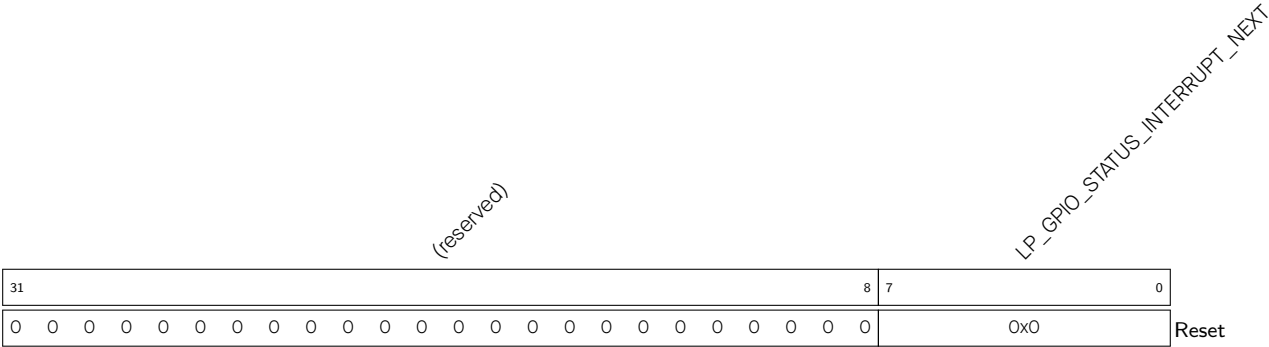
- 0：不清零
- 1： [LP_GPIO_STATUS_INT](#) 中相应位将被清除

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

推荐使用此寄存器来清零 [LP_GPIO_STATUS_INT](#)。

(WT)

Register 5.52. LP_GPIO_STATUS_NEXT_REG (0x002C)



LP_GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0~GPIO6 的中断源状态。

Bit[0]~bit[6] 分别对应 GPIO0~GPIO6。

每一位代表：

- 0：中断源状态无效
- 1：中断源状态无效

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。

(RO)

Register 5.53. LP_GPIO_PIN n _REG (n : 0-6) (0x0030+0x4* n)

(reserved)																								LP_GPIO_PIN0_WAKEUP_ENABLE		LP_GPIO_PIN0_INT_TYPE		(reserved)		LP_GPIO_PIN0_EDGE_WAKEUP_CLR		LP_GPIO_PIN0_SYNC1_BYPASS		LP_GPIO_PIN0_PAD_DRIVER		LP_GPIO_PIN0_SYNC2_BYPASS			
31																								11		10	9	7		6	5	4	3	2	1	0			
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0		0x0		0		0		0x0		0		0x0		Reset	

LP_GPIO_PIN n _SYNC2_BYPASS 配置是否使能 GPIO 输入数据在第二级同步中 LP IO MUX 运行时
钟上升沿同步或下降沿同步。
0: 不同步
1: 在下降沿进行同步
2: 在上升沿进行同步
3: 在上升沿进行同步
(R/W)

LP_GPIO_PIN n _PAD_DRIVER 配置选择管脚的输出驱动模式。
0: 正常输出
1: 开漏输出
(R/W)

LP_GPIO_PIN n _SYNC1_BYPASS 配置是否使能 GPIO 输入数据在第一级同步中 LP IO MUX 运行时
钟上升沿同步或下降沿同步。
0: 不同步
1: 在下降沿进行同步
2: 在上升沿进行同步
3: 在上升沿进行同步
(R/W)

LP_GPIO_PIN n _EDGE_WAKEUP_CLR 配置是否清除 GPIO0~GPIO6 的跳变沿唤醒状态。
0: 无效
1: 清除
(WT)

LP_GPIO_PIN n _INT_TYPE 配置 GPIO n 的中断类型。
0: 关闭 GPIO 中断
1: 上升沿触发
2: 下降沿触发
3: 任一沿触发
4: 低电平触发
5: 高电平触发
(R/W)

见下页

Register 5.53. LP_GPIO_PINn_REG (n: 0-6) (0x0030+0x4*n)

接上页

LP_GPIO_PINn_WAKEUP_ENABLE 配置是否使能 GPIO n 唤醒功能。

0: 不使能

1: 使能

PD_LP_PERI 域掉电后，该唤醒功能将被禁用。

(R/W)

Register 5.54. LP_GPIO_FUNCn_OUT_SEL_CFG_REG (n: 0-6) (0x02B0+0x4*n)

(reserved)																															LP_GPIO_FUNCn_OE_INV_SEL (reserved) LP_GPIO_FUNCn_OUT_INV_SEL																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

LP_GPIO_FUNCn_OUT_INV_SEL 配置是否反转输出值。

0: 不反转

1: 反转

(R/W)

LP_GPIO_FUNCn_OE_INV_SEL 配置是否反转输出使能信号。

0: 不反转

1: 反转

(R/W)

Register 5.55. LP_GPIO_CLOCK_GATE_REG (0x03F8)

(reserved)																															LP_GPIO_CLK_EN		
31																															1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

LP_GPIO_CLK_EN 配置是否使能时钟门控。

0: 不使能

1: 使能

(R/W)

Register 5.56. LP_GPIO_DATE_REG (0x03FC)

(reserved)				LP_GPIO_DATE																								
31	28	27																										0
0	0	0	0																									Reset

LP_GPIO_DATE 版本控制寄存器。
(R/W)

5.19.5 LP IO MUX 寄存器

本小节的所有地址均为相对于 LP IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节[如何配置寄存器的保留域](#)。

Register 5.57. LP_IO_MUX_GPIOn_REG (n: 0-6) (0x0000+0x4*n)

(reserved)																		LP_IO_MUX_GPION_HYS_SEL LP_IO_MUX_GPION_HYS_EN LP_IO_MUX_GPION_FILTER_EN LP_IO_MUX_GPION_MCU_SEL LP_IO_MUX_GPION_FUN_DRV LP_IO_MUX_GPION_FUN_IE LP_IO_MUX_GPION_FUN_WPU LP_IO_MUX_GPION_MCU_WPD LP_IO_MUX_GPION_MCU_DRV LP_IO_MUX_GPION_MCU_IE LP_IO_MUX_GPION_MCU_WPU LP_IO_MUX_GPION_SLP_SEL LP_IO_MUX_GPION_MCU_OE															
31															18	17	16	15	14		12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset							

LP_IO_MUX_GPIOn_MCU_OE 配置是否使能睡眠模式下 GPIOn 的输出。
0: 不使能
1: 使能
(R/W)

LP_IO_MUX_GPIOn_SLP_SEL 配置是否使能 GPIOn 进入睡眠模式。
0: 不使能
1: 使能
(R/W)

LP_IO_MUX_GPIOn_MCU_WPD 配置是否使能睡眠模式下 GPIOn 的下拉电阻。
0: 不使能
1: 使能
(R/W)

LP_IO_MUX_GPIOn_MCU_WPU 配置是否使能睡眠模式下 GPIOn 的上拉电阻。
0: 不使能
1: 使能
(R/W)

LP_IO_MUX_GPIOn_MCU_IE 配置是否使能睡眠模式下 GPIOn 的输入。
0: 不使能
1: 使能
(R/W)

LP_IO_MUX_GPIOn_MCU_DRV 配置睡眠模式下 GPIOn 的驱动强度。
0: ~5 mA
1: ~10 mA
2: ~20 mA
3: ~40 mA
(R/W)

见下页

Register 5.57. LP_IO_MUX_GPIO n _REG (n : 0-6) (0x0000+0x4* n)

接上页

LP_IO_MUX_GPIO n _FUN_WPD 配置是否使能 GPIO n 的下拉电阻。

0: 不使能

1: 使能

(R/W)

LP_IO_MUX_GPIO n _FUN_WPU 配置是否使能 GPIO n 的上拉电阻。

0: 不使能

1: 使能

(R/W)

LP_IO_MUX_GPIO n _FUN_IE 配置是否使能 GPIO n 的输入。

0: 不使能

1: 使能

(R/W)

LP_IO_MUX_GPIO n _FUN_DRV 配置 GPIO n 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

LP_IO_MUX_GPIO n _MCU_SEL 配置选择 GPIO n 的 LP IO MUX 功能。

0: 选择功能 0

1: 选择功能 1

.....

(R/W)

LP_IO_MUX_GPIO n _FILTER_EN 配置是否使能管脚输入信号的滤波功能。

0: 不使能

1: 使能

(R/W)

LP_IO_MUX_GPIO n _HYS_EN 当 LP_IO_MUX_GPIO n _HYS_SEL 设置为 1 时，配置是否使能管脚的迟滞功能。

0: 不使能

1: 使能

(R/W)

见下页

Register 5.57. LP_IO_MUX_GPIOn_REG (n: 0-6) (0x0000+0x4*n)

接上页

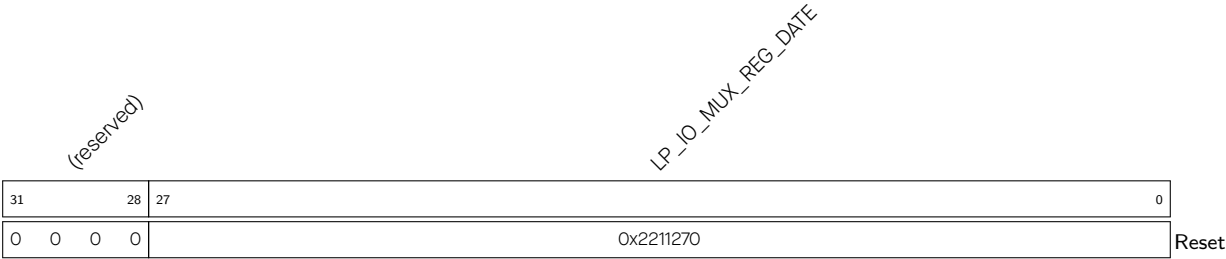
LP_IO_MUX_GPIOn_HYS_SEL 配置选择 GPIOn 迟滞功能的使能信号。

0: 使用 eFuse 的输出使能信号

1: 使用 LP_IO_MUX_GPIOn_HYS_EN 用作输出使能信号

(R/W)

Register 5.58. LP_IO_MUX_DATE_REG (0x01FC)



LP_IO_MUX_REG_DATE 版本控制寄存器。

(R/W)

第 6 章

复位和时钟

6.1 复位

6.1.1 概述

ESP32-C5 提供四种级别的复位方式，分别是 CPU 复位 (CPU reset)、内核复位 (Core reset)、系统复位 (System reset) 和芯片复位 (Chip reset)。除芯片复位外，其他复位方式不影响片上内存存储的数据。图 6.1-1 展示了整个芯片系统的结构以及四种复位等级。

6.1.2 结构图

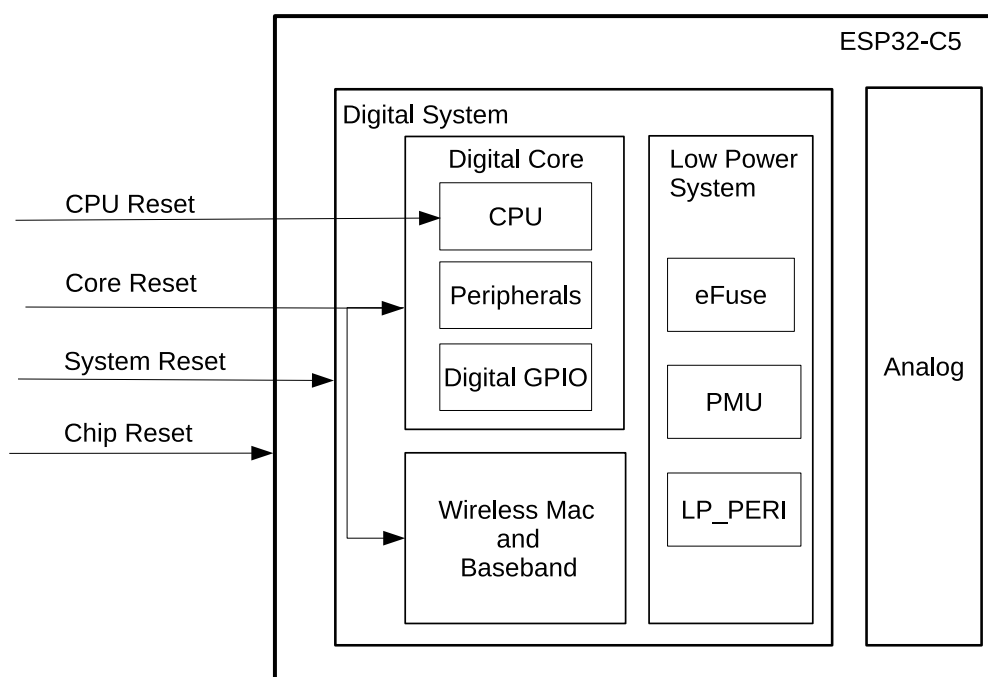


图 6.1-1. 四种复位等级

ESP32-C5 的数字系统分为两部分：高性能系统 (High Performance System, HP system) 和 低功耗系统 (Low Power System, LP system)。其中高性能系统包含数字内核 (Digital Core)、无线 MAC 和基带 (Wireless Mac and Baseband) 和 HP SRAM，低功耗系统中仅包含一些低功耗外设和 LP SRAM。详细信息可参考图 6.1-1 (HP SRAM 和 LP SRAM 不会复位，因此未在图中标注)。

6.1.3 特性

- 支持四种复位等级：

- CPU 复位：复位 CPU 核。复位释放后，程序将从 CPU Reset Vector (0x40000000) 开始执行；
 - 内核复位：复位除低功耗系统以外的其他数字系统，包括 CPU、外设、数字 GPIO、无线 MAC 和基带；
 - 系统复位：复位包括低功耗系统在内的整个数字系统；
 - 芯片复位：复位包括数字系统和模拟系统在内的整个芯片。
- 支持软件复位和硬件复位：
 - 软件复位：CPU 配置相关寄存器可触发软件复位，见章节 4 低功耗管理 [to be added later]；
 - 硬件复位：硬件复位直接由硬件电路触发。

6.1.4 功能描述

上述任一复位发生时，CPU 将立刻复位。复位释放后，CPU 可通过读取 LP_CLKRST_RESET_CAUSE 获取复位源。可通过配置 LP_CLKRST_CORE0_RESET_CAUSE_CLR 清除当前 LP_CLKRST_RESET_CAUSE 寄存器记录的复位源。

表 6.1-1 列出了从上述寄存器中可能读出的复位源及其触发的复位等级。当有多个复位源同时生效时，LP_CLKRST_RESET_CAUSE 中的复位源为表 6.1-1 中最靠上的一个生效复位源。当有多个复位源生效，但不在同一时间生效时，LP_CLKRST_RESET_CAUSE 记录的为最近发生的一次复位的复位源。

表 6.1-1. 复位源

编码	复位源	复位等级	说明
0x1A	CPU 锁死复位	CPU 复位	—
0x01	芯片复位 ¹	芯片复位	—
0x12	超级看门狗复位	系统复位	详见章节 11 看门狗定时器 (WDT)
0x19	电源毛刺复位	系统复位	详见章节 15 电源检测器
0x0F	欠压系统复位	芯片复位或系统复位	欠压检测器触发的复位 ²
0x10	RWDT 系统复位	系统复位	详见章节 11 看门狗定时器 (WDT)
0x09	RWDT 内核复位	内核复位	详见章节 11 看门狗定时器 (WDT)
0x14	eFuse 复位	内核复位	eFuse CRC 校验错误触发复位
0x07	MWDT0 内核复位	内核复位	详见章节 11 看门狗定时器 (WDT)
0x08	MWDT1 内核复位	内核复位	详见章节 11 看门狗定时器 (WDT)
0x15	USB (UART) 复位	内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 Serial 接口将触发此复位，详见章节 28 USB 串口/JTAG 控制器
0x16	USB (JTAG) 复位	内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 JTAG 接口将触发此复位，详见章节 28 USB 串口/JTAG 控制器
0x03	软件系统复位	内核复位	配置 LP_AON_HPSYS_SW_RESET 触发
0x0D	RWDT CPU 复位	CPU 复位	详见章节 11 看门狗定时器 (WDT)
0x0C	软件 CPU 复位	CPU 复位	配置 LP_AON_CPU_CORE0_SW_RESET 触发
0x0B	MWDT0 CPU 复位	CPU 复位	详见章节 11 看门狗定时器 (WDT)
0x11	MWDT1 CPU 复位	CPU 复位	详见章节 11 看门狗定时器 (WDT)
0x18	JTAG CPU 复位	CPU 复位	接收“JDB 复位 CPU”指令时触发

见下页

表 6.1-1 – 接上页

编码	复位源	复位等级	说明
0x05	Deep-sleep 复位	内核复位	详见章节 4 低功耗管理 [to be added later]

¹ 芯片复位的触发源包括以下两项：

- 芯片上电触发芯片复位
- 欠压检测器触发芯片复位

² 欠压检测器在检测到欠压状态时，将根据寄存器配置，选择触发系统复位或者芯片复位，详见章节 4 低功耗管理 [\[to be added later\]](#)。

6.1.5 外设复位

外设可单独复位，也可通过配置内核复位、系统复位、芯片复位连带复位。在解复位后，需要读取各外设的解复位完成指示寄存器，此类寄存器为 1 表示对应外设解复位成功，否则外设可能无法正常工作。

ESP32-C5 高性能系统的外设复位寄存器已整合到 PCR (POWER/CLOCK/RESET) 模块，低功耗系统的外设复位寄存器在低功耗系统时钟复位寄存器中。可通过章节 6.4 寄存器列表 查看各外设对应的复位寄存器以及解复位成功指示寄存器。

6.2 时钟

6.2.1 概述

ESP32-C5 的时钟主要来源于振荡器 (oscillator, OSC)、RC 振荡电路和 PLL 时钟生成电路。上述时钟源产生的时钟经时钟分频器或时钟选择器等时钟模块的处理，使得大部分功能模块可以根据不同功耗和性能需求来获取及选择对应频率的工作时钟。图 6.2-1 为系统时钟结构。

6.2.2 结构图

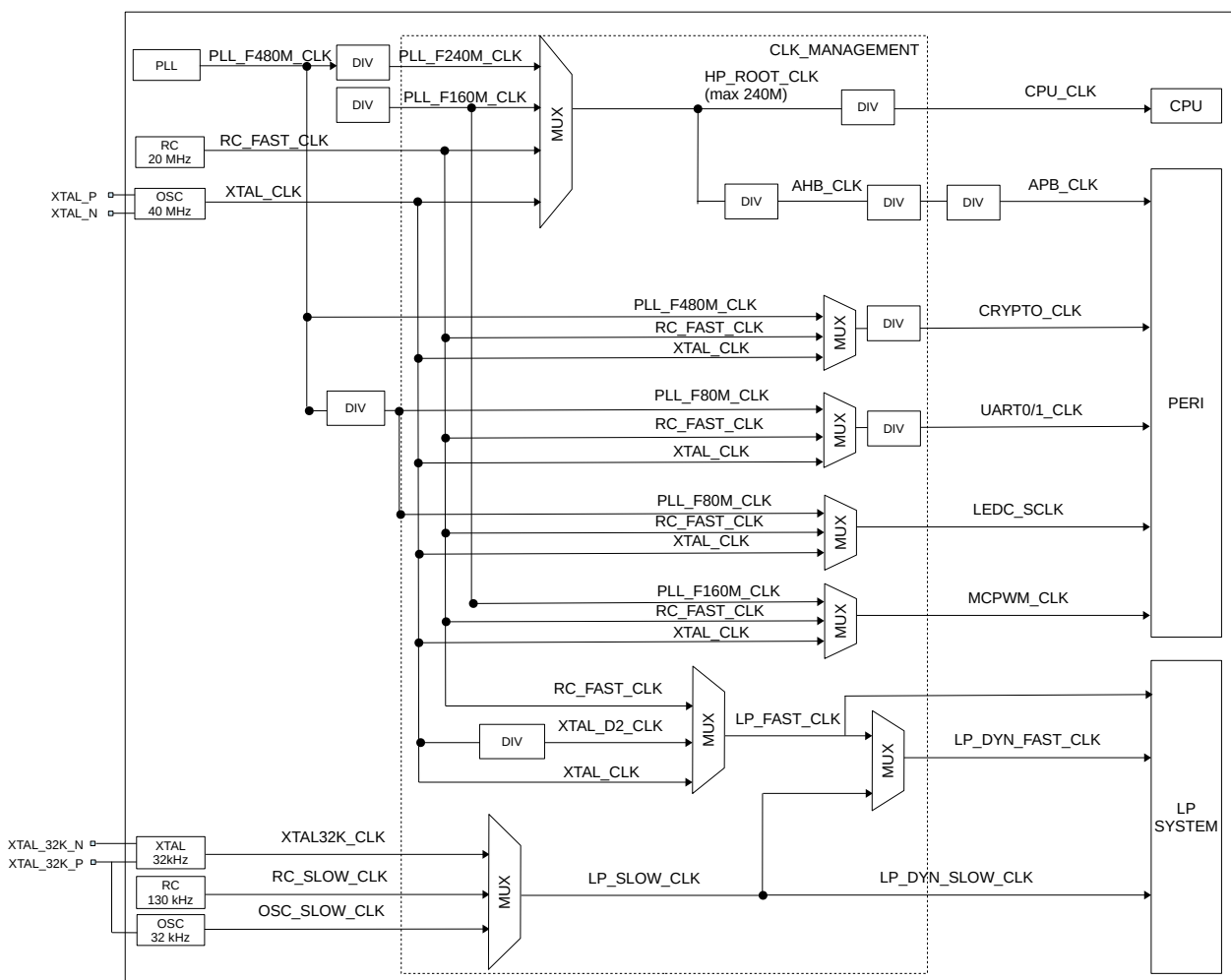


图 6.2-1. 系统时钟

6.2.3 特性

ESP32-C5 的时钟根据频率不同，可分为：

- 高性能时钟，主要为 CPU 和数字外设提供工作时钟
 - PLL_F480M_CLK: 480 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）
 - XTAL_CLK: 48 MHz/40 MHz 外部晶振时钟
- 低功耗时钟，主要为低功耗系统以及部分处于低功耗模式的外设提供工作时钟
 - XTAL32K_CLK: 32 kHz 外部晶振时钟
 - RC_FAST_CLK: 内置快速 RC 振荡器时钟，频率可调节（通常为 20 MHz）
 - RC_SLOW_CLK: 内置慢速 RC 振荡器，通常频率为 130 kHz
 - OSC_SLOW_CLK: 外置低速时钟，通常频率为 32 kHz，通过 [XTAL_32K_P](#) 输入，在配置好这个 GPIO 的状态后需要配置 Hold 功能，具体请参考章节 [5 GPIO 交换矩阵和 IO MUX \(GPIO, IO MUX\)](#) > [5.9 GPIO 管脚的 Hold 特性](#)

ESP32-C5 可选 40 MHz 和 48 MHz 晶振频率。启动模式与 MTMS、EFUSE_XTAL_48M_SEL 和 EFUSE_XTAL_48M_SEL_MODE 共同控制晶振频率。

表 6.2-1. 晶振频率选择

晶振频率	启动模式	eFuse 1 ^a	eFuse 2 ^b	MTMS
40 MHz	SPI Boot	x	偶数个 1	x ^d
48 MHz			奇数个 1	x
40 MHz	Joint Download Boot	1	偶数个 1	x
48 MHz			奇数个 1	x
40 MHz		0	x	0 ^c
48 MHz				1 ^c

^a eFuse 1: EFUSE_XTAL_48M_SEL_MODE

^b eFuse 2: EFUSE_XTAL_48M_SEL[2:0]

^c 在 **Joint Download Boot** 模式下，用户需要指定 strapping 管脚 MTMS 的电平。

^d x: x 表示该值被忽略，任何取值不影响该状态。

6.2.4 功能描述

6.2.4.1 高性能系统时钟

如图 6.2-1 所示，CPU_CLK 为 CPU 主时钟。CPU 主频最高可以达到 240 MHz，并且能够在超低频下工作（通常为 2 MHz），以减少功耗。CPU_CLK 与 AHB_CLK、APB_CLK 共用一个时钟源。用户可配置 PCR_SOC_CLK_SEL 选择 XTAL_CLK、PLL_240M_CLK、PLL_160M_CLK 或 RC_FAST_CLK 作为 CPU_CLK 的时钟源，具体请参考表 6.2-2 和表 6.2-3。默认状态下，CPU 的时钟源为 XTAL_CLK，且分频系数为 1 分频。

表 6.2-2. CPU_CLK 时钟源选择

PCR_SOC_CLK_SEL	时钟源
0	XTAL_CLK
1	RC_FAST_CLK
2	PLL_F160M_CLK
3	PLL_F240M_CLK

表 6.2-3. CPU_CLK、AHB_CLK 和 HP_ROOT_CLK 的时钟频率

时钟名称	时钟源	时钟频率
HP_ROOT_CLK	PLL_F240M_CLK	240 MHz
	PLL_F160M_CLK	160 MHz
	XTAL_CLK	48 MHz/40 MHz
	RC_FAST_CLK	20 MHz
CPU_CLK ¹	HP_ROOT_CLK	$f_{\text{HP\_ROOT\_CLK}} / (\text{PCR\_CPU\_DIV\_NUM} + 1)$
AHB_CLK ²	HP_ROOT_CLK	$f_{\text{HP\_ROOT\_CLK}} / (\text{PCR\_AHB\_DIV\_NUM} + 1)$

¹ CPU_CLK 频率必须大于等于 AHB_CLK 频率，并且必须为 AHB_CLK 频率的整数倍

² AHB_CLK 时钟频率不能超过 XTAL_CLK 频率

说明:

配置 HP_ROOT_CLK 的时钟源选择寄存器或者 CPU_CLK、AHB_CLK 的分频寄存器后，需要额外配置 PCR_BUS_CLOCK_UPDATE 寄存器将上述寄存器配置生效，可通过读取 PCR_BUS_CLOCK_UPDATE 寄存器来判断上述配置是否成功生效。

如图 6.2-1 所示，APB_CLK 是在 AHB_CLK 基础上经过两级分频得到的。其中第一级分频为恒定分频，即始终按照分频系数 (PCR_APB_DIV_NUM + 1) 的值进行分频。第二级分频为自动降频，如果芯片中的主机没有访问外设寄存器的请求时，会以分频系数 (APB_DECREASE_DIV_NUM + 1) 进行二次分频产生 APB_CLK，从而达到节约功耗的目的。但当芯片中的主机发起了访问外设寄存器的请求时，APB_CLK 将恢复至一级分频后的频率。

用户可将 APB_DECREASE_DIV_NUM 设置为 0 以禁用自动降频功能。默认情况下，该功能禁用。

6.2.4.2 低功耗系统时钟

低功耗系统能够在大多数时钟源关闭的状态下工作。低功耗系统时钟包括 LP_SLOW_CLK 时钟和 LP_FAST_CLK 时钟。

LP_SLOW_CLK 和 LP_FAST_CLK 的时钟源为低频时钟，其中：

- LP_SLOW_CLK 时钟有三种可能的时钟源：
 - RC_SLOW_CLK
 - XTAL32K_CLK
 - OSC_SLOW_CLK
- LP_FAST_CLK 时钟有三种可能的时钟源：
 - XTAL_CLK 的 2 分频时钟 XTAL_D2_CLK (20 MHz/24 MHz)
 - RC_FAST_CLK
 - XTAL_CLK

LP_DYN_SLOW_CLK 时钟源为 LP_SLOW_CLK，频率同时钟源频率。

LP_DYN_FAST_CLK 根据芯片电源的模式（详见章节 4 低功耗管理 [to be added later]）选择时钟源：

- 芯片电源为 Active、Modem-sleep 模式时始终选择 LP_FAST_CLK 作为时钟源
- 芯片电源为 Light-sleep、Deep-sleep 模式时选择 LP_SLOW_CLK 作为时钟源

6.2.4.3 外设时钟

表 6.2-4、表 6.2-5、表 6.2-6、和表 6.2-7 分别列出了接入各个外设时钟的衍生高性能时钟源、接入高性能系统各个外设的时钟、衍生低功耗时钟源、以及接入低功耗系统各个外设的时钟。

表 6.2-4. 衍生高性能时钟源

衍生时钟	源时钟		衍生时钟			源时钟				衍生时钟								源时钟
	XTAL_CLK 48/40 MHz	PLL_F480M_CLK 480 MHz	PLL_F240M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	RC_FAST_CLK 20 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	HP_ROOT_CLK 240 MHz/160 MHz/48/(40) MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 24/20 MHz	LP_FAST_CLK	Clock from IO
PLL_F240M_CLK		Y																
PLL_F160M_CLK		Y																
PLL_F120M_CLK		Y																
PLL_F80M_CLK		Y																
PLL_F48M_CLK		Y																
PLL_F40M_CLK		Y																
HP_ROOT_CLK	Y		Y	Y	Y													
CRYPTO_CLK	Y	Y			Y													
APB_CLK									Y									
AHB_CLK									Y									
CPU_CLK									Y									

表 6.2-5. 高性能系统外设时钟

外设	源时钟	衍生时钟					源时钟	HP_ROOT_CLK 240 MHz/160 MHz/48/(40) MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	衍生时钟				LP_FAST_CLK	Clock from IO
	XTAL_CLK 48/40 MHz	PLL_F240M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	PLL_F120M_CLK 120 MHz	PLL_F80M_CLK 80 MHz	PLL_F48M_CLK 48 MHz	RC_FAST_CLK 20 MHz					LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 24/20 MHz			
定时器组 (TIMG)	Y				Y		Y										
主系统看门狗定时器 (MWDT)	Y				Y		Y										
I2S 控制器 (I2S)	Y	Y	Y													I2S_MCLK_in	
UART 控制器 (UART)	Y				Y		Y										
红外遥控 (RMT)	Y					Y											
电机控制脉宽调制器 (MCPWM)	Y		Y				Y										
I2C 控制器 (I2C)	Y						Y										
SPI2	Y		Y	Y			Y										
SAR ADC	Y				Y		Y										
USB 串口/JTAG 控制器						Y											
控制器局域网灵活数据速率 [to be added later]	Y						Y										
LED PWM 控制器 (LEDC)	Y	Y				Y											
系统定时器	Y						Y										
并行 IO 控制器 (PARLIO)	Y	Y					Y									parl_rx_clk_in parl_tx_clk_in	
IO MUX	Y				Y		Y										
AES 加速器 (AES)									Y								
SHA 加速器 (SHA)																	
RSA 加速器 (RSA)																	
ECC 加速器 (ECC)																	
数字签名算法 (DSA)																	
HMAC 加速器 (HMAC)																	
椭圆曲线数字签名算法 (ECDSA)																	
中断矩阵 [to be added later]											Y						
SDIO 从机控制器 (SDIO)											Y						
脉冲计数控制器 (PCNT) [to be added later]										Y							
事件任务矩阵 (ETM)											Y						
通用 DMA 控制器 (GDMA)											Y						
UHCI											Y						
辅助调试												Y					
RISC-V 追踪编码器 (TRACE)											Y	Y					
中断优先级寄存器 (INTPRI)												Y					

表 6.2-6. 衍生低功耗时钟源

	源时钟		衍生时钟		源时钟				衍生时钟								源时钟	
衍生时钟	XTAL_CLK 48/40 MHz	PLL_F480M_CLK 480 MHz	PLL_F240M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	RC_FAST_CLK 20 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	HP_ROOT_CLK 240 MHz/160 MHz/48/(40) MHz/20 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 24/20 MHz	LP_FAST_CLK	Clock from IO
LP_DYN_FAST_CLK						Y	Y	Y									Y	
LP_DYN_SLOW_CLK						Y	Y	Y										
XTAL_D2_CLK	Y																	
LP_FAST_CLK	Y				Y											Y		

表 6.2-7. 低功耗系统外设时钟

衍生时钟	源时钟		衍生时钟		源时钟				HP_ROOT_CLK 240 MHz/160 MHz/48(/40) MHz/20 MHz	衍生时钟								源时钟 Clock from IO
	XTAL_CLK 48/40 MHz	PLL_F480M_CLK 480 MHz	PLL_F120M_CLK 240 MHz	PLL_F160M_CLK 160 MHz	RC_FAST_CLK 20 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz		CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_FAST_CLK	LP_DYN_SLOW_CLK	XTAL_D2_CLK 24/20 MHz	LP_FAST_CLK	
eFuse 控制器 (eFuse)																Y		
RTC 看门狗定时器 (RWDI)														Y	Y			
RTC 定时器 (RTC Timer)														Y	Y			
欠压检测器 (Brownout Detector)														Y				
电源管理单元 (PMU)														Y	Y		Y	

PLL_F480M_CLK 时钟

PLL_F480M_CLK 的频率是 480 MHz。PLL_F240M_CLK、PLL_F160M_CLK、PLL_F80M_CLK、PLL_F48M_CLK、PLL_F40M_CLK 均为 PLL_F480M_CLK 分频所得。

CRYPTO_CLK 时钟

如表 6.2-4 所示，可选择 XTAL_CLK、PLL_F480M_CLK 或 RC_FAST_CLK 作为 CRYPTO_CLK 的时钟源，CRYPTO_CLK 时钟频率最大为 160 MHz。

为了防止 DPA (Differential Power Analysis) 攻击加解密外设，加解密功能时钟采用随机分频策略。根据随机分频范围划分为四个安全等级。可通过配置 HP_SYSTEM_SEC_DPA_CONF_REG 选择安全等级。当 HP_SYSTEM_SEC_DPA_CFG_SEL 为 1 时，安全等级由 eFuse 中的配置信息 (EFUSE_SEC_DPA_LEVEL) 决定，否则由 HP_SYSTEM_SEC_DPA_LEVEL 的值决定。

LED_PWM 时钟

LEDC 模块能将 PLL_F80M_CLK、RC_FAST_CLK 以及 XTAL_CLK 作为时钟源使用，即在 APB_CLK 关闭的时候，LEDC 也可工作。换言之，当系统处于低功耗模式时，其它外设都将停止工作 (APB_CLK 关闭)，但是 LEDC 仍然可以通过 RC_FAST_CLK 来正常工作。

TIMGO 定时器组慢速时钟

TIMGO 定时器组可以以 XTAL_CLK 为基准时钟，计算 ESP32-C5 提供的慢速时钟的实际频率。慢速时钟源可以通过配置 PCR_32K_SEL 进行选择，对应关系如下：

- 0：选择 XTAL32K_CLK 时钟
- 1：选择 OSC_SLOW_CLK 时钟
- 2：选择 RC_SLOW_CLK 时钟
- 3：选择 RC_FAST_CLK 时钟

低功耗系统时钟

低功耗系统外设的时钟相关寄存器在低功耗系统时钟复位寄存器中。可通过章节 6.4.2 低功耗系统时钟寄存器列表 查看各外设对应的时钟相关寄存器。

6.2.4.4 PMU 控制高性能系统时钟门控

在 ESP32-C5 的各个工作模式下，可提前配置以下寄存器字段，实现 电源管理单元 (PMU) 对高性能系统外设的时钟门控：

- PMU_HP_x_DIG_ICG_APB_EN (x = ACTIVE/MODEM/SLEEP)：控制高性能系统外设读写寄存器的时钟门控
- PMU_HP_x_DIG_ICG_FUNC_EN (x = ACTIVE/MODEM/SLEEP)：控制高性能系统外设的功能时钟门控。

具体配置流程请参考章节 低功耗管理 [to be added later]。

表 6.2-8 和表 6.2-9 列出了 PMU 预配置寄存器位与高性能系统时钟门控的对应关系。

表 6.2-8. 控制模块的读写寄存器时钟门控与 PMU 相关寄存器映射关系

PMU_HP_x_DIG_ICG_APB_EN bit 位	控制模块的读写寄存器时钟门控
0	AES 加速器 (AES) SHA 加速器 (SHA) RSA 加速器 (RSA) ECC 加速器 (ECC) 数字签名算法 (DSA) HMAC 加速器 (HMAC) 椭圆曲线数字签名算法 (ECDSA)
1	通用 DMA 控制器 (GDMA)
2	SPI2
3	中断矩阵 [to be added later]
4	I2S 控制器 (I2S)
6	UART0
7	UART1
8	UHCI
11	定时器组 0
12	定时器组 1
13	I2C 控制器 (I2C)
14	LED PWM 控制器 (LEDC)
15	红外遥控 (RMT)
16	系统定时器
17	USB 串口/JTAG 控制器
18	双线汽车接口 0
19	双线汽车接口 1
20	脉冲计数控制器 (PCNT) [to be added later]
21	电机控制脉宽调制器 (MCPWM)
22	事件任务矩阵 (ETM)
23	并行 IO 控制器 (PARLIO)
25	辅助调试
26	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)
28	比特调节器

表 6.2-9. 控制模块的工作时钟门控与 PMU 相关寄存器映射关系

PMU_HP_x_DIG_ICG_FUNC_EN bit 位	控制模块的功能时钟门控
0	通用 DMA 控制器 (GDMA)
1	SPI2
2	I2S 接收侧
3	UART0
4	UART1
5	UHCI
6	USB 串口/JTAG 控制器
7	I2S 发送
10	MEM_MONITOR
11	SDIO 从机控制器 (SDIO)
12	温度传感器
13	定时器组 0
14	定时器组 1
16	事件任务矩阵 (ETM)
17	高性能处理器 [to be added later] ASSIST_DEBUG
18	系统定时器
19	AES 加速器 (AES) SHA 加速器 (SHA) RSA 加速器 (RSA) ECC 加速器 (ECC) 数字签名算法 (DSA) HMAC 加速器 (HMAC) 椭圆曲线数字签名算法 (ECDSA)
21	红外遥控 (RMT)
22	电机控制脉宽调制器 (MCPWM)
24	比特调节器
25	并行 IO 控制器 (PARLIO)
27	LED PWM 控制器 (LEDC)
28	GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)
29	I2C 控制器 (I2C)
30	双线汽车接口 0
31	双线汽车接口 1

6.3 配置流程

6.3.1 高性能系统时钟配置

当配置 HP_ROOT_CLK 的时钟选源 `PCR_SOC_CLK_SEL`，或 CPU_CLK 时钟分频 `PCR_CPU_DIV_NUM`，或 AHB_CLK 时钟分频 `PCR_AHB_DIV_NUM` 时，在配置完上述寄存器后，需要将 `PCR_BUS_CLOCK_UPDATE` 寄存器写 1 来生效上述配置。可通过读 `PCR_BUS_CLOCK_UPDATE` 寄存器判断上述配置是否更新完成，当为 0 时表示配置更新完成，为 1 表示未更新完成。

6.3.2 低功耗系统时钟配置

可配置 `LP_CLKRST_SLOW_CLK_SEL` 选择 `LP_SLOW_CLK` 的时钟源。

可配置 `LP_CLKRST_FAST_CLK_SEL` 选择 `LP_FAST_CLK` 的时钟源。

6.3.3 外设时钟复位配置

注意：

ESP32-C5 具有低功耗特性，因此有些外设时钟默认为关闭状态。在启用这些外设之前，必须将外设的时钟打开，即把相应的 `CLK_EN` 位置为 1，并且解除外设的复位状态，即把相应的 `RST_EN` 位置为 0。

大部分外设模块的时钟都可以分为总线时钟和功能时钟。

- 外设模块的总线时钟主要是用于配置外设模块的寄存器。
- 功能时钟主要是模块工作所使用的时钟，例如 UART 的参考时钟。

大部分模块的功能时钟可以选择多个时钟源。在门控寄存器的描述中会说明该寄存器属于总线（`AHB_CLK`、`APB_CLK`）时钟门控寄存器还是功能时钟门控寄存器。

ESP32-C5 的总线时钟开关、功能时钟的开关以及时钟源选择和时钟分频的配置寄存器均放在了 PCR 模块中。更多信息，详见 6.4 寄存器列表。

功能模块不工作时，可以配置 PCR 模块中对应寄存器关闭功能时钟，关闭模块功能时钟不会影响系统其他部分。

下文以 I2C 时钟配置为例：

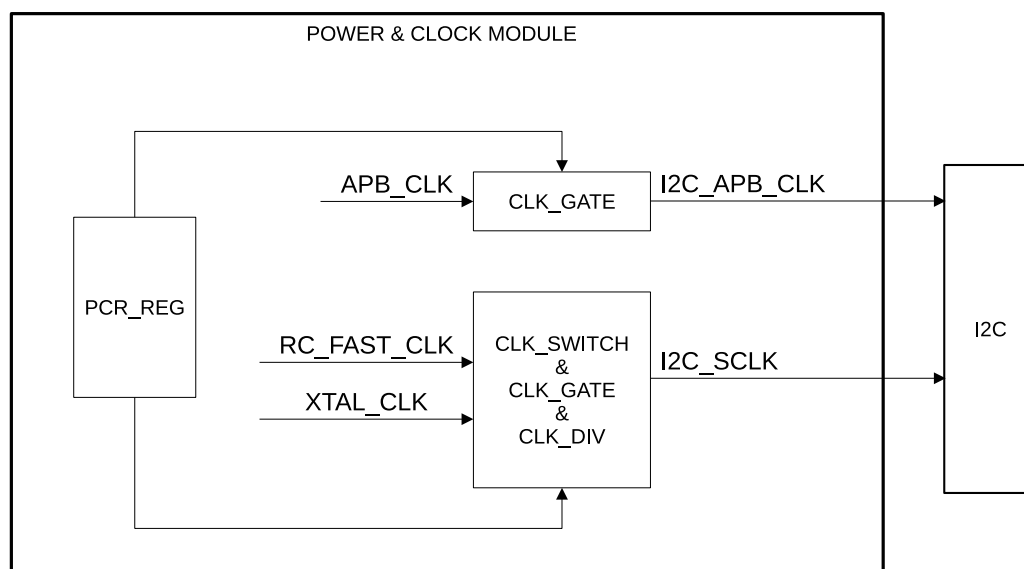


图 6.3-1. 时钟配置示例

图 6.3-1 是 I2C 的时钟结构图，其他模块的时钟结构类似。其中 `CLK_SWITCH` 的功能是选择一个时钟输出，`CLK_GATE` 的功能是打开/关闭时钟。

对于要求低功耗的场景，当模块不工作时，除了关闭功能时钟，还可以关闭模块的总线时钟来进一步降低功耗。

注意，如果先关闭模块总线时钟，那么模块的功能时钟还是有可能继续工作，所以推荐先关闭功能时钟再关闭总线时钟，打开时钟时推荐先打开总线时钟再打开功能时钟。

说明：

本章节中所有分频寄存器配置的值 of 实际分频值减去 1。

6.4 寄存器列表

6.4.1 PCR 模块寄存器列表

本小节的所有地址均为相对于电源/时钟/复位寄存器（PCR）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PCR_UART0_CONF_REG	UART0 配置寄存器	0x0000	varies
PCR_UART0_SCLK_CONF_REG	UART0_SCLK 配置寄存器	0x0004	R/W
PCR_UART0_PD_CTRL_REG	UART0 电源控制寄存器	0x0008	R/W
PCR_UART1_CONF_REG	UART1 配置寄存器	0x000C	varies
PCR_UART1_SCLK_CONF_REG	UART1_SCLK 配置寄存器	0x0010	R/W
PCR_UART1_PD_CTRL_REG	UART1 电源控制寄存器	0x0014	R/W
PCR_I2C_CONF_REG	I2C 配置寄存器	0x0020	R/W
PCR_I2C_SCLK_CONF_REG	I2C_SCLK 配置寄存器	0x0024	R/W
PCR_TWAIO_CONF_REG	TWAIO 配置寄存器	0x0028	varies
PCR_TWAIO_FUNC_CLK_CONF_REG	TWAIO_FUNC_CLK 配置寄存器	0x002C	R/W
PCR_TWAI1_CONF_REG	TWAI1 配置寄存器	0x0030	varies
PCR_TWAI1_FUNC_CLK_CONF_REG	TWAI1_FUNC_CLK 配置寄存器	0x0034	R/W
PCR_UHCI_CONF_REG	UHCI 配置寄存器	0x0038	varies
PCR_RMT_CONF_REG	RMT 配置寄存器	0x003C	R/W
PCR_RMT_SCLK_CONF_REG	RMT_SCLK 配置寄存器	0x0040	R/W
PCR_RMT_PD_CTRL_REG	RMT 电源控制寄存器	0x0044	R/W
PCR_LEDC_CONF_REG	LEDC 配置寄存器	0x0048	varies
PCR_LEDC_SCLK_CONF_REG	LEDC_SCLK 配置寄存器	0x004C	R/W
PCR_LEDC_PD_CTRL_REG	LEDC 电源控制寄存器	0x0050	R/W
PCR_TIMERGROUPO_CONF_REG	TIMERGROUPO 配置寄存器	0x0054	varies
PCR_TIMERGROUPO_TIMER_CLK_CONF_REG	TIMERGROUPO_TIMER_CLK 配置寄存器	0x0058	R/W
PCR_TIMERGROUPO_WDT_CLK_CONF_REG	TIMERGROUPO_WDT_CLK 配置寄存器	0x005C	R/W
PCR_TIMERGROUP1_CONF_REG	TIMERGROUP1 配置寄存器	0x0060	varies
PCR_TIMERGROUP1_TIMER_CLK_CONF_REG	TIMERGROUP1_TIMER_CLK 配置寄存器	0x0064	R/W
PCR_TIMERGROUP1_WDT_CLK_CONF_REG	TIMERGROUP1_WDT_CLK 配置寄存器	0x0068	R/W
PCR_SYSTIMER_CONF_REG	SYSTIMER 配置寄存器	0x006C	varies
PCR_SYSTIMER_FUNC_CLK_CONF_REG	SYSTIMER_FUNC_CLK 配置寄存器	0x0070	R/W
PCR_I2S_CONF_REG	I2S 配置寄存器	0x0074	varies
PCR_I2S_TX_CLKM_CONF_REG	I2S_TX_CLKM 配置寄存器	0x0078	R/W
PCR_I2S_TX_CLKM_DIV_CONF_REG	I2S_TX_CLKM_DIV 配置寄存器	0x007C	R/W
PCR_I2S_RX_CLKM_CONF_REG	I2S_RX_CLKM 配置寄存器	0x0080	R/W
PCR_I2S_RX_CLKM_DIV_CONF_REG	I2S_RX_CLKM_DIV 配置寄存器	0x0084	R/W

名称	描述	地址	访问
PCR_SARADC_CONF_REG	SARADC 配置寄存器	0x0088	R/W
PCR_SARADC_CLKM_CONF_REG	SARADC_CLKM 配置寄存器	0x008C	R/W
PCR_TSENS_CLK_CONF_REG	TSENS_CLK 配置寄存器	0x0090	R/W
PCR_USB_DEVICE_CONF_REG	USB_DEVICE 配置寄存器	0x0094	varies
PCR_INTMTX_CONF_REG	INTMTX 配置寄存器	0x0098	varies
PCR_PCNT_CONF_REG	PCNT 配置寄存器	0x009C	varies
PCR_ETM_CONF_REG	ETM 配置寄存器	0x00A0	varies
PCR_PWM_CONF_REG	PWM 配置寄存器	0x00A4	varies
PCR_PWM_CLK_CONF_REG	PWM_CLK 配置寄存器	0x00A8	R/W
PCR_PARL_IO_CONF_REG	PARL_IO 配置寄存器	0x00AC	varies
PCR_PARL_CLK_RX_CONF_REG	PARL_CLK_RX 配置寄存器	0x00B0	R/W
PCR_PARL_CLK_TX_CONF_REG	PARL_CLK_TX 配置寄存器	0x00B4	R/W
PCR_GDMA_CONF_REG	GDMA 配置寄存器	0x00C0	R/W
PCR_SPI2_CONF_REG	SPI2 配置寄存器	0x00C4	varies
PCR_SPI2_CLKM_CONF_REG	SPI2_CLKM 配置寄存器	0x00C8	R/W
PCR_AES_CONF_REG	AES 配置寄存器	0x00CC	varies
PCR_SHA_CONF_REG	SHA 配置寄存器	0x00D0	varies
PCR_RSA_CONF_REG	RSA 配置寄存器	0x00D4	varies
PCR_RSA_PD_CTRL_REG	RSA 电源控制寄存器	0x00D8	R/W
PCR_ECC_CONF_REG	ECC 配置寄存器	0x00DC	varies
PCR_ECC_PD_CTRL_REG	ECC 电源控制寄存器	0x00E0	R/W
PCR_DS_CONF_REG	DS 配置寄存器	0x00E4	varies
PCR_HMAC_CONF_REG	HMAC 配置寄存器	0x00E8	varies
PCR_ECDSA_CONF_REG	ECDSA 配置寄存器	0x00EC	varies
PCR_IOMUX_CONF_REG	IOMUX 配置寄存器	0x00F0	R/W
PCR_IOMUX_CLK_CONF_REG	IOMUX_CLK 配置寄存器	0x00F4	R/W
PCR_TRACE_CONF_REG	TRACE 配置寄存器	0x00FC	R/W
PCR_ASSIST_CONF_REG	ASSIST 配置寄存器	0x0100	R/W
PCR_CACHE_CONF_REG	CACHE 配置寄存器	0x0104	R/W
PCR_TIMEOUT_CONF_REG	TIMEOUT 配置寄存器	0x010C	R/W
PCR_SYSCCLK_CONF_REG	SYSCCLK 配置寄存器	0x0110	varies
PCR_CPU_WAITI_CONF_REG	CPU_WAITI 配置寄存器	0x0114	varies
PCR_CPU_FREQ_CONF_REG	CPU_FREQ 配置寄存器	0x0118	R/W
PCR_AHB_FREQ_CONF_REG	AHB_FREQ 配置寄存器	0x011C	R/W
PCR_APB_FREQ_CONF_REG	APB_FREQ 配置寄存器	0x0120	R/W
PCR_PLL_DIV_CLK_EN_REG	SPLL DIV clock-gating 配置寄存器	0x0128	R/W
PCR_CTRL_32K_CONF_REG	32KHz clock 配置寄存器	0x0130	R/W
PCR_SEC_CONF_REG	片外存储器加密与解密模块时钟源配置寄存器	0x013C	R/W
PCR_BUS_CLK_UPDATE_REG	高性能系统时钟源更新使能寄存器	0x0144	R/ W/ WTC
PCR_SAR_CLK_DIV_REG	SAR ADC 时钟分频系数配置寄存器	0x0148	R/W
PCR_BS_CONF_REG	BS 配置寄存器	0x0150	R/W

名称	描述	地址	访问
PCR_BS_FUNC_CONF_REG	BS_FUNC_CLK 配置寄存器	0x0154	R/W
PCR_BS_PD_CTRL_REG	BS 电源控制寄存器	0x0158	R/W
PCR_TIMERGROUP_WDT_CONF_REG	TIMERGROUP_WDT 配置寄存器	0x015C	R/W
PCR_TIMERGROUP_XTAL_CONF_REG	TIMERGROUP1 配置寄存器	0x0160	R/W
PCR_TCM_MEM_MONITOR_CONF_REG	TCM_MEM_MONITOR 配置寄存器	0x016C	varies
PCR_PSRAM_MEM_MONITOR_CONF_REG	PSRAM_MEM_MONITOR 配置寄存器	0x0170	varies
PCR_HPCORE_0_PD_CTRL_REG	HP CORE0 电源控制寄存器	0x0178	R/W
PCR_SDIO_SLAVE_CONF_REG	SDIO_SLAVE 配置寄存器	0x017C	R/W
频率数据寄存器			
PCR_SYSCLK_FREQ_QUERY_0_REG	SYSCLK frequency query 0 register	0x0124	HRO
版本寄存器			
PCR_DATE_REG	版本控制寄存器	0x0FFC	R/W

6.4.2 低功耗系统时钟寄存器列表

本小节前缀为 LPPERI 的寄存器（最后两个）地址为相对于低功耗外设寄存器（LPPERI）基地址的地址偏移量（相对地址），前缀为 LP_AON 的寄存器（倒数第三、四个）地址为相对于低功耗常开寄存器（LP_AON）基地址的地址偏移量（相对地址），其余所有地址均为相对于低功耗时钟复位寄存器（LP_CLKRST）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_CLKRST_LP_CLK_CONF_REG	LP 系统时钟源配置寄存器	0x0000	R/W
LP_CLKRST_LP_CLK_PO_EN_REG	管脚时钟信号门控配置寄存器	0x0004	R/W
LP_CLKRST_LP_CLK_EN_REG	LP 时钟源门控配置寄存器	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	LP 外设软件复位配置寄存器	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	复位原因状态寄存器	0x0010	varies
LP_CLKRST_CPU_RESET_REG	CPU 复位配置寄存器	0x0014	R/W
LP_CLKRST_FOSC_CNTL_REG	RC_FAST_CLK 时钟频率配置寄存器	0x0018	R/W
LP_CLKRST_CLK_TO_HP_REG	HP 系统时钟信号门控配置寄存器	0x0020	R/W
LP_CLKRST_LPMEM_FORCE_REG	LP 存储器强制开启时钟门控配置寄存器	0x0024	R/W
LP_CLKRST_LPPERI_REG	LP 外设时钟配置寄存器	0x0028	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K 配置寄存器	0x002C	R/W
LP_CLKRST_DATE_REG	版本控制寄存器	0x03FC	R/W
LP_AON_SYS_CFG_REG	软件系统复位寄存器	0x0034	WT
LP_AON_CPUCORE0_CFG_REG	软件 CPU 复位寄存器	0x0038	WT
LPPERI_CLK_EN_REG	eFuse 时钟门控配置寄存器	0x0000	R/W
LPPERI_RESET_EN_REG	eFuse 复位配置寄存器	0x0004	R/W

6.5 寄存器

6.5.1 PCR 模块寄存器

本小节的所有地址均为相对于电源/时钟/复位寄存器（PCR）基地址的地址偏移量（相对地址），具体基地址请见章节 4 [系统和存储器](#) 中的表 4.3-2。

Register 6.1. PCR_UARTO_CONF_REG (0x0000)

(reserved)																																PCR_UARTO_READY PCR_UARTO_RST_EN PCR_UARTO_CLK_EN			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset		

PCR_UARTO_CLK_EN 配置是否使能 UARTO APB_CLK。

0：不使能

1：使能

(R/W)

PCR_UARTO_RST_EN 配置是否复位 UARTO 模块。

0：不复位

1：复位

(R/W)

PCR_UARTO_READY 表示 UARTO 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 6.2. PCR_UART0_SCLK_CONF_REG (0x0004)

(reserved)										PCR_UART0_SCLK_EN				PCR_UART0_SCLK_SEL				PCR_UART0_SCLK_DIV_NUM				PCR_UART0_SCLK_DIV_B				PCR_UART0_SCLK_DIV_A			
31								23		22	21	20		19				12		11		6		5	0				
0 0 0 0 0 0 0 0								1		3		0				0				0				Reset					

PCR_UART0_SCLK_DIV_A 配置 UART0 功能时钟分频系数小数部分的分母。(R/W)

PCR_UART0_SCLK_DIV_B 配置 UART0 功能时钟分频系数小数部分的分子。(R/W)

PCR_UART0_SCLK_DIV_NUM 配置 UART0 功能时钟分频系数的整数部分。(R/W)

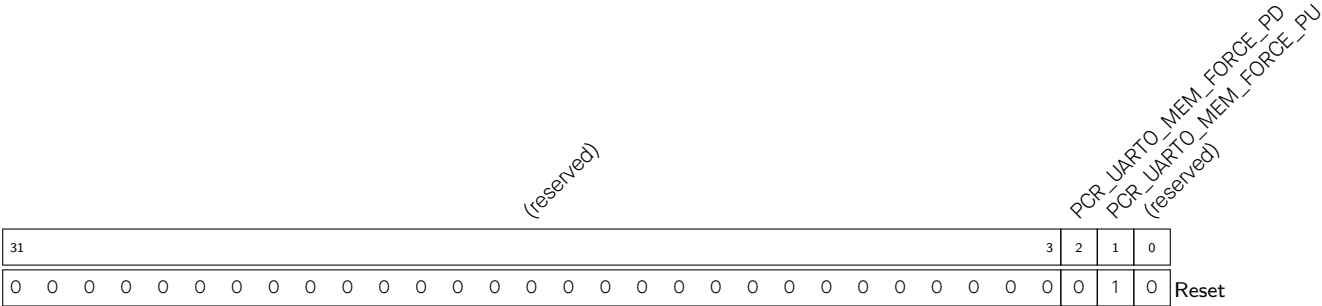
PCR_UART0_SCLK_SEL 配置 UART0 时钟源。

- 0: 不选择任何时钟
 - 1: 选择 PLL_F80M_CLK 时钟
 - 2: 选择 RC_FAST_CLK 时钟
 - 3 (默认): 选择 XTAL_CLK 时钟
- (R/W)

PCR_UART0_SCLK_EN 配置是否使能 UART0 功能时钟。

- 0: 不使能
 - 1: 使能
- (R/W)

Register 6.3. PCR_UART0_PD_CTRL_REG (0x0008)



PCR_UART0_MEM_FORCE_PU 配置是否强制 UART0 存储器上电。

0: 不强制 UART0 存储器上电

1: 强制 UART0 存储器上电

(R/W)

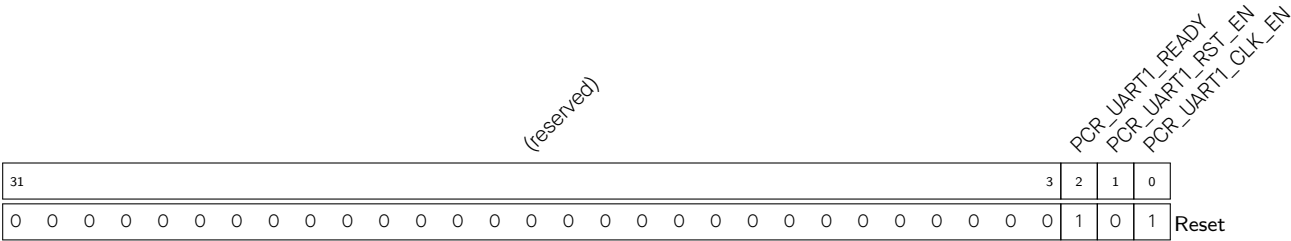
PCR_UART0_MEM_FORCE_PD 配置是否强制 UART0 存储器掉电。

0: 不强制 UART0 存储器掉电

1: 强制 UART0 存储器掉电

(R/W)

Register 6.4. PCR_UART1_CONF_REG (0x000C)



PCR_UART1_CLK_EN 配置是否使能 UART1 APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_UART1_RST_EN 配置是否复位 UART1 模块。

0: 不复位

1: 复位

(R/W)

PCR_UART1_READY 表示 UART1 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.5. PCR_UART1_SCLK_CONF_REG (0x0010)

(reserved)										PCR_UART1_SCLK_EN		PCR_UART1_SCLK_SEL		PCR_UART1_SCLK_DIV_NUM				PCR_UART1_SCLK_DIV_B		PCR_UART1_SCLK_DIV_A						
31											23	22	21	20	19					12	11			6	5	0
0	0	0	0	0	0	0	0	0	0	1	3	0				0		0		0		Reset				

PCR_UART1_SCLK_DIV_A 配置 UART1 功能时钟分频系数小数部分的分母。(R/W)

PCR_UART1_SCLK_DIV_B 配置 UART1 功能时钟分频系数小数部分的分子。(R/W)

PCR_UART1_SCLK_DIV_NUM 配置 UART1 功能时钟分频系数的整数部分。(R/W)

PCR_UART1_SCLK_SEL 配置选择 UART1 时钟源。

- 0: 不选择任何时钟
- 1: 选择 PLL_F80M_CLK 时钟
- 2: 选择 RC_FAST_CLK 时钟
- 3 (默认): 选择 XTAL_CLK 时钟
- (R/W)

PCR_UART1_SCLK_EN 配置是否使能 UART1 功能时钟。

- 0: 不使能
- 1: 使能
- (R/W)

Register 6.6. PCR_UART1_PD_CTRL_REG (0x0014)

(reserved)																															PCR_UART1_MEM_FORCE_PD			PCR_UART1_MEM_FORCE_PU																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_UART1_MEM_FORCE_PU 配置是否强制 UART1 存储器上电。

- 0: 不强制 UART1 存储器上电
- 1: 强制 UART1 存储器上电
- (R/W)

PCR_UART1_MEM_FORCE_PD 配置是否强制 UART1 存储器掉电。

- 0: 不强制 UART1 存储器掉电
- 1: 强制 UART1 存储器掉电
- (R/W)

Register 6.7. PCR_I2C_CONF_REG (0x0020)

(reserved)																												PCR_I2C_READY PCR_I2C_RST_EN PCR_I2C_CLK_EN			
31																												3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_I2C_CLK_EN 配置是否使能 I2C APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_I2C_RST_EN 配置是否复位 I2C 模块。

0: 不复位

1: 复位

(R/W)

PCR_I2C_READY 表示 I2C 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.8. PCR_I2C_SCLK_CONF_REG (0x0024)

(reserved)										PCR_I2C_SCLK_EN (reserved)			PCR_I2C_SCLK_SEL			PCR_I2C_SCLK_DIV_NUM						PCR_I2C_SCLK_DIV_B				PCR_I2C_SCLK_DIV_A			
31										23	22	21	20	19	12						11	6				5	0		
0 0 0 0 0 0 0 0 0 0										1	0	0	0						0				0				Reset		

PCR_I2C_SCLK_DIV_A 配置 I2C 功能时钟分频系数小数部分的分母。(R/W)

PCR_I2C_SCLK_DIV_B 配置 I2C 功能时钟分频系数小数部分的分子。(R/W)

PCR_I2C_SCLK_DIV_NUM 配置 I2C 功能时钟分频系数的整数部分。(R/W)

PCR_I2C_SCLK_SEL 配置选择 I2C 时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

(R/W)

PCR_I2C_SCLK_EN 配置是否使能 I2C 功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.9. PCR_TWAIO_CONF_REG (0x0028)

(reserved)																																PCR_TWAIO_READY PCR_TWAIO_RST_EN PCR_TWAIO_CLK_EN			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset		

PCR_TWAIO_CLK_EN 配置是否使能 TWAIO APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_TWAIO_RST_EN 配置是否复位 TWAIO 模块。

0: 不复位

1: 复位

(R/W)

PCR_TWAIO_READY 表示 TWAIO 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.10. PCR_TWAIO_FUNC_CLK_CONF_REG (0x002C)

(reserved)										PCR_TWAIO_FUNC_CLK_EN (reserved) PCR_TWAIO_FUNC_CLK_SEL										(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
31																				23	22	21	20	19												0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_TWAIO_FUNC_CLK_SEL 配置选择 TWAIO 时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

(R/W)

PCR_TWAIO_FUNC_CLK_EN 配置是否使能 TWAIO 功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.11. PCR_TWAI1_CONF_REG (0x0030)

(reserved)																												PCR_TWAI1_READY PCR_TWAI1_RST_EN PCR_TWAI1_CLK_EN			
31																												3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_TWAI1_CLK_EN 配置是否使能 TWAI1 APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_TWAI1_RST_EN 配置是否复位 TWAI1 模块。

0: 不复位

1: 复位

(R/W)

PCR_TWAI1_READY 表示 TWAI1 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.12. PCR_TWAI1_FUNC_CLK_CONF_REG (0x0034)

(reserved)										PCR_TWAI1_FUNC_CLK_EN (reserved)										PCR_TWAI1_FUNC_CLK_SEL										(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
31																				23	22	21	20	19													0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_TWAI1_FUNC_CLK_SEL 配置选择 TWAI1 时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

(R/W)

PCR_TWAI1_FUNC_CLK_EN 配置是否使能 TWAI1 功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.13. PCR_UHCI_CONF_REG (0x0038)

(reserved)																																POR_UHCI_READY POR_UHCI_RST_EN POR_UHCI_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_UHCI_CLK_EN 配置是否使能 UHCI 时钟。

0: 不使能

1: 使能

(R/W)

PCR_UHCI_RST_EN 配置是否复位 UHCI 模块。

0: 不复位

1: 复位

(R/W)

PCR_UHCI_READY 表示 UHCI 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.14. PCR_RMT_CONF_REG (0x003C)

(reserved)																												PCR_RMT_READY PCR_RMT_RST_EN PCR_RMT_CLK_EN			
31																												3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_RMT_CLK_EN 配置是否使能 RMT APB_CLK。

0: 使能

1: 不使能

(R/W)

PCR_RMT_RST_EN 配置是否复位 RMT 模块。

0: 不复位

1: 复位

(R/W)

PCR_RMT_READY 表示 RMT 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.15. PCR_RMT_SCLK_CONF_REG (0x0040)

(reserved)								PCR_RMT_SCLK_EN			PCR_RMT_SCLK_SEL				PCR_RMT_SCLK_DIV_NUM						PCR_RMT_SCLK_DIV_B				PCR_RMT_SCLK_DIV_A			
31								23	22	21	20	19				12	11					6	5					0
0	0	0	0	0	0	0	0	0	0	0	1	1				0				0				0				Reset

- PCR_RMT_SCLK_DIV_A 配置 RMT 功能时钟分频系数小数部分的分母。(R/W)
- PCR_RMT_SCLK_DIV_B 配置 RMT 功能时钟分频系数小数部分的分子。(R/W)
- PCR_RMT_SCLK_DIV_NUM 配置 RMT 功能时钟分频系数的整数部分。(R/W)
- PCR_RMT_SCLK_SEL 配置选择 RMT 时钟源。

0: 选择 XTAL_CLK 时钟

1 (默认): 选择 RC_FAST_CLK 时钟

(R/W) 2: 选择 PLL_F80M_CLK 时钟

(R/W)
- PCR_RMT_SCLK_EN 配置是否使能 RMT 功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.16. PCR_RMT_PD_CTRL_REG (0x0044)

(reserved)																																PCR_RMT_MEM_FORCE				PCR_RMT_MEM_FORCE			
31																																3	2	1	0				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Reset					

- PCR_RMT_MEM_FORCE_PU 配置是否强制 RMT 存储器上电。

0: 不强制 RMT 存储器上电

1: 强制 RMT 存储器上电

(R/W)
- PCR_RMT_MEM_FORCE_PD 配置是否强制 RMT 存储器掉电。

0: 不强制 RMT 存储器掉电

1: 强制 RMT 存储器掉电

(R/W)

Register 6.17. PCR_LEDC_CONF_REG (0x0048)

(reserved)																																PCR_LEDC_READY PCR_LEDC_RST_EN PCR_LEDC_CLK_EN		
31																													3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset	

PCR_LEDC_CLK_EN 配置是否使能 LEDC APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_LEDC_RST_EN 配置是否复位 LEDC 模块。

0: 不复位

1: 复位

(R/W)

PCR_LEDC_READY 表示 LEDC 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.18. PCR_LEDC_SCLK_CONF_REG (0x004C)

(reserved)										PCR_LEDC_SCLK_EN				PCR_LEDC_SCLK_SEL				(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31																				23	22	21	20	19																				0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_LEDC_SCLK_SEL 配置选择 LEDC 时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

2: 选择 PLL_F80M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR_LEDC_SCLK_EN 配置是否使能 LEDC 功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.19. PCR_LEDC_PD_CTRL_REG (0x0050)

(reserved)																															PCR_LEDC_MEM_FORCE_PD PCR_LEDC_MEM_FORCE_PU (reserved)			
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Reset		

PCR_LEDC_MEM_FORCE_PU 配置是否强制 LEDC 存储器上电。

0：不强制 LEDC 存储器上电

1：强制 LEDC 存储器上电

(R/W)

PCR_LEDC_MEM_FORCE_PD 配置是否强制 LEDC 存储器掉电。

0：不强制 LEDC 存储器掉电

1：强制 LEDC 存储器掉电

(R/W)

Register 6.20. PCR_TIMERGROUPO_CONF_REG (0x0054)

(reserved)																										PCR_TGO_TIMER1_READY PCR_TGO_TIMER0_READY PCR_TGO_WDT_READY PCR_TGO_RST_EN PCR_TGO_CLK_EN					
31																										5	4	3	2	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																										1	1	1	0	1	Reset

PCR_TGO_CLK_EN 配置是否使能定时器组 0 APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_TGO_RST_EN 配置是否复位定时器组 0 模块。

0: 不复位

1: 复位

(R/W)

PCR_TGO_WDT_READY 表示定时器组 0 中的 WDT 是否解复位完成。

0: 未完成

1: 完成

(RO)

PCR_TGO_TIMER0_READY 表示定时器组 0 中的定时器 0 是否解复位完成。

0: 未完成

1: 完成

(RO)

PCR_TGO_TIMER1_READY 表示定时器组 0 中的定时器 1 是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.21. PCR_TIMERGROUPO_TIMER_CLK_CONF_REG (0x0058)

[illegible]

PCR_TGO_TIMER_CLK_SEL 配置选择定时器组 0 通用定时器的时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

2: 选择 PLL_F48M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR TGO TIMER CLK EN 配置是否使能定时器组 0 通用定时器时钟。

0: 不使能

1: 使能

(R/W)

Register 6.22. PCR_TIMERGROUPO_WDT_CLK_CONF_REG (0x005C)

[illegible]

PCR_TGO_WDT_CLK_SEL 配置选择定时器组 0 中 WDT 的时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC FAST CLK 时钟

2: 选择 PLL_F80M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR TGO WDT CLK EN 配置是否使能定时器组 0 中 WDT 的时钟。

0: 不使能

1: 使能

(R/W)

Register 6.23. PCR_TIMERGROUP1_CONF_REG (0x0060)

(reserved)																										PCR_TG1_TIMER1_READY PCR_TG1_TIMER0_READY PCR_TG1_WDT_READY PCR_TG1_RST_EN PCR_TG1_CLK_EN						
31																										5	4	3	2	1	0	Reset
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																										1	1	1	0	1		

- PCR_TG1_CLK_EN

配置是否使能定时器组 1 APB_CLK。

0: 不使能

1: 使能

(R/W)
- PCR_TG1_RST_EN

配置是否复位定时器组 1 模块。

0: 不复位

1: 复位

(R/W)
- PCR_TG1_WDT_READY

表示定时器组 1 中的看门狗定时器是否解复位完成。

0: 未完成

1: 完成

(RO)
- PCR_TG1_TIMER0_READY

表示定时器组 1 中的定时器 0 是否解复位完成。

0: 未完成

1: 完成

(RO)
- PCR_TG1_TIMER1_READY

表示定时器组 1 中的定时器 1 是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.24. PCR_TIMERGROUP1_TIMER_CLK_CONF_REG (0x0064)

(reserved)										PCR_TG1_TIMER_CLK_EN										(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
31										23	22	21	20	19																			0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

PCR_TG1_TIMER_CLK_SEL 配置选择定时器组 1 通用定时器的时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

2: 选择 PLL_F48M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR_TG1_TIMER_CLK_EN 配置是否使能定时器组 1 通用定时器的时钟。

0: 不使能

1: 使能

(R/W)

Register 6.25. PCR_TIMERGROUP1_WDT_CLK_CONF_REG (0x0068)

(reserved)										PCR_TG1_WDT_CLK_EN										PCR_TG1_WDT_CLK_SEL										(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
31									23	22	21	20	19																							0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

PCR_TG1_WDT_CLK_SEL 配置选择定时器组 1 WDT 的时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

2: 选择 PLL_F80M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR_TG1_WDT_CLK_EN 配置是否使能定时器组 1 WDT 的时钟。

0: 不使能

1: 使能

(R/W)

Register 6.26. PCR_SYSTIMER_CONF_REG (0x006C)

(reserved)																																PCR_SYSTIMER_READY PCR_SYSTIMER_RST_EN PCR_SYSTIMER_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_SYSTIMER_CLK_EN 配置是否使能系统定时器 APB_CLK。

0: 不使能
1: 使能
(R/W)

PCR_SYSTIMER_RST_EN 配置是否复位系统定时器模块。

0: 不复位
1: 复位
(R/W)

PCR_SYSTIMER_READY 表示系统定时器模块是否解复位完成。

0: 未完成
1: 完成
(RO)

Register 6.27. PCR_SYSTIMER_FUNC_CLK_CONF_REG (0x0070)

(reserved)										PCR_SYSTIMER_FUNC_CLK_EN (reserved)				PCR_SYSTIMER_FUNC_CLK_SEL (reserved)																					
31																				23	22	21	20	19											0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PCR_SYSTIMER_FUNC_CLK_SEL 配置选择系统定时器时钟源。

0 (默认): 选择 XTAL_CLK 时钟
1: 选择 RC_FAST_CLK 时钟
(R/W)

PCR_SYSTIMER_FUNC_CLK_EN 配置是否使能系统定时器功能时钟。

0: 不使能
1: 使能
(R/W)

Register 6.28. PCR_I2S_CONF_REG (0x0074)

(reserved)																												PCR_I2S_TX_READY PCR_I2S_RX_READY PCR_I2S_RST_EN PCR_I2S_CLK_EN				
31																												4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	Reset

- PCR_I2S_CLK_EN

配置是否使能 I2S APB_CLK。

0: 不使能

1: 使能

(R/W)
- PCR_I2S_RST_EN

配置是否复位 I2S 模块。

0: 不复位

1: 复位

(R/W)
- PCR_I2S_RX_READY

表示 I2S RX 模块是否解复位完成。

0: 未完成

1: 完成

(RO)
- PCR_I2S_TX_READY

表示 I2S TX 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.29. PCR_I2S_TX_CLKM_CONF_REG (0x0078)

(reserved)										PCR_I2S_TX_CLKM_EN				PCR_I2S_TX_CLKM_SEL				PCR_I2S_TX_CLKM_DIV_NUM								(reserved)							
31										23		22	21	20	19				12				11								0		
0 0 0 0 0 0 0 0 0 0										1	0	2				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0								Reset									

PCR_I2S_TX_CLKM_DIV_NUM 配置 I2S TX 时钟分频系数的整数部分。
(R/W)

PCR_I2S_TX_CLKM_SEL 配置选择 I2S TX 模块的时钟源。
0: 选择 XTAL_CLK 时钟
1: 选择 PLL_F240M_CLK 时钟
2: 选择 PLL_F160M_CLK 时钟
3: 选择 I2S_MCLK_in 时钟
(R/W)

PCR_I2S_TX_CLKM_EN 配置是否使能 I2S TX 功能时钟。
0: 不使能
1: 使能
(R/W)

Register 6.30. PCR_I2S_TX_CLKM_DIV_CONF_REG (0x007C)

(reserved)				PCR_I2S_TX_CLKM_DIV_YN1				PCR_I2S_TX_CLKM_DIV_X				PCR_I2S_TX_CLKM_DIV_Y				PCR_I2S_TX_CLKM_DIV_Z			
31	28	27	26	18	17	9	8	0											
0	0	0	0	0	0	1	0	0	Reset										

PCR_I2S_TX_CLKM_DIV_Z $b \leq a/2$ 时, I2S_TX_CLKM_DIV_Z 的值为 b 。 $b > a/2$ 时, I2S_TX_CLKM_DIV_Z 的值为 $a - b$ 。(R/W)

PCR_I2S_TX_CLKM_DIV_Y $b \leq a/2$ 时, I2S_TX_CLKM_DIV_Y 的值为 $a \% b$ 。 $b > a/2$ 时, I2S_TX_CLKM_DIV_Y 的值为 $a \% (a - b)$ 。(R/W)

PCR_I2S_TX_CLKM_DIV_X $b \leq a/2$ 时, I2S_TX_CLKM_DIV_X 的值为 $\text{floor}(a/b) - 1$ 。 $b > a/2$, I2S_TX_CLKM_DIV_X 的值为 $\text{floor}(a/(a - b)) - 1$ 。(R/W)

PCR_I2S_TX_CLKM_DIV_YN1 $b \leq a/2$ 时, I2S_TX_CLKM_DIV_YN1 的值为 0 。 $b > a/2$ 时, I2S_TX_CLKM_DIV_YN1 的值为 1 。(R/W)

说明:
上文所述的“a”和“b”分别为小数分频的分母部分和分子部分。更多信息, 见章节 [I2S 控制器 \(I2S\)](#) > 章节 [27.6](#)。

Register 6.31. PCR_I2S_RX_CLKM_CONF_REG (0x0080)

(reserved)								PCR_I2S_MCLK_SEL				PCR_I2S_RX_CLKM_EN				PCR_I2S_RX_CLKM_SEL				PCR_I2S_RX_CLKM_DIV_NUM								(reserved)										
31								24	23	22	21	20	19								12	11																0
0	0	0	0	0	0	0	0	0	0	1	0	2								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset		

PCR_I2S_RX_CLKM_DIV_NUM 配置 I2S 时钟的整数分频系数。(R/W)

PCR_I2S_RX_CLKM_SEL 配置选择 I2S RX 模块的时钟源。

- 0: 选择 XTAL_CLK 时钟
- 1: 选择 PLL_F240M_CLK 时钟
- 2: 选择 PLL_F160M_CLK 时钟
- 3: 选择 I2S_MCLK_in 时钟
- (R/W)

PCR_I2S_RX_CLKM_EN 配置是否使能 I2S RX 功能时钟。

- 0: 不使能
- 1: 使能
- (R/W)

PCR_I2S_MCLK_SEL 配置选择主时钟。

- 0 (默认): 选择 I2S_RX_CLK
- 1: 选择 I2S_TX_CLK
- (R/W)

Register 6.32. PCR_I2S_RX_CLKM_DIV_CONF_REG (0x0084)

(reserved)				PCR_I2S_RX_CLKM_DIV_YN1				PCR_I2S_RX_CLKM_DIV_X				PCR_I2S_RX_CLKM_DIV_Y				PCR_I2S_RX_CLKM_DIV_Z			
31	28	27	26	18	17	9	8	0											
0	0	0	0	0	0	1	0	0	Reset										

PCR_I2S_RX_CLKM_DIV_Z $b \leq a/2$ 时, I2S_RX_CLKM_DIV_Z 的值为 b 。 $b > a/2$ 时, I2S_RX_CLKM_DIV_Z 的值为 $a - b$ 。
(R/W)

PCR_I2S_RX_CLKM_DIV_Y $b \leq a/2$ 时, I2S_RX_CLKM_DIV_Y 的值为 $a \% b$ 。 $b > a/2$ 时, I2S_RX_CLKM_DIV_Y 的值为 $a \% (a - b)$ 。
(R/W)

PCR_I2S_RX_CLKM_DIV_X $b \leq a/2$ 时, I2S_RX_CLKM_DIV_X 的值为 $\text{floor}(a/b) - 1$ 。 $b > a/2$, I2S_RX_CLKM_DIV_X 的值为 $\text{floor}(a/(a - b)) - 1$ 。
(R/W)

PCR_I2S_RX_CLKM_DIV_YN1 $b \leq a/2$ 时, I2S_RX_CLKM_DIV_YN1 的值为 0 。 $b > a/2$ 时, I2S_RX_CLKM_DIV_YN1 的值为 1 。
(R/W)

说明:
上文所述的“a”和“b”分别为小数分频的分母部分和分子部分。更多信息，见章节 27.6。

Register 6.33. PCR_SARADC_CONF_REG (0x0088)

(reserved)																												PCR_SARADC_REG_RST_EN PCR_SARADC_REG_CLK_EN PCR_SARADC_RST_EN (reserved)				
31																												4	3	2	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																												0	1	0	0	Reset

PCR_SARADC_RST_EN 配置是否复位 SAR ADC 功能寄存器。

0: 不复位

1: 复位

(R/W)

PCR_SARADC_REG_CLK_EN 配置是否使能 SAR ADC APB_CLK。

0: 不使能

1: 使能

(R/W)

PCR_SARADC_REG_RST_EN 配置是否复位 SAR ADC 模块的 APB 寄存器。

0: 不复位

1: 复位

(R/W)

Register 6.34. PCR_SARADC_CLKM_CONF_REG (0x008C)

(reserved)										PCR_SARADC_CLKM_EN										PCR_SARADC_CLKM_SEL										PCR_SARADC_CLKM_DIV_NUM										PCR_SARADC_CLKM_DIV_B										PCR_SARADC_CLKM_DIV_A																																										
31										23										22	21	20	19										12										11	6										5	0																																					
0										0										0										0										0										0										1	0	4										0										0										Reset

PCR_SARADC_CLKM_DIV_A 配置 SAR ADC 功能时钟分频系数小数部分的分母。(R/W)

PCR_SARADC_CLKM_DIV_B 配置 SAR ADC 功能时钟分频系数小数部分的分子。(R/W)

PCR_SARADC_CLKM_DIV_NUM 配置 SAR ADC 功能时钟分频系数的整数部分。(R/W)

PCR_SARADC_CLKM_SEL 配置选择 SAR ADC 时钟源。

- 0 (默认): 选择 XTAL_CLK 时钟
- 1: 选择 RC_FAST_CLK 时钟
- 2: 选择 PLL_F80M_CLK 时钟
- 3: 不选择任何时钟
- (R/W)

PCR_SARADC_CLKM_EN 配置是否使能 SAR ADC 功能时钟。

- 0: 不使能
- 1: 使能
- (R/W)

Register 6.35. PCR_TSENS_CLK_CONF_REG (0x0090)

[illegible]

PCR_TSENS_CLK_SEL 配置选择温度传感器时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

(R/W)

PCR_TSENS_CLK_EN 配置是否使能温度传感器时钟。

0: 不使能

1: 使能

(R/W)

PCR_TSENS_RST_EN 配置是否复位温度传感器模块。

0: 不复位

1: 复位

(R/W)

Register 6.36. PCR_USB_DEVICE_CONF_REG (0x0094)

(reserved)																																PCR_USB_DEVICE_READY PCR_USB_DEVICE_RST_EN PCR_USB_DEVICE_CLK_EN			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset	

PCR_USB_DEVICE_CLK_EN 配置是否使能 USB Serial/JTAG 时钟。

0: 不使能

1: 使能

(R/W)

PCR_USB_DEVICE_RST_EN 配置是否复位 USB Serial/JTAG 模块。

0: 不复位

1: 复位

(R/W)

PCR_USB_DEVICE_READY 表示 USB Serial/JTAG 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.37. PCR_INTMTX_CONF_REG (0x0098)

(reserved)																												PCR_INTMTX_READY PCR_INTMTX_RST_EN PCR_INTMTX_CLK_EN			
31																												3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_INTMTX_CLK_EN 配置是否使能中断矩阵时钟。

0: 不使能

1: 使能

(R/W)

PCR_INTMTX_RST_EN 配置是否复位中断矩阵模块。

0: 不复位

1: 复位

(R/W)

PCR_INTMTX_READY 表示中断矩阵模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.38. PCR_PCNT_CONF_REG (0x009C)

(reserved)																																PCR_PCNT_READY PCR_PCNT_RST_EN PCR_PCNT_CLK_EN			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset	

PCR_PCNT_CLK_EN 配置是否使能 PCNT 时钟。

0: 不使能

1: 使能

(R/W)

PCR_PCNT_RST_EN 配置是否复位 PCNT 模块。

0: 不复位

1: 复位

(R/W)

PCR_PCNT_READY 表示 PCNT 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.39. PCR_ETM_CONF_REG (0x00A0)

(reserved)																																POR_ETM_READY POR_ETM_RST_EN POR_ETM_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_ETM_CLK_EN 配置是否使能 ETM 时钟。

0: 不使能

1: 使能

(R/W)

PCR_ETM_RST_EN 配置是否复位 ETM 模块。

0: 不复位

1: 复位

(R/W)

PCR_ETM_READY 表示 ETM 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.40. PCR_PWM_CONF_REG (0x00A4)

(reserved)																																PCR_PWM_READY PCR_PWM_RST_EN PCR_PWM_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_PWM_CLK_EN 配置是否使能 MCPWM 总线时钟。

0: 不使能

1: 使能

(R/W)

PCR_PWM_RST_EN 配置是否复位 MCPWM 模块。

0: 不复位

1: 复位

(R/W)

PCR_PWM_READY 表示 MCPWM 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.41. PCR_PWM_CLK_CONF_REG (0x00A8)

[illegible]

PCR_PWM_DIV_NUM 配置 MCPWM 功能时钟分频系数的整数部分。(R/W)

PCR PWM CLKM SEL 配置选择 MCPWM 时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC FAST CLK 时钟

2: 选择 PLL_F160M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR_PWM_CLKM_EN 配置是否使能 MCPWM 的功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.42. PCR_PARL_IO_CONF_REG (0x00AC)

(reserved)																															PCR_PARL_READY PCR_PARL_RST_EN PCR_PARL_CLK_EN			
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR PARL CLK EN 配置是否使能并行 IO 控制器 APB CLK。

0: 不使能

1: 使能

(R/W)

PCR PARL RST EN 配置是否复位并行 IO 控制器 APB 寄存器。

0: 不复位

1: 复位

(R/W)

PCR_PARL_READY 表示并行 IO 控制器模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.43. PCR_PARL_CLK_RX_CONF_REG (0x00B0)

(reserved)																PCR_PARL_RX_RST_EN			PCR_PARL_CLK_RX_EN			PCR_PARL_CLK_RX_SEL			PCR_PARL_CLK_RX_DIV_NUM															
31																20	19	18	17	16	15															0				
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	1	0	0																Reset					

PCR_PARL_CLK_RX_DIV_NUM 配置并行 IO 控制器 RX 时钟的整数分频系数。(R/W)

PCR_PARL_CLK_RX_SEL 配置并行 IO 控制器 RX 选择时钟源。

- 0 (默认): 选择 XTAL_CLK 时钟
- 1: 选择 RC_FAST_CLK 时钟
- 2: 选择 PLL_F240M_CLK 时钟
- 3: 使用来自管脚的时钟
- (R/W)

PCR_PARL_CLK_RX_EN 配置是否使能并行 IO 控制器 RX 时钟。

- 0: 不使能
- 1: 使能
- (R/W)

PCR_PARL_RX_RST_EN 配置是否复位并行 IO 控制器 RX 模块。

- 0: 不复位
- 1: 复位
- (R/W)

Register 6.44. PCR_PARL_CLK_TX_CONF_REG (0x00B4)

(reserved)												PCR_PARL_TX_RST_EN			PCR_PARL_CLK_TX_EN			PCR_PARL_CLK_TX_SEL			PCR_PARL_CLK_TX_DIV_NUM														
31																		20	19	18	17	16	15												0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0										Reset			

PCR_PARL_CLK_TX_DIV_NUM 配置并行 IO 控制器 TX 时钟的整数分频系数。(R/W)

PCR_PARL_CLK_TX_SEL 配置选择并行 IO 控制器 TX 时钟源。

- 0 (默认): 选择 XTAL_CLK 时钟
- 1: 选择 RC_FAST_CLK 时钟
- 2: 选择 PLL_F240M_CLK 时钟
- 3: 使用来自管脚的时钟
- (R/W)

PCR_PARL_CLK_TX_EN 配置是否使能并行 IO 控制器 TX 时钟。

- 0: 不使能
- 1: 使能
- (R/W)

PCR_PARL_TX_RST_EN 配置是否复位并行 IO 控制器 TX 模块。

- 0: 不复位
- 1: 复位
- (R/W)

Register 6.45. PCR_GDMA_CONF_REG (0x00C0)

(reserved)																															PCR_GDMA_RST PCR_GDMA				
31																															2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

PCR_GDMA_CLK_EN 配置是否使能 GDMA 时钟。

- 0: 不使能
- 1: 使能
- (R/W)

PCR_GDMA_RST_EN 配置是否复位 GDMA 模块。

- 0: 不复位
- 1: 复位
- (R/W)

Register 6.46. PCR_SPI2_CONF_REG (0x00C4)

(reserved)																																PCR_SPI2_READY PCR_SPI2_RST_EN PCR_SPI2_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_SPI2_CLK_EN 配置是否使能 SPI2 APB_CLK。

0: 不使能
1: 使能
(R/W)

PCR_SPI2_RST_EN 配置是否复位 SPI2 模块。

0: 不复位
1: 复位
(R/W)

PCR_SPI2_READY 表示 SPI2 模块是否解复位完成。

0: 未完成
1: 完成
(RO)

Register 6.47. PCR_SPI2_CLKM_CONF_REG (0x00C8)

(reserved)										PCR_SPI2_CLKM_EN PCR_SPI2_CLKM_SEL										(reserved)																	
31																					23	22	21	20	19												0
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PCR_SPI2_CLKM_SEL 配置选择 SPI2 时钟源。

0 (默认): 选择 XTAL_CLK 时钟
1: 选择 PLL_F160M_CLK 时钟
2: 选择 RC_FAST_CLK 时钟
3: 选择 PLL_F120M_CLK 时钟
(R/W)

PCR_SPI2_CLKM_EN 配置是否使能 SPI2 功能时钟。

0: 不使能
1: 使能
(R/W)

Register 6.48. PCR_AES_CONF_REG (0x00CC)

(reserved)																																PCR_AES_READY PCR_AES_RST_EN PCR_AES_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_AES_CLK_EN 配置是否使能 AES 时钟。

0: 不使能

1: 使能

(R/W)

PCR_AES_RST_EN 配置是否复位 AES 模块。

0: 不复位

1: 复位

(R/W)

PCR_AES_READY 表示 AES 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.49. PCR_SHA_CONF_REG (0x00D0)

(reserved)																																PCR_SHA_READY PCR_SHA_RST_EN PCR_SHA_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_SHA_CLK_EN 配置是否使能 SHA 时钟。

0: 不使能

1: 使能

(R/W)

PCR_SHA_RST_EN 配置是否复位 SHA 模块。

0: 不复位

1: 复位

(R/W)

PCR_SHA_READY 表示 SHA 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.50. PCR_RSA_CONF_REG (0x00D4)

(reserved)																																PCR_RSA_READY PCR_RSA_RST_EN PCR_RSA_CLK_EN		
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

- PCR_RSA_CLK_EN

配置是否使能 RSA 时钟。

0: 不使能

1: 使能

(R/W)
- PCR_RSA_RST_EN

配置是否复位 RSA 模块。

0: 不复位

1: 复位

(R/W)
- PCR_RSA_READY

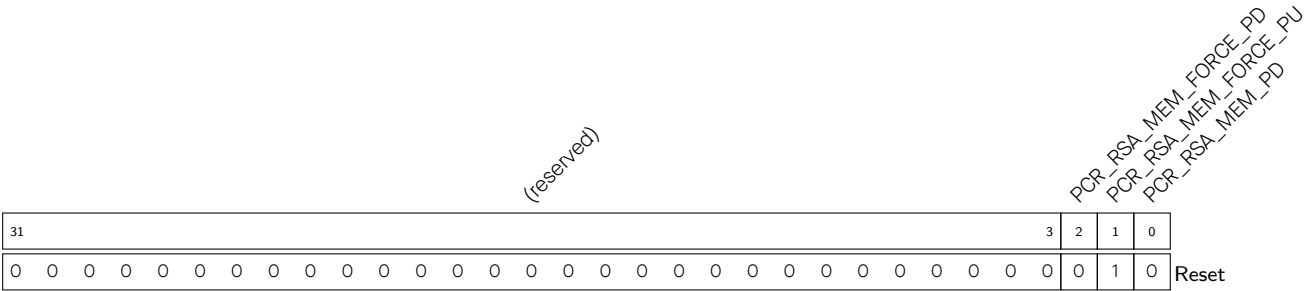
表示 RSA 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.51. PCR_RSA_PD_CTRL_REG (0x00D8)



PCR_RSA_MEM_PD 配置是否使 RSA 内部存储器掉电。

0: 不掉电

1: 强制掉电

(R/W)

PCR_RSA_MEM_FORCE_PU 配置是否强制 RSA 内部存储器上电。

0: 不强制上电

1: 强制上电

(R/W)

PCR_RSA_MEM_FORCE_PD 配置是否强制 RSA 内部存储器掉电。

0: 不强制掉电

1: 强制掉电

(R/W)

Register 6.52. PCR_ECC_CONF_REG (0x00DC)

(reserved)																																PCR_ECC_READY PCR_ECC_RST_EN PCR_ECC_CLK_EN			
31																															3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1			

PCR_ECC_CLK_EN 配置是否使能 ECC 时钟。

0: 不使能

1: 使能

(R/W)

PCR_ECC_RST_EN 配置是否复位 ECC 模块。

0: 不复位

1: 复位

(R/W)

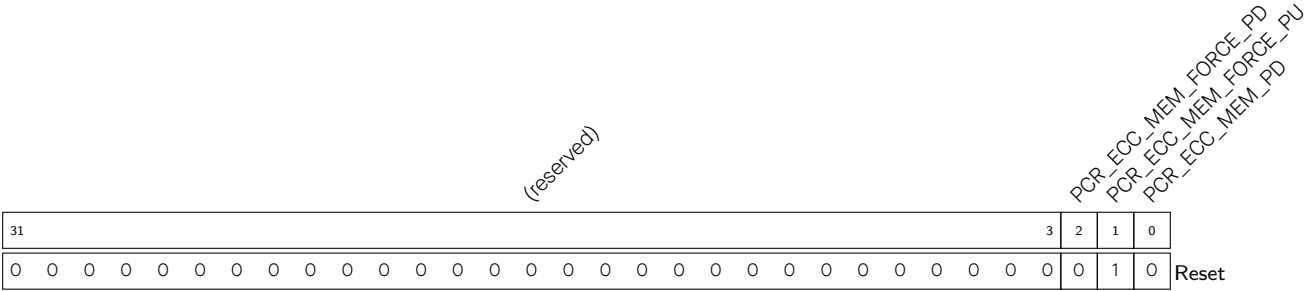
PCR_ECC_READY 表示 ECC 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.53. PCR_ECC_PD_CTRL_REG (0x00E0)



PCR_ECC_MEM_PD 配置是否使 ECC 内部存储器掉电。

0: 不掉电

1: 掉电

(R/W)

PCR_ECC_MEM_FORCE_PU 配置是否强制 ECC 内部存储器上电。

0: 不强制上电

1: 强制上电

(R/W)

PCR_ECC_MEM_FORCE_PD 配置是否强制 ECC 内部存储器掉电。

0: 不强制掉电

1: 强制掉电

(R/W)

Register 6.54. PCR_DS_CONF_REG (0x00E4)

(reserved)																																PCR_DS_READY PCR_DS_RST_EN PCR_DS_CLK_EN			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset		

PCR_DS_CLK_EN 配置是否使能 DS 时钟。

0: 不使能

1: 使能

(R/W)

PCR_DS_RST_EN 配置是否复位 DS 模块。

0: 不复位

1: 复位

(R/W)

PCR_DS_READY 表示 DS 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.55. PCR_HMAC_CONF_REG (0x00E8)

(reserved)																																PCR_HMAC_READY PCR_HMAC_RST_EN PCR_HMAC_CLK_EN			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset	

PCR_HMAC_CLK_EN 配置是否使能 HMAC 时钟。

0: 不使能

1: 使能

(R/W)

PCR_HMAC_RST_EN 配置是否复位 HMAC 模块。

0: 不复位

1: 复位

(R/W)

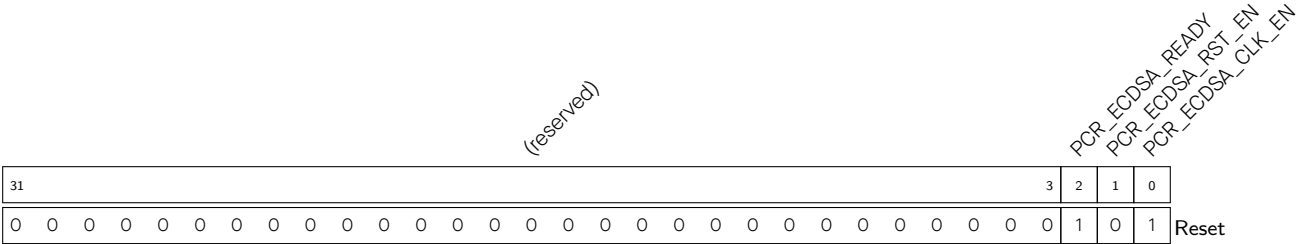
PCR_HMAC_READY 表示 HMAC 模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.56. PCR_ECDSA_CONF_REG (0x00EC)



PCR_ECDSA_CLK_EN 配置是否使能 ECDSA 时钟。

0: 不使能
1: 使能
(R/W)

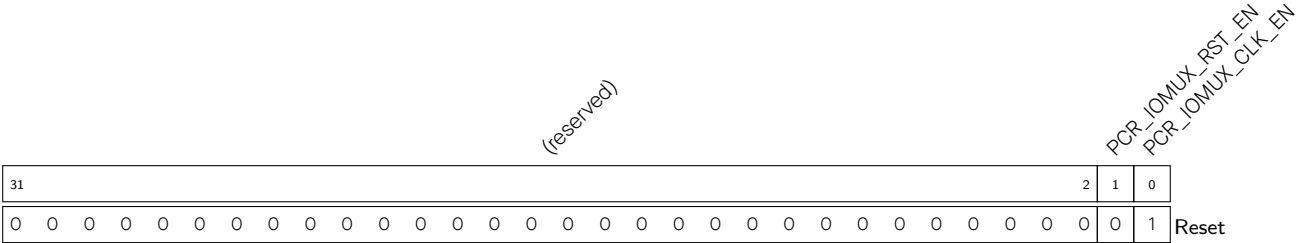
PCR_ECDSA_RST_EN 配置是否复位 ECDSA 模块。

0: 不复位
1: 复位
(R/W)

PCR_ECDSA_READY 表示 ECDSA 模块是否解复位完成。

0: 未完成
1: 完成
(RO)

Register 6.57. PCR_IOMUX_CONF_REG (0x00F0)



PCR_IOMUX_CLK_EN 配置是否使能 IO MUX APB_CLK。

0: 不使能
1: 使能
(R/W)

PCR_IOMUX_RST_EN 配置是否复位 IO MUX 模块。

0: 不复位
1: 复位
(R/W)

Register 6.58. PCR_IOMUX_CLK_CONF_REG (0x00F4)

(reserved)										PCR_IOMUX_FUNC_CLK_EN PCR_IOMUX_FUNC_CLK_SEL										(reserved)													
31											23	22	21	20	19											0							
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PCR_IOMUX_FUNC_CLK_SEL 配置选择 IO MUX 时钟源。

0 (默认): 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

2: 选择 PLL_F48M_CLK 时钟

3: 不选择任何时钟

(R/W)

PCR_IOMUX_FUNC_CLK_EN 配置是否使能 IO MUX 功能时钟。

0: 不使能

1: 使能

(R/W)

Register 6.59. PCR_TRACE_CONF_REG (0x00FC)

(reserved)																															PCR_TRACE_RST PCR_TRACE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
31																															2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_TRACE_CLK_EN 配置是否使能 RISC-V 追踪编码器时钟。

0: 不使能

1: 使能

(R/W)

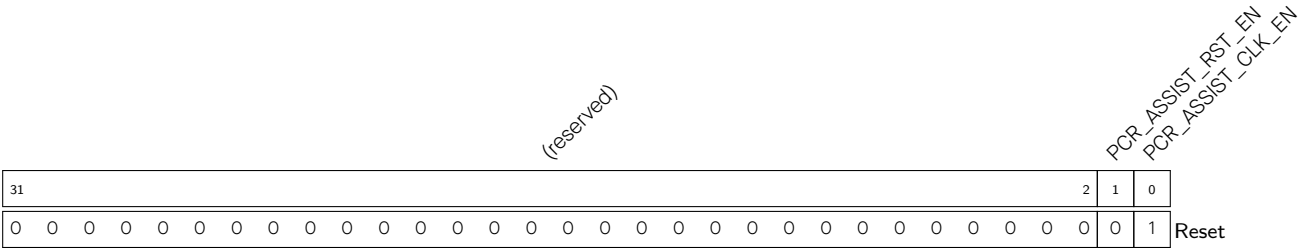
PCR_TRACE_RST_EN 配置是否复位 RISC-V 追踪编码器模块。

0: 不复位

1: 复位

(R/W)

Register 6.60. PCR_ASSIST_CONF_REG (0x0100)



PCR_ASSIST_CLK_EN 配置是否使能辅助调试模块时钟。

0: 不使能

1: 使能

(R/W)

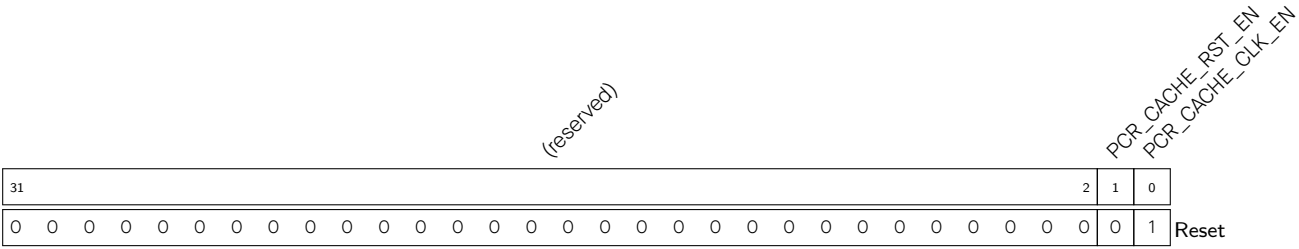
PCR_ASSIST_RST_EN 配置是否复位辅助调试模块。

0: 不复位

1: 复位

(R/W)

Register 6.61. PCR_CACHE_CONF_REG (0x0104)



PCR_CACHE_CLK_EN 配置是否使能 Cache 时钟。

0: 不使能

1: 使能

(R/W)

PCR_CACHE_RST_EN 配置是否复位 Cache 模块。

0: 不复位

1: 复位

(R/W)

Register 6.62. PCR_TIMEOUT_CONF_REG (0x010C)

(reserved)																												PCR_HP_TIMEOUT_RST_EN PCR_CPU_TIMEOUT_RST_EN (reserved)			
31																											3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PCR_CPU_TIMEOUT_RST_EN 配置是否复位 CPU 外设超时保护模块。

0: 不复位

1: 复位

(R/W)

PCR_HP_TIMEOUT_RST_EN 配置是否复位 HP 外设超时保护模块。

0: 不复位

1: 复位

(R/W)

Register 6.63. PCR_SYSCLK_CONF_REG (0x0110)

(reserved)				PCR_CLK_XTAL_FREQ				(reserved)				PCR_SOC_CLK_SEL				(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31	30	24		23	18		17	16	15	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	32			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_SOC_CLK_SEL 配置选择 HP_ROOT_CLK 时钟源。

0: 选择 XTAL_CLK 时钟

1: 选择 RC_FAST_CLK 时钟

2: 选择 PLL_F160M_CLK 时钟

3: 选择 PLL_F240M_CLK 时钟

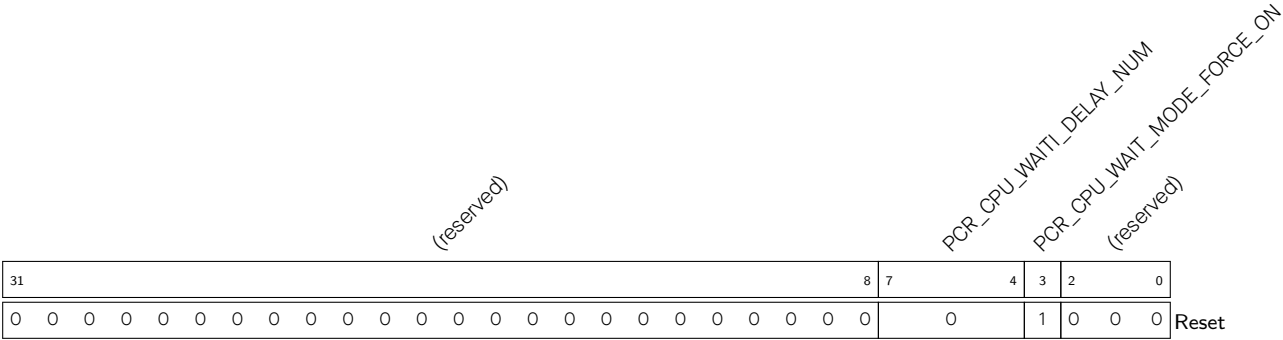
(R/W)

PCR_CLK_XTAL_FREQ 表示 XTAL_CLK 时钟频率。

单位: MHz。

(RO)

Register 6.64. PCR_CPU_WAITI_CONF_REG (0x0114)



PCR_CPU_WAIT_MODE_FORCE_ON 配置是否强制打开 CPU 等待中断模式下的门控时钟。

0: 不强制使能
1: 强制使能

通常情况下，CPU 执行 WFI (wait for interrupt) 指令后会进入等待中断模式。在此模式下 CPU 的时钟门控一直处于关闭状态，直到中断产生，因此可降低功耗。若此位置 1，CPU 的门控时钟会被强制打开，不受 WFI 指令的影响。

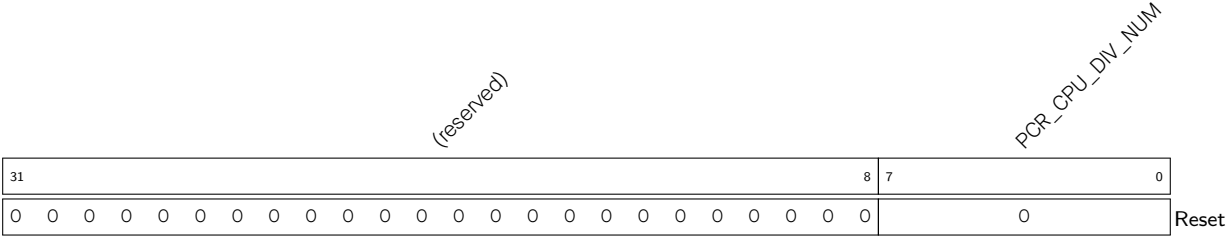
(R/W)

PCR_CPU_WAITI_DELAY_NUM 配置 CPU 在收到 WFI 指令后进入 CPU 等待中断模式后，关闭 CPU 的门控时钟需要的等待周期。

单位：CPU_CLK 周期。

(R/W)

Register 6.65. PCR_CPU_FREQ_CONF_REG (0x0118)

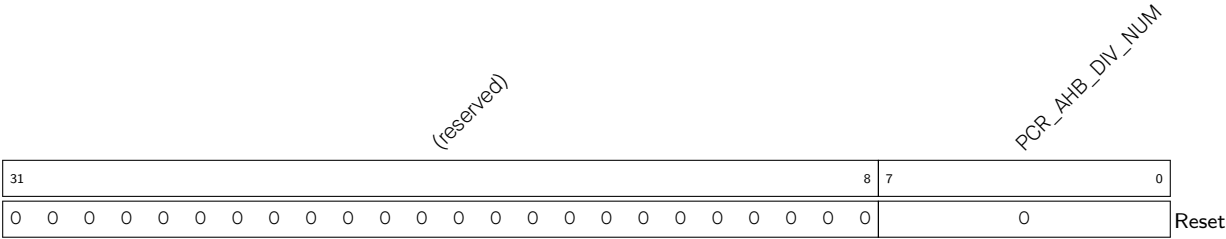


PCR_CPU_DIV_NUM 配置 HP_ROOT_CLK 生成 CPU_CLK 时使用的分频系数。

该字段需要与 [PCR_AHB_DIV_NUM](#) 搭配使用。

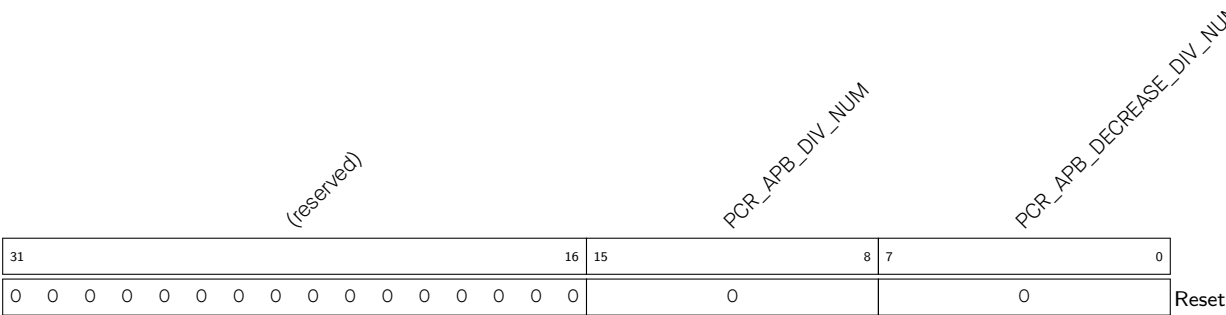
(R/W)

Register 6.66. PCR_AHB_FREQ_CONF_REG (0x011C)



PCR_AHB_DIV_NUM 配置 HP_ROOT_CLK 生成 AHB_CLK 时使用的分频系数。
该字段需要与 **PCR_CPU_DIV_NUM** 搭配使用。
(R/W)

Register 6.67. PCR_APB_FREQ_CONF_REG (0x0120)



PCR_APB_DECREASE_DIV_NUM 配置 AHB_CLK 生成 APB_CLK 时使用的第二级分频系数。
(R/W)

PCR_APB_DIV_NUM 配置 AHB_CLK 生成 APB_CLK 时使用的第一级分频系数。
(R/W)

Register 6.68. PCR_PLL_DIV_CLK_EN_REG (0x0128)

(reserved)																								PCR_PLL_48M_CLK_EN						
																								reserved						
																								PCR_PLL_80M_CLK_EN						
																								PCR_PLL_120M_CLK_EN						
																								PCR_PLL_160M_CLK_EN						
																								PCR_PLL_240M_CLK_EN						
31																					6	5	4	3	2	1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	Reset

PCR_PLL_240M_CLK_EN 配置是否使能 PLL_F240M_CLK 时钟。

0: 不使能

1 (默认): 使能

(R/W)

PCR_PLL_160M_CLK_EN 配置是否使能 PLL_F160M_CLK 时钟。

0: 不使能

1 (默认): 使能

(R/W)

PCR_PLL_120M_CLK_EN 配置是否使能 PLL_F120M_CLK 时钟。

0: 不使能

1 (默认): 使能

(R/W)

PCR_PLL_80M_CLK_EN 配置是否使能 PLL_F80M_CLK 时钟。

0: 不使能

1 (默认): 使能

(R/W)

PCR_PLL_48M_CLK_EN 配置是否使能 PLL_F48M_CLK 时钟。

0: 不使能

1 (默认): 使能

(R/W)

Register 6.69. PCR_CTRL_32K_CONF_REG (0x0130)

(reserved)																PCR_FOSC_TICK_NUM								(reserved)				PCR_32K_SEL																															
31																16																15								8								7				3				2		0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																7																0 0 0 0 0 0								0				Reset															

PCR_32K_SEL 为 TIMER_GROUP 配置选择一个 32 kHz 时钟。

- 0: 选择 XTAL32K_CLK 时钟
- 1: 选择 OSC_SLOW_CLK 时钟
- 2: 选择 RC_SLOW_CLK 时钟
- 3: 选择 RC_FAST_CLK 时钟

(R/W)

PCR_FOSC_TICK_NUM 配置待进入校准模块 RC_FAST_CLK 的分频系数。(当 PCR_32K_SEL 为 4 时生效)。

(R/W)

Register 6.70. PCR_SEC_CONF_REG (0x013C)

(reserved)																															PCR_SEC_RST_EN		PCR_SEC_CLK																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31																													3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

PCR_SEC_CLK_SEL 配置选择片外存储器加密与解密模块的时钟源。

- 0 (默认): 选择 XTAL_CLK 时钟
- 1: 选择 RC_FAST_CLK 时钟
- 2: 选择 PLL_F480M_CLK 时钟
- 3: 不选择任何时钟

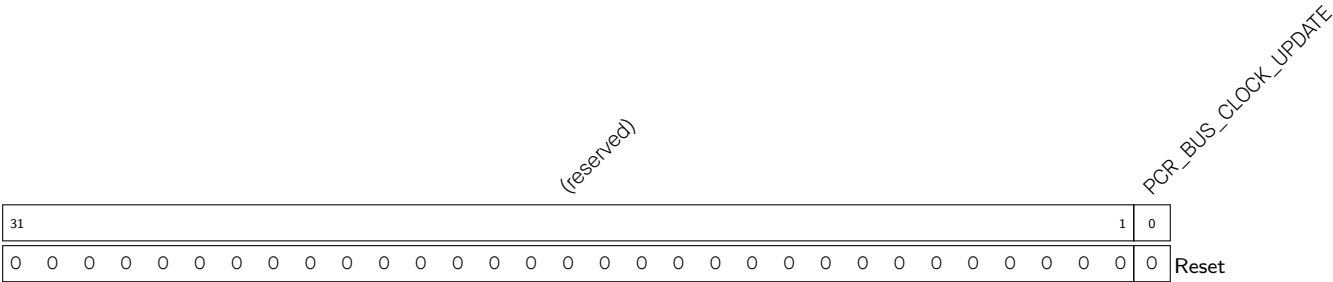
(R/W)

PCR_SEC_RST_EN 配置是否复位 SEC 模块。

- 0: 不复位
- 1: 复位

(R/W)

Register 6.71. PCR_BUS_CLK_UPDATE_REG (0x0144)



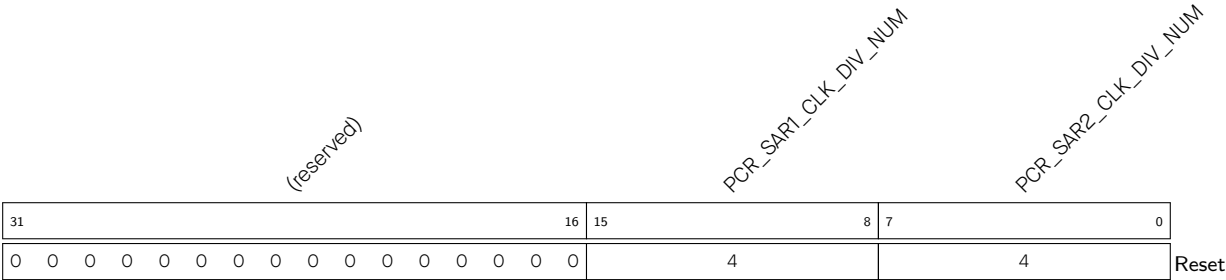
PCR_BUS_CLOCK_UPDATE 配置是否更新 CPU_CLK 分频、AHB_CLK 分频、HP_ROOT_CLK 时钟源选择的配置。

0：不更新配置

1：更新配置

此位在更新完成后自动清零。(R/W/WTC)

Register 6.72. PCR_SAR_CLK_DIV_REG (0x0148)



PCR_SAR2_CLK_DIV_NUM SAR ADC2 采样逻辑时钟的分频系数，用来产生 ADC 模拟电路控制信号。(R/W)

PCR_SAR1_CLK_DIV_NUM SAR ADC 采样逻辑时钟的分频系数，用来产生 ADC 模拟电路控制信号。(R/W)

Register 6.73. PCR_BS_CONF_REG (0x0150)

(reserved)																												PCR_BS_RST_EN PCR_BS_CLK_EN	
31																											2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																													Reset

PCR_BS_CLK_EN 配置是否使能比特传输控制器模块时钟。

0: 不使能

1: 使能

(R/W)

PCR_BS_RST_EN 配置是否复位比特传输控制器模块。

0: 不复位

1: 复位

(R/W)

Register 6.74. PCR_BS_FUNC_CONF_REG (0x0154)

(reserved)								PCR_BS_RX_RST_EN PCR_BS_TX_RST_EN				(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31								25	24	23	22																		0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PCR_BS_TX_RST_EN 配置是否复位比特传输控制器模块的发送侧。

0: 不复位

1: 复位

(R/W)

PCR_BS_RX_RST_EN 配置是否复位比特传输控制器模块的接收侧。

0: 不复位

1: 复位

(R/W)

Register 6.75. PCR_BS_PD_CTRL_REG (0x0158)

(reserved)																															PCR_BS_MEM_FORCE_PD PCR_BS_MEM_FORCE_PU (reserved)			
31																															3	2	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																															1	0	0	Reset

PCR_BS_MEM_FORCE_PU 配置是否强制 BS 模块存储器上电。

0：不强制 BS 模块存储器上电

1：强制 BS 模块存储器上电

(R/W)

PCR_BS_MEM_FORCE_PD 配置是否强制 BS 模块存储器掉电。

0：不强制 BS 模块存储器掉电

1：强制 BS 模块存储器掉电

(R/W)

Register 6.76. PCR_TIMERGROUP_WDT_CONF_REG (0x015C)

(reserved)																															PCR_TG1_WDT_RST_EN PCR_TGO_WDT_RST_EN				
31																														2	1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PCR_TGO_WDT_RST_EN 配置是否复位 timergroup0 模块看门狗功能。

0：不复位

1：复位

(R/W)

PCR_TG1_WDT_RST_EN 配置是否复位 timergroup1 模块看门狗功能。

0：不复位

1：复位

(R/W)

Register 6.77. PCR_TIMERGROUP_XTAL_CONF_REG (0x0160)

(reserved)																															PCR_TGO_XTAL_CLK_EN reserved PCR_TGO_XTAL_RST_EN			
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Reset		

PCR_TGO_XTAL_RST_EN 配置是否复位 timergroup0 模块时钟校准功能。

0: 不复位

1: 复位

(R/W)

PCR_TGO_XTAL_CLK_EN 配置是否使能 timergroup0 模块时钟校准功能。

0: 不使能

1: 使能

(R/W)

Register 6.78. PCR_TCM_MEM_MONITOR_CONF_REG (0x016C)

(reserved)																															PCR_TCM_MEM_MONITOR_READY PCR_TCM_MEM_MONITOR_RST_EN PCR_TCM_MEM_MONITOR_CLK_EN			
31																															3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset		

PCR_TCM_MEM_MONITOR_CLK_EN 配置是否使能内存访问监控模块时钟。

0: 不使能

1: 使能

(R/W)

PCR_TCM_MEM_MONITOR_RST_EN 配置是否复位内存访问监控模块。

0: 不复位

1: 复位

(R/W)

PCR_TCM_MEM_MONITOR_READY 表示内存访问监控模块是否解复位完成。

0: 未完成

1: 完成

(RO)

Register 6.79. PCR_PSRAM_MEM_MONITOR_CONF_REG (0x0170)

(reserved)																												PCR_PSRAM_MEM_MONITOR_READY PCR_PSRAM_MEM_MONITOR_RST_EN PCR_PSRAM_MEM_MONITOR_CLK_EN			
31																											3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	Reset

PCR_PSRAM_MEM_MONITOR_CLK_EN 配置是否使能外存访问监控模块时钟。

0：不使能

1：使能

(R/W)

PCR_PSRAM_MEM_MONITOR_RST_EN 配置是否复位外存访问监控模块。

0：不复位

1：复位

(R/W)

PCR_PSRAM_MEM_MONITOR_READY 表示外存访问监控模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 6.80. PCR_HPCORE_0_PD_CTRL_REG (0x0178)

(reserved)																																PCR_HPCORE_0_MEM_FORCE_PD PCR_HPCORE_0_MEM_FORCE_PU (reserved)				
31																															3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Reset

PCR_HPCORE_0_MEM_FORCE_PU 配置是否强制 HP CORE0 存储器上电。

0: 不强制上电

1: 强制上电

(R/W)

PCR_HPCORE_0_MEM_FORCE_PD 配置是否强制 HP CORE0 存储器掉电。

0: 不强制掉电

1: 强制掉电

(R/W)

Register 6.81. PCR_SDIO_SLAVE_CONF_REG (0x017C)

(reserved)																															PCR_SDIO_SLAVE_RST_EN PCR_SDIO_SLAVE_CLK_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31																															2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

PCR_SDIO_SLAVE_RST_EN 配置是否复位 SDIO slave 模块。

0: 不复位

1: 复位

(R/W)

PCR_SDIO_SLAVE_CLK_EN 配置是否使能 SDIO slave 模块时钟。

0: 不使能

1: 使能

(R/W)

Register 6.82. PCR_SYSCLK_FREQ_QUERY_0_REG (0x0120)

(reserved)														PCR_PLL_FREQ																PCR_FOSC_FREQ																																																																																																																																											
31														18														17																8																7																0																																																																																													
0														0														0														0																0																0																0																0																96																8																Reset															

PCR_FOSC_FREQ 表示 RC_FAST_CLK 时钟频率。
单位：MHz
(HRO)

PCR_PLL_FREQ 表示 PLL_F480M_CLK 时钟频率。
单位：MHz
(HRO)

Register 6.83. PCR_DATE_REG (0x0FFC)

(reserved)				PCR_DATE																										
31				28	27																									0
0	0	0	0	0x2210080																										Reset

PCR_DATE 版本控制寄存器。(R/W)

6.5.2 低功耗系统时钟复位寄存器

本小节前缀为 LPPERI 的寄存器（最后两个）地址为相对于低功耗外设寄存器（LPPERI）基地址的地址偏移量（相对地址），前缀为 LP_AON 的寄存器（倒数第三、四个）地址为相对于低功耗常开寄存器（LP_AON）基地址的地址偏移量（相对地址），其余所有地址均为相对于低功耗时钟复位寄存器（LP_CLKRST）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

Register 6.84. LP_CLKRST_LP_CLK_CONF_REG (0x0000)

[illegible]

LP_CLKRST_SLOW_CLK_SEL 配置 LP_SLOW_CLK 的时钟源。

0: RC_SLOW_CLK

1: XTAL32K_CLK

2: OSC_SLOW_CLK

3: 无效值。没有作用

(R/W)

LP_CLKRST_FAST_CLK_SEL 配置 LP_FAST_CLK 的时钟源。

0: RC_FAST_CLK

1: XTAL_D2_CLK

(R/W)

Register 6.85. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)

(reserved)																						LP_CLKRST_LPBUS_OEN LP_CLKRST_RING_OEN LP_CLKRST_FAST_OEN LP_CLKRST_SLOW_OEN LP_CLKRST_CORE_OEN (reserved)) LP_CLKRST_XTAL32K_OEN LP_CLKRST_FOSC_OEN LP_CLKRST_SOSC_OEN LP_CLKRST_AON_FAST_OEN LP_CLKRST_AON_SLOW_OEN										
31																				11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	Reset

LP_CLKRST_AON_SLOW_OEN 配置是否使能 LP_DYN_SLOW_CLK 输出到 pad 的时钟门控。
0：禁止时钟信号通过
1：使能时钟信号通过
(R/W)

LP_CLKRST_AON_FAST_OEN 配置是否使能 LP_DYN_FAST_CLK 输出到 pad 的时钟门控。
0：禁止时钟信号通过
1：使能时钟信号通过
(R/W)

LP_CLKRST_SOSC_OEN 配置是否使能 OSC_SLOW_CLK 输出到 pad 的时钟门控。
0：禁止时钟信号通过
1：使能时钟信号通过
(R/W)

LP_CLKRST_FOSC_OEN 配置是否使能 RC_FAST_CLK 输出到 pad 的时钟门控。
0：禁止时钟信号通过
1：使能时钟信号通过
(R/W)

LP_CLKRST_XTAL32K_OEN 配置是否使能 XTAL32K_CLK 输出到 pad 的时钟门控。
0：禁止时钟信号通过
1：使能时钟信号通过
(R/W)

见下页...

Register 6.85. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)

接上页...

LP_CLKRST_CORE_EFUSE_OEN 配置是否使能 EFUSE_CTRL 时钟输出到 pad 的时钟门控。

0: 禁止时钟信号通过

1: 使能时钟信号通过

(R/W)

LP_CLKRST_SLOW_OEN 配置是否使能 LP_SLOW_CLK 输出到 pad 的时钟门控。

0: 禁止时钟信号通过

1: 使能时钟信号通过

(R/W)

LP_CLKRST_FAST_OEN 配置是否使能 LP_FAST_CLK 输出到 pad 的时钟门控。

0: 禁止时钟信号通过

1: 使能时钟信号通过

(R/W)

LP_CLKRST_RNG_OEN 配置是否使能 RNG 时钟输出到 pad 的时钟门控。

0: 禁止时钟信号通过

1: 使能时钟信号通过

(R/W)

LP_CLKRST_LPBUS_OEN 配置是否使能 LP 总线时钟输出到 pad 的时钟门控。

0: 禁止时钟信号通过

1: 使能时钟信号通过

(R/W)

Register 6.86. LP_CLKRST_LP_CLK_EN_REG (0x0008)

Diagram of the LP_CLKRST_FAST_ORI_GATE register structure:

- Bit 31: LP_CLKRST_FAST_ORI_GATE
- Bit 30: LP_CLKRST_FAST_ORI_GATE
- Bits 29-1: (reserved)
- Bit 0: Reset

LP_CLKRST_FAST_ORI_GATE 配置 LP_FAST_CLK 的时钟门控。

0: 无效。时钟门控由硬件 FSM 控制

1: 强制时钟通过门控

(R/W)

Register 6.87. LP_CLKRST_LP_RST_EN_REG (0x000C)

LP_CLKRST_ANA_PERI_RESET_EN																																	
LP_CLKRST_WDT_RESET_EN																																	
LP_CLKRST_LP_TIMER_RESET_EN																																	
LP_CLKRST_AON_EFUSE_CORE_RESET_EN																																	
(reserved)																																	
31	30	29	28	0																													
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_CLKRST_AON_EFUSE_CORE_RESET_EN 配置复位 EFUSE_CTRL 常开部分。

0: 无效

1: 复位

(R/W)

LP_CLKRST_LP_TIMER_RESET_EN 配置复位 RTC 定时器。

0: 无效

1: 复位

(R/W)

LP_CLKRST_WDT_RESET_EN 配置复位 RWDT 和超级看门狗定时器。

0: 无效

1: 复位

(R/W)

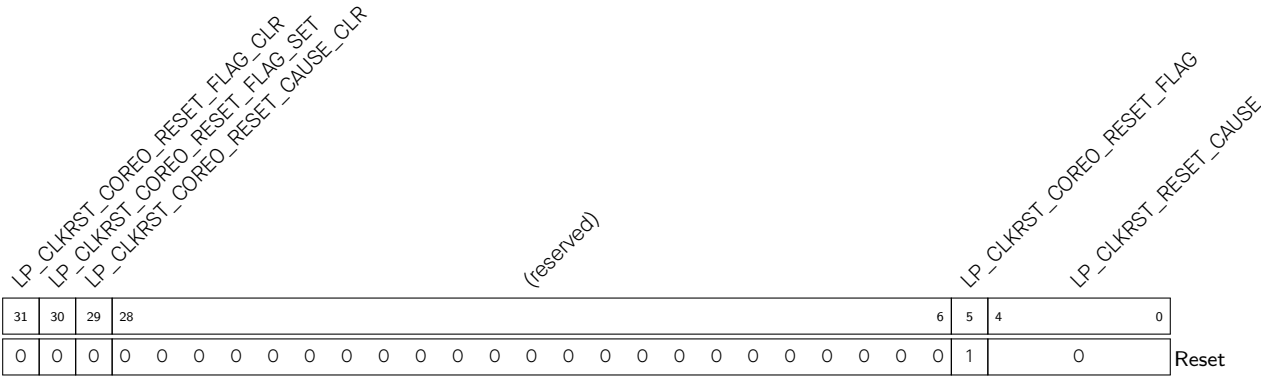
LP_CLKRST_ANA_PERI_RESET_EN 配置复位模拟外设，包括欠压检测器。

0: 无效

1: 复位

(R/W)

Register 6.88. LP_CLKRST_RESET_CAUSE_REG (0x0010)



- LP_CLKRST_RESET_CAUSE 表示复位原因。(RO)
- LP_CLKRST_COREO_RESET_FLAG 表示发生的复位是否为 LP_CLKRST_RESET_CAUSE 中记录的复位。
0: 非法复位
1: 已记录的复位
(RO)
- LP_CLKRST_COREO_RESET_CAUSE_CLR 写 1 清除 LP_CLKRST_RESET_CAUSE。(WT)
- LP_CLKRST_COREO_RESET_FLAG_SET 写 1 置位 LP_CLKRST_COREO_RESET_FLAG。(WT)
- LP_CLKRST_COREO_RESET_FLAG_CLR 写 1 清除 LP_CLKRST_COREO_RESET_FLAG。(WT)

Register 6.89. LP_CLKRST_CPU_RESET_REG (0x0014)

[illegible]

LP_CLKRST_RTC_WDT_CPU_RESET_LENGTH 配置 RWDT 复位 CPU 的时间长度。

单位: LP_DYN_FAST_CLK 周期。

(R/W)

LP_CLKRST_RTC_WDT_CPU_RESET_EN 配置 RWDT 是否能复位 CPU。

O: RWDT 超时时无法复位 CPU

1: RWDT 超时复位 CPU

(R/W)

LP_CLKRST_CPU_STALL_WAIT 配置 CPU 停顿状态 (CPU Stall) 到复位之间的时间间隔。

单位：LP_DYN_FAST_CLK 周期。

(R/W)

LP_CLKRST_CPU_STALL_EN 配置在 RWDT 和软件复位 CPU 之前是否进入 CPU 停顿状态。

0: 否

1: 是

(R/W)

Register 6.90. LP_CLKRST_FOSC_CNTL_REG (0x0018)

[illegible]

LP_CLKRST_FOSC_DFREQ 配置 RC_FAST_CLK 频率，值越大，时钟周期越大。(R/W)

Register 6.91. LP_CLKRST_CLK_TO_HP_REG (0x0020)

Diagram illustrating the Reset Vector field (bits 31:0) in the Reset Vector register. The field is divided into four 8-bit sections:

- Bits 31:28: LP_CLKRST_ICG_HP_FOSC (reserved)
- Bits 27:20: LP_CLKRST_ICG_HP_SOSC (reserved)
- Bits 19:12: LP_CLKRST_ICG_HP_XTAL32K (reserved)
- Bits 11:4: (reserved)
- Bits 3:0: Reset

LP_CLKRST_ICG_HP_XTAL32K 配置 XTAL32K_CLK 到 HP System 的时钟门控。

0: 时钟无法到达 HP System

1: 时钟能够到达 HP System

(R/W)

LP_CLKRST_ICG_HP_SOSC 配置 RC_SLOW_CLK 到 HP System 的时钟门控。

0: 时钟无法到达 HP System

1: 时钟能够到达 HP System

(R/W)

LP_CLKRST_ICG_HP_FOSC 配置 RC_FAST_CLK 到 HP System 的时钟门控。

0: 时钟无法到达 HP System

1: 时钟能够到达 HP System

(R/W)

Register 6.92. LP_CLKRST_LPMEM_FORCE_REG (0x0024)

Diagram of the LP_CLKRST_LPMEM_CLK_FORCE_ON register. The register is 32 bits wide, with bit 31 labeled 'LP_CLKRST_LPMEM_CLK_FORCE_ON' and bit 0 labeled 'Reset'. The register is divided into two sections: a 1-bit section for bit 31 and a 31-bit section for bits 30-0. The 31-bit section is labeled '(reserved)'.

LP_CLKRST_LPMEM_CLK_FORCE_ON 配置强制打开 LP 存储器的时钟门控。

0: 无效。时钟门控由硬件 FSM 控制

1: 强制打开时钟门控

(R/W)

Register 6.93. LP_CLKRST_LPPERI_REG (0x0028)

(reserved)				LP_CLKRST_LP_SEL_XTAL32K				LP_CLKRST_LP_SEL_XTAL				LP_CLKRST_LP_SEL_OSC_FAST				LP_CLKRST_LP_SEL_OSC_SLOW				(reserved)				LP_CLKRST_LP_BLETIMER_DIV_NUM								(reserved)			
31	30	29	28	27	26	25	24	23					12	11													0								
0	1	0	0	0	0	0		0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset					

LP_CLKRST_LP_BLETIMER_DIV_NUM 配置 RTC BLE 定时器的时钟分频器的分频系数, RTC BLE 定时器的工作时钟频率 = 源时钟频率 / ((LP_CLKRST_LP_BLETIMER_DIV_NUM - 1) / 2) (R/W)

LP_CLKRST_LP_SEL_OSC_SLOW 配置 RTC BLE 定时器的时钟源
1: 选择 RC_SLOW_CLK 为 RTC BLE 定时器的源时钟
0: 禁止 RC_SLOW_CLK 为 RTC BLE 定时器的源时钟
(R/W)

LP_CLKRST_LP_SEL_OSC_FAST 配置 RTC BLE 定时器的时钟源
1: 选择 RC_FAST_CLK 为 RTC BLE 定时器的源时钟
0: 禁止 RC_FAST_CLK 为 RTC BLE 定时器的源时钟 (R/W)

LP_CLKRST_LP_SEL_XTAL 配置 RTC BLE 定时器的时钟源
1: 选择 XTAL_CLK 为 RTC BLE 定时器的源时钟
0: 禁止 XTAL_CLK 为 RTC BLE 定时器的源时钟
(R/W)

LP_CLKRST_LP_SEL_XTAL32K 配置 RTC BLE 定时器的时钟源
1: 选择 XTAL32K_CLK 为 RTC BLE 定时器的源时钟
0: 禁止 XTAL32K_CLK 为 RTC BLE 定时器的源时钟
(R/W)

Register 6.94. LP_CLKRST_XTAL32K_REG (0x002C)

[illegible]

LP_CLKRST_DRES_XTAL32K 配置 DRES。(R/W)

LP_CLKRST_DGM_XTAL32K 配置 DGM。(R/W)

LP_CLKRST_DBUF_XTAL32K 配置 DBUF。(R/W)

LP_CLKRST_DAC_XTAL32K 配置 DAC。(R/W)

Register 6.95. LP_CLKRST_DATE_REG (0x03FC)

Diagram of the LP_CLKRST_CLKRST register. The register is 32 bits wide. The top 2 bits (bits 31-30) are labeled LP_CLKRST_CLK_EN. The bottom 30 bits (bits 0-31) are labeled LP_CLKRST_CLKRST_DATE. The register is shown with a value of 0x2207280. The bit 0 is labeled Reset.

LP_CLKRST_CLKRST_DATE 版本控制寄存器。(R/W)

LP_CLKRST_CLK_EN 配置强制开启寄存器的配置的时钟。0: 无效值

1: 强制开启

(R/W)

Register 6.96. LP_AON_SYS_CFG_REG (0x0034)

Diagram of the LP_AON_HPSYS_SW_RESET register. The register is 32 bits wide. Bit 31 is labeled LP_AON_HPSYS_SW_RESET. Bits 30 through 0 are labeled (reserved). The register value is shown as 0.

LP_AON_HPSYS_SW_RESET 配置系统的软件复位。

0: 不复位

1: 复位

(WT)

Register 6.97. LP_AON_CPUCORE0_CFG_REG (0x0038)

Diagram of the LP_AON_CPU_CORE0_SW_RESET register (32 bits). The register is divided into three sections: bits 31-29 are reserved; bits 28-27 are reserved; and bits 26-0 are reserved. The register is labeled 'LP_AON_CPU_CORE0_SW_RESET'.

LP_AON_CPU_CORE0_SW_RESET 配置 CPU 的软件复位。

0: 不复位

1: 复位

(WT)

Register 6.98. LPPERI_CLK_EN_REG (0x0000)

(reserved)		LPPERI_EFUSE_CK_EN				(reserved)		LPPERI_EFUSE_CK_EN				(reserved)																				0		
31	30	29		25	24	23																											0	
0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LPPERI_RNG_CLK_EN 配置是否使能 RNG 模块的时钟门控。

- 0: 禁止时钟信号通过
 - 1: 使能时钟信号通过
- (R/W)

LPPERI_EFUSE_CLK_EN 配置是否使能 eFuse 控制器模块的时钟门控。

- 0: 禁止时钟信号通过
 - 1: 使能时钟信号通过
- (R/W)

Register 6.99. LPPERI_RESET_EN_REG (0x0004)

(reserved)		LPPERI_EFUSE_RESET_EN				(reserved)		LPPERI_RNG_RESET_EN				(reserved)																				0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
31	30	29		25	24	23																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								

LPPERI_RNG_RESET_EN 配置 RNG 控制器模块的软件复位。

- 0: 不复位
 - 1: 复位
- (R/W)

LPPERI_EFUSE_RESET_EN 配置 eFuse 控制器模块的软件复位。

- 0: 不复位
 - 1: 复位
- (R/W)

第 7 章

芯片 Boot 控制

7.1 概述

芯片在上电或硬件复位时，可以通过 strapping 管脚和 eFuse 参数配置如下启动参数，无需微处理器的参与：

- 芯片启动模式
 - Strapping 管脚：GPIO26、GPIO27 和 GPIO28
- SDIO 输入采样沿和输出驱动沿控制
 - Strapping 管脚：GPIO25 和 MTDI
- ROM 代码日志打印
 - Strapping 管脚：GPIO27
 - eFuse 参数：EFUSE_UART_PRINT_CONTROL 和 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT
- JTAG 信号源
 - Strapping 管脚：GPIO7
 - eFuse 参数：EFUSE_DIS_PAD_JTAG、EFUSE_DIS_USB_JTAG 和 EFUSE_JTAG_SEL_ENABLE
- 晶振频率选择
 - Strapping 管脚：MTMS（仅在 Joint Download Boot 模式下有效）
 - eFuse 参数：EFUSE_XTAL_48M_SEL 和 EFUSE_XTAL_48M_SEL_MODE

上述 eFuse 位的默认值均为 0，也就是说没有烧写过。eFuse 只能烧写一次，一旦烧写为 1，便不能恢复为 0。有关烧写 eFuse 的信息，请参考章节 2 [eFuse 控制器 \(eFuse\) \[to be added later\]](#)。

在芯片复位过程中（请参考章节 6 [复位和时钟](#)），硬件将采样 strapping 管脚电平存储到锁存器中，并一直保持到芯片掉电或下一次芯片复位。strapping 管脚的锁存值可以通过软件从寄存器 [GPIO_STRAPPING](#) 中读取。

7.2 功能描述

本小节主要介绍芯片复位时的功能以及控制该功能使用到的 strapping 管脚和 eFuse 组合模式。

注意： 请使用本章节所介绍的组合，其它组合可能会导致不可控结果。

7.2.1 默认配置

上述 strapping 管脚如果没有连接任何电路或连接的电路处于高阻抗状态，则其默认值（即逻辑电平值）取决于管脚内部弱上拉/下拉电阻在复位时的状态。

表 7.2-1. Strapping 管脚默认配置

Strapping 管脚	默认配置	值
GPIO25	浮空	–
GPIO26	浮空	–
GPIO27	上拉	1
GPIO28	上拉	1
GPIO7	浮空	–
MTMS	浮空	–
MTDI	浮空	–

如需改变 strapping 管脚的默认值，用户可以应用外部下拉/上拉电阻，或者应用主机 MCU 的 GPIO 来控制 ESP32-C5 上电复位时的 strapping 管脚电平。复位释放后，strapping 管脚和普通管脚功能相同。

7.2.2 芯片启动模式控制

复位释放后，GPIO26、GPIO28 和 GPIO27 共同决定启动模式，详见表 7.2-2 芯片启动模式控制。

表 7.2-2. 芯片启动模式控制

启动模式	GPIO26	GPIO27	GPIO28
SPI Boot¹	任意值	任意值	1¹
Joint Download Boot 0 ²	任意值	1	0
Joint Download Boot 1 ³	0	0	0

¹ 加粗表示默认值和默认配置。

² Joint Download Boot 0 模式下支持以下下载方式：

- USB-Serial-JTAG Download Boot
- UART Download Boot

³ Joint Download Boot 1 模式下支持以下下载方式：

- UART Download Boot
- SDIO Download Boot

在 SPI Boot 模式下，ROM 引导加载程序通过从 SPI flash 中读取程序来启动系统。

在 Joint Download Boot 0 模式下，用户可通过 UART0、USB 或 SPI Slave 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并运行 SRAM 中的程序。

在 Joint Download Boot 1 模式下，用户可通过 UART0、SDIO 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并运行 SRAM 中的程序。

芯片启动的具体流程见图 7.2-1 芯片启动流程。

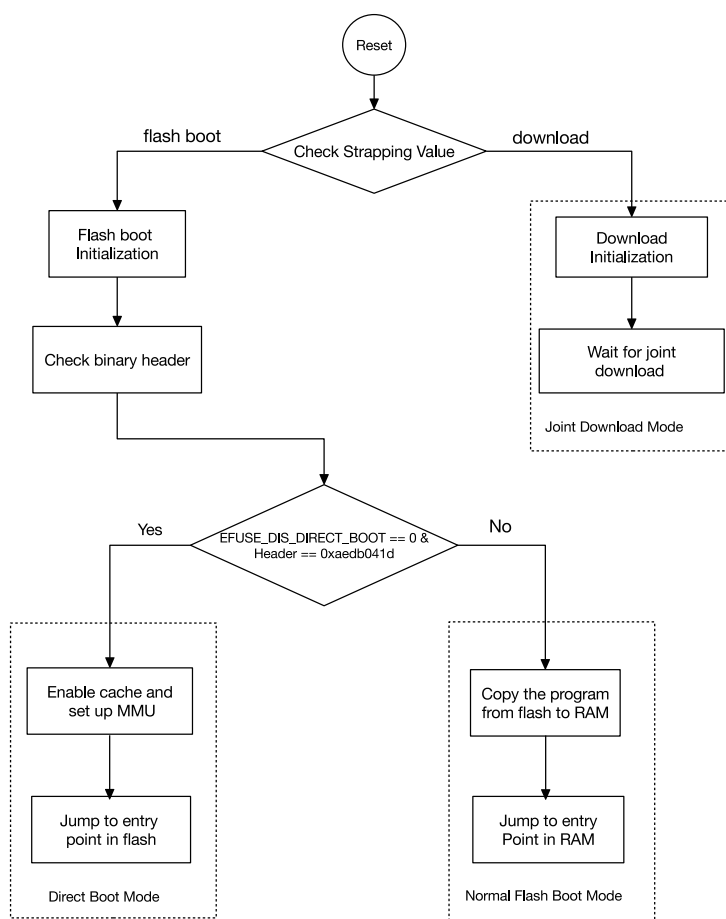


图 7.2-1. 芯片启动流程

下面几个 eFuse 可用于控制启动模式的具体行为：

• EFUSE_DIS_FORCE_DOWNLOAD

- 如果此 eFuse 设置为 0（默认），软件可通过设置 LP_AON_FORCE_DOWNLOAD_BOOT，触发 CPU 复位，将芯片启动模式强制从 SPI Boot 模式切换至 Joint Download Boot 模式。这种情况下，硬件会将寄存器 [GPIO_STRAPPING\[4:3\]](#) 的值从“1x”覆盖为“01”；
- 如果此 eFuse 设置为 1，则禁用 LP_AON_FORCE_DOWNLOAD_BOOT，寄存器 [GPIO_STRAPPING](#) 的值不受影响。

• EFUSE_DIS_DOWNLOAD_MODE

如果此 eFuse 设置为 1，则禁用 Joint Download Boot 模式，寄存器 [GPIO_STRAPPING](#) 的值则不受 LP_AON_FORCE_DOWNLOAD_BOOT 的影响。

• EFUSE_ENABLE_SECURITY_DOWNLOAD

如果此 eFuse 设置为 1，则在 Joint Download Boot 模式下，只允许读取、写入和擦除明文 flash，不支持 SRAM 或寄存器操作。如已禁用 Joint Download Boot 模式，请忽略此 eFuse。

• EFUSE_DIS_DIRECT_BOOT

如果此 eFuse 设置为 1，则禁用 Direct Boot 模式。

USB 串口/JTAG 控制器可将芯片从 SPI Boot 模式强制切换到 Joint Download Boot 模式，或从 Joint Download Boot 模式强制切换到 SPI Boot 模式。更多信息，请参考章节 [28 USB 串口/JTAG 控制器](#)。

7.2.3 SDIO 输入采样沿和输出驱动沿控制

GPIO25 和 MTDI 管脚可用于调节 SDIO 输入采样沿和输出驱动沿。详见表 [7.2-3 SDIO 输入采样沿/输出驱动沿控制](#)。

表 7.2-3. SDIO 输入采样沿/输出驱动沿控制

沿控制	GPIO25	MTDI
下降沿采样下降沿输出	0	0
下降沿采样上升沿输出	0	1
上升沿采样下降沿输出	1	0
上升沿采样上升沿输出	1	1

¹ GPIO25 和 MTDI 默认浮空，以上均非默认配置。

7.2.4 ROM 代码日志打印控制

系统启动过程中，ROM 代码日志可打印至：

- (默认) UART0 和 USB 串口/JTAG 控制器
- UART0
- USB 串口/JTAG 控制器

要将 ROM 日志打印至 **UART0** 或 **USB 串口/JTAG 控制器**，详见下文。

EFUSE_UART_PRINT_CONTROL 和 GPIO27 控制 ROM 日志打印至 **UART0**，如表 [7.2-4 UART0 ROM 日志打印控制](#) 所示。

表 7.2-4. UART0 ROM 日志打印控制

UART0 ROM 日志打印	Register ²	eFuse ³	GPIO27
启动过程中，ROM 代码日志始终打印至 UART0	0	0(0b00)	x ⁴
启动过程中使能打印		1(0b01)	0
启动过程中关闭打印			1
启动过程中关闭打印		2(0b10)	0
启动过程中使能打印			1
启动过程中关闭打印		3(0b11)	x
启动过程中关闭打印	1	x	x

¹ 加粗表示默认值和默认配置。
² 寄存器：LP_AON_STORE4_REG[0]
³ eFuse：EFUSE_UART_PRINT_CONTROL
⁴ x: x 表示该值被忽略，任何取值不影响该状态。

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 控制 **USB 串口/JTAG 控制器** ROM 日志打印，如表 [7.2-5 USB 串口/JTAG ROM 日志打印控制](#) 所示。

表 7.2-5. USB 串口/JTAG ROM 日志打印控制

USB 串口/JTAG ROM 日志打印控制	EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT
使能	0
关闭	1
	忽略

¹ 加粗表示默认值和默认配置。

7.2.5 JTAG 信号源控制

在系统启动早期阶段，GPIO7 可用于控制 JTAG 信号源。该管脚没有内部上下拉电阻，strapping 的值必须由不处于高阻抗状态的外部电路控制。

GPIO7 与 EFUSE_DIS_PAD_JTAG、EFUSE_DIS_USB_JTAG 和 EFUSE_JTAG_SEL_ENABLE 共同控制 JTAG 信号源，见表 7.2-6 JTAG 信号源控制。

表 7.2-6. JTAG 信号源控制

JTAG 信号源	eFuse 1 ²	eFuse 2 ³	eFuse 3 ⁴	GPIO7
USB 串口/JTAG 控制器 ⁶	0	0	0	x ⁵
			1	1
JTAG 管脚 MTDI、MTCK、MTMS 和 MTDO		x	x	0
	1	1		x
USB 串口/JTAG 控制器 ⁶		0		
JTAG 关闭		x		
		1		

- ¹ 加粗表示默认值和默认配置。
- ² eFuse 1: EFUSE_DIS_PAD_JTAG
- ³ eFuse 2: EFUSE_DIS_USB_JTAG
- ⁴ eFuse 3: EFUSE_JTAG_SEL_ENABLE
- ⁵ x: x 表示该值被忽略，任何取值不影响该状态。
- ⁶ 在 Joint Download Boot 1 模式下，USB 串口/JTAG 控制器被强制关闭，JTAG 信号源只能来自 JTAG 管脚。如果 PAD_JTAG 也被禁用，则 JTAG 功能关闭。

7.2.6 晶振频率选择

ESP32-C5 可选 40 MHz 和 48 MHz 晶振频率。

系统启动模式与 MTMS、EFUSE_XTAL_48M_SEL 和 EFUSE_XTAL_48M_SEL_MODE 共同控制晶振频率。

表 7.2-7. 晶振频率选择

晶振频率	启动模式	eFuse 1 ²	eFuse 2 ³	MTMS
40 MHz	SPI Boot	x ⁴	偶数个 1	x
48 MHz			奇数个 1	
40 MHz	Joint Download Boot 0/1	1	偶数个 1	
48 MHz			奇数个 1	
40 MHz		0	x	0 ⁵
48 MHz				1 ⁵

¹ 加粗表示默认值和默认配置。

² eFuse 1: EFUSE_XTAL_48M_SEL_MODE

³ eFuse 2: EFUSE_XTAL_48M_SEL

⁴ x: x 表示该值被忽略，任何取值不影响该状态。

⁵ 在 Joint Download Boot 0/1 模式下，用户需要指定 strapping 管脚 MTMS 的电平。

第 8 章

事件任务矩阵 (ETM)

8.1 概述

事件任务矩阵 (ETM) 外设包含 50 个可配置通道。每个通道可以将任意指定外设的事件映射到任意指定外设的任务，从而触发外设执行指定任务，无需 CPU 干预。

8.2 特性

事件任务矩阵具有以下特性：

- 支持从多个外设接收多种事件
- 支持为多个外设生成多种任务
- 拥有 50 个可独立配置的 ETM 通道
- 每个通道接收到的事件可以是所有事件中的任意一个，每个通道接收到的事件可以映射到任意的任务上输出
- 每个 ETM 通道都可以独立使能或关闭。通道关闭时，不会响应所配置的事件，也不会生成要映射到的任务
- 支持查看每个事件和任务的触发状态
- 能够产生事件、接收任务的外设有：GPIO、LED PWM、通用定时器、RTC 定时器、系统定时器、MCPWM、温度传感器、ADC、I2S、LP CPU、GDMA 和 PMU

由于 50 个 ETM 通道的功能和操作都是相同的。以下部分将 ETM 通道统称为通道 n （其中 n 范围从 0 到 49）。

8.3 功能描述

8.3.1 架构

事件任务矩阵有 50 个独立通道，每个通道可以选择所有事件中的任意一个作为该通道的输入（参考章节 8.3.2），并且可以选择映射事件到任意任务（参考章节 8.3.3）。每个通道都有独立的使能配置位（参考章节 8.3.6）。

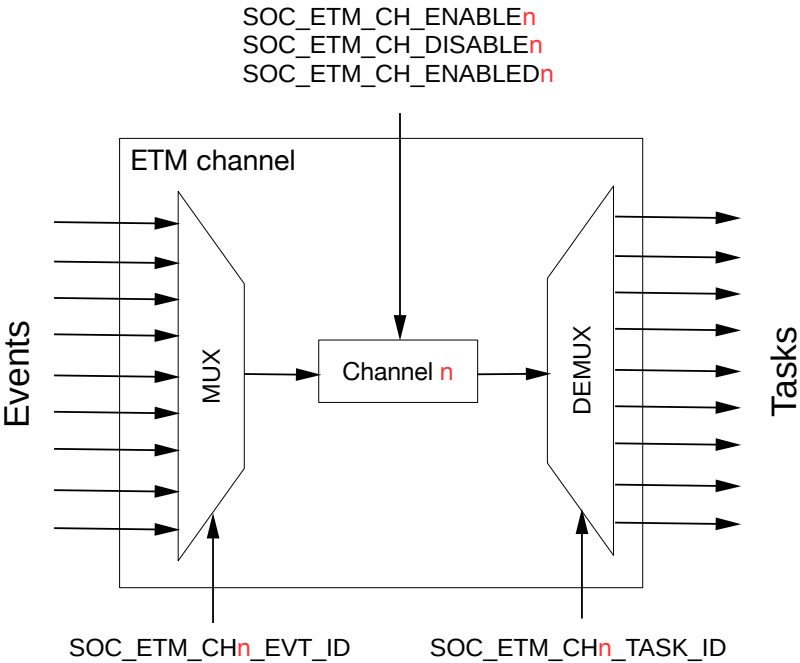


图 8.3-1. 事件任务矩阵通道n架构

图 8.3-1 展示了单个事件任务矩阵通道的结构，其中 `SOC_ETM_CHn_EVT_ID` 字段配置 MUX 选择任意一个事件作为通道 `n` 的输入，`SOC_ETM_CHn_TASK_ID` 字段配置 DEMUX 以映射通道 `n` 所选择的事件到任意一个任务。字段 `SOC_ETM_CH_ENABLEn` 和 `SOC_ETM_CH_DISABLEn` 用于使能或关闭通道 `n`，字段 `SOC_ETM_CH_ENABLEDn` 用于指示通道 `n` 的开启或关闭状态。

8.3.2 事件

每个事件任务矩阵通道可以通过字段 `SOC_ETM_CHn_EVT_ID` 指定选择接收的事件。字段 `SOC_ETM_CHn_EVT_ID` 的配置值与所选择的事件的关系如表 8.3-1 所示。

表 8.3-1. 事件任务矩阵通道n可选事件

SOC_ETM_CHn_EVT_ID	所选事件	产生该事件的外设
1	GPIO_EVT_CHO_RISE_EDGE	GPIO
2	GPIO_EVT_CH1_RISE_EDGE	
3	GPIO_EVT_CH2_RISE_EDGE	
4	GPIO_EVT_CH3_RISE_EDGE	
5	GPIO_EVT_CH4_RISE_EDGE	
6	GPIO_EVT_CH5_RISE_EDGE	
7	GPIO_EVT_CH6_RISE_EDGE	
8	GPIO_EVT_CH7_RISE_EDGE	
9	GPIO_EVT_CHO_FALL_EDGE	
10	GPIO_EVT_CH1_FALL_EDGE	
11	GPIO_EVT_CH2_FALL_EDGE	
12	GPIO_EVT_CH3_FALL_EDGE	

SOC_ETM_CH n _EVT_ID	所选事件	产生该事件的外设
13	GPIO_EVT_CH4_FALL_EDGE	
14	GPIO_EVT_CH5_FALL_EDGE	
15	GPIO_EVT_CH6_FALL_EDGE	
16	GPIO_EVT_CH7_FALL_EDGE	
17	GPIO_EVT_CH0_ANY_EDGE	
18	GPIO_EVT_CH1_ANY_EDGE	
19	GPIO_EVT_CH2_ANY_EDGE	
20	GPIO_EVT_CH3_ANY_EDGE	
21	GPIO_EVT_CH4_ANY_EDGE	
22	GPIO_EVT_CH5_ANY_EDGE	
23	GPIO_EVT_CH6_ANY_EDGE	
24	GPIO_EVT_CH7_ANY_EDGE	
25	GPIO_EVT_ZERO_DET_POSO	
26	GPIO_EVT_ZERO_DET_NEGO	
27	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM 控制器 (LEDC)
28	LEDC_EVT_DUTY_CHNG_END_CH1	
29	LEDC_EVT_DUTY_CHNG_END_CH2	
30	LEDC_EVT_DUTY_CHNG_END_CH3	
31	LEDC_EVT_DUTY_CHNG_END_CH4	
32	LEDC_EVT_DUTY_CHNG_END_CH5	
33	LEDC_EVT_OVF_CNT_PLS_CH0	
34	LEDC_EVT_OVF_CNT_PLS_CH1	
35	LEDC_EVT_OVF_CNT_PLS_CH2	
36	LEDC_EVT_OVF_CNT_PLS_CH3	
37	LEDC_EVT_OVF_CNT_PLS_CH4	
38	LEDC_EVT_OVF_CNT_PLS_CH5	
39	LEDC_EVT_TIME_OVF_TIMER0	
40	LEDC_EVT_TIME_OVF_TIMER1	
41	LEDC_EVT_TIME_OVF_TIMER2	
42	LEDC_EVT_TIME_OVF_TIMER3	
43	LEDC_EVT_TIMER0_CMP	
44	LEDC_EVT_TIMER1_CMP	
45	LEDC_EVT_TIMER2_CMP	
46	LEDC_EVT_TIMER3_CMP	
47	TGO_EVT_CNT_CMP_TIMER0	通用定时器组 0
49	TG1_EVT_CNT_CMP_TIMER0	通用定时器组 1
51	SYSTIMER_EVT_CNT_CMPO	系统定时器
52	SYSTIMER_EVT_CNT_CMP1	
53	SYSTIMER_EVT_CNT_CMP2	
54	MCPWMO_EVT_TIMER0_STOP	MCPWMO
55	MCPWMO_EVT_TIMER1_STOP	
56	MCPWMO_EVT_TIMER2_STOP	
57	MCPWMO_EVT_TIMER0_TEZ	
58	MCPWMO_EVT_TIMER1_TEZ	
59	MCPWMO_EVT_TIMER2_TEZ	

SOC_ETM_CH n _EVT_ID	所选事件	产生该事件的外设
60	MCPWM0_EVT_TIMER0_TEP	
61	MCPWM0_EVT_TIMER1_TEP	
62	MCPWM0_EVT_TIMER2_TEP	
63	MCPWM0_EVT_OPO_TEA	
64	MCPWM0_EVT_OP1_TEA	
65	MCPWM0_EVT_OP2_TEA	
66	MCPWM0_EVT_OPO_TEB	
67	MCPWM0_EVT_OP1_TEB	
68	MCPWM0_EVT_OP2_TEB	
69	MCPWM0_EVT_F0	
70	MCPWM0_EVT_F1	
71	MCPWM0_EVT_F2	
72	MCPWM0_EVT_F0_CLR	
73	MCPWM0_EVT_F1_CLR	
74	MCPWM0_EVT_F2_CLR	
75	MCPWM0_EVT_TZO_CBC	
76	MCPWM0_EVT_TZ1_CBC	
77	MCPWM0_EVT_TZ2_CBC	
78	MCPWM0_EVT_TZO_OST	
79	MCPWM0_EVT_TZ1_OST	
80	MCPWM0_EVT_TZ2_OST	
81	MCPWM0_EVT_CAP0	
82	MCPWM0_EVT_CAP1	
83	MCPWM0_EVT_CAP2	
84	MCPWM0_EVT_OPO_TEE1	
85	MCPWM0_EVT_OP1_TEE1	
86	MCPWM0_EVT_OP2_TEE1	
87	MCPWM0_EVT_OPO_TEE2	
88	MCPWM0_EVT_OP1_TEE2	
89	MCPWM0_EVT_OP2_TEE2	
90	ADC_EVT_CONV_CMPLT0	ADC 控制器
91	ADC_EVT_EQ_ABOVE_THRESH0	
92	ADC_EVT_EQ_ABOVE_THRESH1	
93	ADC_EVT_EQ_BELOW_THRESH0	
94	ADC_EVT_EQ_BELOW_THRESH1	
96	ADC_EVT_STOPPED0	
97	ADC_EVT_STARTED0	
106	TMPSNSR_EVT_OVER_LIMIT	温度传感器
107	I2S0_EVT_RX_DONE	I2S
108	I2S0_EVT_TX_DONE	
109	I2S0_EVT_X_WORDS_RECEIVED	
110	I2S0_EVT_X_WORDS_SENT	
111	ULP_EVT_ERR_INTR	低功耗处理器
113	ULP_EVT_START_INTR	
114	RTC_EVT_TICK	实时时钟定时器

SOC_ETM_CHn_EVT_ID	所选事件	产生该事件的外设
115	RTC_EVT_OVF	通用 DMA 控制器 (GDMA)
116	RTC_EVT_CMP	
117	GDMA_EVT_IN_DONE_CH0	
118	GDMA_EVT_IN_DONE_CH1	
119	GDMA_EVT_IN_DONE_CH2	
120	GDMA_EVT_IN_SUC_EOF_CH0	
121	GDMA_EVT_IN_SUC_EOF_CH1	
122	GDMA_EVT_IN_SUC_EOF_CH2	
123	GDMA_EVT_IN_FIFO_EMPTY_CH0	
124	GDMA_EVT_IN_FIFO_EMPTY_CH1	
125	GDMA_EVT_IN_FIFO_EMPTY_CH2	
126	GDMA_EVT_IN_FIFO_FULL_CH0	
127	GDMA_EVT_IN_FIFO_FULL_CH1	
128	GDMA_EVT_IN_FIFO_FULL_CH2	
129	GDMA_EVT_OUT_DONE_CH0	
130	GDMA_EVT_OUT_DONE_CH1	
131	GDMA_EVT_OUT_DONE_CH2	
132	GDMA_EVT_OUT_EOF_CH0	
133	GDMA_EVT_OUT_EOF_CH1	
134	GDMA_EVT_OUT_EOF_CH2	
135	GDMA_EVT_OUT_TOTAL_EOF_CH0	
136	GDMA_EVT_OUT_TOTAL_EOF_CH1	
137	GDMA_EVT_OUT_TOTAL_EOF_CH2	
138	GDMA_EVT_OUT_FIFO_EMPTY_CH0	
139	GDMA_EVT_OUT_FIFO_EMPTY_CH1	
140	GDMA_EVT_OUT_FIFO_EMPTY_CH2	
141	GDMA_EVT_OUT_FIFO_FULL_CH0	
142	GDMA_EVT_OUT_FIFO_FULL_CH1	
143	GDMA_EVT_OUT_FIFO_FULL_CH2	
144	PMU_EVT_SLEEP_WEEKUP	PMU

以上任意事件产生时，对应外设会生成一个脉冲信号。脉冲信号为高电平时，视为接收到相应的事件。

对于事件更详细的描述，请参考产生该事件外设的章节。

8.3.3 任务

每个事件任务矩阵通道可以通过字段 SOC_ETM_CHn_TASK_ID 选择要映射到的任务。字段 SOC_ETM_CHn_TASK_ID 的配置值与要映射到的任务的关系如表 8.3-2 所示。

表 8.3-2. 事件任务矩阵通道n可映射任务

SOC_ETM_CHn_TASK_ID	所映射任务	接收该任务的外设
1	GPIO_TASK_CH0_SET	GPIO
2	GPIO_TASK_CH1_SET	
3	GPIO_TASK_CH2_SET	

SOC_ETM_CH n _TASK_ID	所映射任务	接收该任务的外设
4	GPIO_TASK_CH3_SET	
5	GPIO_TASK_CH4_SET	
6	GPIO_TASK_CH5_SET	
7	GPIO_TASK_CH6_SET	
8	GPIO_TASK_CH7_SET	
9	GPIO_TASK_CH0_CLEAR	
10	GPIO_TASK_CH1_CLEAR	
11	GPIO_TASK_CH2_CLEAR	
12	GPIO_TASK_CH3_CLEAR	
13	GPIO_TASK_CH4_CLEAR	
14	GPIO_TASK_CH5_CLEAR	
15	GPIO_TASK_CH6_CLEAR	
16	GPIO_TASK_CH7_CLEAR	
17	GPIO_TASK_CH0_TOGGLE	
18	GPIO_TASK_CH1_TOGGLE	
19	GPIO_TASK_CH2_TOGGLE	
20	GPIO_TASK_CH3_TOGGLE	
21	GPIO_TASK_CH4_TOGGLE	
22	GPIO_TASK_CH5_TOGGLE	
23	GPIO_TASK_CH6_TOGGLE	
24	GPIO_TASK_CH7_TOGGLE	
25	LEDC_TASK_TIMER0_RES_UPDATE	LED PWM 控制器 (LEDC)
26	LEDC_TASK_TIMER1_RES_UPDATE	
27	LEDC_TASK_TIMER2_RES_UPDATE	
28	LEDC_TASK_TIMER3_RES_UPDATE	
29	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
30	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
35	LEDC_TASK_TIMER0_CAP	
36	LEDC_TASK_TIMER1_CAP	
37	LEDC_TASK_TIMER2_CAP	
38	LEDC_TASK_TIMER3_CAP	
39	LEDC_TASK_SIG_OUT_DIS_CH0	
40	LEDC_TASK_SIG_OUT_DIS_CH1	
41	LEDC_TASK_SIG_OUT_DIS_CH2	
42	LEDC_TASK_SIG_OUT_DIS_CH3	
43	LEDC_TASK_SIG_OUT_DIS_CH4	
44	LEDC_TASK_SIG_OUT_DIS_CH5	
45	LEDC_TASK_OVF_CNT_RST_CH0	
46	LEDC_TASK_OVF_CNT_RST_CH1	
47	LEDC_TASK_OVF_CNT_RST_CH2	
48	LEDC_TASK_OVF_CNT_RST_CH3	

SOC_ETM_CH n _TASK_ID	所映射任务	接收该任务的外设
49	LEDC_TASK_OVF_CNT_RST_CH4	
50	LEDC_TASK_OVF_CNT_RST_CH5	
51	LEDC_TASK_TIMER0_RST	
52	LEDC_TASK_TIMER1_RST	
53	LEDC_TASK_TIMER2_RST	
54	LEDC_TASK_TIMER3_RST	
55	LEDC_TASK_TIMER0_RESUME	
56	LEDC_TASK_TIMER1_RESUME	
57	LEDC_TASK_TIMER2_RESUME	
58	LEDC_TASK_TIMER3_RESUME	
59	LEDC_TASK_TIMER0_PAUSE	
60	LEDC_TASK_TIMER1_PAUSE	
61	LEDC_TASK_TIMER2_PAUSE	
62	LEDC_TASK_TIMER3_PAUSE	
63	LEDC_TASK_GAMMA_RESTART_CH0	
64	LEDC_TASK_GAMMA_RESTART_CH1	
65	LEDC_TASK_GAMMA_RESTART_CH2	
66	LEDC_TASK_GAMMA_RESTART_CH3	
67	LEDC_TASK_GAMMA_RESTART_CH4	
68	LEDC_TASK_GAMMA_RESTART_CH5	
69	LEDC_TASK_GAMMA_PAUSE_CH0	
70	LEDC_TASK_GAMMA_PAUSE_CH1	
71	LEDC_TASK_GAMMA_PAUSE_CH2	
72	LEDC_TASK_GAMMA_PAUSE_CH3	
73	LEDC_TASK_GAMMA_PAUSE_CH4	
74	LEDC_TASK_GAMMA_PAUSE_CH5	
75	LEDC_TASK_GAMMA_RESUME_CH0	
76	LEDC_TASK_GAMMA_RESUME_CH1	
77	LEDC_TASK_GAMMA_RESUME_CH2	
78	LEDC_TASK_GAMMA_RESUME_CH3	
79	LEDC_TASK_GAMMA_RESUME_CH4	
80	LEDC_TASK_GAMMA_RESUME_CH5	
81	TGO_TASK_CNT_START_TIMER0	通用定时器组 0
82	TGO_TASK_ALARM_START_TIMER0	
83	TGO_TASK_CNT_STOP_TIMER0	
84	TGO_TASK_CNT_RELOAD_TIMER0	
85	TGO_TASK_CNT_CAP_TIMER0	
91	TG1_TASK_CNT_START_TIMER0	通用定时器组 1
92	TG1_TASK_ALARM_START_TIMER0	
93	TG1_TASK_CNT_STOP_TIMER0	
94	TG1_TASK_CNT_RELOAD_TIMER0	
95	TG1_TASK_CNT_CAP_TIMER0	
101	MCPWM0_TASK_CMPRO_A_UP	MCPWM0
102	MCPWM0_TASK_CMPR1_A_UP	
103	MCPWM0_TASK_CMPR2_A_UP	

SOC_ETM_CH n _TASK_ID	所映射任务	接收该任务的外设
104	MCPWMO_TASK_CMPRO_B_UP	
105	MCPWMO_TASK_CMPR1_B_UP	
106	MCPWMO_TASK_CMPR2_B_UP	
107	MCPWMO_TASK_GEN_STOP	
108	MCPWMO_TASK_TIMER0_SYN	
109	MCPWMO_TASK_TIMER1_SYN	
110	MCPWMO_TASK_TIMER2_SYN	
111	MCPWMO_TASK_TIMER0_PERIOD_UP	
112	MCPWMO_TASK_TIMER1_PERIOD_UP	
113	MCPWMO_TASK_TIMER2_PERIOD_UP	
114	MCPWMO_TASK_TZO_OST	
115	MCPWMO_TASK_TZ1_OST	
116	MCPWMO_TASK_TZ2_OST	
117	MCPWMO_TASK_CLRO_OST	
118	MCPWMO_TASK_CLR1_OST	
119	MCPWMO_TASK_CLR2_OST	
120	MCPWMO_TASK_CAPO	
121	MCPWMO_TASK_CAP1	
122	MCPWMO_TASK_CAP2	
123	ADC_TASK_SAMPLEO	ADC 控制器
125	ADC_TASK_STARTO	
126	ADC_TASK_STOPO	
131	TMPSNSR_TASK_START_SAMPLE	温度传感器
132	TMPSNSR_TASK_STOP_SAMPLE	
133	I2SO_TASK_START_RX	I2S
134	I2SO_TASK_START_TX	
135	I2SO_TASK_STOP_RX	
136	I2SO_TASK_STOP_TX	
137	ULP_TASK_WEAKUP_CPU	低功耗处理器
143	GDMA_TASK_IN_START_CHO	通用 DMA 控制器 (GDMA)
144	GDMA_TASK_IN_START_CH1	
145	GDMA_TASK_IN_START_CH2	
146	GDMA_TASK_OUT_START_CHO	
147	GDMA_TASK_OUT_START_CH1	
148	GDMA_TASK_OUT_START_CH2	
149	PMU_TASK_SLEEP_REQ	PMU

一个通道接收到一个有效事件脉冲信号时，会产生映射任务脉冲信号。

对于任务更详细的描述，请参考接收该任务外设的章节。

来自多个不同通道的事件可以选择映射到同一个任务。例如，可以将多个通道的字段 SOC_ETM_CH n _TASK_ID 都配置为相同的值，而字段 SOC_ETM_CH n _EVT_ID 可以配置为相同或不同的值。在这种情况下，当任意一个通道接收到的事件有效时，指定的任务就会被生成。当多个通道接收到的事件同时有效时，指定的任务只会产生一次。

8.3.4 事件任务触发状态

ETM 外设支持通过读取寄存器来查看事件和任务的触发状态。

- 事件触发状态：读取寄存器 `SOC_ETM_EVT_STn_REG` 中所要查看事件对应的字段，如果该字段的值为 1，表示对应的事件被 ETM 接收过，如果该字段的值为 0，表示对应的事件未被 ETM 接收到。向寄存器 `SOC_ETM_EVT_STn_CLR_REG` 中所要查看事件对应的字段写 1，可以将寄存器 `SOC_ETM_EVT_STn_REG` 中所要查看事件对应的字段清 0。
- 任务触发状态：读取寄存器 `SOC_ETM_TASK_STn_REG` 中所要查看任务对应的字段，如果该字段的值为 1，表示 ETM 产生过对应的任务，如果该字段的值为 0，表示 ETM 未产生过对应的任务。向寄存器 `SOC_ETM_TASK_STn_CLR_REG` 中所要查看任务对应的字段写 1，可以将寄存器 `SOC_ETM_TASK_STn_REG` 中所要查看任务对应的字段清 0。

8.3.5 时序考虑因素

ETM 外设接收的事件、发送的任务和 ETM 通道之间驱动时钟的结构如图 8.3-2 所示。

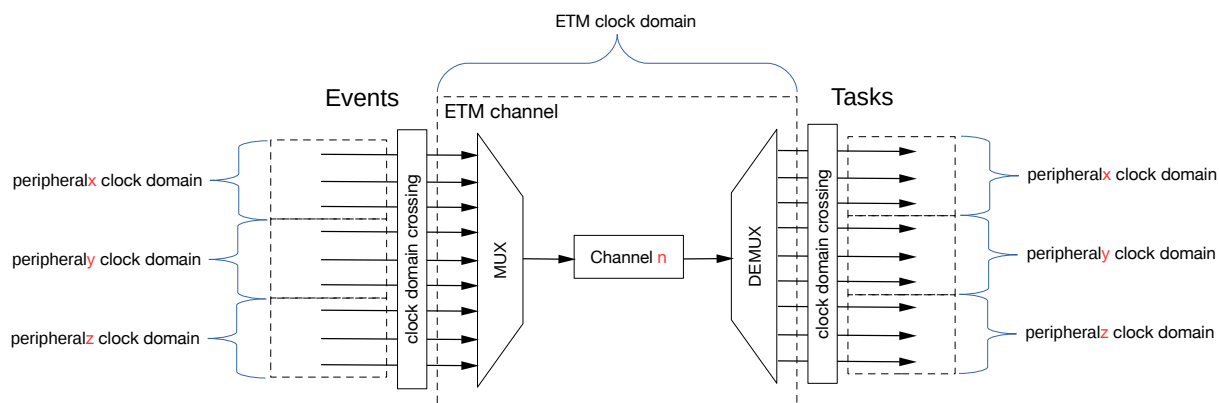


图 8.3-2. 事件任务矩阵时钟结构

ETM 运行在 `SYS_CLK` 时钟域下（详见章节 6 复位和时钟），每个事件对应一个由相应外设在其时钟域中产生的脉冲信号，每个任务被 ETM 映射为对应外设时钟域下的脉冲信号。由于产生事件的外设、事件任务矩阵和接收任务的外设不一定由同个时钟驱动，需要同步处理，两个连续时间之间需要根据生成事件外设的时钟、ETM 的时钟、执行任务外设的时钟设置延迟，否则会丢失事件。

为了让事件任务矩阵成功接收到每个事件，对于每个可以产生事件的外设，其产生的两个连续事件脉冲之间的时间间隔必须大于一个事件任务矩阵的时钟周期，即 $\text{ceil}(\frac{\text{peripheral\_clock\_frequency}}{\text{ETM\_clock\_frequency}})$ 个外设时钟周期。

例如，假设外设 A 所在时钟域（`PLL_F80M_CLK`）为 80 MHz，ETM 所在时钟域（`SYS_CLK`）为 40 MHz，外设 A 生成事件 1，ETM 接收事件 1。要成功接收每个事件 1，两个连续事件 1 之间的时间间隔必须大于外设 A 的两个时钟周期（即一个 ETM 时钟周期）。

同样的，为了使事件任务矩阵成功地将接收到的事件（即同步到事件任务矩阵时钟域的事件）映射为外设的任务，事件任务矩阵时钟域中的两个连续事件脉冲之间的时间间隔必须大于一个外设的时钟周期，即 $\text{ceil}(\frac{\text{ETM\_clock\_frequency}}{\text{peripheral\_clock\_frequency}})$ 个事件任务矩阵时钟周期。

例如，假设外设 B 所在时钟域（`RC_FAST_CLK`）为 20 MHz，ETM 所在时钟域（`SYS_CLK`）为 40 MHz，ETM 将事件映射到任务 1，外设 B 接收任务 1。要成功将每个接收到的事件映射到任务 1，两个连续事件之间的时间

间隔必须大于两个 ETM 时钟周期（即一个外设 B 时钟周期）。

因此，要将外设 A 产生的两个连续事件完整映射到外设 B 的两个任务中，这两个事件之间的时间间隔必须大于 $\text{ceil}(\frac{\text{peripheral\_A\_clock\_frequency}}{\text{ETM\_clock\_frequency}}) * \text{ceil}(\frac{\text{ETM\_clock\_frequency}}{\text{peripheral\_B\_clock\_frequency}})$ 个外设 A 时钟周期。

例如，假设外设 A 所在时钟域（PLL_F80M_CLK）为 80 MHz，外设 B 所在时钟域（RC_FAST_CLK）为 20 MHz，ETM 所在时钟域（SYS_CLK）为 40 MHz，外设 A 生成事件 1，ETM 将事件映射到任务 1，外设 B 接收任务 1。要成功将每个事件 1（外设 A 生成）映射到任务 1（外设 B 接收），两个连续事件 1 之间的时间间隔必须大于 $2 * 2 = 4$ 个外设 A 时钟周期。

8.3.6 通道控制

事件任务矩阵的每个通道都可以独立配置为使能或关闭。当通道 n 配置为使能时，如果该通道接收到所配置的事件（字段 `SOC_ETM_CH $n$ _EVT_ID`），该事件将映射到所配置的任务上（字段 `SOC_ETM_CH $n$ _TASK_ID`）。当通道 n 配置为不使能时，即使该通道接收到所配置的事件（字段 `SOC_ETM_CH $n$ _EVT_ID`），也不会产生任何任务。

使能事件任务矩阵通道 n ，需向字段 `SOC_ETM_CH_ENABLE $n$` 写 1。关闭事件任务矩阵通道 n ，需向字段 `SOC_ETM_CH_DISABLE $n$` 写 1。

通道 n 的状态可通过读取字段 `SOC_ETM_CH_ENABLED $n$` 的值获取。1 表示通道 n 已使能，0 表示通道 n 未使能。

如果将字段 `SOC_ETM_CH $n$ _EVT_ID` 的值配置为 0 或将字段 `SOC_ETM_CH $n$ _TASK_ID` 的值配置为 0，事件任务矩阵通道 n 也将被关闭。

事件任务矩阵通道 n 的完整配置过程如下：

1. 向字段 `PCR_ETM_CLK_EN` 写 1，开启 ETM 的时钟
2. 配置字段 `SOC_ETM_CH $n$ _EVT_ID` 的值，选择通道 n 所要接收的事件
3. 配置字段 `SOC_ETM_CH $n$ _TASK_ID` 的值，选择通道 n 所接收到的事件要映射到的任务
4. 向字段 `SOC_ETM_CH_ENABLE $n$` 写 1 使能通道
5. 当通道 n 不再需要将配置的事件映射到配置的任务时，向字段 `SOC_ETM_CH_DISABLE $n$` 写 1 来关闭通道 n 。之后，如果要为通道 n 配置新的事件和任务的映射关系，则重复步骤 2-4，如果不再配置，则通道 n 保持停止运行状态
6. 若需复位整个 ETM，即复位所有通道的配置，可通过向 `PCR_ETM_RST_EN` 字段先写 1 再写 0 来实现

8.4 寄存器列表

本小节的所有地址均为相对于事件任务矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
状态寄存器			
SOC_ETM_CH_ENA_ADO_REG	通道使能状态寄存器	0x0000	R/WTC/WTS
SOC_ETM_CH_ENA_AD1_REG	通道使能状态寄存器	0x000C	R/WTC/WTS
SOC_ETM_EVT_ST0_REG	事件触发状态寄存器	0x01A8	R/WTC/SS
SOC_ETM_EVT_ST1_REG	事件触发状态寄存器	0x01B0	R/WTC/SS
SOC_ETM_EVT_ST2_REG	事件触发状态寄存器	0x01B8	R/WTC/SS
SOC_ETM_EVT_ST3_REG	事件触发状态寄存器	0x01C0	R/WTC/SS
SOC_ETM_EVT_ST4_REG	事件触发状态寄存器	0x01C8	R/WTC/SS
SOC_ETM_TASK_STO_REG	任务触发状态寄存器	0x01D0	R/WTC/SS
SOC_ETM_TASK_ST1_REG	任务触发状态寄存器	0x01D8	R/WTC/SS
SOC_ETM_TASK_ST2_REG	任务触发状态寄存器	0x01E0	R/WTC/SS
SOC_ETM_TASK_ST3_REG	任务触发状态寄存器	0x01E8	R/WTC/SS
SOC_ETM_TASK_ST4_REG	任务触发状态寄存器	0x01F0	R/WTC/SS
配置寄存器			
SOC_ETM_CH_ENA_ADO_SET_REG	通道使能寄存器	0x0004	WT
SOC_ETM_CH_ENA_ADO_CLR_REG	通道关闭寄存器	0x0008	WT
SOC_ETM_CH_ENA_AD1_SET_REG	通道使能寄存器	0x0010	WT
SOC_ETM_CH_ENA_AD1_CLR_REG	通道关闭寄存器	0x0014	WT
SOC_ETM_CH0_EVT_ID_REG	通道 0 事件 ID 寄存器	0x0018	R/W
SOC_ETM_CH0_TASK_ID_REG	通道 0 任务 ID 寄存器	0x001C	R/W
SOC_ETM_CH1_EVT_ID_REG	通道 1 事件 ID 寄存器	0x0020	R/W
SOC_ETM_CH1_TASK_ID_REG	通道 1 任务 ID 寄存器	0x0024	R/W
SOC_ETM_CH2_EVT_ID_REG	通道 2 事件 ID 寄存器	0x0028	R/W
SOC_ETM_CH2_TASK_ID_REG	通道 2 任务 ID 寄存器	0x002C	R/W
SOC_ETM_CH3_EVT_ID_REG	通道 3 事件 ID 寄存器	0x0030	R/W
SOC_ETM_CH3_TASK_ID_REG	通道 3 任务 ID 寄存器	0x0034	R/W
SOC_ETM_CH4_EVT_ID_REG	通道 4 事件 ID 寄存器	0x0038	R/W
SOC_ETM_CH4_TASK_ID_REG	通道 4 任务 ID 寄存器	0x003C	R/W
SOC_ETM_CH5_EVT_ID_REG	通道 5 事件 ID 寄存器	0x0040	R/W
SOC_ETM_CH5_TASK_ID_REG	通道 5 任务 ID 寄存器	0x0044	R/W
SOC_ETM_CH6_EVT_ID_REG	通道 6 事件 ID 寄存器	0x0048	R/W
SOC_ETM_CH6_TASK_ID_REG	通道 6 任务 ID 寄存器	0x004C	R/W
SOC_ETM_CH7_EVT_ID_REG	通道 7 事件 ID 寄存器	0x0050	R/W
SOC_ETM_CH7_TASK_ID_REG	通道 7 任务 ID 寄存器	0x0054	R/W
SOC_ETM_CH8_EVT_ID_REG	通道 8 事件 ID 寄存器	0x0058	R/W
SOC_ETM_CH8_TASK_ID_REG	通道 8 任务 ID 寄存器	0x005C	R/W
SOC_ETM_CH9_EVT_ID_REG	通道 9 事件 ID 寄存器	0x0060	R/W

名称	描述	地址	访问
SOC_ETM_CH9_TASK_ID_REG	通道 9 任务 ID 寄存器	0x0064	R/W
SOC_ETM_CH10_EVT_ID_REG	通道 10 事件 ID 寄存器	0x0068	R/W
SOC_ETM_CH10_TASK_ID_REG	通道 10 任务 ID 寄存器	0x006C	R/W
SOC_ETM_CH11_EVT_ID_REG	通道 11 事件 ID 寄存器	0x0070	R/W
SOC_ETM_CH11_TASK_ID_REG	通道 11 任务 ID 寄存器	0x0074	R/W
SOC_ETM_CH12_EVT_ID_REG	通道 12 事件 ID 寄存器	0x0078	R/W
SOC_ETM_CH12_TASK_ID_REG	通道 12 任务 ID 寄存器	0x007C	R/W
SOC_ETM_CH13_EVT_ID_REG	通道 13 事件 ID 寄存器	0x0080	R/W
SOC_ETM_CH13_TASK_ID_REG	通道 13 任务 ID 寄存器	0x0084	R/W
SOC_ETM_CH14_EVT_ID_REG	通道 14 事件 ID 寄存器	0x0088	R/W
SOC_ETM_CH14_TASK_ID_REG	通道 14 任务 ID 寄存器	0x008C	R/W
SOC_ETM_CH15_EVT_ID_REG	通道 15 事件 ID 寄存器	0x0090	R/W
SOC_ETM_CH15_TASK_ID_REG	通道 15 任务 ID 寄存器	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	通道 16 事件 ID 寄存器	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	通道 16 任务 ID 寄存器	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	通道 17 事件 ID 寄存器	0x00A0	R/W
SOC_ETM_CH17_TASK_ID_REG	通道 17 任务 ID 寄存器	0x00A4	R/W
SOC_ETM_CH18_EVT_ID_REG	通道 18 事件 ID 寄存器	0x00A8	R/W
SOC_ETM_CH18_TASK_ID_REG	通道 18 任务 ID 寄存器	0x00AC	R/W
SOC_ETM_CH19_EVT_ID_REG	通道 19 事件 ID 寄存器	0x00B0	R/W
SOC_ETM_CH19_TASK_ID_REG	通道 19 任务 ID 寄存器	0x00B4	R/W
SOC_ETM_CH20_EVT_ID_REG	通道 20 事件 ID 寄存器	0x00B8	R/W
SOC_ETM_CH20_TASK_ID_REG	通道 20 任务 ID 寄存器	0x00BC	R/W
SOC_ETM_CH21_EVT_ID_REG	通道 21 事件 ID 寄存器	0x00C0	R/W
SOC_ETM_CH21_TASK_ID_REG	通道 21 任务 ID 寄存器	0x00C4	R/W
SOC_ETM_CH22_EVT_ID_REG	通道 22 事件 ID 寄存器	0x00C8	R/W
SOC_ETM_CH22_TASK_ID_REG	通道 22 任务 ID 寄存器	0x00CC	R/W
SOC_ETM_CH23_EVT_ID_REG	通道 23 事件 ID 寄存器	0x00D0	R/W
SOC_ETM_CH23_TASK_ID_REG	通道 23 任务 ID 寄存器	0x00D4	R/W
SOC_ETM_CH24_EVT_ID_REG	通道 24 事件 ID 寄存器	0x00D8	R/W
SOC_ETM_CH24_TASK_ID_REG	通道 24 任务 ID 寄存器	0x00DC	R/W
SOC_ETM_CH25_EVT_ID_REG	通道 25 事件 ID 寄存器	0x00E0	R/W
SOC_ETM_CH25_TASK_ID_REG	通道 25 任务 ID 寄存器	0x00E4	R/W
SOC_ETM_CH26_EVT_ID_REG	通道 26 事件 ID 寄存器	0x00E8	R/W
SOC_ETM_CH26_TASK_ID_REG	通道 26 任务 ID 寄存器	0x00EC	R/W
SOC_ETM_CH27_EVT_ID_REG	通道 27 事件 ID 寄存器	0x00F0	R/W
SOC_ETM_CH27_TASK_ID_REG	通道 27 任务 ID 寄存器	0x00F4	R/W
SOC_ETM_CH28_EVT_ID_REG	通道 28 事件 ID 寄存器	0x00F8	R/W
SOC_ETM_CH28_TASK_ID_REG	通道 28 任务 ID 寄存器	0x00FC	R/W
SOC_ETM_CH29_EVT_ID_REG	通道 29 事件 ID 寄存器	0x0100	R/W
SOC_ETM_CH29_TASK_ID_REG	通道 29 任务 ID 寄存器	0x0104	R/W
SOC_ETM_CH30_EVT_ID_REG	通道 30 事件 ID 寄存器	0x0108	R/W
SOC_ETM_CH30_TASK_ID_REG	通道 30 任务 ID 寄存器	0x010C	R/W

名称	描述	地址	访问
SOC_ETM_CH31_EVT_ID_REG	通道 31 事件 ID 寄存器	0x0110	R/W
SOC_ETM_CH31_TASK_ID_REG	通道 31 任务 ID 寄存器	0x0114	R/W
SOC_ETM_CH32_EVT_ID_REG	通道 32 事件 ID 寄存器	0x0118	R/W
SOC_ETM_CH32_TASK_ID_REG	通道 32 任务 ID 寄存器	0x011C	R/W
SOC_ETM_CH33_EVT_ID_REG	通道 33 事件 ID 寄存器	0x0120	R/W
SOC_ETM_CH33_TASK_ID_REG	通道 33 任务 ID 寄存器	0x0124	R/W
SOC_ETM_CH34_EVT_ID_REG	通道 34 事件 ID 寄存器	0x0128	R/W
SOC_ETM_CH34_TASK_ID_REG	通道 34 任务 ID 寄存器	0x012C	R/W
SOC_ETM_CH35_EVT_ID_REG	通道 35 事件 ID 寄存器	0x0130	R/W
SOC_ETM_CH35_TASK_ID_REG	通道 35 任务 ID 寄存器	0x0134	R/W
SOC_ETM_CH36_EVT_ID_REG	通道 36 事件 ID 寄存器	0x0138	R/W
SOC_ETM_CH36_TASK_ID_REG	通道 36 任务 ID 寄存器	0x013C	R/W
SOC_ETM_CH37_EVT_ID_REG	通道 37 事件 ID 寄存器	0x0140	R/W
SOC_ETM_CH37_TASK_ID_REG	通道 37 任务 ID 寄存器	0x0144	R/W
SOC_ETM_CH38_EVT_ID_REG	通道 38 事件 ID 寄存器	0x0148	R/W
SOC_ETM_CH38_TASK_ID_REG	通道 38 任务 ID 寄存器	0x014C	R/W
SOC_ETM_CH39_EVT_ID_REG	通道 39 事件 ID 寄存器	0x0150	R/W
SOC_ETM_CH39_TASK_ID_REG	通道 39 任务 ID 寄存器	0x0154	R/W
SOC_ETM_CH40_EVT_ID_REG	通道 40 事件 ID 寄存器	0x0158	R/W
SOC_ETM_CH40_TASK_ID_REG	通道 40 任务 ID 寄存器	0x015C	R/W
SOC_ETM_CH41_EVT_ID_REG	通道 41 事件 ID 寄存器	0x0160	R/W
SOC_ETM_CH41_TASK_ID_REG	通道 41 任务 ID 寄存器	0x0164	R/W
SOC_ETM_CH42_EVT_ID_REG	通道 42 事件 ID 寄存器	0x0168	R/W
SOC_ETM_CH42_TASK_ID_REG	通道 42 任务 ID 寄存器	0x016C	R/W
SOC_ETM_CH43_EVT_ID_REG	通道 43 事件 ID 寄存器	0x0170	R/W
SOC_ETM_CH43_TASK_ID_REG	通道 43 任务 ID 寄存器	0x0174	R/W
SOC_ETM_CH44_EVT_ID_REG	通道 44 事件 ID 寄存器	0x0178	R/W
SOC_ETM_CH44_TASK_ID_REG	通道 44 任务 ID 寄存器	0x017C	R/W
SOC_ETM_CH45_EVT_ID_REG	通道 45 事件 ID 寄存器	0x0180	R/W
SOC_ETM_CH45_TASK_ID_REG	通道 45 任务 ID 寄存器	0x0184	R/W
SOC_ETM_CH46_EVT_ID_REG	通道 46 事件 ID 寄存器	0x0188	R/W
SOC_ETM_CH46_TASK_ID_REG	通道 46 任务 ID 寄存器	0x018C	R/W
SOC_ETM_CH47_EVT_ID_REG	通道 47 事件 ID 寄存器	0x0190	R/W
SOC_ETM_CH47_TASK_ID_REG	通道 47 任务 ID 寄存器	0x0194	R/W
SOC_ETM_CH48_EVT_ID_REG	通道 48 事件 ID 寄存器	0x0198	R/W
SOC_ETM_CH48_TASK_ID_REG	通道 48 任务 ID 寄存器	0x019C	R/W
SOC_ETM_CH49_EVT_ID_REG	通道 49 事件 ID 寄存器	0x01A0	R/W
SOC_ETM_CH49_TASK_ID_REG	通道 49 任务 ID 寄存器	0x01A4	R/W
SOC_ETM_EVT_ST0_CLR_REG	事件触发状态清除寄存器	0x01AC	WT
SOC_ETM_EVT_ST1_CLR_REG	事件触发状态清除寄存器	0x01B4	WT
SOC_ETM_EVT_ST2_CLR_REG	事件触发状态清除寄存器	0x01BC	WT
SOC_ETM_EVT_ST3_CLR_REG	事件触发状态清除寄存器	0x01C4	WT
SOC_ETM_EVT_ST4_CLR_REG	事件触发状态清除寄存器	0x01CC	WT

名称	描述	地址	访问
SOC_ETM_TASK_ST0_CLR_REG	任务触发状态清除寄存器	0x01D4	WT
SOC_ETM_TASK_ST1_CLR_REG	任务触发状态清除寄存器	0x01DC	WT
SOC_ETM_TASK_ST2_CLR_REG	任务触发状态清除寄存器	0x01E4	WT
SOC_ETM_TASK_ST3_CLR_REG	任务触发状态清除寄存器	0x01EC	WT
SOC_ETM_TASK_ST4_CLR_REG	任务触发状态清除寄存器	0x01F4	WT
SOC_ETM_CLK_EN_REG	ETM 时钟使能寄存器	0x01F8	R/W
版本寄存器			
SOC_ETM_DATE_REG	版本控制寄存器	0x01FC	R/W

8.5 寄存器

本小节的所有地址均为相对于事件任务矩阵基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 8.1. SOC_ETM_CH_ENA_ADO_REG (0x0000)

SOC_ETM_CH_ENABLED31	SOC_ETM_CH_ENABLED30	SOC_ETM_CH_ENABLED29	SOC_ETM_CH_ENABLED28	SOC_ETM_CH_ENABLED27	SOC_ETM_CH_ENABLED26	SOC_ETM_CH_ENABLED25	SOC_ETM_CH_ENABLED24	SOC_ETM_CH_ENABLED23	SOC_ETM_CH_ENABLED22	SOC_ETM_CH_ENABLED21	SOC_ETM_CH_ENABLED20	SOC_ETM_CH_ENABLED19	SOC_ETM_CH_ENABLED18	SOC_ETM_CH_ENABLED17	SOC_ETM_CH_ENABLED16	SOC_ETM_CH_ENABLED15	SOC_ETM_CH_ENABLED14	SOC_ETM_CH_ENABLED13	SOC_ETM_CH_ENABLED12	SOC_ETM_CH_ENABLED11	SOC_ETM_CH_ENABLED10	SOC_ETM_CH_ENABLED9	SOC_ETM_CH_ENABLED8	SOC_ETM_CH_ENABLED7	SOC_ETM_CH_ENABLED6	SOC_ETM_CH_ENABLED5	SOC_ETM_CH_ENABLED4	SOC_ETM_CH_ENABLED3	SOC_ETM_CH_ENABLED2	SOC_ETM_CH_ENABLED1	SOC_ETM_CH_ENABLED0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_CH_ENABLED(*n*: 0-31) 表示通道 *n* 的使能状态。

0: 关闭

1: 使能

(R/WTC/WTS)

Register 8.2. SOC_ETM_CH_ENA_AD1_REG (0x000C)

(reserved)																	SOC_ETM_CH_ENABLED49																	SOC_ETM_CH_ENABLED48																	SOC_ETM_CH_ENABLED47																	SOC_ETM_CH_ENABLED46																	SOC_ETM_CH_ENABLED45																	SOC_ETM_CH_ENABLED44																	SOC_ETM_CH_ENABLED43																	SOC_ETM_CH_ENABLED42																	SOC_ETM_CH_ENABLED41																	SOC_ETM_CH_ENABLED40																	SOC_ETM_CH_ENABLED39																	SOC_ETM_CH_ENABLED38																	SOC_ETM_CH_ENABLED37																	SOC_ETM_CH_ENABLED36																	SOC_ETM_CH_ENABLED35																	SOC_ETM_CH_ENABLED34																	SOC_ETM_CH_ENABLED33																	SOC_ETM_CH_ENABLED32																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
31																	18																	17																	16																	15																	14																	13																	12																	11																	10																	9																	8																	7																	6																	5																	4																	3																	2																	1																	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																	0																

Reset

SOC_ETM_CH_ENABLED(*n*: 32-49) 表示通道 *n* 的使能状态。

0: 关闭

1: 使能

(R/WTC/WTS)

Register 8.3. SOC_ETM_EVT_STO_REG (0x01A8)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST																															SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST																															SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST																															SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST																															SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST																															SOC_ETM_GPIO_EVT_ZERO_DET_NEGO_ST																															SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST																															SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST																															SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST																															SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST																															Reset																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															

SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST 表示 GPIO_EVT_CH0_RISE_EDGE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST 表示 GPIO_EVT_CH1_RISE_EDGE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST 表示 GPIO_EVT_CH2_RISE_EDGE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST 表示 GPIO_EVT_CH3_RISE_EDGE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST 表示 GPIO_EVT_CH4_RISE_EDGE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST 表示 GPIO_EVT_CH5_RISE_EDGE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.3. SOC_ETM_EVT_STO_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST 表示 GPIO_EVT_CH6_RISE_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST 表示 GPIO_EVT_CH7_RISE_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST 表示 GPIO_EVT_CH0_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST 表示 GPIO_EVT_CH1_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST 表示 GPIO_EVT_CH2_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST 表示 GPIO_EVT_CH3_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST 表示 GPIO_EVT_CH4_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST 表示 GPIO_EVT_CH5_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST 表示 GPIO_EVT_CH6_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.3. SOC_ETM_EVT_STO_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST 表示 GPIO_EVT_CH7_FALL_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST 表示 GPIO_EVT_CH0_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST 表示 GPIO_EVT_CH1_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST 表示 GPIO_EVT_CH2_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST 表示 GPIO_EVT_CH3_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST 表示 GPIO_EVT_CH4_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST 表示 GPIO_EVT_CH5_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST 表示 GPIO_EVT_CH6_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.3. SOC_ETM_EVT_STO_REG (0x01A8)

接上页...

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST 表示 GPIO_EVT_CH7_ANY_EDGE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST 表示 GPIO_EVT_ZERO_DET_POS0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST 表示 GPIO_EVT_ZERO_DET_NEG0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CHO_ST 表示 LEDC_EVT_DUTY_CHNG_END_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH3 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH4 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.3. SOC_ETM_EVT_STO_REG (0x01A8)

[接上页...](#)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST 表示 LEDC_EVT_DUTY_CHNG_END_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

Register 8.4. SOC_ETM_EVT_ST1_REG (0x01B0)

SOC_ETM_MCPWMO_EVT_OP1_TEA_ST	SOC_ETM_MCPWMO_EVT_OP0_TEA_ST	SOC_ETM_MCPWMO_EVT_TIMER2_TEP_ST	SOC_ETM_MCPWMO_EVT_TIMER1_TEP_ST	SOC_ETM_MCPWMO_EVT_TIMER0_TEP_ST	SOC_ETM_MCPWMO_EVT_TIMER2_TEZ_ST	SOC_ETM_MCPWMO_EVT_TIMER1_TEZ_ST	SOC_ETM_MCPWMO_EVT_TIMER0_TEZ_ST	SOC_ETM_MCPWMO_EVT_TIMER2_STOP_ST	SOC_ETM_MCPWMO_EVT_TIMER1_STOP_ST	SOC_ETM_MCPWMO_EVT_TIMER0_STOP_ST	SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST	SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST	SOC_ETM_TG1_EVT_CNT_CMP0_ST	SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST	SOC_ETM_TG0_EVT_CNT_CMP_TIMER0_ST	SOC_ETM_LEDC_EVT_TIMER3_ST	SOC_ETM_LEDC_EVT_TIMER2_ST	SOC_ETM_LEDC_EVT_TIMER1_ST	SOC_ETM_LEDC_EVT_TIMER0_ST	SOC_ETM_LEDC_EVT_TIMER2_CMP_ST	SOC_ETM_LEDC_EVT_TIMER1_CMP_ST	SOC_ETM_LEDC_EVT_TIMER0_CMP_ST	SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST	SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST	SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST	SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST	SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST	SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST	SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST	SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST	SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH0_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH3 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH4 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST 表示 LEDC_EVT_OVF_CNT_PLS_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

Register 8.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST 表示 LEDC_EVT_TIME_OVF_TIMER0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST 表示 LEDC_EVT_TIME_OVF_TIMER1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST 表示 LEDC_EVT_TIME_OVF_TIMER2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST 表示 LEDC_EVT_TIME_OVF_TIMER3 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST 表示 LEDC_EVT_TIMER0_CMP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST 表示 LEDC_EVT_TIMER1_CMP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST 表示 LEDC_EVT_TIMER2_CMP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST 表示 LEDC_EVT_TIMER3_CMP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_TGO_EVT_CNT_CMP_TIMER0_ST 表示 TGO_EVT_CNT_CMP_TIMER0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_TG0_EVT_CNT_CMP_TIMER1_ST 表示 TG0_EVT_CNT_CMP_TIMER1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST 表示 TG1_EVT_CNT_CMP_TIMER0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST 表示 TG1_EVT_CNT_CMP_TIMER1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMPO_ST 表示 SYSTIMER_EVT_CNT_CMPO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST 表示 SYSTIMER_EVT_CNT_CMP1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST 表示 SYSTIMER_EVT_CNT_CMP2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER0_STOP_ST 表示 MCPWMO_EVT_TIMER0_STOP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER1_STOP_ST 表示 MCPWMO_EVT_TIMER1_STOP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER2_STOP_ST 表示 MCPWMO_EVT_TIMER2_STOP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.4. SOC_ETM_EVT_ST1_REG (0x01B0)

接上页...

SOC_ETM_MCPWMO_EVT_TIMER0_TEZ_ST 表示 MCPWMO_EVT_TIMER0_TEZ 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER1_TEZ_ST 表示 MCPWMO_EVT_TIMER1_TEZ 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER2_TEZ_ST 表示 MCPWMO_EVT_TIMER2_TEZ 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER0_TEP_ST 表示 MCPWMO_EVT_TIMER0_TEP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER1_TEP_ST 表示 MCPWMO_EVT_TIMER1_TEP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TIMER2_TEP_ST 表示 MCPWMO_EVT_TIMER2_TEP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OPO_TEA_ST 表示 MCPWMO_EVT_OPO_TEA 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP1_TEA_ST 表示 MCPWMO_EVT_OP1_TEA 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.5. SOC_ETM_EVT_ST2_REG (0x01B8)

SOC_ETM_ADC_EVT_STOPPED0_ST	SOC_ETM_ADC_EVT_RESULT_DONE0_ST	SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST	SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST	SOC_ETM_ADC_EVT_CONV_CMPLT0_ST	SOC_ETM_MCPWMO_EVT_OP2_TEE2_ST	SOC_ETM_MCPWMO_EVT_OP1_TEE2_ST	SOC_ETM_MCPWMO_EVT_OP2_TEE2_ST	SOC_ETM_MCPWMO_EVT_OP1_TEE1_ST	SOC_ETM_MCPWMO_EVT_OP0_TEE1_ST	SOC_ETM_MCPWMO_EVT_CAP2_ST	SOC_ETM_MCPWMO_EVT_CAP1_ST	SOC_ETM_MCPWMO_EVT_CAPO_ST	SOC_ETM_MCPWMO_EVT_TZ2_OST_ST	SOC_ETM_MCPWMO_EVT_TZ1_OST_ST	SOC_ETM_MCPWMO_EVT_TZ0_OST_ST	SOC_ETM_MCPWMO_EVT_TZ2_CBC_ST	SOC_ETM_MCPWMO_EVT_TZ1_CBC_ST	SOC_ETM_MCPWMO_EVT_TZ0_CBC_ST	SOC_ETM_MCPWMO_EVT_F2_CLR_ST	SOC_ETM_MCPWMO_EVT_F1_CLR_ST	SOC_ETM_MCPWMO_EVT_F0_CLR_ST	SOC_ETM_MCPWMO_EVT_F2_ST	SOC_ETM_MCPWMO_EVT_F1_ST	SOC_ETM_MCPWMO_EVT_F0_ST	SOC_ETM_MCPWMO_EVT_OP2_TEB_ST	SOC_ETM_MCPWMO_EVT_OP1_TEB_ST	SOC_ETM_MCPWMO_EVT_OP0_TEB_ST	SOC_ETM_MCPWMO_EVT_OP2_TEA_ST			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_MCPWMO_EVT_OP2_TEA_ST 表示 MCPWMO_EVT_OP2_TEA 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP0_TEB_ST 表示 MCPWMO_EVT_OP0_TEB 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP1_TEB_ST 表示 MCPWMO_EVT_OP1_TEB 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP2_TEB_ST 表示 MCPWMO_EVT_OP2_TEB 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_MCPWMO_EVT_F0_ST 表示 MCPWMO_EVT_F0 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_MCPWMO_EVT_F1_ST 表示 MCPWMO_EVT_F1 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_MCPWMO_EVT_F2_ST 表示 MCPWMO_EVT_F2 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

Register 8.5. SOC_ETM_EVT_ST2_REG (0x01B8)

[接上页...](#)

SOC_ETM_MCPWMO_EVT_F0_CLR_ST 表示 MCPWMO_EVT_F0_CLR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_F1_CLR_ST 表示 MCPWMO_EVT_F1_CLR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_F2_CLR_ST 表示 MCPWMO_EVT_F2_CLR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TZ0_CBC_ST 表示 MCPWMO_EVT_TZ0_CBC 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TZ1_CBC_ST 表示 MCPWMO_EVT_TZ1_CBC 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TZ2_CBC_ST 表示 MCPWMO_EVT_TZ2_CBC 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TZ0_OST_ST 表示 MCPWMO_EVT_TZ0_OST 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TZ1_OST_ST 表示 MCPWMO_EVT_TZ1_OST 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_TZ2_OST_ST 表示 MCPWMO_EVT_TZ2_OST 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

[见下页...](#)

Register 8.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_MCPWMO_EVT_CAPO_ST 表示 MCPWMO_EVT_CAPO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_CAP1_ST 表示 MCPWMO_EVT_CAP1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_CAP2_ST 表示 MCPWMO_EVT_CAP2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OPO_TEE1_ST 表示 MCPWMO_EVT_OPO_TEE1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP1_TEE1_ST 表示 MCPWMO_EVT_OP1_TEE1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP2_TEE1_ST 表示 MCPWMO_EVT_OP2_TEE1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OPO_TEE2_ST 表示 MCPWMO_EVT_OPO_TEE2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP1_TEE2_ST 表示 MCPWMO_EVT_OP1_TEE2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_EVT_OP2_TEE2_ST 表示 MCPWMO_EVT_OP2_TEE2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.5. SOC_ETM_EVT_ST2_REG (0x01B8)

接上页...

SOC_ETM_ADC_EVT_CONV_CMPLTO_ST 表示 ADC_EVT_CONV_CMPLTO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH0_ST 表示 ADC_EVT_EQ_ABOVE_THRESH0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST 表示 ADC_EVT_EQ_ABOVE_THRESH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST 表示 ADC_EVT_EQ_BELOW_THRESH0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST 表示 ADC_EVT_EQ_BELOW_THRESH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST 表示 ADC_EVT_RESULT_DONE0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_EVT_STOPPED0_ST 表示 ADC_EVT_STOPPED0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.6. SOC_ETM_EVT_ST3_REG (0x01C0)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH2_ST	SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH1_ST	SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH2_ST	SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH1_ST	SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH2_ST	SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH1_ST	SOC_ETM_GDMA_EVT_IN_DONE_CH2_ST	SOC_ETM_GDMA_EVT_IN_DONE_CH1_ST	SOC_ETM_RTC_EVT_CMP_ST	SOC_ETM_RTC_EVT_OVF_ST	SOC_ETM_ULP_EVT_TICK_ST	SOC_ETM_ULP_EVT_START_INTR_ST	SOC_ETM_ULP_EVT_HALT_ST	SOC_ETM_I2SO_EVT_ERR_INTR_ST	SOC_ETM_I2SO_EVT_X_WORDS_SENT_ST	SOC_ETM_I2SO_EVT_TX_DONE_ST	SOC_ETM_I2SO_EVT_RX_DONE_ST	SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST	(reserved)	SOC_ETM_ADC_EVT_STARTEDO_ST								
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8		1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_ADC_EVT_STARTEDO_ST 表示 ADC_EVT_STARTEDO 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST 表示 TMPSNSR_EVT_OVER_LIMIT 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_I2SO_EVT_RX_DONE_ST 表示 I2SO_EVT_RX_DONE 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_I2SO_EVT_TX_DONE_ST 表示 I2SO_EVT_TX_DONE 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_I2SO_EVT_X_WORDS_RECEIVED_ST 表示 I2SO_EVT_X_WORDS_RECEIVED 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_I2SO_EVT_X_WORDS_SENT_ST 表示 I2SO_EVT_X_WORDS_SENT 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_ULP_EVT_ERR_INTR_ST 表示 ULP_EVT_ERR_INTR 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

Register 8.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_ULP_EVT_HALT_ST 表示 ULP_EVT_HALT 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ULP_EVT_START_INTR_ST 表示 ULP_EVT_START_INTR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_EVT_TICK_ST 表示 RTC_EVT_TICK 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_EVT_OVF_ST 表示 RTC_EVT_OVF 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_EVT_CMP_ST 表示 RTC_EVT_CMP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_DONE_CHO_ST 表示 GDMA_EVT_IN_DONE_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_DONE_CH1_ST 表示 GDMA_EVT_IN_DONE_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_DONE_CH2_ST 表示 GDMA_EVT_IN_DONE_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CHO_ST 表示 GDMA_EVT_IN_SUC_EOF_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.6. SOC_ETM_EVT_ST3_REG (0x01C0)

接上页...

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH1_ST 表示 GDMA_EVT_IN_SUC_EOF_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH2_ST 表示 GDMA_EVT_IN_SUC_EOF_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CHO_ST 表示 GDMA_EVT_IN_FIFO_EMPTY_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH1_ST 表示 GDMA_EVT_IN_FIFO_EMPTY_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH2_ST 表示 GDMA_EVT_IN_FIFO_EMPTY_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CHO_ST 表示 GDMA_EVT_IN_FIFO_FULL_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

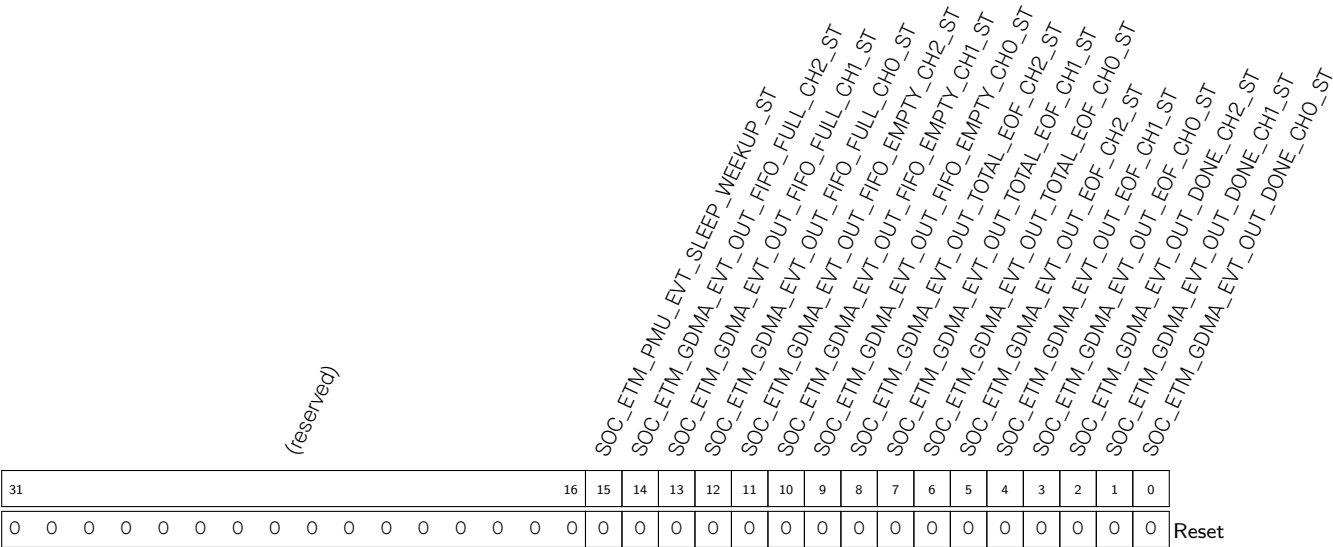
SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH1_ST 表示 GDMA_EVT_IN_FIFO_FULL_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH2_ST 表示 GDMA_EVT_IN_FIFO_FULL_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.7. SOC_ETM_EVT_ST4_REG (0x01C8)



SOC_ETM_GDMA_EVT_OUT_DONE_CH0_ST 表示 GDMA_EVT_OUT_DONE_CH0 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_DONE_CH1_ST 表示 GDMA_EVT_OUT_DONE_CH1 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_DONE_CH2_ST 表示 GDMA_EVT_OUT_DONE_CH2 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_EOF_CH0_ST 表示 GDMA_EVT_OUT_EOF_CH0 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_EOF_CH1_ST 表示 GDMA_EVT_OUT_EOF_CH1 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_EOF_CH2_ST 表示 GDMA_EVT_OUT_EOF_CH2 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH0_ST 表示 GDMA_EVT_OUT_TOTAL_EOF_CH0 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.7. SOC_ETM_EVT_ST4_REG (0x01C8)

接上页...

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH1_ST 表示 GDMA_EVT_OUT_TOTAL_EOF_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH2_ST 表示 GDMA_EVT_OUT_TOTAL_EOF_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH0_ST 表示 GDMA_EVT_OUT_FIFO_EMPTY_CH0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH1_ST 表示 GDMA_EVT_OUT_FIFO_EMPTY_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH2_ST 表示 GDMA_EVT_OUT_FIFO_EMPTY_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH0_ST 表示 GDMA_EVT_OUT_FIFO_FULL_CH0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH1_ST 表示 GDMA_EVT_OUT_FIFO_FULL_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.7. SOC_ETM_EVT_ST4_REG (0x01C8)

[接上页...](#)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH2_ST 表示 GDMA_EVT_OUT_FIFO_FULL_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST 表示 PMU_EVT_SLEEP_WEEKUP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

Register 8.8. SOC_ETM_TASK_STO_REG (0x01D0)

Soc																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SOC_ETM_LEDC_Task_Duty_Scale_Update_Ch3_St																																
SOC_ETM_LEDC_Task_Duty_Scale_Update_Ch2_St																																
SOC_ETM_LEDC_Task_Duty_Scale_Update_Ch1_St																																
SOC_ETM_LEDC_Task_Timer3_Res_Update_Ch0_St																																
SOC_ETM_LEDC_Task_Timer2_Res_Update_St																																
SOC_ETM_GPIO_Task_Timer1_Res_Update_St																																
SOC_ETM_GPIO_Task_Ch7_Toggle_St																																
SOC_ETM_GPIO_Task_Ch6_Toggle_St																																
SOC_ETM_GPIO_Task_Ch5_Toggle_St																																
SOC_ETM_GPIO_Task_Ch4_Toggle_St																																
SOC_ETM_GPIO_Task_Ch3_Toggle_St																																
SOC_ETM_GPIO_Task_Ch2_Toggle_St																																
SOC_ETM_GPIO_Task_Ch1_Toggle_St																																
SOC_ETM_GPIO_Task_Ch0_Toggle_St																																
SOC_ETM_GPIO_Task_Ch7_Clear_St																																
SOC_ETM_GPIO_Task_Ch6_Clear_St																																
SOC_ETM_GPIO_Task_Ch5_Clear_St																																
SOC_ETM_GPIO_Task_Ch4_Clear_St																																
SOC_ETM_GPIO_Task_Ch3_Clear_St																																
SOC_ETM_GPIO_Task_Ch2_Clear_St																																
SOC_ETM_GPIO_Task_Ch1_Clear_St																																
SOC_ETM_GPIO_Task_Ch0_Clear_St																																
SOC_ETM_GPIO_Task_Ch7_Set_St																																
SOC_ETM_GPIO_Task_Ch6_Set_St																																
SOC_ETM_GPIO_Task_Ch5_Set_St																																
SOC_ETM_GPIO_Task_Ch4_Set_St																																
SOC_ETM_GPIO_Task_Ch3_Set_St																																
SOC_ETM_GPIO_Task_Ch2_Set_St																																
SOC_ETM_GPIO_Task_Ch1_Set_St																																
SOC_ETM_GPIO_Task_Ch0_Set_St																																

SOC_ETM_GPIO_TASK_CH0_SET_ST 表示 GPIO_TASK_CH0_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_SET_ST 表示 GPIO_TASK_CH1_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_SET_ST 表示 GPIO_TASK_CH2_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_SET_ST 表示 GPIO_TASK_CH3_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_SET_ST 表示 GPIO_TASK_CH4_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_SET_ST 表示 GPIO_TASK_CH5_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_SET_ST 表示 GPIO_TASK_CH6_SET 的触发状态。

- 0: 未触发
- 1: 已触发
- (R/WTC/SS)

Register 8.8. SOC_ETM_TASK_STO_REG (0x01D0)

接上页...

SOC_ETM_GPIO_TASK_CH7_SET_ST 表示 GPIO_TASK_CH7_SET 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST 表示 GPIO_TASK_CH0_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST 表示 GPIO_TASK_CH1_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST 表示 GPIO_TASK_CH2_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST 表示 GPIO_TASK_CH3_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST 表示 GPIO_TASK_CH4_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST 表示 GPIO_TASK_CH5_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST 表示 GPIO_TASK_CH6_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST 表示 GPIO_TASK_CH7_CLEAR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.8. SOC_ETM_TASK_STO_REG (0x01D0)

[接上页...](#)

SOC_ETM_GPIO_TASK_CHO_TOGGLE_ST 表示 GPIO_TASK_CHO_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST 表示 GPIO_TASK_CH1_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST 表示 GPIO_TASK_CH2_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST 表示 GPIO_TASK_CH3_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST 表示 GPIO_TASK_CH4_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST 表示 GPIO_TASK_CH5_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST 表示 GPIO_TASK_CH6_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST 表示 GPIO_TASK_CH7_TOGGLE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

[见下页...](#)

Register 8.8. SOC_ETM_TASK_STO_REG (0x01D0)

接上页...

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST 表示 LEDC_TASK_TIMER0_RES_UPDATE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST 表示 LEDC_TASK_TIMER1_RES_UPDATE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST 表示 LEDC_TASK_TIMER2_RES_UPDATE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST 表示 LEDC_TASK_TIMER3_RES_UPDATE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CHO_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CHO 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH3 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

Register 8.9. SOC_ETM_TASK_ST1_REG (0x01D8)

SOC_ETM_LEDC_Task_GAMMA_RESTART_CH1_ST	SOC_ETM_LEDC_Task_GAMMA_RESTART_CH0_ST	SOC_ETM_LEDC_Task_TIMER3_PAUSE_ST	SOC_ETM_LEDC_Task_TIMER2_PAUSE_ST	SOC_ETM_LEDC_Task_TIMER1_PAUSE_ST	SOC_ETM_LEDC_Task_TIMER0_PAUSE_ST	SOC_ETM_LEDC_Task_TIMER3_RESUME_ST	SOC_ETM_LEDC_Task_TIMER2_RESUME_ST	SOC_ETM_LEDC_Task_TIMER1_RESUME_ST	SOC_ETM_LEDC_Task_TIMER0_RESUME_ST	SOC_ETM_LEDC_Task_TIMER3_RST_ST	SOC_ETM_LEDC_Task_TIMER2_RST_ST	SOC_ETM_LEDC_Task_TIMER1_RST_ST	SOC_ETM_LEDC_Task_TIMER0_RST_ST	SOC_ETM_LEDC_Task_OVF_CNT_RST_CH5_ST	SOC_ETM_LEDC_Task_OVF_CNT_RST_CH4_ST	SOC_ETM_LEDC_Task_OVF_CNT_RST_CH3_ST	SOC_ETM_LEDC_Task_OVF_CNT_RST_CH2_ST	SOC_ETM_LEDC_Task_SIG_OUT_RST_CH1_ST	SOC_ETM_LEDC_Task_SIG_OUT_DIS_CH5_ST	SOC_ETM_LEDC_Task_SIG_OUT_DIS_CH4_ST	SOC_ETM_LEDC_Task_SIG_OUT_DIS_CH3_ST	SOC_ETM_LEDC_Task_SIG_OUT_DIS_CH2_ST	SOC_ETM_LEDC_Task_TIMER3_CAP_ST	SOC_ETM_LEDC_Task_TIMER2_CAP_ST	SOC_ETM_LEDC_Task_TIMER1_CAP_ST	SOC_ETM_LEDC_Task_TIMER0_CAP_ST	Reset				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH4 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST 表示 LEDC_TASK_DUTY_SCALE_UPDATE_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST 表示 LEDC_TASK_TIMER0_CAP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST 表示 LEDC_TASK_TIMER1_CAP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST 表示 LEDC_TASK_TIMER2_CAP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST 表示 LEDC_TASK_TIMER3_CAP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.9. SOC_ETM_TASK_ST1_REG (0x01D8)

[接上页...](#)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CHO_ST 表示 LEDC_TASK_SIG_OUT_DIS_CHO 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH3 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH4 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST 表示 LEDC_TASK_SIG_OUT_DIS_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CHO_ST 表示 LEDC_TASK_OVF_CNT_RST_CHO 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

[见下页...](#)

Register 8.9. SOC_ETM_TASK_ST1_REG (0x01D8)

[接上页...](#)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST 表示 LEDC_TASK_OVF_CNT_RST_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST 表示 LEDC_TASK_OVF_CNT_RST_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST 表示 LEDC_TASK_OVF_CNT_RST_CH3 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST 表示 LEDC_TASK_OVF_CNT_RST_CH4 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST 表示 LEDC_TASK_OVF_CNT_RST_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST 表示 LEDC_TASK_TIMER0_RST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RST_ST 表示 LEDC_TASK_TIMER1_RST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

[见下页...](#)

Register 8.9. SOC_ETM_TASK_ST1_REG (0x01D8)

接上页...

SOC_ETM_LEDC_TASK_TIMER2_RST_ST 表示 LEDC_TASK_TIMER2_RST 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST 表示 LEDC_TASK_TIMER3_RST 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST 表示 LEDC_TASK_TIMER0_RESUME 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST 表示 LEDC_TASK_TIMER1_RESUME 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST 表示 LEDC_TASK_TIMER2_RESUME 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST 表示 LEDC_TASK_TIMER3_RESUME 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.9. SOC_ETM_TASK_ST1_REG (0x01D8)

[接上页...](#)

SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST 表示 LEDC_TASK_TIMER0_PAUSE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST 表示 LEDC_TASK_TIMER1_PAUSE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST 表示 LEDC_TASK_TIMER2_PAUSE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST 表示 LEDC_TASK_TIMER3_PAUSE 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CHO_ST 表示 LEDC_TASK_GAMMA_RESTART_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST 表示 LEDC_TASK_GAMMA_RESTART_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.10. SOC_ETM_TASK_ST2_REG (0x01E0)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST																															
SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST																															
SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST																															
SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST																															
SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST																															
SOC_ETM_TG0_TASK_CNT_START_TIMER0_ST																															
SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST																															
SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER0_ST																															
SOC_ETM_TG0_TASK_ALARM_START_TIMER0_ST																															
SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST																															
SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST																															
SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST																															
SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST																															
SOC_ETM_TG0_TASK_CNT_START_TIMER0_ST																															
SOC_ETM_TG0_TASK_ALARM_START_TIMER0_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST																															

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST 表示 LEDC_TASK_GAMMA_RESTART_CH2 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST 表示 LEDC_TASK_GAMMA_RESTART_CH3 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST 表示 LEDC_TASK_GAMMA_RESTART_CH4 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST 表示 LEDC_TASK_GAMMA_RESTART_CH5 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CHO_ST 表示 LEDC_TASK_GAMMA_PAUSE_CHO 的触发状态。
0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH3 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH4 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST 表示 LEDC_TASK_GAMMA_PAUSE_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST 表示 LEDC_TASK_GAMMA_RESUME_CH0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST 表示 LEDC_TASK_GAMMA_RESUME_CH1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST 表示 LEDC_TASK_GAMMA_RESUME_CH2 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST 表示 LEDC_TASK_GAMMA_RESUME_CH3 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST 表示 LEDC_TASK_GAMMA_RESUME_CH4 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST 表示 LEDC_TASK_GAMMA_RESUME_CH5 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST 表示 TGO_TASK_CNT_START_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST 表示 TGO_TASK_ALARM_START_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST 表示 TGO_TASK_CNT_STOP_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.10. SOC_ETM_TASK_ST2_REG (0x01E0)

接上页...

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST 表示 TGO_TASK_CNT_RELOAD_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_CAP_TIMER0_ST 表示 TGO_TASK_CNT_CAP_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_START_TIMER1_ST 表示 TGO_TASK_CNT_START_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_ALARM_START_TIMER1_ST 表示 TGO_TASK_ALARM_START_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER1_ST 表示 TGO_TASK_CNT_STOP_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER1_ST 表示 TGO_TASK_CNT_RELOAD_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TGO_TASK_CNT_CAP_TIMER1_ST 表示 TGO_TASK_CNT_CAP_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST 表示 TG1_TASK_CNT_START_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.10. SOC_ETM_TASK_ST2_REG (0x01E0)

[接上页...](#)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST 表示 TG1_TASK_ALARM_START_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST 表示 TG1_TASK_CNT_STOP_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST 表示 TG1_TASK_CNT_RELOAD_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST 表示 TG1_TASK_CNT_CAP_TIMER0 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST 表示 TG1_TASK_CNT_START_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

Register 8.11. SOC_ETM_TASK_ST3_REG (0x01E8)

[illegible]

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST 表示 TG1_TASK_ALARM_START_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST 表示 TG1_TASK_CNT_STOP_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST 表示 TG1_TASK_CNT_RELOAD_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST 表示 TG1_TASK_CNT_CAP_TIMER1 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CMPRO_A_UP_ST 表示 MCPWMO_TASK_CMPRO_A_UP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWM0_TASK_CMPR1_A_UP_ST 表示 MCPWM0_TASK_CMPR1_A_UP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.11. SOC_ETM_TASK_ST3_REG (0x01E8)

接上页...

SOC_ETM_MCPWMO_TASK_CMPR2_A_UP_ST 表示 MCPWMO_TASK_CMPR2_A_UP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CMPRO_B_UP_ST 表示 MCPWMO_TASK_CMPRO_B_UP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CMPR1_B_UP_ST 表示 MCPWMO_TASK_CMPR1_B_UP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CMPR2_B_UP_ST 表示 MCPWMO_TASK_CMPR2_B_UP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_GEN_STOP_ST 表示 MCPWMO_TASK_GEN_STOP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TIMER0_SYN_ST 表示 MCPWMO_TASK_TIMER0_SYN 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TIMER1_SYN_ST 表示 MCPWMO_TASK_TIMER1_SYN 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TIMER2_SYN_ST 表示 MCPWMO_TASK_TIMER2_SYN 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.11. SOC_ETM_TASK_ST3_REG (0x01E8)

接上页...

SOC_ETM_MCPWMO_TASK_TIMER0_PERIOD_UP_ST 表示 MCPWMO_TASK_TIMER0_PERIOD_UP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TIMER1_PERIOD_UP_ST 表示 MCPWMO_TASK_TIMER1_PERIOD_UP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TIMER2_PERIOD_UP_ST 表示 MCPWMO_TASK_TIMER2_PERIOD_UP 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TZO_OST_ST 表示 MCPWMO_TASK_TZO_OST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TZ1_OST_ST 表示 MCPWMO_TASK_TZ1_OST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_TZ2_OST_ST 表示 MCPWMO_TASK_TZ2_OST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CLR0_OST_ST 表示 MCPWMO_TASK_CLR0_OST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CLR1_OST_ST 表示 MCPWMO_TASK_CLR1_OST 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)

见下页...

Register 8.11. SOC_ETM_TASK_ST3_REG (0x01E8)

[接上页...](#)

SOC_ETM_MCPWMO_TASK_CLR2_OST_ST 表示 MCPWMO_TASK_CLR2_OST 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CAP0_ST 表示 MCPWMO_TASK_CAP0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CAP1_ST 表示 MCPWMO_TASK_CAP1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_MCPWMO_TASK_CAP2_ST 表示 MCPWMO_TASK_CAP2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_TASK_SAMPLE0_ST 表示 ADC_TASK_SAMPLE0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

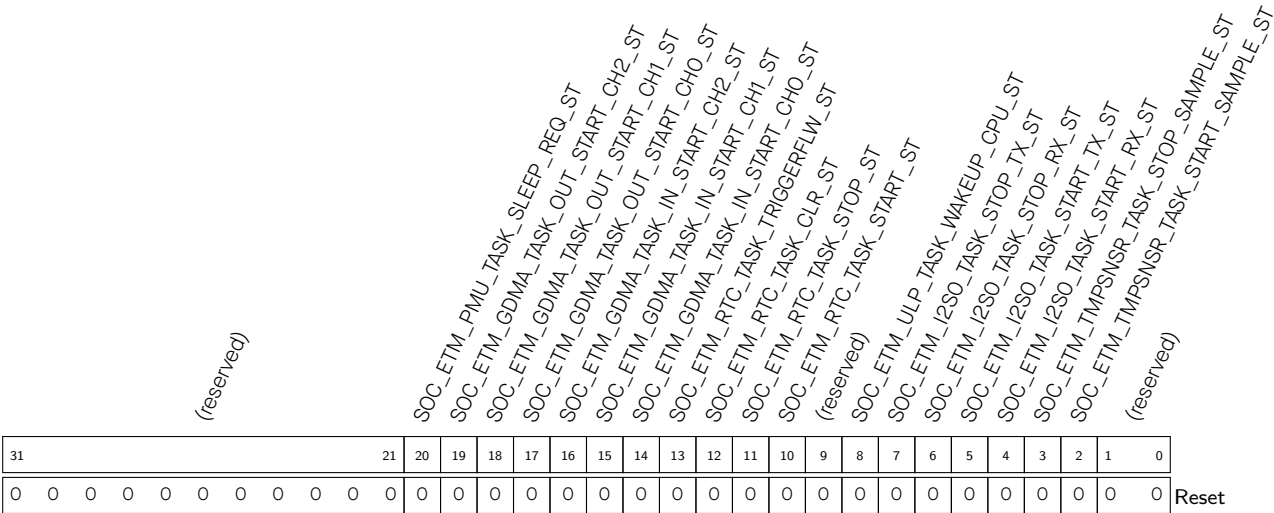
SOC_ETM_ADC_TASK_START0_ST 表示 ADC_TASK_START0 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ADC_TASK_STOPO_ST 表示 ADC_TASK_STOPO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.12. SOC_ETM_TASK_ST4_REG (0x01F0)



- SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST

表示 TMPSNSR_TASK_START_SAMPLE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)
- SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST

表示 TMPSNSR_TASK_STOP_SAMPLE 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)
- SOC_ETM_I2SO_TASK_START_RX_ST

表示 I2SO_TASK_START_RX 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)
- SOC_ETM_I2SO_TASK_START_TX_ST

表示 I2SO_TASK_START_TX 的触发状态。

0: 未触发

1: 已触发

(R/WTC/SS)
- 见下页...

Register 8.12. SOC_ETM_TASK_ST4_REG (0x01F0)

接上页...

SOC_ETM_I2SO_TASK_STOP_RX_ST 表示 I2SO_TASK_STOP_RX 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_I2SO_TASK_STOP_TX_ST 表示 I2SO_TASK_STOP_TX 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST 表示 ULP_TASK_WAKEUP_CPU 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_TASK_START_ST 表示 RTC_TASK_START 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_TASK_STOP_ST 表示 RTC_TASK_STOP 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_TASK_CLR_ST 表示 RTC_TASK_CLR 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_RTC_TASK_TRIGGERFLW_ST 表示 RTC_TASK_TRIGGERFLW 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

见下页...

Register 8.12. SOC_ETM_TASK_ST4_REG (0x01F0)

接上页...

SOC_ETM_GDMA_TASK_IN_START_CHO_ST 表示 GDMA_TASK_IN_START_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_TASK_IN_START_CH1_ST 表示 GDMA_TASK_IN_START_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_TASK_IN_START_CH2_ST 表示 GDMA_TASK_IN_START_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_TASK_OUT_START_CHO_ST 表示 GDMA_TASK_OUT_START_CHO 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_TASK_OUT_START_CH1_ST 表示 GDMA_TASK_OUT_START_CH1 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_GDMA_TASK_OUT_START_CH2_ST 表示 GDMA_TASK_OUT_START_CH2 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST 表示 PMU_TASK_SLEEP_REQ 的触发状态。

0: 未触发
1: 已触发
(R/WTC/SS)

Register 8.13. SOC_ETM_CH_ENA_ADO_SET_REG (0x0004)

SOC_ETM_CH_ENABLE31	SOC_ETM_CH_ENABLE30	SOC_ETM_CH_ENABLE29	SOC_ETM_CH_ENABLE28	SOC_ETM_CH_ENABLE27	SOC_ETM_CH_ENABLE26	SOC_ETM_CH_ENABLE25	SOC_ETM_CH_ENABLE24	SOC_ETM_CH_ENABLE23	SOC_ETM_CH_ENABLE22	SOC_ETM_CH_ENABLE21	SOC_ETM_CH_ENABLE20	SOC_ETM_CH_ENABLE19	SOC_ETM_CH_ENABLE18	SOC_ETM_CH_ENABLE17	SOC_ETM_CH_ENABLE16	SOC_ETM_CH_ENABLE15	SOC_ETM_CH_ENABLE14	SOC_ETM_CH_ENABLE13	SOC_ETM_CH_ENABLE12	SOC_ETM_CH_ENABLE11	SOC_ETM_CH_ENABLE10	SOC_ETM_CH_ENABLE9	SOC_ETM_CH_ENABLE8	SOC_ETM_CH_ENABLE7	SOC_ETM_CH_ENABLE6	SOC_ETM_CH_ENABLE5	SOC_ETM_CH_ENABLE4	SOC_ETM_CH_ENABLE3	SOC_ETM_CH_ENABLE2	SOC_ETM_CH_ENABLE1	SOC_ETM_CH_ENABLE0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_CH_ENABLE n (n : 0-31) 配置是否使能通道 n 。

0: 无效值, 没有作用

1: 使能

(WT)

Register 8.14. SOC_ETM_CH_ENA_ADO_CLR_REG (0x0008)

SOC_ETM_CH_DISABLE31	SOC_ETM_CH_DISABLE30	SOC_ETM_CH_DISABLE29	SOC_ETM_CH_DISABLE28	SOC_ETM_CH_DISABLE27	SOC_ETM_CH_DISABLE26	SOC_ETM_CH_DISABLE25	SOC_ETM_CH_DISABLE24	SOC_ETM_CH_DISABLE23	SOC_ETM_CH_DISABLE22	SOC_ETM_CH_DISABLE21	SOC_ETM_CH_DISABLE20	SOC_ETM_CH_DISABLE19	SOC_ETM_CH_DISABLE18	SOC_ETM_CH_DISABLE17	SOC_ETM_CH_DISABLE16	SOC_ETM_CH_DISABLE15	SOC_ETM_CH_DISABLE14	SOC_ETM_CH_DISABLE13	SOC_ETM_CH_DISABLE12	SOC_ETM_CH_DISABLE11	SOC_ETM_CH_DISABLE10	SOC_ETM_CH_DISABLE9	SOC_ETM_CH_DISABLE8	SOC_ETM_CH_DISABLE7	SOC_ETM_CH_DISABLE6	SOC_ETM_CH_DISABLE5	SOC_ETM_CH_DISABLE4	SOC_ETM_CH_DISABLE3	SOC_ETM_CH_DISABLE2	SOC_ETM_CH_DISABLE1	SOC_ETM_CH_DISABLE0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_CH_DISABLE n (n : 0-31) 配置是否关闭通道 n 。

0: 无效值, 没有作用

1: 关闭

(WT)

Register 8.15. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)

(reserved)																															SOC_ETM_CH_ENABLE49 SOC_ETM_CH_ENABLE48 SOC_ETM_CH_ENABLE47 SOC_ETM_CH_ENABLE46 SOC_ETM_CH_ENABLE45 SOC_ETM_CH_ENABLE44 SOC_ETM_CH_ENABLE43 SOC_ETM_CH_ENABLE42 SOC_ETM_CH_ENABLE41 SOC_ETM_CH_ENABLE40 SOC_ETM_CH_ENABLE39 SOC_ETM_CH_ENABLE38 SOC_ETM_CH_ENABLE37 SOC_ETM_CH_ENABLE36 SOC_ETM_CH_ENABLE35 SOC_ETM_CH_ENABLE34 SOC_ETM_CH_ENABLE33																
31														18														17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset											

SOC_ETM_CH_ENABLE n (n : 32-49) 配置是否使能通道 n 。

0: 无效值, 没有作用

1: 使能

(WT)

Register 8.16. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)

(reserved)																															SOC_ETM_CH_DISABLE49 SOC_ETM_CH_DISABLE48 SOC_ETM_CH_DISABLE47 SOC_ETM_CH_DISABLE46 SOC_ETM_CH_DISABLE45 SOC_ETM_CH_DISABLE44 SOC_ETM_CH_DISABLE43 SOC_ETM_CH_DISABLE42 SOC_ETM_CH_DISABLE41 SOC_ETM_CH_DISABLE40 SOC_ETM_CH_DISABLE39 SOC_ETM_CH_DISABLE38 SOC_ETM_CH_DISABLE37 SOC_ETM_CH_DISABLE36 SOC_ETM_CH_DISABLE35 SOC_ETM_CH_DISABLE34 SOC_ETM_CH_DISABLE33																
31																	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0												
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset													

SOC_ETM_CH_DISABLE n (n : 32-49) 配置是否关闭通道 n 。

0: 无效值, 没有作用

1: 清除

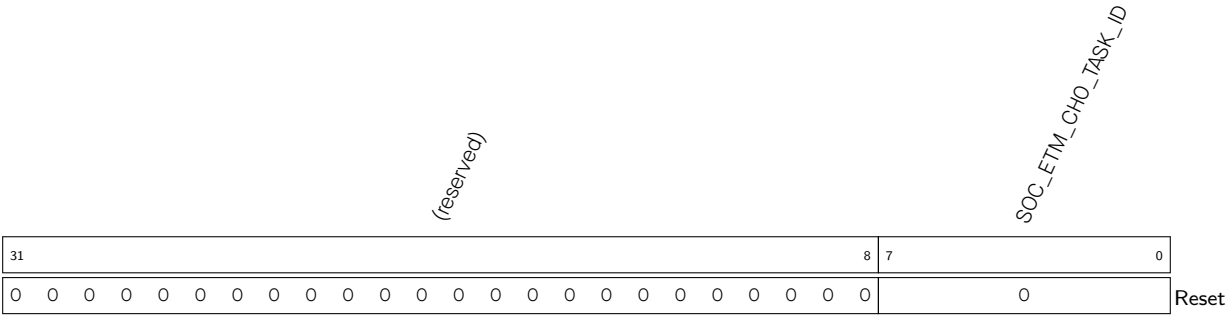
(WT)

Register 8.17. SOC_ETM_CH n _EVT_ID_REG (n : 0-49) (0x0018+0x8* n)

(reserved)																								SOC_ETM_CH0_EVT_ID																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																								8	7	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SOC_ETM_CH n _EVT_ID 配置通道 n 的事件 ID。(R/W)

Register 8.18. SOC_ETM_CH n _TASK_ID_REG (n : 0-49) (0x001C+0x8* n)



SOC_ETM_CH n _TASK_ID 配置通道 n 的任务 ID。(R/W)

Register 8.19. SOC_ETM_EVT_STO_CLR_REG (0x01AC)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR	31	0
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR	30	0
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR	29	0
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR	28	0
SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR	27	0
SOC_ETM_GPIO_EVT_ZERO_DET_NEGO_CH0_ST_CLR	26	0
SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR	25	0
SOC_ETM_GPIO_EVT_ANY_EDGE_ST_CLR	24	0
SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR	23	0
SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR	22	0
SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR	21	0
SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR	20	0
SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR	19	0
SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR	18	0
SOC_ETM_GPIO_EVT_CH0_ANY_EDGE_ST_CLR	17	0
SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR	16	0
SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR	15	0
SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR	14	0
SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR	13	0
SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR	12	0
SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR	11	0
SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR	10	0
SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR	9	0
SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR	8	0
SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR	7	0
SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR	6	0
SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR	5	0
SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR	4	0
SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR	3	0
SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR	2	0
SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR	1	0
Reset	0	0

SOC_ETM_GPIO_EVT_CH0_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH0_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH1_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH1_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH2_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH2_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH3_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH3_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH4_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH4_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.19. SOC_ETM_EVT_STO_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH5_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH5_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH6_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH6_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH7_RISE_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH7_RISE_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH0_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH0_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH1_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH1_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH2_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH2_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH3_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH3_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.19. SOC_ETM_EVT_STO_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH4_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH4_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH5_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH5_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH6_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH6_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH7_FALL_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH7_FALL_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CHO_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CHO_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH1_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH1_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH2_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH2_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.19. SOC_ETM_EVT_STO_CLR_REG (0x01AC)

接上页...

SOC_ETM_GPIO_EVT_CH3_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH3_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH4_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH4_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH5_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH5_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH6_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH6_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_CH7_ANY_EDGE_ST_CLR 配置是否清除 GPIO_EVT_CH7_ANY_EDGE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_POS0_ST_CLR 配置是否清除 GPIO_EVT_ZERO_DET_POS0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_EVT_ZERO_DET_NEG0_ST_CLR 配置是否清除 GPIO_EVT_ZERO_DET_NEG0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.19. SOC_ETM_EVT_STO_CLR_REG (0x01AC)

接上页...

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH0_ST_CLR	配	置	是	否	清	除

LEDC_EVT_DUTY_CHNG_END_CH0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH1_ST_CLR	配	置	是	否	清	除

LEDC_EVT_DUTY_CHNG_END_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH2_ST_CLR	配	置	是	否	清	除

LEDC_EVT_DUTY_CHNG_END_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH3_ST_CLR	配	置	是	否	清	除

LEDC_EVT_DUTY_CHNG_END_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH4_ST_CLR	配	置	是	否	清	除

LEDC_EVT_DUTY_CHNG_END_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_DUTY_CHNG_END_CH5_ST_CLR	配	置	是	否	清	除

LEDC_EVT_DUTY_CHNG_END_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

[illegible]

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CHO_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH1_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH2_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH3_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH4_ST_CLR 配 置 是 否 清 除

LEDC_EVT_OVF_CNT_PLS_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_LEDC_EVT_OVF_CNT_PLS_CH5_ST_CLR 配置是否清除 LEDC_EVT_OVF_CNT_PLS_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER0_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER1_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER2_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIME_OVF_TIMER3_ST_CLR 配置是否清除 LEDC_EVT_TIME_OVF_TIMER3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIMER0_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER0_CMP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIMER1_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER1_CMP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

[接上页...](#)

SOC_ETM_LEDC_EVT_TIMER2_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER2_CMP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_EVT_TIMER3_CMP_ST_CLR 配置是否清除 LEDC_EVT_TIMER3_CMP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_EVT_CNT_CMP_TIMER0_ST_CLR 配置是否清除 TGO_EVT_CNT_CMP_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_EVT_CNT_CMP_TIMER1_ST_CLR 配置是否清除 TGO_EVT_CNT_CMP_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER0_ST_CLR 配置是否清除 TG1_EVT_CNT_CMP_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_EVT_CNT_CMP_TIMER1_ST_CLR 配置是否清除 TG1_EVT_CNT_CMP_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMPO_ST_CLR 配置是否清除 SYSTIMER_EVT_CNT_CMPO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

[见下页...](#)

Register 8.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_SYSTIMER_EVT_CNT_CMP1_ST_CLR 配置是否清除 SYSTIMER_EVT_CNT_CMP1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_SYSTIMER_EVT_CNT_CMP2_ST_CLR 配置是否清除 SYSTIMER_EVT_CNT_CMP2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER0_STOP_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER0_STOP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER1_STOP_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER1_STOP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER2_STOP_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER2_STOP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER0_TEZ_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER0_TEZ 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER1_TEZ_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER1_TEZ 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.20. SOC_ETM_EVT_ST1_CLR_REG (0x01B4)

接上页...

SOC_ETM_MCPWMO_EVT_TIMER2_TEZ_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER2_TEZ 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER0_TEP_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER0_TEP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER1_TEP_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER1_TEP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TIMER2_TEP_ST_CLR 配置是否清除 MCPWMO_EVT_TIMER2_TEP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OPO_TEA_ST_CLR 配置是否清除 MCPWMO_EVT_OPO_TEA 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OP1_TEA_ST_CLR 配置是否清除 MCPWMO_EVT_OP1_TEA 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

SOC_ETM_ADC_EVT_STOPPED0_ST_CLR	SOC_ETM_ADC_EVT_RESULT_DONE0_ST_CLR	SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR	SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST_CLR	SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR	SOC_ETM_ADC_EVT_CONV_CMPLT0_ST_CLR	SOC_ETM_MCPWMO_EVT_OP2_TEE2_ST_CLR	SOC_ETM_MCPWMO_EVT_OP1_TEE2_ST_CLR	SOC_ETM_MCPWMO_EVT_OP2_TEE1_ST_CLR	SOC_ETM_MCPWMO_EVT_OP1_TEE1_ST_CLR	SOC_ETM_MCPWMO_EVT_CAP2_ST_CLR	SOC_ETM_MCPWMO_EVT_CAP1_ST_CLR	SOC_ETM_MCPWMO_EVT_CAPO_ST_CLR	SOC_ETM_MCPWMO_EVT_TZ2_OST_ST_CLR	SOC_ETM_MCPWMO_EVT_TZ1_OST_ST_CLR	SOC_ETM_MCPWMO_EVT_TZ0_OST_ST_CLR	SOC_ETM_MCPWMO_EVT_TZ2_OBC_ST_CLR	SOC_ETM_MCPWMO_EVT_TZ1_OBC_ST_CLR	SOC_ETM_MCPWMO_EVT_TZ0_OBC_ST_CLR	SOC_ETM_MCPWMO_EVT_F2_CLR_ST_CLR	SOC_ETM_MCPWMO_EVT_F1_CLR_ST_CLR	SOC_ETM_MCPWMO_EVT_F0_CLR_ST_CLR	SOC_ETM_MCPWMO_EVT_F2_ST_CLR	SOC_ETM_MCPWMO_EVT_F1_ST_CLR	SOC_ETM_MCPWMO_EVT_F0_ST_CLR	SOC_ETM_MCPWMO_EVT_OP2_TEB_ST_CLR	SOC_ETM_MCPWMO_EVT_OP1_TEB_ST_CLR	SOC_ETM_MCPWMO_EVT_OP2_TEA_ST_CLR				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SOC_ETM_MCPWMO_EVT_OP2_TEA_ST_CLR 配置是否清除 MCPWMO_EVT_OP2_TEA 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_MCPWMO_EVT_OP0_TEB_ST_CLR 配置是否清除 MCPWMO_EVT_OP0_TEB 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_MCPWMO_EVT_OP1_TEB_ST_CLR 配置是否清除 MCPWMO_EVT_OP1_TEB 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_MCPWMO_EVT_OP2_TEB_ST_CLR 配置是否清除 MCPWMO_EVT_OP2_TEB 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_MCPWMO_EVT_F0_ST_CLR 配置是否清除 MCPWMO_EVT_F0 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

见下页...

Register 8.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWMO_EVT_F1_ST_CLR 配置是否清除 MCPWMO_EVT_F1 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_F2_ST_CLR 配置是否清除 MCPWMO_EVT_F2 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_FO_CLR_ST_CLR 配置是否清除 MCPWMO_EVT_FO_CLR 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_F1_CLR_ST_CLR 配置是否清除 MCPWMO_EVT_F1_CLR 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_F2_CLR_ST_CLR 配置是否清除 MCPWMO_EVT_F2_CLR 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_TZ0_CBC_ST_CLR 配置是否清除 MCPWMO_EVT_TZ0_CBC 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_TZ1_CBC_ST_CLR 配置是否清除 MCPWMO_EVT_TZ1_CBC 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_MCPWMO_EVT_TZ2_CBC_ST_CLR 配置是否清除 MCPWMO_EVT_TZ2_CBC 的触发状态。

0: 无效值, 没有作用
1: 清除
(WT)

见下页...

Register 8.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_MCPWMO_EVT_TZ0_OST_ST_CLR 配置是否清除 MCPWMO_EVT_TZ0_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TZ1_OST_ST_CLR 配置是否清除 MCPWMO_EVT_TZ1_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_TZ2_OST_ST_CLR 配置是否清除 MCPWMO_EVT_TZ2_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_CAPO_ST_CLR 配置是否清除 MCPWMO_EVT_CAPO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_CAP1_ST_CLR 配置是否清除 MCPWMO_EVT_CAP1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_CAP2_ST_CLR 配置是否清除 MCPWMO_EVT_CAP2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OPO_TEE1_ST_CLR 配置是否清除 MCPWMO_EVT_OPO_TEE1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OP1_TEE1_ST_CLR 配置是否清除 MCPWMO_EVT_OP1_TEE1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

[接上页...](#)

SOC_ETM_MCPWMO_EVT_OP2_TEE1_ST_CLR 配置是否清除 MCPWMO_EVT_OP2_TEE1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OPO_TEE2_ST_CLR 配置是否清除 MCPWMO_EVT_OPO_TEE2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OP1_TEE2_ST_CLR 配置是否清除 MCPWMO_EVT_OP1_TEE2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_EVT_OP2_TEE2_ST_CLR 配置是否清除 MCPWMO_EVT_OP2_TEE2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_EVT_CONV_CMPLT0_ST_CLR 配置是否清除 ADC_EVT_CONV_CMPLT0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESHO_ST_CLR 配置是否清除 ADC_EVT_EQ_ABOVE_THRESHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_EVT_EQ_ABOVE_THRESH1_ST_CLR 配置是否清除 ADC_EVT_EQ_ABOVE_THRESH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

[见下页...](#)

Register 8.21. SOC_ETM_EVT_ST2_CLR_REG (0x01BC)

接上页...

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH0_ST_CLR 配置是否清除

ADC_EVT_EQ_BELOW_THRESH0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_EVT_EQ_BELOW_THRESH1_ST_CLR 配置是否清除

ADC_EVT_EQ_BELOW_THRESH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_EVT_RESULT_DONE0_ST_CLR 配置是否清除 ADC_EVT_RESULT_DONE0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_EVT_STOPPED0_ST_CLR 配置是否清除 ADC_EVT_STOPPED0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

[illegible]

SOC_ETM_ADC_EVT_STARTEDDO_ST_CLR 配置是否清除 ADC_EVT_STARTEDDO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TMPSNSR_EVT_OVER_LIMIT_ST_CLR 配置是否清除 TMPSNSR_EVT_OVER_LIMIT 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_I2SO_EVT_RX_DONE_ST_CLR 配置是否清除 I2SO_EVT_RX_DONE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_I2SO_EVT_TX_DONE_ST_CLR 配置是否清除 I2SO_EVT_TX_DONE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_I2SO_EVT_X_WORDS_RECEIVED_ST_CLR 配 置 是 否 清 除 I2SO_EVT_X_WORDS_RECEIVED 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_I2SO_EVT_X_WORDS_SENT_ST_CLR 配置是否清除 I2SO_EVT_X_WORDS_SENT 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_ULP_EVT_ERR_INTR_ST_CLR 配置是否清除 ULP_EVT_ERR_INTR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ULP_EVT_HALT_ST_CLR 配置是否清除 ULP_EVT_HALT 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ULP_EVT_START_INTR_ST_CLR 配置是否清除 ULP_EVT_START_INTR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_RTC_EVT_TICK_ST_CLR 配置是否清除 RTC_EVT_TICK 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_RTC_EVT_OVF_ST_CLR 配置是否清除 RTC_EVT_OVF 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_RTC_EVT_CMP_ST_CLR 配置是否清除 RTC_EVT_CMP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_DONE_CHO_ST_CLR 配置是否清除 GDMA_EVT_IN_DONE_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_DONE_CH1_ST_CLR 配置是否清除 GDMA_EVT_IN_DONE_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_DONE_CH2_ST_CLR 配置是否清除 GDMA_EVT_IN_DONE_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CHO_ST_CLR 配置是否清除
GDMA_EVT_IN_SUC_EOF_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH1_ST_CLR 配置是否清除 GDMA_EVT_IN_SUC_EOF_CH1
的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_SUC_EOF_CH2_ST_CLR 配置是否清除
GDMA_EVT_IN_SUC_EOF_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CHO_ST_CLR 配置是否清除
GDMA_EVT_IN_FIFO_EMPTY_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH1_ST_CLR 配置是否清除
GDMA_EVT_IN_FIFO_EMPTY_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_EMPTY_CH2_ST_CLR 配置是否清除
GDMA_EVT_IN_FIFO_EMPTY_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CHO_ST_CLR 配置是否清除
GDMA_EVT_IN_FIFO_FULL_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.22. SOC_ETM_EVT_ST3_CLR_REG (0x01C4)

接上页...

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH1_ST_CLR 配置 是否 清除
GDMA_EVT_IN_FIFO_FULL_CH1 的触发状态。
0: 无效值, 没有作用
1: 清除
(WT)

SOC_ETM_GDMA_EVT_IN_FIFO_FULL_CH2_ST_CLR 配置 是否 清除
GDMA_EVT_IN_FIFO_FULL_CH2 的触发状态。
0: 无效值, 没有作用
1: 清除
(WT)

Register 8.23. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

(reserved)

SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR
SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH2_ST_CLR
SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH1_ST_CLR
SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR
SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR
SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH2_ST_CLR
SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH1_ST_CLR
SOC_ETM_GDMA_EVT_OUT_EOF_CH2_ST_CLR
SOC_ETM_GDMA_EVT_OUT_EOF_CH1_ST_CLR
SOC_ETM_GDMA_EVT_OUT_DONE_CH2_ST_CLR
SOC_ETM_GDMA_EVT_OUT_DONE_CH1_ST_CLR

31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SOC_ETM_GDMA_EVT_OUT_DONE_CHO_ST_CLR 配置是否清除 GDMA_EVT_OUT_DONE_CHO 的触发状态。
0: 无效值, 没有作用
1: 清除 (WT)

SOC_ETM_GDMA_EVT_OUT_DONE_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_DONE_CH1 的触发状态。
0: 无效值, 没有作用
1: 清除 (WT)

SOC_ETM_GDMA_EVT_OUT_DONE_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_DONE_CH2 的触发状态。
0: 无效值, 没有作用
1: 清除 (WT)

SOC_ETM_GDMA_EVT_OUT_EOF_CHO_ST_CLR 配置是否清除 GDMA_EVT_OUT_EOF_CHO 的触发状态。
0: 无效值, 没有作用
1: 清除 (WT)

SOC_ETM_GDMA_EVT_OUT_EOF_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_EOF_CH1 的触发状态。
0: 无效值, 没有作用
1: 清除 (WT)

见下页...

Register 8.23. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

接上页...

SOC_ETM_GDMA_EVT_OUT_EOF_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_EOF_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CHO_ST_CLR 配置是否清除 GDMA_EVT_OUT_TOTAL_EOF_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_TOTAL_EOF_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_TOTAL_EOF_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_TOTAL_EOF_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CHO_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_EMPTY_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH1_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_EMPTY_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_EMPTY_CH2_ST_CLR 配置是否清除 GDMA_EVT_OUT_FIFO_EMPTY_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.23. SOC_ETM_EVT_ST4_CLR_REG (0x01CC)

[接上页...](#)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH0_ST_CLR 配置是否清除

GDMA_EVT_OUT_FIFO_FULL_CH0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH1_ST_CLR 配置是否清除

GDMA_EVT_OUT_FIFO_FULL_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_EVT_OUT_FIFO_FULL_CH2_ST_CLR 配置是否清除

GDMA_EVT_OUT_FIFO_FULL_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_PMU_EVT_SLEEP_WEEKUP_ST_CLR 配置是否清除 PMU_EVT_SLEEP_WEEKUP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.24. SOC_ETM_TASK_STO_CLR_REG (0x01D4)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR																																SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR																																SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR																																SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_CH0_ST_CLR																																SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR																																SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR																																SOC_ETM_GPIO_TASK_TIMER0_RES_UPDATE_ST_CLR																																SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR																																SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR																																SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR																																SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR																																																	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SOC_ETM_GPIO_TASK_CH0_SET_ST_CLR 配置是否清除 GPIO_TASK_CH0_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH1_SET_ST_CLR 配置是否清除 GPIO_TASK_CH1_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH2_SET_ST_CLR 配置是否清除 GPIO_TASK_CH2_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH3_SET_ST_CLR 配置是否清除 GPIO_TASK_CH3_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH4_SET_ST_CLR 配置是否清除 GPIO_TASK_CH4_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH5_SET_ST_CLR 配置是否清除 GPIO_TASK_CH5_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.24. SOC_ETM_TASK_STO_CLR_REG (0x01D4)

接上页...

SOC_ETM_GPIO_TASK_CH6_SET_ST_CLR 配置是否清除 GPIO_TASK_CH6_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH7_SET_ST_CLR 配置是否清除 GPIO_TASK_CH7_SET 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH0_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH0_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH1_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH1_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH2_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH2_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH3_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH3_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH4_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH4_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH5_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH5_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.24. SOC_ETM_TASK_STO_CLR_REG (0x01D4)

接上页...

SOC_ETM_GPIO_TASK_CH6_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH6_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH7_CLEAR_ST_CLR 配置是否清除 GPIO_TASK_CH7_CLEAR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH0_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH0_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH1_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH1_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH2_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH2_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH3_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH3_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH4_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH4_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH5_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH5_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.24. SOC_ETM_TASK_STO_CLR_REG (0x01D4)

接上页...

SOC_ETM_GPIO_TASK_CH6_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH6_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GPIO_TASK_CH7_TOGGLE_ST_CLR 配置是否清除 GPIO_TASK_CH7_TOGGLE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_RES_UPDATE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_RES_UPDATE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_RES_UPDATE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_RES_UPDATE_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_RES_UPDATE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CHO_ST_CLR 配置是否清除 LEDC_TASK_DUTY_SCALE_UPDATE_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.24. SOC_ETM_TASK_STO_CLR_REG (0x01D4)

接上页...

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH1_ST_CLR 配置 是否 清除

LEDC_TASK_DUTY_SCALE_UPDATE_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH2_ST_CLR 配置 是否 清除

LEDC_TASK_DUTY_SCALE_UPDATE_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH3_ST_CLR 配置 是否 清除

LEDC_TASK_DUTY_SCALE_UPDATE_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH0_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR																															
SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR																															
SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR																															
SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR																															
SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR																															
SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR																															
SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR																															
SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR																															
SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR																															
SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR																															
Reset																															

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH4_ST_CLR 配置是否清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_DUTY_SCALE_UPDATE_CH5_ST_CLR 配置是否清除 LEDC_TASK_DUTY_SCALE_UPDATE_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_CAP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_CAP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_CAP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_TIMER3_CAP_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_CAP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CHO_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH1_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH2_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH3_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH4_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_SIG_OUT_DIS_CH5_ST_CLR 配置是否清除 LEDC_TASK_SIG_OUT_DIS_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CHO_ST_CLR	配	置	是	否	清	除

LEDC_TASK_OVF_CNT_RST_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH1_ST_CLR	配	置	是	否	清	除

LEDC_TASK_OVF_CNT_RST_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH2_ST_CLR	配	置	是	否	清	除

LEDC_TASK_OVF_CNT_RST_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH3_ST_CLR	配	置	是	否	清	除

LEDC_TASK_OVF_CNT_RST_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH4_ST_CLR	配	置	是	否	清	除

LEDC_TASK_OVF_CNT_RST_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_OVF_CNT_RST_CH5_ST_CLR	配	置	是	否	清	除

LEDC_TASK_OVF_CNT_RST_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RST_ST_CLR	配置是否清除	LEDC_TASK_TIMER0_RST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_TIMER1_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_RST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_RST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_RST_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_RST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER0_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_RESUME 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_RESUME 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_RESUME 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_RESUME_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_RESUME 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.25. SOC_ETM_TASK_ST1_CLR_REG (0x01DC)

接上页...

SOC_ETM_LEDC_TASK_TIMER0_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER0_PAUSE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER1_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER1_PAUSE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER2_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER2_PAUSE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_TIMER3_PAUSE_ST_CLR 配置是否清除 LEDC_TASK_TIMER3_PAUSE 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CHO_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH1_ST_CLR 配置是否清除 LEDC_TASK_GAMMA_RESTART_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR																															
SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR																															
SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR																															
SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR																															
SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_START_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_ALARM_START_TIMER1_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_START_TIMER1_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_CAP_TIMER1_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER1_ST_CLR																															
SOC_ETM_TG0_TASK_ALARM_START_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_START_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_CAP_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_CNT_RELOAD_TIMER0_ST_CLR																															
SOC_ETM_TG0_TASK_ALARM_START_TIMER0_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH0_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR																															
SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR																															
Reset																															

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH2_ST_CLR 配置 是否 清除

LEDC_TASK_GAMMA_RESTART_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH3_ST_CLR 配置 是否 清除

LEDC_TASK_GAMMA_RESTART_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH4_ST_CLR 配置 是否 清除

LEDC_TASK_GAMMA_RESTART_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_LEDC_TASK_GAMMA_RESTART_CH5_ST_CLR 配置 是否 清除

LEDC_TASK_GAMMA_RESTART_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CHO_ST_CLR 配置 是否 清除

LEDC_TASK_GAMMA_PAUSE_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除 (WT)

见下页...

Register 8.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH1_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_PAUSE_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH2_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_PAUSE_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH3_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_PAUSE_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH4_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_PAUSE_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_PAUSE_CH5_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_PAUSE_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH0_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_RESUME_CH0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH1_ST_CLR	配	置	是	否	清	除

LEDC_TASK_GAMMA_RESUME_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH2_ST_CLR 配置 是 否 清除
LEDC_TASK_GAMMA_RESUME_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH3_ST_CLR 配置 是 否 清除
LEDC_TASK_GAMMA_RESUME_CH3 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH4_ST_CLR 配置 是 否 清除
LEDC_TASK_GAMMA_RESUME_CH4 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_LEDC_TASK_GAMMA_RESUME_CH5_ST_CLR 配置 是 否 清除
LEDC_TASK_GAMMA_RESUME_CH5 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_START_TIMER0_ST_CLR 配置 是 否 清除
TGO_TASK_CNT_START_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER0_ST_CLR 配置 是 否 清除
TGO_TASK_ALARM_START_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER0_ST_CLR 配置 是 否 清除
TGO_TASK_CNT_STOP_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

[接上页...](#)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER0_ST_CLR 配置是否清除 TGO_TASK_CNT_RELOAD_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_CAP_TIMER0_ST_CLR 配置是否清除 TGO_TASK_CNT_CAP_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_START_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_START_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_ALARM_START_TIMER1_ST_CLR 配置是否清除 TGO_TASK_ALARM_START_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_STOP_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_STOP_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_RELOAD_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_RELOAD_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TGO_TASK_CNT_CAP_TIMER1_ST_CLR 配置是否清除 TGO_TASK_CNT_CAP_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

[见下页...](#)

Register 8.26. SOC_ETM_TASK_ST2_CLR_REG (0x01E4)

接上页...

SOC_ETM_TG1_TASK_CNT_START_TIMER0_ST_CLR 配置是否清除
TG1_TASK_CNT_START_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_ALARM_START_TIMER0_ST_CLR 配置是否清除
TG1_TASK_ALARM_START_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER0_ST_CLR 配置是否清除 TG1_TASK_CNT_STOP_TIMER0
的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER0_ST_CLR 配置是否清除
TG1_TASK_CNT_RELOAD_TIMER0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER0_ST_CLR 配置是否清除 TG1_TASK_CNT_CAP_TIMER0 的
触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_START_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_START_TIMER1
的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																															
(reserved)																															
SOC_ETM_ADC_TASK_STOPO_ST_CLR																															
SOC_ETM_ADC_TASK_STARTO_ST_CLR																															
(reserved)																															
SOC_ETM_MCPWMO_TASK_SAMPLEO_ST_CLR																															
SOC_ETM_MCPWMO_TASK_CAP2_ST_CLR																															
SOC_ETM_MCPWMO_TASK_CAP1_ST_CLR																															
SOC_ETM_MCPWMO_TASK_CAPO_ST_CLR																															
SOC_ETM_MCPWMO_TASK CLR2_OST_ST_CLR																															
SOC_ETM_MCPWMO_TASK CLR1_OST_ST_CLR																															
SOC_ETM_MCPWMO_TASK CLR0_OST_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TZ2_OST_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TZ1_OST_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TZO_OST_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TIMER2_STOP_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TIMER1_PERIOD_UP_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TIMER0_PERIOD_UP_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TIMER2_SYN_ST_CLR																															
SOC_ETM_MCPWMO_TASK_TIMER0_SYN_ST_CLR																															
SOC_ETM_MCPWMO_TASK_GEN_STOP_ST_CLR																															
SOC_ETM_MCPWMO_TASK_CMPR2_B_UP_ST_CLR																															
SOC_ETM_MCPWMO_TASK_CMPR1_B_UP_ST_CLR																															
SOC_ETM_MCPWMO_TASK_CMPRO_B_UP_ST_CLR																															
SOC_ETM_TG1_TASK_CMPR2_A_UP_ST_CLR																															
SOC_ETM_TG1_TASK_CMPR1_A_UP_ST_CLR																															
SOC_ETM_TG1_TASK_CNT_CAP_CPRO_A_UP_ST_CLR																															
SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_STOP_ST_CLR																															
SOC_ETM_TG1_TASK_ALARM_START_TIMER1_STOP_ST_CLR																															

SOC_ETM_TG1_TASK_ALARM_START_TIMER1_ST_CLR 配置是否清除 TG1_TASK_ALARM_START_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_STOP_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_STOP_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_RELOAD_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_RELOAD_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_TG1_TASK_CNT_CAP_TIMER1_ST_CLR 配置是否清除 TG1_TASK_CNT_CAP_TIMER1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CMPRO_A_UP_ST_CLR 配置是否清除 MCPWMO_TASK_CMPRO_A_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_MCPWMO_TASK_CMPR1_A_UP_ST_CLR	配	置	是	否	清	除

MCPWMO_TASK_CMPR1_A_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CMPR2_A_UP_ST_CLR	配	置	是	否	清	除

MCPWMO_TASK_CMPR2_A_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CMPRO_B_UP_ST_CLR	配	置	是	否	清	除

MCPWMO_TASK_CMPRO_B_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CMPR1_B_UP_ST_CLR	配	置	是	否	清	除

MCPWMO_TASK_CMPR1_B_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CMPR2_B_UP_ST_CLR	配	置	是	否	清	除

MCPWMO_TASK_CMPR2_B_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_GEN_STOP_ST_CLR	配置是否清除 MCPWMO_TASK_GEN_STOP 的触

发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TIMER0_SYN_ST_CLR	配置是否清除 MCPWMO_TASK_TIMER0_SYN

的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_MCPWMO_TASK_TIMER1_SYN_ST_CLR 配置是否清除 MCPWMO_TASK_TIMER1_SYN 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TIMER2_SYN_ST_CLR 配置是否清除 MCPWMO_TASK_TIMER2_SYN 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TIMER0_PERIOD_UP_ST_CLR 配置是否清除 MCPWMO_TASK_TIMER0_PERIOD_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TIMER1_PERIOD_UP_ST_CLR 配置是否清除 MCPWMO_TASK_TIMER1_PERIOD_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TIMER2_PERIOD_UP_ST_CLR 配置是否清除 MCPWMO_TASK_TIMER2_PERIOD_UP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TZO_OST_ST_CLR 配置是否清除 MCPWMO_TASK_TZO_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_TZ1_OST_ST_CLR 配置是否清除 MCPWMO_TASK_TZ1_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

接上页...

SOC_ETM_MCPWMO_TASK_TZ2_OST_ST_CLR 配置是否清除 MCPWMO_TASK_TZ2_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CLR0_OST_ST_CLR 配置是否清除 MCPWMO_TASK_CLR0_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CLR1_OST_ST_CLR 配置是否清除 MCPWMO_TASK_CLR1_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CLR2_OST_ST_CLR 配置是否清除 MCPWMO_TASK_CLR2_OST 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CAPO_ST_CLR 配置是否清除 MCPWMO_TASK_CAPO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CAP1_ST_CLR 配置是否清除 MCPWMO_TASK_CAP1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_MCPWMO_TASK_CAP2_ST_CLR 配置是否清除 MCPWMO_TASK_CAP2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

见下页...

Register 8.27. SOC_ETM_TASK_ST3_CLR_REG (0x01EC)

[接上页...](#)

SOC_ETM_ADC_TASK_SAMPLE0_ST_CLR 配置是否清除 ADC_TASK_SAMPLE0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_TASK_START0_ST_CLR 配置是否清除 ADC_TASK_START0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ADC_TASK_STOP0_ST_CLR 配置是否清除 ADC_TASK_STOP0 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.28. SOC_ETM_TASK_ST4_CLR_REG (0x01F4)

(reserved)																															SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR SOC_ETM_GDMA_TASK_OUT_START_CH2_ST_CLR SOC_ETM_GDMA_TASK_OUT_START_CH1_ST_CLR SOC_ETM_GDMA_TASK_IN_START_CH0_ST_CLR SOC_ETM_GDMA_TASK_IN_START_CH2_ST_CLR SOC_ETM_GDMA_TASK_IN_START_CH1_ST_CLR SOC_ETM_RTC_TASK_TRIGGERFLW_ST_CLR SOC_ETM_RTC_TASK_CLR_ST_CLR (reserved) SOC_ETM_ULP_TASK_WAKEUP_OPU_ST_CLR SOC_ETM_I2SO_TASK_STOP_TX_ST_CLR SOC_ETM_I2SO_TASK_STOP_RX_ST_CLR SOC_ETM_I2SO_TASK_START_TX_ST_CLR SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR (reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																				21										20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SOC_ETM_TMPSNSR_TASK_START_SAMPLE_ST_CLR 配置是否清除 TMP-SNSR_TASK_START_SAMPLE 的触发状态。

0: 无效值，没有作用

1: 清除 (WT)

SOC_ETM_TMPSNSR_TASK_STOP_SAMPLE_ST_CLR 配置是否清除 TMP-SNSR_TASK_STOP_SAMPLE 的触发状态。

0: 无效值，没有作用

1: 清除 (WT)

SOC_ETM_I2SO_TASK_START_RX_ST_CLR 配置是否清除 I2SO_TASK_START_RX 的触发状态。

0: 无效值，没有作用

1: 清除 (WT)

SOC_ETM_I2SO_TASK_START_TX_ST_CLR 配置是否清除 I2SO_TASK_START_TX 的触发状态。

0: 无效值，没有作用

1: 清除 (WT)

见下页...

Register 8.28. SOC_ETM_TASK_ST4_CLR_REG (0x01F4)

[接上页...](#)

SOC_ETM_I2SO_TASK_STOP_RX_ST_CLR 配置是否清除 I2SO_TASK_STOP_RX 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_I2SO_TASK_STOP_TX_ST_CLR 配置是否清除 I2SO_TASK_STOP_TX 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_ULP_TASK_WAKEUP_CPU_ST_CLR 配置是否清除 ULP_TASK_WAKEUP_CPU 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_RTC_TASK_START_ST_CLR 配置是否清除 RTC_TASK_START 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_RTC_TASK_STOP_ST_CLR 配置是否清除 RTC_TASK_STOP 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_RTC_TASK_CLR_ST_CLR 配置是否清除 RTC_TASK_CLR 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

[见下页...](#)

Register 8.28. SOC_ETM_TASK_ST4_CLR_REG (0x01F4)

接上页...

SOC_ETM_RTC_TASK_TRIGGERFLW_ST_CLR 配置是否清除 RTC_TASK_TRIGGERFLW 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_TASK_IN_START_CHO_ST_CLR 配置是否清除 GDMA_TASK_IN_START_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_TASK_IN_START_CH1_ST_CLR 配置是否清除 GDMA_TASK_IN_START_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_TASK_IN_START_CH2_ST_CLR 配置是否清除 GDMA_TASK_IN_START_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_TASK_OUT_START_CHO_ST_CLR 配置是否清除 GDMA_TASK_OUT_START_CHO 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_TASK_OUT_START_CH1_ST_CLR 配置是否清除 GDMA_TASK_OUT_START_CH1 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_GDMA_TASK_OUT_START_CH2_ST_CLR 配置是否清除 GDMA_TASK_OUT_START_CH2 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

SOC_ETM_PMU_TASK_SLEEP_REQ_ST_CLR 配置是否清除 PMU_TASK_SLEEP_REQ 的触发状态。

0: 无效值, 没有作用

1: 清除

(WT)

Register 8.29. SOC_ETM_CLK_EN_REG (0x01F8)

(reserved)																																SOC_ETM_CLK_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SOC_ETM_CLK_EN 配置寄存器时钟门控。

0: 仅在应用写寄存器时开启时钟

1: 强制为寄存器开启时钟门控

(R/W)

Register 8.30. SOC_ETM_DATE_REG (0x01FC)

(reserved)				SOC_ETM_DATE																							27	28	31	0	
0	0	0	0	0x2311281																											Reset

SOC_ETM_DATE 版本控制寄存器。(R/W)

第 9 章

系统定时器

9.1 概述

ESP32-C5 芯片内置 52 位系统定时器。该定时器可用于生成操作系统所需的滴答定时中断，也可以用作普通的定时器生成周期或单次延时中断。

9.2 主要特性

系统定时器具有如下特性：

- 集成两个 52 位计数器和三个 52 位比较器
- 支持软件访问由 APB_CLK 时钟驱动的寄存器
- 支持 CNT_CLK 时钟计数，两次计数周期的平均频率为 16 MHz
- 配置 XTAL_CLK (40 MHz/48 MHz) 作为 CNT_CLK 时钟源
- 支持 52 位报警值 (t) 和 26 位报警周期 (δt)
- 支持两种报警模式：
 - 单次报警模式：根据设定的目标报警值 (t)，生成一次性报警
 - 周期报警模式：根据设定的报警周期 (δt)，生成周期性报警
- 支持根据设置的报警值 (t) 或报警周期 (δt)，通过三个比较器生成三个独立中断
- 支持软件配置基准计数值。例如，支持从 Light-sleep 唤醒之后，系统定时器通过软件加载 RTC 定时器记录的睡眠时间，并进行补偿
- CPU 处于停止状态或处于在线调试状态时，系统定时器可选择停止运行或继续运行
- 支持事件任务矩阵 (ETM) 事件报警

9.3 系统定时器架构

系统定时器内置两个计数器（UNIT0 和 UNIT1）以及三个比较器（COMP0、COMP1 和 COMP2）。比较器用于监控计数器的计数值是否达到报警值。定时器的功能块图见图 9.3-1。

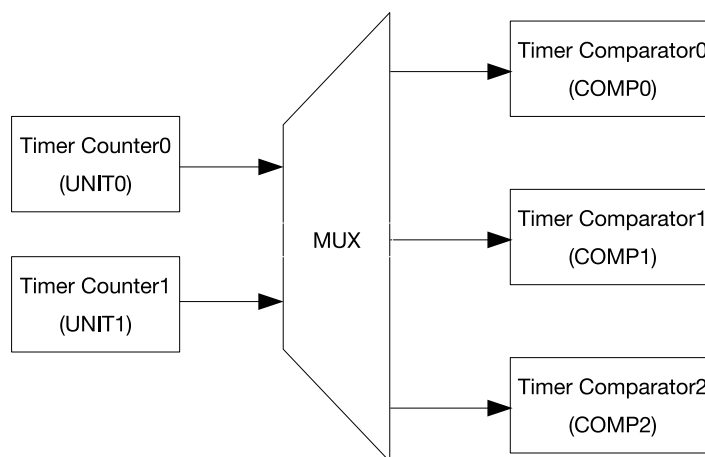


图 9.3-1. 系统定时器结构图

9.4 时钟源选择

计数器和比较器使用 XTAL_CLK 或者 RC_FAST_CLK 作为时钟源。可通过配置寄存器 `PCR_SYSTIMER_FUNC_CLK_CONF_REG` 的 `PCR_SYSTIMER_FUNC_CLK_SEL` 字段来选择时钟源。XTAL_CLK 经分频后，生成平均频率为 16 MHz 的计数器时钟信号 CNT_CLK，如图 9.5-1 所示。每个 CNT_CLK 时钟周期，计数递增 $1/16 \mu s$ ，即 16 个周期递增 $1 \mu s$ 。

APB_CLK 为配置寄存器等软件操作提供时钟信号。更多有关 APB_CLK 的信息，见章节 6 复位和时钟。

用户可使用系统寄存器的两个相关位来控制系统定时器，具体如下：

- 置位寄存器 `PCR_SYSTIMER_CONF_REG` 中 `PCR_SYSTIMER_CLK_EN` 位，使能系统定时器的 APB_CLK 信号。
- 置位寄存器 `PCR_SYSTIMER_CONF_REG` 中 `PCR_SYSTIMER_RST_EN` 位，复位系统定时器。

注意，复位后，系统定时器的寄存器将恢复到默认值。更多信息可参考章节 6 复位和时钟。

9.5 功能描述

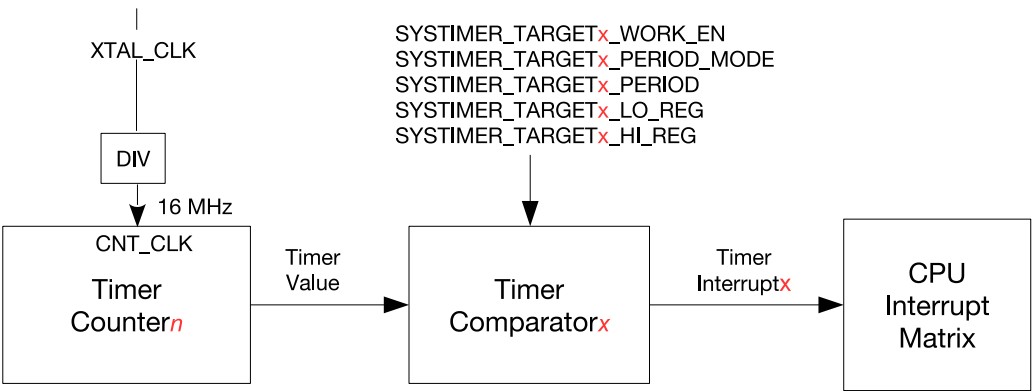


图 9.5-1. 系统定时器生成报警

图 9.5-1 展示了系统定时器生成报警的过程。过程中需要用到一个计数器和一个比较器，比较器将根据比较结果，生成报警中断。

9.5.1 计数器

系统定时器集成两个 52 位计数器，用 UNIT n 表示， n 可以取 0 或 1。计数器使用 16 MHz CNT_CLK 作为计数时钟。用户可通过配置寄存器 SYSTIMER_CONF_REG 中的下面三个位来控制计数器 UNIT n ，具体如下：

- SYSTIMER_TIMER_UNIT n _WORK_EN：置位此位，使能计数器 UNIT n 。
- SYSTIMER_TIMER_UNIT n _CORE0_STALL_EN：置位此位，CPU0 停止运行后，计数器 UNIT n 也将停止工作。CPU 恢复运行后，计数器重新开始工作。
- SYSTIMER_TIMER_UNIT n _CORE1_STALL_EN：置位此位，CPU1 停止运行后，计数器 UNIT n 也将停止工作。CPU 恢复运行后，计数器重新开始工作。

假设 CPU 当前状态为停止工作，UNIT n 的具体配置见下表：

表 9.5-1. UNIT n 配置控制位

SYSTIMER_TIMER_UNIT n _WORK_EN	SYSTIMER_TIMER_UNIT n _CORE0_STALL_EN	SYSTIMER_TIMER_UNIT n _CORE1_STALL_EN	计数器 UNIT n
0	x [*]	x [*]	未处于工作状态。
1	x	1	暂停计数，但在 CPU core1 恢复运行后继续计数。
1	1	x	暂停计数，但在 CPU core0 恢复运行后继续计数。
1	0	0	不受影响，照常计数。

^{*} x：无关项

计数器 UNIT n 处于工作状态时，计数值按计数周期递增。UNIT n 停止工作时，计数值将保持不变，不再递增。

计数起始值的低 32 位和高 20 位分别从 SYSTIMER_TIMER_UNIT n _LOAD_LO 和 SYSTIMER_TIMER_

UNIT n _LOAD_HI 中装载。置位 SYSTIMER_TIMER_UNIT n _LOAD 将触发重装载事件，当前计数起始值立即更新。如果计数器 UNIT n 处于工作状态，则从新装载的计数值开始计数。

置位 SYSTIMER_TIMER_UNIT n _UPDATE 将触发更新事件。当前计数值的低 32 位和高 20 位被锁存至寄存器 SYSTIMER_TIMER_UNIT n _VALUE_LO 和 SYSTIMER_TIMER_UNIT n _VALUE_HI 后，SYSTIMER_TIMER_UNIT n _VALUE_VALID 会被置起。在下次更新事件发生前，SYSTIMER_TIMER_UNIT n _VALUE_LO 和 SYSTIMER_TIMER_UNIT n _VALUE_HI 寄存器中的值保持不变。

9.5.2 比较器和报警

系统定时器有三个 52 位比较器，用 COMP x 表示，其中 x 可以取 0、1、2。比较器可根据设置的不同报警值 (t) 或报警周期 (δt)，触发不同的中断。

用户可配置寄存器 SYSTIMER_TARGET x _PERIOD_MODE，选择比较器 COMP x 生成报警的模式：

- 1：周期报警模式
- 0：单次报警模式

选择周期报警模式时，寄存器 SYSTIMER_TARGET x _PERIOD 中的值为报警周期 (δt)。假设当前计数值为 t_1 ，经过一段时间，当计数值达到 $t_1 + \delta t$ 时，将触发一次报警中断。再经过一段时间，当计数值达到 $t_1 + 2 \times \delta t$ 时，将再一次触发报警中断，以此类推。通过上述方式即可实现周期性报警。

选择单次报警模式时，SYSTIMER_TIMER_TARGET x _LO 和 SYSTIMER_TIMER_TARGET x _HI 分别提供报警值 (t) 的低 32 位和高 20 位。假设当前计数值为 t_2 ($t_2 \leq t$)，经过一段时间，当计数到报警值 (t) 时，触发一次报警。与周期报警模式不同，单次报警模式仅生成一次报警中断。

用户可通过寄存器 SYSTIMER_TARGET x _TIMER_UNIT_SEL 选择用于与 COMP x 进行比较的计数器值，然后生成报警：

- 1：与计数器 UNIT 1 的计数值进行比较
- 0：与计数器 UNIT 0 的计数值进行比较

置位 SYSTIMER_TARGET x _WORK_EN，COMP x 开始进行比较：

- 在单次报警模式下，COMP x 将比较计数器中的实际计数值与寄存器中设置的报警值 (t)；
- 在周期报警模式下，COMP x 将比较计数器中的实际计数值与 $t_1 + n \times \delta t$ ($n = 1, 2, 3, \dots$)。

在单次报警模式下，当实际计数值等于报警值 (t)，或在周期报警模式下，实际计数值等于 $t_1 + n \times \delta t$ ($n = 1, 2, 3, \dots$) 时，将触发一次报警中断。但如果设定的报警值 (t) 小于当前计数值，即报警值 (t) 已成为过去，在当前计数值超过设定的报警值 (t) 一定范围 ($0 \sim 2^{51} - 1$) 时，也将立即触发中断。无论是在单次报警模式还是在周期报警模式下，都可以从 SYSTIMER_TARGET x _LO_RO 和 SYSTIMER_TARGET x _HI_RO 中读取实际报警值的低 32 位和高 20 位。当前计数值 t_c 、报警值 t_t 和触发报警的关系如下表所示：

表 9.5-2. 报警触发条件

t_c 与 t_t 的关系	触发条件
$t_c - t_t \leq 0$	当 $t_c = t_t$ 时，触发报警
$0 \leq t_c - t_t < 2^{51} - 1$ ($t_c < 2^{51}$ 且 $t_t < 2^{51}$; 或 $t_c \geq 2^{51}$ 且 $t_t \geq 2^{51}$)	立即触发报警

$t_c - t_t \geq 2^{51} - 1$	t_c 达到最大值 52'hffffffffffff 后溢出，然后从 0 开始计数，计数达到 t_t 时触发报警

9.5.3 事件任务矩阵

在 ESP32-C5 中，系统定时器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发系统定时器的 ETM 任务，或者通过系统定时器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与系统定时器相关的 ETM 任务和 ETM 事件。

系统定时器可产生的 ETM 事件有：

- SYSTIMER_EVT_CNT_CMPx: 表示 COMPx 产生的报警事件。

当 SYSTIMER_ETM_EN 置 1 时，对应的报警事件会传到 ETM 模块。

9.5.4 同步操作

计数器与比较器的工作时钟和软件的配置时钟频率并不相同，因此在配置部分寄存器时，必须进行同步。完整的同步过程包括下面两个步骤：

1. 通过软件向配置寄存器字段写入合适的值，见表 9.5-3 第一列。
2. 通过软件置位相应的同步使能位，开始同步操作，见表 9.5-3 第二列。

表 9.5-3. 同步操作

需要同步的字段	同步使能位
SYSTIMER_TIMER_UNITn_LOAD_LO SYSTIMER_TIMER_UNITn_LOAD_HI	SYSTIMER_TIMER_UNITn_LOAD
SYSTIMER_TARGETx_PERIOD SYSTIMER_TIMER_TARGETx_HI SYSTIMER_TIMER_TARGETx_LO	SYSTIMER_TIMER_COMPx_LOAD

计数器与比较器的工作时钟和软件的读取时钟频率并不相同，因此在读取部分寄存器时，也需要进行同步。完整的同步步骤如下：

1. 软件将 1 写入更新寄存器 SYSTIMER_TIMER_UNITn_UPDATE 中。
2. 软件读取相应位 SYSTIMER_TIMER_UNITn_VALUE_VALID 为有效，即表示同步完成。
3. 软件读取相应的状态寄存器 SYSTIMER_TIMER_UNITn_VALUE_HI 和 SYSTIMER_TIMER_UNITn_VALUE_LO。

9.6 中断

ESP32-C5 的系统定时器可以生成以下中断信号并发送给 中断矩阵。

- SYSTIMER_TARGET0_INT
- SYSTIMER_TARGET1_INT

- SYSTIMER_TARGET2_INT

以上中断信号由系统定时器的内部中断源生成。表 9.6-1 列出了系统定时器的内部中断源、中断触发条件以及生成的中断信号。

表 9.6-1. 系统定时器的内部中断源

内部中断源	触发条件	中断信号
SYSTIMER_INT0	比较器 COMPO 报警时	SYSTIMER_TARGET0_INT
SYSTIMER_INT1	比较器 COMP1 报警时	SYSTIMER_TARGET1_INT
SYSTIMER_INT2	比较器 COMP2 报警时	SYSTIMER_TARGET2_INT

以上中断为电平类型中断。比较器开始报警时，中断信号拉高。中断信号将一直保持高电平，直至软件清除中断。用户可置位 SYSTIMER_TARGETx_INT_ENA 使能中断。

说明：
中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 [to be added later] > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 9.8 寄存器列表。

9.7 编程示例

注意，在配置 COMPx 和 UNITn 过程中，需保证对应的 COMP 和 UNIT 处于工作状态。

9.7.1 读取当前计数器的值

1. 置位 SYSTIMER_TIMER_UNITn_UPDATE，将计数器 UNITn 的值更新至寄存器 SYSTIMER_TIMER_UNITn_VALUE_HI 和 SYSTIMER_TIMER_UNITn_VALUE_LO；
2. 轮询 SYSTIMER_TIMER_UNITn_VALUE_VALID，直至其值为 1。之后，用户可从寄存器 SYSTIMER_TIMER_UNITn_VALUE_HI 和 SYSTIMER_TIMER_UNITn_VALUE_LO 中读取计数器的值；
3. 读取寄存器 SYSTIMER_TIMER_UNITn_VALUE_LO（低 32 位）和 SYSTIMER_TIMER_UNITn_VALUE_HI（高 20 位）。

9.7.2 在单次报警模式下配置一次性报警

1. 设置 SYSTIMER_TARGETx_TIMER_UNIT_SEL 选择与 COMPx 进行比较的计数器（UNIT0 或 UNIT1）。
2. 读取当前计数器的值，步骤见章节 9.7.1。读取的当前值可用于计算步骤 4 中的报警值 (t)。
3. 清除 SYSTIMER_TARGETx_PERIOD_MODE，使能单次报警模式。
4. 设置报警值 (t)，并将报警值 (t) 的低 32 位和高 20 位分别写入 SYSTIMER_TIMER_TARGETx_LO 和 SYSTIMER_TIMER_TARGETx_HI。
5. 置位 SYSTIMER_TIMER_COMPx_LOAD，同步报警值 (t)，即将报警值 (t) 装载至比较器 COMPx。
6. 置位 SYSTIMER_TARGETx_WORK_EN 使能选择的比较器 COMPx；比较器 COMPx 开始比较计数值与报警值 (t)。

- 置位 `SYSTIMER_TARGET $x$ _INT_ENA`，使能中断。UNIT n 达到报警值 (t) 时，触发一次报警中断 `SYSTIMER_TARGET $x$ _INT`。

9.7.3 在周期报警模式下配置周期性报警

- 设置 `SYSTIMER_TARGET $x$ _TIMER_UNIT_SEL` 选择与 `COMP $x$` 进行比较的计数器。
- 将报警周期 (δt) 写入 `SYSTIMER_TARGET $x$ _PERIOD`。
- 置位 `SYSTIMER_TIMER_COMP $x$ _LOAD` 同步报警周期值，即将 (δt) 装载至比较器 `COMP $x$` 。
- 先清除再置位 `SYSTIMER_TARGET $x$ _PERIOD_MODE` 将 `COMP $x$` 配置为周期报警模式。
- 置位 `SYSTIMER_TARGET $x$ _WORK_EN` 使能选择的比较器 `COMP $x$` ；比较器 `COMP $x$` 开始将计数值与计数初始值 + $n \times \delta t$ ($n = 1, 2, 3 \dots$) 进行比较。
- 置位 `SYSTIMER_TARGET $x$ _INT_ENA`，使能中断。UNIT n 计数达到计数初始值 + $n \times \delta t$ ($n = 1, 2, 3 \dots$) 时，触发一次 `SYSTIMER_TARGET $x$ _INT` 中断。

9.7.4 唤醒后时间补偿

- 在芯片进入 Light-sleep 模式之前，用户需配置 RTC 定时器，精确记录睡眠时间，见章节 4 [低功耗管理 \[to be added later\]](#)。
- 从 Light-sleep 模式唤醒后，读取 RTC 定时器记录的睡眠时间。
- 读取当前系统定时器的计数值，见章节 9.7.1。
- 将 RTC 记录的以 `RTC_SLOW_CLK` 周期为单位的睡眠时间，转换成以 `CNT_CLK` (16 MHz) 周期为单位的睡眠时间。例如，如果 `RTC_SLOW_CLK` 频率为 32 kHz，则 RTC 定时器记录的时间乘以 500。
- 将 RTC 定时器转换后的值加到系统定时器的当前计数值：
 - 将计算所得值的低 32 位写入 `SYSTIMER_TIMER_UNIT $n$ _LOAD_LO`，高 20 位写入 `SYSTIMER_TIMER_UNIT $n$ _LOAD_HI`。
 - 置位 `SYSTIMER_TIMER_UNIT $n$ _LOAD`，将新的定时器值装载到系统定时器。至此，系统定时器更新完成。

9.8 寄存器列表

本小节的所有地址均为相对于系统定时器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时钟控制寄存器			
SYSTIMER_CONF_REG	配置系统定时器时钟	0x0000	R/W
UNIT0 控制和配置寄存器			
SYSTIMER_UNIT0_OP_REG	读取 UNTO 的值到相应寄存器	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	待装载至 UNIT0 的值，高 20 位	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	待装载至 UNIT0 的值，低 32 位	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 的读值，高 20 位	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 的读值，低 32 位	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	计数器 UNIT0 的装载同步寄存器	0x005C	WT
UNIT1 控制和配置寄存器			
SYSTIMER_UNIT1_OP_REG	读取计数器 UNIT1 的值	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	待装载至计数器 UNIT1 的值，高 20 位	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	待装载至计数器 UNIT1 的值，低 32 位	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	计数器 UNIT1 的读值，高 20 位	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	计数器 UNIT1 的读值，低 32 位	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	计数器 UNIT1 的装载同步寄存器	0x0060	WT
比较器 COMP0 的控制和配置寄存器			
SYSTIMER_TARGET0_HI_REG	待装载至比较器 COMP0 的报警值，高 20 位	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	待装载至比较器 COMP0 的报警值，低 32 位	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	配置比较器 COMP0 的报警模式	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	比较器 COMP0 的装载同步寄存器	0x0050	WT
比较器 COMP1 的控制和配置寄存器			
SYSTIMER_TARGET1_HI_REG	待装载至比较器 COMP1 的报警值，高 20 位	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	待装载至比较器 COMP1 的报警值，低 32 位	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	配置比较器 COMP1 的报警模式	0x0038	R/W
SYSTIMER_COMP1_LOAD_REG	比较器 COMP1 的装载同步寄存器	0x0054	WT
比较器 COMP2 的控制和配置寄存器			
SYSTIMER_TARGET2_HI_REG	待装载至比较器 COMP2 的报警值，高 20 位	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	待装载至比较器 COMP2 的报警值，低 32 位	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	配置比较器 COMP2 的报警模式	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	比较器 COMP2 的装载同步寄存器	0x0058	WT
中断寄存器			
SYSTIMER_INT_ENA_REG	系统定时器的中断使能寄存器	0x0064	R/W
SYSTIMER_INT_RAW_REG	系统定时器的原始中断寄存器	0x0068	R/ WTC/ SS
SYSTIMER_INT_CLR_REG	系统定时器的中断清除寄存器	0x006C	WT

名称	描述	地址	访问
SYSTIMER_INT_ST_REG	系统定时器的中断状态寄存器	0x0070	RO
COMP0 状态寄存器			
SYSTIMER_REAL_TARGET0_LO_REG	COMP0 的实际报警值，低 32 位	0x0074	RO
SYSTIMER_REAL_TARGET0_HI_REG	COMP0 的实际报警值，高 20 位	0x0078	RO
COMP1 状态寄存器			
SYSTIMER_REAL_TARGET1_LO_REG	COMP1 的实际报警值，低 32 位	0x007C	RO
SYSTIMER_REAL_TARGET1_HI_REG	COMP1 的实际报警值，高 20 位	0x0080	RO
COMP2 状态寄存器			
SYSTIMER_REAL_TARGET2_LO_REG	COMP2 的实际报警值，低 32 位	0x0084	RO
SYSTIMER_REAL_TARGET2_HI_REG	COMP2 的实际报警值，高 20 位	0x0088	RO
版本寄存器			
SYSTIMER_DATE_REG	版本控制寄存器	0x00FC	R/W

9.9 寄存器

本小节的所有地址均为相对于系统定时器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

Register 9.1. SYSTIMER_CONF_REG (0x0000)

SYSTIMER_CLK_EN										(reserved)										SYSTIMER_ETM_EN								
SYSTIMER_TIMER_UNIT0_WORK_EN										(reserved)										(reserved)								
SYSTIMER_TIMER_UNIT1_WORK_EN										(reserved)										(reserved)								
SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN										(reserved)										(reserved)								
SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN										(reserved)										(reserved)								
SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN										(reserved)										(reserved)								
SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN										(reserved)										(reserved)								
SYSTIMER_TARGET0_WORK_EN										(reserved)										(reserved)								
SYSTIMER_TARGET1_WORK_EN										(reserved)										(reserved)								
SYSTIMER_TARGET2_WORK_EN										(reserved)										(reserved)								
31	30	29	28	27	26	25	24	23	22	21											2	1	0					
0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																												

SYSTIMER_ETM_EN 配置是否使能生成 ETM 事件。

0: 无效

1: 使能生成 ETM 事件

(R/W)

SYSTIMER_TARGET2_WORK_EN 配置是否使能 COMP2。

0: 不使能

1: 使能 COMP2

(R/W)

SYSTIMER_TARGET1_WORK_EN 配置是否使能 COMP1。具体配置值请见 [SYSTIMER_TARGET2_WORK_EN](#)。(R/W)

SYSTIMER_TARGET0_WORK_EN 配置是否使能 COMP0。具体配置值请见 [SYSTIMER_TARGET2_WORK_EN](#)。(R/W)

SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT1 是否也将停止工作。

0: 无效

1: 计数器 UNIT1 停止工作

(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT1 是否也将停止工作。具体配置值请见 [SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN](#)。(R/W)

SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 [SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN](#)。(R/W)

SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 [SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN](#)。(R/W)

见下页...

Register 9.1. SYSTIMER_CONF_REG (0x0000)

... 接上页

SYSTIMER_TIMER_UNIT1_WORK_EN 配置是否使能计数器 UNIT1。

0: 无效

1: 使能计数器 UNIT1

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN 配置是否使能计数器 UNIT0。

0: 无效

1: 使能计数器 UNITO

(R/W)

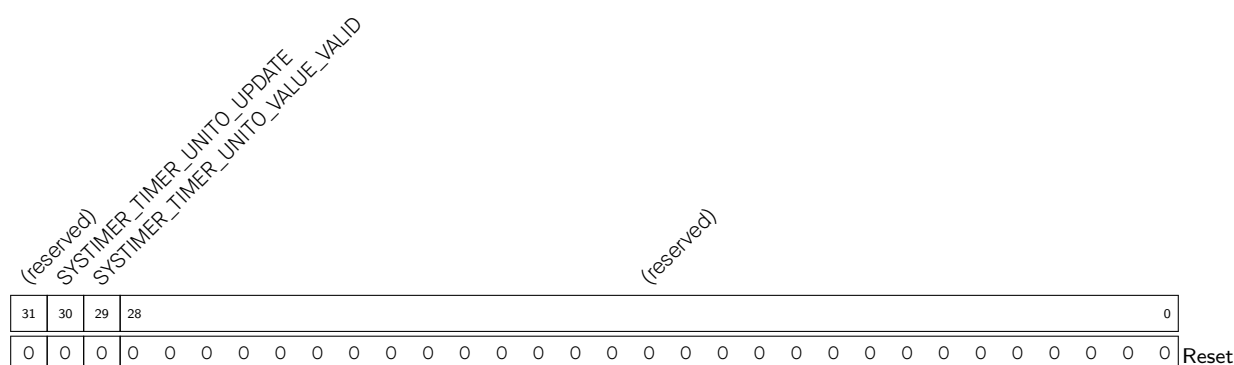
SYSTIMER CLK EN 配置寄存器时钟门控。

1: 持续开启读写寄存器的时钟。

0: 只在读写寄存器时打开所需时钟。

(R/W)

Register 9.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID 表示计数器 UNIT0 的值是否已同步读取至寄存器，且读值有效。

0: UNIT0 的值未同步读取至寄存器, 读值无效

1: UNIT0 的值已同步读取至寄存器，读值有效

(R/SS/WTC)

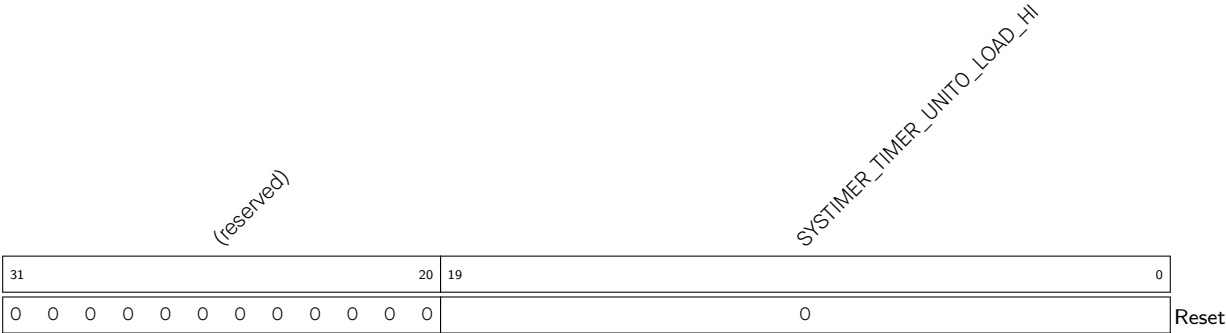
SYSTIMER_TIMER_UNIT0_UPDATE 配置是否更新计数器 UNIT0，即读取计数器 UNIT0 的值到寄存器 **SYSTIMER_TIMER_UNIT0_VALUE_HI** 和 **SYSTIMER_TIMER_UNIT0_VALUE_LO**。

0: 无效

1: 更新计数器 UNIT0

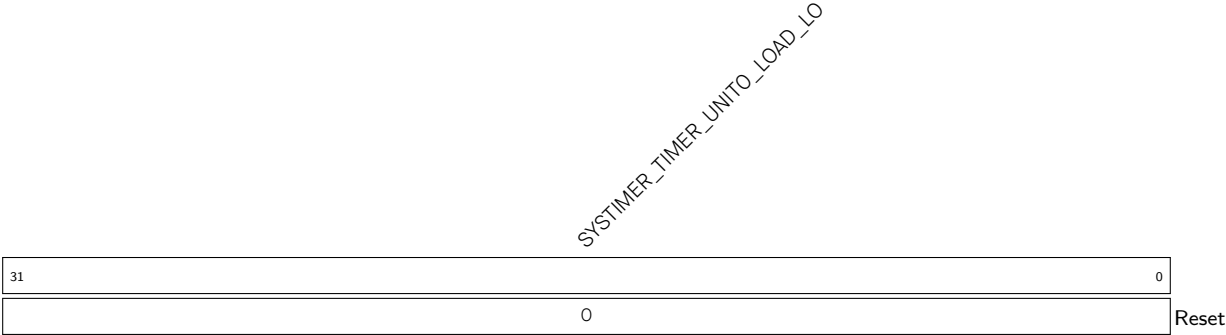
(WT)

Register 9.3. SYSTIMER_UNITO_LOAD_HI_REG (0x000C)



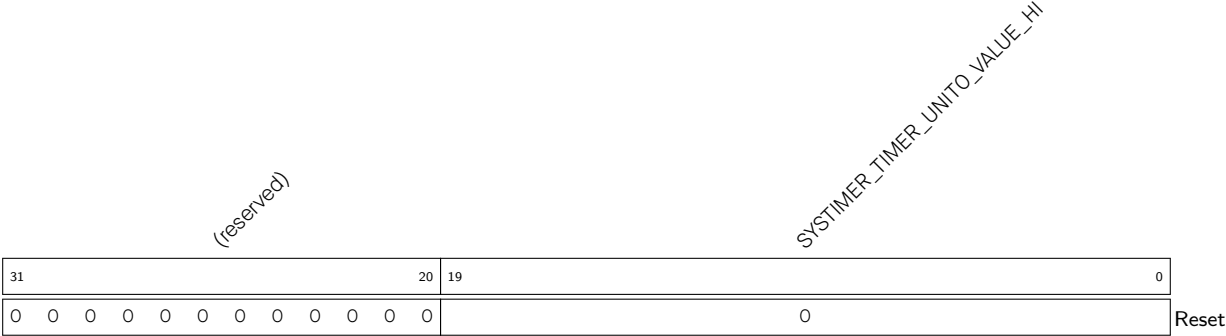
SYSTIMER_TIMER_UNITO_LOAD_HI 配置待装载至计数器 UNITO 的值，高 20 位。(R/W)

Register 9.4. SYSTIMER_UNITO_LOAD_LO_REG (0x0010)



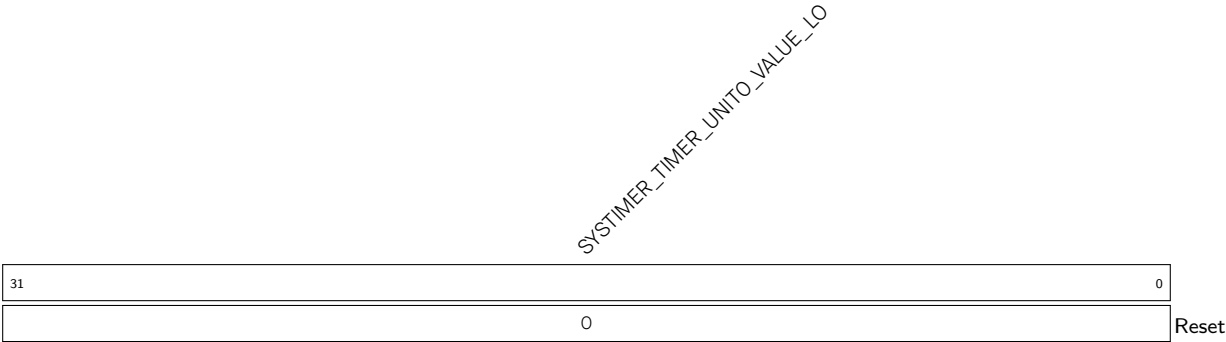
SYSTIMER_TIMER_UNITO_LOAD_LO 配置待装载至计数器 UNITO 的值，低 32 位。(R/W)

Register 9.5. SYSTIMER_UNITO_VALUE_HI_REG (0x0040)



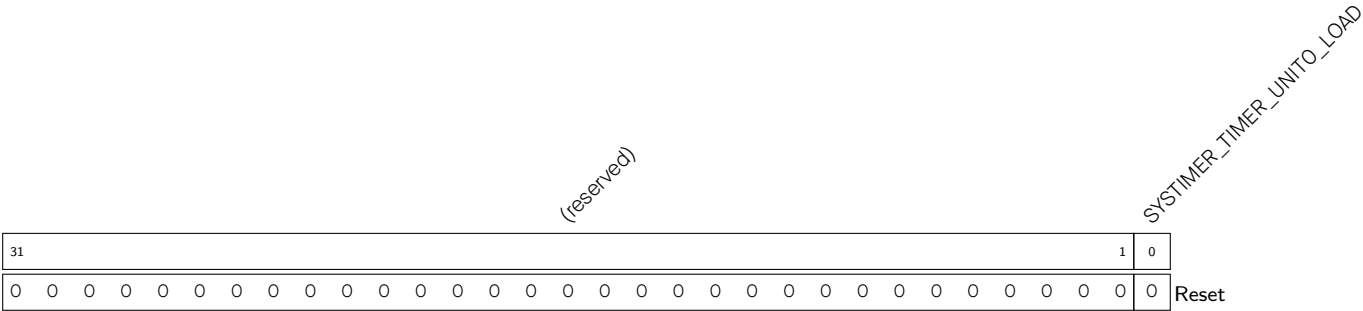
SYSTIMER_TIMER_UNITO_VALUE_HI 表示计数器 UNITO 的读数，高 20 位。(RO)

Register 9.6. SYSTIMER_UNITO_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNITO_VALUE_LO 表示计数器 UNITO 的读数，低 32 位。(RO)

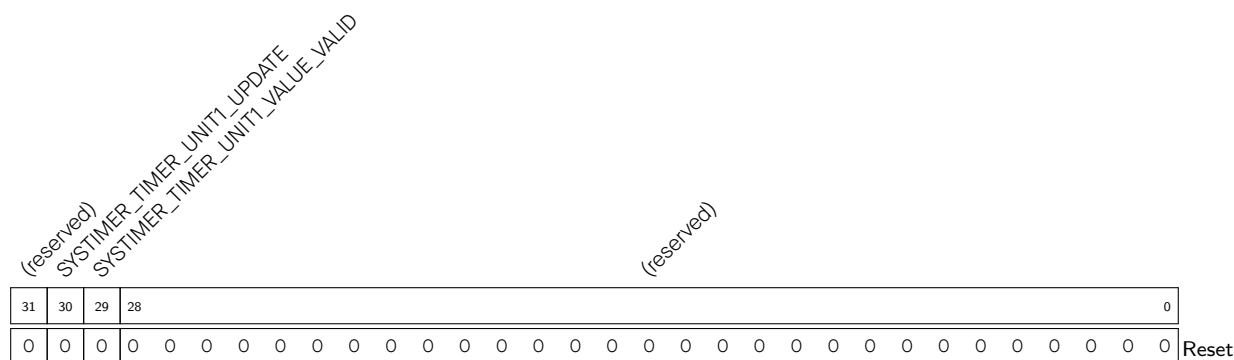
Register 9.7. SYSTIMER_UNITO_LOAD_REG (0x005C)



SYSTIMER_TIMER_UNITO_LOAD 配置是否重新装载计数器 UNITO，即是否重新装载寄存器 SYSTIMER_TIMER_UNITO_LOAD_HI 和 SYSTIMER_TIMER_UNITO_LOAD_LO 的值到计数器 UNITO。

- 0: 无效
- 1: 重新装载计数器 UNITO (WT)

Register 9.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID 表示计数器 UNIT1 的值已同步读取至寄存器，且读值有效。
(R/SS/WTC)

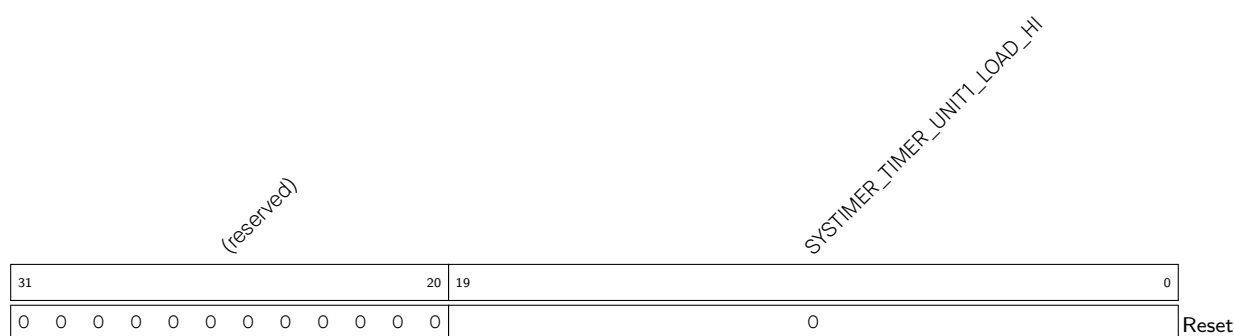
SYSTIMER_TIMER_UNIT1_UPDATE 配置是否更新计数器 UNIT1，即将计数器 UNIT1 的值读取到寄存器 **SYSTIMER_TIMER_UNIT1_VALUE_HI** 和 **SYSTIMER_TIMER_UNIT1_VALUE_LO**。

0: 无效

1: 更新计数器 UNIT1

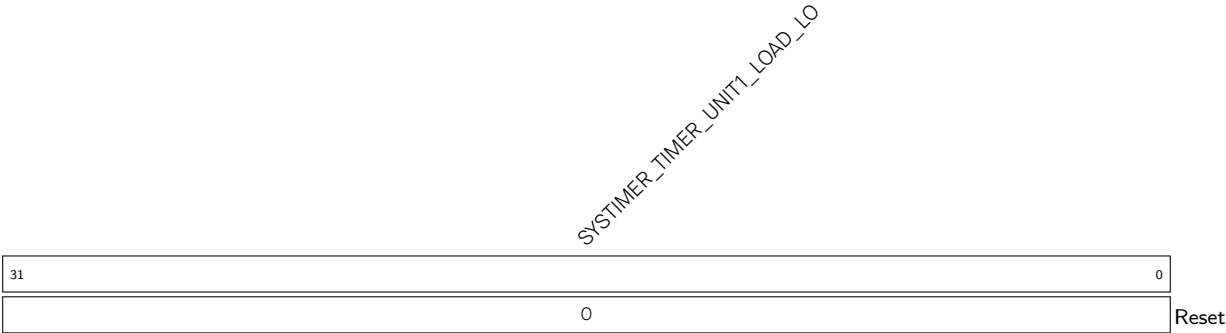
(WT)

Register 9.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



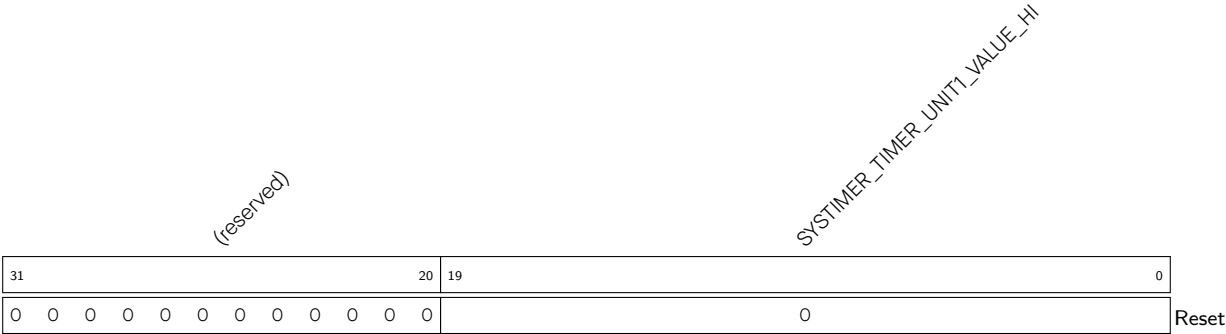
SYSTIMER_TIMER_UNIT1_LOAD_HI 配置待装载至计数器 UNIT1 的值，高 20 位。(R/W)

Register 9.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



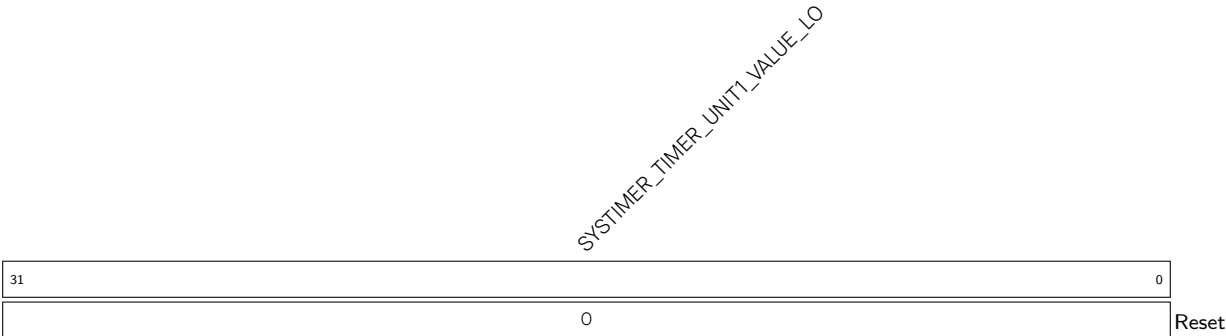
SYSTIMER_TIMER_UNIT1_LOAD_LO 配置待装载至计数器 UNIT1 的值，低 32 位。(R/W)

Register 9.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



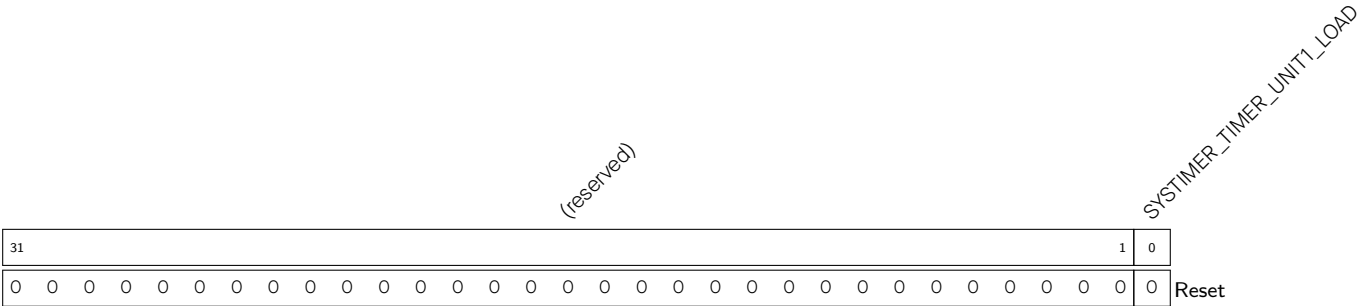
SYSTIMER_TIMER_UNIT1_VALUE_HI 表示计数器 UNIT1 的读数，高 20 位。(RO)

Register 9.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



SYSTIMER_TIMER_UNIT1_VALUE_LO 表示计数器 UNIT1 的读数，低 32 位。(RO)

Register 9.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)

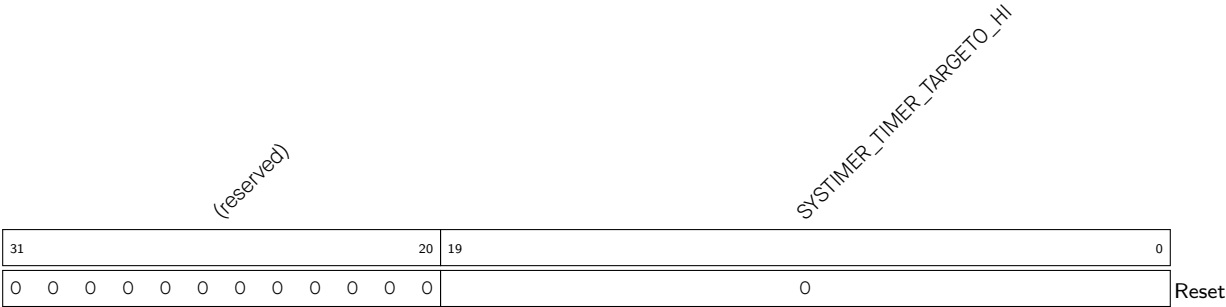


SYSTIMER_TIMER_UNIT1_LOAD 配置是否重新装载计数器 UNIT1，即是否重新装载寄存器 **SYSTIMER_TIMER_UNIT1_LOAD_HI** 和 **SYSTIMER_TIMER_UNIT1_LOAD_LO** 的值到计数器 UNIT1。

0: 无效

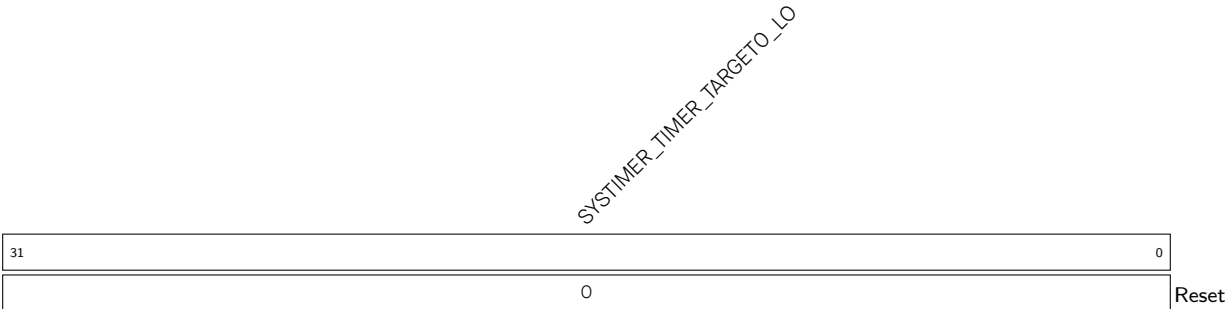
1: 重新装载计数器 UNIT1 (WT)

Register 9.14. SYSTIMER_TARGETO_HI_REG (0x001C)



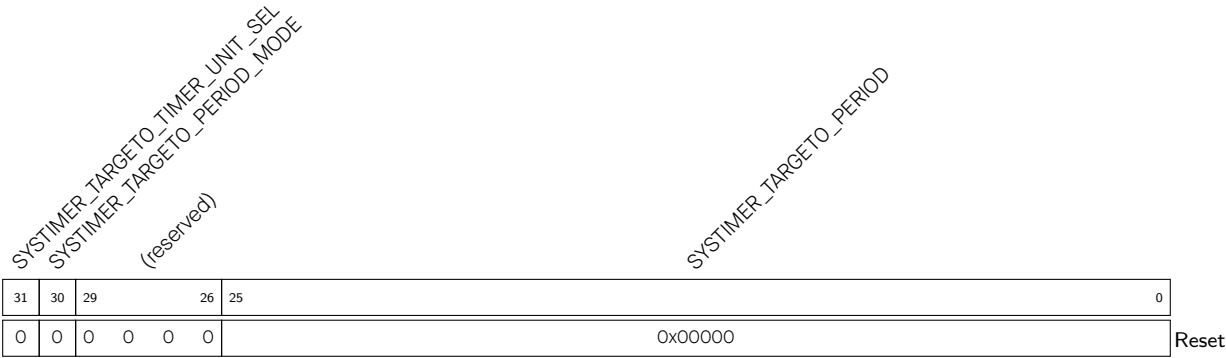
SYSTIMER_TIMER_TARGETO_HI 配置待装载至 COMPO 的报警值，高 20 位。(R/W)

Register 9.15. SYSTIMER_TARGETO_LO_REG (0x0020)



SYSTIMER_TIMER_TARGETO_LO 配置待装载至 COMPO 的报警值，低 32 位。(R/W)

Register 9.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



- SYSTIMER_TARGET0_PERIOD

配置待装载至 COMPO 的报警周期。(R/W)
- SYSTIMER_TARGET0_PERIOD_MODE

选择比较器 COMPO 生成报警的模式。

0：单次报警模式

1：周期报警模式

(R/W)
- SYSTIMER_TARGET0_TIMER_UNIT_SEL

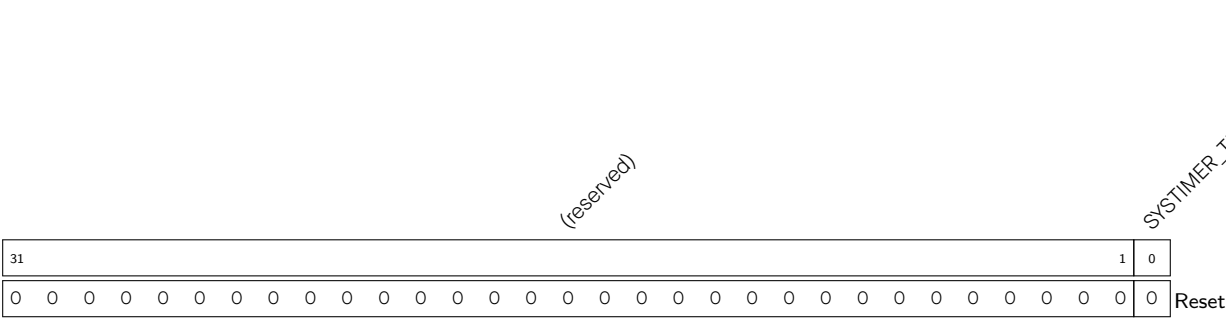
选择要与 COMPO 比较的计数器。

0：与计数器 UNIT0 的计数值进行比较

1：与计数器 UNIT1 的计数值进行比较

(R/W)

Register 9.17. SYSTIMER_COMPO_LOAD_REG (0x0050)



- SYSTIMER_TIMER_COMPO_LOAD

配置是否使能比较器 COMPO 同步，即是否重新装载报警值或报警周期到 COMPO。

0：无效

1：使能比较器 COMPO 同步

(WT)

Register 9.18. SYSTIMER_TARGET1_HI_REG (0x0024)

Diagram illustrating the structure of the SYSTIMER_TARGET1 register (32 bits):

- Bits 31 to 20: (reserved)
- Bits 19 to 0: SYSTIMER_TARGET1_HI
- Reset value: 0

SYSTIMER_TIMER_TARGET1_HI 配置待装载至 COMP1 的报警值，高 20 位。(R/W)

Register 9.19. SYSTIMER_TARGET1_LO_REG (0x0028)

Diagram of the SYSTIMER_TARGET1_LO register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. Bit 0 is labeled "Reset".

SYSTIMER_TIMER_TARGET1_LO 配置待装载至 COMP1 的报警值，低 32 位。(R/W)

Register 9.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

Diagram illustrating the structure of the SYSTIMER_TARGET1 register. The register is 32 bits wide. The fields are defined as follows:

- Bits 31-29: SYSTIMER_TARGET1_TIMER_UNIT_SEL
- Bits 28-25: SYSTIMER_TARGET1_PERIOD_MODE
- Bits 24-0: SYSTIMER_TARGET1_PERIOD

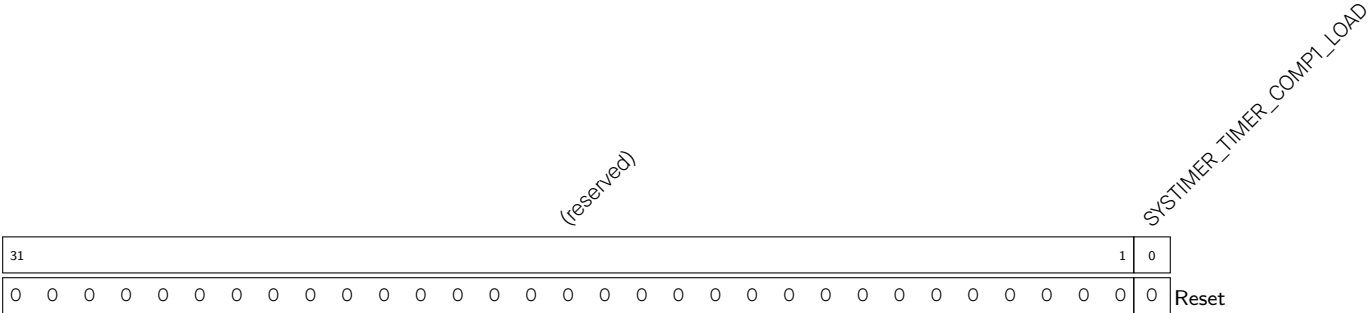
The register is shown with a value of 0x000000.

SYSTIMER_TARGET1_PERIOD 配置待装载至 COMP1 的报警周期。(R/W)

SYSTIMER_TARGET1_PERIOD_MODE 选择比较器 COMP1 生成报警的模式。具体配置值请见 [SYSTIMER_TARGET0_PERIOD_MODE](#)。(R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL 选择要与 COMP1 比较的计数器。具体配置值请见 [SYSTIMER_TARGET0_TIMER_UNIT_SEL](#)。(R/W)

Register 9.21. SYSTIMER_COMP1_LOAD_REG (0x0054)

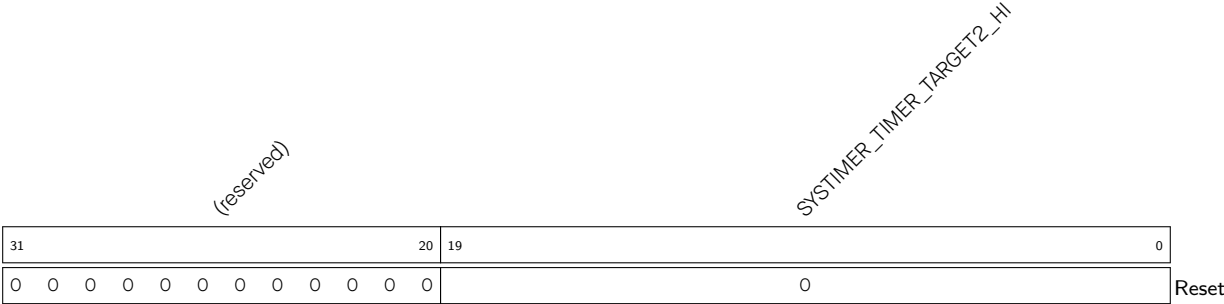


SYSTIMER_TIMER_COMP1_LOAD 配置是否使能比较器 COMP1 同步，即是否重新装载报警值或报警周期到 COMP1。

0：无效

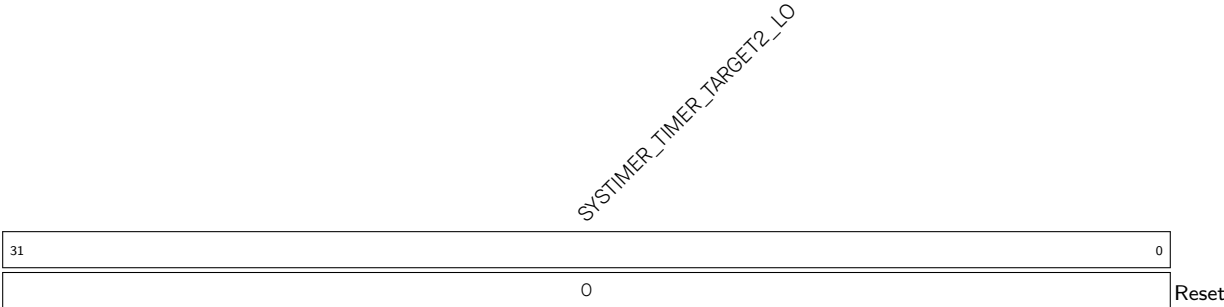
1：使能比较器 COMP1 同步 (WT)

Register 9.22. SYSTIMER_TARGET2_HI_REG (0x002C)



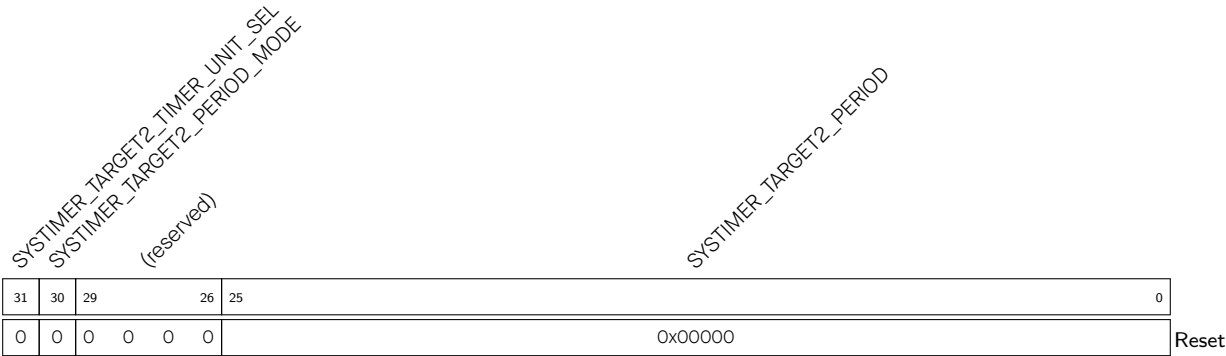
SYSTIMER_TIMER_TARGET2_HI 配置待装载至比较器 COMP2 的报警值，高 20 位。(R/W)

Register 9.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO 配置待装载至比较器 COMP2 的报警值，低 32 位。(R/W)

Register 9.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

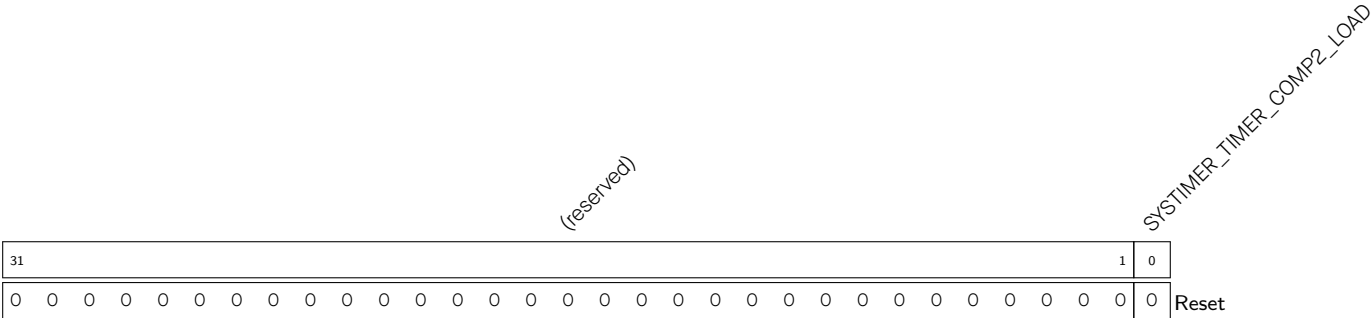


SYSTIMER_TARGET2_PERIOD 配置待装载至 COMP2 的报警周期。(R/W)

SYSTIMER_TARGET2_PERIOD_MODE 选择比较器 COMP2 生成报警的模式。具体配置值请见 [SYSTIMER_TARGET0_PERIOD_MODE](#)。(R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL 选择要与 COMP2 比较的计数器。具体配置值请见 [SYSTIMER_TARGET0_TIMER_UNIT_SEL](#)。(R/W)

Register 9.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



SYSTIMER_TIMER_COMP2_LOAD 配置是否使能比较器 COMP2 同步，即是否重新装载报警值或报警周期到 COMP2。

0: 无效

1: 使能比较器 COMP2 同步 (WT)

Register 9.26. SYSTIMER_INT_ENA_REG (0x0064)

Diagram of the 32-bit SYSTEMER register structure:

- Bits 31 to 3: (reserved)
- Bits 2 to 1: SYSTEMER_TARGET2_INT_ENA
- Bit 0: SYSTEMER_TARGET1_INT_ENA
- Reset bit (indicated by a diagonal line through the bit)

SYSTIMER_TARGETO_INT_ENA 写1使能 SYSTIMER_TARGETO_INT。(R/W)

SYSTIMER_TARGET1_INT_ENA 写1使能 SYSTIMER_TARGET1_INT。(R/W)

SYSTIMER_TARGET2_INT_ENA 写1使能 SYSTIMER_TARGET2_INT。(R/W)

Register 9.27. SYSTIMER_INT_RAW_REG (0x0068)

Diagram illustrating the structure of the SYSTEMER register (32 bits total):

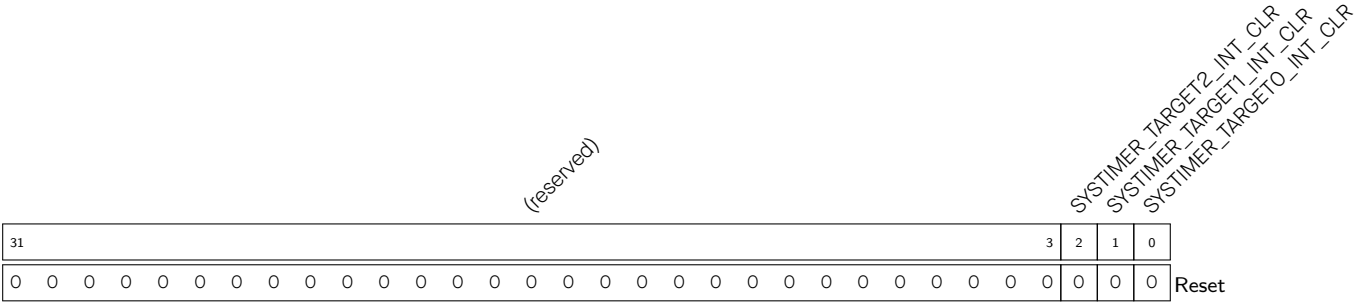
- Bits 31-28: SYSTEMER_TARGET2_INT_RAW (4 bits)
- Bits 27-24: SYSTEMER_TARGET1_INT_RAW (4 bits)
- Bits 23-20: SYSTEMER_TARGET0_INT_RAW (4 bits)
- Bits 19-16: (reserved) (4 bits)
- Bits 3-0: Reset (4 bits)

SYSTIMER_TARGETO_INT_RAW SYSTIMER_TARGETO_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW SYSTIMER_TARGET1_INT 的原始中断状态。(R/WTC/SS)

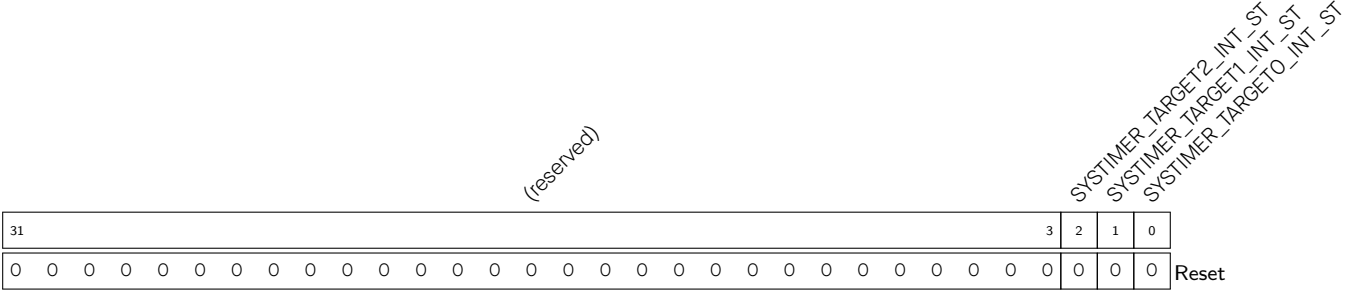
SYSTIMER_TARGET2_INT_RAW SYSTIMER_TARGET2_INT 的原始中断状态。(R/WTC/SS)

Register 9.28. SYSTIMER_INT_CLR_REG (0x006C)



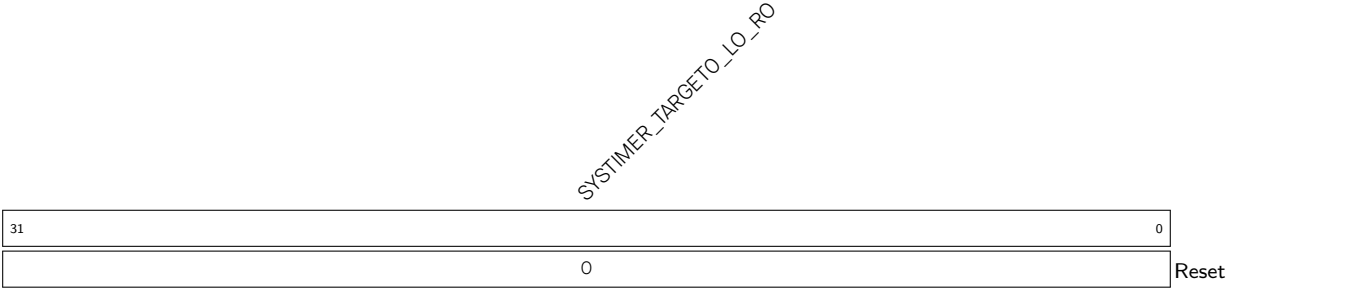
- SYSTIMER_TARGET0_INT_CLR 写 1 清除 SYSTIMER_TARGET0_INT。(WT)
- SYSTIMER_TARGET1_INT_CLR 写 1 清除 SYSTIMER_TARGET1_INT。(WT)
- SYSTIMER_TARGET2_INT_CLR 写 1 清除 SYSTIMER_TARGET2_INT。(WT)

Register 9.29. SYSTIMER_INT_ST_REG (0x0070)



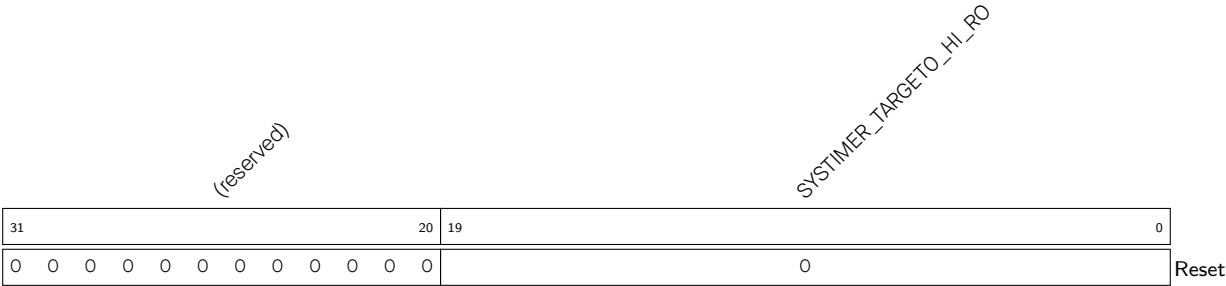
- SYSTIMER_TARGET0_INT_ST SYSTIMER_TARGET0_INT 的屏蔽中断状态。(RO)
- SYSTIMER_TARGET1_INT_ST SYSTIMER_TARGET1_INT 的屏蔽中断状态。(RO)
- SYSTIMER_TARGET2_INT_ST SYSTIMER_TARGET2_INT 的屏蔽中断状态。(RO)

Register 9.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)



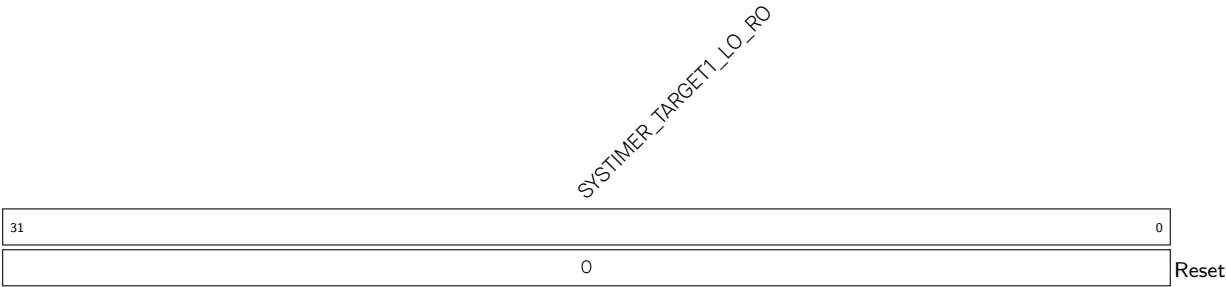
- SYSTIMER_TARGET0_LO_RO 表示 COMPO 的实际报警值，低 32 位。(RO)

Register 9.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)



SYSTIMER_TARGET0_HI_RO 表示 COMPO 的实际报警值，高 20 位。(RO)

Register 9.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)



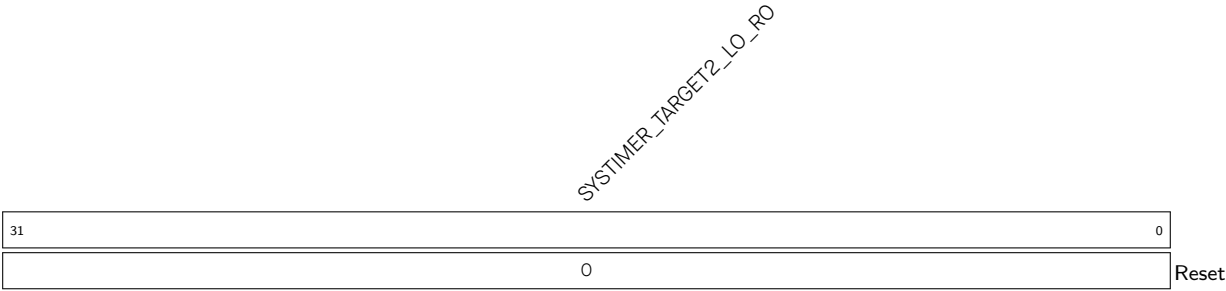
SYSTIMER_TARGET1_LO_RO 表示 COMP1 的实际报警值，低 32 位。(RO)

Register 9.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)



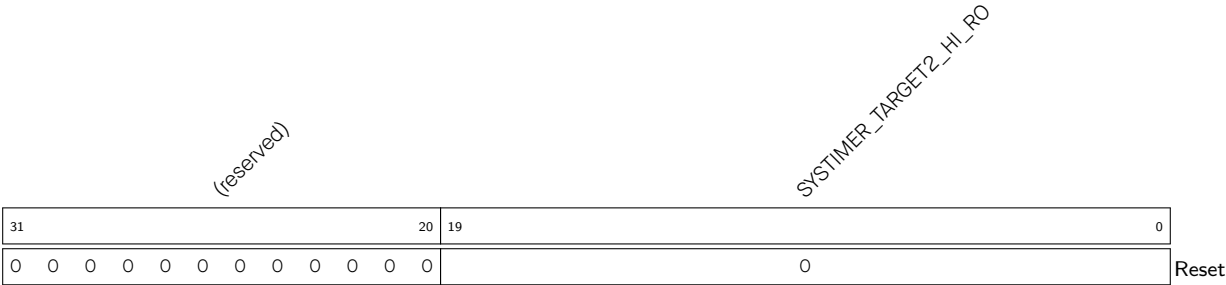
SYSTIMER_TARGET1_HI_RO 表示 COMP1 的实际报警值，高 20 位。(RO)

Register 9.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)



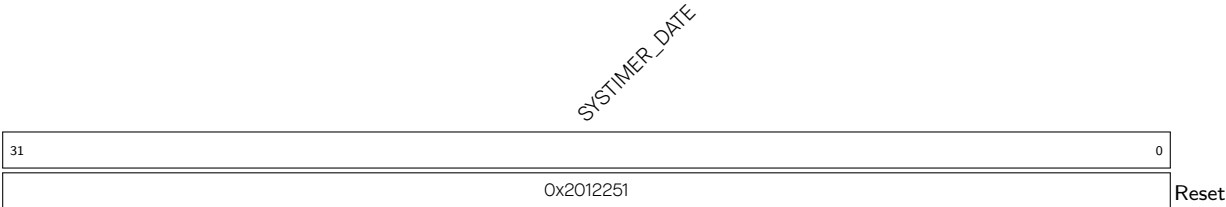
SYSTIMER_TARGET2_LO_RO 表示 COMP2 的实际报警值，低 32 位。(RO)

Register 9.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)



SYSTIMER_TARGET2_HI_RO 表示 COMP2 的实际报警值，高 20 位。(RO)

Register 9.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE 版本控制寄存器。(R/W)

第 10 章

定时器组 (TIMG)

10.1 概述

通用定时器可用于准确设定时间间隔、在一定间隔后触发（周期或非周期的）中断或充当硬件时钟。如图 10.1-1 所示，ESP32-C5 包含两个定时器组，即定时器组 0 和定时器组 1（下文用 TIMG n 表示， n 为 0 或 1）。每个定时器组有一个通用定时器（下文用 T0 表示）和一个主系统看门狗定时器。通用定时器基于 16 位预分频器和 54 位可自动重新加载的可逆计数器。

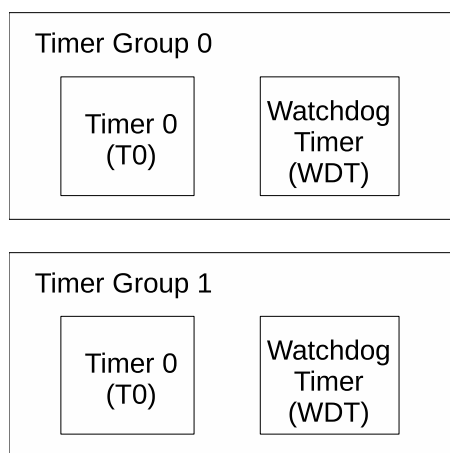


图 10.1-1. 定时器组概览

本章虽然包含主系统看门狗定时器的寄存器描述，但其功能描述请参阅章节 11 看门狗定时器 (WDT)。本章中“定时器”指代通用定时器。

10.2 主要特性

定时器具有如下功能：

- 54 位时基计数器，可配置成递增或递减
- 三个时钟源：PLL_F80M_CLK 或 XTAL_CLK 或 RC_FAST_CLK 时钟
- 16 位时钟预分频器，分频系数为 2 到 65536
- 可读取时基计数器的实时值
- 暂停和恢复时基计数器
- 可配置的警报生成机制
- 计数器值重新加载——报警时自动重新加载或软件控制的即时重新加载
- TIMGO 定时器组慢速时钟频率计算

- 电平触发中断
- 支持多个 ETM 任务和事件

10.3 结构概览

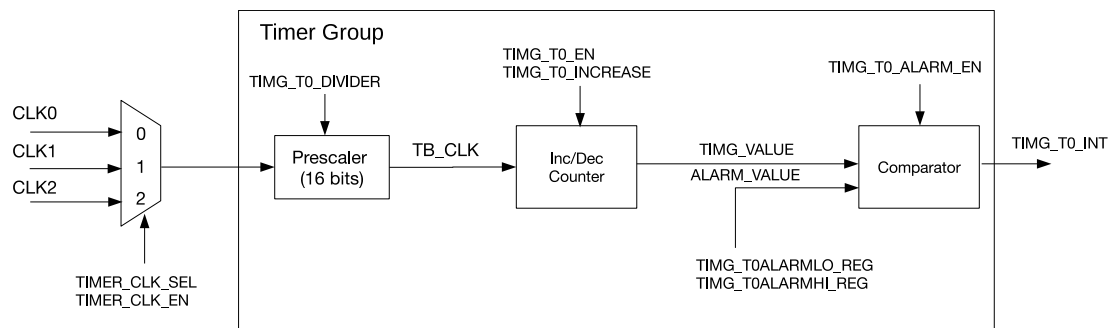


图 10.3-1. 定时器组架构

图 10.3-1 为定时器组的 TO。TO 包含一个 16 位整数预分频器、一个时基计数器和一个用于产生警报的比较器。

10.4 功能描述

10.4.1 16 位预分频器与时钟选择器

以定时器组 0 中的 TO 定时器为例：

- 该定时器可通过配置 TIMER_CLK_SEL 来选择时钟源，TIMER_CLK_SEL 是寄存器 PCR_TIMERGROUPO_TIMER_CLK_CONF_REG 的 PCR_TGO_TIMER_CLK_SEL 字段。该字段为 0 时，选择 CLK0 时钟源进行输入，CLK0 是 XTAL_CLK；为 1 时，选择 CLK1 时钟源进行输入，CLK1 是 RC_FAST_CLK；为 2 时，选择 CLK2 时钟源进行输入，CLK2 是 PLL_F80M_CLK 为时钟源。
- 要开启所选时钟需配置 TIMER_CLK_EN，TIMER_CLK_EN 是 PCR_TIMERGROUPO_TIMER_CLK_CONF_REG 寄存器的 PCR_TGO_TIMER_CLK_EN 字段。要开启时钟需置位该字段，要关闭时钟需清零该字段。所选时钟经 16 位预分频器分频，产生时基计数器使用的时基计数器时钟 (TB_CLK)。16 位预分频器的分频系数可通过 TIMG_TO_DIVIDER 字段配置。

TIMG_TO_DIVIDER 字段可配置为 0 ~ 65535 之间的任意值，分频系数的范围为 2 ~ 65536。具体来说：

- 当 TIMG_TO_DIVIDER 字段置 0 时：分频系数为 65536。
- 当 TIMG_TO_DIVIDER 字段置 1 时：分频系数为 2。
- 当 TIMG_TO_DIVIDER 字段置 2 时：分频系数仍为 2。
- 当 TIMG_TO_DIVIDER 字段置 3 ~ 65525 时：分频系数为 3 ~ 65525。

要更改 16 位预分频器，需先重新配置 TIMG_TO_DIVIDER 字段，再将 TIMG_TO_DIVCNT_RST 置 1，同时需要关闭定时器（即 TIMG_TO_EN 必须清零），否则会造成不可预知的结果。

10.4.2 54 位时基计数器

54 位时基计数器基于 TB_CLK，可通过 TIMG_TO_INCREASE 字段配置为递增或递减。时基计数器可通过置位或清零 TIMG_TO_EN 字段使能或关闭。使能时，时基计数器的值会在每个 TB_CLK 周期递增或递减。关闭时，时基计数器暂停计数。注意，无论 TIMG_TO_EN 是否置位，TIMG_TO_INCREASE 字段都可以更改，时基计数器可立即改变计数方向。

时基计数器 54 位定时器的当前值必须被锁入两个寄存器，才能被 CPU 读取（因为 CPU 为 32 位）。向 TIMG_TOUUPDATE_REG 写入任意值时，54 位定时器的值开始被锁入寄存器 TIMG_TOLO_REG 和 TIMG_TOHI_REG，两个寄存器分别锁存低 32 位和高 22 位。当 TIMG_TOUUPDATE_REG 被硬件清零，表明锁存操作已经完成，可以从这两个寄存器中读取当前计数值。在 TIMG_TOUUPDATE_REG 被写入新值之前，保持寄存器 TIMG_TOLO_REG 和 TIMG_TOHI_REG 的值不变，以供 32 位的 CPU 读值。

10.4.3 警报生成

定时器可配置为在当前值与报警值相同时触发报警。报警会产生中断，同时（可选择）让定时器的当前值自动重新加载（详见第 10.4.4 节）。

54 位报警值可在 TIMG_TOALARMLO_REG 和 TIMG_TOALARMHI_REG 配置，两者分别代表报警值的低 32 位和高 22 位。但是，只有置位 TIMG_TO_ALARM_EN 字段使能报警功能后，配置的报警值才会生效。为解决报警使能“过晚”（即报警使能时，定时器的值已过报警值），出现以下情况时硬件会立即触发报警：

- 可逆计数器向上计数时，定时器的当前值高于报警值（在一定范围内）
- 可逆计数器向下计数时，定时器的当前值低于报警值（在一定范围内）

表 10.4-1 和表 10.4-2 说明了定时器当前值、报警值与报警触发的关系。假设定时器当前值和报警值如下：

- TIMG_VALUE = {TIMG_TOHI_REG, TIMG_TOLO_REG}
- ALARM_VALUE = {TIMG_TOALARMHI_REG, TIMG_TOALARMLO_REG}

表 10.4-1. 可逆计数器向上计数时的报警场景

场景	范围	报警
1	$ALARM\_VALUE - TIMG\_VALUE > 2^{53}$	触发
2	$0 < ALARM\_VALUE - TIMG\_VALUE \leq 2^{53}$	可逆计数器向上计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
3	$0 \leq TIMG\_VALUE - ALARM\_VALUE < 2^{53}$	触发
4	$TIMG\_VALUE - ALARM\_VALUE \geq 2^{53}$	可逆计数器向上计数达到最大值时，重新开始从 0 向上计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

表 10.4-2. 可逆计数器向下计数时的报警场景

场景	范围	报警
5	$TIMG\_VALUE - ALARM\_VALUE > 2^{53}$	触发
6	$0 < TIMG\_VALUE - ALARM\_VALUE \leq 2^{53}$	可逆计数器向下计数, $TIMG\_VALUE$ 达到 $ALARM\_VALUE$ 时报警
7	$0 \leq ALARM\_VALUE - TIMG\_VALUE < 2^{53}$	触发
8	$ALARM\_VALUE - TIMG\_VALUE \geq 2^{53}$	可逆计数器向下计数达到最小值时, 重新开始从最大值向下计数, $TIMG\_VALUE$ 达到 $ALARM\_VALUE$ 时触发报警

报警时, `TIMG_TO_ALARM_EN` 字段自动清零, 在下次置位 `TIMG_TO_ALARM_EN` 前不会再次报警。

10.4.4 定时器重新加载

定时器重新加载指将定时器的低 32 位和高 22 位分别更新为 `TIMG_TO_LOAD_LO` 和 `TIMG_TO_LOAD_HI` 字段存储的重新加载值。但是, 把重新加载值写入 `TIMG_TO_LOAD_LO` 和 `TIMG_TO_LOAD_HI` 字段不会改变定时器的当前值。写入的重新加载值会被定时器忽视, 直到重新加载事件被触发。重新加载事件可由软件即时重新加载或报警时自动重新加载触发。

CPU 在寄存器 `TIMG_TOLOAD_REG` 写任意值会触发软件即时重新加载, 定时器的当前值会立即改变。若此时 `TIMG_TO_EN` 是置位状态, 定时器会继续从新数值开始递增或递减计数。若此时 `TIMG_TO_ALARM_EN` 置位, 仍会根据表 10.4-1 或表 10.4-2 中的关系在对应时刻产生报警。若 `TIMG_TO_EN` 是清零状态, 定时器将保持当前值, 直至计数重新使能。

报警时, 自动重新加载功能可让定时器在报警时重新加载, 从重新加载值开始继续递增或递减计数。该功能通常用于周期性报警时重置定时器的值。要能使报警时自动重新加载, 需将 `TIMG_TO_AUTORELOAD` 字段置 1。如未使能该功能, 报警后定时器的值会在过报警值后继续递增或递减。

10.4.5 TIMGO 定时器组慢速时钟频率计算

TIMGO 定时器组可以以 `XTAL_CLK` 为基准时钟, 计算 ESP32-C5 提供的慢速时钟的实际频率, 输入给定时器组的具体慢速时钟解释请查看章节 12 实时时钟定时器。计算方式如下:

1. 通过周期性或单次计算的方式启动频率计算模块 (详见章节 10.7.5);
2. 在接收到计算开始的信号后, 两个分别工作在 `XTAL_CLK` 以及慢速时钟的计数器同时开始计数, 当慢速时钟的计数器达到设定的计算周期 `CO` 时, 同时停止两个计数器;
3. 通过 `XTAL_CLK` 的计数器值 `C1` 即可计算慢速时钟的时钟频率: $f_{rtc} = \frac{CO \times f_{XTAL\_CLK}}{C1}$

请注意慢速时钟频率计算功能使用时应保证输入频率不大于 $\frac{f_{XTAL\_CLK}}{2}$ 。

10.5 事件任务矩阵功能

在 ESP32-C5 中, 定时器组支持 ETM 功能, 即可以通过任意外设的 ETM 事件触发定时器组的 ETM 任务, 或者通过定时器组的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息, 请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与定时器组相关的 ETM 任务和 ETM 事件。

定时器组可接收的 ETM 任务有:

- TGO_TASK_CNT_START_TIMER0: 触发时启用时基计数器。
- TG1_TASK_CNT_START_TIMER0: 触发时启用时基计数器。
- TGO_TASK_CNT_STOP_TIMER0: 触发时关闭时基计数器。
- TG1_TASK_CNT_STOP_TIMER0: 触发时关闭时基计数器。

说明:

以上两个 ETM 任务与 APB 配置 [TIMG_TO_EN](#) 实现的功能相同。当这些操作同时发生时, 各操作优先级由高到低如下:

1. TGO_TASK_CNT_START_TIMER0 与 TG1_TASK_CNT_START_TIMER0: 触发时启用时基计数器
2. TGO_TASK_CNT_STOP_TIMER0 与 TG1_TASK_CNT_STOP_TIMER0: 触发时关闭时基计数器
3. APB 方式配置 [TIMG_TO_EN](#): 启用或关闭时基计数器

- TGO_TASK_ALARM_START_TIMER0: 触发时启用警报生成。
- TG1_TASK_ALARM_START_TIMER0: 触发时启用警报生成。

说明:

通过 APB 方式配置 [TIMG_ALARM_EN](#) 以及硬件事件触发也可以启用警报生成功能。当这些操作同时发生时, 各个操作的优先级从高到低依次为:

1. TGO_TASK_ALARM_START_TIMER0 与 TG1_TASK_ALARM_START_TIMER0: 触发时启用警报生成
2. 警报事件: 触发时关闭警报生成
3. APB 方式配置 [TIMG_ALARM_EN](#): 启用/关闭警报生成

- TGO_TASK_CNT_CAP_TIMER0: 触发时会将当前计数器的值更新至 [TIMG_TOLO_REG](#) 和 [TIMG_TOHI_REG](#) 寄存器。
- TG1_TASK_CNT_CAP_TIMER0: 触发时会将当前计数器的值更新至 [TIMG_TOLO_REG](#) 和 [TIMG_TOHI_REG](#) 寄存器。
- TGO_TASK_CNT_RELOAD_TIMER0: 触发时会用存储在 [TIMG_TO_LOAD_LO](#) 和 [TIMG_TO_LOAD_HI](#) 中的重新加载值覆盖当前计数器值。
- TG1_TASK_CNT_RELOAD_TIMER0: 触发时会用存储在 [TIMG_TO_LOAD_LO](#) 和 [TIMG_TO_LOAD_HI](#) 中的重新加载值覆盖当前计数器值。

定时器组可产生的 ETM 事件有:

- TGO_EVT_CNT_CMP_TIMER0: 表示 TIMGO 的 TO 定时器中断事件。
- TG1_EVT_CNT_CMP_TIMER0: 表示 TIMG1 的 TO 定时器中断事件。

只有当 [TIMG_ETM_EN](#) 设置为 1 时, ETM 任务和事件才会生效。

在具体应用中, 定时器组的 ETM 事件可以用来触发定时器组的 ETM 任务。例如, [TGO_EVT_CNT_CMP_TIMER0](#) 与 [TG1_EVT_CNT_CMP_TIMER0](#) 事件分别可以触发 [TGO_TASK_ALARM_START_TIMER0](#) 与 [TG1_TASK_ALARM_START_TIMER0](#) 任务, 从而实现周期性报警。具体配置步骤请参考 [10.7.4 通过 ETM 设置定时器用于周期性报警](#)。

10.6 中断

ESP32-C5 的 TIMG n 可以生成以下中断信号并发送给中断矩阵。

- TG n _TO_INTR
- TG n _WDT_INTR

这些中断信号由 TIMG n 的内部中断源生成。表 10.6-1 列出了 TIMG n 的内部中断源、中断触发条件、以及生成的中断信号。

表 10.6-1. TIMG n 的内部中断源

内部中断源	触发条件	中断信号
TIMG_TO_INT	TO 定时器产生告警	TG n _TO_INTR
TIMG_WDT_INT	看门狗定时器产生告警	TG n _WDT_INTR

说明:

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 [to be added later] > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 10.8 寄存器列表。

10.7 配置流程

10.7.1 定时器用作简单时钟

- 配置时基计数器。
 - 置位或清除 PCR_TGO_TIMER_CLK_SEL 字段选择时钟源。
 - 置位 TIMG_TO_DIVIDER 配置 16 位预分频器。
 - 置位或清除 TIMG_TO_INCREASE 配置定时器方向。
 - 在 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 上写初始值设置定时器的初始值，然后在 TIMG_TOLOAD_REG 上写任意值将初始值重新加载进定时器。
- 置位 TIMG_TO_EN 开启定时器。
- 获得定时器的当前值。
 - 在 TIMG_TOUPDATE_REG 上写任意值锁存定时器的当前值。
 - 等待硬件将 TIMG_TOUPDATE_REG 清 0。
 - 从 TIMG_TOLO_REG 和 TIMG_TOHI_REG 读取锁存的定时器值。

10.7.2 定时器用于单次报警

- 按照第 10.7.1 节的第 1 步配置时基计数器。
- 配置报警。

- 置位 `TIMG_TO_ALARMLO_REG` 和 `TIMG_TO_ALARMHI_REG` 配置报警值。
 - 置位 `TIMG_TO_INT_ENA` 使能中断。
3. 清零 `TIMG_TO_AUTORELOAD` 关闭自动重新加载。
 4. 置位 `TIMG_TO_ALARM_EN` 开启报警。
 5. 处理报警中断。
 - 置位定时器在 `TIMG_TO_INT_CLR` 的对应位清除中断。
 - 清零 `TIMG_TO_EN` 关闭定时器。

10.7.3 通过 APB 设置定时器用于周期性报警

1. 按照第 10.7.1 节的第 1 步配置时基计数器。
2. 按照第 10.7.2 节的第 2 步配置报警。
3. 置位 `TIMG_TO_AUTORELOAD` 使能自动重新加载，将重新加载值写入 `TIMG_TO_LOAD_LO` 和 `TIMG_TO_LOAD_HI`。
4. 置位 `TIMG_TO_ALARM_EN` 开启报警。
5. 处理报警中断（每次报警时重复）。
 - 置位定时器在 `TIMG_TO_INT_CLR` 的对应位清除中断。
 - 如下一次报警需要新的报警值和重新加载值（即每次都有不同的报警间隔），则应根据需要重新配置 `TIMG_TO_ALARMLO_REG`、`TIMG_TO_ALARMHI_REG`、`TIMG_TO_LOAD_LO` 和 `TIMG_TO_LOAD_HI`。否则，上述寄存器应保持不变。
 - 置位 `TIMG_TO_ALARM_EN` 重新使能报警。
- 6.（最后一次报警时）关闭定时器。
 - 置位定时器在 `TIMG_TO_INT_CLR` 的对应位清除中断。
 - 清零 `TIMG_TO_EN` 关闭定时器。

10.7.4 通过 ETM 设置定时器用于周期性报警

1. 使能 ETM 模块的时钟
2. 将 ETM 事件映射到相应的 ETM 任务（使用 ETM 事件触发对应的 ETM 任务）
 - 如果 `TIMG_TO_AUTORELOAD` 置 1，通过一个 ETM 通道映射 `TGO_EVT_CNT_CMP_TIMER0` 与 `TG1_EVT_CNT_CMP_TIMER0` 事件到 `TGO_TASK_ALARM_START_TIMER0` 与 `TG1_TASK_ALARM_START_TIMER0` 任务。
 - 如果 `TIMG_TO_AUTORELOAD` 置 0，除了映射 `TGO_EVT_CNT_CMP_TIMER0` 与 `TG1_EVT_CNT_CMP_TIMER0` 事件到 `TGO_TASK_ALARM_START_TIMER0` 与 `TG1_TASK_ALARM_START_TIMER0` 任务外，还需要用另一个 ETM 通道映射 `TGO_EVT_CNT_CMP_TIMER0` 与 `TG1_EVT_CNT_CMP_TIMER0` 事件到 `TGO_TASK_CNT_RELOAD_TIMER0` 与 `TG1_TASK_CNT_RELOAD_TIMER0` 任务。
3. 选择使能一个或两个 ETM 通道。

4. 将 `TIMER_ETM_EN` 置 1 从而使能定时器组的 ETM 事件和任务。
5. 按照第 10.7.1 节的第 1 步配置时基计数器。
6. 按照第 10.7.2 节的第 2 步配置报警。
7. 通过 `TIMG_TO_LOAD_LO` 和 `TIMG_TO_LOAD_HI` 配置重新加载值。
8. 处理 `TGO_EVT_CNT_CMP_TIMER0` 与 `TG1_EVT_CNT_CMP_TIMER0`
 - 当警报生成时, `TGO_EVT_CNT_CMP_TIMER0` 与 `TG1_EVT_CNT_CMP_TIMER0` 事件也会产生, 此时报警器关闭警报生成。
 - 如果 `TIMG_TO_AUTORELOAD` 为 1, 重新加载值会覆盖当前计数器的值。
`TGO_TASK_ALARM_START_TIMER0` 与 `TG1_TASK_ALARM_START_TIMER0` 会重新使能警报生成。
 - 如果 `TIMG_TO_AUTORELOAD` 为 0, 由于 `TGO_TASK_CNT_RELOAD_TIMER0` 与 `TG1_TASK_CNT_RELOAD_TIMER0` 任务, 重新加载值会覆盖当前计数器的值。
`TGO_TASK_ALARM_START_TIMER0` 与 `TG1_TASK_ALARM_START_TIMER0` 任务会重新使能警报生成。
9. (最后一次报警时) 关闭定时器。
 - 关闭用于映射定时器组事件和任务的 ETM 通道。
 - 设置 `TIMER_ETM_EN` 为 0。
 - 置位定时器在 `TIMG_TO_INT_CLR` 的对应位清除中断。
 - 清零 `TIMG_TO_EN` 关闭定时器。

10.7.5 慢速时钟频率计算

1. 单次计算

- 设置 `PCR_32_SEL` 选择需要计算频率的时钟, 该寄存器设置值与时钟源具体对应关系参考章节 12 实时时钟定时器。
- 设置 `PCR_FOSC_TICK_NUM` 对频率较高的 `RC_FAST_CLK` 进行分频, 以满足慢速时钟频率计算的前提条件。
- 设置 `TIMG_RTC_CALI_MAX` 配置频率计算时间。
- 清空 `TIMG_RTC_CALI_START_CYCLING` 选择单次校准模式, 然后配置 `TIMG_RTC_CALI_START` 开启两个计数器。
- 等待 `TIMG_RTC_CALI_RDY` 的值变为 1, 读取 `TIMG_RTC_CALI_VALUE` 获取 `XTAL_CLK` 计数器值, 根据章节 10.4.5 的公式计算慢速时钟的频率。

2. 周期性计算

- 设置 `PCR_32_SEL` 选择需要计算频率的时钟。
- 设置 `PCR_FOSC_TICK_NUM` 对频率较高的 `RC_FAST_CLK` 进行分频, 以满足慢速时钟频率计算的前提条件。
- 设置 `TIMG_RTC_CALI_MAX` 配置频率计算时间。
- 使能 `TIMG_RTC_CALI_START_CYCLING`, 硬件将不间断进行频率计算过程。

- 只要 `TIMG_RTC_CALI_CYCLING_DATA_VLD` 为 1，即表示 `TIMG_RTC_CALI_VALUE` 有效。

3. 超时

如果慢速时钟的计数器没有在 `TIMG_RTC_CALI_TIMEOUT_RST_CNT` 的 XTAL_CLK 计数器内完成计数，将置位 `TIMG_RTC_CALI_TIMEOUT` 标记计算超时。

10.8 寄存器列表

本小节的所有地址均为相对于 **定时器组** 基地址的地址偏移量（相对地址），具体基地址请见章节 4 **系统和存储器** 中的表 4.3-2。

请查看章节 **寄存器的访问类型**，了解“访问”列缩写的含义。

名称	描述	地址	访问
定时器 0 控制和配置寄存器			
TIMG_TOCONFIG_REG	定时器 0 配置寄存器	0x0000	varies
TIMG_TOLO_REG	定时器 0 的当前值，低 32 位	0x0004	RO
TIMG_TOHI_REG	定时器 0 的当前值，高 22 位	0x0008	RO
TIMG_TOUUPDATE_REG	写 值 将 当 前 定 时 器 的 值 复 制 到 TIMG_TOLO_REG 或 TIMG_TOHI_REG	0x000C	R/ W/ SC
TIMG_TOALARMLO_REG	定时器 0 的报警值，低 32 位	0x0010	R/W
TIMG_TOALARMHI_REG	定时器 0 的报警值，高 22 位	0x0014	R/W
TIMG_TOLOADLO_REG	定时器 0 的重新加载值，低 32 位	0x0018	R/W
TIMG_TOLOADHI_REG	定时器 0 的重新加载值，高 22 位	0x001C	R/W
TIMG_TOLOAD_REG	写 值 从 TIMG_TOLOADLO_REG 或 TIMG_TOLOADHI_REG 上加载定时器	0x0020	WT
看门狗定时器控制和配置寄存器			
TIMG_WDTCONFIG0_REG	看门狗定时器配置寄存器	0x0048	varies
TIMG_WDTCONFIG1_REG	看门狗定时器预分频器寄存器	0x004C	varies
TIMG_WDTCONFIG2_REG	看门狗定时器阶段 0 超时值	0x0050	R/W
TIMG_WDTCONFIG3_REG	看门狗定时器阶段 1 超时值	0x0054	R/W
TIMG_WDTCONFIG4_REG	看门狗定时器阶段 2 超时值	0x0058	R/W
TIMG_WDTCONFIG5_REG	看门狗定时器阶段 3 超时值	0x005C	R/W
TIMG_WDTFEED_REG	写值喂看门狗定时器	0x0060	WT
TIMG_WDTWPROTECT_REG	看门狗写保护寄存器	0x0064	R/W
RTC 校准控制和配置寄存器			
TIMG_RTCCALICFG_REG	RTC 校准配置寄存器	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC 校准配置寄存器 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC 校准配置寄存器 2	0x0080	varies
中断寄存器			
TIMG_INT_ENA_TIMERS_REG	中断使能位	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	原始中断状态	0x0074	R/ SS/ WTC
TIMG_INT_ST_TIMERS_REG	屏蔽中断状态	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	中断清除位	0x007C	WT
版本寄存器			
TIMG_NTIMERS_DATE_REG	版本控制寄存器	0x00F8	R/W
时钟配置寄存器			
TIMG_REGCLK_REG	定时器组时钟门控寄存器	0x00FC	R/W

10.9 寄存器

本小节的所有地址均为相对于 定时器组 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

Register 10.1. TIMG_TOCONFIG_REG (0x0000)

TIMG_TO_EN TIMG_TO_INCREASE TIMG_TO_AUTORELOAD				TIMG_TO_DIVIDER								TIMG_TO_DIVCNT_RST (reserved) TIMG_TO_ALARM_EN				(reserved)											
31	30	29	28									13	12	11	10	9									0		
0	1	1	0x01									0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

TIMG_TO_ALARM_EN 配置是否使能定时器 0 的报警功能。报警时，此位自动清零。

- 0: 禁用
- 1: 使能
- (R/W/SC)

TIMG_TO_DIVCNT_RST 配置是否复位定时器 0 时钟分频器的计数器。

- 0: 无效
- 1: 复位
- (WT)

TIMG_TO_DIVIDER 表示定时器 0 时钟 (TO_clk) 的预分频器值。(R/W)

TIMG_TO_AUTORELOAD 配置是否使能定时器 0 报警时自动重新加载使能。

- 0: 无效
- 1: 使能
- (R/W)

TIMG_TO_INCREASE 配置定时器 0 的时基计数器的计数方向。

- 0: 递减
- 1: 递增
- (R/W)

TIMG_TO_EN 配置是否使能定时器 0 的时基计数器。

- 0: 禁用
- 1: 使能
- (R/W/SS/SC)

Register 10.2. TIMG_TOLO_REG (0x0004)

TIMG_TO_LO																															
31																															0
0x000000																															
Reset																															

TIMG_TO_LO 表示定时器 0 时基计数器的低 32 位的值。仅在 **TIMG_TOUPLICATE_REG** 写值后有效。
计数单位：TO_clk。
(RO)

Register 10.3. TIMG_TOHI_REG (0x0008)

(reserved)											TIMG_TO_HI																														
31											22											21																			0
0 0 0 0 0 0 0 0 0 0 0											0x0000																			Reset											

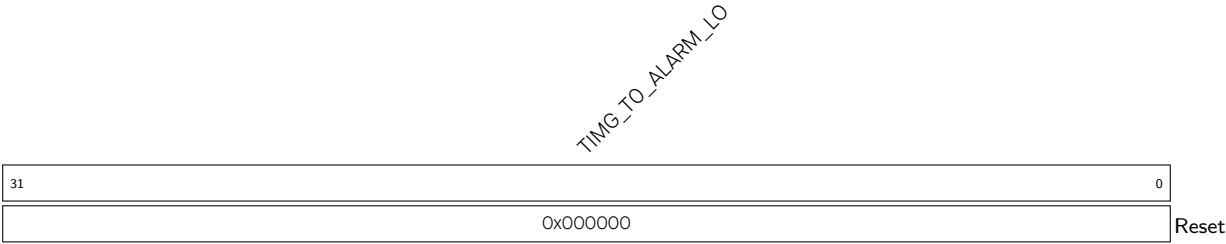
TIMG_TO_HI 表示定时器 0 时基计数器的高 22 位的值。仅在 **TIMG_TOUPLICATE_REG** 写值后有效。
计数单位：TO_clk。
(RO)

Register 10.4. TIMG_TOUPLICATE_REG (0x000C)

TIMG_TO_UPDATE																(reserved)													
31	30																												0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																													Reset

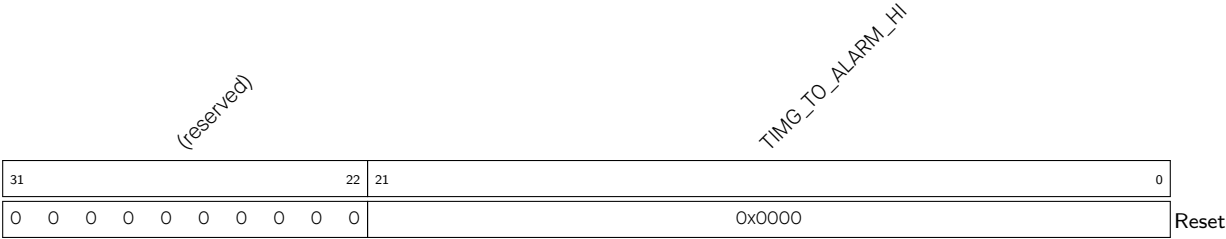
TIMG_TO_UPDATE 配置后会锁存计数器的值。
0：锁存
1：锁存
注意：无论该位写 0 或写 1，计数器的值均将被锁存。
(R/W/SC)

Register 10.5. TIMG_TOALARMLO_REG (0x0010)



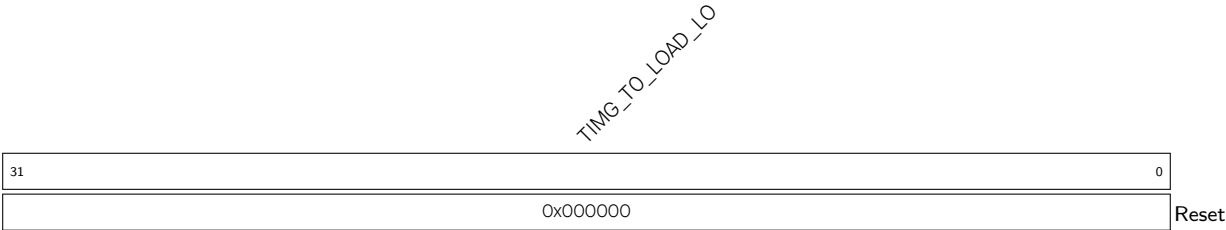
TIMG_TO_ALARM_LO 配置定时器 0 时基计数器触发报警值的低 32 位。仅在 [TIMG_TO_ALARM_EN](#) 为 1 时有效。
计数单位：TO_clk。
(R/W)

Register 10.6. TIMG_TOALARMHI_REG (0x0014)



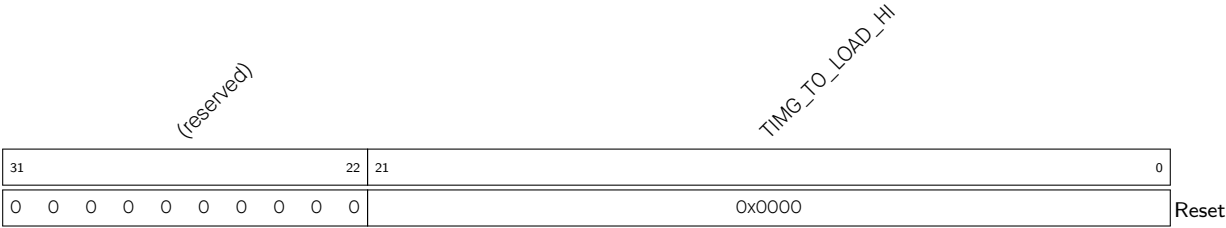
TIMG_TO_ALARM_HI 配置定时器 0 时基计数器触发报警值的高 22 位。仅在 [TIMG_TO_ALARM_EN](#) 为 1 时有效。
计数单位：TO_clk。
(R/W)

Register 10.7. TIMG_TOLOADLO_REG (0x0018)



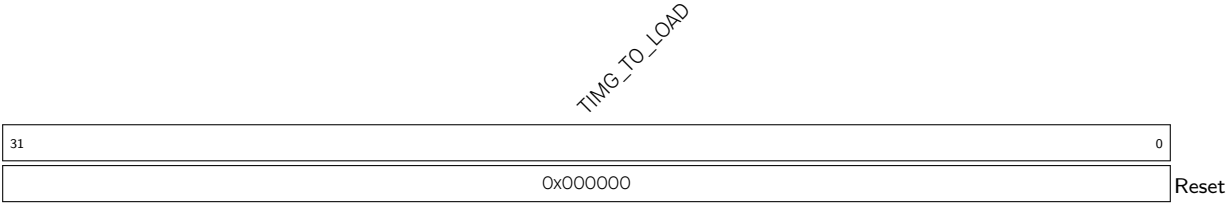
TIMG_TO_LOAD_LO 配置定时器 0 时基计数器重新加载的低 32 位值。
计数单位：TO_clk。
(R/W)

Register 10.8. TIMG_TOLOADHI_REG (0x001C)



TIMG_TO_LOAD_HI 配置定时器 0 时基计数器重新加载的高 22 位值。
计数单位：TO_clk。
(R/W)

Register 10.9. TIMG_TOLOAD_REG (0x0020)



TIMG_TO_LOAD 写任意值触发定时器 0 时基计数器重新加载。(WT)

Register 10.10. TIMG_WDTCONFIG0_REG (0x0048)

TIMG_WDT_EN		TIMG_WDT_STG0		TIMG_WDT_STG1		TIMG_WDT_STG2		TIMG_WDT_STG3		(reserved)		TIMG_WDT_CONF_UPDATE_EN		TIMG_WDT_CPU_RESET_LENGTH		TIMG_WDT_SYS_RESET_LENGTH		TIMG_WDT_FLASHBOOT_MOD_EN		TIMG_WDT_PROCPU_RESET_EN		(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
31	30	29	28	27	26	25	24	23	22	21	20	18	17	15	14	13	12	11														0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
0	0	0	0	0	0	0	0	0	0	0	0x1		0x1		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

TIMG_WDT_PROCPU_RESET_EN 配置是否屏蔽 MWDT 产生的 CPU 复位。单核默认使用，仅在禁用写保护时有效。

0: 屏蔽

1: 不屏蔽

(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN 配置是否使能 flash 启动保护。

0: 禁用

1: 使能

(R/W)

TIMG_WDT_SYS_RESET_LENGTH 配置系统复位信号长度。仅在禁用写保护时有效。

计数单位：mwdt_clk。

0: 8	4: 40
1: 16	5: 64
2: 24	6: 128
3: 32	7: 256

(R/W)

TIMG_WDT_CPU_RESET_LENGTH 配置 CPU 复位信号长度。仅在禁用写保护时有效。

计数单位：mwdt_clk。

0: 8	4: 40
1: 16	5: 64
2: 24	6: 128
3: 32	7: 256

(R/W)

见下页...

Register 10.10. USB_SERIAL_JTAG_CONFO_REG (0x0018)

接上页...

TIMG_WDT_CONF_UPDATE_EN 配置以更新看门狗定时器的配置寄存器。

- 0: 无效
- 1: 更新
(WT)

TIMG_WDT_STG3 配置阶段 3 的超时动作。具体可参考 [TIMG_WDT_STGO](#)。仅在禁用写保护时有效。
(R/W)

TIMG_WDT_STG2 配置阶段 2 的超时动作。具体可参考 [TIMG_WDT_STGO](#)。仅在禁用写保护时有效。
(R/W)

TIMG_WDT_STG1 配置阶段 1 的超时动作。具体可参考 [TIMG_WDT_STGO](#)。仅在禁用写保护时有效。
(R/W)

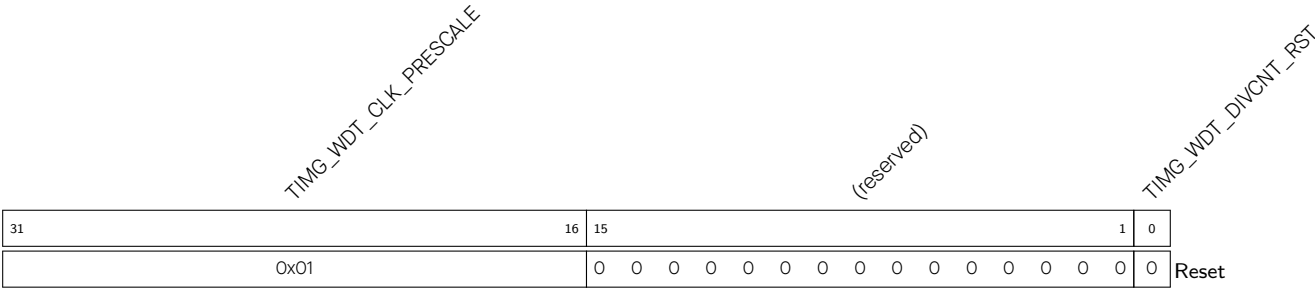
TIMG_WDT_STG0 配置阶段 0 的超时动作。仅在禁用写保护时有效。

- 0: 无效
- 1: 中断
- 2: 复位 CPU
- 3: 复位系统
(R/W)

TIMG_WDT_EN 配置是否使能 MWDt。仅在禁用写保护时有效。

- 0: 禁用
- 1: 使能
(R/W)

Register 10.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST 配置是否复位看门狗定时器时钟分频器的计数器。

0: 无效

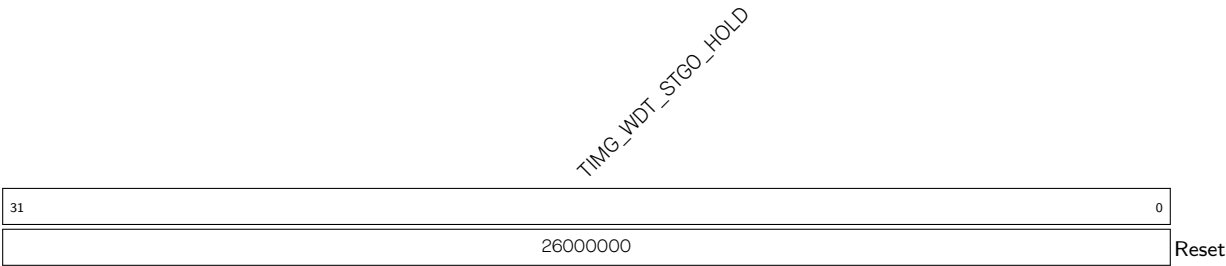
1: 复位 (WT)

TIMG_WDT_CLK_PRESCALE 配置 MWDT 时钟预分频器值。仅在禁用写保护时有效。

MWDT 时钟周期 = MWDT 时钟源周期 * TIMG_WDT_CLK_PRESCALE。

(R/W)

Register 10.12. TIMG_WDTCONFIG2_REG (0x0050)



TIMG_WDT_STGO_HOLD 配置阶段 0 超时时间。仅在禁用写保护时有效。

计数单位: mwdt_clk。

(R/W)

Register 10.13. TIMG_WDTCONFIG3_REG (0x0054)



TIMG_WDT_STG1_HOLD 配置阶段 1 超时时间。仅在禁用写保护时有效。

计数单位: mwdt_clk。

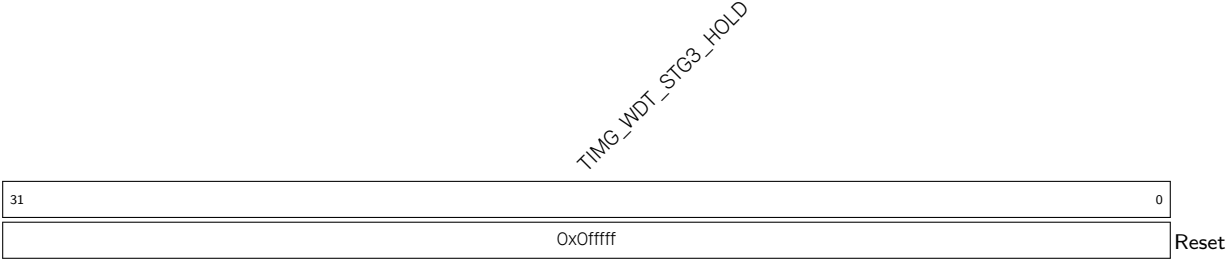
(R/W)

Register 10.14. TIMG_WDTCONFIG4_REG (0x0058)



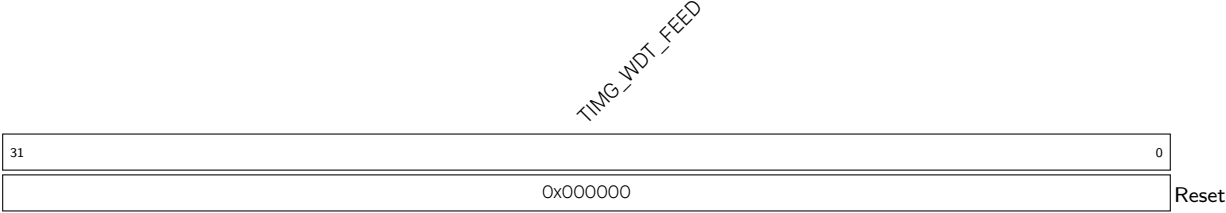
TIMG_WDT_STG2_HOLD 配置阶段 2 超时时间。仅在禁用写保护时有效。
计数单位：mwdt_clk。
(R/W)

Register 10.15. TIMG_WDTCONFIG5_REG (0x005C)



TIMG_WDT_STG3_HOLD 配置阶段 3 超时时间。仅在禁用写保护时有效。
计数单位：mwdt_clk。
(R/W)

Register 10.16. TIMG_WDTFEED_REG (0x0060)



TIMG_WDT_FEED 写任意值喂 MWDT。仅在禁用写保护时有效。(WT)

Register 10.17. TIMG_WDTWPROTECT_REG (0x0064)

TIMG_WDT_WKEY																															
31																															0
0x50d83aa1																															
Reset																															

TIMG_WDT_WKEY 配置一个与复位值不同的值以使能写保护。(R/W)

Register 10.18. TIMG_RTCALICFG_REG (0x0068)

TIMG_RTC_CALI_START																																																		
TIMG_RTC_CALI_MAX																TIMG_RTC_CALI_RDY																																		
																(reserved)																																		
																TIMG_RTC_CALI_START_CYCLING																																		
																(reserved)																																		
31	30															16															15	14	13	12	11															0
0	0x01															0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset											

TIMG_RTC_CALI_START_CYCLING 配置频率计算模式。
0：单次频率计算模式；
1：周期性频率计算模式。
(R/W)

TIMG_RTC_CALI_RDY 表示单次频率计算是否完成。
0：未完成
1：完成
(RO)

TIMG_RTC_CALI_MAX 配置计算 RTC 慢速时钟 SLOW_CLK 频率的时间。
计数单位：XTAL_CLK。
(R/W)

TIMG_RTC_CALI_START 配置是否使能单次频率计算。
0：禁用
1：使能
(R/W)

Register 10.19. TIMG_RTCCALICFG1_REG (0x006C)

TIMG_RTC_CALI_VALUE										(reserved)				TIMG_RTC_CALI_CYCLING_DATA_VLD			
31							7	6	1				0				
0x00000									0	0	0	0	0	0	0	Reset	

- TIMG_RTC_CALI_CYCLING_DATA_VLD** 表示周期性频率计算是否完成。
- 0: 未完成
 - 1: 完成
- (RO)
- TIMG_RTC_CALI_VALUE** 表示单次或周期性频率计算完成时 XTAL_CLK 计数的值。用于计算 RTC 慢时钟的频率。(RO)

Register 10.20. TIMG_RTCCALICFG2_REG (0x0080)

TIMG_RTC_CALI_TIMEOUT_THRES							TIMG_RTC_CALI_TIMEOUT_RST_CNT				(reserved)				TIMG_RTC_CALI_TIMEOUT				
31							7	6	3				2	1	0				
0x1ffffff							3		0		0	0	Reset						

- TIMG_RTC_CALI_TIMEOUT** 表示频率计算是否超时。
- 0: 未超时
 - 1: 超时
- (RO)
- TIMG_RTC_CALI_TIMEOUT_RST_CNT** 配置频率计算超时复位的周期。
- 计数单位: XTAL_CLK。(R/W)
- TIMG_RTC_CALI_TIMEOUT_THRES** 配置 RTC 频率计算定时器的阈值。如果频率计算定时器的值超过此值时触发超时。
- 计数单位: XTAL_CLK。
- (R/W)

Register 10.21. TIMG_INT_ENA_TIMERS_REG (0x0070)

(reserved)																																TIMG_WDT_INT_ENA (reserved) TIMG_TO_INT_ENA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TIMG_TO_INT_ENA 写 1 使能 TIMG_TO_INT 中断。(R/W)

TIMG_WDT_INT_ENA 写 1 使能 TIMG_WDT_INT 中断。(R/W)

Register 10.22. TIMG_INT_RAW_TIMERS_REG (0x0074)

(reserved)																																TIMG_WDT_INT_RAW (reserved) TIMG_TO_INT_RAW																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TIMG_TO_INT_RAW TIMG_TO_INT 中断的原始中断状态位。(R/SS/WTC)

TIMG_WDT_INT_RAW TIMG_WDT_INT 中断的原始中断状态位。(R/SS/WTC)

Register 10.23. TIMG_INT_ST_TIMERS_REG (0x0078)

(reserved)																																TIMG_WDT_INT_ST (reserved) TIMG_TO_INT_ST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

TIMG_TO_INT_ST TIMG_TO_INT 中断的屏蔽中断状态位。(RO)

TIMG_WDT_INT_ST TIMG_WDT_INT 中断的屏蔽中断状态位。(RO)

Register 10.24. TIMG_INT_CLR_TIMERS_REG (0x007C)

(reserved)																										TIMG_WDT_INT_CLR (reserved) TIMG_TO_INT_CLR				
31																										3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

TIMG_TO_INT_CLR 写 1 清除 TIMG_TO_INT 中断。(WT)

TIMG_WDT_INT_CLR 写 1 清除 TIMG_WDT_INT 中断。(WT)

Register 10.25. TIMG_NTIMERS_DATE_REG (0x00F8)

(reserved)				TIMG_NTIMGS_DATE																										
31	28			27																								0		
0	0	0	0	0x2209142																							Reset			

TIMG_NTIMGS_DATE 版本控制寄存器。(R/W)

Register 10.26. TIMG_REGCLK_REG (0x00FC)

TIMG_CLK_EN (reserved)				(reserved)																									
31	30	29	28	27																									
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

TIMG_ETM_EN 配置是否使能定时器的 ETM 任务和事件。

0: 禁用

1: 使能

(R/W)

TIMG_CLK_EN 配置是否使能寄存器的门时钟信号。

0: 强制使能寄存器的时钟

1: 仅在软件读取或写入寄存器时启动时钟。

(R/W)

第 11 章

看门狗定时器 (WDT)

11.1 概述

看门狗定时器是一种硬件定时器，用于检测和修复故障。软件必须定期喂狗（复位），以防超时。系统或软件若出现不可预知的问题（比如软件卡在某个循环或逾期事件中）将无法按时喂狗，造成看门狗超时。因此，看门狗定时器有助于检测、处理系统或软件的错误行为。

如图 11.1-1 所示，ESP32-C5 中有三个数字看门狗定时器：章节 10 定时器组 (TIMG) 描述的两个定时器组中各有一个（称作主系统看门狗定时器，缩写为 MWDT），RTC 模块中有一个（称作 RTC 看门狗定时器，缩写为 RWDT）。数字看门狗在运行期间会经历四个阶段（除非看门狗按时喂狗或者处于关闭状态），每个阶段均可配置单独的超时时间和超时动作，其中 MWDT 支持中断、CPU 复位和内核复位三种超时动作，RWDT 支持中断、CPU 复位、内核复位和系统复位四种超时动作（详见章节 11.2.2.2 阶段与超时动作）。每个阶段的超时时间都可单独设置。

在 flash 引导模式下，RWDT 和定时器组 0 的 MWDT 会默认使能，以检测引导过程中发生的错误，并恢复运行。

ESP32-C5 中还有一个模拟看门狗定时器——超级看门狗 (SWD)。超级看门狗是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。

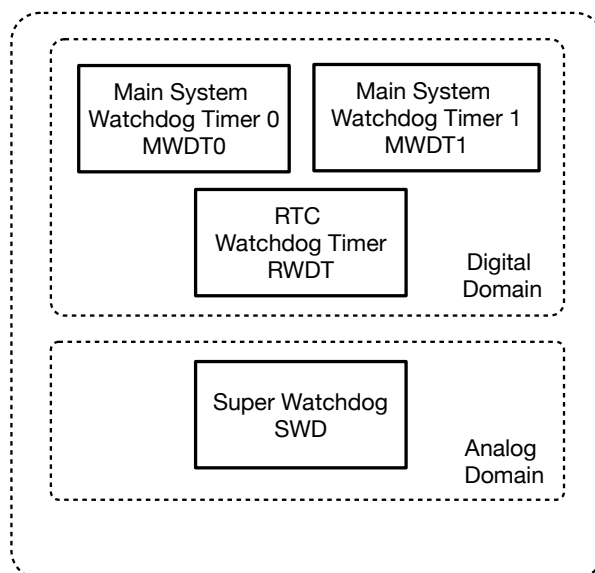


图 11.1-1. 看门狗定时器概览

请注意，本章节提供了看门狗定时器的功能描述，而 MWDT 寄存器部分详见章节 10 定时器组 (TIMG)，RWDT 和 SWD 寄存器部分详见 11.5 寄存器列表。

11.2 数字看门狗定时器

11.2.1 主要特性

看门狗定时器具有如下特性：

- 四个阶段，每个阶段都可配置超时时间和超时动作
- 超时动作
 - MWDT：中断、CPU 复位、内核复位
 - RWDT：中断、CPU 复位、内核复位、系统复位
- 阶段 0 Flash 启动保护：
 - MWDT0：超时触发内核复位
 - RWDT：超时触发系统复位
- 写保护，使能时寄存器仅可读取
- 32 位超时计数器
- 时钟源：
 - MWDT：PLL_F80M_CLK、RC_FAST_CLK 或 XTAL_CLK
 - RWDT：RTC_SLOW_CLK

11.2.2 功能描述

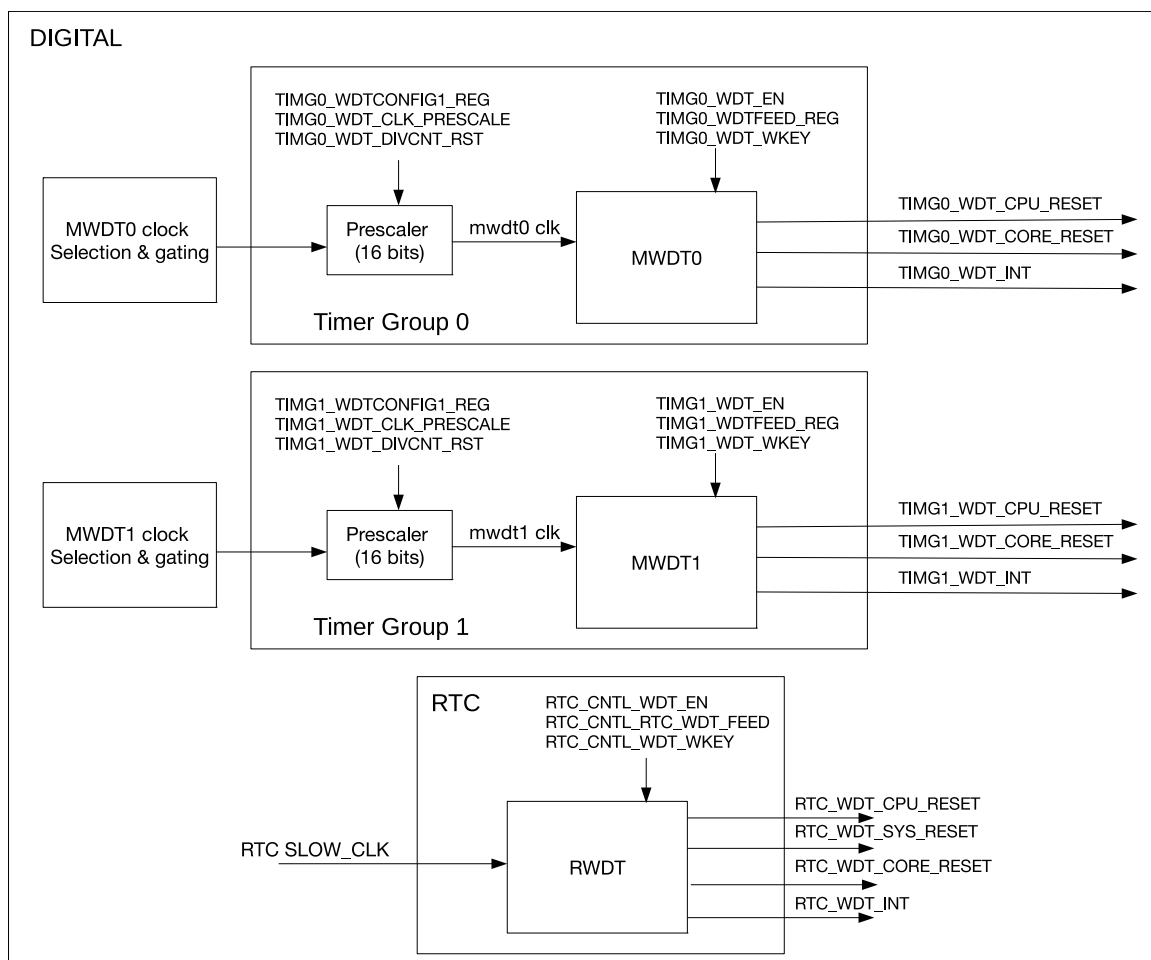


图 11.2-1. ESP32-C5 的数字看门狗定时器

图 11.2-1 为 ESP32-C5 数字系统中的三个看门狗定时器。

11.2.2.1 时钟源与 32 位计数器

看门狗定时器的核心是一个 32 位计数器。

以 MWDT0 为例：

- MWDT0 可通过设置 `PCR_TIMERGROUPO_WDT_CLK_CONF_REG` 寄存器的 `PCR_TGO_WDT_CLK_SEL` 字段选择 `PLL_F80M_CLK` 时钟、`RC_FAST_CLK` 或 `XTAL_CLK`（外部时钟）作为时钟源。
- 将 `PCR_TIMERGROUPO_WDT_CLK_CONF_REG` 寄存器的 `PCR_TGO_WDT_CLK_EN` 字段置 1 开启时钟，清零关闭时钟。时钟经由可配置的 16 位预分频器分频。详见章节 6 复位和时钟 的表 6.2-2。

MWDT 的 16 位预分频器系数可通过 `TIMG_WDTCONFIG1_REG` 寄存器的 `TIMG_WDT_CLK_PRESCALE` 字段配置。`TIMG_WDT_DIVCNT_RST` 字段置位时，预分频器复位，可立即重新配置。

RWDT 直接将 RTC 慢速时钟 `RTC_SLOW_CLK`（详见章节 6 复位和时钟）用作时钟源。

MWDT 和 RWDT 看门狗可分别通过设置 `TIMG_WDT_EN` 和 `RTC_WDT_EN` 字段使能。看门狗使能后，其内部 32 位计数器的值会在每个时钟源周期内累加 1，直到达到该阶段的超时时间（即在该阶段发生超时）。如发生超时，计数器的值会重置为 0，同时看门狗进入下一阶段。如果软件成功喂狗，看门狗定时器会回到阶段 0，并将计

数器的值重置为 0。软件向 `TIMG_WDTFEED_REG` 寄存器内写入任意值，便可为 MDWT 喂狗；向 `RTC_WDT_FEED` 写 1，便可为 RWDT 喂狗。

11.2.2.2 阶段与超时动作

定时器在各阶段可以配置不同的超时时间和对应的超时动作。某一阶段超时会触发对应的超时动作，同时计数器的值被重置为 0，看门狗进入下一阶段。

MWDT 和 RWDT 有四个阶段（称为阶段 0 至阶段 3）。看门狗定时器会循环工作（即从阶段 0 至阶段 3，再回到阶段 0）。

MWDT 每个阶段的超时时间可用 `TIMG_WDTCONFIGi_REG`（*i* 的范围是 2 到 5）寄存器配置，RWDT 的超时时间可用 `RTC_WDT_STGj_HOLD`（*j* 的范围是 0 到 3）字段配置。

值得注意的是，RWDT 在阶段 0 的超时时间 (T_{hold0}) 受 eFuse 寄存器 `EFUSE_RD_REPEAT_DATA0_REG` 的 `EFUSE_WDT_DELAY_SEL` 字段和 `RTC_WDT_STG0_HOLD` 字段共同影响，关系如下：

$$T_{hold0} = RTC\_WDT\_STG0\_HOLD \ll (EFUSE\_WDT\_DELAY\_SEL + 1)$$

其中， $\ll$ 为左移运算符。例如，`RTC_WDT_STG0_HOLD` 配置为 100，`EFUSE_WDT_DELAY_SEL` 为 1，则 T_{hold0} 为 400 个时钟周期。

如某个阶段超时，下列超时动作之一将会执行：

表 11.2-1. 超时动作

超时动作	描述
中断	触发中断
CPU 复位	详见章节 6 复位和时钟 > 小节 6.1.3 特性
内核复位	
系统复位	
关闭	对系统不产生影响

MWDT 所有阶段的超时动作均在 `TIMG_WDTCONFIG0_REG` 寄存器中配置。RWDT 的超时动作可在 `RTC_WDT_CONFIG0_REG` 寄存器配置。

11.2.2.3 写保护

看门狗定时器对于检测和处理系统或软件错误而言至关重要，不应轻易关闭（例如，因写寄存器位置错误而误将看门狗关闭）。因此，MWDT 和 RWDT 引入写保护机制，防止看门狗因无意的写操作而被关闭或篡改。

写保护机制通过每个看门狗定时器的写密钥字段运行（MWDT 看门狗使用 `TIMG_WDT_WKEY`，RWDT 看门狗使用 `RTC_WDT_WKEY`）。必须向看门狗定时器的写密钥字段写入 0x50D83AA1，才能修改其它看门狗寄存器。如果写密钥字段的值不是 0x50D83AA1，任何试图向看门狗定时器寄存器（除了向写密钥字段本身）写值的操作都会被忽略。推荐按以下步骤访问看门狗定时器：

1. 将 0x0D83AA1 写入看门狗定时器的写密钥字段，关闭写保护。
2. 根据需要修改看门狗，如喂狗或改变配置。
3. 向看门狗定时器的写密钥字段上写入除 0x50D83AA1 以外的任意值，重新使能写保护。

11.2.2.4 Flash 引导保护

在 flash 引导模式下，MWDT0 和 RWDT 会默认使能。MWDT0 的阶段 0 的默认超时动作为内核复位（复位主系统）。RWDT 的阶段 0 超时动作为系统复位（复位主系统和 RTC）。引导后，应将 `TIMG_WDT_FLASHBOOT_MOD_EN` 和 `RTC_WDT_FLASHBOOT_MOD_EN` 位清零，分别关闭 MWDT0 和 RWDT 的 flash 引导保护。然后，软件可以配置 MWDT0 和 RWDT。

11.3 超级看门狗定时器

超级看门狗 (SWD) 是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。SWD 包含一个看门狗电路，需在每个超时阶段（约不足一秒）至少喂狗一次。该电路会在看门狗超时时间约 100 ms 之前发送 `WD_INTR` 信号提醒系统喂狗。

如果系统不回应 SWD 的喂狗请求，看门狗超时，SWD 会产生系统电平信号 `SWD_RSTB`，复位芯片上的整个数字电路（即超时动作中的系统复位）。

SWD 的时钟源固定，不可选择。

11.3.1 主要特性

SWD 具有如下特性：

- 独立于数字电路的模拟看门狗，可以工作数字电路时钟和电压异常的环境
- 用中断提醒 SWD 即将超时
- 软件有多种专用的方法喂 SWD，让 SWD 监控整个操作系统的工作状态

11.3.2 SWD 控制器

11.3.2.1 结构

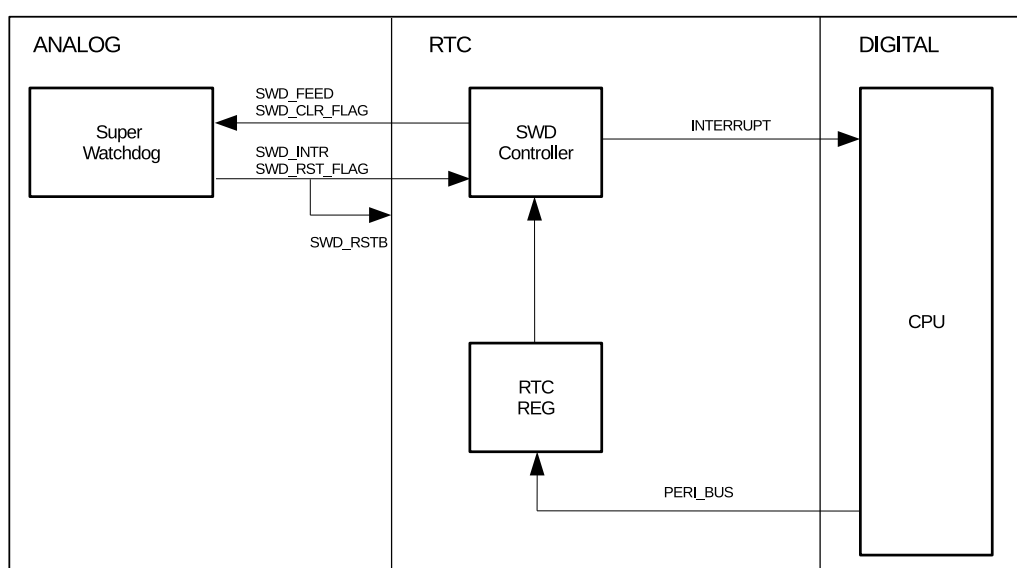


图 11.3-1. SWD 控制器结构

11.3.2.2 工作流程

正常状态下：

- SWD 控制器收到 SWD 的喂狗请求。
- SWD 控制器向 HP CPU 发送中断。
- HP CPU 通过置位 `RTC_WDT_SWD_FEED` 直接喂狗。
- CPU 喂狗时，需要先向 `RTC_WDT_SWD_WKEY` 写 `0x50D83AA1` 关闭 SWD 控制器的写保护。这样做可以防止系统在数字电路异常状态下运行时误喂 SWD。
- 如将 `RTC_WDT_SWD_AUTO_FEED_EN` 置 1，SWD 控制器也可配置为在不需要 CPU 干预的情况下接收 SWD 发出的中断喂 SWD。

复位后：

- 可查看 `LP_CLKRST_RESET_CAUSE[4:0]` 获知 CPU 复位原因。
如 `LP_CLKRST_RESET_CAUSE[4:0] == 0x12`，则表示上一次复位的原因是 SWD 复位。
- 置位 `RTC_WDT_SWD_RST_FLAG_CLR` 清除 SWD 复位标志。

11.4 中断

看门狗定时器中断，请前往章节 10 定时器组 (TIMG) 的第 10.6 节 中断 查看。

11.5 寄存器列表

本小节的地址均为相对于低功耗看门狗定时器 (RTC_WDT) 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
<code>RTC_WDT_CONFIG0_REG</code>	RWDT 配置寄存器	0x0000	R/W
<code>RTC_WDT_CONFIG1_REG</code>	配置 RWDT 阶段 0 的超时时间	0x0004	R/W
<code>RTC_WDT_CONFIG2_REG</code>	配置 RWDT 阶段 1 的超时时间	0x0008	R/W
<code>RTC_WDT_CONFIG3_REG</code>	配置 RWDT 阶段 2 的超时时间	0x000C	R/W
<code>RTC_WDT_CONFIG4_REG</code>	配置 RWDT 阶段 2 的超时时间	0x0010	R/W
<code>RTC_WDT_FEED_REG</code>	配置喂狗 RWDT	0x0014	WT
<code>RTC_WDT_WPROTECT_REG</code>	配置锁住 RWDT 的配置寄存器	0x0018	R/W
<code>RTC_WDT_SWD_CONFIG_REG</code>	SWD 配置寄存器	0x001C	varies
<code>RTC_WDT_SWD_WPROTECT_REG</code>	配置锁住 SWD 的配置寄存器	0x0020	R/W
<code>RTC_WDT_INT_RAW_REG</code>	配置是否产生 WDT 超时中断	0x0024	R/ WTC/ SS
<code>RTC_WDT_INT_ST_REG</code>	WDT 中断状态寄存器	0x0028	RO
<code>RTC_WDT_INT_ENA_REG</code>	WDT 中断使能寄存器	0x002C	R/W

名称	描述	地址	访问
RTC_WDT_INT_CLR_REG	WDT 中断清除寄存器	0x0030	WT
RTC_WDT_DATE_REG	版本控制寄存器	0x03FC	R/W

11.6 寄存器

MWDT 寄存器是定时器组模块的一部分，在章节 [10 定时器组 \(TIMG\)](#) 的第 [10.8 节 寄存器列表](#) 中有详细描述。本小节包含 RWD 和 SWD 相关寄存器介绍，其地址均为相对于低功耗看门狗定时器 (RTC_WDT) 基地址的地址偏移量（相对地址），具体基地址请见章节 [4 系统和存储器](#) 中的表 [4.3-2](#)。

Register 11.1. RTC_WDT_CONFIG0_REG (0x0000)

[illegible]

RTC_WDT_PAUSE_IN_SLP 配置芯片处于 light sleep 或 deep sleep 模式时是否禁用 RWDT。

0: 使能

1: 禁用

(R/W)

RTC_WDT_PROCPU_RESET_EN 配置是否使能 RWDT 进行 CPU 复位。

0: 禁用

1: 使能

(R/W)

RTC_WDT_FLASHBOOT_MOD_EN 配置在 SPI boot 模式下是否自动使能 RWDT。

0: 禁用

1: 使能

(R/W)

RTC_WDT_SYS_RESET_LENGTH 配置内核复位时间。

单位: LP_DYN_FAST_CLK

(R/W)

RTC_WDT_CPU_RESET_LENGTH 配置 CPU 复位时间。

单位: LP DYN FAST CLK

(R/W)

RTC WDT STG3 配置阶段 3 的超时复位动作。

0: 无效

1: 中断

2: CPU 复位

3: 内核复位

4: 系统复位

(R/W)

见下页...

Register 11.1. RTC_WDT_CONFIG0_REG (0x0000)

接上页...

RTC_WDT_STG2 配置阶段 2 的超时复位动作。

- 0: 无效
 - 1: 中断
 - 2: CPU 复位
 - 3: 内核复位
 - 4: 系统复位
- (R/W)

RTC_WDT_STG1 配置阶段 1 的超时复位动作。

- 0: 无效
 - 1: 中断
 - 2: CPU 复位
 - 3: 内核复位
 - 4: 系统复位
- (R/W)

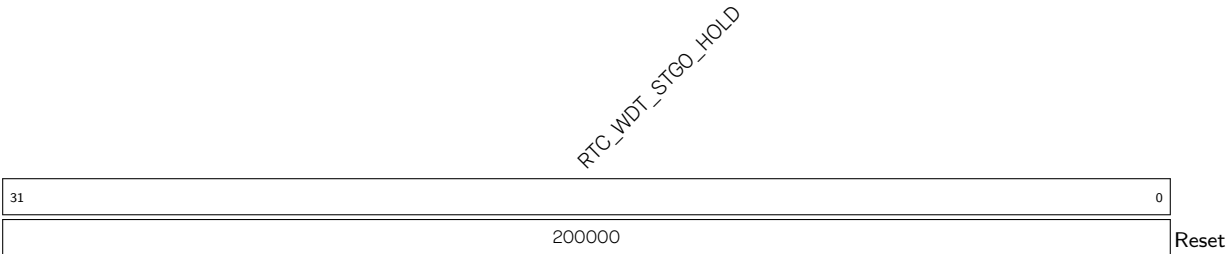
RTC_WDT_STG0 配置阶段 0 的超时复位动作。

- 0: 无效
 - 1: 中断
 - 2: CPU 复位
 - 3: 内核复位
 - 4: 系统复位
- (R/W)

RTC_WDT_EN 配置是否使能 RWDT。

- 0: 禁用 RWDT
 - 1: 使能 RWDT
- (R/W)

Register 11.2. RTC_WDT_CONFIG1_REG (0x0004)



RTC_WDT_STG0_HOLD 配置阶段 0 的超时时间。

- 单位: LP_DYN_SLOW_CLK
- (R/W)

Register 11.3. RTC_WDT_CONFIG2_REG (0x0008)

RTC_WDT_STG1_HOLD	
31	0
80000	
Reset	

RTC_WDT_STG1_HOLD 配置阶段 1 的超时时间。
单位: LP_DYN_SLOW_CLK
(R/W)

Register 11.4. RTC_WDT_CONFIG3_REG (0x000C)

RTC_WDT_STG2_HOLD	
31	0
0x000fff	
Reset	

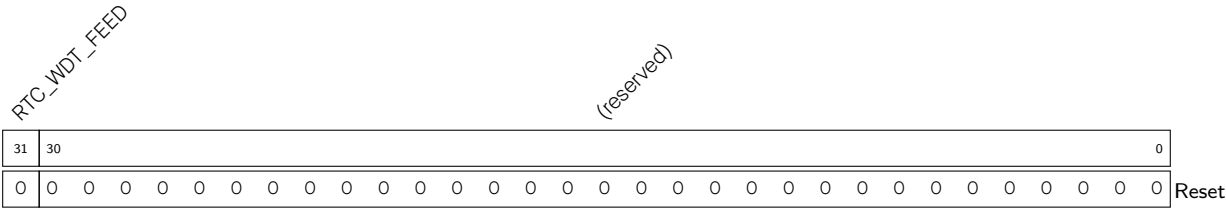
RTC_WDT_STG2_HOLD 配置阶段 2 的超时时间。
单位: LP_DYN_SLOW_CLK
(R/W)

Register 11.5. RTC_WDT_CONFIG4_REG (0x0010)

RTC_WDT_STG3_HOLD	
31	0
0x000fff	
Reset	

RTC_WDT_STG3_HOLD 配置阶段 3 的超时时间。
单位: LP_DYN_SLOW_CLK
(R/W)

Register 11.6. RTC_WDT_FEED_REG (0x0014)

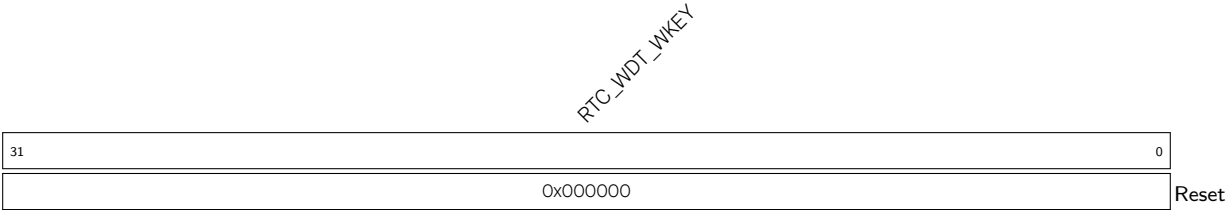


RTC_WDT_FEED 配置此位喂狗 RWDt。

0: 无效

1: 喂狗 (WT)

Register 11.7. RTC_WDT_WPROTECT_REG (0x0018)



RTC_WDT_WKEY 配置该寄存器来锁住或解锁 RWDt 的配置寄存器。

0x50D83AA1: 解锁 RWDt 的配置寄存器

Others value: 锁住 RWDt 的配置寄存器, 软件不能修改 RWDt 的配置 (R/W)

Register 11.8. RTC_WDT_SWD_CONFIG_REG (0x001C)

RTC_WDT_SWD_FEED		RTC_WDT_SWD_DISABLE		RTC_WDT_SWD_SIGNAL_WIDTH				RTC_WDT_SWD_RST_FLAG_CLR				RTC_WDT_SWD_AUTO_FEED_EN				(reserved)														RTC_WDT_SWD_RESET_FLAG									
31	30	29																20	19	18	17																1	0	
0	0	300															0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_WDT_SWD_RESET_FLAG 表示 SWD 是否产生复位信号。

0: 否

1: 是

(RO)

RTC_WDT_SWD_AUTO_FEED_EN 配置此位使能硬件自动喂狗。

0: 禁止硬件自动喂狗

1: 使能硬件自动喂狗

(R/W)

RTC_WDT_SWD_RST_FLAG_CLR 配置此位清除 SWD 复位标志。

0: 无效

1: 清除复位标志

(WT)

RTC_WDT_SWD_SIGNAL_WIDTH 配置输出到模拟电路的信号长度。

单位: LP_DYN_FAST_CLK (R/W)

RTC_WDT_SWD_DISABLE 配置此位禁用 SWD。

0: 使能

1: 禁用

(R/W)

RTC_WDT_SWD_FEED 配置此位喂狗 SWD。

0: 无效

1: 喂狗 SWD

(WT)

Register 11.9. RTC_WDT_SWD_WPROTECT_REG (0x0020)

RTC_WDT_SWD_WKEY																															
31																															0
0x000000																															
Reset																															

RTC_WDT_SWD_WKEY 配置该寄存器来锁住或解锁 SWDT 的配置寄存器。

0x50D83AA1: 解锁 SWD 的配置寄存器

Others value: 锁住 SWD 的配置寄存器, 软件不能修改 SWD 的配置

(R/W)

Register 11.10. RTC_WDT_INT_RAW_REG (0x0024)

RTC_WDT_SWD_INT_RAW																												(reserved)				
31	30	29																														0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_WDT_SWD_INT_RAW RTC_WDT_SWD_INT 的原始中断状态。(R/WTC/SS)

RTC_WDT_INT_RAW RTC_WDT_INT 的原始中断状态。(R/WTC/SS)

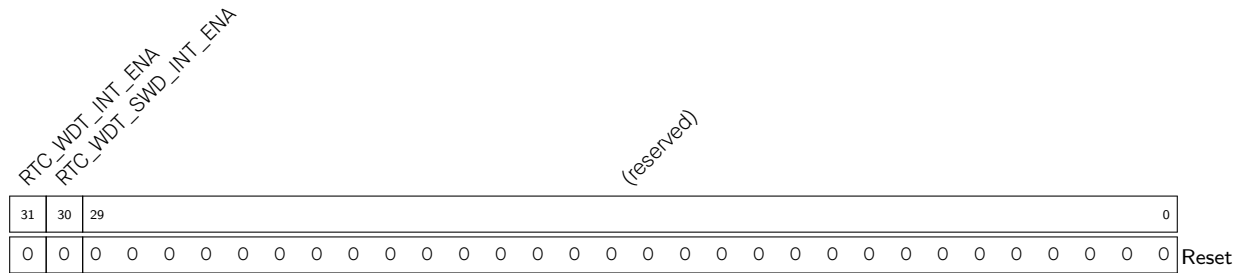
Register 11.11. RTC_WDT_INT_ST_REG (0x0028)

RTC_WDT_SWD_INT_ST																												(reserved)				
31	30	29																														0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_WDT_SWD_INT_ST RTC_WDT_SWD_INT 的屏蔽中断状态。(RO)

RTC_WDT_INT_ST RTC_WDT_INT 的屏蔽中断状态。(RO)

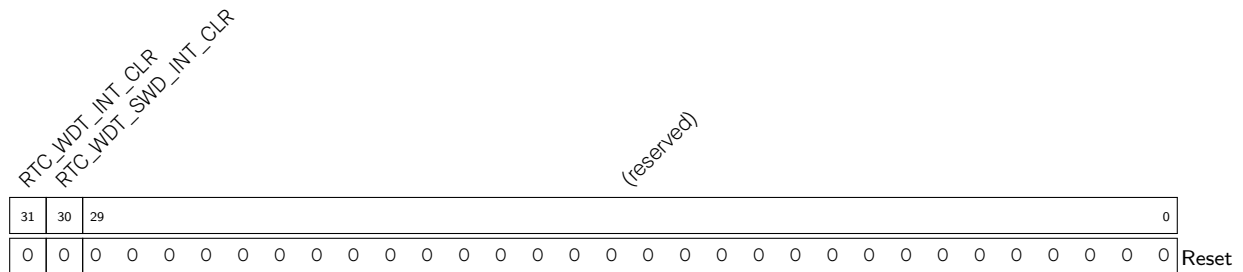
Register 11.12. RTC_WDT_INT_ENA_REG (0x002C)



RTC_WDT_SWD_INT_ENA 写1使能 RTC_WDT_SWD_INT. (R/W)

RTC_WDT_INT_ENA 写1使能 RTC_WDT_INT. (R/W)

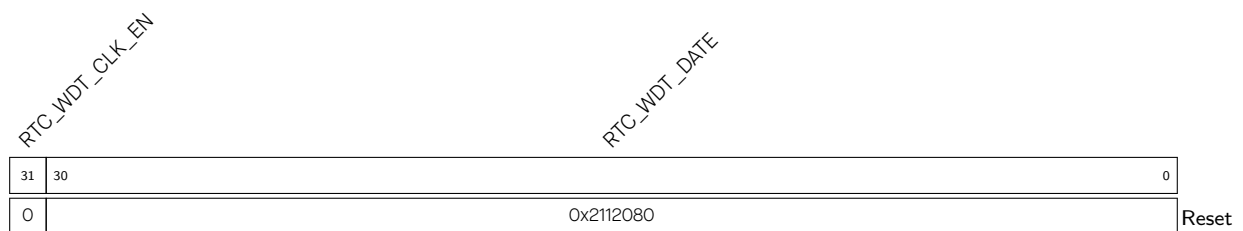
Register 11.13. RTC_WDT_INT_CLR_REG (0x0030)



RTC_WDT_SWD_INT_CLR 写1清除 RTC_WDT_SWD_INT. (WT)

RTC_WDT_INT_CLR 写 1 清除 RTC_WDT_INT. (WT)

Register 11.14. RTC_WDT_DATE_REG (0x03FC)



RTC_WDT_DATE 版本控制寄存器 (R/W)

RTC_WDT_CLK_EN 保留 (R/W)

第 12 章

实时时钟定时器

12.1 概述

实时时钟定时器（RTC 定时器）是一个 48 位的可读计时器，能够在芯片的任意功耗模式下工作，当 HP 系统中的定时器外设不可用时，继续为系统提供定时器服务。同时还支持配置定时器中断、记录系统中特定事件发生的时刻。

12.2 主要特性

- 48 位的计时器
- 触发特定事件时可记录事件发生的时刻，支持的特定事件有：
 - HP 系统复位
 - CPU 进入 stall 状态
 - CPU 退出 stall 状态
 - 晶振时钟开启
 - 晶振时钟关闭
- 通过配置寄存器触发 RTC 定时器记录当前时间
- 支持缓存最近两次特定事件的发生的时间
- 支持在目标时刻产生中断，目标时刻可配置，可同时配置两个目标时刻
- 除 LP 系统上电复位外的其余任何复位/睡眠均不会使 RTC 定时器停止或复位
- 可用作唤醒源（详见章节 4 低功耗管理 [\[to be added later\]](#)）

12.3 功能描述

- 48 位计时器
 - RTC 定时器的实现基于一个 48 位的计时器，其驱动时钟是由位于 Always-On 电源域的 RTC_SLOW_CLK 提供，采用不间断的循环计时（LP 系统复位除外），当 48 位计时器溢出时会产生溢出中断。关于电源域的更多信息，请查看章节 4 低功耗管理 [\[to be added later\]](#)。
- 记录特定事件的发生时刻
 - RTC 定时器支持记录三类特定事件的发生时刻：
 - * HP 系统复位

- * CPU 进入或退出 stall 状态
- * 晶振时钟开启或关闭
- 三类事件都有各自独立的配置寄存器，当配置寄存器使能后，若发生对应的事件，产生的硬件脉冲就会触发 RTC 定时器记录当前时间。对应的配置寄存器见下表：

表 12.3-1. 配置 RTC 定时器记录特定事件的发生时刻

配置寄存器	描述
RTC_TIMER_MAIN_TIMER_SYS_RST	使能 RTC 定时器记录系统复位时刻。
RTC_TIMER_MAIN_TIMER_SYS_STALL	使能 RTC 定时器记录 CPU 进入或退出 stall 状态时刻。
RTC_TIMER_MAIN_TIMER_XTAL_OFF	使能 RTC 定时器记录晶振时钟开启或关闭时刻。

- 通过配置寄存器触发 RTC 定时器记录当前时间
 - 除了上述三类特定事件发生时可以触发 RTC 定时器记录当前时间，还可以通过配置寄存器 `RTC_TIMER_MAIN_TIMER_UPDATE` 触发 RTC 定时器记录当前时间。
 - 配置 `RTC_TIMER_MAIN_TIMER_UPDATE` 寄存器会产生一个脉冲信号，触发 RTC 定时器记录当前时间。
- 目标时刻产生计数中断
 - RTC 定时器支持同时配置两个目标时刻，分别为目标时刻 0 和目标时刻 1，当计时器到达目标时刻 0 时会触发 `RTC_TIMER_CMPO_INT` 中断，当计时器到达目标时刻 1 时会触发 `RTC_TIMER_CMP1_INT` 中断。
 - 目标时刻的配置寄存器为：

表 12.3-2. RTC 定时器目标时刻配置

配置寄存器	描述
RTC_TIMER_MAIN_TIMER_TAR_EN n	$n = 0$ 或 1 ，配置目标时刻使能。
RTC_TIMER_MAIN_TIMER_TAR_LOW n	$n = 0$ 或 1 ，配置目标时刻的低 32 位。
RTC_TIMER_MAIN_TIMER_TAR_HIGH n	$n = 0$ 或 1 ，配置目标时刻的高 16 位。

- 读取缓存时刻
 - RTC 定时器有两组寄存器用于缓存特定事件的发生时刻，分别为寄存器组 0 和寄存器组 1；
 - 寄存器组 0 包括寄存器 `RTC_TIMER_MAIN_BUFO_LOW_REG` 和 `RTC_TIMER_MAIN_BUFO_HIGH_REG`，用于记录 RTC 定时器在当前触发下的计数值；
 - 寄存器组 1 包括寄存器 `RTC_TIMER_MAIN_BUF1_LOW_REG` 和 `RTC_TIMER_MAIN_BUF1_HIGH_REG`，用于记录 RTC 定时器在上一次触发下的计数值；
 - 每次有新的触发，上一次触发时的记录将从寄存器组 0 移至寄存器组 1（寄存器组 1 中之前的记录将被覆盖），而本次触发的记录将存储在寄存器组 0。因此，RTC 定时器最多可同时记录两次触发的值。

12.4 事件任务矩阵功能

在 ESP32-C5 中，RTC 定时器支持 ETM 功能，即可以通过 RTC 定时器的 ETM 事件触发任意外设的 ETM 任务。RTC 定时器不支持 ETM 任务。关于 ETM 更多详细信息，请参考[章节 8 事件任务矩阵 \(ETM\)](#)。这里仅介绍与 RTC 定时器相关的 ETM 事件。

RTC 定时器可产生的 ETM 事件有：

- RTC_EVT_CMP：表示 RTC 定时器计时到达寄存器 [RTC_TIMER_MAIN_TIMER_TAR_LOW0](#) 和 [RTC_TIMER_MAIN_TIMER_TAR_HIGH0](#) 所配置的目标时刻 0。
- RTC_EVT_OVF：表示 48 位计时器计数溢出。
- RTC_EVT_TICK：表示 RTC 计时器递增 1 时触发的事件。

12.5 中断

ESP32-C5 的 RTC 定时器可以生成以下中断信号并发送给[中断矩阵](#)。

- LP_TIMER_REG_0_INTR（该中断供 HP CPU 使用）
- LP_TIMER_REG_1_INTR（该中断供 LP CPU 使用）

这些中断信号由 RTC 定时器的内部中断源生成。表 [12.5-1](#) 列出了 RTC 定时器的内部中断源、中断触发条件、以及生成的中断信号。

表 12.5-1. RTC 定时器的内部中断源

内部中断源	触发条件	中断信号
RTC_TIMER_CMPO_INT	RTC 定时器计时到达目标时刻 0	LP_TIMER_REG_0_INTR
RTC_TIMER_CMP1_INT	RTC 定时器计时到达目标时刻 1	LP_TIMER_REG_1_INTR
RTC_TIMER_OVERFLOW_INT	48 位计时器计数溢出	LP_TIMER_REG_0_INTR LP_TIMER_REG_1_INTR

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考[章节 3 中断矩阵 \[to be added later\]](#) > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见[章节 中断配置寄存器](#)。具体寄存器名称请查看[12.6 寄存器列表](#)。

12.6 寄存器列表

本小节的所有地址均为相对于 RTC 定时器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

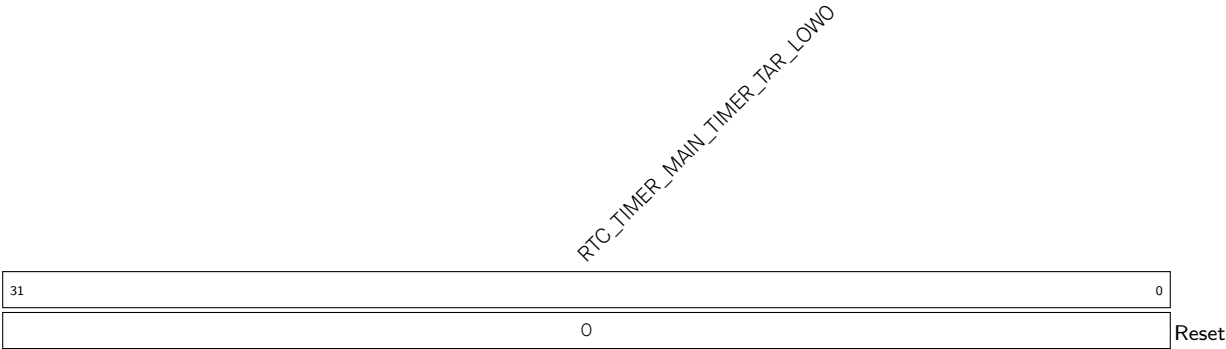
名称	描述	地址	访问
配置寄存器			
RTC_TIMER_TARO_LOW_REG	配置目标时刻 0 的低 32 位	0x0000	R/W
RTC_TIMER_TARO_HIGH_REG	配置目标时刻 0 的高 16 位	0x0004	varies
RTC_TIMER_TAR1_LOW_REG	配置目标时刻 1 的低 32 位	0x0008	R/W
RTC_TIMER_TAR1_HIGH_REG	配置目标时刻 1 的高 16 位	0x000C	varies
RTC_TIMER_UPDATE_REG	配置使能 RTC 定时器锁存当前值	0x0010	varies
RTC_TIMER_MAIN_BUFO_LOW_REG	RTC 定时器计数器缓存值 0 的低 32 位	0x0014	RO
RTC_TIMER_MAIN_BUFO_HIGH_REG	RTC 定时器计数器缓存值 0 的高 16 位	0x0018	RO
RTC_TIMER_MAIN_BUF1_LOW_REG	RTC 定时器计数器缓存值 1 的低 32 位	0x001C	RO
RTC_TIMER_MAIN_BUF1_HIGH_REG	RTC 定时器计数器缓存值 1 的高 16 位	0x0020	RO
中断寄存器			
RTC_TIMER_INT_RAW_REG	计时器到达目标时刻 0 的原始中断状态寄存器	0x0028	R/ WTC/ SS
RTC_TIMER_INT_ST_REG	计时器到达目标时刻 0 的中断状态寄存器	0x002C	RO
RTC_TIMER_INT_ENA_REG	计时器到达目标时刻 0 的中断使能寄存器	0x0030	R/W
RTC_TIMER_INT_CLR_REG	计时器到达目标时刻 0 的中断清除寄存器	0x0034	WT
RTC_TIMER_LP_INT_RAW_REG	计时器到达目标时刻 1 的原始中断状态寄存器	0x0038	R/ WTC/ SS
RTC_TIMER_LP_INT_ST_REG	计时器到达目标时刻 1 的中断状态寄存器	0x003C	RO
RTC_TIMER_LP_INT_ENA_REG	计时器到达目标时刻 1 的中断使能寄存器	0x0040	R/W
RTC_TIMER_LP_INT_CLR_REG	计时器到达目标时刻 1 的中断清除寄存器	0x0044	WT
版本寄存器			
RTC_TIMER_DATE_REG	版本控制寄存器	0x03FC	R/W

12.7 寄存器

本小节的所有地址均为相对于 RTC 定时器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

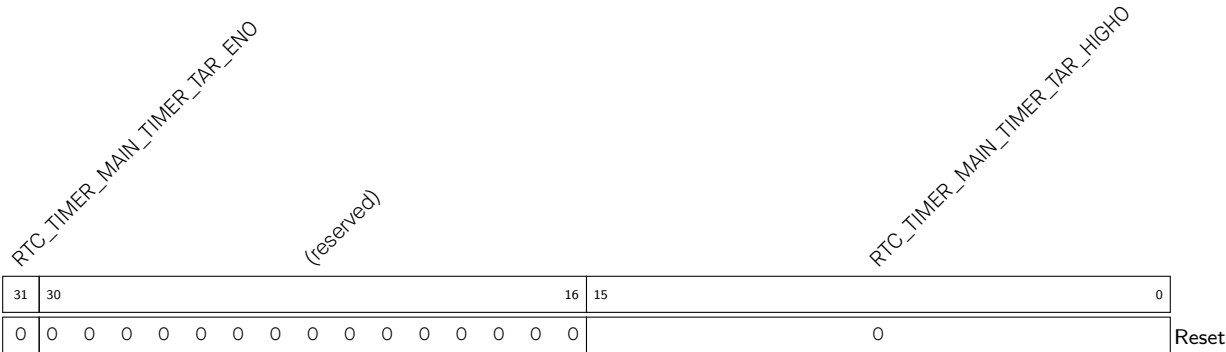
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 12.1. RTC_TIMER_TARO_LOW_REG (0x0000)



RTC_TIMER_MAIN_TIMER_TAR_LOW0 配置计时器的目标时刻 0 的低 32 位。(R/W)

Register 12.2. RTC_TIMER_TARO_HIGH_REG (0x0004)

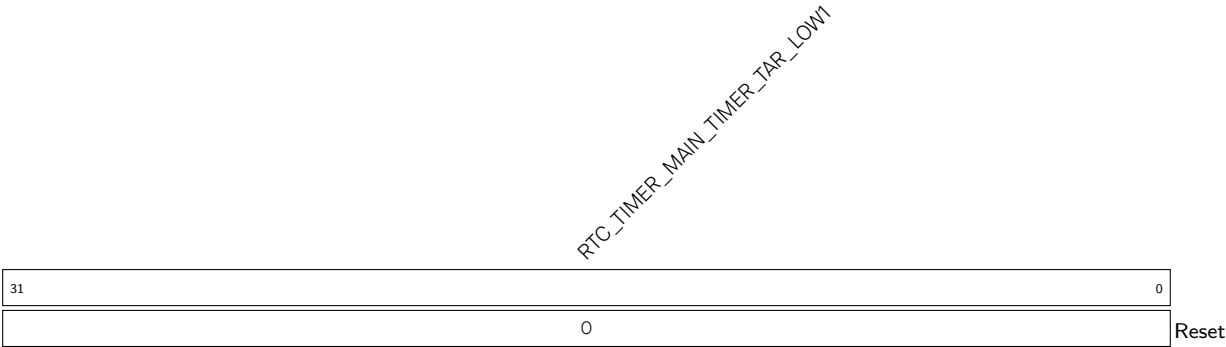


RTC_TIMER_MAIN_TIMER_TAR_HIGH0 配置计时器的目标时刻 0 的高 16 位。(R/W)

RTC_TIMER_MAIN_TIMER_TAR_EN0 配置是否使能计时器的目标时刻 0。

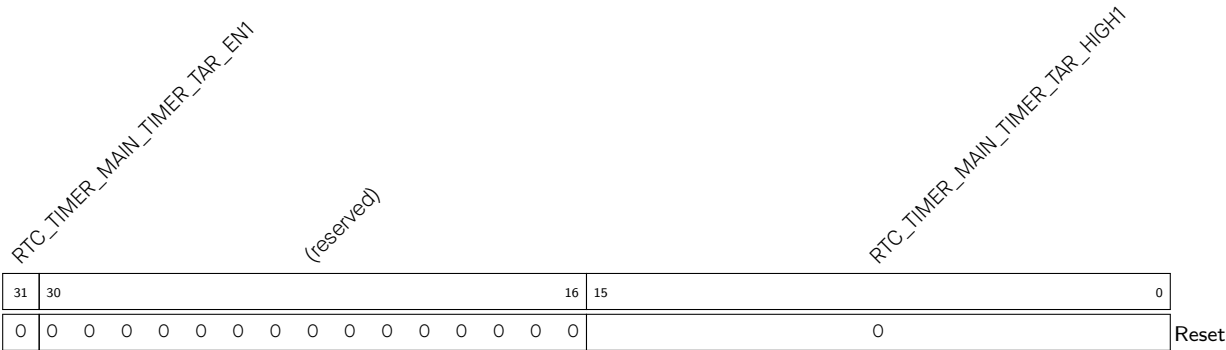
- 0: 不使能
 - 1: 使能
- (WT)

Register 12.3. RTC_TIMER_TAR1_LOW_REG (0x0008)



RTC_TIMER_MAIN_TIMER_TAR_LOW1 配置计时器的目标时刻 1 的低 32 位。(R/W)

Register 12.4. RTC_TIMER_TAR1_HIGH_REG (0x000C)

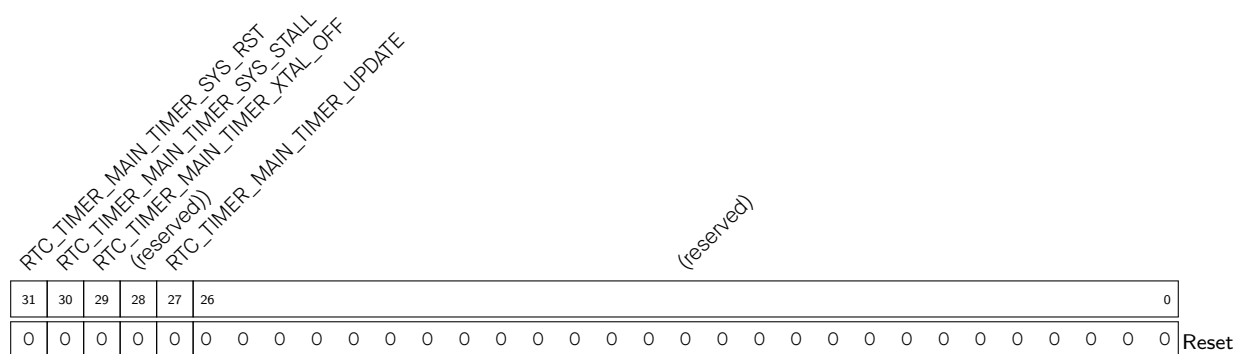


RTC_TIMER_MAIN_TIMER_TAR_HIGH1 配置计时器的目标时刻 1 的高 16 位。(R/W)

RTC_TIMER_MAIN_TIMER_TAR_EN1 计时器的目标时刻 1 使能。

- 0: 不使能
 - 1: 使能
- (WT)

Register 12.5. RTC_TIMER_UPDATE_REG (0x0010)



RTC TIMER MAIN TIMER UPDATE 配置是否通过软件配置记录当前时间。

0: 不记录

1: 记录

(WT)

RTC_TIMER_MAIN_TIMER_XTAL_OFF 配置是否使能 RTC 定时器记录晶振时钟开启或关闭时刻。

0: 不使能

1: 使能

(R/W)

RTC_TIMER_MAIN_TIMER_SYS_STALL 配置是否使能 RTC 定时器记录 CPU 进入或退出 stall 状态时刻。

0: 不使能

1: 使能

(R/W)

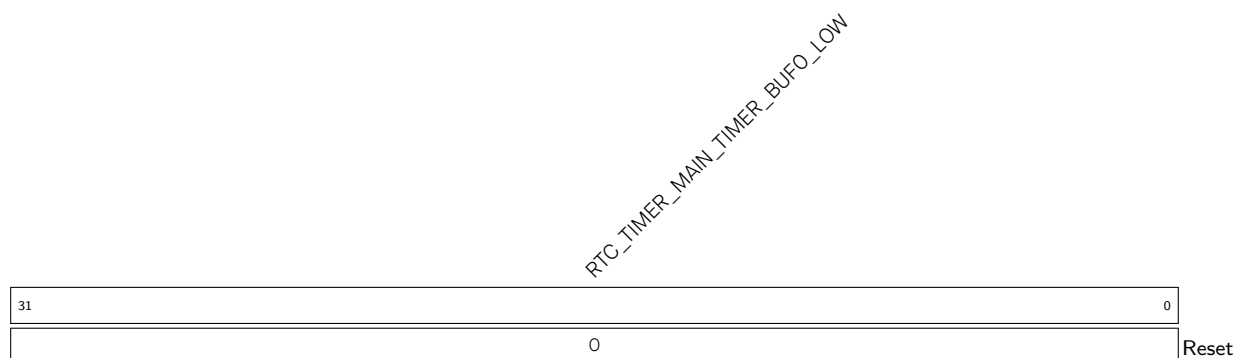
RTC_TIMER_MAIN_TIMER_SYS_RST 配置是否使能 RTC 定时器记录系统复位时刻。

0: 不使能

1: 使能

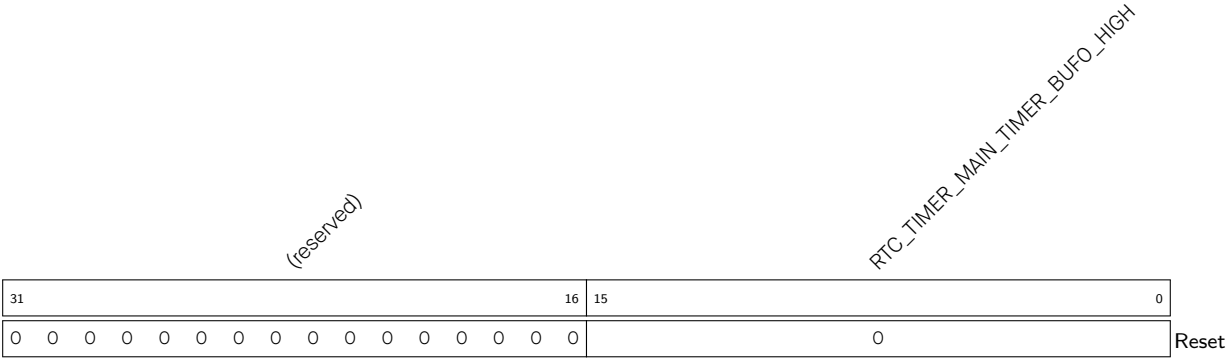
(R/W)

Register 12.6. RTC_TIMER_MAIN_BUFO_LOW_REG (0x0014)



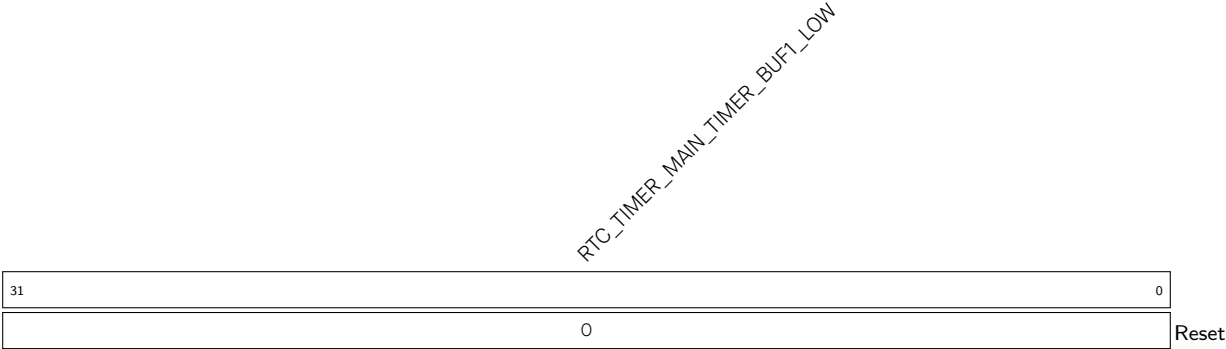
RTC_TIMER_MAIN_TIMER_BUFO_LOW 表示计时器缓存值 0 的低 32 位。(RO)

Register 12.7. RTC_TIMER_MAIN_BUFO_HIGH_REG (0x0018)



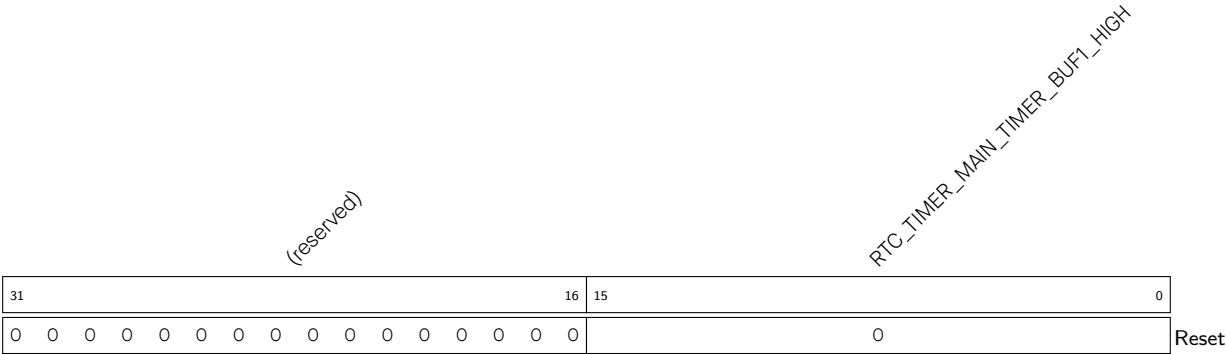
RTC_TIMER_MAIN_TIMER_BUFO_HIGH 表示计时器缓存值 0 的高 16 位。(RO)

Register 12.8. RTC_TIMER_MAIN_BUF1_LOW_REG (0x001C)



RTC_TIMER_MAIN_TIMER_BUF1_LOW 表示计时器缓存值 1 的低 32 位。(RO)

Register 12.9. RTC_TIMER_MAIN_BUF1_HIGH_REG (0x0020)



RTC_TIMER_MAIN_TIMER_BUF1_HIGH 表示计时器缓存值 1 的高 16 位。(RO)

Register 12.10. RTC_TIMER_INT_RAW_REG (0x0028)

Diagram illustrating the structure of the RTC_TMR register (32 bits):

- Bits 31 and 30: RTC_TIMER_SOC_WAKEUP_INT_RAW
- Bit 29: RTC_TIMER_OVERFLOW_RAW
- Bits 0-28: (reserved)

RTC_TIMER_OVERFLOW_RAW RTC_TIMER_OVERFLOW_INT (HP CPU) 的原始中断状态。(R/WTC/SS)

RTC TIMER SOC WAKEUP INT RAW RTC TIMER CMPO INT 原始中断状态。(R/WTC/SS)

Register 12.11. RTC_TIMER_INT_ST_REG (0x002C)

Diagram of the 32-bit register structure for `RTC_TMR_OFLW_ST`:

- Bits 31-30: `RTC_TIMER_SOC_WAKEUP_INT_ST` (2 bits)
- Bits 29-0: `RTC_TIMER_OVERFLOW_ST` (30 bits)
- Bits 31-0: (reserved) (32 bits)

RTC_TIMER_OVERFLOW_ST RTC_TIMER_OVERFLOW_INT (HP CPU) 的屏蔽中断状态。(RO)

RTC_TIMER_SOC_WAKEUP_INT_ST **RTC_TIMER_CMPO_INT** 的屏蔽中断状态。(RO)

Register 12.12. RTC_TIMER_INT_ENA_REG (0x0030)

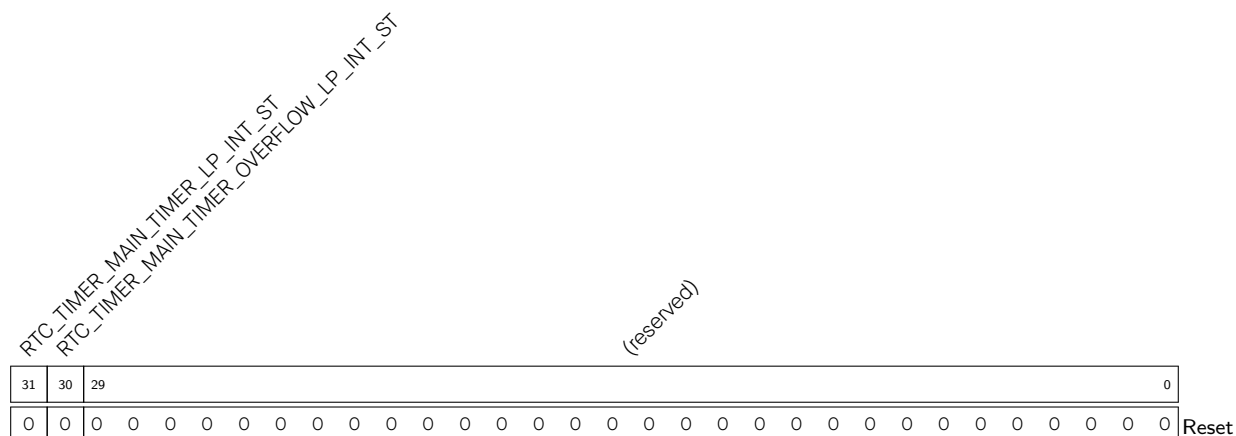
Diagram illustrating the structure of the RTC_TMR register (32 bits wide):

- Bit 31: RTC_TIMER_OVERFLOW_ENA
- Bit 30: RTC_TIMER_WAKEUP_INT_ENA
- Bits 29 through 0: (reserved)

RTC_TIMER_OVERFLOW_ENA 写1使能 RTC_TIMER_OVERFLOW_INT (HP CPU) 中断。(R/W)

RTC_TIMER_SOC_WAKEUP_INT_ENA 写1使能 RTC_TIMER_CMPO_INT 中断。(R/W)

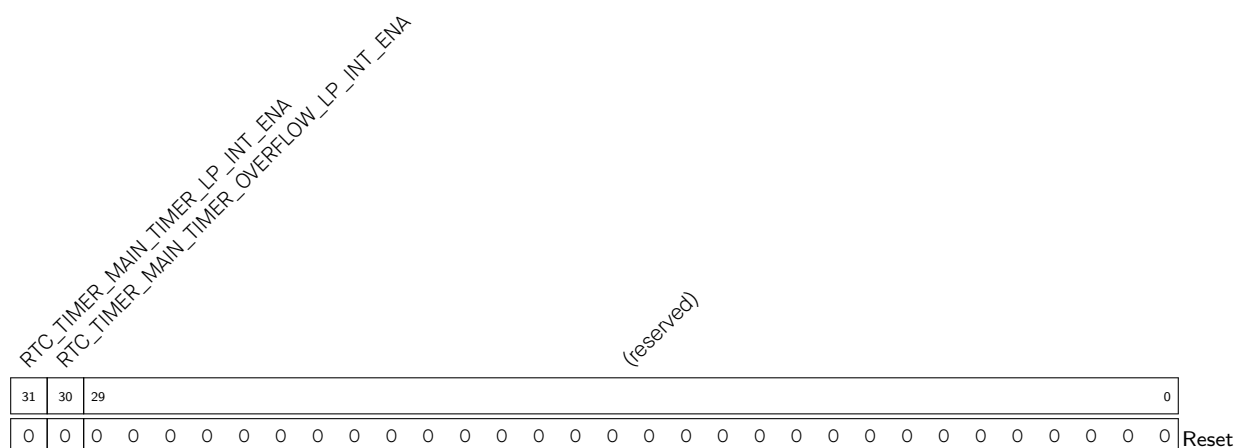
Register 12.15. RTC_TIMER_LP_INT_ST_REG (0x003C)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ST RTC_TIMER_OVERFLOW_INT (LP CPU) 的屏蔽状态中断。(RO)

RTC_TIMER_MAIN_TIMER_LP_INT_ST **RTC_TIMER_CMP1_INT** 屏蔽中断状态。(RO)

Register 12.16. RTC_TIMER_LP_INT_ENA_REG (0x0040)



RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_ENA 写 1 使能 RTC_TIMER_OVERFLOW_INT (LP CPU) 中断。(R/W)

RTC_TIMER_MAIN_TIMER_LP_INT_ENA 写1使能 RTC_TIMER_CMP1_INT 中断。(R/W)

Register 12.17. RTC_TIMER_LP_INT_CLR_REG (0x0044)

Diagram of the RTC_TIMER_MAIN_TIMER_LP_INT_CLR register. The register is 32 bits wide. Bits 31 and 30 are labeled as RTC_TIMER_MAIN_TIMER_LP_INT_CLR and RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_CLR respectively. Bits 29 through 0 are reserved. The register is shown as a horizontal bar with a 'Reset' button on the right.

RTC_TIMER_MAIN_TIMER_OVERFLOW_LP_INT_CLR 写 1 清除 RTC_TIMER_OVERFLOW_INT (LP CPU) 中断。(WT)

RTC_TIMER_MAIN_TIMER_LP_INT_CLR 写1清除 RTC_TIMER_CMP1_INT 中断。(WT)

Register 12.18. RTC_TIMER_DATE_REG (0x03FC)

31	30	0
0	0x2311090	Reset

RTC_TIMER_DATE 版本控制寄存器。(R/W)

RTC_TIMER_CLK_EN 配置 RTC 定时器时钟门控。

0: 仅在应用程序写入寄存器时支持时钟。

1: 始终强制打开寄存器的时钟。

(R/W)

第 13 章

权限控制 (PMS)

13.1 概述

ESP32-C5 整个系统的权限控制可以分为两部分：PMP（Physical Memory Protection，物理存储器保护）和 APM（Access Permission Management，访问权限管理）。

PMP 位于 HP CPU 内部，能够管控 HP CPU 访问所有的地址空间。APM 则是一个作用于总线端口处的访问权限管理模块，可以与 PMP 一起管理 HP CPU 访问部分存储器和外设的访问权限，还可以独立管理 LP CPU 访问部分内存和外存的访问权限。此外，APM 还可以独立控制 GDMA，MEM_MONITOR 等可以发起总线读写请求的主设备访问部分存储器的访问权限。

PMP 和 APM 管控的区域分布如表 13.1-1 所示。对于主机 HP CPU，PMP 和 APM 的管控关系见图 13.1-1。

表 13.1-1. PMP 和 APM 管控的区域分布

从机 主机	ROM	HP SRAM	LP SRAM	HP CPU_PERI ¹	HP_PERI ²	LP_PERI ³	EXT MEM ⁴
HP CPU	PMP	PMP+ CPU_APM	PMP + LP_APM/ LP_APM0	PMP + HP_APM + PERI_APM	PMP + HP_APM + PERI_APM	PMP + LP_APM + PERI_APM	PMP
LP CPU	N/A	HP_APM	LP_APM/ LP_APM0	N/A	HP_APM + PERI_APM	LP_APM + PERI_APM	N/A
GDMA	N/A	HP_APM	LP_APM/ LP_APM0	N/A	N/A	N/A	HP_APM
其他主机 ⁵	N/A	HP_APM	LP_APM/ LP_APM0	N/A	N/A	N/A	N/A

¹ HP CPU 相关外设寄存器，地址范围：0x600C_0000 – 0x600C_FFFF

² 高性能系统相关外设寄存器，地址范围：0x6000_0000 – 0x600A_FFFF

³ 低功耗系统相关外设寄存器，地址范围：0x600B_0000 – 0x600B_FFFF

⁴ 外部存储器，如 flash、PSRAM。

⁵ 如 TCM_MEM_MONITOR、PSRAM_MEM_MONITOR 等其他可以发起总线读写请求的主机，可参考表 13.5-1。

⁶ CPU_APM、HP_APM、LP_APM、LP_APM0、PERI_APM 的介绍参考章节 13.5.2。

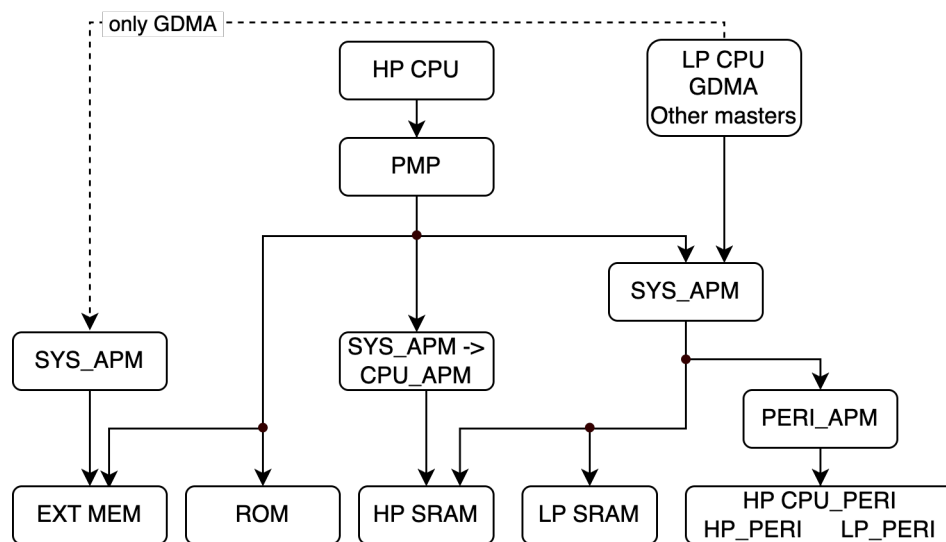


图 13.1-1. PMP 和 APM 的管控关系

如图所示，HP CPU 访问 ROM 和 EXT MEM 只受 PMP 管控，APM 无法控制。HP CPU 访问 HP SRAM 等其他数据通路，则需要先接受 PMP 的权限管控，再接受 APM 的权限管控。如果 PMP 检查不通过，则不会触发 APM 权限管控。对于外设寄存器，会经过两级 APM 权限管控，第一级根据不同地址区域划分管控范围，第二级根据不同外设划分管控范围。

PMP 相关的寄存器位于 HP CPU 内部，需要通过特殊指令进行获取或配置。关于如何配置 PMP，请参考章节 [高性能处理器 \[to be added later\]](#) > [存储器保护 \[to be added later\]](#)。

以下章节将详细介绍 APM 模块的特性、功能和配置。

13.2 APM 简介

APM 访问权限管理模块包含两部分：TEE（Trusted Execution Environment，可信执行环境）控制器和 APM 控制器。他们分别包含自己的寄存器模块：TEE 寄存器和 APM 寄存器。

- TEE 控制器负责配置 ESP32-C5 中主机（如 GDMA）访问存储器或外设寄存器的安全模式，共支持四种安全模式：TEE、REE0（Rich Execution Environment）、REE1 和 REE2。
- APM 控制器分为两类，SYS_APM 控制器和 PERI_APM 控制器。SYS_APM 控制器的寄存器在 APM 寄存器中，PERI_APM 控制器的寄存器在 TEE 寄存器中。
 - SYS_APM 控制器负责管理主机访问存储器和外设寄存器的权限。通过将预先配置好的地址范围及每个地址范围对应的读、写以及执行权限，与总线上携带的信息，如 ID 号（具体可参考表 13.5-1）、安全模式、访问地址、读写执行等信息做对比，从而判断是否允许访问。
 - PERI_APM 控制器负责管理主机访问外设寄存器的权限。通过将预先配置好的每个外设一组的读、写权限寄存器，与总线上携带的信息，如安全模式、访问地址所属外设、读写执行等信息做对比，从而判断是否允许访问。

TEE 寄存器用于配置每个主机的安全模式。APM 寄存器用于指定每一个安全模式的访问权限和访问地址范围。通过 TEE 控制器和 APM 控制器的配合，可以精确控制所有主机对存储器和外设寄存器的访问权限。

13.3 主要特性

ESP32-C5 共有两个 TEE 控制器：HP_TEE 和 LP_TEE。

- HP_TEE 具有以下特性：
 - 支持 4 种安全模式
 - 支持 32 个主机安全模式配置
- LP_TEE 具有以下特性：
 - 支持 4 种安全模式
 - 支持 1 个主机 (LP CPU) 安全模式配置

ESP32-C5 共有 4 个 SYS_APM 控制器：HP_APM_CTRL、LP_APM_CTRL、LP_APMO_CTRL 和 CPU_APM_CTRL。

- HP_APM_CTRL 控制器具有以下特性：
 - 最多支持 16 组地址范围的配置
 - 支持 HP SRAM、HP CPU 外设寄存器、HP 系统外设寄存器以及外部存储器的访问权限管理
 - 支持中断功能
 - 支持异常信息记录功能
- LP_APM_CTRL 控制器具有以下特性：
 - 最多支持 8 组地址范围的配置
 - 支持 LP SRAM 低速访问的权限管理以及 LP 外设寄存器的访问权限管理
 - 支持中断功能
 - 支持异常信息记录功能
- LP_APMO_CTRL 控制器具有以下特性：
 - 最多支持 8 组地址范围的配置
 - 支持 LP SRAM 高速访问的权限管理
 - 支持中断功能
 - 支持异常信息记录功能
- CPU_APM_CTRL 控制器具有以下特性：
 - 最多支持 8 组地址范围的配置
 - 支持 HP CPU 访问 HP SRAM 的权限管理
 - 支持中断功能
 - 支持异常信息记录功能

ESP32-C5 共有 2 个 PERI_APM 控制器：HP_PERI_APM_CTRL、LP_PERI_APM_CTRL。

- HP_PERI_APM_CTRL 和 LP_PERI_APM_CTRL 具有以下相同特性：
 - 支持对每个外设寄存器单独的权限配置

- 支持通过系统总线返回错误状态到主机的功能
- 支持异常信息记录功能

13.4 TEE 与 REE 术语

TEE 代表可信执行环境 (Trusted Execution Environment)，它是与主操作系统隔离的安全区域，可以为执行敏感操作提供安全环境。

REE 代表常规执行环境 (Rich Execution Environment)，作为一般用途的执行环境，用于运行大多数应用程序。

表 13.4-2. TEE 与 REE 之间的比较

方面	TEE	REE
安全性	安全增强	普通
访问权限	TEE 模式下的主机始终拥有地址范围内的读、写、执行权限。	由用户配置安全权限。REE0/1/2 的安全等级也由用户自定义。

13.5 功能描述

13.5.1 TEE 控制器功能描述

ESP32-C5 芯片支持四种安全模式：TEE、REE0、REE1、REE2。

当 HP CPU 作为主机访问存储器或外设寄存器时，可先配置 HP CPU 的用户模式或机器模式，然后通过配置 HP_TEE 寄存器模块配置其安全模式。关于 CPU 机器模式及用户模式的配置方法请参考 RISC-V 指令集手册 v1.10 第二卷“特权架构” (RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10)。

- 当 HP CPU 为机器模式时，其安全模式为 TEE 模式。
- 当 HP CPU 为用户模式时，其安全模式为 REE 模式，具体还需设置 TEE_MO_MODE_CTRL_REG 寄存器中的 TEE_MO_MODE 指定 REE0、REE1 或 REE2 模式：
 - 若 TEE_MO_MODE 为 0，即选择 TEE 模式，但是在 HP CPU 的用户模式下，其实际的安全模式为 REE0 模式。
 - 若 TEE_MO_MODE 为 1、2 或 3，则安全模式分别为 REE0、REE1、REE2。

对于 LP CPU 访问存储器或外设寄存器，可通过配置 LP_TEE 寄存器模块的 LP_TEE_MO_MODE 来配置其安全模式。

对于除了 LP CPU 以外的其他主机访问存储器或外设寄存器，可通过配置 HP_TEE 寄存器模块的 TEE_Mn_MODE 来配置其安全模式，其中 *n* 与表 13.5-1 中的主机 ID 号相对应。

表 13.5-1. HP 系统主机访问来源

<i>n</i> 值	来源
0	HP CPU

值	来源
1	保留
2	保留
3	保留
4	保留
5	TCM_MEM_MONITOR
6	TRACE
7	保留
8	PSRAM_MEM_MONITOR
9 ~ 15	保留
16 ~ 31	详见章节 3 通用 DMA 控制器 (GDMA) > 表 3.4-1 中 0 ~ 15 对应的外设。例如，16 对应的来源为表 3.4-1 中 0 对应的外设，17 对应的来源为表 3.4-1 中 1 对应的外设，以此类推。

可以配置 `TEE_Mn_LOCK` 和 `LP_TEE_MO_LOCK` 锁定主机安全模式的配置，只有芯片复位、系统复位及内核复位可以解除锁定并将 TEE 寄存器所有配置恢复成默认值。

13.5.2 APM 控制器功能描述

13.5.2.1 结构概览

图 13.5-1 为 APM 控制器的结构图，展示了 APM 控制器管控的总线访问路径。

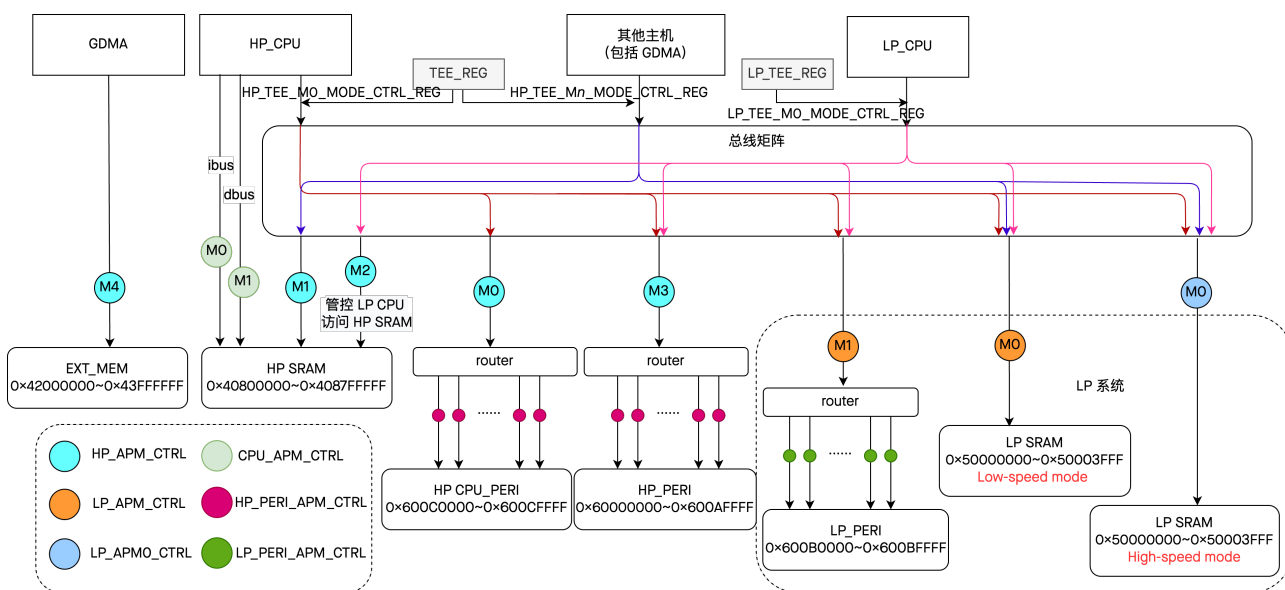


图 13.5-1. APM 控制器结构图

说明:

图中 Low-speed mode 和 High-speed mode 访问区别，请参考章节 4 系统和存储器。

SYS_APM 控制器共有 4 个：

- `HP_APM_CTRL`，由 `HP_APM_REG` 寄存器模块配置。

- LP_APMO_CTRL，由 LP_APMO_REG 寄存器模块配置。
- LP_APM_CTRL，由 LP_APM_REG 寄存器模块配置。
- CPU_APM_CTRL，由 CPU_APM_REG 寄存器模块配置。

PERI_APM 控制器共有 2 个：

- HP_PERI_APM_CTRL，由 TEE_REG 寄存器模块配置。
- LP_PERI_APM_CTRL，由 LP_TEE_REG 寄存器模块配置。

下面具体介绍各 APM 控制器的访问路径和权限检查。

- HP_APM_CTRL 管控 5 条访问路径，即图 13.5-1 中蓝色的 MO-M4，可通过配置 HP_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- LP_APMO_CTRL 管控 1 条访问路径，即图 13.5-1 中最右边的 MO，可通过配置 LP_APMO_FUNC_CTRL_REG 使能该路径的权限检查（默认使能）。
- LP_APM_CTRL 管控 2 条访问路径，即图 13.5-1 中橙色的 MO-M1，可通过配置 LP_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- CPU_APM_CTRL 管控 2 条访问路径，即图 13.5-1 中绿色的 MO-M1，可通过配置 CPU_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- HP_PERI_APM_CTRL 管控高性能系统所有外设寄存器访问路径，可通过配置 TEE_PERI_CTRL_REG 使能各路径的权限检查。
- LP_PERI_APM_CTRL 管控低功耗系统所有外设寄存器访问路径，可通过配置 LP_TEE_PERI_CTRL_REG 使能各路径的权限检查。

各 APM 控制器的详细信息可参考下表 13.5-2：

表 13.5-2. APM 控制器配置信息

APM 控制器	寄存器模块	管控的访问 路径数	访问路径权限检查使能	可配地址范 围数	地址范围使能
HP_APM_CTRL	HP_APM_REG	5	HP_APM_FUNC_CTRL_REG	16	HP_APM_REGION_FILTER_EN_REG
LP_APMO_CTRL	LP_APMO_REG	1	LP_APMO_FUNC_CTRL_REG	8	LP_APMO_REGION_FILTER_EN_REG
LP_APM_CTRL	LP_APM_REG	2	LP_APM_FUNC_CTRL_REG	8	LP_APM_REGION_FILTER_EN_REG
CPU_APM_CTRL	CPU_APM_REG	2	CPU_APM_FUNC_CTRL_REG	8	CPU_APM_REGION_FILTER_EN_REG
HP_PERI_APM_CTRL	TEE_REG	与高性能系统寄存器模块数量相同	N/A	与高性能系统寄存器模块数量相同	N/A
LP_PERI_APM_CTRL	LP_TEE_REG	与低功耗系统寄存器模块数量相同	N/A	与低功耗系统寄存器模块数量相同	N/A

13.5.2.2 SYS_APM 控制器

13.5.2.2.1 地址范围

- `HP_APM_REG` 寄存器模块可配置 16 组地址范围（即区域 n ， $n=0-15$ ）。`HP_APM_REGION $n$ _ADDR_START` 和 `HP_APM_REGION $n$ _ADDR_END` 分别配置区域 n 的起始地址和结束地址。可通过配置 `HP_APM_REGION_FILTER_EN_REG` 的 bit n 使能区域 n ，默认使能区域 0，即第 1 组的地址范围。
- `LP_APM0_REG` 寄存器模块可配置 8 组地址范围（即区域 n ， $n=0-7$ ）。`LP_APM0_REGION $n$ _ADDR_START` 和 `LP_APM0_REGION $n$ _ADDR_END` 分别配置区域 n 的起始地址和结束地址。可通过配置 `LP_APM0_REGION_FILTER_EN_REG` 的 bit n 使能区域 n ，默认使能区域 0，即第 1 组的地址范围。
- `LP_APM_REG` 寄存器模块可配置 8 组地址范围（即区域 n ， $n=0-7$ ）。`LP_APM_REGION $n$ _ADDR_START` 和 `LP_APM_REGION $n$ _ADDR_END` 分别配置区域 n 的起始地址和结束地址。可通过配置 `LP_APM_REGION_FILTER_EN_REG` 的 bit n 使能区域 n ，默认使能区域 0，即第 1 组的地址范围。
- `CPU_APM_REG` 寄存器模块可配置 8 组地址范围（即区域 n ， $n=0-7$ ）。`CPU_APM_REGION $n$ _ADDR_START` 和 `CPU_APM_REGION $n$ _ADDR_END` 分别配置区域 n 的起始地址和结束地址。可通过配置 `CPU_APM_REGION_FILTER_EN_REG` 的 bit n 使能区域 n ，默认使能区域 0，即第 1 组的地址范围。

配置地址范围时，地址的最小对齐单位是 4 字节，即地址的低两位为 0。例如，地址范围可以设置为 `0x4080000C ~ 0x40808774` 或 `0x600C0008 ~ 0x600CFF70`。

13.5.2.2.2 地址范围的访问权限

每个地址范围内，均可以为不同的安全模式单独配置访问权限（读/写/执行）：

- TEE 模式下的主机始终拥有地址范围内的读、写、执行权限。
- REE0、REE1 和 REE2 模式下的主机，根据不同的访问路径，其读写执行权限由 `HP_APM_REGION $n$ _ATTR_REG`、`LP_APM_REGION $n$ _ATTR_REG` 或 `LP_APM0_REGION $n$ _ATTR_REG` 配置。

由同一 APM 寄存器模块管理的所有访问路径共享该模块的配置，即配置的地址范围及其权限。例如上图中 `HP_APM_CTRL M0-M4` 在进行访问管控时，都依照 `HP_APM_REG` 寄存器模块中配置的 16 组地址范围及其对应的读、写、执行权限。`LP_APM_CTRL M0-M1` 在进行访问管控时，都依照 `LP_APM_REG` 寄存器模块中配置的 4 组地址范围及其对应的读、写、执行权限。

以 HP SRAM 为例。HP CPU 可通过 PMP 和 `CPU_APM_CTRL` 访问 HP SRAM，LP CPU 可通过 `HP_APM_CTRL M2` 访问 HP SRAM，其他主机都需要通过 `HP_APM_CTRL M1` 访问 HP SRAM。假设 `HP_APM_M1_FUNC_EN` 使能，REE1 安全模式的主机需要通过 `HP_APM_CTRL M1` 这条访问路径读取 HP SRAM 的数据，具体流程如下：

1. `HP_APM_CTRL M1` 会先判断请求访问的地址是否在 `HP_APM_REG` 寄存器模块中配置的 16 组地址范围内。
2. 假设该访问地址在配置的第 2 组地址范围内，则需判断该组地址范围是否使能，即 `HP_APM_REGION_FILTER_EN` 的 bit 1 是否为 1。
3. 若地址范围已使能，判断主机在 REE1 模式下对第 2 组地址范围是否具有读的权限，即 `HP_APM_REGION1_ATTR_REG` 中的 `HP_APM_REGION1_R1_R` 是否有效（即为 1），若有效，则此次读请求会被正确响应，否则读到的数据恒为 0。

当 HP 电源域（请参考章节 4 低功耗管理 *[to be added later]*）掉电重启后，LP CPU 默认没有 HP SRAM 的访问权限。可以在主机 TEE 模式下配置 LP 电源域中的 `LP_TEE_FORCE_ACC_HPMEM_EN` 寄存器，当该寄存器使能时，LP CPU 可以访问 HP SRAM，且不需要经过 SYS_APM 控制器的权限检查。

说明：

- 芯片上电时，只有 HP CPU 默认处于 TEE 模式，其他主机均处于 REE2 模式。并且默认状态下，SYS_APM 控制器会阻止所有处于 REE0、REE1 和 REE2 模式的主机的访问请求。
- 章节 13.8 寄存器列表 中列出的所有寄存器均只有处于 TEE 安全模式的主机才可以配置。
- 在 APM 寄存器配置的多个区域的地址空间重叠的情况下，例如一个区域设置为不可读，一个区域设置为可读，两个区域的地址重叠，这部分重叠的地址空间的属性是可读的。写以及执行权限适用同样的规则。
- 可以配置 `HP_APM_REGION $n$ _LOCK`、`LP_APM_REGION $n$ _LOCK` 和 `LP_APM0_REGION $n$ _LOCK` 锁定 APM 寄存器中 REGION n 的首尾地址及权限的配置，只有芯片复位、系统复位及内核复位可以解除锁定并将 APM 寄存器所有配置恢复成默认值。

13.5.2.3 PERI_APM 控制器

PERI_APM 不是以地址划分权限管控的范围，而是以每个外设寄存器为单位进行权限管控。每一个外设寄存器 *PERI* 都有以下 8 个域：

- HP/LP_TEE_WRITE_TEE_*PERI*
- HP/LP_TEE_WRITE_REE0_*PERI*
- HP/LP_TEE_WRITE_REE1_*PERI*
- HP/LP_TEE_WRITE_REE2_*PERI*
- HP/LP_TEE_READ_TEE_*PERI*
- HP/LP_TEE_READ_REE0_*PERI*
- HP/LP_TEE_READ_REE1_*PERI*
- HP/LP_TEE_READ_REE2_*PERI*

通过上述 8 个域，分别控制从机 *PERI* 在 TEE、REE0、REE1、REE2 四个安全模式下的读/写权限，为 1 表示有对应安全模式下的读/写权限。

复位后默认状态下，每个外设寄存器仅有 HP/LP_TEE_WRITE_TEE_*PERI* 和 HP/LP_TEE_READ_TEE_*PERI* 为 1，其他域均为 0，表示默认状态下仅有 TEE 安全模式下的主机可以配置外设寄存器。

本文所提及的六个寄存器模块，也是受 PERI_APM 权限管控的外设。六个寄存器模块和 PERI_APM 中对应的权限配置寄存器如下所示：

- `HP_APM_REG`，权限配置寄存器为 `TEE_HP_APM_CTRL_REG`
- `LP_APM0_REG`，权限配置寄存器为 `TEE_HP_APM_CTRL_REG`
- `LP_APM_REG`，权限配置寄存器为 `LP_TEE_LP_APM_CTRL_REG`
- `CPU_APM_REG`，权限配置寄存器为 `TEE_CPU_APM_CTRL_REG`
- `TEE_REG`，权限配置寄存器为 `TEE_TEE_CTRL_REG`
- `LP_TEE_REG`，权限配置寄存器为 `LP_TEE_LP_TEE_CTRL_REG`

这些权限配置寄存器中，仅有 HP/LP_TEE_WRTIE/READ_TEE_ *PERI* 域可配，HP/LP_TEE_WRTIE/READ_REEO/1/2_ *PERI* 域不可配置且固定为 0，表示这些寄存器不允许安全模式不是 TEE 的主机访问。

13.6 非法访问与中断

13.6.1 SYS_APM 控制器

若总线上携带的信息与配置不一致时，ESP32-C5 将视其为非法访问，并进行如下处理：

- 拒绝该访问请求，并返回默认值，具体表现为：
 - 读取和执行操作时返回 0
 - 写入操作时无效
- 触发中断

SYS_APM 控制器会自动记录非法访问的相关信息，包括主机的 ID、安全模式、访问地址、非法访问原因（地址超限/权限不足）以及各访问路径的权限判定状态。所有这些信息都可以从章节 13.8 寄存器列表 列出的相关寄存器中获取。

以 HP_APM_CTRL MO 数据访问路径为例，当出现非法访问时：

- HP_APM_MO_EXCEPTION_ID 记录访问主机的 ID。
- HP_APM_MO_EXCEPTION_MODE 记录访问安全模式。
- HP_APM_MO_EXCEPTION_ADDR 记录访问地址。
- HP_APM_MO_EXCEPTION_STATUS 记录非法访问原因。
 - 当访问的地址不在 16 组地址范围中配置的区域时，bit 1 会被置 1，表示地址超限。
 - 当访问的地址在某一组或多组地址范围内，但访问主机在这一组或多组地址范围内没有对应的读/写/执行操作权限时，bit 0 会被置 1，表示权限不足。
- HP_APM_MO_EXCEPTION_REGION 记录各地址范围的权限判定状态。该寄存器总共 16 个 bit，对应 16 组地址范围，bit 0 对应第 1 组地址范围。若访问的地址在配置使能的某一组地址范围内，但在这组地址范围内没有要求的读/写/执行操作权限时，本寄存器对应于这组地址范围的 bit 位会被置 1。

SYS_APM 控制器可产生 10 个中断信号，并将其发送给中断矩阵：

- HP_APM_MO_INTR
- HP_APM_M1_INTR
- HP_APM_M2_INTR
- HP_APM_M3_INTR
- HP_APM_M4_INTR
- LP_APM_MO_INTR
- LP_APM_M1_INTR
- LP_APM_MO_INTR
- CPU_APM_MO_INTR

- CPU_APM_M1_INTR

这 10 个中断信号与图 13.5-1 中受管控的访问路径一一对应。

每个中断都配置一个中断使能寄存器 APM_INT_EN 和一个中断清除寄存器 EXCEPTION_STATUS_CLR。当中断使能寄存器为 1 时，当一个受管控的访问路径有非法访问，则对应的中断会被产生，配置中断清除寄存器，则中断信号会被清除，直至下一次非法访问到来。

13.6.2 PERI_APM 控制器

若总线上携带的信息与配置不一致时，ESP32-C5 将视其为非法访问，并进行如下处理：

- 拒绝该访问请求，并返回默认值，具体表现为：
 - 读取和执行操作时返回 0
 - 写入操作时无效
- 通过总线返回错误信息到主机，使主机进入异常状态

可配置 HP/LP_BUS_ERR_RESP_EN 为 1 使能非法访问返回总线错误信息的功能。

PERI_APM 控制器不会记录非法访问的相关信息。

13.7 配置流程

主机访问存储器或外设寄存器路径中 APM 管控的配置流程为：

1. 配置 HP CPU 到机器模式（即 TEE 模式）。
2. 根据表 13.5-1 中的主机 ID 号，配置 TEE_Mn_MODE 寄存器，或者 LP_TEE_Mn_MODE 寄存器，配置 ESP32-C5 中主机的安全模式。
3. 配置 HP_APM_REGIONn_ADDR_START、HP_APM_REGIONn_ADDR_END 寄存器，或者 LP_APM_REGIONn_ADDR_START、LP_APM_REGIONn_ADDR_END 寄存器，或者 LP_APMO_REGIONn_ADDR_START、LP_APMO_REGIONn_ADDR_END 寄存器，或者 CPU_APM_REGIONn_ADDR_START、CPU_APM_REGIONn_ADDR_END 寄存器，配置 APM 寄存器中的各组地址范围。
4. 配置 HP_APM_REGIONn_ATTR_REG 寄存器，或者 LP_APM_REGIONn_ATTR_REG 寄存器，或者 LP_APMO_REGIONn_ATTR_REG 寄存器，或者 CPU_APM_REGIONn_ATTR_REG 寄存器，配置 APM 寄存器中的各组地址范围的访问权限。
5. 配置 HP_APM_REGION_FILTER_EN_REG 寄存器，或者 LP_APM_REGION_FILTER_EN_REG 寄存器，或者 LP_APMO_REGION_FILTER_EN_REG 寄存器，或者 CPU_APM_REGION_FILTER_EN_REG 寄存器，使能对应地址范围的权限检查。
6. 配置 HP_APM_FUNC_CTRL_REG、LP_APM_FUNC_CTRL_REG、LP_APMO_FUNC_CTRL_REG 或 CPU_APM_FUNC_CTRL_REG 寄存器使能对应路径的 APM 控制器，默认使能。
7. （仅访问外设寄存器时需要）配置 TEE_PERI_CTRL_REG 或 LP_TEE_PERI_CTRL_REG，配置 PERI 寄存器的访问权限。

以 I2S 通过 GDMA 访问 HP SRAM 为例，假设仅允许其第 4 组地址范围 0x40805000 ~ 0x4080FFFF 地址范围内读写，则配置流程如下：

1. 配置 HP CPU 为机器模式（即 TEE 模式）。
2. 根据表 13.5-1 中的主机 ID 号, 配置 `TEE_M19_MODE` 寄存器为 1, 将 I2S 通过 GDMA 访问的安全模式设置为 REEO 模式。
3. 配置 `HP_APM_REGION3_ADDR_START` 为 0x40805000, 配置 `HP_APM_REGION3_ADDR_END` 为 0x4080EFFF。
4. 配置 `HP_APM_REGION3_RO_W` 和 `HP_APM_REGION3_RO_R` 为 1, 使能对地址范围的写入、读取权限。
5. 配置 `HP_APM_REGION_FILTER_EN` 的 bit 3 为 1, 使能区域 3（即第 4 组地址范围）。
6. 配置 `HP_APM_M1_FUNC_EN` 为 1, 使能 HP_APM_CTRL M1 检查权限。

通过以上配置, I2S 可通过 GDMA 在 HP SRAM 中的 0x40805000 ~ 0x4080FFFF 地址范围内读写。

13.8 寄存器列表

13.8.1 HP_APM_REG

本小节的地址均为相对于 HP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型, 了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
<code>HP_APM_REGION_FILTER_EN_REG</code>	使能区域地址范围寄存器	0x0000	R/W
<code>HP_APM_REGION0_ADDR_START_REG</code>	配置区域地址寄存器	0x0004	R/W
<code>HP_APM_REGION0_ADDR_END_REG</code>	配置区域地址寄存器	0x0008	R/W
<code>HP_APM_REGION1_ADDR_START_REG</code>	配置区域地址寄存器	0x0010	R/W
<code>HP_APM_REGION1_ADDR_END_REG</code>	配置区域地址寄存器	0x0014	R/W
<code>HP_APM_REGION2_ADDR_START_REG</code>	配置区域地址寄存器	0x001C	R/W
<code>HP_APM_REGION2_ADDR_END_REG</code>	配置区域地址寄存器	0x0020	R/W
<code>HP_APM_REGION3_ADDR_START_REG</code>	配置区域地址寄存器	0x0028	R/W
<code>HP_APM_REGION3_ADDR_END_REG</code>	配置区域地址寄存器	0x002C	R/W
<code>HP_APM_REGION4_ADDR_START_REG</code>	配置区域地址寄存器	0x0034	R/W
<code>HP_APM_REGION4_ADDR_END_REG</code>	配置区域地址寄存器	0x0038	R/W
<code>HP_APM_REGION5_ADDR_START_REG</code>	配置区域地址寄存器	0x0040	R/W
<code>HP_APM_REGION5_ADDR_END_REG</code>	配置区域地址寄存器	0x0044	R/W
<code>HP_APM_REGION6_ADDR_START_REG</code>	配置区域地址寄存器	0x004C	R/W
<code>HP_APM_REGION6_ADDR_END_REG</code>	配置区域地址寄存器	0x0050	R/W
<code>HP_APM_REGION7_ADDR_START_REG</code>	配置区域地址寄存器	0x0058	R/W
<code>HP_APM_REGION7_ADDR_END_REG</code>	配置区域地址寄存器	0x005C	R/W
<code>HP_APM_REGION8_ADDR_START_REG</code>	配置区域地址寄存器	0x0064	R/W
<code>HP_APM_REGION8_ADDR_END_REG</code>	配置区域地址寄存器	0x0068	R/W
<code>HP_APM_REGION9_ADDR_START_REG</code>	配置区域地址寄存器	0x0070	R/W
<code>HP_APM_REGION9_ADDR_END_REG</code>	配置区域地址寄存器	0x0074	R/W
<code>HP_APM_REGION10_ADDR_START_REG</code>	配置区域地址寄存器	0x007C	R/W

名称	描述	地址	访问
HP_APM_REGION10_ADDR_END_REG	配置区域地址寄存器	0x0080	R/W
HP_APM_REGION11_ADDR_START_REG	配置区域地址寄存器	0x0088	R/W
HP_APM_REGION11_ADDR_END_REG	配置区域地址寄存器	0x008C	R/W
HP_APM_REGION12_ADDR_START_REG	配置区域地址寄存器	0x0094	R/W
HP_APM_REGION12_ADDR_END_REG	配置区域地址寄存器	0x0098	R/W
HP_APM_REGION13_ADDR_START_REG	配置区域地址寄存器	0x00A0	R/W
HP_APM_REGION13_ADDR_END_REG	配置区域地址寄存器	0x00A4	R/W
HP_APM_REGION14_ADDR_START_REG	配置区域地址寄存器	0x00AC	R/W
HP_APM_REGION14_ADDR_END_REG	配置区域地址寄存器	0x00B0	R/W
HP_APM_REGION15_ADDR_START_REG	配置区域地址寄存器	0x00B8	R/W
HP_APM_REGION15_ADDR_END_REG	配置区域地址寄存器	0x00BC	R/W
HP_APM_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
HP_APM_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
HP_APM_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
HP_APM_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W
HP_APM_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
HP_APM_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
HP_APM_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
HP_APM_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
HP_APM_REGION8_ATTR_REG	区域访问权限配置寄存器	0x006C	R/W
HP_APM_REGION9_ATTR_REG	区域访问权限配置寄存器	0x0078	R/W
HP_APM_REGION10_ATTR_REG	区域访问权限配置寄存器	0x0084	R/W
HP_APM_REGION11_ATTR_REG	区域访问权限配置寄存器	0x0090	R/W
HP_APM_REGION12_ATTR_REG	区域访问权限配置寄存器	0x009C	R/W
HP_APM_REGION13_ATTR_REG	区域访问权限配置寄存器	0x00A8	R/W
HP_APM_REGION14_ATTR_REG	区域访问权限配置寄存器	0x00B4	R/W
HP_APM_REGION15_ATTR_REG	区域访问权限配置寄存器	0x00C0	R/W
HP_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
HP_APM_M0_STATUS_REG	HP_APM_CTRL M0 状态寄存器	0x00C8	RO
HP_APM_M0_STATUS_CLR_REG	HP_APM_CTRL M0 状态清除寄存器	0x00CC	WT
HP_APM_M0_EXCEPTION_INFO0_REG	HP_APM_CTRL M0 异常信息寄存器	0x00D0	RO
HP_APM_M0_EXCEPTION_INFO1_REG	HP_APM_CTRL M0 异常信息寄存器	0x00D4	RO
HP_APM_M1_STATUS_REG	HP_APM_CTRL M1 状态寄存器	0x00D8	RO
HP_APM_M1_STATUS_CLR_REG	HP_APM_CTRL M1 状态清除寄存器	0x00DC	WT
HP_APM_M1_EXCEPTION_INFO0_REG	HP_APM_CTRL M1 异常信息寄存器	0x00E0	RO
HP_APM_M1_EXCEPTION_INFO1_REG	HP_APM_CTRL M1 异常信息寄存器	0x00E4	RO
HP_APM_M2_STATUS_REG	HP_APM_CTRL M2 状态寄存器	0x00E8	RO
HP_APM_M2_STATUS_CLR_REG	HP_APM_CTRL M2 状态清除寄存器	0x00EC	WT
HP_APM_M2_EXCEPTION_INFO0_REG	HP_APM_CTRL M2 异常信息寄存器	0x00F0	RO
HP_APM_M2_EXCEPTION_INFO1_REG	HP_APM_CTRL M2 异常信息寄存器	0x00F4	RO
HP_APM_M3_STATUS_REG	HP_APM_CTRL M3 状态寄存器	0x00F8	RO
HP_APM_M3_STATUS_CLR_REG	HP_APM_CTRL M3 状态清除寄存器	0x00FC	WT

名称	描述	地址	访问
HP_APM_M3_EXCEPTION_INFO0_REG	HP_APM_CTRL M3 异常信息寄存器	0x0100	RO
HP_APM_M3_EXCEPTION_INFO1_REG	HP_APM_CTRL M3 异常信息寄存器	0x0104	RO
HP_APM_M4_STATUS_REG	HP_APM_CTRL M4 状态寄存器	0x0108	RO
HP_APM_M4_STATUS_CLR_REG	HP_APM_CTRL M4 状态清除寄存器	0x010C	WT
HP_APM_M4_EXCEPTION_INFO0_REG	HP_APM_CTRL M4 异常信息寄存器	0x0110	RO
HP_APM_M4_EXCEPTION_INFO1_REG	HP_APM_CTRL M4 异常信息寄存器	0x0114	RO
中断寄存器			
HP_APM_INT_EN_REG	HP_APM_CTRL M0/1/2/3/4 中断使能寄存器	0x0118	R/W
时钟门控寄存器			
HP_APM_CLOCK_GATE_REG	时钟门控寄存器	0x07F8	R/W
版本控制寄存器			
HP_APM_DATE_REG	版本控制寄存器	0x07FC	R/W

13.8.2 LP_APM_REG

本小节的地址均为相对于 LP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
LP_APM_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	R/W
LP_APM_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	R/W
LP_APM_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	R/W
LP_APM_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	R/W
LP_APM_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	R/W
LP_APM_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	R/W
LP_APM_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	R/W
LP_APM_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	R/W
LP_APM_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	R/W
LP_APM_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	R/W
LP_APM_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	R/W
LP_APM_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	R/W
LP_APM_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	R/W
LP_APM_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	R/W
LP_APM_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	R/W
LP_APM_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	R/W
LP_APM_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
LP_APM_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
LP_APM_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
LP_APM_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W

名称	描述	地址	访问
LP_APM_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
LP_APM_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
LP_APM_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
LP_APM_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
LP_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
LP_APM_M0_STATUS_REG	LP_APM_CTRL M0 状态寄存器	0x00C8	RO
LP_APM_M0_STATUS_CLR_REG	LP_APM_CTRL M0 状态清除寄存器	0x00CC	WT
LP_APM_M0_EXCEPTION_INFO0_REG	LP_APM_CTRL M0 异常信息寄存器	0x00D0	RO
LP_APM_M0_EXCEPTION_INFO1_REG	LP_APM_CTRL M0 异常信息寄存器	0x00D4	RO
LP_APM_M1_STATUS_REG	LP_APM_CTRL M1 状态寄存器	0x00D8	RO
LP_APM_M1_STATUS_CLR_REG	LP_APM_CTRL M1 状态清除寄存器	0x00DC	WT
LP_APM_M1_EXCEPTION_INFO0_REG	LP_APM_CTRL M1 异常信息寄存器	0x00E0	RO
LP_APM_M1_EXCEPTION_INFO1_REG	LP_APM_CTRL M1 异常信息寄存器	0x00E4	RO
中断寄存器			
LP_APM_INT_EN_REG	LP_APM_CTRL M0/1 中断使能寄存器	0x00E8	R/W
时钟门控寄存器			
LP_APM_CLOCK_GATE_REG	时钟门控寄存器	0x00EC	R/W
版本控制寄存器			
LP_APM_DATE_REG	版本控制寄存器	0x00FC	R/W

13.8.3 LP_APMO_REG

本小节的地址均为相对于 LP_APMO 的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_APMO_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
LP_APMO_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	R/W
LP_APMO_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	R/W
LP_APMO_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	R/W
LP_APMO_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	R/W
LP_APMO_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	R/W
LP_APMO_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	R/W
LP_APMO_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	R/W
LP_APMO_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	R/W
LP_APMO_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	R/W
LP_APMO_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	R/W
LP_APMO_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	R/W
LP_APMO_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	R/W
LP_APMO_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	R/W

名称	描述	地址	访问
LP_APMO_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	R/W
LP_APMO_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	R/W
LP_APMO_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	R/W
LP_APMO_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
LP_APMO_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
LP_APMO_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
LP_APMO_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W
LP_APMO_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
LP_APMO_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
LP_APMO_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
LP_APMO_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
LP_APMO_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
LP_APMO_MO_STATUS_REG	LP_APMO_CTRL MO 状态寄存器	0x00C8	RO
LP_APMO_MO_STATUS_CLR_REG	LP_APMO_CTRL MO 状态清除寄存器	0x00CC	WT
LP_APMO_MO_EXCEPTION_INFO0_REG	LP_APMO_CTRL MO 异常信息寄存器	0x00D0	RO
LP_APMO_MO_EXCEPTION_INFO1_REG	LP_APMO_CTRL MO 异常信息寄存器	0x00D4	RO
中断寄存器			
LP_APMO_INT_EN_REG	LP_APMO_CTRL 中断使能寄存器	0x00D8	R/W
时钟门控寄存器			
LP_APMO_CLOCK_GATE_REG	时钟门控寄存器	0x00DC	R/W
版本控制寄存器			
LP_APMO_DATE_REG	版本控制寄存器	0x07FC	R/W

13.8.4 CPU_APM_REG

本小节的地址均为相对于 CPU_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
CPU_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
配置区域地址寄存器			
CPU_APM_REGION0_ADDR_START_REG	配置区域地址寄存器	0x0004	varies
CPU_APM_REGION0_ADDR_END_REG	配置区域地址寄存器	0x0008	varies
CPU_APM_REGION1_ADDR_START_REG	配置区域地址寄存器	0x0010	varies
CPU_APM_REGION1_ADDR_END_REG	配置区域地址寄存器	0x0014	varies
CPU_APM_REGION2_ADDR_START_REG	配置区域地址寄存器	0x001C	varies
CPU_APM_REGION2_ADDR_END_REG	配置区域地址寄存器	0x0020	varies
CPU_APM_REGION3_ADDR_START_REG	配置区域地址寄存器	0x0028	varies
CPU_APM_REGION3_ADDR_END_REG	配置区域地址寄存器	0x002C	varies
CPU_APM_REGION4_ADDR_START_REG	配置区域地址寄存器	0x0034	varies

名称	描述	地址	访问
CPU_APM_REGION4_ADDR_END_REG	配置区域地址寄存器	0x0038	varies
CPU_APM_REGION5_ADDR_START_REG	配置区域地址寄存器	0x0040	varies
CPU_APM_REGION5_ADDR_END_REG	配置区域地址寄存器	0x0044	varies
CPU_APM_REGION6_ADDR_START_REG	配置区域地址寄存器	0x004C	varies
CPU_APM_REGION6_ADDR_END_REG	配置区域地址寄存器	0x0050	varies
CPU_APM_REGION7_ADDR_START_REG	配置区域地址寄存器	0x0058	varies
CPU_APM_REGION7_ADDR_END_REG	配置区域地址寄存器	0x005C	varies
CPU_APM_REGION0_ATTR_REG	区域访问权限配置寄存器	0x000C	R/W
CPU_APM_REGION1_ATTR_REG	区域访问权限配置寄存器	0x0018	R/W
CPU_APM_REGION2_ATTR_REG	区域访问权限配置寄存器	0x0024	R/W
CPU_APM_REGION3_ATTR_REG	区域访问权限配置寄存器	0x0030	R/W
CPU_APM_REGION4_ATTR_REG	区域访问权限配置寄存器	0x003C	R/W
CPU_APM_REGION5_ATTR_REG	区域访问权限配置寄存器	0x0048	R/W
CPU_APM_REGION6_ATTR_REG	区域访问权限配置寄存器	0x0054	R/W
CPU_APM_REGION7_ATTR_REG	区域访问权限配置寄存器	0x0060	R/W
CPU_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
CPU_APM_M0_STATUS_REG	CPU_APM_CTRL M0 状态寄存器	0x00C8	RO
CPU_APM_M0_STATUS_CLR_REG	CPU_APM_CTRL M0 状态清除寄存器	0x00CC	WT
CPU_APM_M0_EXCEPTION_INFO0_REG	CPU_APM_CTRL M0 异常信息寄存器	0x00D0	RO
CPU_APM_M0_EXCEPTION_INFO1_REG	CPU_APM_CTRL M0 异常信息寄存器	0x00D4	RO
CPU_APM_M1_STATUS_REG	CPU_APM_CTRL M1 状态寄存器	0x00D8	RO
CPU_APM_M1_STATUS_CLR_REG	CPU_APM_CTRL M1 状态清除寄存器	0x00DC	WT
CPU_APM_M1_EXCEPTION_INFO0_REG	CPU_APM_CTRL M1 异常信息寄存器	0x00E0	RO
CPU_APM_M1_EXCEPTION_INFO1_REG	CPU_APM_CTRL M1 异常信息寄存器	0x00E4	RO
中断寄存器			
CPU_APM_INT_EN_REG	CPU_APM_CTRL M0/1 中断使能寄存器	0x0118	R/W
时钟门控寄存器			
CPU_APM_CLOCK_GATE_REG	时钟门控寄存器	0x07F8	R/W
版本控制寄存器			
CPU_APM_DATE_REG	版本控制寄存器	0x07FC	R/W

13.8.5 TEE_REG

本小节的地址均为相对于 TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	权限
配置寄存器			
TEE_M0_MODE_CTRL_REG	安全模式配置寄存器	0x0000	R/W
TEE_M1_MODE_CTRL_REG	安全模式配置寄存器	0x0004	R/W

名称	描述	地址	权限
TEE_M2_MODE_CTRL_REG	安全模式配置寄存器	0x0008	R/W
TEE_M3_MODE_CTRL_REG	安全模式配置寄存器	0x000C	R/W
TEE_M4_MODE_CTRL_REG	安全模式配置寄存器	0x0010	R/W
TEE_M5_MODE_CTRL_REG	安全模式配置寄存器	0x0014	R/W
TEE_M6_MODE_CTRL_REG	安全模式配置寄存器	0x0018	R/W
TEE_M7_MODE_CTRL_REG	安全模式配置寄存器	0x001C	R/W
TEE_M8_MODE_CTRL_REG	安全模式配置寄存器	0x0020	R/W
TEE_M9_MODE_CTRL_REG	安全模式配置寄存器	0x0024	R/W
TEE_M10_MODE_CTRL_REG	安全模式配置寄存器	0x0028	R/W
TEE_M11_MODE_CTRL_REG	安全模式配置寄存器	0x002C	R/W
TEE_M12_MODE_CTRL_REG	安全模式配置寄存器	0x0030	R/W
TEE_M13_MODE_CTRL_REG	安全模式配置寄存器	0x0034	R/W
TEE_M14_MODE_CTRL_REG	安全模式配置寄存器	0x0038	R/W
TEE_M15_MODE_CTRL_REG	安全模式配置寄存器	0x003C	R/W
TEE_M16_MODE_CTRL_REG	安全模式配置寄存器	0x0040	R/W
TEE_M17_MODE_CTRL_REG	安全模式配置寄存器	0x0044	R/W
TEE_M18_MODE_CTRL_REG	安全模式配置寄存器	0x0048	R/W
TEE_M19_MODE_CTRL_REG	安全模式配置寄存器	0x004C	R/W
TEE_M20_MODE_CTRL_REG	安全模式配置寄存器	0x0050	R/W
TEE_M21_MODE_CTRL_REG	安全模式配置寄存器	0x0054	R/W
TEE_M22_MODE_CTRL_REG	安全模式配置寄存器	0x0058	R/W
TEE_M23_MODE_CTRL_REG	安全模式配置寄存器	0x005C	R/W
TEE_M24_MODE_CTRL_REG	安全模式配置寄存器	0x0060	R/W
TEE_M25_MODE_CTRL_REG	安全模式配置寄存器	0x0064	R/W
TEE_M26_MODE_CTRL_REG	安全模式配置寄存器	0x0068	R/W
TEE_M27_MODE_CTRL_REG	安全模式配置寄存器	0x006C	R/W
TEE_M28_MODE_CTRL_REG	安全模式配置寄存器	0x0070	R/W
TEE_M29_MODE_CTRL_REG	安全模式配置寄存器	0x0074	R/W
TEE_M30_MODE_CTRL_REG	安全模式配置寄存器	0x0078	R/W
TEE_M31_MODE_CTRL_REG	安全模式配置寄存器	0x007C	R/W
外设读/写控制寄存器			
TEE_UART0_CTRL_REG	UART0 读/写控制寄存器	0x0088	R/W
TEE_UART1_CTRL_REG	UART1 读/写控制寄存器	0x008C	R/W
TEE_UHCIO_CTRL_REG	UHCI 读/写控制寄存器	0x0090	R/W
TEE_I2C_EXT0_CTRL_REG	I2C 读/写控制寄存器	0x0094	R/W
TEE_I2S_CTRL_REG	I2S 读/写控制寄存器	0x009C	R/W
TEE_PARL_IO_CTRL_REG	PARL_IO 读/写控制寄存器	0x00A0	R/W
TEE_PWM_CTRL_REG	MCPWM 读/写控制寄存器	0x00A4	R/W
TEE_LEDC_CTRL_REG	LEDC 读/写控制寄存器	0x00AC	R/W
TEE_TWAIO_CTRL_REG	TWAIO 读/写控制寄存器	0x00B0	R/W
TEE_USB_SERIAL_JTAG_CTRL_REG	USB_SERIAL_JTAG 读/写控制寄存器	0x00B4	R/W
TEE_RMT_CTRL_REG	RMT 读/写控制寄存器	0x00B8	R/W

名称	描述	地址	权限
TEE_GDMA_CTRL_REG	GDMA 读/写控制寄存器	0x00BC	R/W
TEE_ETM_CTRL_REG	SOC_ETM 读/写控制寄存器	0x00C4	R/W
TEE_INTMTX_CTRL_REG	INTMTX 读/写控制寄存器	0x00C8	R/W
TEE_APB_ADC_CTRL_REG	SAR ADC 读/写控制寄存器	0x00D0	R/W
TEE_TIMERGROUPO_CTRL_REG	TIMGO 读/写控制寄存器	0x00D4	R/W
TEE_TIMERGROUPI_CTRL_REG	TIMG1 读/写控制寄存器	0x00D8	R/W
TEE_SYSTIMER_CTRL_REG	SYSTIMER 读/写控制寄存器	0x00DC	R/W
TEE_PCNT_CTRL_REG	PCNT 读/写控制寄存器	0x00F4	R/W
TEE_IOMUX_CTRL_REG	IO MUX 读/写控制寄存器	0x00F8	R/W
TEE_PSRAM_MEM_MONITOR_CTRL_REG	PSRAM_MEM_MONITOR 读/写控制寄存器	0x00FC	R/W
TEE_MEM_ACS_MONITOR_CTRL_REG	TCM_MEM_MONITOR 读/写控制寄存器	0x0100	R/W
TEE_HP_SYSTEM_REG_CTRL_REG	HP_SYSREG 读/写控制寄存器	0x0104	R/W
TEE_PCR_REG_CTRL_REG	PCR 读/写控制寄存器	0x0108	R/W
TEE_MSPI_CTRL_REG	SPI01 读/写控制寄存器	0x010C	R/W
TEE_HP_APM_CTRL_REG	HP_APM 和 LP_APMO 读/写控制寄存器	0x0110	varies
TEE_CPU_APM_CTRL_REG	CPU_APM_REG 读/写控制寄存器	0x0114	varies
TEE_TEE_CTRL_REG	TEE 读/写控制寄存器	0x0118	varies
TEE_CRYPT_CTRL_REG	CRYPT 读/写控制寄存器, 包含 AES 到 ECDSA 地址区间段内的所有安全外设	0x011C	R/W
TEE_TRACE_CTRL_REG	TRACE 读/写控制寄存器	0x0120	R/W
TEE_CPU_BUS_MONITOR_CTRL_REG	BUS_MONITOR 读/写控制寄存器	0x0128	R/W
TEE_INTPRI_REG_CTRL_REG	INTPRI_REG 读/写控制寄存器	0x012C	R/W
TEE_TWA11_CTRL_REG	TWA11 读/写控制寄存器	0x0138	R/W
TEE_SPI2_CTRL_REG	SPI2 读/写控制寄存器	0x013C	R/W
TEE_BS_CTRL_REG	BITSCRAMBLER 读/写控制寄存器	0x0140	R/W
TEE_BUS_ERR_CONF_REG	错误信息返回配置寄存器	0x0FF0	R/W
时钟门控寄存器			
TEE_CLOCK_GATE_REG	时钟门控寄存器	0x0FF8	R/W
版本控制寄存器			
TEE_DATE_REG	版本控制寄存器	0x0FFC	R/W

13.8.6 LP_TEE_REG

本小节的地址均为相对于 LP_TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 4 [系统和存储器](#) 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	权限
配置寄存器			
LP_TEE_MO_MODE_CTRL_REG	安全模式配置寄存器	0x0000	R/W
外设读/写控制寄存器			
LP_TEE_EFUSE_CTRL_REG	eFuse 读/写控制寄存器	0x0004	R/W
LP_TEE_PMU_CTRL_REG	PMU 读/写控制寄存器	0x0008	R/W
LP_TEE_CLKRST_CTRL_REG	LP_CLKRST 读/写控制寄存器	0x000C	R/W
LP_TEE_LP_AON_CTRL_CTRL_REG	LP_AON 读/写控制寄存器	0x0010	R/W
LP_TEE_LP_TIMER_CTRL_REG	LP_TIMER 读/写控制寄存器	0x0014	R/W
LP_TEE_LP_WDT_CTRL_REG	LP_WDT 读/写控制寄存器	0x0018	R/W
LP_TEE_LP_PERI_CTRL_REG	LPPERI 读/写控制寄存器	0x001C	R/W
LP_TEE_LP_ANA_PERI_CTRL_REG	LP_ANA_PERI 读/写控制寄存器	0x0020	R/W
LP_TEE_LP_IO_CTRL_REG	LP_GPIO 和 LP_IO_MUX 读/写控制寄存器	0x002C	R/W
LP_TEE_LP_TEE_CTRL_REG	LP_TEE 读/写控制寄存器	0x0034	varies
LP_TEE_UART_CTRL_REG	LP_UART 读/写控制寄存器	0x0038	R/W
LP_TEE_I2C_EXT_CTRL_REG	LP_I2C 读/写控制寄存器	0x0040	R/W
LP_TEE_I2C_ANA_MST_CTRL_REG	I2C_ANA_MST 读/写控制寄存器	0x0044	R/W
LP_TEE_LP_APM_CTRL_REG	LP_APM 读/写控制寄存器	0x004C	varies
LP_TEE_FORCE_ACC_HP_REG	LP CPU 强制访问 HP SRAM 配置寄存器	0x0090	R/W
LP_TEE_BUS_ERR_CONF_REG	错误信息返回配置寄存器	0x00F0	R/W
时钟门控寄存器			
LP_TEE_CLOCK_GATE_REG	时钟门控寄存器	0x00F8	R/W
版本控制寄存器			
LP_TEE_DATE_REG	版本控制寄存器	0x00FC	R/W

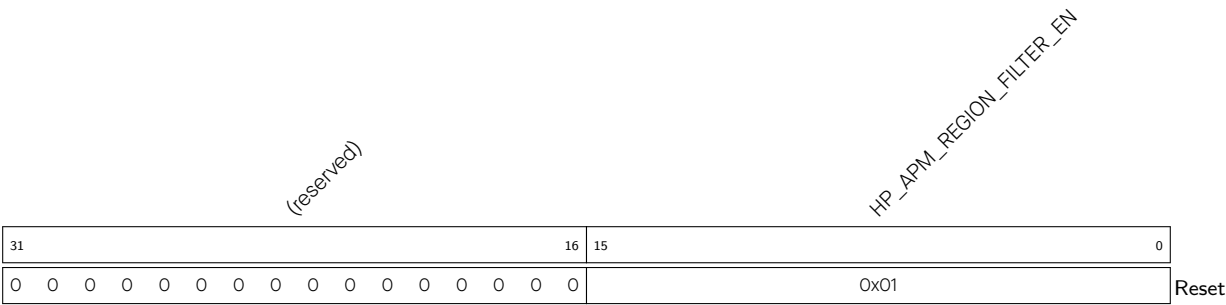
13.9 寄存器

13.9.1 HP_APM_REG

本小节的地址均为相对于 HP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.1. HP_APM_REGION_FILTER_EN_REG (0x0000)



HP_APM_REGION_FILTER_EN 配置 bit *n* (0-15) 使能区域 *n* (0-15) 的地址范围。

- 0: 禁用
- 1: 使能

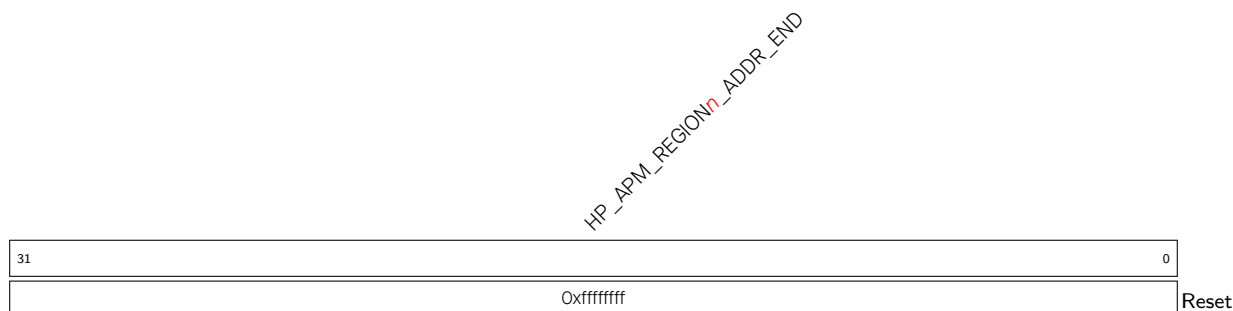
(R/W)

Register 13.2. HP_APM_REGION*n*_ADDR_START_REG (*n*: 0-15) (0x0004+0xC**n*)



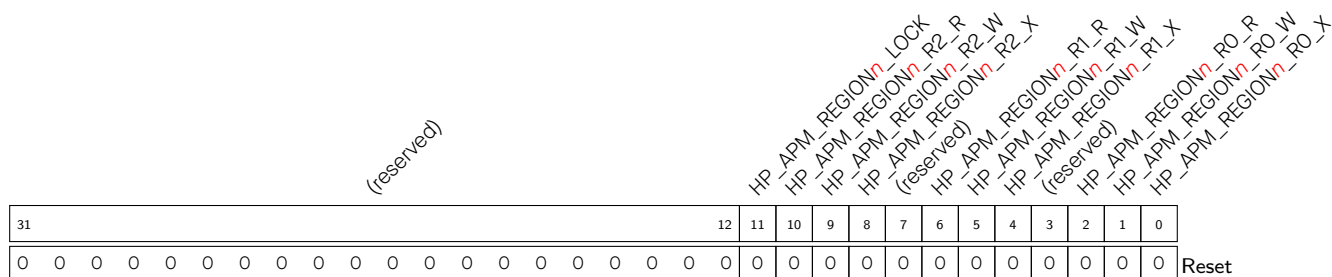
HP_APM_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

Register 13.3. HP_APM_REGION n _ADDR_END_REG (n : 0-15) (0x0008+0xC* n)



HP_APM_REGION n _ADDR_END 配置区域 n 的结束地址。(R/W)

Register 13.4. HP_APM_REGION n _ATTR_REG (n : 0-15) (0x000C+0xC* n)



HP_APM_REGION n _RO_X 配置 REEO 模式下对区域 n 的执行权限。(R/W)

HP APM REGION *n* RO W 配置 REEO 模式下对区域 *n* 的写入权限。(R/W)

HP_APM_REGION n _RO R 配置 REEO 模式下对区域 n 的读取权限。(R/W)

HP APM REGION *n* R1 X 配置 REE1 模式下对区域 *n* 的执行权限。(R/W)

HP_APM_REGION n _R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGION n _R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)

HP APM REGION n R2 X 配置 REE2 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGION n _R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGION n R2 R 配置 REE2 模式下对区域 n 的读取权限。(R/W)

HP_APM_REGION n _LOCK 锁定区域 n 的首尾地址及权限的配置
(HP_APM_REGION n _ADDR_START_REG、HP_APM_REGION n _ADDR_END_REG 和
HP_APM_REGION n _ATTR_REG)。

0: 不锁定

1: 锁定

(R/W)

Register 13.5. HP_APM_FUNC_CTRL_REG (0x00C4)

(reserved)																												HP_APM_M4_FUNC_EN HP_APM_M3_FUNC_EN HP_APM_M2_FUNC_EN HP_APM_M1_FUNC_EN HP_APM_M0_FUNC_EN						
31																												5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	Reset	

HP_APM_MO_FUNC_EN 配置使能 HP_APM_CTRL MO 权限检查。(R/W)

HP_APM_M1_FUNC_EN 配置使能 HP_APM_CTRL M1 权限检查。(R/W)

HP_APM_M2_FUNC_EN 配置使能 HP_APM_CTRL M2 权限检查。(R/W)

HP_APM_M3_FUNC_EN 配置使能 HP_APM_CTRL M3 权限检查。(R/W)

HP_APM_M4_FUNC_EN 配置使能 HP_APM_CTRL M4 权限检查。(R/W)

Register 13.6. HP_APM_MO_STATUS_REG (0x00C8)

Diagram of the HP_APM_MO_EXCEPTION_STATUS register. The register is 32 bits wide. Bits 31 down to 2 are reserved. Bit 1 is the APM bit, and bit 0 is the MO bit. The register is shown with a 'Reset' value of 0 for both bits.

31																														2	1	0	
																															0	0	Reset

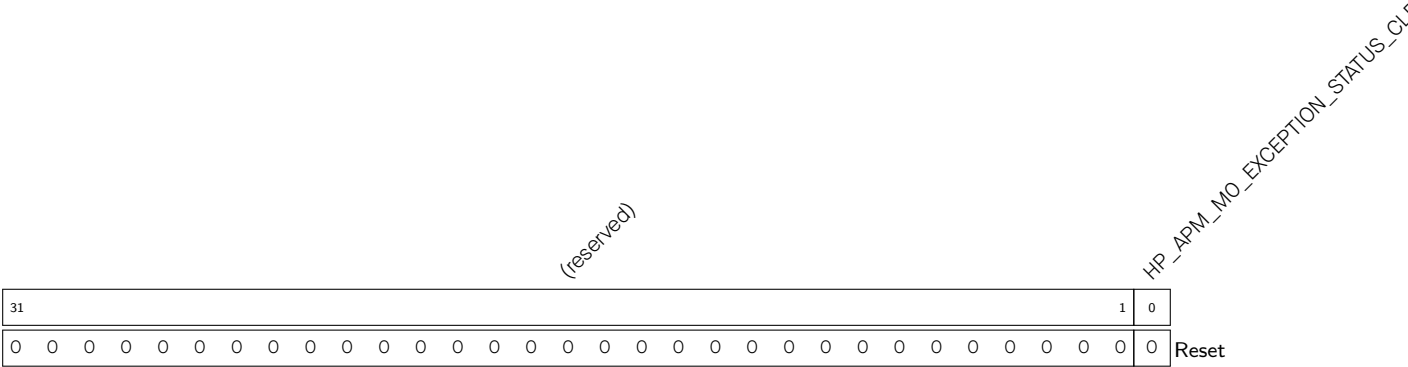
HP_APM_MO_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

bit1: 1 代表地址越限

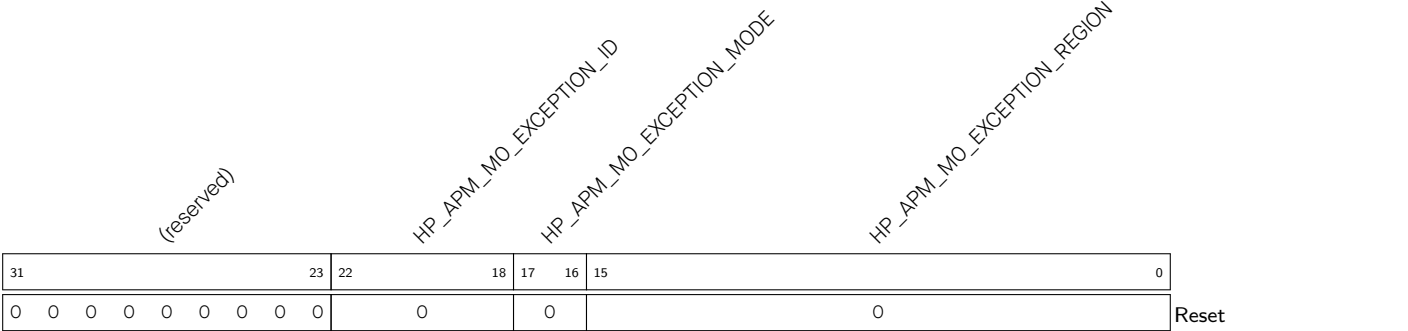
(RO)

Register 13.7. HP_APM_MO_STATUS_CLR_REG (0x00CC)



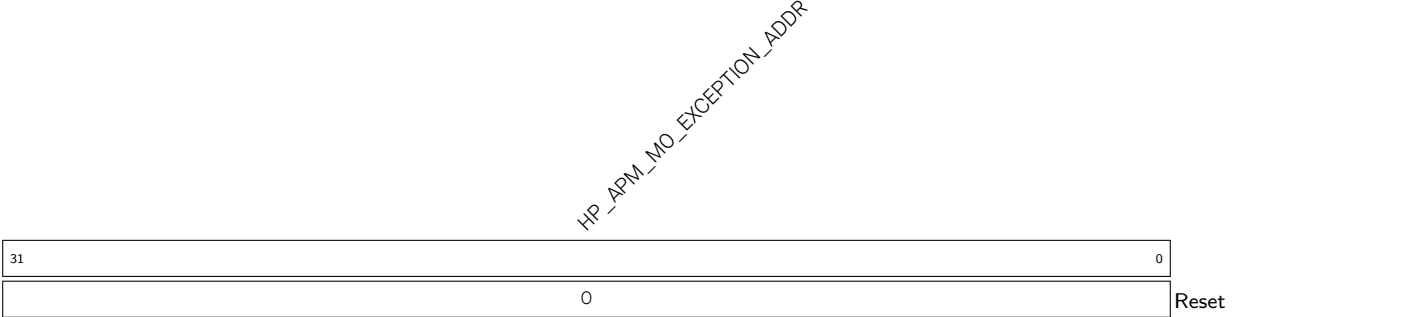
HP_APM_MO_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.8. HP_APM_MO_EXCEPTION_INFO0_REG (0x00D0)



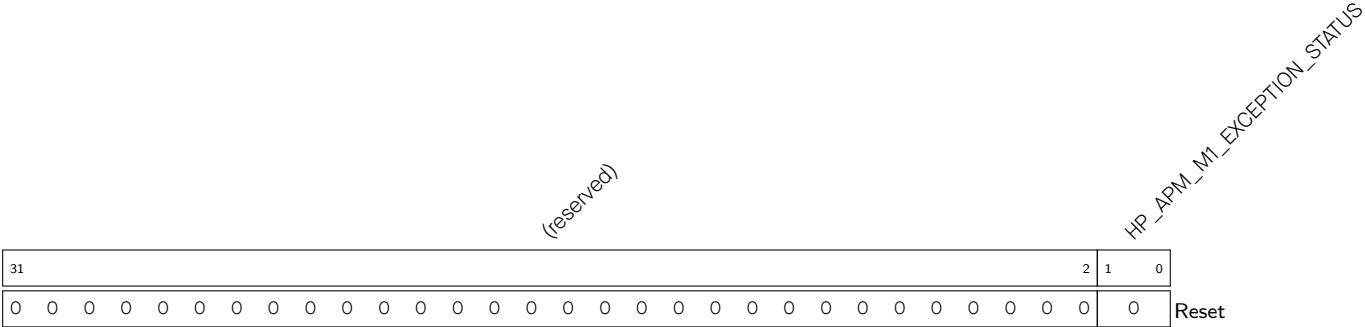
- HP_APM_MO_EXCEPTION_REGION 表示异常区域。(RO)
- HP_APM_MO_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- HP_APM_MO_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.9. HP_APM_MO_EXCEPTION_INFO1_REG (0x00D4)



HP_APM_MO_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.10. HP_APM_M1_STATUS_REG (0x00D8)

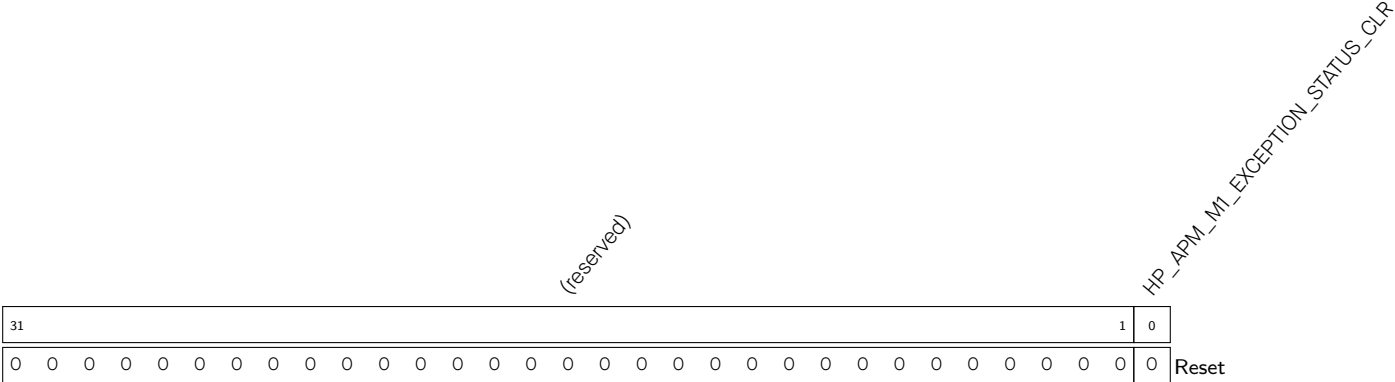


HP_APM_M1_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

bit1: 1 代表地址越限 (RO)

Register 13.11. HP_APM_M1_STATUS_CLR_REG (0x00DC)



HP_APM_M1_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.12. HP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

[illegible]

HP_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

HP_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.13. HP_APM_M1_EXCEPTION_INFO01_REG (0x00E4)

Diagram of the HP_APM_M1_EXCEPTION_ADDR register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. The label "HP_APM_M1_EXCEPTION_ADDR" is written diagonally above the register box. The word "Reset" is written below the register box.

HP_APM_M1_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.14. HP_APM_M2_STATUS_REG (0x00E8)

Diagram of the HP_APM_M2_EXCEPTION_STATUS register structure:

- Bit 31: (reserved)
- Bits 31-2: (reserved)
- Bit 1: HP_APM_M2_EXCEPTION_STATUS
- Bit 0: Reset

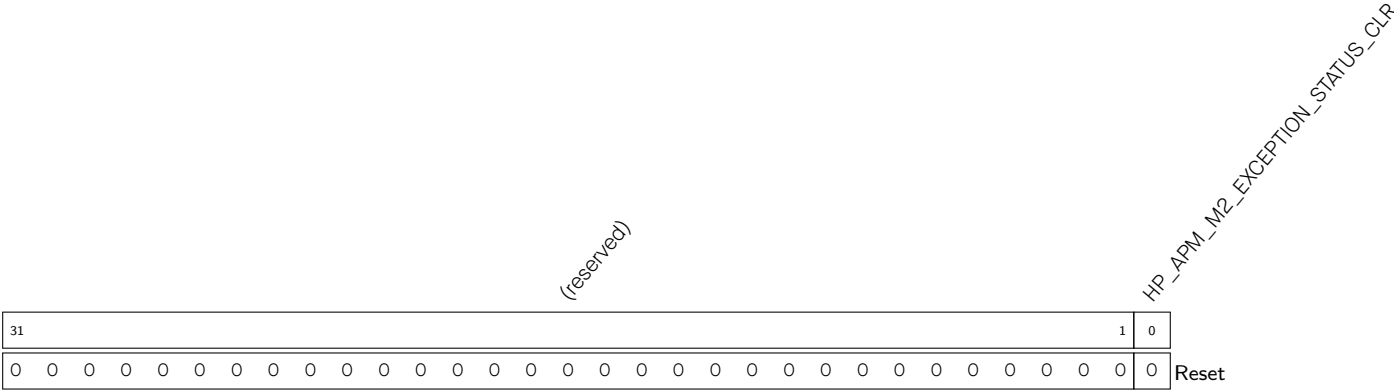
HP_APM_M2_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

bit1: 1 代表地址越限

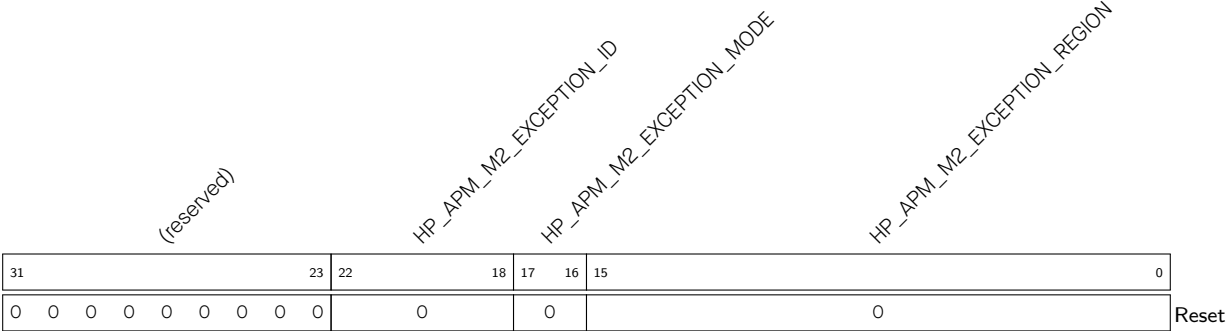
(RO)

Register 13.15. HP_APM_M2_STATUS_CLR_REG (0x00EC)



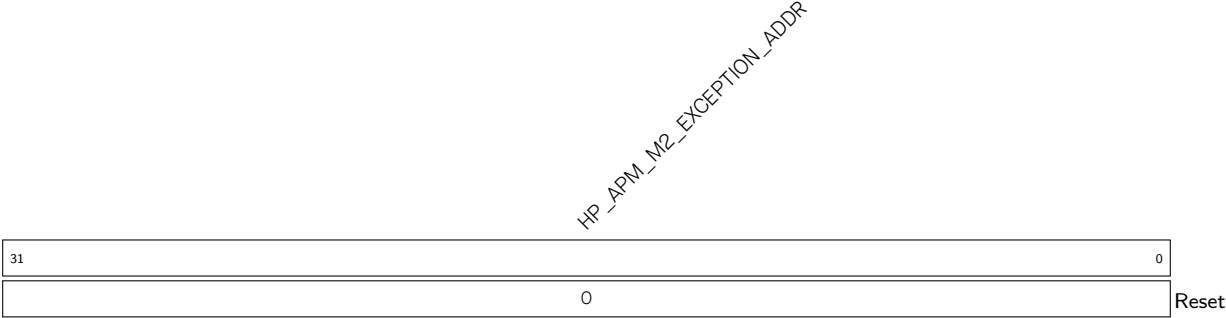
HP_APM_M2_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.16. HP_APM_M2_EXCEPTION_INFO0_REG (0x00F0)



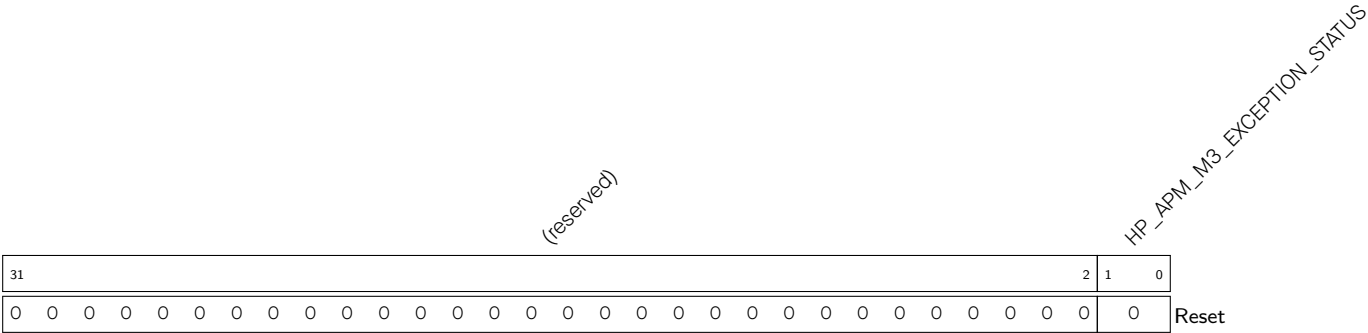
- HP_APM_M2_EXCEPTION_REGION 表示异常区域。(RO)
- HP_APM_M2_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- HP_APM_M2_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.17. HP_APM_M2_EXCEPTION_INFO1_REG (0x00F4)



HP_APM_M2_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.18. HP_APM_M3_STATUS_REG (0x00F8)

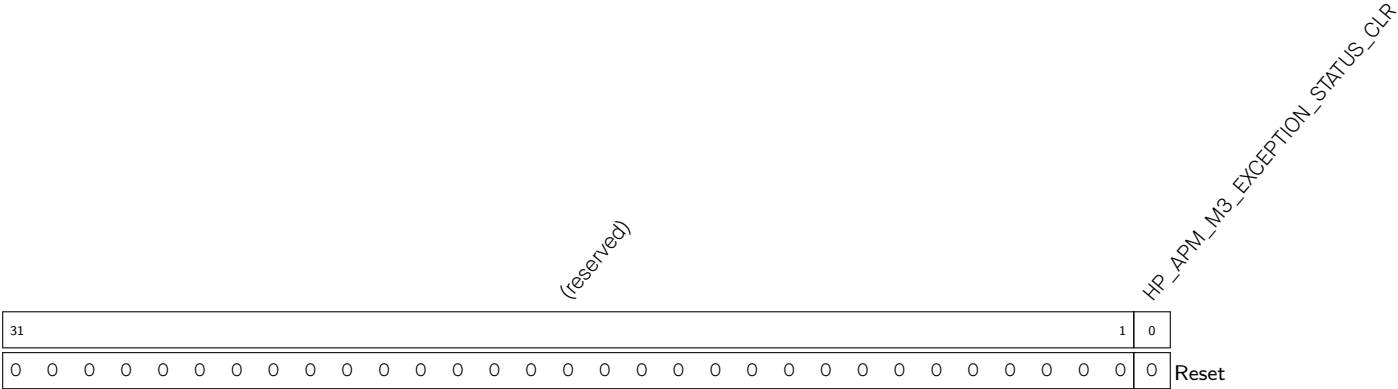


HP_APM_M3_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

bit1: 1 代表地址超限 (RO)

Register 13.19. HP_APM_M3_STATUS_CLR_REG (0x00FC)



HP_APM_M3_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.20. HP_APM_M3_EXCEPTION_INFO0_REG (0x0100)

(reserved)										HP_APM_M3_EXCEPTION_ID				HP_APM_M3_EXCEPTION_MODE				HP_APM_M3_EXCEPTION_REGION																																	
31									23	22			18	17	16	15																			0																
0	0	0	0	0	0	0	0	0	0	0		0				0																		Reset																	

- HP_APM_M3_EXCEPTION_REGION 表示异常区域。(RO)
- HP_APM_M3_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- HP_APM_M3_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.21. HP_APM_M3_EXCEPTION_INFO1_REG (0x0104)

HP_APM_M3_EXCEPTION_ADDR																															
31																															0
0																															
Reset																															

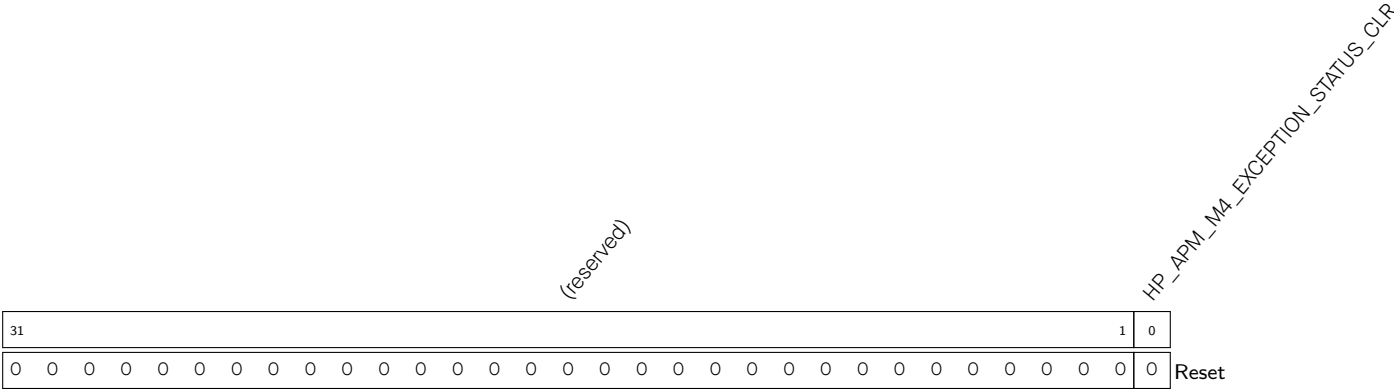
- HP_APM_M3_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.22. HP_APM_M4_STATUS_REG (0x0108)

(reserved)																															HP_APM_M4_EXCEPTION_STATUS				
31																															2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

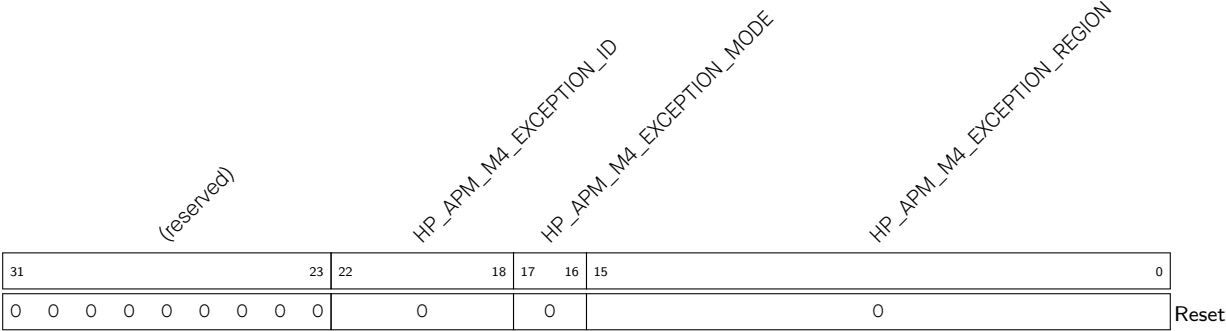
- HP_APM_M4_EXCEPTION_STATUS 表示异常状态。
- bit0: 1 代表权限不足
- bit1: 1 代表地址超限
- (RO)

Register 13.23. HP_APM_M4_STATUS_CLR_REG (0x010C)



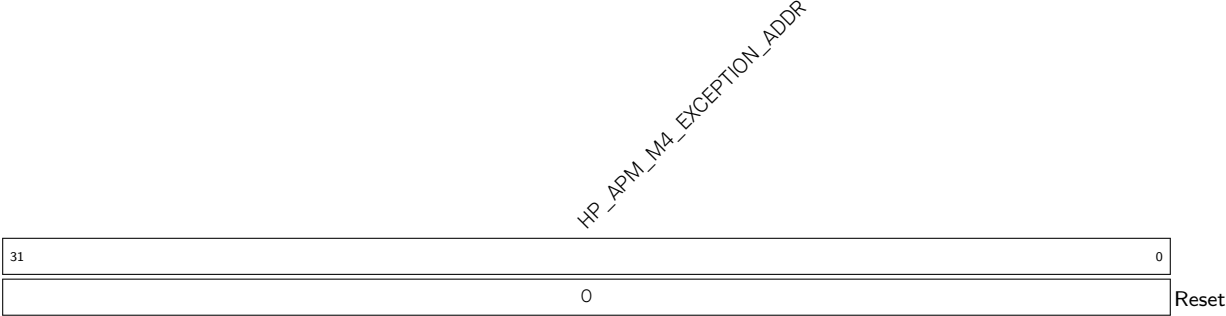
HP_APM_M4_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.24. HP_APM_M4_EXCEPTION_INFO0_REG (0x0110)



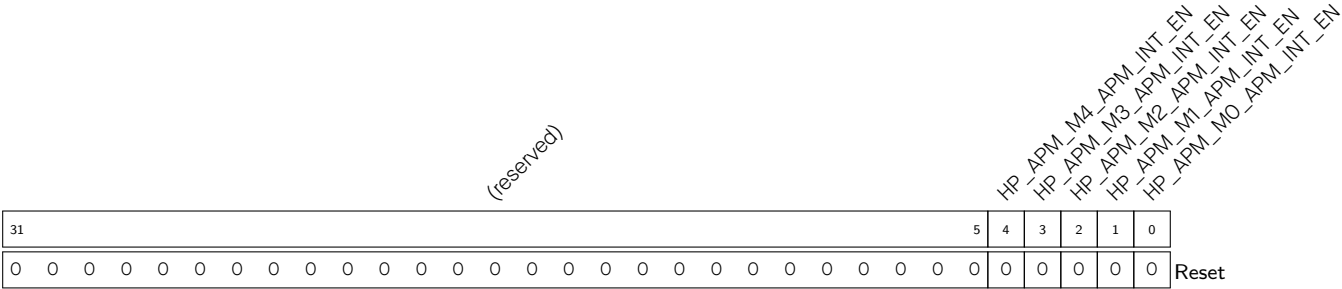
- HP_APM_M4_EXCEPTION_REGION 表示异常区域。(RO)
- HP_APM_M4_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- HP_APM_M4_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.25. HP_APM_M4_EXCEPTION_INFO1_REG (0x0114)



HP_APM_M4_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.26. HP_APM_INT_EN_REG (0x0118)



HP_APM_M0_APM_INT_EN 配置是否使能 HP_APM_CTRL M0 中断。

0: 禁用

1: 使能

(R/W)

HP_APM_M1_APM_INT_EN 配置是否使能 HP_APM_CTRL M1 中断。

0: 禁用

1: 使能

(R/W)

HP_APM_M2_APM_INT_EN 配置是否使能 HP_APM_CTRL M2 中断。

0: 禁用

1: 使能

(R/W)

HP_APM_M3_APM_INT_EN 配置是否使能 HP_APM_CTRL M3 中断。

0: 禁用

1: 使能

(R/W)

HP_APM_M4_APM_INT_EN 配置是否使能 HP_APM_CTRL M4 中断。

0: 禁用

1: 使能

(R/W)

Register 13.27. HP_APM_CLOCK_GATE_REG (0x07F8)

(reserved)																																HP_APM_CLK_EN		
31																															1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

HP_APM_CLK_EN 配置是否保持时钟始终开启。

0: 使能自动时钟门控

1: 保持时钟始终开启

(R/W)

Register 13.28. HP_APM_DATE_REG (0x07FC)

(reserved)				HP _APM_DATE																									Reset			
31			28	27	0																											
0	0	0	0	0x2312010																												

HP_APM_DATE 版本控制寄存器。(R/W)

13.9.2 LP_APM_REG

本小节的地址均为相对于 LP_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.29. LP_APM_REGION_FILTER_EN_REG (0x0000)

(reserved)																								LP_APM_REGION_FILTER_EN									
31																								8	7	0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0x1								Reset	

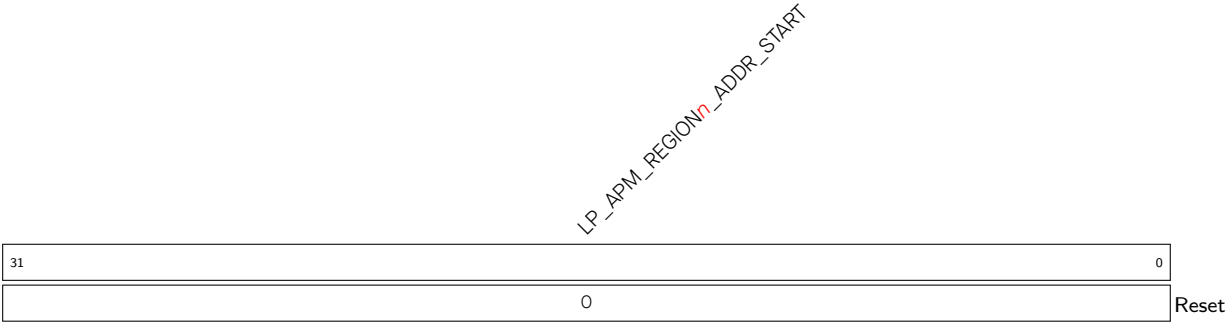
LP_APM_REGION_FILTER_EN 配置 bit *n* (0-7) 使能区域 *n* (0-7) 的地址范围。

0: 禁用

1: 使能

(R/W)

Register 13.30. LP_APM_REGION n _ADDR_START_REG (n : 0-7) (0x0004+0xC* n)



LP_APM_REGION n _ADDR_START 配置区域 n 的起始地址。(R/W)

Register 13.31. LP_APM_REGION n _ADDR_END_REG (n : 0-7) (0x0008+0xC* n)



LP_APM_REGION n _ADDR_END 配置区域 n 的结束地址。(R/W)

Register 13.32. LP_APM_REGIONn_ATTR_REG (n: 0-7) (0x000C+0xC*n)

(reserved)																								LP_APM_REGIONn_LOCK LP_APM_REGIONn_R2_R LP_APM_REGIONn_R2_W LP_APM_REGIONn_R2_X (reserved) LP_APM_REGIONn_R1_R LP_APM_REGIONn_R1_W LP_APM_REGIONn_R1_X (reserved) LP_APM_REGIONn_RO_R LP_APM_REGIONn_RO_W LP_APM_REGIONn_RO_X																		
31																								12				11	10	9	8	7	6	5	4	3	2	1	0			
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

LP_APM_REGIONn_RO_X 配置 REEO 模式下对区域 n 的执行权限。(R/W)

LP_APM_REGIONn_RO_W 配置 REEO 模式下对区域 n 的写入权限。(R/W)

LP_APM_REGIONn_RO_R 配置 REEO 模式下对区域 n 的读取权限。(R/W)

LP_APM_REGIONn_R1_X 配置 REE1 模式下对区域 n 的执行权限。(R/W)

LP_APM_REGIONn_R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)

LP_APM_REGIONn_R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)

LP_APM_REGIONn_R2_X 配置 REE2 模式下对区域 n 的执行权限。(R/W)

LP_APM_REGIONn_R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)

LP_APM_REGIONn_R2_R 配置 REE2 模式下对区域 n 的读取权限。(R/W)

LP_APM_REGIONn_LOCK 锁定区域 n 的首尾地址及权限的配置
(LP_APM_REGIONn_ADDR_START_REG、LP_APM_REGIONn_ADDR_END_REG 和
LP_APM_REGIONn_ATTR_REG)。

0: 不锁定

1: 锁定

(R/W)

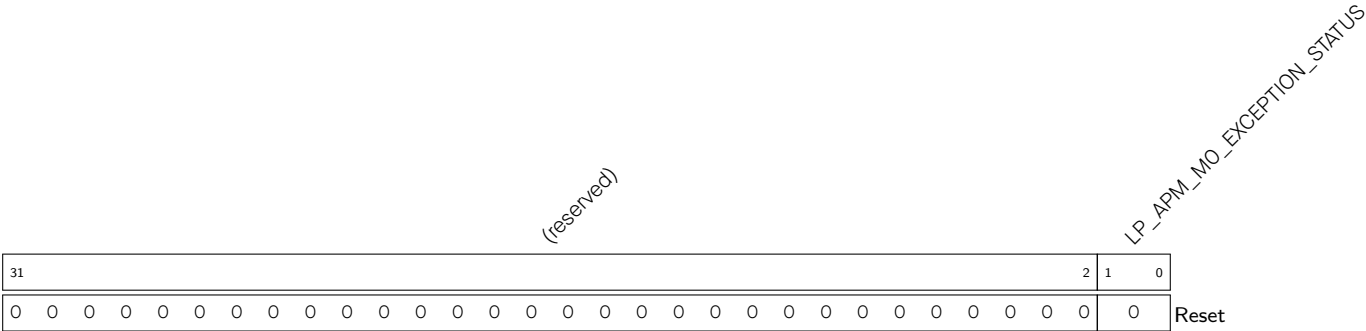
Register 13.33. LP_APM_FUNC_CTRL_REG (0x00C4)

(reserved)																								LP_APM_M1_FUNC_EN LP_APM_MO_FUNC_EN		
31																							2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1		

LP_APM_MO_FUNC_EN 配置使能 LP_APM_CTRL MO 权限检查。(R/W)

LP_APM_M1_FUNC_EN 配置使能 LP_APM_CTRL M1 权限检查。(R/W)

Register 13.34. LP_APM_MO_STATUS_REG (0x00C8)

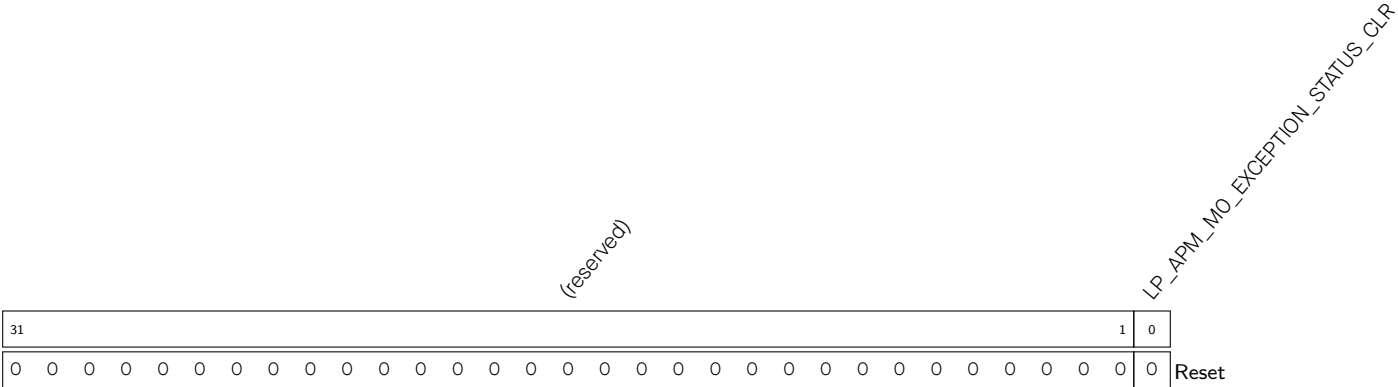


LP_APM_MO_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

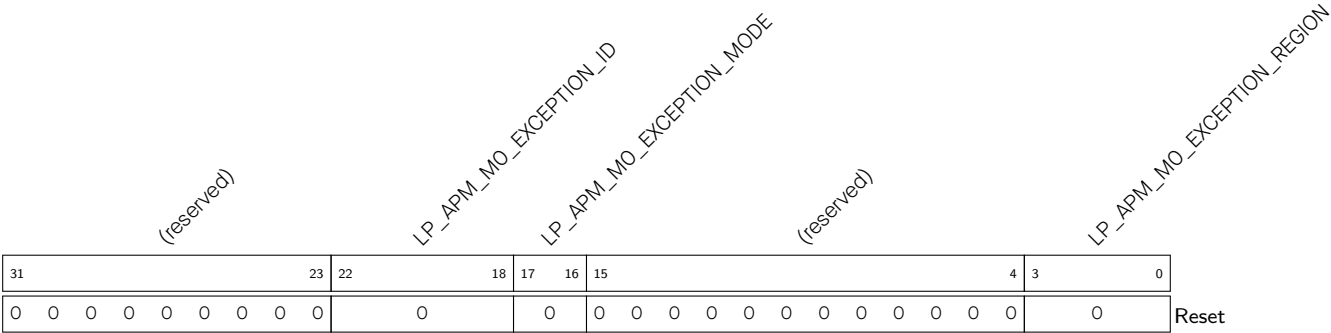
bit1: 1 代表地址超限 (RO)

Register 13.35. LP_APM_MO_STATUS_CLR_REG (0x00CC)



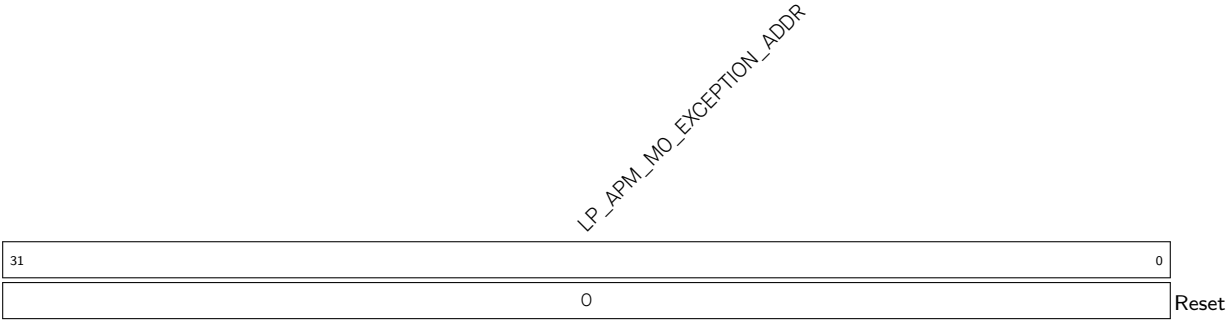
LP_APM_MO_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.36. LP_APM_MO_EXCEPTION_INFO0_REG (0x00D0)



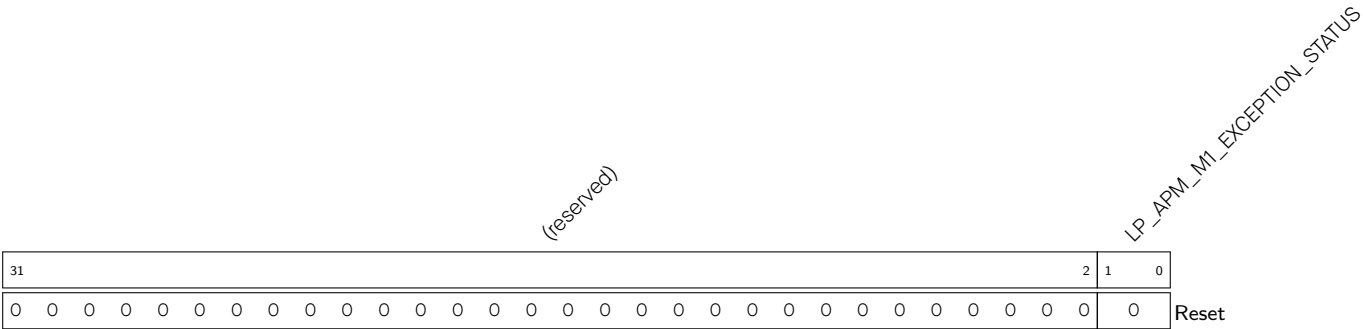
- LP_APM_MO_EXCEPTION_REGION 表示异常区域。(RO)
- LP_APM_MO_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- LP_APM_MO_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.37. LP_APM_MO_EXCEPTION_INFO1_REG (0x00D4)



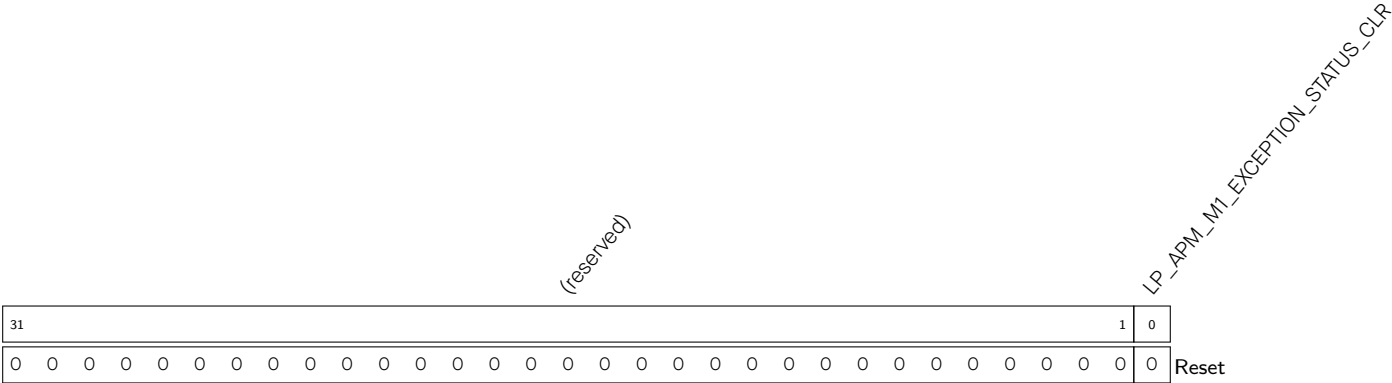
- LP_APM_MO_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.38. LP_APM_M1_STATUS_REG (0x00D8)



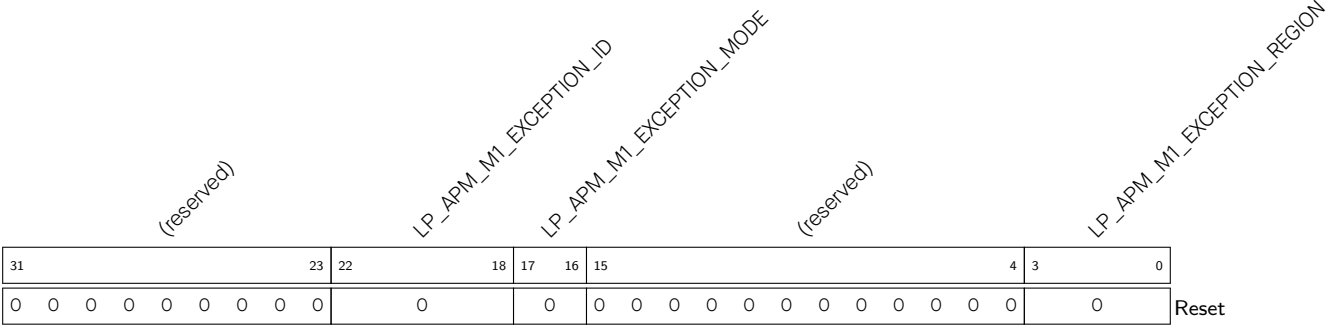
- LP_APM_M1_EXCEPTION_STATUS 表示异常状态。
 - bit0: 1 代表权限不足
 - bit1: 1 代表地址超限(RO)

Register 13.39. LP_APM_M1_STATUS_CLR_REG (0x00DC)



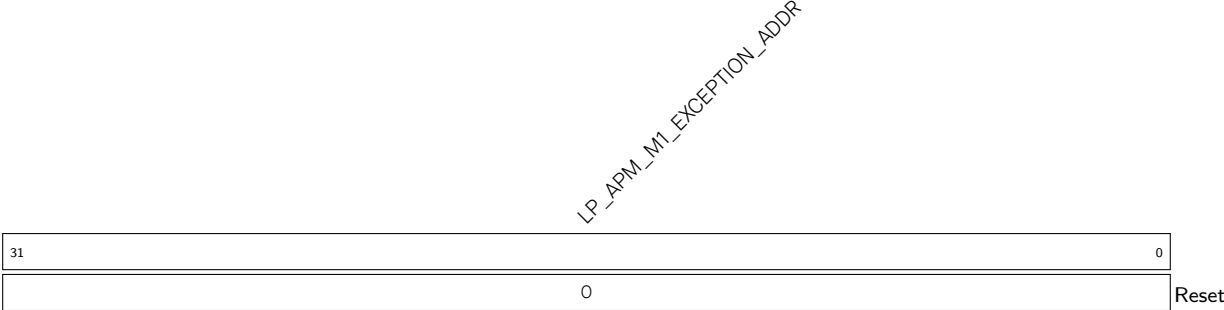
LP_APM_M1_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.40. LP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)



- LP_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)
- LP_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- LP_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.41. LP_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



LP_APM_M1_EXCEPTION_ADDR 表示异常地址。(RO)

Register 13.42. LP_APM_INT_EN_REG (0x00E8)

(reserved)																															LP_APM_M1_APM_INT_EN LP_APM_MO_APM_INT_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
31																															2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LP_APM_MO_APM_INT_EN 配置使能 LP_APM_CTRL MO 中断。

0: 禁用

1: 使能

(R/W)

LP_APM_M1_APM_INT_EN 配置使能 LP_APM_CTRL M1 中断。

0: 禁用

1: 使能

(R/W)

Register 13.43. LP_APM_CLOCK_GATE_REG (0x00EC)

(reserved)																											LP_APM_CLK_EN			
31																											1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

LP_APM_CLK_EN 配置是否保持时钟始终开启。

0: 使能自动时钟门控

1: 保持时钟始终开启

(R/W)

Register 13.44. LP_APM_DATE_REG (0x00FC)

(reserved)				LP_APM_DATE																							31	28	27	0	
0	0	0	0	0x2212160																											Reset

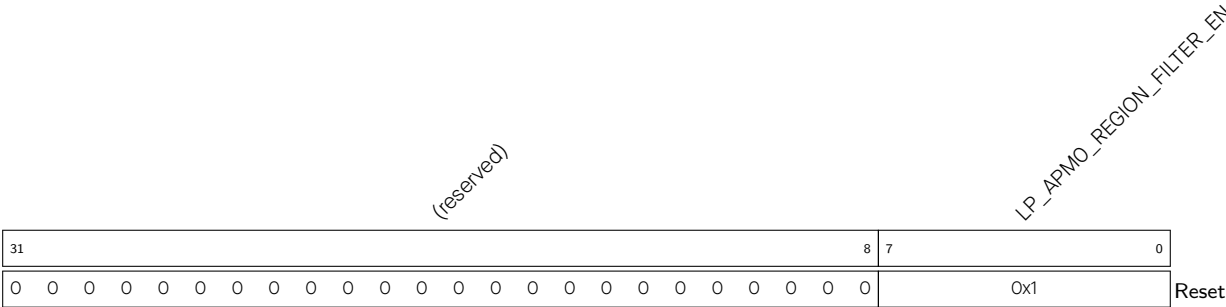
LP_APM_DATE 版本控制寄存器。(R/W)

13.9.3 LP_APMO_REG

本小节的地址均为相对于 LP_APMO 的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.45. LP_APMO_REGION_FILTER_EN_REG (0x0000)

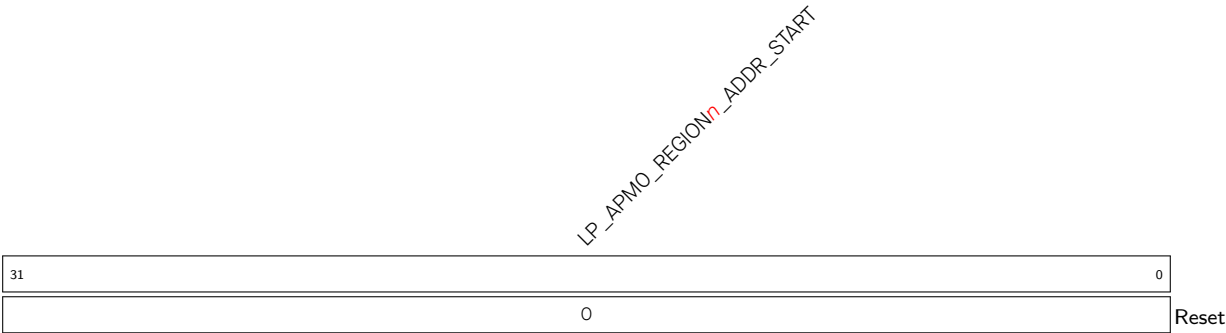


LP_APMO_REGION_FILTER_EN 配置 bit *n* (0-7) 使能区域 *n* (0-7)。

- 0: 禁用
- 1: 使能

(R/W)

Register 13.46. LP_APMO_REGION*n*_ADDR_START_REG (*n*: 0-7) (0x0004+0xC**n*)



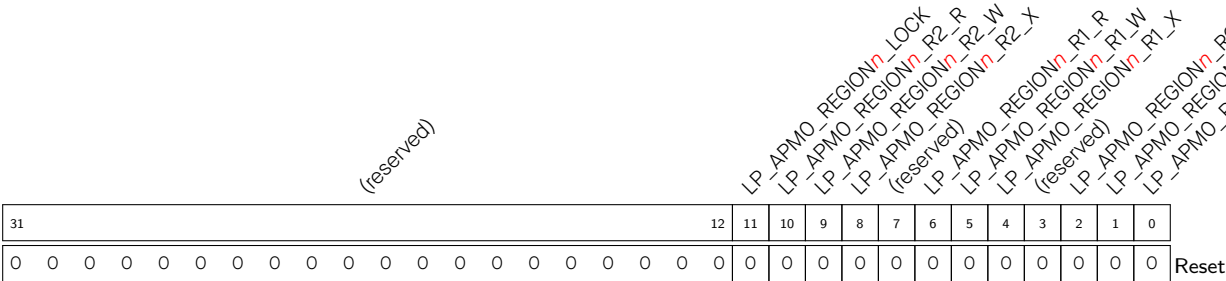
LP_APMO_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

Register 13.47. LP_APMO_REGION n _ADDR_END_REG (n : 0-7) (0x0008+0xC* n)



LP_APMO_REGION n _ADDR_END 配置区域 n 的结束地址。(R/W)

Register 13.48. LP_APMO_REGION n _ATTR_REG (n : 0-7) (0x000C+0xC* n)



- LP_APMO_REGION n _RO_X 配置 REEO 模式下对区域 n 的执行权限。(R/W)
- LP_APMO_REGION n _RO_W 配置 REEO 模式下对区域 n 的写入权限。(R/W)
- LP_APMO_REGION n _RO_R 配置 REEO 模式下对区域 n 的读取权限。(R/W)
- LP_APMO_REGION n _R1_X 配置 REE1 模式下对区域 n 的执行权限。(R/W)
- LP_APMO_REGION n _R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)
- LP_APMO_REGION n _R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)
- LP_APMO_REGION n _R2_X 配置 REE2 模式下对区域 n 的执行权限。(R/W)
- LP_APMO_REGION n _R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)
- LP_APMO_REGION n _R2_R 配置 REE2 模式下对区域 n 的读取权限。(R/W)
- LP_APMO_REGION n _LOCK 锁定区域 n 的首尾地址及权限的配置
(LP_APMO_REGION n _ADDR_START_REG、LP_APMO_REGION n _ADDR_END_REG 和
LP_APMO_REGION n _ATTR_REG)。
0: 不锁定
1: 锁定
(R/W)

Register 13.49. LP_APMO_FUNC_CTRL_REG (0x00C4)

(reserved)																																LP_APMO_MO_FUNC_EN		
31																															1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

LP_APMO_MO_FUNC_EN 配置使能 LP_APMO_CTRL MO 权限检查。(R/W)

Register 13.50. LP_APMO_MO_STATUS_REG (0x00C8)

(reserved)																															LP_APMO_MO_EXCEPTION_STATUS				
31																															2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_APMO_MO_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

bit1: 1 代表地址越限

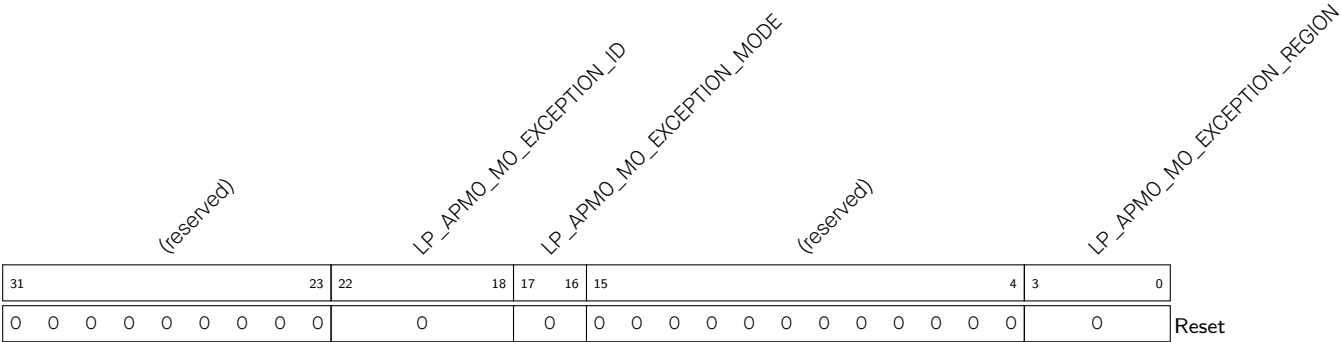
(RO)

Register 13.51. LP_APMO_MO_STATUS_CLR_REG (0x00CC)

(reserved)																															LP_APMO_MO_EXCEPTION_STATUS_CLR		
31																															1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_APMO_MO_EXCEPTION_STATUS_CLR 配置清除异常状态。(WT)

Register 13.52. LP_APMO_MO_EXCEPTION_INFO0_REG (0x00D0)

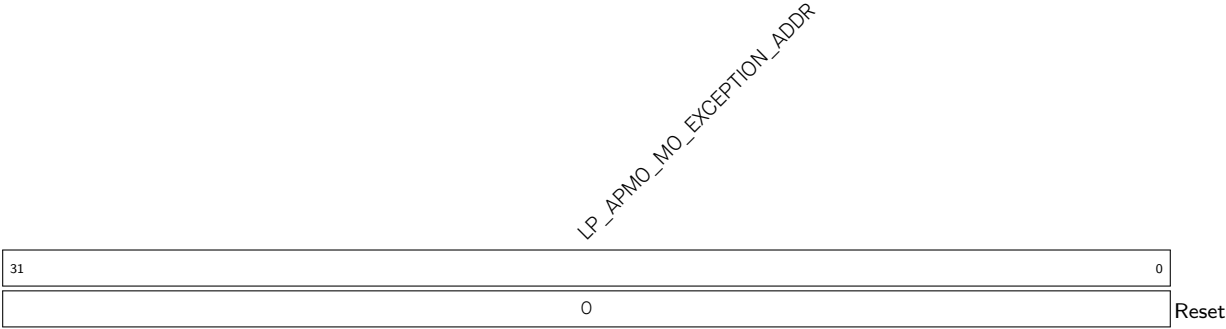


LP_APMO_MO_EXCEPTION_REGION 表示异常区域。(RO)

LP_APMO_MO_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

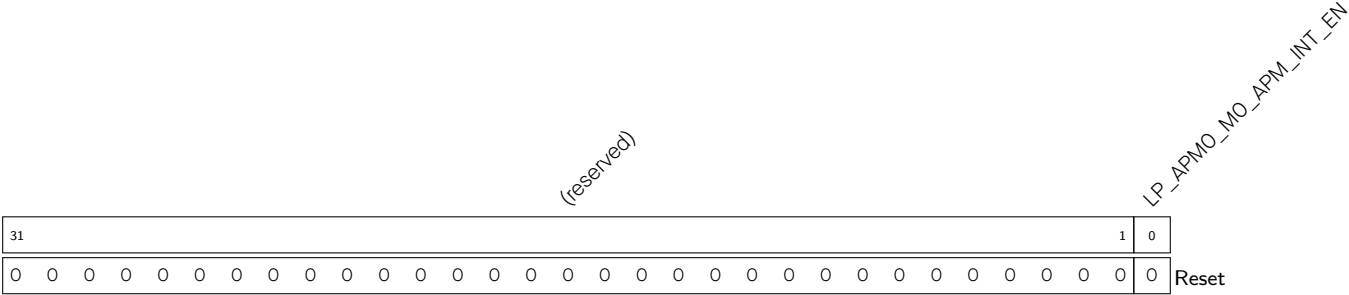
LP_APMO_MO_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.53. LP_APMO_MO_EXCEPTION_INFO1_REG (0x00D4)



LP_APMO_MO_EXCEPTION_ADDR 表示异常地址。(RO)

Register 13.54. LP_APMO_INT_EN_REG (0x00D8)



LP_APMO_MO_APM_INT_EN 配置是否使能 LP_APMO_CTRL MO 中断。

0：禁用

1：使能

(R/W)

Register 13.55. LP_APMO_CLOCK_GATE_REG (0x00DC)

(reserved)																														LP_APMO_CLK_EN		
31																														1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

LP_APMO_CLK_EN 配置是否保持时钟始终开启。

0: 使能自动时钟门控

1: 保持时钟始终开启

(R/W)

Register 13.56. LP_APMO_DATE_REG (0x07FC)

(reserved)				LP_APMO_DATE																											
31	28	27																											0		
0	0	0	0	0x2212160																										Reset	

LP_APMO_DATE 版本控制寄存器。(R/W)

13.9.4 CPU_APM_REG

本小节的地址均为相对于 CPU_APM 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.57. CPU_APM_REGION_FILTER_EN_REG (0x0000)

(reserved)																								CPU_APM_REGION_FILTER_EN															
31																								7								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0x1								Reset							

CPU_APM_REGION_FILTER_EN 配置 bit *n* (0-7) 使能区域 *n* (0-7) 的地址范围。

0: 禁用

1: 使能

(R/W)

Register 13.58. CPU_APM_REGION n _ADDR_START_REG (n : 0-7) (0x0004+0xC* n)



CPU_APM_REGION n _ADDR_START_L Low 12 bit, start address of region n . (HRO)

CPU_APM_REGION n _ADDR_START 配置区域 n 的起始地址。(R/W)

CPU_APM_REGION n _ADDR_START_H High 13 bit, start address of region n . (HRO)

Register 13.59. CPU_APM_REGION n _ADDR_END_REG (n : 0-7) (0x0008+0xC* n)



CPU_APM_REGION n _ADDR_END_L Low 12 bit, end address of region n . (HRO)

CPU_APM_REGION n _ADDR_END 配置区域 n 的结束地址。(R/W)

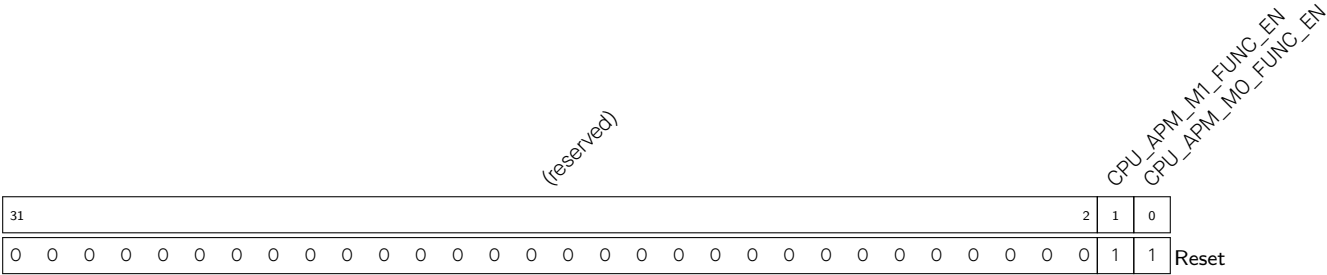
CPU_APM_REGION n _ADDR_END_H High 13 bit, end address of region n . (HRO)

Register 13.60. CPU_APM_REGION n _ATTR_REG (n : 0-7) (0x000C+0xC* n)

(reserved)																				CPU_APM_REGION _n _LOCK CPU_APM_REGION _n _R2_R CPU_APM_REGION _n _R2_W (reserved) CPU_APM_REGION _n _R2_X CPU_APM_REGION _n _R1_R (reserved) CPU_APM_REGION _n _R1_W CPU_APM_REGION _n _R1_X CPU_APM_REGION _n _RO_R CPU_APM_REGION _n _RO_W CPU_APM_REGION _n _RO_X												
31												12								11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

- CPU_APM_REGION n _RO_X 配置 REEO 模式下对区域 n 的执行权限。(R/W)
- CPU_APM_REGION n _RO_W 配置 REEO 模式下对区域 n 的写入权限。(R/W)
- CPU_APM_REGION n _RO_R 配置 REEO 模式下对区域 n 的读取权限。(R/W)
- CPU_APM_REGION n _R1_X 配置 REE1 模式下对区域 n 的执行权限。(R/W)
- CPU_APM_REGION n _R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)
- CPU_APM_REGION n _R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)
- CPU_APM_REGION n _R2_X 配置 REE2 模式下对区域 n 的执行权限。(R/W)
- CPU_APM_REGION n _R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)
- CPU_APM_REGION n _R2_R 配置 REE2 模式下对区域 n 的读取权限。(R/W)
- CPU_APM_REGION n _LOCK 锁定区域 n 的首尾地址及权限的配置
(CPU_APM_REGION n _ADDR_START_REG、CPU_APM_REGION n _ADDR_END_REG、和
CPU_APM_REGION n _ATTR_REG)。
0: 不锁定
1: 锁定
(R/W)

Register 13.61. CPU_APM_FUNC_CTRL_REG (0x00C4)



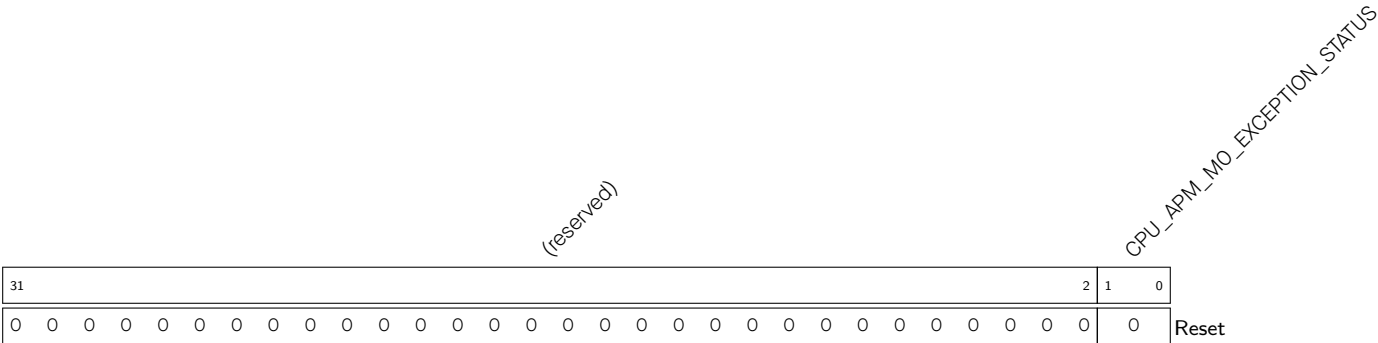
CPU_APM_MO_FUNC_EN 配置使能 CPU_APM_CTRL MO 权限检查。

0: 禁用
1: 使能
(R/W)

CPU_APM_M1_FUNC_EN 配置使能 CPU_APM_CTRL M1 权限检查。

0: 禁用
1: 使能
(R/W)

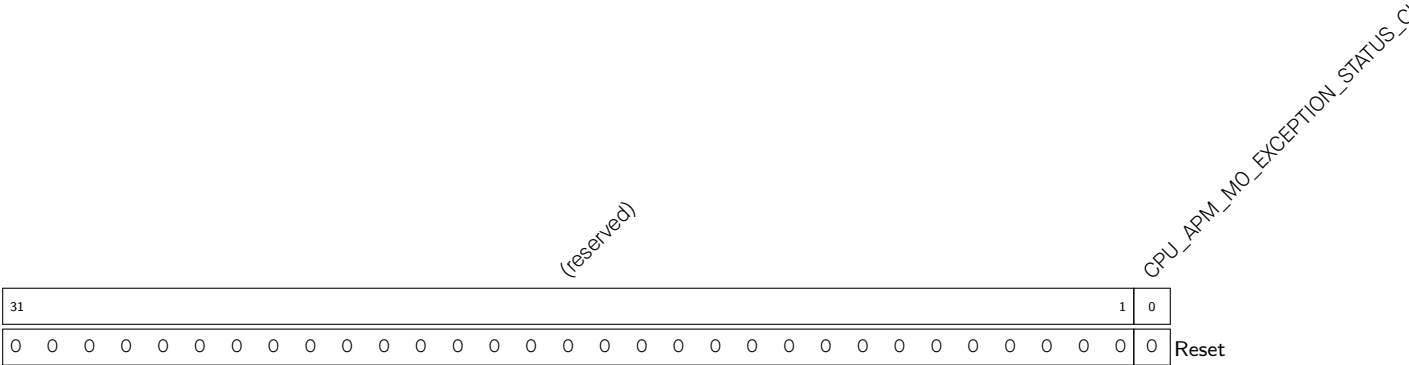
Register 13.62. CPU_APM_MO_STATUS_REG (0x00C8)



CPU_APM_MO_EXCEPTION_STATUS 表示异常状态。

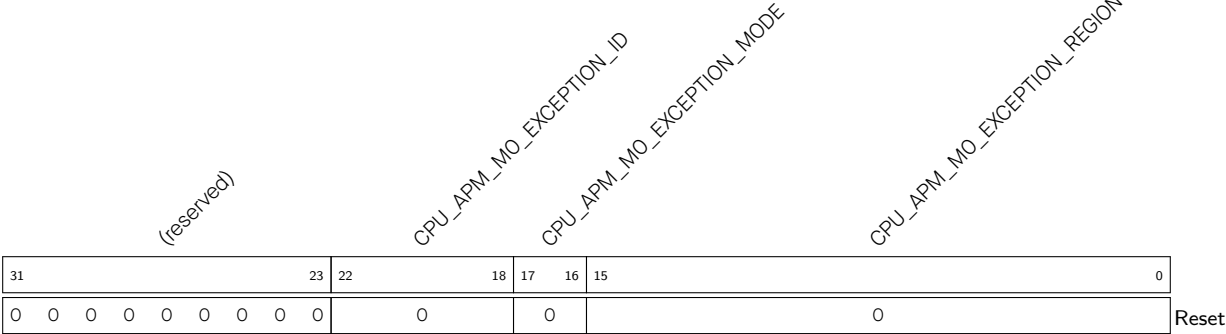
bit0: 1 代表权限不足
bit1: 1 代表地址超限
(RO)

Register 13.63. CPU_APM_MO_STATUS_CLR_REG (0x00CC)



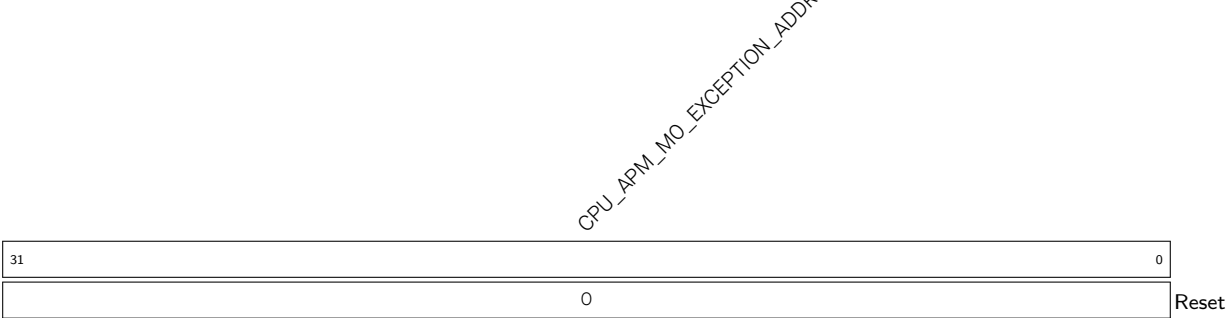
CPU_APM_MO_EXCEPTION_STATUS_CLR 写 1 清除异常状态。(WT)

Register 13.64. CPU_APM_MO_EXCEPTION_INFO0_REG (0x00D0)



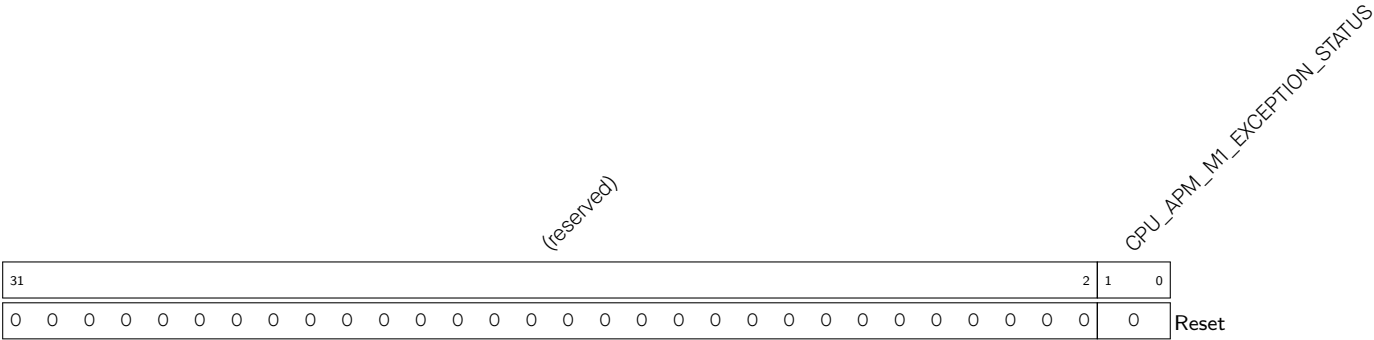
- CPU_APM_MO_EXCEPTION_REGION 表示异常区域。(RO)
- CPU_APM_MO_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- CPU_APM_MO_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.65. CPU_APM_MO_EXCEPTION_INFO1_REG (0x00D4)



CPU_APM_MO_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.66. CPU_APM_M1_STATUS_REG (0x00D8)

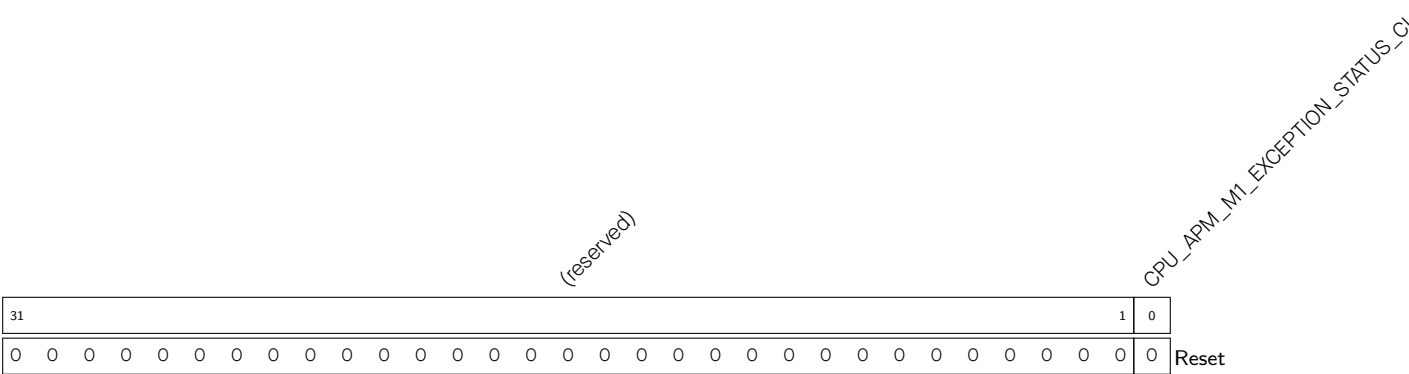


CPU_APM_M1_EXCEPTION_STATUS 表示异常状态。

bit0: 1 代表权限不足

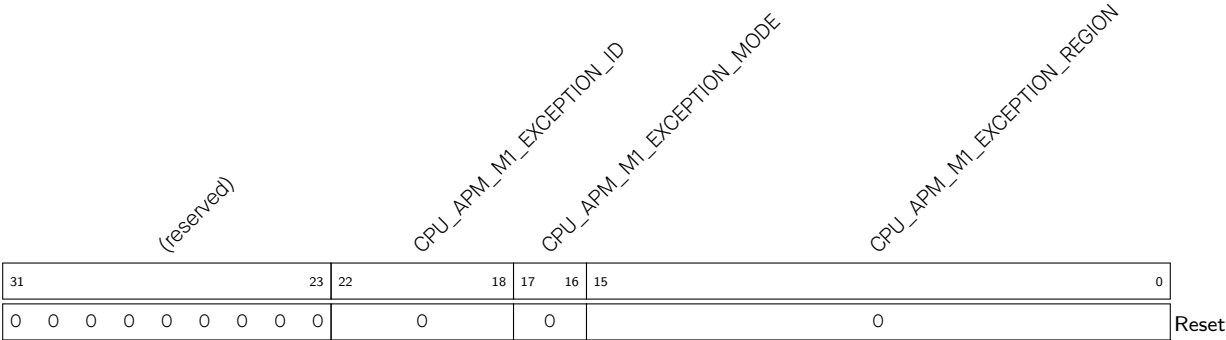
bit1: 1 代表地址超限 (RO)

Register 13.67. CPU_APM_M1_STATUS_CLR_REG (0x00DC)



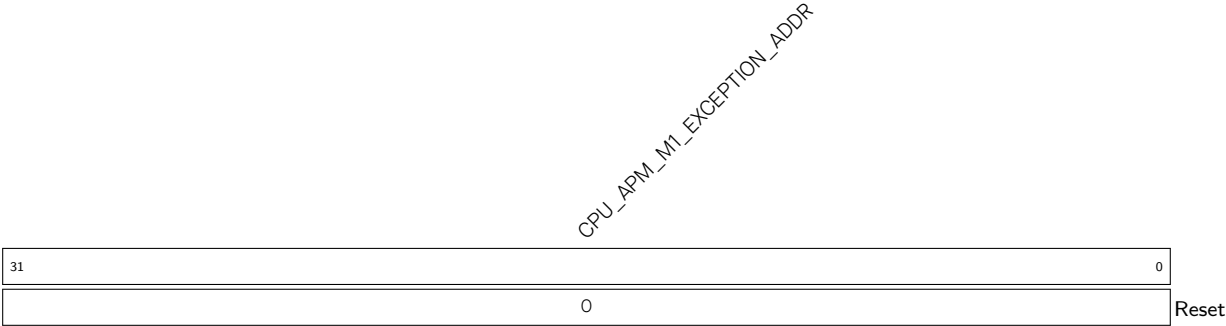
CPU_APM_M1_EXCEPTION_STATUS_CLR 写 1 清除异常状态。(WT)

Register 13.68. CPU_APM_M1_EXCEPTION_INFO0_REG (0x00E0)



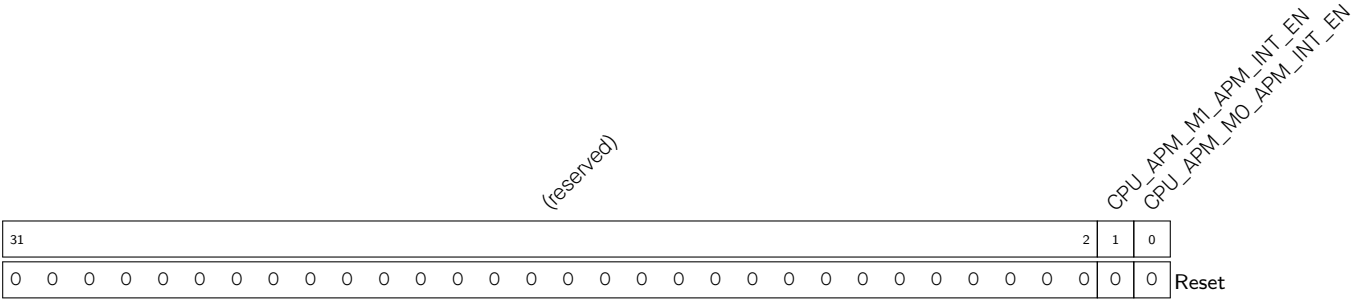
- CPU_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)
- CPU_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)
- CPU_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 13.69. CPU_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



- CPU_APM_M1_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 13.70. CPU_APM_INT_EN_REG (0x0118)



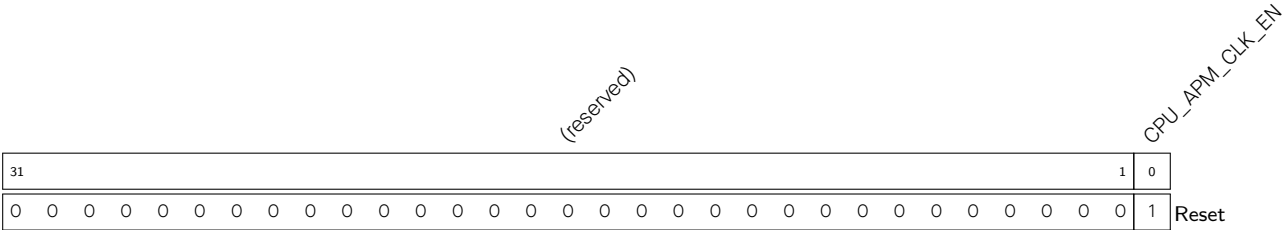
CPU_APM_MO_APM_INT_EN 配置是否使能 CPU_APM_CTRL MO 中断。

0: 禁用
1: 使能
(R/W)

CPU_APM_M1_APM_INT_EN 配置是否使能 CPU_APM_CTRL M1 中断。

0: 禁用
1: 使能
(R/W)

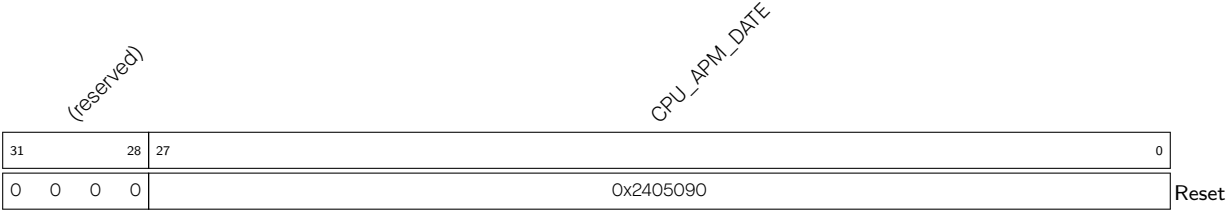
Register 13.71. CPU_APM_CLOCK_GATE_REG (0x07F8)



CPU_APM_CLK_EN 配置是否保持时钟始终开启。

0: 使能自动时钟门控
1: 保持时钟始终开启
(R/W)

Register 13.72. CPU_APM_DATE_REG (0x07FC)



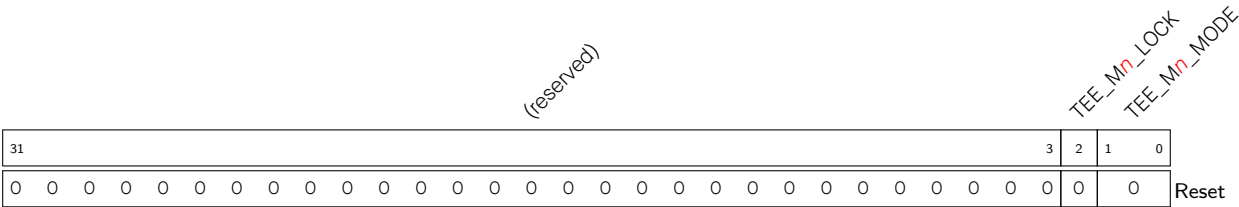
CPU_APM_DATE 版本控制寄存器。(R/W)

13.9.5 TEE_REG

本小节的地址均为相对于 TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.73. TEE_Mn_MODE_CTRL_REG (n: 0-31) (0x0000+0x4*n)



- TEE_Mn_MODE 配置主机 n 的安全模式。
0: TEE
1: REEO
2: REE1
3: REE2
(R/W)
- TEE_Mn_LOCK 配置是否锁定 TEE_Mn_MODE 的值。
0: 不锁定
1: 锁定
(R/W)

- Register 13.74. TEE_UART0_CTRL_REG (0x0088)
- Register 13.75. TEE_UART1_CTRL_REG (0x008C)
- Register 13.76. TEE_UHCIO_CTRL_REG (0x0090)
- Register 13.77. TEE_I2C_EXT0_CTRL_REG (0x0094)
- Register 13.78. TEE_I2S_CTRL_REG (0x009C)
- Register 13.79. TEE_PARL_IO_CTRL_REG (0x00A0)
- Register 13.80. TEE_PWM_CTRL_REG (0x00A4)
- Register 13.81. TEE_LEDC_CTRL_REG (0x00AC)
- Register 13.82. TEE_TWAIO_CTRL_REG (0x00B0)
- Register 13.83. TEE_USB_SERIAL_JTAG_CTRL_REG (0x00B4)
- Register 13.84. TEE_RMT_CTRL_REG (0x00B8)
- Register 13.85. TEE_GDMA_CTRL_REG (0x00BC)
- Register 13.86. TEE_ETM_CTRL_REG (0x00C4)
- Register 13.87. TEE_INTMTX_CTRL_REG (0x00C8)
- Register 13.88. TEE_APB_ADC_CTRL_REG (0x00D0)

Register 13.110. TEE_BUS_ERR_CONF_REG (0x0FF0)

(reserved)																												TEE_BUS_ERR_RESP_EN	
31																												1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

TEE_BUS_ERR_RESP_EN 配置是否使能非法访问返回总线错误信息的功能。

0: 禁用

1: 使能

(R/W)

Register 13.111. TEE_CLOCK_GATE_REG (0x0FF8)

(reserved)																												TEE_CLK_EN	
31																												1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Reset

TEE_CLK_EN 配置是否保持时钟始终开启。

0: 使能自动时钟门控

1: 保持时钟始终开启

(R/W)

Register 13.112. TEE_DATE_REG (0x0FFC)

(reserved)				TEE_DATE																										
31				28	27																								0	
0	0	0	0	0x2406200																										Reset

TEE_DATE 版本控制寄存器。(R/W)

13.9.6 LP_TEE_REG

本小节的地址均为相对于 LP_TEE 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 13.113. LP_TEE_MO_MODE_CTRL_REG (0x0000)

(reserved)																																LP_TEE_MO_LOCK LP_TEE_MO_MODE			
31																															3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	Reset

- LP_TEE_MO_MODE

配置 LP CPU 的安全模式。

0: TEE

1: REEO

2: REE1

3: REE2

(R/W)
- LP_TEE_MO_LOCK

配置是否锁定 LP_TEE_MO_MODE 的值。

0: 不锁定

1: 锁定

(R/W)

- Register 13.114. LP_TEE_EFUSE_CTRL_REG (0x0004)
- Register 13.115. LP_TEE_PMU_CTRL_REG (0x0008)
- Register 13.116. LP_TEE_CLKRST_CTRL_REG (0x000C)
- Register 13.117. LP_TEE_LP_AON_CTRL_CTRL_REG (0x0010)
- Register 13.118. LP_TEE_LP_TIMER_CTRL_REG (0x0014)
- Register 13.119. LP_TEE_LP_WDT_CTRL_REG (0x0018)
- Register 13.120. LP_TEE_LP_PERI_CTRL_REG (0x001C)
- Register 13.121. LP_TEE_LP_ANA_PERI_CTRL_REG (0x0020)
- Register 13.122. LP_TEE_LP_IO_CTRL_REG (0x002C)
- Register 13.123. LP_TEE_LP_TEE_CTRL_REG (0x0034)
- Register 13.124. LP_TEE_UART_CTRL_REG (0x0038)
- Register 13.125. LP_TEE_I2C_EXT_CTRL_REG (0x0040)
- Register 13.126. LP_TEE_I2C_ANA_MST_CTRL_REG (0x0044)
- Register 13.127. LP_TEE_LP_APM_CTRL_REG (0x004C)

Register 13.128. LP_TEE_ *PERI* _CTRL_REG (0x0004-0x004C)

(reserved)																								LP_TEE_WRITE_REE2_PERI LP_TEE_WRITE_REE1_PERI LP_TEE_WRITE_REEO_PERI LP_TEE_READ_REE2_PERI LP_TEE_READ_REE1_PERI LP_TEE_READ_TEE_PERI										
31																								8	7	6	5	4	3	2	1	0		
0																								0	0	0	1	0	0	0	1	Reset		

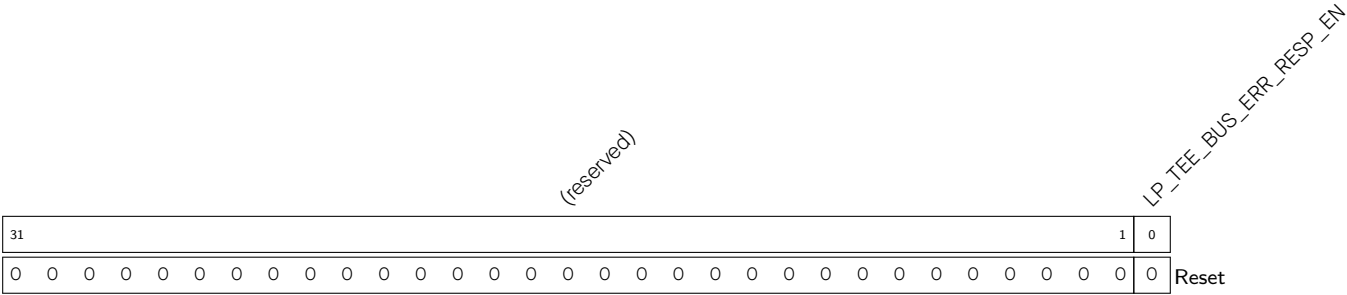
- LP_TEE_READ_TEE_ *PERI* 配置 TEE 模式下 *PERI* 的读取权限。(R/W)
- LP_TEE_READ_REEO_ *PERI* 配置 REEO 模式下 *PERI* 的读取权限。(R/W)
- LP_TEE_READ_REE1_ *PERI* 配置 REE1 模式下 *PERI* 的读取权限。(R/W)
- LP_TEE_READ_REE2_ *PERI* 配置 REE2 模式下 *PERI* 的读取权限。(R/W)
- LP_TEE_WRITE_TEE_ *PERI* 配置 TEE 模式下 *PERI* 的写入权限。(R/W)
- LP_TEE_WRITE_REEO_ *PERI* 配置 REEO 模式下 *PERI* 的写入权限。(R/W)
- LP_TEE_WRITE_REE1_ *PERI* 配置 REE1 模式下 *PERI* 的写入权限。(R/W)
- LP_TEE_WRITE_REE2_ *PERI* 配置 REE2 模式下 *PERI* 的写入权限。(R/W)

Register 13.129. LP_TEE_FORCE_ACC_HP_REG (0x0090)

(reserved)																LP_TEE_FORCE_ACC_HPMEM_EN			
31															1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																0	Reset		

- LP_TEE_FORCE_ACC_HPMEM_EN 配置是否允许 LP CPU 不经过 SYS_APM 控制器的权限检查而强制访问 HP SRAM。
- 0: 不允许
- 1: 允许
- (R/W)

Register 13.130. LP_TEE_BUS_ERR_CONF_REG (0x00F0)



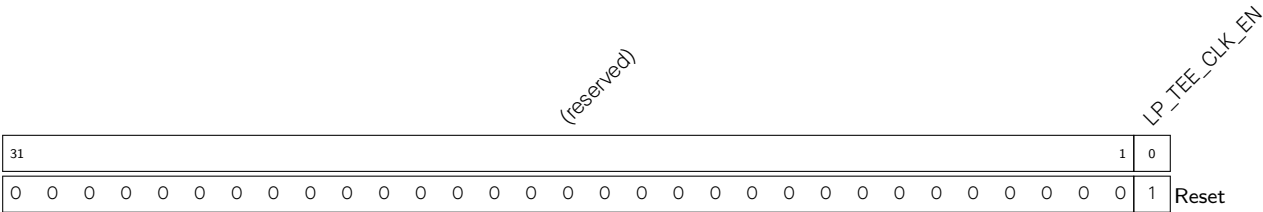
LP_TEE_BUS_ERR_RESP_EN 配置是否使能非法访问返回总线错误信息的功能。

0: 禁用

1: 使能

(R/W)

Register 13.131. LP_TEE_CLOCK_GATE_REG (0x00F8)



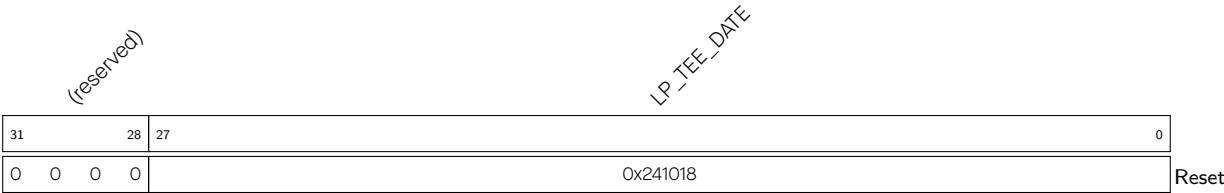
LP_TEE_CLK_EN 配置是否保持时钟始终开启。

0: 使能自动时钟门控

1: 保持时钟始终开启

(R/W)

Register 13.132. LP_TEE_DATE_REG (0x00FC)



LP_TEE_DATE 版本控制寄存器。(R/W)

第 14 章

辅助调试

14.1 概述

ESP32-C5 辅助调试模块提供一套调试功能，可在软件开发时用于调试，定位问题所在。

14.2 主要特性

辅助调试模块具有以下特性：

- **区域读写监测**：监测高性能处理器 (High-Performance CPU, HP CPU) 总线是否在限定的存储器地址范围内进行读写操作，若在该地址范围内发生读写操作则触发中断。
- **栈指针 (SP) 监测**：监测栈指针是否超出限定的范围，若超出范围则产生中断。
- **程序计数器 (PC) 记录**：记录 PC，可以获取上一次 HP CPU 复位时的 PC 值。
- **总线访问记录**：记录总线访问信息，当 HP CPU、低功耗处理器 (Low-Power CPU, LP CPU) 或 DMA 写了某个特殊值时，会记录此次写操作的总线类型、地址和 PC 值（仅记录 HP CPU 写操作的 PC），并将这些信息记录到 HP SRAM 中。

14.3 功能描述

14.3.1 区域读写监测

为确认 HP CPU 是否在某段地址范围（即区域）进行过读写操作，辅助调试模块能够监测 HP CPU 总线，当总线在监测地址范围进行读写操作时会触发中断，详情请见章节 1 [高性能处理器 \[to be added later\]](#) 中的表 1。

当 HP CPU 总线访问数据空间时，本章节将其称为**数据总线**；而当 HP CPU 总线访问 AHB 外设空间时，本章节将其称为**外设总线**。辅助调试模块可在数据总线上同时监测在两段地址范围内的读写操作，假设命名为 HP CPU 数据总线区域 0 和 HP CPU 数据总线区域 1；也可在外设总线上同时监测两段地址范围，假设命名为 HP CPU 外设总线区域 0 和 HP CPU 外设总线区域 1。区域 0 和区域 1 根据需要进行配置。

详细区域读写监测模式如下。

- 数据总线读写监测
 - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 0 是否进行读操作
 - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 0 是否进行写操作
 - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 1 是否进行读操作
 - 监测 HP CPU 数据总线在 HP CPU 数据总线区域 1 是否进行写操作

- 外设总线读写监测
 - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 0 是否进行读操作
 - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 0 是否进行写操作
 - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 1 是否进行读操作
 - 监测 HP CPU 外设总线在 HP CPU 外设总线区域 1 是否进行写操作

14.3.2 栈指针监测

为防止 HP CPU 栈溢出或者错误的压栈出栈，辅助调试模块能够监测栈指针。当 HP CPU 栈指针超过 HP CPU 栈指针监测区域限定的上下边界时，模块会记录 PC 指针并产生中断，边界由软件进行配置。

详细栈指针监测模式如下。

- 监测 HP CPU 栈指针是否大于 HP CPU 栈指针监测区域上边界
- 监测 HP CPU 栈指针是否小于 HP CPU 栈指针监测区域下边界

14.3.3 PC 记录

在某些情况下，软件开发者希望知道上次 HP CPU 复位时的 PC 指针。比如，程序卡死只能复位时，开发者可能希望读取复位时的 PC 指针，以便知道程序在哪里卡死，然后进行调试。辅助调试模块可以记录 HP CPU 复位时的 PC，方便开发者进行调试。

14.3.4 CPU/DMA 总线访问记录

辅助调试模块能够实时记录 HP CPU 数据总线、LP CPU 总线和 DMA 总线的写操作，当总线在某个范围内发生写操作或者在某个范围内写某个特定的值时，此次操作的总线类型、地址和 PC（仅记录 HP CPU 写操作的 PC）等信息会被记录下来，并按照一定的格式存储到 HP SRAM 中。此处提到的地址范围只能处于 HP SRAM 或外部 RAM 的范围区间。

14.4 中断

ESP32-C5 的辅助调试模块可以生成 BUS_MONITOR_INTR 中断信号并发送给[中断矩阵](#)。这一中断信号由以下辅助调试模块的内部中断源生成。

- BUS_MONITOR_CORE_0_AREA_DRAMO_0_RD_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 0 读操作时触发。
- BUS_MONITOR_CORE_0_AREA_DRAMO_0_WR_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 0 写操作时触发。
- BUS_MONITOR_CORE_0_AREA_DRAMO_1_RD_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 1 读操作时触发。
- BUS_MONITOR_CORE_0_AREA_DRAMO_1_WR_INT：当 HP CPU 数据总线在 HP CPU 数据总线区域 1 写操作时触发。
- BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INT：当 HP CPU 外设总线在 HP CPU 外设总线区域 0 读操作时触发。

- BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INT: 当 HP CPU 外设总线在 HP CPU 外设总线区域 0 写操作时时触发。
- BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INT: 当 HP CPU 外设总线在 HP CPU 外设总线区域 1 读操作时时触发。
- BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INT: 当 HP CPU 外设总线在 HP CPU 外设总线区域 1 写操作时时触发。
- BUS_MONITOR_CORE_0_SP_SPILL_MIN_INT: 当 HP CPU 栈指针小于 HP CPU 栈指针监测区域的下边界时触发。
- BUS_MONITOR_CORE_0_SP_SPILL_MAX_INT: 当 HP CPU 栈指针大于 HP CPU 栈指针监测区域的上边界时触发。

说明:

对于中断、中断信号、中断源及其关系的详细描述, 可以查看章节 3 中断矩阵 [to be added later] > Section 3.1 中断术语。

每个中断源都有一组通用的配置寄存器, 详见章节 中断配置寄存器。具体寄存器名称请查看 14.6 寄存器列表。

14.5 配置流程

14.5.1 区域监测和栈指针监测配置

区域监测和栈指针监测的配置流程如下。

1. 配置监测区域和栈指针。
 - HP CPU0 数据总线区域 0 由 BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG 进行配置。
 - HP CPU 数据总线区域 1 由 BUS_MONITOR_CORE_0_AREA_DRAMO_1_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_DRAMO_1_MAX_REG 进行配置。
 - HP CPU 外设总线区域 0 由 BUS_MONITOR_CORE_0_AREA_PIF_0_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_PIF_0_MAX_REG 进行配置。
 - HP CPU 外设总线区域 1 由 BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG 和 BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG 进行配置。
 - HP CPU 栈指针监测区域由 BUS_MONITOR_CORE_0_SP_MIN_REG 和 BUS_MONITOR_CORE_0_SP_MAX_REG 进行配置。
2. 配置 BUS_MONITOR_CORE_0_INTR_ENA_REG 使能不同模式的中断。
3. 配置 BUS_MONITOR_CORE_0_MONTR_ENA_REG 使能不同的监测模式, 可同时使能。
4. 读取 BUS_MONITOR_CORE_0_INTR_RAW_REG 查询不同模式的原始中断状态。
5. 配置 BUS_MONITOR_CORE_0_INTR_CLR_REG 清除不同模式的中断。

比如，若想监测 HP CPU 数据总线地址 A 到地址 B 范围内是否进行过写操作，可通过 HP CPU 数据总线区域 0 或者区域 1 进行监测。以 HP CPU 数据总线区域 0 为例：

1. 配置 `BUS_MONITOR_CORE_0_RCD_PDEBUGEN` 为 1，使能 HP CPU 更新 PC 信号给辅助调试模块。
2. 配置 `BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG` 为 A 地址。
3. 配置 `BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG` 为 B 地址。
4. 配置 `BUS_MONITOR_CORE_0_INTR_ENA_REG` bit[1] 使能 HP CPU 数据总线区域 0 写监测中断。
5. 配置 `BUS_MONITOR_CORE_0_MONTR_ENA_REG` bit[1] 使能 HP CPU 数据总线区域 0 写监测。
6. 配置中断矩阵将 `BUS_MONITOR_INTR` 映射到 HP CPU 中断上（参考章节 3 中断矩阵 [to be added later]）。
7. 触发中断后：
 - 读取 `BUS_MONITOR_CORE_0_INTR_RAW_REG` 确认中断源。
 - 如果是 HP CPU 区域监测引发的中断，读取 `BUS_MONITOR_CORE_0_AREA_PC` 可获取触发中断时刻的 HP CPU PC 值，读取 `BUS_MONITOR_CORE_0_AREA_SP` 可获取触发中断时刻的 HP CPU 栈指针。
 - 如果是栈指针监测引发的中断，读取 `BUS_MONITOR_CORE_0_SP_PC` 可获取触发中断时刻的 HP CPU PC 值。
 - 配置 `BUS_MONITOR_CORE_0_INTR_CLR_REG` 对应位为 1 可清除中断。

14.5.2 PC 记录配置

配置 `BUS_MONITOR_CORE_0_RCD_PDEBUGEN` 为 1，使能 HP CPU 更新 PC 信号给辅助调试模块。若 `BUS_MONITOR_CORE_0_RCD_RECORDEN` 也配置为 1，`BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG` 会记录 HP CPU PC 信号，同时 `BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG` 会记录 HP CPU 栈指针，否则保留原值。

当 HP CPU 发生复位时，`BUS_MONITOR_CORE_0_RCD_EN_REG` 会被复位，但是 `BUS_MONITOR_CORE_0_RCD_PDEBUGPC_REG` 和 `BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG` 不会被复位，因此这两个寄存器会一直保留 HP CPU 复位时刻的 PC 值和栈指针。

14.5.3 CPU/DMA 总线访问记录配置

14.5.3.1 总线访问记录 1 配置

总线访问记录 1 包括 HP CPU 总线访问记录（HP SRAM 和外部 RAM 监控功能）、LP CPU 总线访问记录（HP SRAM 监控功能）和 DMA 总线访问记录（HP SRAM 监控功能），其配置流程如下。

1. 配置监测地址范围：配置 `MEM_MONITOR_LOG_MIN_REG` 和 `MEM_MONITOR_LOG_MAX_REG` 确定地址范围。对于 HP CPU，监测地址范围需处于 HP SRAM (0x4080_0000 ~ 0x4085_FFFF) 或外部 RAM (0x4200_0000 ~ 0x43FF_FFFF) 的地址区间内；对于 LP CPU 或 DMA 监测地址，范围需处于 HP SRAM (0x4080_0000 ~ 0x4085_FFFF) 的地址区间内。
2. 配置监测模式 (`MEM_MONITOR_LOG_MODE`)
 - 写监测（监测总线是否发生写操作）
 - 字 (word) 监测（监测总线是否写某个特定 word）

- 半字 (halfword) 监测 (监测总线是否写某个特定 halfword)
- 字节 (byte) 监测 (监测总线是否写某个特定 byte)

3. 配置需要监测的特殊值

- 监测模式为 word 监测时, `MEM_MONITOR_LOG_CHECK_DATA_REG` 即为监测的特定 word。
- 监测模式为 halfword 监测时, `MEM_MONITOR_LOG_CHECK_DATA_REG[15:0]` 即为监测的特定 halfword。
- 监测模式为 byte 监测时, `MEM_MONITOR_LOG_CHECK_DATA_REG[7:0]` 即为监测的特定 byte。
- `MEM_MONITOR_LOG_DATA_MASK_REG` 用于屏蔽 `MEM_MONITOR_LOG_CHECK_DATA_REG` 的相应 byte。当某一 byte 被屏蔽, 表示该 byte 可为任意值, 比如在 word 监测模式下, `MEM_MONITOR_LOG_CHECK_DATA_REG` 配置为 `0x01020304`, `MEM_MONITOR_LOG_DATA_MASK_REG` 配置为 `0x1`, 此时只要总线写 `0x010203XX`, 都会被记录。

4. 配置记录信息的存储地址范围

- `MEM_MONITOR_LOG_MEM_START_REG` 和 `MEM_MONITOR_LOG_MEM_END_REG` 为存储地址范围配置寄存器。记录信息的存储地址范围为 `0x4080_0000 ~ 0x4085_FFFF`。
- 置位 `MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG`, 使 `MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG` 更新为 `MEM_MONITOR_LOG_MEM_START_REG`。
- 配置辅助调试模块对 HP SRAM 的使用权限, 只有开启了辅助调试模块对 HP SRAM 的使用权限才能访问 HP SRAM。关于如何配置, 详情见章节 13 权限控制 (PMS)。

5. 配置记录信息的存储模式, 分为 loop 模式和非 loop 模式

- loop 模式下, 将记录信息循环写进配置地址范围内, 当写到结束地址时回到起始地址开始写, 覆盖之前记录的信息。置位 `MEM_MONITOR_LOG_MEM_LOOP_ENABLE` 使能 loop 模式。
例如, 存储记录信息的地址范围为 `0 ~ 4`, 总线访问过程中有 `1 ~ 10` 共十次写记录信息的操作, 那么第 5 次写操作写到地址 4 后, 第 6 次写操作会回到地址 0 开始写, 第 6 ~ 10 写操作会覆盖之前的第 1 ~ 5 写操作的数据。
- 非 loop 模式下, 当将记录信息写到结束地址后, 会一直停留在结束地址, 并不再写记录信息, 不会覆盖最开始的记录信息, 并且丢弃后面的记录信息。清零 `MEM_MONITOR_LOG_MEM_LOOP_ENABLE` 使能非 loop 模式。
例如, 存储记录信息的地址范围为 `0 ~ 4`, 总线访问过程中有 `1 ~ 10` 共十次写记录信息的操作, 那么第 5 次写操作写到地址 4 后, 第 6 ~ 10 次写操作的地址会一直在地址 4 上, 但不再进行写操作, 因此地址范围 `0 ~ 4` 保留前 5 次写操作信息, 并且丢弃第 6 ~ 10 次写操作信息。

6. 配置总线使能

- 配置 `MEM_MONITOR_LOG_CORE_ENA`、`MEM_MONITOR_LOG_DMA_1_ENA` 和 `MEM_MONITOR_LOG_DMA_0_ENA` 分别使能 HP CPU、LP CPU 和 DMA 总线访问记录, 可同时使能。

辅助调试模块会先将记录信息缓存到一个内部 buffer A 里, 然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时, 连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包, 此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包, 并在 buffer 不满的时候缓存一个 LOST 数据包作为替代。此时仅能获取 LOST 数据包生成时所舍弃的记录数据包的总线类型, 但无法得知在此之前舍弃了多少个记录数据包。

当总线访问记录结束后需要从内存中读取记录数据并进行解码。记录数据里只有四种包格式，HP CPU 数据包、LP CPU 数据包、DMA 数据包和 LOST 数据包，前三种数据包分别对应 HP CPU 数据总线记录信息、LP CPU 数据总线记录信息和 DMA 数据总线记录信息，四种包格式如表 14.5-1、14.5-2、14.5-3 和 14.5-4 所示：

表 14.5-1. HP CPU 包格式

Bit[63:33]	Bit[32]	Bit[31:3]	Bit[2:1]	Bit[0]
pc_offset	anchored(1)	addr_offset	format	anchored(0)

表 14.5-2. LP CPU 包格式

Bit[31:8]	Bit[7:3]	Bit[2:1]	Bit[0]
addr_offset	保留	format	anchored(0)

表 14.5-3. DMA 包格式

Bit[31:8]	Bit[7:3]	Bit[2:1]	Bit[0]
addr_offset	dma_source	format	anchored(0)

表 14.5-4. LOST 包格式

Bit[31:7]	Bit[6:3]	Bit[2:1]	Bit[0]
保留	lost_source	format	anchored(0)

从包格式可以看出，HP CPU 数据包共 64 位，其他数据包均为 32 位。以下为各个域的含义：

- **format** 表示此次数据包的类型：
 - 0：HP CPU 数据包
 - 1：DMA 数据包
 - 2：LP CPU 数据包
 - 3：LOST 数据包
- **pc_offset** 记录了 HP CPU 的 PC 指针的偏移量，实际 PC = pc_offset + 0x0000_0000。
- **addr_offset** 记录了此次写操作的地址偏移，实际地址 = addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}。
- **dma_source** 记录了 DMA 的访问来源，详见表 14.5-5，其中值 16 ~ 31 对应的来源参考 3 通用 DMA 控制器 (GDMA)。
- **lost_source** 记录了 LOST 数据包生成时舍弃了哪些数据包：
 - Bit[4:3]:
 - * 0：未舍弃 HP CPU 数据包
 - * 3：舍弃了 HP CPU 数据包
 - * 其他值：保留
 - Bit[5]:

- * 0: 未舍弃 DMA 数据包
- * 1: 舍弃了 DMA 数据包
- Bit[6]:
 - * 0: 未舍弃 LP CPU 数据包
 - * 1: 舍弃了 LP CPU 数据包
- **anchored** 表示该 32 位数据在数据包中的位置:
 - 0: 该 32 位数据是数据包的低 32 位
 - 1: 该 32 位数据是数据包的高 32 位

表 14.5-5. DMA 访问来源

值	来源
0	HP CPU
1	LP CPU
2	保留
3	保留
4	保留
5	MEM_MONITOR
6	TRACE
7	保留
8	PSRAM_MEM_MONITOR
9 ~ 15	保留
16 ~ 31	详见章节 3 通用 DMA 控制器 (GDMA) > 表 3.4-1 中 0 ~ 15 对应的外设。例如，16 对应的来源为该表中 0 对应的外设，17 对应的来源为该表中 1 对应的外设，以此类推

模块内部 buffer 的数据宽度是 32 位。若同时使能 HP CPU、LP CPU 和 DMA 总线访问记录，并且在某一时刻同时生成了记录数据，则先将 HP CPU 数据包缓存到 buffer 中，再缓存 DMA 数据包，最后缓存 LP CPU 数据包。辅助调试模块会自动将缓存数据从 buffer 取出，并将取出的数据再以 32 位数据宽度存入指定范围内存中。

当启用 loop 模式时，若在配置的存储地址范围内循环了若干次，可能存在残留数据干扰解析，即 HP CPU 数据包的低 32 位数据被覆盖，使得高 32 位数据成为残留数据，因此确定第一个有效数据包的位置时，必须过滤可能存在的残留数据。读取 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 可确定数据包的起始位置，检查数据包 anchored 位的数值，是 0 则保留，是 1 则舍弃。

数据包解析流程如下。

- 读取 MEM_MONITOR_LOG_MEM_FULL_FLAG 确定数据是否溢出配置范围。
 - 如果未溢出，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4；
 - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG 和

MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4;

- 若溢出并且未开启 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG。

- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过置位 MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 清除 MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

14.5.3.2 总线访问记录 2 配置

总线访问记录 2 包括 DMA 总线访问记录（外部 RAM 监控功能），其配置流程如下。

1. 配置监测地址范围：配置 MEM_MONITOR_LOG_MIN_REG 和 MEM_MONITOR_LOG_MAX_REG 确定地址范围。监测地址范围需处于 0x4200_0000 ~ 0x43FF_FFFF 区间内。
2. 配置 MEM_MONITOR_LOG_MODE 选择监测模式
 - 写监测（监测总线是否发生写操作）
 - 字 (word) 监测（监测总线是否写某个特定 word）
 - 半字 (halfword) 监测（监测总线是否写某个特定 halfword）
 - 字节 (byte) 监测（监测总线是否写某个特定 byte）
3. 配置需要监测的特殊值
 - 监测模式为 word 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG 即为监测的特定 word。
 - 监测模式为 halfword 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG[15:0] 即为监测的特定 halfword。
 - 监测模式为 byte 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG[7:0] 即为监测的特定 byte。
 - MEM_MONITOR_LOG_DATA_MASK_REG 用于屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 的相应 byte。当某一 byte 被屏蔽，表示该 byte 可为任意的值，比如 word 监测，MEM_MONITOR_LOG_CHECK_DATA_REG 配置为 0x01020304，MEM_MONITOR_LOG_DATA_MASK_REG 配置为 0x1，因此只要总线写 0x010203XX 都会被记录。
4. 配置记录信息的存储地址范围
 - MEM_MONITOR_LOG_MEM_START_REG 和 MEM_MONITOR_LOG_MEM_END_REG 为存储地址范围配置寄存器。记录信息的存储地址范围为 0x4080_0000 ~ 0x4085_FFFF。
 - 置位 MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG，使 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 更新为 MEM_MONITOR_LOG_MEM_START_REG。
 - 配置辅助调试模块对 HP SRAM 的使用权限，只有开启了辅助调试模块对 HP SRAM 的使用权限才能访问 HP SRAM。关于如何配置，详情见章节 13 权限控制 (PMS)。
5. 配置记录信息的存储模式，分为 loop 模式和非 loop 模式
 - loop 模式下，将记录信息循环写进配置地址范围内，当写到结束地址时回到起始地址开始写，覆盖之前记录的信息。置位 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 启用 loop 模式。

- 非 loop 模式下，当将记录信息写到结束地址后，会一直停留在结束地址，并不再写记录信息，不会覆盖最开始的记录信息，并且丢弃后面的记录信息。清零
`MEM_MONITOR_LOG_MEM_LOOP_ENABLE` 启用非 loop 模式。
- 示例参考小节 14.5.3.1 的步骤 5。

6. 配置 `MEM_MONITOR_LOG_DMA_0_ENA` 使能 DMA_0 通道组总线访问记录。

辅助调试模块会先将 DMA 记录信息缓存到一个内部 buffer B 里，然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时，连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包，此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包，并在 buffer 不满的时候缓存一个 DMA LOST 数据包作为替代，此时仅能知道 DMA LOST 数据包生成时所舍弃的记录数据包的总线类型，但不能知道在此之前舍弃了多少个记录数据包。

当总线访问记录结束后，需要从内存中读取记录数据并进行解码。记录数据里只有两种包格式，DMA_0 数据包和 DMA LOST 数据包，DMA_0 数据包对应 DMA_0 通道组数据总线记录信息，两种包格式如表 14.5-6 和 14.5-7 所示：

表 14.5-6. DMA_0 包格式

Bit[31:7]	Bit[6:2]	Bit[1]	Bit[0]
addr_offset	dma_source	format	anchored(0)

表 14.5-7. DMA LOST 包格式

Bit[31:3]	Bit[2]	Bit[1]	Bit[0]
保留	lost_source	format	anchored(0)

从包格式可以看出，DMA_0 和 DMA LOST 数据包均为 32 位。以下为各个域的含义：

- **format** 表示此次数据包的类型：
 - 0: DMA_0 数据包
 - 1: DMA LOST 数据包
- **addr_offset** 记录了此次写操作的地址偏移，实际地址 = `addr_offset + {MEM_MONITOR_LOG_MIN_REG[31:2], 2'b0}`。
- **dma_source** 记录了 DMA 访问来源，详见表 14.5-5。
- **lost_source** 记录了 DMA LOST 数据包生成时舍弃了哪些数据包：
 - 0: 未舍弃 DMA_0 数据包
 - 1: 舍弃了 DMA_0 数据包
- **anchored** 表示该 32 位数据在数据包中的位置
 - 0: 该 32 位数据是数据包的低 32 位
 - 1: 该 32 位数据是数据包的高 32 位

模块内部 buffer 的数据宽度是 32 位。若使能 DMA_0 通道组总线访问记录，并且在某一时刻生成了记录数据，DMA_0 的数据包会缓存到 buffer 中。辅助调试模块会自动将缓存数据从 buffer 取出，并将取出的数据再以 32 位数据宽度存入指定范围内存。

数据包解析流程如下。

- 读取 `MEM_MONITOR_LOG_MEM_FULL_FLAG` 确定数据是否溢出配置范围。
 - 如果未溢出，则数据读取的地址范围为 `MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4`；
 - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 `MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG` 和 `MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4`；
 - 若溢出并且未开启 loop 模式，则数据读取的地址范围为 `MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG`。
- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过置位 `MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG` 清除 `MEM_MONITOR_LOG_MEM_FULL_FLAG` 标志位。

14.6 寄存器列表

本小节中**总线记录配置寄存器**的地址为相对于存储器访问监控 1 (TCM_MEM_MONITOR) 基地址和存储器访问监控 2 (PSRAM_MEM_MONITOR) 基地址的地址偏移量（相对地址），详见小节 14.6.1，其它寄存器的所有地址均为相对于总线访问监控 (BUS_MONITOR) 基地址的地址偏移量（相对地址），详见小节 14.6.2，具体地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

14.6.1 总线记录配置寄存器列表

名称	描述	地址	访问
总线记录配置寄存器			
MEM_MONITOR_LOG_SETTING_REG	总线访问记录配置寄存器	0x0000	R/W
MEM_MONITOR_LOG_SETTING1_REG	总线访问记录配置寄存器	0x0004	R/W
MEM_MONITOR_LOG_CHECK_DATA_REG	总线访问监测数据配置寄存器	0x0008	R/W
MEM_MONITOR_LOG_DATA_MASK_REG	总线访问监测数据屏蔽配置寄存器	0x000C	R/W
MEM_MONITOR_LOG_MIN_REG	总线访问监测地址下边界配置寄存器	0x0010	R/W
MEM_MONITOR_LOG_MAX_REG	总线访问监测地址上边界配置寄存器	0x0014	R/W
MEM_MONITOR_LOG_MON_ADDR_UPDATE_0_REG	HP CPU 总线访问监测范围更新寄存器	0x0018	WT
MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG	DMA_0 总线访问监测范围更新寄存器	0x001C	WT
MEM_MONITOR_LOG_MEM_START_REG	记录数据写入内存的起始地址	0x0020	R/W
MEM_MONITOR_LOG_MEM_END_REG	记录数据写入内存的结束地址	0x0024	R/W
MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	表示下一次写内存的写地址	0x0028	RO
MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	更新下一次写内存的写地址为记录数据写入内存的起始地址	0x002C	WT
MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	记录溢出状态寄存器	0x0030	varies
时钟控制寄存器			
MEM_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x0034	R/W
版本寄存器			
MEM_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

14.6.2 其他寄存器列表

名称	描述	地址	访问
监测配置寄存器			
BUS_MONITOR_CORE_O_MONTR_ENA_REG	HP CPU 监测使能配置寄存器	0x0000	R/W
BUS_MONITOR_CORE_O_AREA_DRAMO_O_MIN_REG	数据总线区域 0 监测地址下边界配置寄存器	0x0010	R/W
BUS_MONITOR_CORE_O_AREA_DRAMO_O_MAX_REG	数据总线区域 0 监测地址上边界配置寄存器	0x0014	R/W
BUS_MONITOR_CORE_O_AREA_DRAMO_1_MIN_REG	数据总线区域 1 监测地址下边界配置寄存器	0x0018	R/W
BUS_MONITOR_CORE_O_AREA_DRAMO_1_MAX_REG	数据总线区域 1 监测地址上边界配置寄存器	0x001C	R/W
BUS_MONITOR_CORE_O_AREA_PIF_O_MIN_REG	外设总线区域 0 监测地址下边界配置寄存器	0x0020	R/W
BUS_MONITOR_CORE_O_AREA_PIF_O_MAX_REG	外设总线区域 0 监测地址上边界配置寄存器	0x0024	R/W
BUS_MONITOR_CORE_O_AREA_PIF_1_MIN_REG	外设总线区域 1 监测地址下边界配置寄存器	0x0028	R/W
BUS_MONITOR_CORE_O_AREA_PIF_1_MAX_REG	外设总线区域 1 监测地址上边界配置寄存器	0x002C	R/W
BUS_MONITOR_CORE_O_AREA_PC_REG	HP CPU 区域监测 PC 状态寄存器	0x0030	RO
BUS_MONITOR_CORE_O_AREA_SP_REG	HP CPU 区域监测栈指针状态寄存器	0x0034	RO
BUS_MONITOR_CORE_O_SP_MIN_REG	栈指针监测地址下边界配置寄存器	0x0038	R/W
BUS_MONITOR_CORE_O_SP_MAX_REG	栈指针监测地址上边界配置寄存器	0x003C	R/W
BUS_MONITOR_CORE_O_SP_PC_REG	HP CPU 栈指针监测 PC 状态寄存器	0x0040	RO
中断配置寄存器			
BUS_MONITOR_CORE_O_INTR_RAW_REG	HP CPU 监测原始中断状态寄存器	0x0004	RO
BUS_MONITOR_CORE_O_INTR_ENA_REG	HP CPU 监测中断使能配置寄存	0x0008	R/W
BUS_MONITOR_CORE_O_INTR_CLR_REG	HP CPU 监测清除中断配置寄存器	0x000C	WT
PC 记录配置寄存器			
BUS_MONITOR_CORE_O_RCD_EN_REG	PC 记录使能配置寄存器	0x0044	R/W
PC 记录状态寄存器			
BUS_MONITOR_CORE_O_RCD_PDEBUGPC_REG	HP CPU PC 记录寄存器	0x0048	RO
BUS_MONITOR_CORE_O_RCD_PDEBUGSP_REG	HP CPU 栈指针记录寄存器	0x004C	RO
CPU 状态寄存器			
BUS_MONITOR_CORE_O_LASTPC_BEFORE_EXCEPTION_REG	HP CPU 异常前的最后一条指令的 PC	0x0070	RO

名称	描述	地址	访问
BUS_MONITOR_CORE_0_DEBUG_MODE_REG	HP CPU 处于调试模式的标志寄存器	0x0074	RO
时钟控制寄存器			
BUS_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x0108	R/W
版本寄存器			
BUS_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

14.7 寄存器

本小节中**总线记录配置寄存器**的地址为相对于存储器访问监控 1 (TCM_MEM_MONITOR) 基地址和存储器访问监控 2 (PSRAM_MEM_MONITOR) 基地址的地址偏移量（相对地址），详见小节 14.7.1，其它寄存器的所有地址均为相对于总线访问监控 (BUS_MONITOR) 基地址的地址偏移量（相对地址），详见小节 14.7.2，具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

14.7.1 总线记录配置寄存器

Register 14.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)

MEM_MONITOR_LOG_DMA_1_ENA								MEM_MONITOR_LOG_DMA_0_ENA								MEM_MONITOR_LOG_CORE_ENA								(reserved)				MEM_MONITOR_LOG_MEM_LOOP_ENABLE								MEM_MONITOR_LOG_LOG_MODE							
31					24	23					16	15					8	7					5	4	3					0													
0								0								0								0 0 0 1				0								Reset							

MEM_MONITOR_LOG_MODE 配置监测模式。

- 1: 使能写监测
- 2: 使能 word 监测
- 4: 使能 halfword 监测
- 8: 使能 byte 监测
- 其他值: 无效
- (R/W)

MEM_MONITOR_LOG_MEM_LOOP_ENABLE 配置写内存的存储模式。

- 0: 非 loop 模式
- 1: loop 模式
- (R/W)

MEM_MONITOR_LOG_CORE_ENA 配置是否使能 HP CPU 总线访问记录。

- 0: 不使能
- 1: 使能
- 其他值: 无效
- (R/W)

见下页...

Register 14.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)

接上页...

MEM_MONITOR_LOG_DMA_O_ENA 配置是否使能 DMA_O 总线访问记录。

0: 不使能

1: 使能

其他值：无效

(R/W)

MEM_MONITOR_LOG_DMA_1_ENA 配置是否使能 DMA_1 总线访问记录。

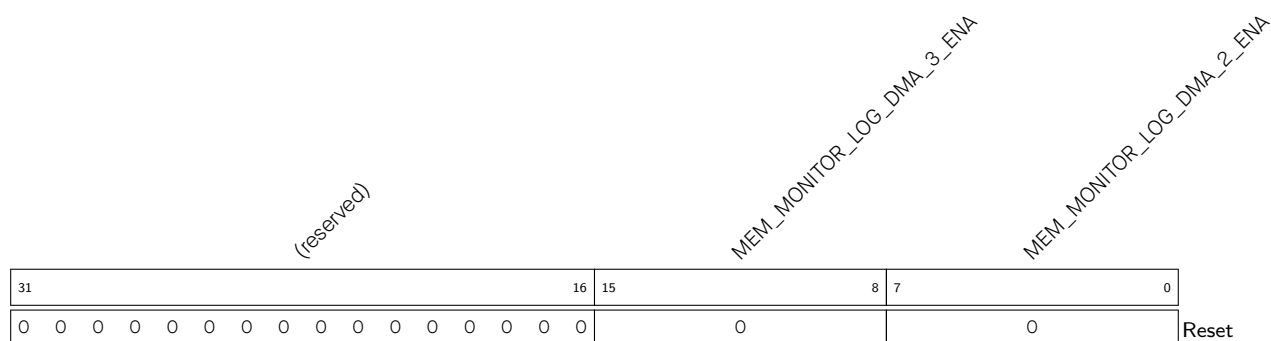
0: 不使能

1: 使能

其他值：无效

(R/W)

Register 14.2. MEM_MONITOR_LOG_SETTING1_REG (0x0004)



MEM_MONITOR_LOG_DMA_2_ENA 配置是否使能 DMA_2 总线访问记录。

0: 不使能

1: 使能

其他值：无效

(R/W)

MEM_MONITOR_LOG_DMA_3_ENA 配置是否使能 DMA_3 总线访问记录。

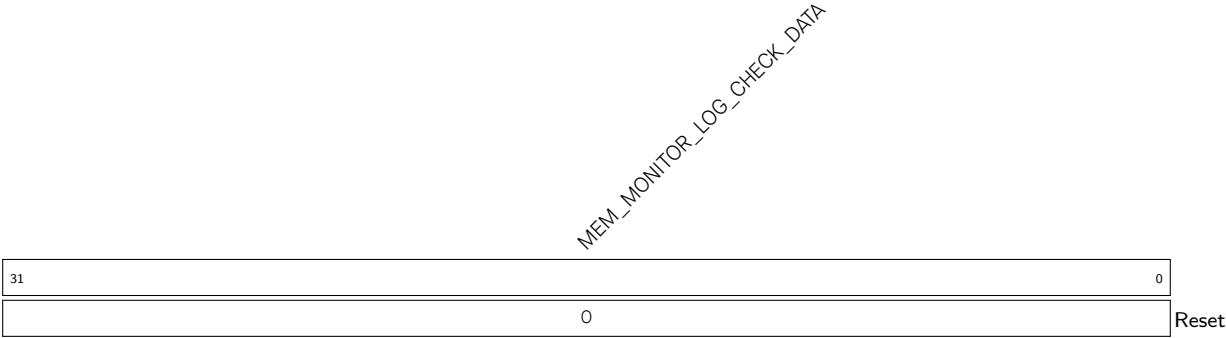
0: 不使能

1: 使能

其他值：无效

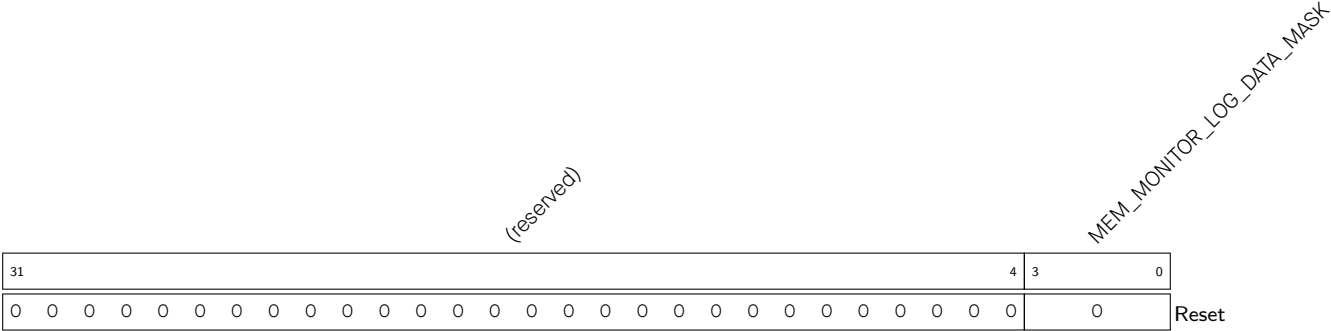
(R/W)

Register 14.3. MEM_MONITOR_LOG_CHECK_DATA_REG (0x0008)



MEM_MONITOR_LOG_CHECK_DATA 配置总线访问监测的特殊值。(R/W)

Register 14.4. MEM_MONITOR_LOG_DATA_MASK_REG (0x000C)



MEM_MONITOR_LOG_DATA_MASK 配置是否屏蔽 [MEM_MONITOR_LOG_CHECK_DATA_REG](#) 的相应字节。可同时屏蔽多个字节。

bit[0]: 配置是否屏蔽 [MEM_MONITOR_LOG_CHECK_DATA_REG](#) 最低位字节。

0: 不屏蔽

1: 屏蔽

bit[1]: 配置是否屏蔽 [MEM_MONITOR_LOG_CHECK_DATA_REG](#) 第二个低位字节。

0: 不屏蔽

1: 屏蔽

bit[2]: 配置是否屏蔽 [MEM_MONITOR_LOG_CHECK_DATA_REG](#) 第二个高位字节。

0: 不屏蔽

1: 屏蔽

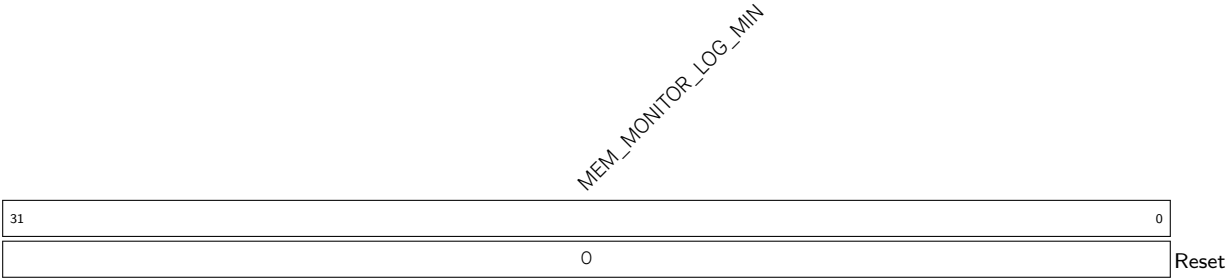
bit[3]: 配置是否屏蔽 [MEM_MONITOR_LOG_CHECK_DATA_REG](#) 最高位字节。

0: 不屏蔽

1: 屏蔽

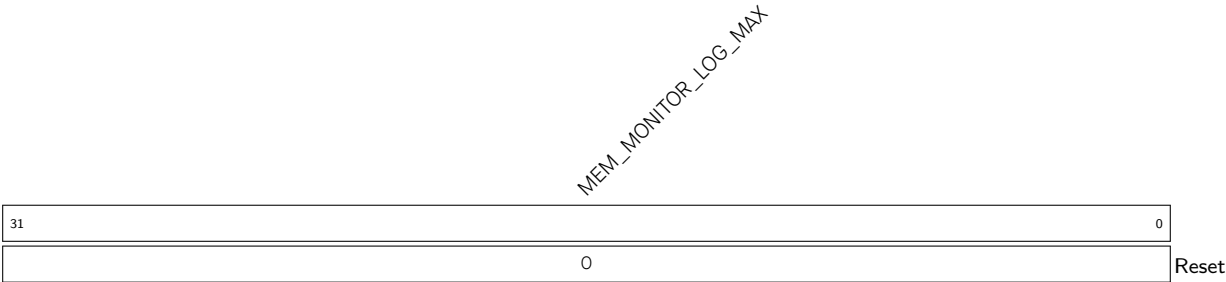
(R/W)

Register 14.5. MEM_MONITOR_LOG_MIN_REG (0x0010)



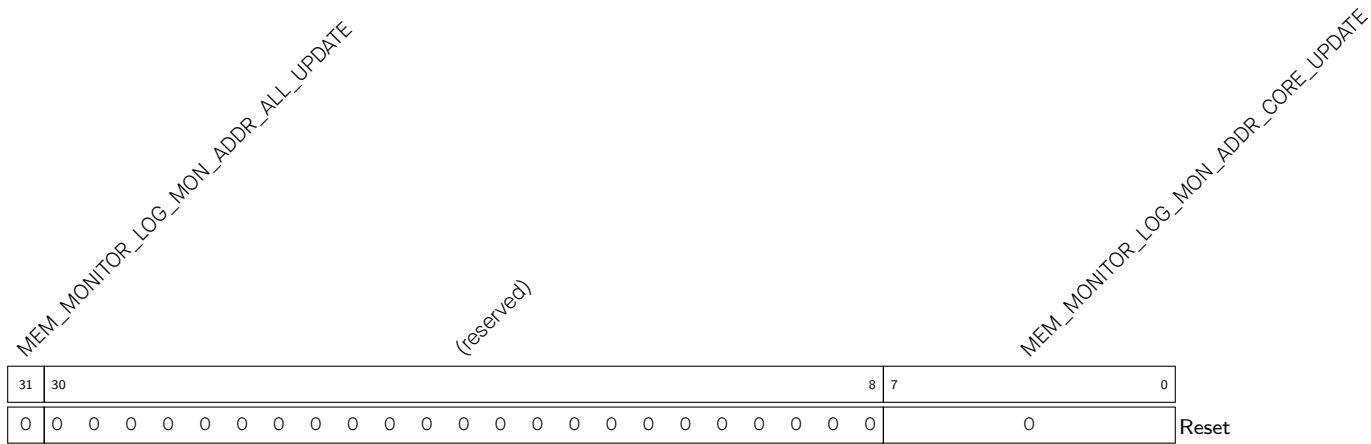
MEM_MONITOR_LOG_MIN 配置监测地址的下边界。(R/W)

Register 14.6. MEM_MONITOR_LOG_MAX_REG (0x0014)



MEM_MONITOR_LOG_MAX 配置监测地址的上边界。(R/W)

Register 14.7. MEM_MONITOR_LOG_MON_ADDR_UPDATE_O_REG (0x0018)



MEM_MONITOR_LOG_MON_ADDR_CORE_UPDATE 配置是否将 [MEM_MONITOR_LOG_MIN_REG](#) 到 [MEM_MONITOR_LOG_MAX_REG](#) 的地址更新为 HP CPU 总线的监测地址空间。

0: 不更新

1: 更新

其他值: 无效

(WT)

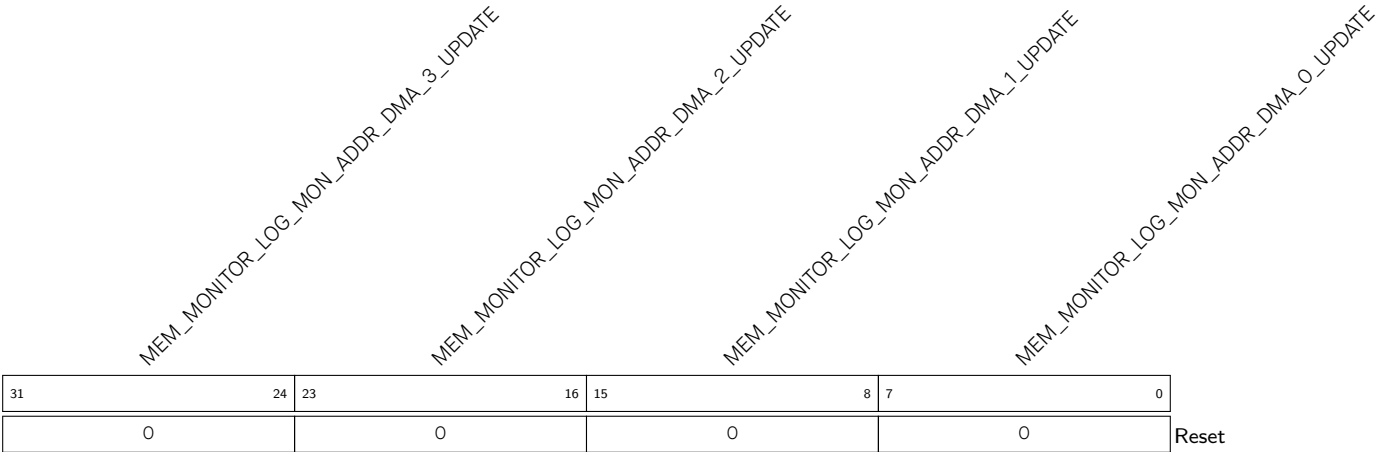
MEM_MONITOR_LOG_MON_ADDR_ALL_UPDATE 配置是否将 [MEM_MONITOR_LOG_MIN_REG](#) 到 [MEM_MONITOR_LOG_MAX_REG](#) 的地址更新为所有主机的监测地址空间。

0: 不更新

1: 更新

(WT)

Register 14.8. MEM_MONITOR_LOG_MON_ADDR_UPDATE_1_REG (0x001C)



MEM_MONITOR_LOG_MON_ADDR_DMA_0_UPDATE [MEM_MONITOR_LOG_MIN_REG](#) 到 [MEM_MONITOR_LOG_MAX_REG](#) 的地址更新为 DMA_0 总线的监测地址空间。

0: 不更新
1: 更新
其他值: 无效
(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_1_UPDATE [MEM_MONITOR_LOG_MIN_REG](#) 到 [MEM_MONITOR_LOG_MAX_REG](#) 的地址更新为 DMA_1 总线的监测地址空间。

0: 不更新
1: 更新
其他值: 无效
(WT)

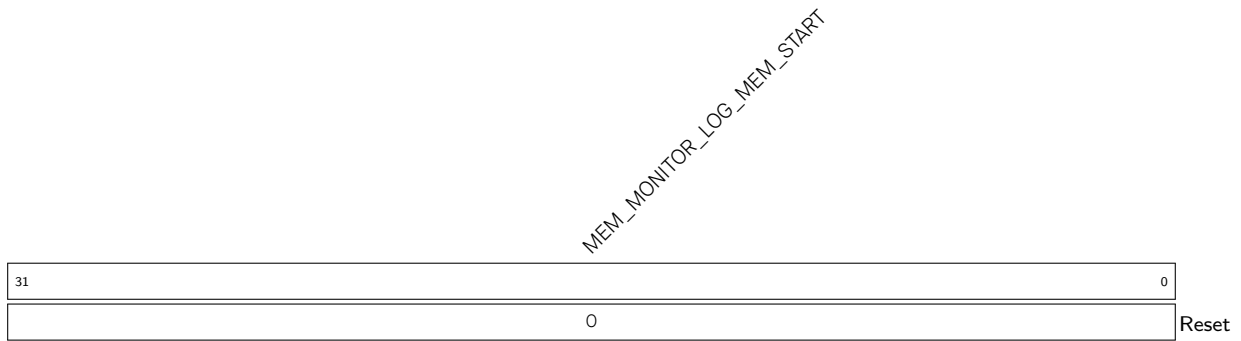
MEM_MONITOR_LOG_MON_ADDR_DMA_2_UPDATE [MEM_MONITOR_LOG_MIN_REG](#) 到 [MEM_MONITOR_LOG_MAX_REG](#) 的地址更新为 DMA_2 总线的监测地址空间。

0: 不更新
1: 更新
其他值: 无效
(WT)

MEM_MONITOR_LOG_MON_ADDR_DMA_3_UPDATE [MEM_MONITOR_LOG_MIN_REG](#) 到 [MEM_MONITOR_LOG_MAX_REG](#) 的地址更新为 DMA_3 总线的监测地址空间。

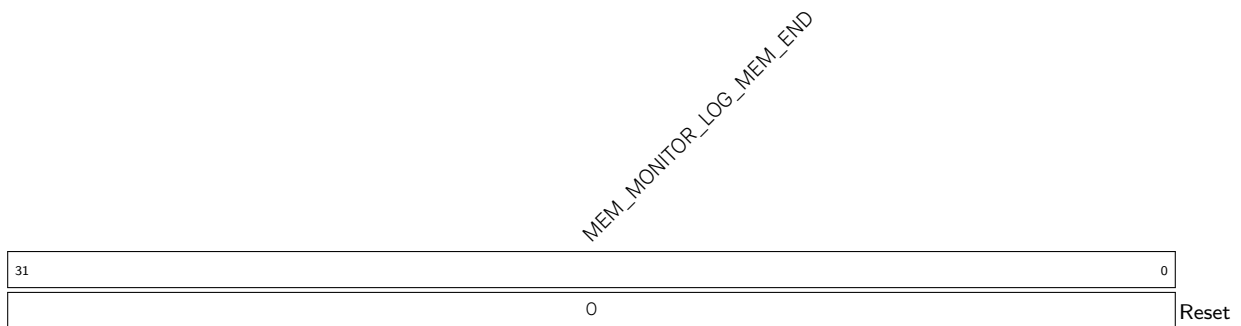
0: 不更新
1: 更新
其他值: 无效
(WT)

Register 14.9. MEM_MONITOR_LOG_MEM_START_REG (0x0020)



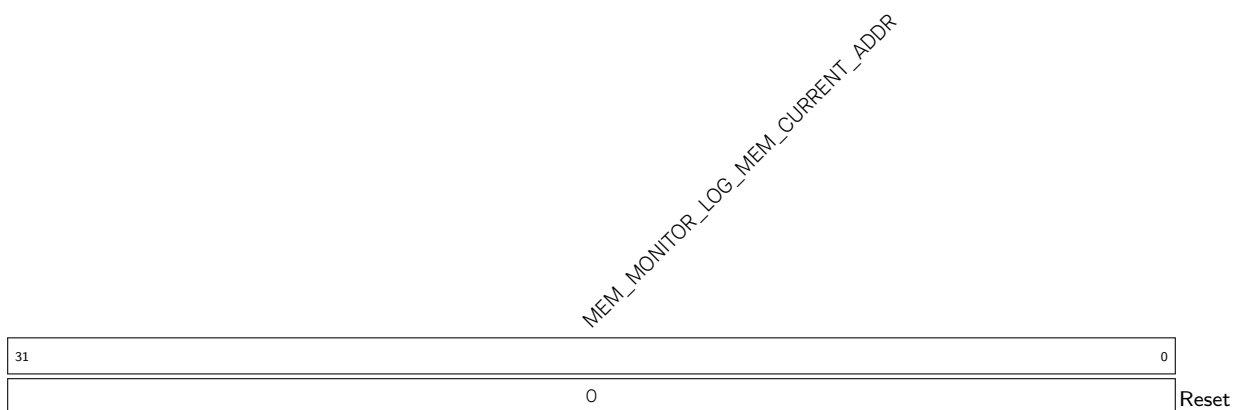
MEM_MONITOR_LOG_MEM_START 配置记录数据写入内存的起始地址。(R/W)

Register 14.10. MEM_MONITOR_LOG_MEM_END_REG (0x0024)



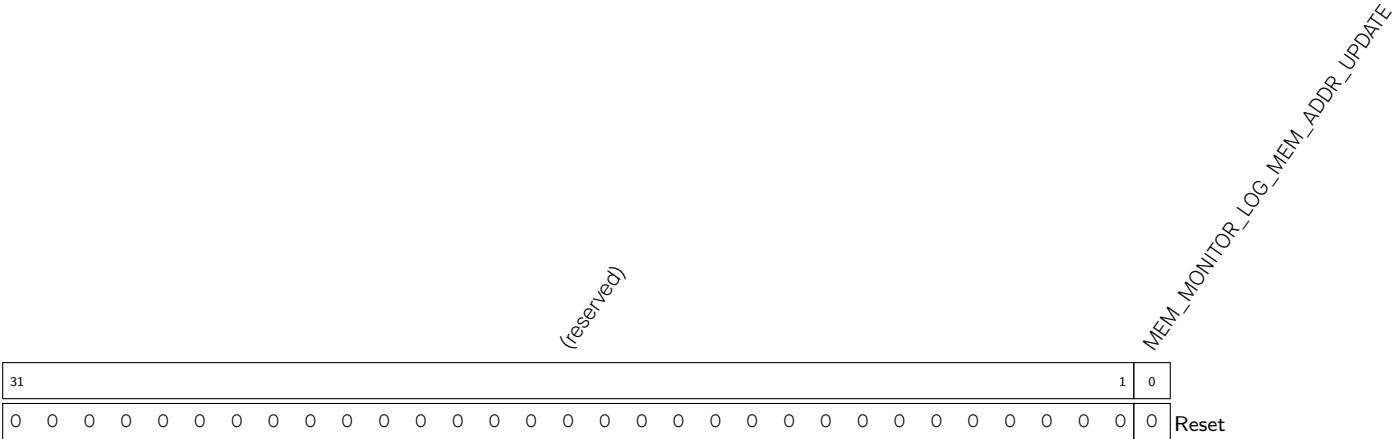
MEM_MONITOR_LOG_MEM_END 配置记录数据写入内存的结束地址。(R/W)

Register 14.11. MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x0028)



MEM_MONITOR_LOG_MEM_CURRENT_ADDR 表示下一次写内存的写地址。(RO)

Register 14.12. MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x002C)



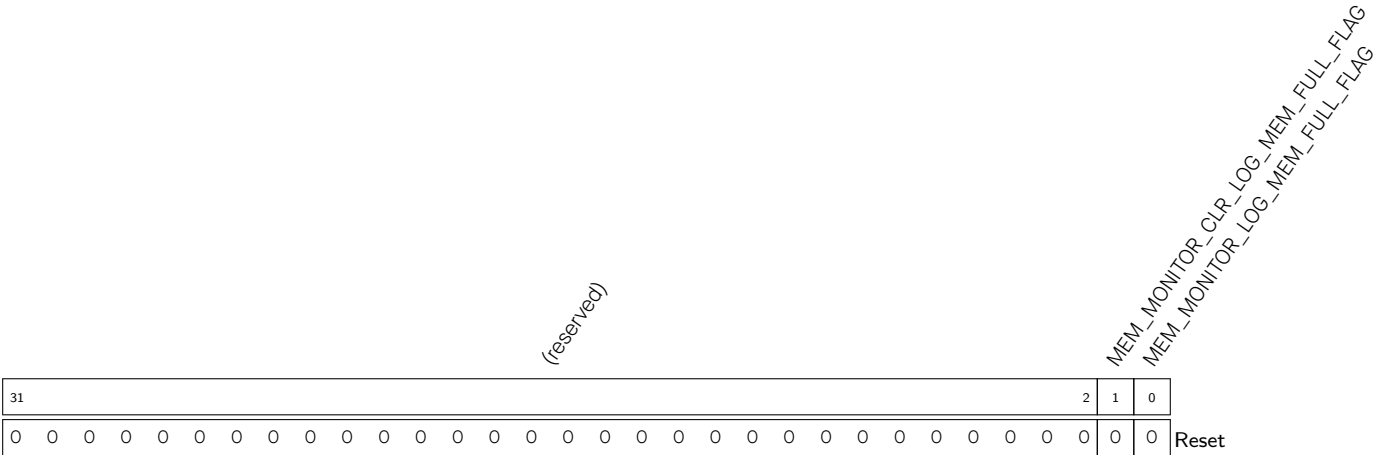
MEM_MONITOR_LOG_MEM_ADDR_UPDATE 配置是否将 **MEM_MONITOR_LOG_MEM_START_REG** 中的值更新为 **MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG** 中的值。

0: 不更新

1: 更新

(WT)

Register 14.13. MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0030)



MEM_MONITOR_LOG_MEM_FULL_FLAG 表示数据是否溢出存储地址范围。

0: 未溢出

1: 溢出

(RO)

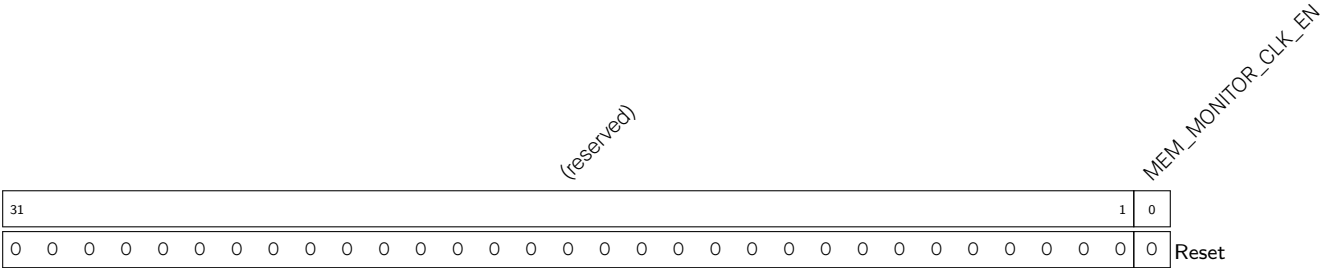
MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 配置是否清除 **MEM_MONITOR_LOG_MEM_FULL_FLAG** 标志位。

0: 不清除（默认）

1: 清除

(WT)

Register 14.14. MEM_MONITOR_CLOCK_GATE_REG (0x0034)



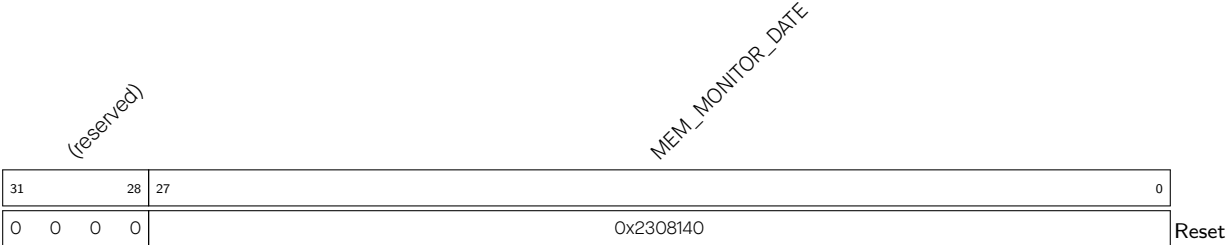
MEM_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0: 不使能

1: 使能

(R/W)

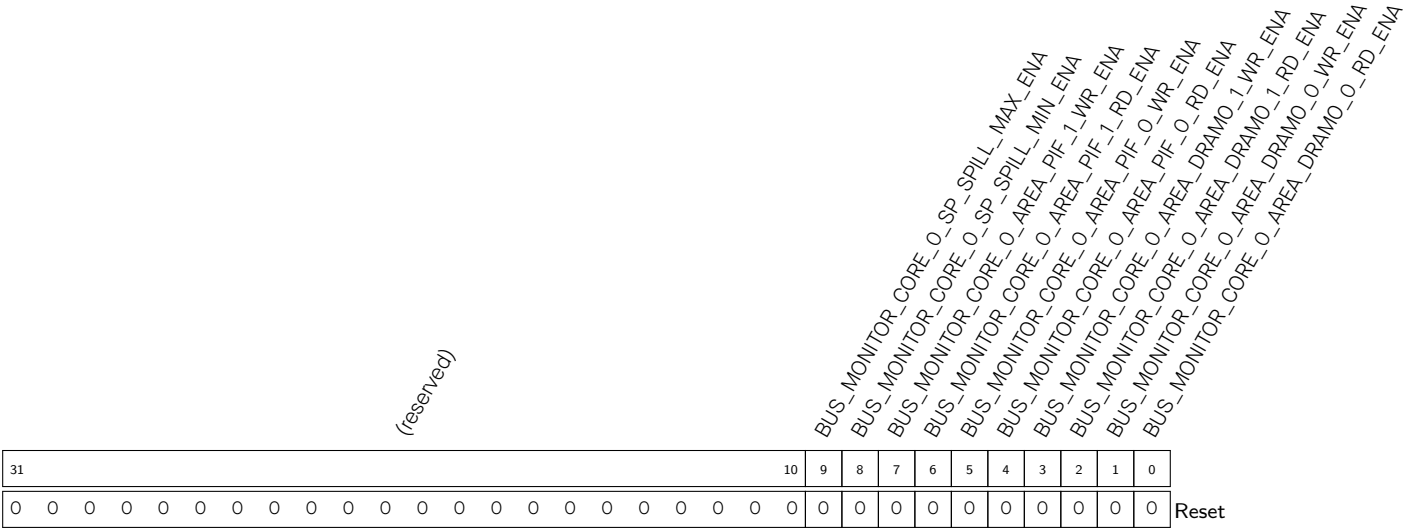
Register 14.15. MEM_MONITOR_DATE_REG (0x03FC)



MEM_MONITOR_DATE 版本控制寄存器。(R/W)

14.7.2 其他寄存器

Register 14.16. BUS_MONITOR_CORE_O_MONTR_ENA_REG (0x0000)



BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_ENA 配置是否监测数据总线在数据总线区域 0 内的读操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_ENA 配置是否监测数据总线在数据总线区域 0 内的写操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_ENA 配置是否监测数据总线在数据总线区域 1 内的读操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_ENA 配置是否监测数据总线在数据总线区域 1 内的写操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_O_AREA_PIF_0_RD_ENA 配置是否监测外设总线在数据总线区域 0 内的读操作。

0: 不监测

1: 监测

(R/W)

见下页...

Register 14.16. BUS_MONITOR_CORE_0_MONTR_ENA_REG (0x0000)

[接上页...](#)

BUS_MONITOR_CORE_0_AREA_PIF_0_WR_ENA 配置是否监测外设总线在数据总线区域 0 内的写操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_1_RD_ENA 配置是否监测外设总线在数据总线区域 1 内的读操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_0_AREA_PIF_1_WR_ENA 配置是否监测外设总线在数据总线区域 1 内的写操作。

0: 不监测

1: 监测

(R/W)

BUS_MONITOR_CORE_0_SP_SPILL_MIN_ENA 配置是否监测栈指针小于栈指针监测区域的下边界。

0: 不监测

1: 监测

(R/W)

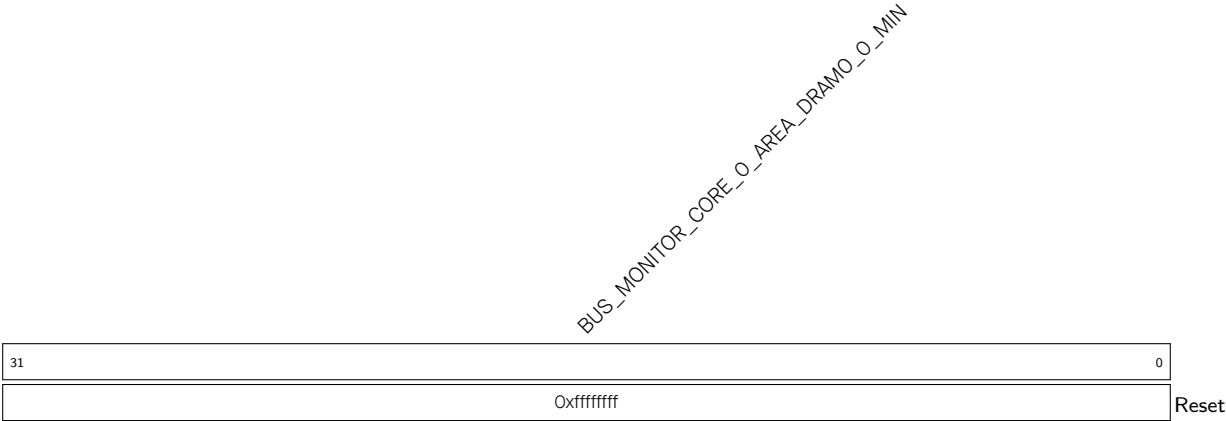
BUS_MONITOR_CORE_0_SP_SPILL_MAX_ENA 配置是否监测栈指针大于栈指针监测区域的上边界。

0: 不监测

1: 监测

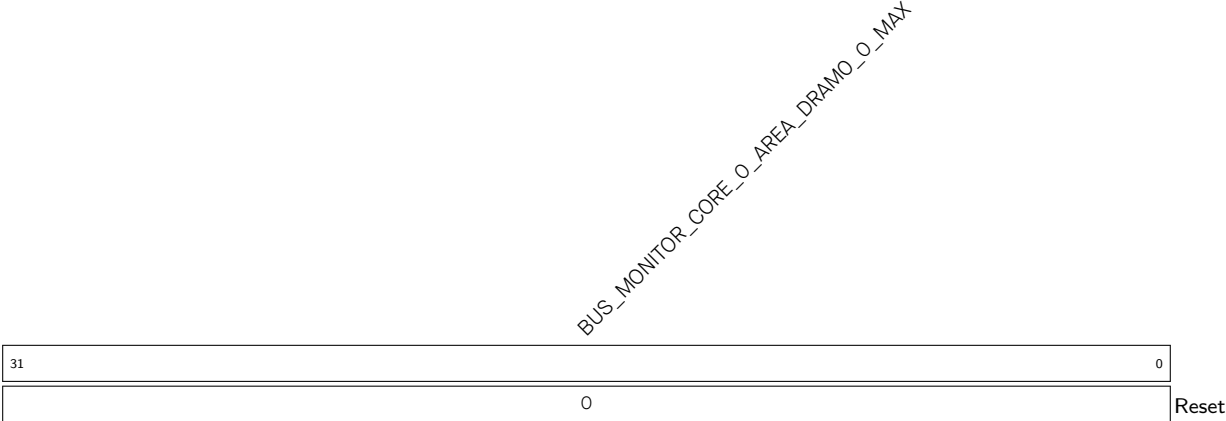
(R/W)

Register 14.17. BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN_REG (0x0010)



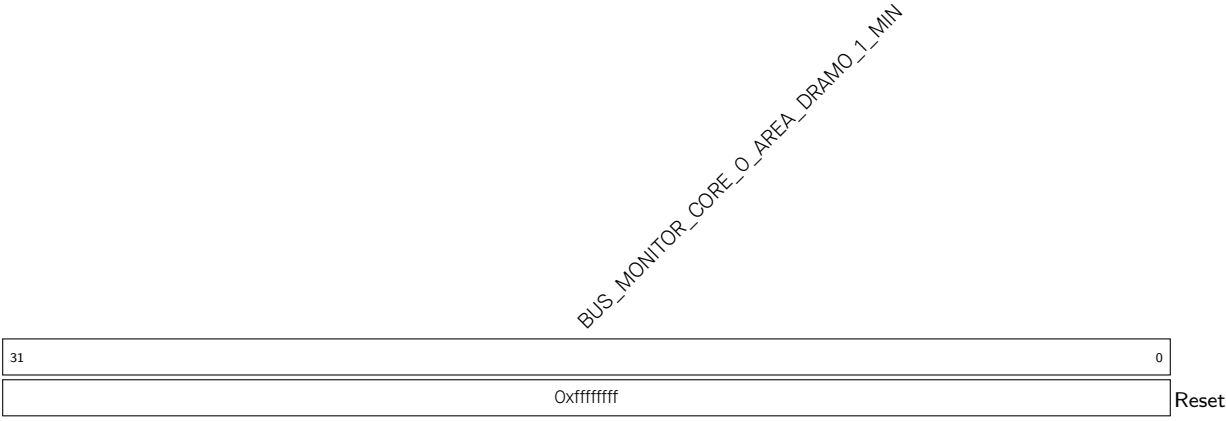
BUS_MONITOR_CORE_0_AREA_DRAMO_0_MIN 配置数据总线区域 0 的下边界。(R/W)

Register 14.18. BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX_REG (0x0014)



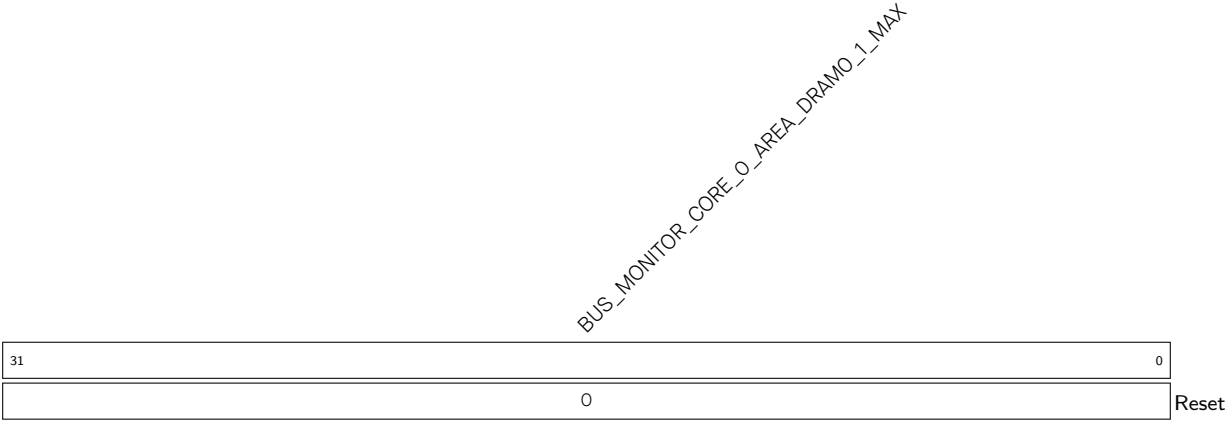
BUS_MONITOR_CORE_0_AREA_DRAMO_0_MAX 配置数据总线区域 0 的上边界。(R/W)

Register 14.19. BUS_MONITOR_CORE_0_AREA_DRAMO_1_MIN_REG (0x0018)



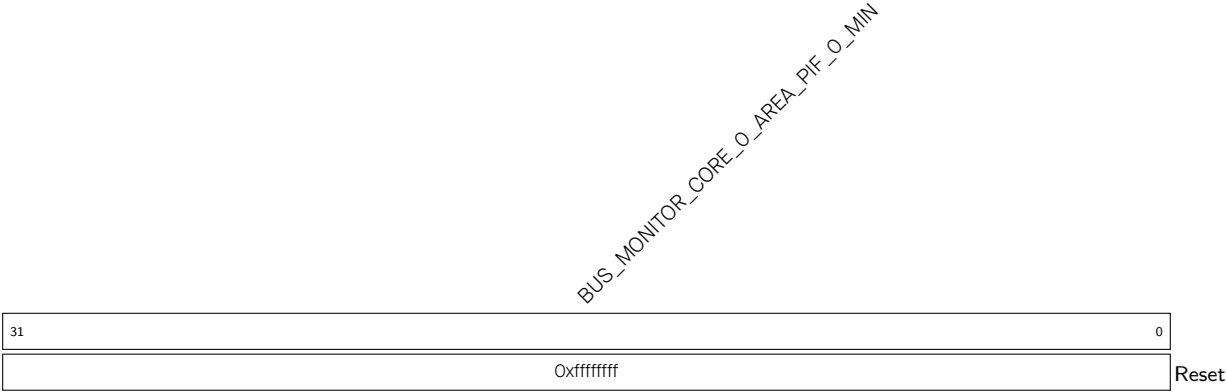
BUS_MONITOR_CORE_0_AREA_DRAMO_1_MIN 配置数据总线区域 1 的下边界。(R/W)

Register 14.20. BUS_MONITOR_CORE_O_AREA_DRAM0_1_MAX_REG (0x001C)



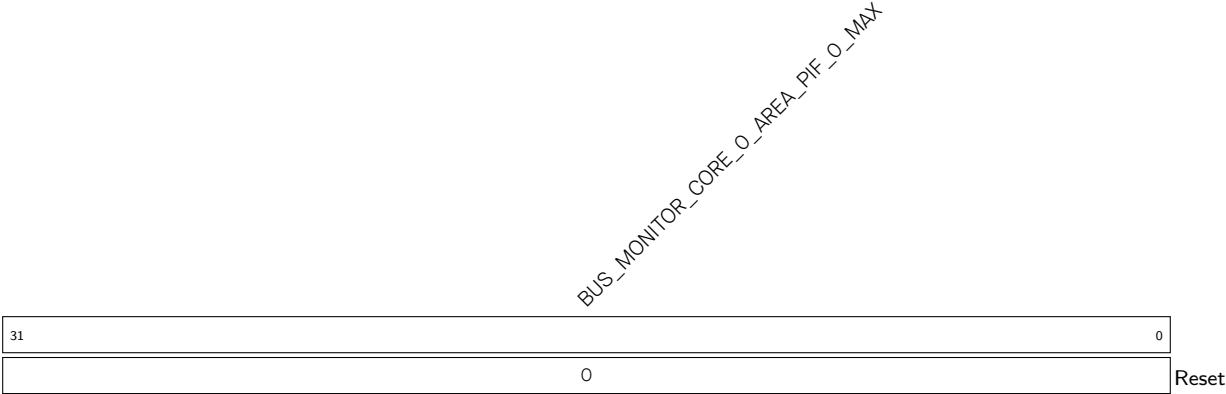
`BUS_MONITOR_CORE_O_AREA_DRAM0_1_MAX` 配置数据总线区域 1 的上边界。(R/W)

Register 14.21. BUS_MONITOR_CORE_O_AREA_PIF_O_MIN_REG (0x0020)



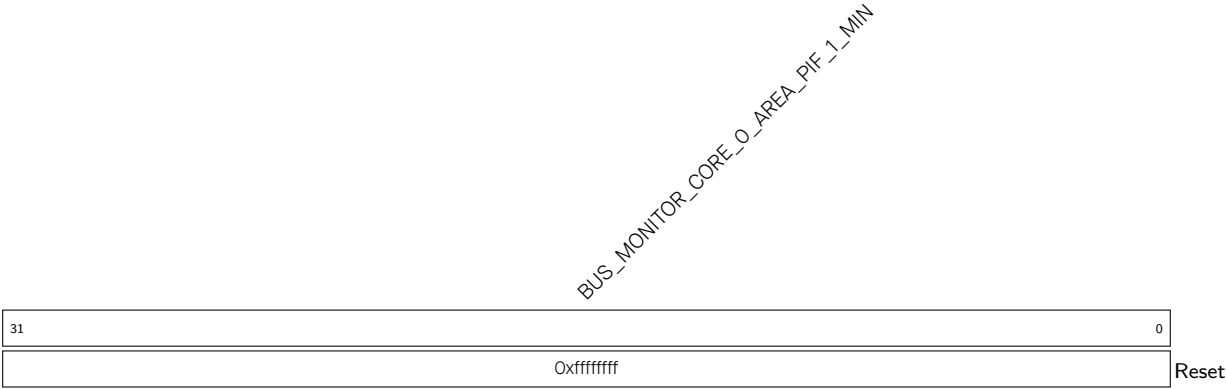
`BUS_MONITOR_CORE_O_AREA_PIF_O_MIN` 配置外设总线区域 0 的下边界。(R/W)

Register 14.22. BUS_MONITOR_CORE_O_AREA_PIF_O_MAX_REG (0x0024)



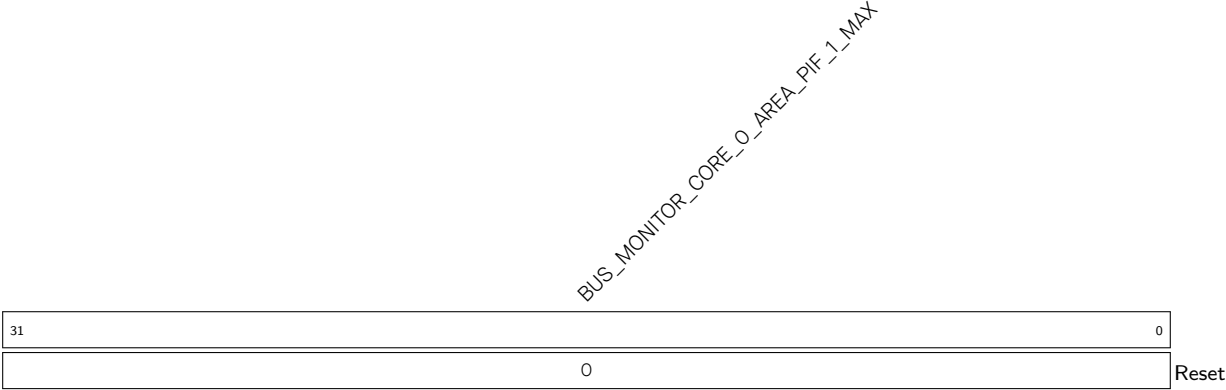
`BUS_MONITOR_CORE_O_AREA_PIF_O_MAX` 配置外设总线区域 0 的上边界。(R/W)

Register 14.23. BUS_MONITOR_CORE_0_AREA_PIF_1_MIN_REG (0x0028)



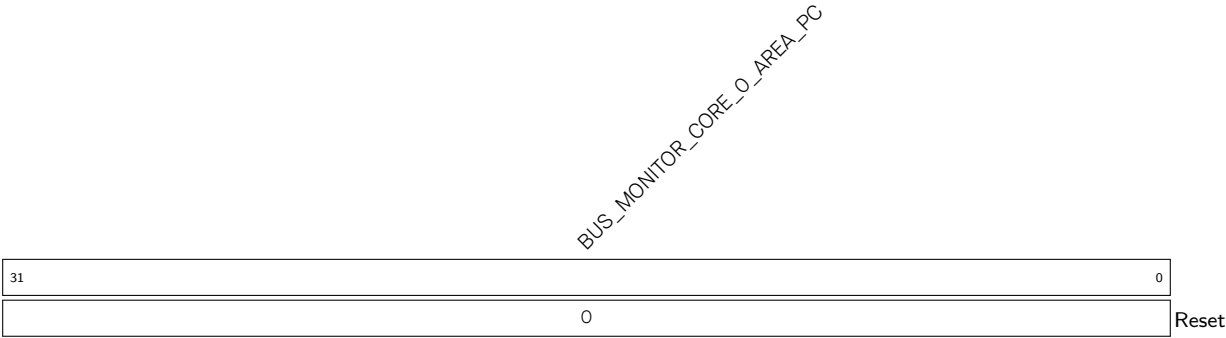
BUS_MONITOR_CORE_0_AREA_PIF_1_MIN 配置外设总线区域 1 的下边界。(R/W)

Register 14.24. BUS_MONITOR_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



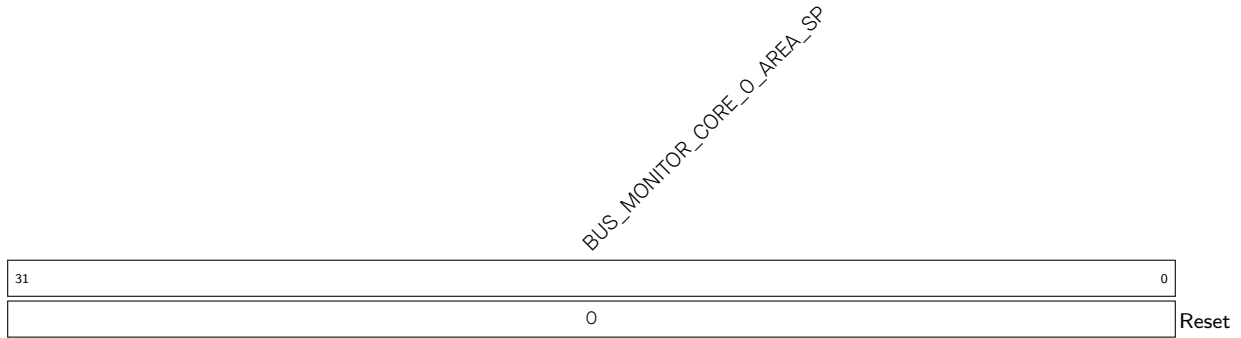
BUS_MONITOR_CORE_0_AREA_PIF_1_MAX 配置外设总线区域 1 的上边界。(R/W)

Register 14.25. BUS_MONITOR_CORE_0_AREA_PC_REG (0x0030)



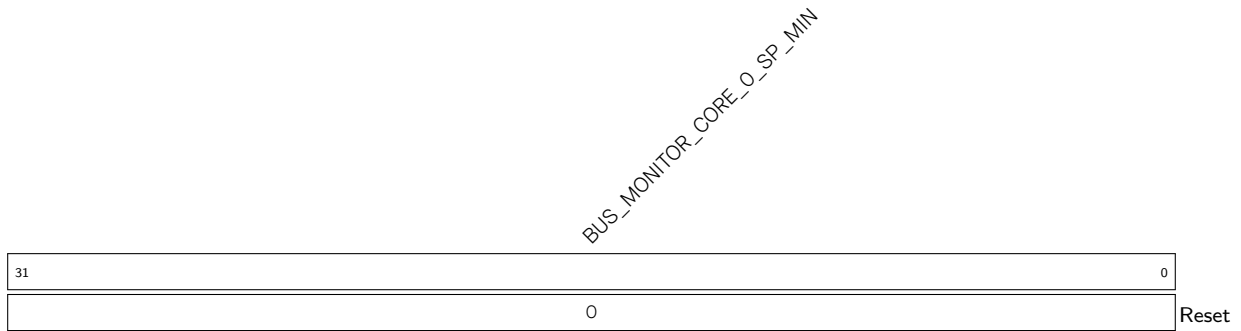
BUS_MONITOR_CORE_0_AREA_PC 表示区域监测下触发中断时刻的 PC 值。(RO)

Register 14.26. BUS_MONITOR_CORE_0_AREA_SP_REG (0x0034)



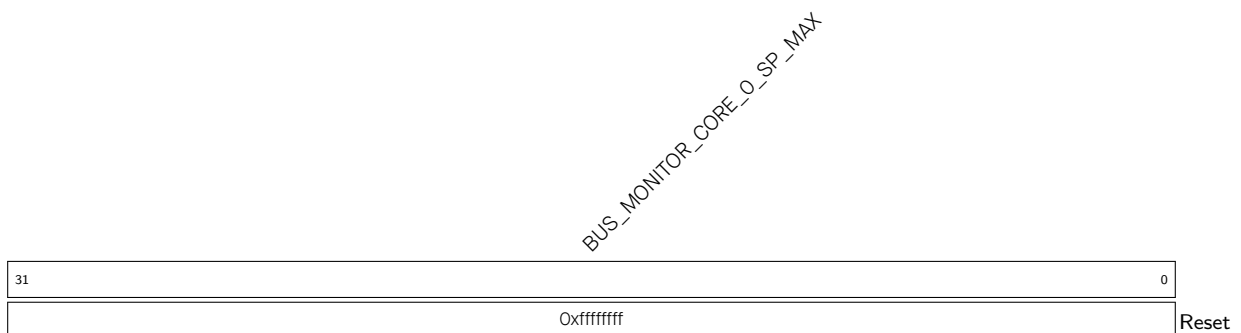
BUS_MONITOR_CORE_0_AREA_SP 表示区域监测下触发中断时刻的栈指针。(RO)

Register 14.27. BUS_MONITOR_CORE_0_SP_MIN_REG (0x0038)



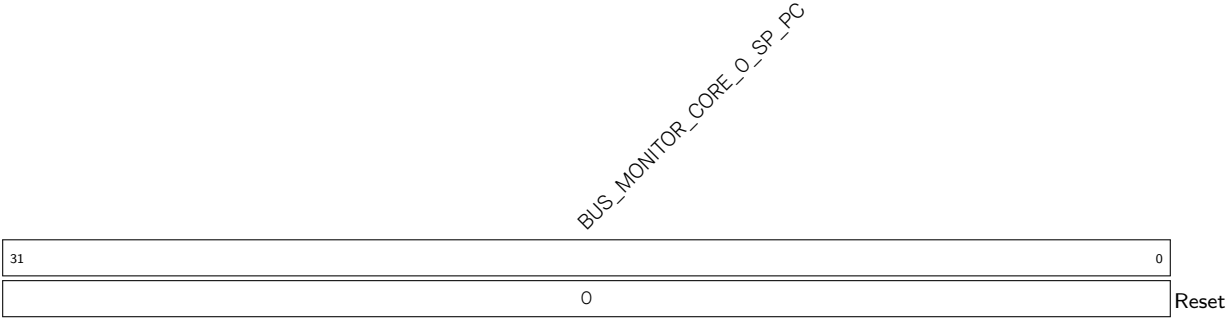
BUS_MONITOR_CORE_0_SP_MIN 配置栈指针监测区域的下边界。(R/W)

Register 14.28. BUS_MONITOR_CORE_0_SP_MAX_REG (0x003C)



BUS_MONITOR_CORE_0_SP_MAX 配置栈指针监测区域的上边界。(R/W)

Register 14.29. BUS_MONITOR_CORE_0_SP_PC_REG (0x0040)



BUS_MONITOR_CORE_0_SP_PC 表示栈指针监测的 PC 值。(RO)

Register 14.30. BUS_MONITOR_CORE_O_INTR_RAW_REG (0x0004)

(reserved)																						BUS_MONITOR_CORE_0_SP_SPILL_MAX_RAW BUS_MONITOR_CORE_0_SP_SPILL_MIN_RAW BUS_MONITOR_CORE_0_AREA_PIF_0_RD_RAW BUS_MONITOR_CORE_0_AREA_PIF_0_WR_RAW BUS_MONITOR_CORE_0_AREA_PIF_1_RD_RAW BUS_MONITOR_CORE_0_AREA_PIF_1_WR_RAW BUS_MONITOR_CORE_0_AREA_DRAMO_0_RD_RAW BUS_MONITOR_CORE_0_AREA_DRAMO_0_WR_RAW BUS_MONITOR_CORE_0_AREA_DRAMO_1_RD_RAW BUS_MONITOR_CORE_0_AREA_DRAMO_1_WR_RAW																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
31																						10	9	8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_RAW BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_RAW BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_RAW BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_RAW BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_PIF_0_RD_RAW BUS_MONITOR_CORE_O_AREA_PIF_0_RD_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_PIF_0_WR_RAW BUS_MONITOR_CORE_O_AREA_PIF_0_WR_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_PIF_1_RD_RAW BUS_MONITOR_CORE_O_AREA_PIF_1_RD_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_AREA_PIF_1_WR_RAW BUS_MONITOR_CORE_O_AREA_PIF_1_WR_INT
的原始中断状态。(RO)

BUS_MONITOR_CORE_O_SP_SPILL_MIN_RAW BUS_MONITOR_CORE_O_SP_SPILL_MIN_INT 的
原始中断状态。(RO)

BUS_MONITOR_CORE_O_SP_SPILL_MAX_RAW BUS_MONITOR_CORE_O_SP_SPILL_MAX_INT
的原始中断状态。(RO)

Register 14.31. BUS_MONITOR_CORE_O_INTR_ENA_REG (0x0008)

(reserved)																						BUS_MONITOR_CORE_0_SP_SPILL_MAX_INTR_ENA BUS_MONITOR_CORE_0_SP_SPILL_MIN_INTR_ENA BUS_MONITOR_CORE_0_AREA_PIF_1_WR_INTR_ENA BUS_MONITOR_CORE_0_AREA_PIF_1_RD_INTR_ENA BUS_MONITOR_CORE_0_AREA_PIF_0_WR_INTR_ENA BUS_MONITOR_CORE_0_AREA_PIF_0_RD_INTR_ENA BUS_MONITOR_CORE_0_AREA_DRAMO_1_WR_INTR_ENA BUS_MONITOR_CORE_0_AREA_DRAMO_1_RD_INTR_ENA BUS_MONITOR_CORE_0_AREA_DRAMO_0_WR_INTR_ENA BUS_MONITOR_CORE_0_AREA_DRAMO_0_RD_INTR_ENA									
31																				10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										

- BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_INTR_ENA 写

BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_INT。(R/W)

1

使 能
- BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_INTR_ENA 写

BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_INT。(R/W)

1

使 能
- BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_INTR_ENA 写

BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_INT。(R/W)

1

使 能
- BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_INTR_ENA 写

BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_INT。(R/W)

1

使 能
- BUS_MONITOR_CORE_O_AREA_PIF_0_RD_INTR_ENA 写1使能

BUS_MONITOR_CORE_O_AREA_PIF_0_RD_INT。(R/W)
- BUS_MONITOR_CORE_O_AREA_PIF_0_WR_INTR_ENA 写1使能

BUS_MONITOR_CORE_O_AREA_PIF_0_WR_INT。(R/W)
- BUS_MONITOR_CORE_O_AREA_PIF_1_RD_INTR_ENA 写1使能

BUS_MONITOR_CORE_O_AREA_PIF_1_RD_INT。(R/W)
- BUS_MONITOR_CORE_O_AREA_PIF_1_WR_INTR_ENA 写1使能

BUS_MONITOR_CORE_O_AREA_PIF_1_WR_INT。(R/W)
- BUS_MONITOR_CORE_O_SP_SPILL_MIN_INTR_ENA 写1使能

BUS_MONITOR_CORE_O_SP_SPILL_MIN_INT。(R/W)
- BUS_MONITOR_CORE_O_SP_SPILL_MAX_INTR_ENA 写1使能

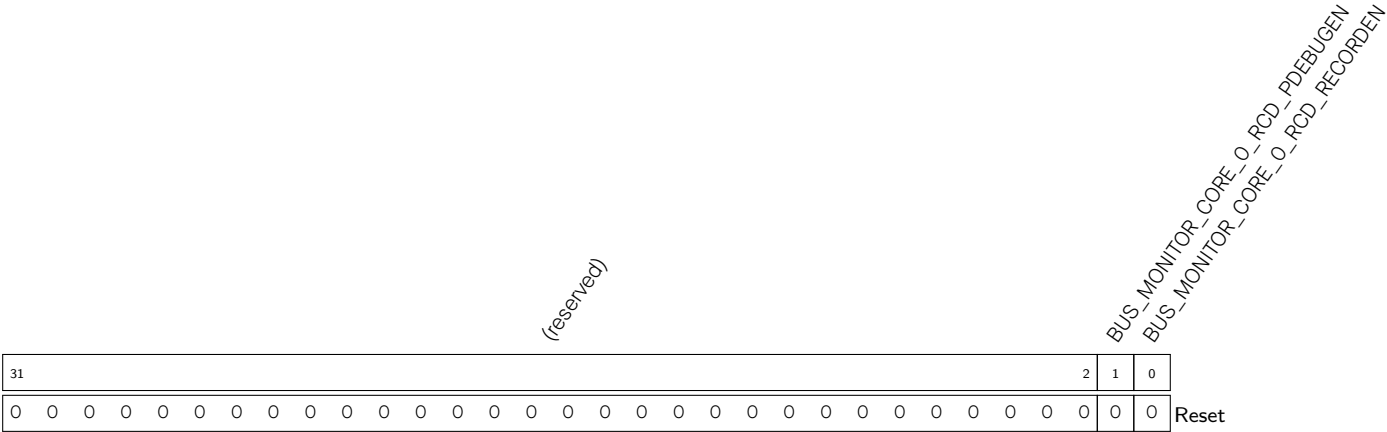
BUS_MONITOR_CORE_O_SP_SPILL_MAX_INT。(R/W)

Register 14.32. BUS_MONITOR_CORE_O_INTR_CLR_REG (0x000C)

(reserved)																						BUS_MONITOR_CORE_O_SP_SPILL_MAX_CLR BUS_MONITOR_CORE_O_SP_SPILL_MIN_CLR BUS_MONITOR_CORE_O_AREA_PIF_1_WR_CLR BUS_MONITOR_CORE_O_AREA_PIF_1_RD_CLR BUS_MONITOR_CORE_O_AREA_PIF_0_WR_CLR BUS_MONITOR_CORE_O_AREA_PIF_0_RD_CLR BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_CLR BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_CLR BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_CLR BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_CLR									
31																				10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										

- BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_DRAMO_0_RD_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_DRAMO_0_WR_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_DRAMO_1_RD_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_DRAMO_1_WR_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_PIF_0_RD_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_PIF_0_RD_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_PIF_0_WR_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_PIF_0_WR_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_PIF_1_RD_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_PIF_1_RD_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_AREA_PIF_1_WR_CLR 写1清除 [BUS_MONITOR_CORE_O_AREA_PIF_1_WR_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_SP_SPILL_MIN_CLR 写1清除 [BUS_MONITOR_CORE_O_SP_SPILL_MIN_INT](#)。
(WT)
- BUS_MONITOR_CORE_O_SP_SPILL_MAX_CLR 写1清除 [BUS_MONITOR_CORE_O_SP_SPILL_MAX_INT](#)。
(WT)

Register 14.33. BUS_MONITOR_CORE_O_RCD_EN_REG (0x0044)



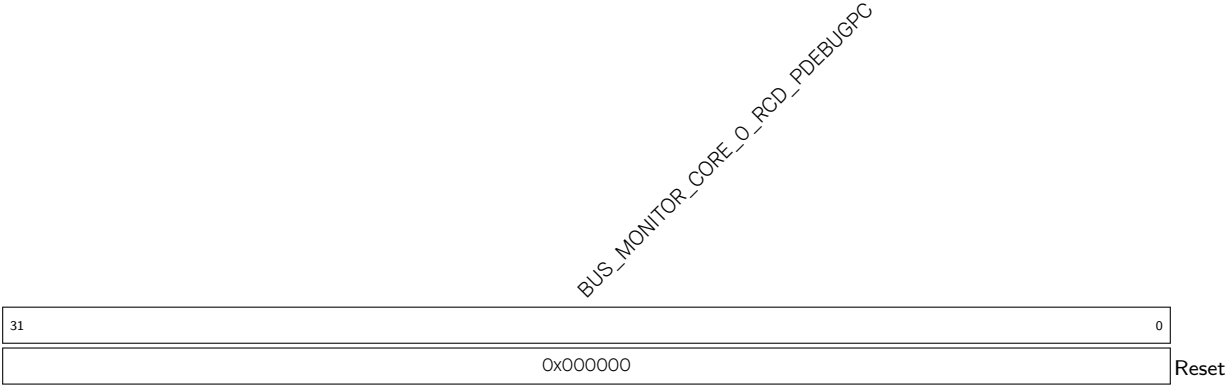
BUS_MONITOR_CORE_O_RCD_RECORDEN 配置是否使能 PC 记录。

- 0: 不使能
- 1: [BUS_MONITOR_CORE_O_RCD_PDEBUGPC_REG](#) 开始实时记录 PC (R/W)

BUS_MONITOR_CORE_O_RCD_PDEBUGEN 配置是否使能 HP CPU 调试。

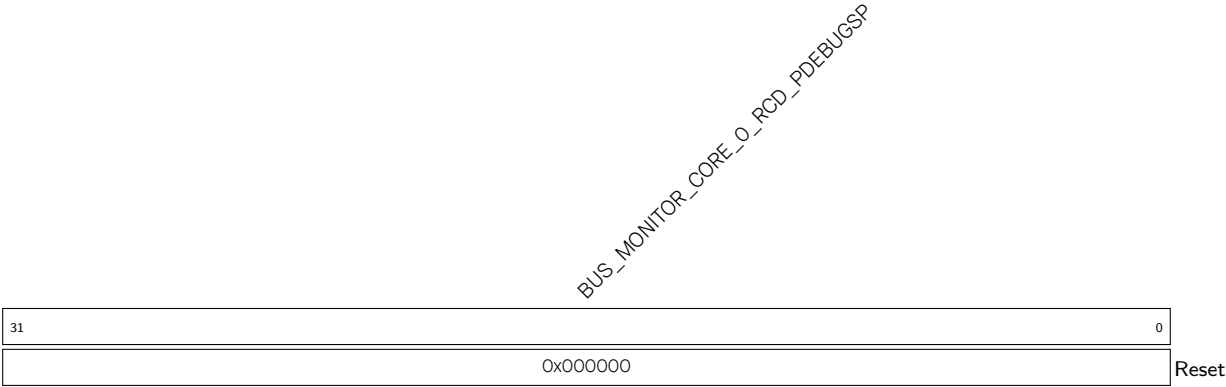
- 0: 不使能
- 1: HP CPU 输出 PC (R/W)

Register 14.34. BUS_MONITOR_CORE_O_RCD_PDEBUGPC_REG (0x0048)



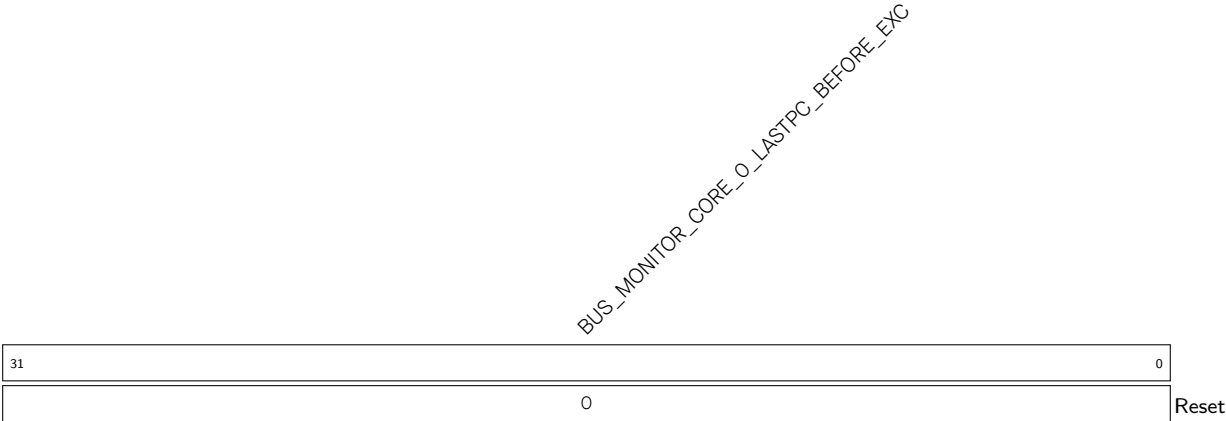
BUS_MONITOR_CORE_O_RCD_PDEBUGPC 表示 HP CPU 复位时刻的 PC 值。(RO)

Register 14.35. BUS_MONITOR_CORE_0_RCD_PDEBUGSP_REG (0x004C)



BUS_MONITOR_CORE_0_RCD_PDEBUGSP 表示栈指针。(RO)

Register 14.36. BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



BUS_MONITOR_CORE_0_LASTPC_BEFORE_EXC 表示 HP CPU 异常前的最后一条指令的 PC。(RO)

Register 14.37. BUS_MONITOR_CORE_0_DEBUG_MODE_REG (0x0074)

(reserved)																															BUS_MONITOR_CORE_0_DEBUG_MODULE_ACTIVE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
																															BUS_MONITOR_CORE_0_DEBUG_MODE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31																													2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

BUS_MONITOR_CORE_0_DEBUG_MODE 表示 RISC-V CPU (HP CPU) 是否处于调试模式。

1: 处于调试模式
0: 未处于调试模式
(RO)

BUS_MONITOR_CORE_0_DEBUG_MODULE_ACTIVE 表示 RISC-V CPU (HP CPU) 调试模块的工作状态。

1: 处于工作状态
其它: 未处于工作状态
(RO)

Register 14.38. BUS_MONITOR_CLOCK_GATE_REG (0x0108)

(reserved)																																BUS_MONITOR		
																																1	0	
31																															1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

BUS_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0: 不使能
1: 使能
(R/W)

Register 14.39. BUS_MONITOR_DATE_REG (0x03FC)

(reserved)				BUS_MONITOR_DATE																										
31	28	27	0																											
0	0	0	0	0x2109130																										Reset

BUS_MONITOR_DATE 版本控制寄存器。(R/W)

第 15 章

电源检测器

15.1 概述

ESP32-C5 芯片上的电源检测器能够监测片上时钟和数字电路相关的电源管脚上的电压。它包含一个欠压检测器和四个电压毛刺保护器。欠压检测器防止欠压复位，电压毛刺保护器防止电压毛刺攻击。电源检测器属于 always-on（常开）电源区域，可以在任何时刻监控电压。

15.2 主要特性

- 欠压检测器支持两种欠压检测模式（模式 0 和模式 1）
 - 模式 0 支持如下动作：
 - * 发出中断
 - * 关闭射频电路
 - * 请求 flash 进入 suspend 状态
 - * 发出系统复位
 - 模式 1 支持如下动作：
 - * 发出系统复位直到电压恢复正常
- 电压毛刺保护器能够检测约 50 ns 以上的电压毛刺并触发系统复位。

15.3 功能描述

15.3.1 架构图

图 15.3-1 为电源检测器的架构图。由图可知，电源检测器包含一个欠压检测器 (Brown-out Detector) 和四个电压毛刺保护器 (Voltage Glitch Detector)。

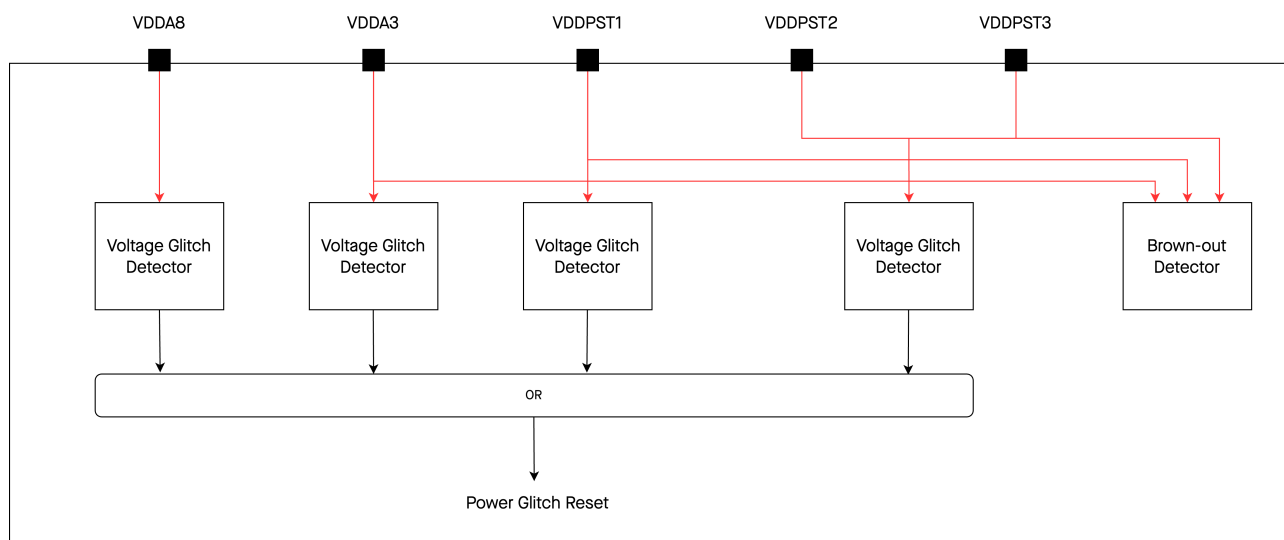


图 15.3-1. 电源检测器架构图

15.3.2 欠压检测器

ESP32-C5 的欠压检测器可以周期性轮流检查管脚 VDDA3、VDDPST1、VDDPST2、VDDPST3 的电压，周期约 $280\ \mu\text{s}$ 。一旦被检测电压下落至预设阈值（默认为 $2.7\ \text{V}$ ）以下时，欠压检测器发出触发信号，并进行相应处理，关闭功耗较高的射频电路，为数字模块争取更多时间，用以保存、转移重要数据。欠压检测器的功耗非常低，在芯片开启时将永远保持开启。

`LP_ANA_BOD_MODE0_INT_RAW` 用于指示欠压检测器的输出结果，默认为 0，可在检测管脚电压下降至阈值以下时置位。

欠压检测器检测到欠压信号后，有两种处理模式（默认为模式 1）：

- 模式 0：
 - 此模式需通过 `bod_mode0_en` (`LP_ANA_BOD_MODE0_INTR_ENA`) 使能。
 - 当欠压计数器达到 Int Comparer (`LP_ANA_BOD_MODE0_INTR_WAIT`) 中设定的阈值后触发中断，如果使能了 `LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA`，将会使 flash 进入 SUSPEND 状态；如果使能了 `LP_ANA_BOD_MODE0_PD_RF_ENA`，将会关闭射频模块。
 - 当欠压计数器达到 Rst Comparer (`LP_ANA_BOD_MODE0_RESET_WAIT`) 设定的阈值后根据 `bod_mode0_rst_sel` (`LP_ANA_BOD_MODE0_RESET_SEL`) 的配置选择复位方式。复位需通过 `bod_mode0_rst_en` (`LP_ANA_BOD_MODE0_RESET_ENA`) 使能。
- 模式 1：直接触发系统复位。

详情如下图所示。

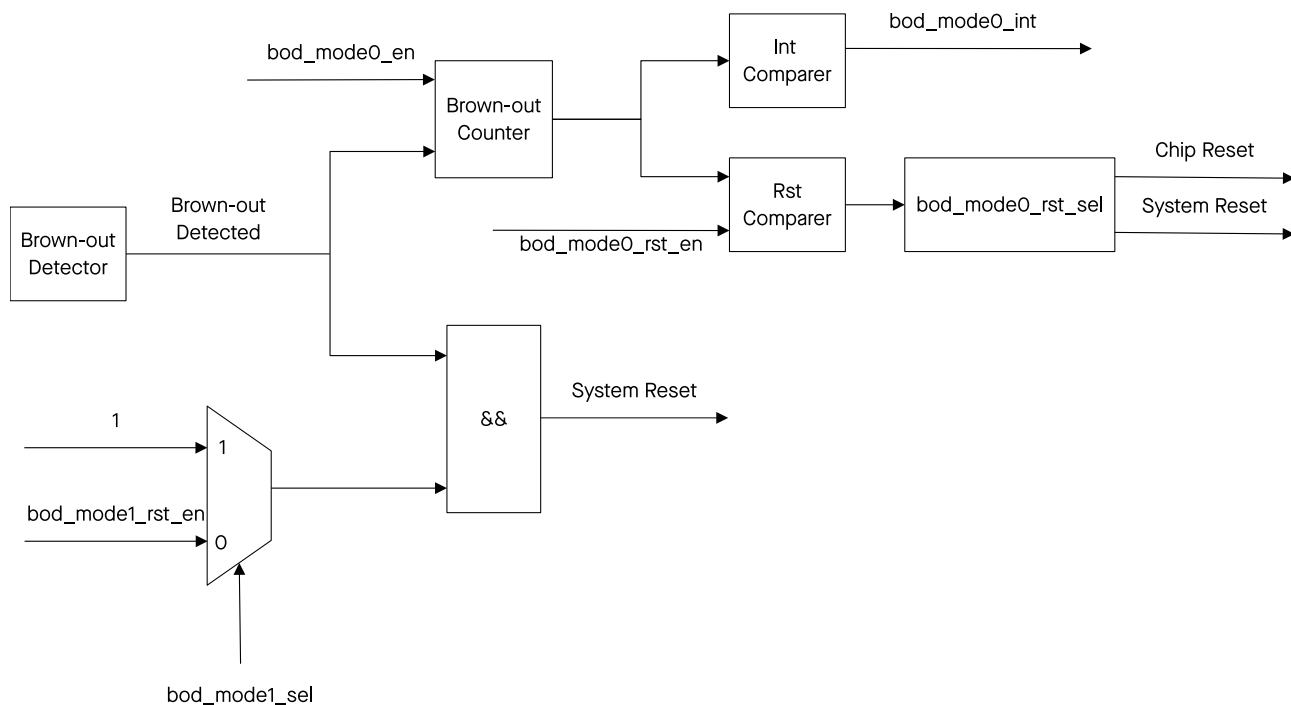


图 15.3-2. 欠压检测器处理方式

图中的选择信号均由寄存器控制，对应关系如下：

- `bod_mode0_en`: [LP_ANA_BOD_MODE0_INTR_ENA](#)
- `bod_mode0_rst_en`: [LP_ANA_BOD_MODE0_RESET_ENA](#)
- `bod_mode0_rst_sel`: [LP_ANA_BOD_MODE0_RESET_SEL](#) 用于选择模式 0 欠压检测器被触发后的复位方式。
 - 0: 芯片复位
 - 1: 系统复位

更多有关复位的信息请参考章节 [6 复位和时钟](#)。

- `bod_mode1_sel`: [LP_ANA_ANA_FIB_ENA](#) 的第一位。
- `bod_mode1_rst_en`: [LP_ANA_BOD_MODE1_RESET_ENA](#)

15.3.3 电压毛刺检测器

ESP32-C5 的电压毛刺检测器的结构如下图所示：

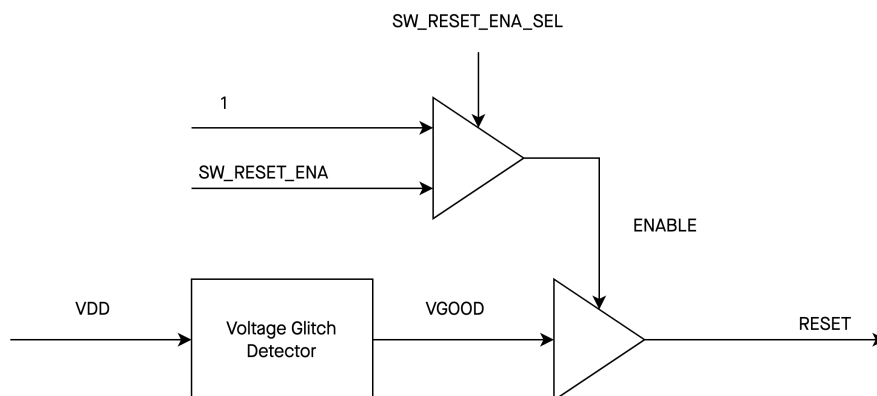


图 15.3-3. 电压毛刺检测器

ESP32-C5 有 4 个电源检测器，每个电源检测器可以检测对应电源上大于 50 ns 的毛刺，包括 VDDPST1、VDDPST2、VDDPST3、VDDA8、VDDA3，其中 VDDPST2 和 VDDPST3 互连，因此可以共用一个电源检测器，所有检测器产生的复位合并为一个 voltage glitch 复位。

电压毛刺检测器的使能控制由 SW_RESET_ENA_SEL 控制，选择为 1 时，由硬件强制开启，选 0 时，由软件寄存器 SW_RESET_ENA 控制。对于不同的电源管脚检测的配置如下表所示：

表 15.3-1. 电源管脚检测的配置

VDD 管脚	SW_RESET_ENA	SW_RESET_ENA_SEL
VDDPST2/3	LP_ANA_PWR_GLITCH_RESET_ENA[0]	LP_ANA_ANA_FIB_ENA[2]
VDDPST1	LP_ANA_PWR_GLITCH_RESET_ENA[1]	LP_ANA_ANA_FIB_ENA[3]
VDDA3	LP_ANA_PWR_GLITCH_RESET_ENA[2]	LP_ANA_ANA_FIB_ENA[4]
VDDA8	LP_ANA_PWR_GLITCH_RESET_ENA[3]	LP_ANA_ANA_FIB_ENA[5]

15.4 中断

ESP32-C5 的电源检测器可以生成以下中断信号并发送给中断矩阵。

- LP_ANA_INTR（该中断供 HP CPU 使用）
- LP_ANA_LP_INTR（该中断供 LP CPU 使用）

这些中断信号由电源检测器的内部中断源生成。表 15.4-1 列出了电源检测器的内部中断源、中断触发条件、以及生成的中断信号。

表 15.4-1. 电源检测器的内部中断源

内部中断源	触发条件	中断信号
LP_ANA_BOD_MODE0_INT	欠压检测器检测到电压低于阈值时触发	LP_ANA_INTR
LP_ANA_BOD_MODE0_LP_INT	欠压检测器检测到电压低于阈值时触发	LP_ANA_LP_INTR

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 [to be added later] > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [15.5 寄存器列表](#)。

15.5 寄存器列表

本小节的所有地址均为相对于电源检测器基地址的地址偏移量（相对地址），具体基地址请见章节 4 [系统和存储器](#) 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_ANA_BOD_MODE0_CNTL_REG	配置欠压检测器模式 0	0x0000	R/W
LP_ANA_BOD_MODE1_CNTL_REG	配置欠压检测器模式 1	0x0004	R/W
LP_ANA_POWER_GLITCH_CNTL_REG	电压毛刺控制寄存器	0x0008	R/W
LP_ANA_FIB_ENABLE_REG	电压毛刺检测器使能控制器	0x000C	R/W
LP_ANA_INT_RAW_REG	LP_ANA_BOD_MODE0_INT 原始中断状态	0x0010	R/WTC/SS
LP_ANA_INT_ST_REG	LP_ANA_BOD_MODE0_INT 屏蔽中断状态	0x0014	RO
LP_ANA_INT_ENA_REG	LP_ANA_BOD_MODE0_INT 使能中断状态	0x0018	R/W
LP_ANA_INT_CLR_REG	LP_ANA_BOD_MODE0_INT 中断清除	0x001C	WT
LP_ANA_LP_INT_RAW_REG	LP_ANA_BOD_MODE0_LP_INT 原始中断状态	0x0020	R/WTC/SS
LP_ANA_LP_INT_ST_REG	LP_ANA_BOD_MODE0_LP_INT 屏蔽中断状态	0x0024	RO
LP_ANA_LP_INT_ENA_REG	LP_ANA_BOD_MODE0_LP_INT 使能中断状态	0x0028	R/W
LP_ANA_LP_INT_CLR_REG	LP_ANA_BOD_MODE0_LP_INT 中断清除	0x002C	WT
版本控制寄存器			
LP_ANA_DATE_REG	版本控制寄存器	0x03FC	R/W

15.6 寄存器

本小节的所有地址均为相对于电源检测器 z 基地址的地址偏移量（相对地址），具体基地址请见章节 4 [系统和存储器](#) 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 15.1. LP_ANA_BOD_MODE0_CNTL_REG (0x0000)

LP_ANA_BOD_MODE0_RESET_ENA				LP_ANA_BOD_MODE0_RESET_SEL				LP_ANA_BOD_MODE0_INTR_WAIT				LP_ANA_BOD_MODE0_PD_RF_ENA				(reserved)								
LP_ANA_BOD_MODE0_INTR_ENA				LP_ANA_BOD_MODE0_CNT_CLR				LP_ANA_BOD_MODE0_RESET_WAIT				LP_ANA_BOD_MODE0_INTR_WAIT				LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA								
31	30	29	28	27	18				17	8				7	6	5	0							
0	0	0	0	0x3ff				1				0				0	0	0	0	0	0	0	0	Reset

LP_ANA_BOD_MODE0_CLOSE_FLASH_ENA 配置是否使能欠压检测器触发 flash 挂起。

0: 禁用

1: 使能

(R/W)

LP_ANA_BOD_MODE0_PD_RF_ENA 配置是否使能欠压检测器关闭射频模块。

0: 禁用

1: 使能

(R/W)

LP_ANA_BOD_MODE0_INTR_WAIT 配置欠压信号有效后, 产生中断的时间, 单位是 LP_FAST_CLK。

(R/W)

LP_ANA_BOD_MODE0_RESET_WAIT 配置欠压信号有效后, 产生复位的时间, 单位是 LP_FAST_CLK。(R/W)

LP_ANA_BOD_MODE0_CNT_CLR 配置是否清空欠压检测器计数器。

0: 撤销

1: 清空

(R/W)

LP_ANA_BOD_MODE0_INTR_ENA 配置是否使能欠压检测器模式 0 的中断。此域为 1 时 [LP_ANA_BOD_MODE0_INT_RAW](#) 和 [LP_ANA_BOD_MODE0_LP_INT_RAW](#) 才有效。

0: 禁用

1: 使能

(R/W)

LP_ANA_BOD_MODE0_RESET_SEL 配置欠压检测器被触发后的复位方式。

0: 芯片复位

1: 系统复位

(R/W)

LP_ANA_BOD_MODE0_RESET_ENA 配置是否使能欠压检测器复位。

0: 禁用

1: 使能

(R/W)

Register 15.2. LP_ANA_BOD_MODE1_CNTL_REG (0x0004)

Diagram of the LP_ANA_BOD_MODE1_RESET_ENA register. The register is 32 bits wide. Bit 31 is labeled LP_ANA_BOD_MODE1_RESET_ENA. Bits 0-30 are labeled (reserved). The register value is shown as 0.

LP_ANA_BOD_MODE1_RESET_ENA 配置是否使能欠压检测器模式 1。

0: 禁用

1: 使能

(R/W)

Register 15.3. LP_ANA_POWER_GLITCH_CNTL_REG (0x0008)

Diagram illustrating the structure of the LP_ANA_PWR_GLITCH_RESET_ENA register:

- Bit 31: (reserved)
- Bits 30-28: (reserved)
- Bit 27: LP_ANA_PWR_GLITCH_RESET_ENA
- Bits 26-0: (reserved)

LP_ANA_PWR_GLITCH_RESET_ENA 配置是否使能电压毛刺检测器。其中 bit0, bit1, bit2, bit3 分别对应 VDDPST2/3, VDDPST1, VDDA3, VDDA8。

0: 禁用

1: 使能

(R/W)

Register 15.4. LP_ANA_FIB_ENABLE_REG (0x000C)

LP_ANA_FIB_ENA																															
31																															0
0xffffffff																															
Reset																															

LP_ANA_FIB_ENABLE 配置电压毛刺检测器的使能控制。其中 bit2, bit3, bit4, bit5 分别对应 VD-DPST2/3, VDDPST1, VDDA3, VDDA8。

0: 由 **LP_ANA_PWR_GLITCH_RESET_ENA** 控制

1: 由硬件强制开启

(R/W)

Register 15.5. LP_ANA_INT_RAW_REG (0x0010)

LP_ANA_BOD_MODE0_INT_RAW																													
31	30	(reserved)																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																													

LP_ANA_BOD_MODE0_INT_RAW **LP_ANA_BOD_MODE0_INT** 的原始中断状态。(R/WTC/SS)

Register 15.6. LP_ANA_INT_ST_REG (0x0014)

LP_ANA_BOD_MODE0_INT_ST																													
31	30	(reserved)																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																													

LP_ANA_BOD_MODE0_INT_ST **LP_ANA_BOD_MODE0_INT** 的屏蔽中断状态。(RO)

Register 15.7. LP_ANA_INT_ENA_REG (0x0018)

Diagram illustrating the structure of the LP_AVA_BOD_MODE0_INT_ENA register. The register is 32 bits wide. Bit 31 is labeled LP_AVA_BOD_MODE0_INT_ENA. Bits 30 to 0 are labeled (reserved).

LP_ANA_BOD_MODE0_INT_ENA 写1使能 LP_ANA_BOD_MODE0_INT。(R/W)

Register 15.8. LP_ANA_INT_CLR_REG (0x001C)

Diagram illustrating the structure of the LP_AVA_BOD_MODE0_INT_CLR register. The register is 32 bits wide. Bit 31 is labeled LP_AVA_BOD_MODE0_INT_CLR. Bits 30-0 are labeled (reserved). The register is reset to 0.

LP_ANA_BOD_MODE0_INT_CLR 写1清除 LP_ANA_BOD_MODE0_INT。(WT)

Register 15.9. LP_ANA_LP_INT_RAW_REG (0x0020)

Diagram illustrating the structure of the LP_ANA_BOD_MODE0_LP_INT_RAW register. The register is 32 bits wide, with bits 31 down to 30 labeled as LP_ANA_BOD_MODE0_LP_INT_RAW. Bit 0 is labeled as Reset.

LP_ANA_BOD_MODE0_LP_INT_RAW LP_ANA_BOD_MODE0_LP_INT 的原始中断状态。(R/WTC/SS)

Register 15.10. LP_ANA_LP_INT_ST_REG (0x0024)

LP_ANA_BOD_MODE0_LP_INT_ST																																(reserved)															
31	30																															0															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset												

LP_ANA_BOD_MODE0_LP_INT_ST [LP_ANA_BOD_MODE0_LP_INT](#) 的屏蔽中断状态。(RO)

Register 15.11. LP_ANA_LP_INT_ENA_REG (0x0028)

LP_ANA_BOD_MODE0_LP_INT_ENA																																(reserved)		0
31	30																														0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

LP_ANA_BOD_MODE0_LP_INT_ENA 写 1 使能 [LP_ANA_BOD_MODE0_LP_INT](#)。(R/W)

Register 15.12. LP_ANA_LP_INT_CLR_REG (0x002C)

LP_ANA_BOD_MODE0_LP_INT_CLR																																(reserved)																0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
31	30																															0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

LP_ANA_BOD_MODE0_LP_INT_CLR 写 1 清除 [LP_ANA_BOD_MODE0_LP_INT](#)。(WT)

Register 15.13. LP_ANA_DATE_REG (0x03FC)

LP_ANA_CLK_EN		LP_ANA_LP_ANA_DATE	
31	30	0	
0	0x2312280		Reset

LP_ANA_LP_ANA_DATE 版本控制寄存器。(R/W)

LP_ANA_CLK_EN 使能强制打开配置寄存器时钟。

- 0: 自动时钟门控
- 1: 强制打开寄存器时钟
- 此域的配置不影响寄存器的读写。
- (R/W)

第 IV 卷

加密和安全组件

该部分专门介绍芯片的安全功能，包括 AES、ECC 等加密加速器，数字签名、随机数生成、加密/解密算法等安全特性，展示了芯片在加密和安全数据处理方面的性能。

第 16 章

AES 加速器 (AES)

16.1 概述

ESP32-C5 内置 AES（高级加密标准）硬件加速器可使用 AES 算法，完成数据的加解密运算，具有 [Typical AES](#) 和 [DMA-AES](#) 两种工作模式。整体而言，相比基于纯软件的 AES 运算，AES 硬件加速器能够极大地提高运算速度。

16.2 主要特性

ESP32-C5 支持以下特性：

- Typical AES 工作模式
 - AES-128/AES-256 加解密运算
- DMA-AES 工作模式
 - AES-128/AES-256 加解密运算
 - 块（加密）模式
 - * ECB (Electronic Codebook)
 - * CBC (Cipher Block Chaining)
 - * OFB (Output Feedback)
 - * CTR (Counter)
 - * CFB8 (8-bit Cipher Feedback)
 - * CFB128 (128-bit Cipher Feedback)
 - 中断发生

16.3 时钟和复位

AES 加速器的激活需使能 [PCR_AES_CONF_REG](#) 寄存器中的 [PCR_AES_CLK_EN](#) 位，并同时清零 [PCR_AES_RST_EN](#) 位。此外，由于密码学加速器模块之间的资源复用，用户还需要额外清零 [PCR_DS_CONF_REG](#) 寄存器中的 [PCR_DS_RST_EN](#) 位。

16.4 工作模式简介

ESP32-C5 内置的 AES 加速器支持 [Typical AES](#) 和 [DMA-AES](#) 两种工作模式。

- Typical AES 工作模式：
 - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 [NIST FIPS 197](#) 标准中的 AES-128 和 AES-256 加解密运算。

这种情况下，明文和密文的读写操作统一通过 CPU 访问完成。

- DMA-AES 工作模式：
 - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 [NIST FIPS 197](#) 标准中的 AES-128 和 AES-256 加解密运算；
 - 还支持 [NIST SP 800-38A](#) 标准中的 ECB、CBC、OFB、CTR、CFB8、CFB128 块加密模式运算。

在这种情况下，明文和密文的传输通过硬件上的 DMA 完成，运算完成时会有中断发生。

用户可通过配置 [AES_DMA_ENABLE_REG](#) 选择 AES 加速器的工作模式，具体参考表 16.4-1。

表 16.4-1. 工作模式

AES_DMA_ENABLE_REG	工作模式
0	Typical AES
1	DMA-AES

用户可通过配置 [AES_MODE_REG](#) 寄存器选择密钥长度和加解密方向，具体可参考表 16.4-2。

表 16.4-2. 密钥长度和加解密方向

AES_MODE_REG [2:0]	密钥长度和加解密方向
0	AES-128 加密
1	保留
2	AES-256 加密
3	保留
4	AES-128 解密
5	保留
6	AES-256 解密
7	保留

有关 Typical AES 和 DMA-AES 两种工作模式的具体介绍，请见下方 16.5 章节和 16.6 章节。

注意：

ESP32-C5 的[数字签名算法 \(DSA\)](#) 模块也会调用 AES 加速器。此时，用户无法正常访问 AES 加速器。

16.5 Typical AES 工作模式

在 Typical AES 工作模式下，AES 加速器的状态值可查看寄存器 `AES_STATE_REG`，具体见表 16.5-1 所示：

表 16.5-1. Typical AES 模式下状态返回值

AES_STATE_REG	描述	状态说明
0	IDLE	加速器空闲或运算完成
1	WORK	加速器忙于运算

16.5.1 密钥、明文、密文

寄存器 `AES_KEY_n_REG` 用于存放密钥，由 8 个 32 位寄存器组成。

- 如果为 AES-128 加解密运算，则 128 位密钥在寄存器 `AES_KEY_0_REG` ~ `AES_KEY_3_REG` 中。
- 如果为 AES-256 加解密运算，则 256 位密钥在寄存器 `AES_KEY_0_REG` ~ `AES_KEY_7_REG` 中。

寄存器 `AES_TEXT_IN_m_REG` 和 `AES_TEXT_OUT_m_REG` 用于存放明文和密文，各由 4 个 32 位寄存器组成。

- 如果为 AES-128 或 AES-256 加密运算，则运算开始之前用明文初始化寄存器 `AES_TEXT_IN_m_REG`。运算完成之后，AES 加速器将把密文更新入寄存器 `AES_TEXT_OUT_m_REG`。
- 如果为 AES-128 或 AES-256 解密运算，则运算开始之前用密文初始化寄存器 `AES_TEXT_IN_m_REG`。运算完成之后，AES 加速器将把明文更新入寄存器 `AES_TEXT_OUT_m_REG`。

16.5.2 字节序

文本字节序

在 Typical AES 工作模式下，AES 加速器可以使用密钥对 128 位的块进行加解密。在操作寄存器 `AES_TEXT_IN_m_REG` 和 `AES_TEXT_OUT_m_REG` 中的数据时，用户应遵循表 16.5-2 中定义的文本字节序。

表 16.5-2. Typical AES 文本字节序

明文/密文					
State ¹		c ²			
		0	1	2	3
r	0	AES_TEXT_x_0_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_3_REG[7:0]
	1	AES_TEXT_x_0_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_3_REG[15:8]
	2	AES_TEXT_x_0_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_3_REG[23:16]
	3	AES_TEXT_x_0_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_3_REG[31:24]

¹ 有关“State（以及 c 和 r）”的详细定义，请参考 [NIST FIPS 197](#) 中 3.4 The State 章节。

² 其中，x = IN 或 OUT。

密钥字节序

在 Typical AES 工作模式下，在向寄存器 `AES_KEY_n_REG` 中填入数据时，用户应遵循表 16.5-3 和表 16.5-4 中定义的文本字节序。

表 16.5-3. AES-128 密钥字节序

Bit ¹	w[0]	w[1]	w[2]	w[3] ²
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

¹ Bit 列代表 w[0] ~ w[3] 每个 word 中的各个字节。

² w[0] ~ w[3] 符合标准 [NIST FIPS 197](#) 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

表 16.5-4. AES-256 密钥字节序

Bit ¹	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7] ²
[31:24]	AES_KEY_O_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
[23:16]	AES_KEY_O_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_7_REG[15:8]
[15:8]	AES_KEY_O_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_7_REG[23:16]
[7:0]	AES_KEY_O_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

¹ Bit 列代表 w[0] ~ w[7] 每个 word 中的各个字节。
² w[0] ~ w[7] 符合标准 [NIST FIPS 197](#) 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

16.5.3 Typical AES 工作模式的流程

单次运算

1. 对寄存器 `AES_DMA_ENABLE_REG` 写入 0。
2. 初始化寄存器 `AES_MODE_REG`、`AES_KEY_n_REG`、`AES_TEXT_IN_m_REG`。
3. 启动运算。对寄存器 `AES_TRIGGER_REG` 写入 1。
4. 等待运算完成。轮询寄存器 `AES_STATE_REG`，直到读到 0。
5. 从寄存器 `AES_TEXT_OUT_m_REG` 读取结果。

连续运算

在连续运算过程中，每次运算完成之后，只有寄存器 `AES_TEXT_IN_m_REG` 和 `AES_TEXT_OUT_m_REG` (m : 0-3) 会被 AES 加速器更新，而 `AES_DMA_ENABLE_REG`、`AES_MODE_REG`、`AES_KEY_n_REG` 等寄存器中的内容不会变化。所以进行连续运算时可以简化初始化操作。

1. 第一次运算之前对寄存器 `AES_DMA_ENABLE_REG` 写入 0。
2. 第一次运算之前初始化寄存器 `AES_MODE_REG` 和 `AES_KEY_n_REG`。
3. 更新寄存器 `AES_TEXT_IN_m_REG`。
4. 启动运算。对寄存器 `AES_TRIGGER_REG` 写入 1。
5. 等待运算完成。轮询寄存器 `AES_STATE_REG`，直到读到 0。
6. 从寄存器 `AES_TEXT_OUT_m_REG` 读取结果。返回步骤 3，进行下一轮运算。

16.6 DMA-AES 工作模式

在 DMA-AES 工作模式下，AES 加速器可支持 ECB、CBC、OFB、CTR、CFB8、CFB128 六种块模式运算。用户可通过配置 `AES_BLOCK_MODE_REG` 寄存器选择具体运算类型，具体可参考表 16.6-1。

表 16.6-1. 块模式选择

<code>AES_BLOCK_MODE_REG[2:0]</code>	块模式
0	ECB (Electronic Code Book)
1	CBC (Cipher Block Chaining)
2	OFB (Output FeedBack)
3	CTR (Counter)
4	CFB8 (8-bit Cipher FeedBack)
5	CFB128 (128-bit Cipher FeedBack)
6	保留
7	保留

AES 加速器的状态值可查看寄存器 [AES_STATE_REG](#)，具体见表 16.6-2 所示：

表 16.6-2. DMA-AES 模式下状态返回值

AES_STATE_REG	描述	状态说明
0	IDLE	加速器空闲
1	WORK	加速器忙于运算
2	DONE	加速器运算完成

AES 加速器在 DMA-AES 工作模式下允许中断发生，软件清零。中断功能默认关闭，用户可通过将 [AES_INT_ENA_REG](#) 寄存器配置为 1 开启中断。如开启中断功能，AES 加速器在完成运算时，中断发生。

16.6.1 密钥、明文、密文

块运算模式

在块运算模式下，AES 加速器的源数据来自 DMA，结果数据也将被写入 DMA。

- 如果为加密运算，则 DMA 从存储器中读取明文数据流并将其传给 AES。AES 运算出密文后将密文写入 DMA。DMA 再将密文写入存储器。
- 如果为解密运算，则 DMA 从存储器中读取密文数据流并将其传给 AES。AES 运算出明文后将明文写入 DMA。DMA 再将明文写入存储器。

AES 加速器在进行块运算时，结果数据与源数据的大小保持一致。此时，DMA 的数据搬运过程和 AES 的运算过程有所交叠，因此总工作时间有所减少。

值得注意的是，AES 加速器在 DMA-AES 工作模式下要求源数据的大小必须是 128 位的整数倍，否则需要将原始明文封装为 128 位的整数倍，即在原比特串 (bit string) 尾部尽可能少的补 “0”，具体过程见表 16.6-3 所示。

表 16.6-3. TEXT-PADDING

函数： TEXT-PADDING()
输入： X ，比特串。
输出： $Y = \text{TEXT-PADDING}(X)$ ，长度为最接近 128 比特整数倍的值。
步骤： 假设 X 是一个数据流，可以按如下方式分成 n 个部分： $X = X_1 X_2 \cdots X_{n-1} X_n$ 此处， $X_1 X_2 \cdots X_{n-1}$ 的长度均为 128 比特，而 X_n 的长度为 t ($0 \leq t \leq 127$)。 如果 $t = 0$ ，则 $\text{TEXT-PADDING}(X) = X;$ 如果 $0 < t \leq 127$ ，定义 X_n^* 块的长度为 128 比特，且 $X_n^* = X_n 0^{128-t}$ ，则 $\text{TEXT-PADDING}(X) = X_1 X_2 \cdots X_{n-1} X_n^* = X 0^{128-t}$

16.6.2 字节序

在 DMA-AES 工作模式下，源数据和结果数据的传输完全由 DMA 完成，因此不支持字节序的控制调节，但要求它们在存储器中以一定的方式来存放，且要求数据量必须是块的整数倍。

举例说明，假设 DMA 需要搬运源数据：

- 十六进制：0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- 数据大小：相当于 2 个块

假设起始地址为 0x0280，则源数据在存储器中的存放位置如表 16.6-4 所示。结果数据也遵从相同的存放规则，在此不多做介绍。

表 16.6-4. DMA-AES 存储字节序

地址	字节	地址	字节	地址	字节	地址	字节
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

16.6.3 标准增量函数

AES 加速器在进行 CTR 块运算时，还可提供两种标准增量函数供用户选择：INC₃₂ 和 INC₁₂₈。用户可通过将寄存器 AES_INC_SEL_REG 置为 0 或 1 选择 INC₃₂ 或 INC₁₂₈ 标准增量函数。更多有关标准增量函数的内容，请见 [NIST SP 800-38A](#) 标准中的 B.1 The Standard Incrementing Function 章节。

16.6.4 块个数

寄存器 AES_BLOCK_NUM_REG 存放明文或密文的块个数 (Block Number)，其值等于 $\text{length}(\text{TEXT-PADDING}(P))/128$ ，也等于 $\text{length}(\text{TEXT-PADDING}(C))/128$ 。这里的 P 指明文 (plaintext)， C 指密文 (ciphertext)。该寄存器仅在 DMA-AES 工作模式下有意义。

16.6.5 初始向量

存储器 AES_IV_MEM 的空间大小为 16 字节，仅在块运算模式下有效。对于 CBC/OFB/CFB8/CFB128 操作，AES_IV_MEM 用于存放初始向量 (Initialization Vector, IV) 的值。对于 CTR 操作，AES_IV_MEM 存放初始计数器 (Initial Counter Block, ICB) 的值。

IV 和 ICB 都是 128 位的比特串，从左向右被分割成 16 个字节 (Byte0, Byte1, Byte2, ..., Byte15)，构成一个字节序列，在 AES_IV_MEM 中存放时需要遵循表 16.6-4 中的字节序规则，即 Byte0 存放在 AES_IV_MEM 中的最低地址中，Byte15 存放在 AES_IV_MEM 中的最高地址中。

更多有关 IV 和 ICB 的信息，请参考 [NIST SP 800-38A](#) 标准。

16.6.6 DMA-AES 工作模式的流程

1. 选择一条 DMA 通道与 AES 加速器连接，配置 DMA 链表，而后启动 DMA。详情请见章节 3 通用 DMA 控制器 (GDMA)。
2. 配置 AES：
 - 对寄存器 AES_DMA_ENABLE_REG 写入 1。

- 选择是否开启中断。根据需要设置寄存器 `AES_INT_ENA_REG` 的值。
 - 初始化 `AES_MODE_REG` 和 `AES_KEY_n_REG` 寄存器。
 - 配置 `AES_BLOCK_MODE_REG` 寄存器，选择具体块加密模式。详见表 16.6-1。
 - 初始化寄存器 `AES_BLOCK_NUM_REG`，请参照章节 16.6.4。
 - 初始化寄存器 `AES_INC_SEL_REG`（仅在 CTR 块模式下使用）。
 - 初始化存储器 `AES_IV_MEM`（在 ECB 块模式下不使用）。
3. 启动运算。对寄存器 `AES_TRIGGER_REG` 写入 1。
 4. 等待运算完成。轮询寄存器 `AES_STATE_REG`，直到读到 2。如果开启了中断功能，也可以等待 `AES_INT` 中断产生。
 5. 确认 DMA 完成从 AES 到内存的数据传输。此时，结果数据已经被 DMA 写入 memory，可以直接从中读取。详情请参考章节 3 通用 DMA 控制器 (GDMA)。
 6. 如果开启了中断，当处理中断程序完成后，请及时对寄存器 `AES_INT_CLR_REG` 写 1 以清除中断。
 7. 对寄存器 `AES_DMA_EXIT_REG` 写入 1 释放 AES 加速器。之后如果再读取寄存器 `AES_STATE_REG` 将读到 0。该步操作可以提前完成，但必须在步骤 4 之后。

16.7 抗攻击伪轮函数

为了增强抗攻击性能，ESP32-C5 的 AES 引入了伪轮函数。

启用伪轮函数之后，AES 会在原始操作轮次的前后随机插入伪轮次。在伪轮次中，AES 使用硬件自动生成的伪密钥执行一轮伪操作。伪轮次中的操作不会影响原始 AES 操作的结果。

用户可以将 `AES_PSEUDO_EN` 配置为 1 来启用伪轮函数。伪轮次的配置如下。

伪轮次总数

以下寄存器字段控制在 AES 操作中随机插入的伪轮次总数：

- `pseudo_base`： `AES_PSEUDO_BASE` 的值，配置伪轮次的基本次数。
- `pseudo_inc`： `AES_PSEUDO_INC` 的值，配置伪轮次的随机增量次数。

伪轮次总数在以下范围内随机分布：

$$[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$$

随机数更新频率

用户可以通过配置 `AES_PSEUDO_RNG_CNT` 来确定伪轮函数中随机密钥更新的频率。配置的值越大，更新频率越高。一般来说，建议设置为最大值 7。

16.8 存储器列表

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 [4 系统和存储器](#) 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	大小（比特）	起始地址	结束地址	访问权限
AES_IV_MEM	存储器 IV	16 字节	0x0050	0x005F	(R/W)

16.9 寄存器列表

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

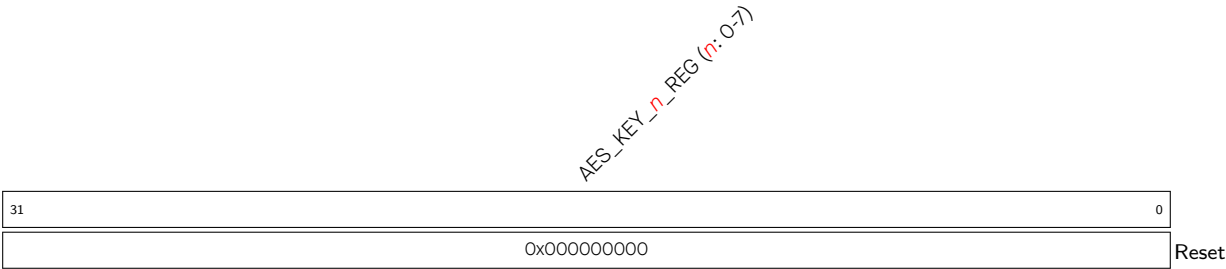
名称	描述	地址	访问
密钥寄存器			
AES_KEY_0_REG	AES 密钥资料寄存器 0	0x0000	R/W
AES_KEY_1_REG	AES 密钥资料寄存器 1	0x0004	R/W
AES_KEY_2_REG	AES 密钥资料寄存器 2	0x0008	R/W
AES_KEY_3_REG	AES 密钥资料寄存器 3	0x000C	R/W
AES_KEY_4_REG	AES 密钥资料寄存器 4	0x0010	R/W
AES_KEY_5_REG	AES 密钥资料寄存器 5	0x0014	R/W
AES_KEY_6_REG	AES 密钥资料寄存器 6	0x0018	R/W
AES_KEY_7_REG	AES 密钥资料寄存器 7	0x001C	R/W
TEXT_IN 寄存器			
AES_TEXT_IN_0_REG	源数据资料寄存器 0	0x0020	R/W
AES_TEXT_IN_1_REG	源数据资料寄存器 1	0x0024	R/W
AES_TEXT_IN_2_REG	源数据资料寄存器 2	0x0028	R/W
AES_TEXT_IN_3_REG	源数据资料寄存器 3	0x002C	R/W
TEXT_OUT 寄存器			
AES_TEXT_OUT_0_REG	结果数据资料寄存器 0	0x0030	RO
AES_TEXT_OUT_1_REG	结果数据资料寄存器 1	0x0034	RO
AES_TEXT_OUT_2_REG	结果数据资料寄存器 2	0x0038	RO
AES_TEXT_OUT_3_REG	结果数据资料寄存器 3	0x003C	RO
控制/配置寄存器			
AES_MODE_REG	密钥长度和加解密方向配置寄存器	0x0040	R/W
AES_TRIGGER_REG	开始运算控制寄存器	0x0048	WT
AES_DMA_ENABLE_REG	工作模式配置寄存器	0x0090	R/W
AES_BLOCK_MODE_REG	块运算模式配置寄存器	0x0094	R/W
AES_BLOCK_NUM_REG	块数量配置寄存器	0x0098	R/W
AES_INC_SEL_REG	标准增量函数配置寄存器	0x009C	R/W
AES_DMA_EXIT_REG	退出运算控制寄存器	0x00B8	WO
AES_PSEUDO_REG	伪轮函数配置寄存器	0x00D0	R/W
状态寄存器			
AES_STATE_REG	运算状态寄存器	0x004C	RO
中断寄存器			
AES_INT_CLR_REG	DMA-AES 中断清除寄存器	0x00AC	WT
AES_INT_ENA_REG	DMA-AES 中断使能寄存器	0x00B0	R/W
版本控制寄存器			
AES_DATE_REG	版本控制寄存器	0x00B4	R/W

16.10 寄存器

本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

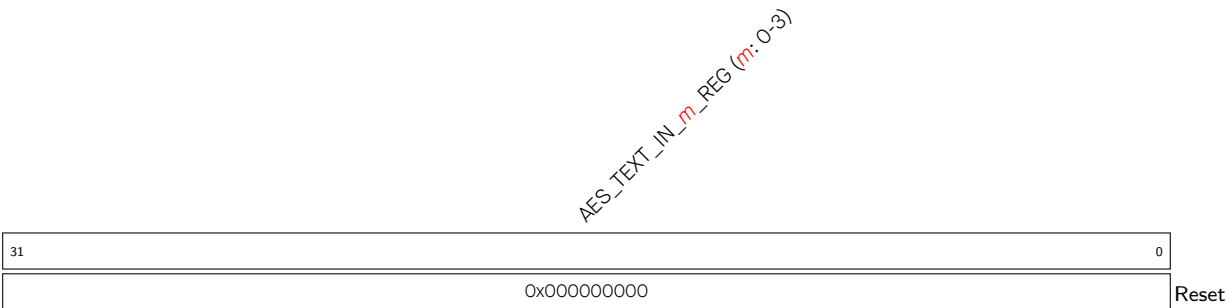
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 16.1. AES_KEY_ *n* _REG (*n*: 0-7) (0x0000+4**n*)



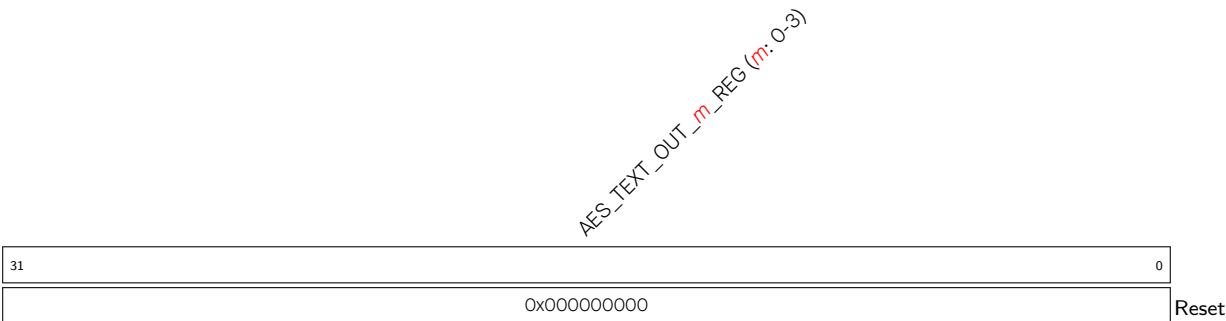
AES_KEY_ *n* _REG (*n*: 0-7) 表示 AES 密钥资料。(R/W)

Register 16.2. AES_TEXT_IN_ *m* _REG (*m*: 0-3) (0x0020+4**m*)



AES_TEXT_IN_ *m* _REG (*m*: 0-3) 表示 Typical AES 工作模式下的输入文本。(R/W)

Register 16.3. AES_TEXT_OUT_ *m* _REG (*m*: 0-3) (0x0030+4**m*)



AES_TEXT_OUT_ *m* _REG (*m*: 0-3) 表示 Typical AES 工作模式下的输出文本。(RO)

Register 16.4. AES_MODE_REG (0x0040)

(reserved)																AES_MODE	
31															3	2	0
0x00000000																0	Reset

AES_MODE 配置 AES 加速器的密钥长度和解密方向。

- 0: AES-128 加密
- 1: 保留
- 2: AES-256 加密
- 3: 保留
- 4: AES-128 解密
- 5: 保留
- 6: AES-256 解密
- 7: 保留

(R/W)

Register 16.5. AES_DMA_ENABLE_REG (0x0090)

(reserved)																AES_DMA_ENABLE	
31															1	0	
0x00000000																0	Reset

AES_DMA_ENABLE 配置 AES 加速器的工作模式。

- 0: Typical AES
- 1: DMA-AES

(R/W)

Register 16.6. AES_BLOCK_MODE_REG (0x0094)

(reserved)																AES_BLOCK_MODE			
																3	2		
																		0	
0x00000000																		0	Reset

AES_BLOCK_MODE 配置 AES 加速器在 DMA-AES 工作模式下的块模式。

- 0: ECB (Electronic Code Block)
- 1: CBC (Cipher Block Chaining)
- 2: OFB (Output FeedBack)
- 3: CTR (Counter)
- 4: CFB8 (8-bit Cipher FeedBack)
- 5: CFB128 (128-bit Cipher FeedBack)
- 6: 保留
- 7: 保留

(R/W)

Register 16.7. AES_BLOCK_NUM_REG (0x0098)

AES_BLOCK_NUM																			
0x00000000																			

AES_BLOCK_NUM 代表 DMA-AES 工作模式下待加解密的文本块数。详情请见章节 16.6.4。(R/W)

Register 16.8. AES_INC_SEL_REG (0x009C)

(reserved)																AES_INC_SEL			
																1	0		
																		0	
0x00000000																		0	Reset

AES_INC_SEL 配置 CTR 块模式使用的标准增量函数。

- 0: INC₃₂ 标准增量函数
- 1: INC₁₂₈ 标准增量函数

(R/W)

Register 16.9. AES_TRIGGER_REG (0x0048)

(reserved)																															AES_TRIGGER	
31																															1	0
0x00000000																															x	Reset

AES_TRIGGER 配置是否启动 AES 运算。

- 0: 无效果
- 1: 启动 (WT)

Register 16.10. AES_STATE_REG (0x004C)

(reserved)																AES_STATE		
31															2	1	0	
0x00000000																0x0		Reset

AES_STATE 代表 AES 加速器的状态。

在 Typical AES 工作模式下，

- 0: IDLE
- 1: WORK
- 2: 无效果
- 3: 无效果

在 DMA-AES 工作模式下，

- 0: IDLE
- 1: WORK
- 2: DONE
- 3: 无效果 (RO)

Register 16.11. AES_DMA_EXIT_REG (0x00B8)

(reserved)															AES_DMA_EXIT						
31																		1	0		
0x00000000																			x	Reset	

AES_DMA_EXIT 配置是否退出 AES 运算。

0: 无效果

1: 退出

仅在 DMA-AES 模式下有效。(WO)

Register 16.12. AES_PSEUDO_REG (0x00D0)

(reserved)																				AES_PSEUDO_RNG_CNT		AES_PSEUDO_INC		AES_PSEUDO_BASE		AES_PSEUDO_EN		
31										10										9	7	6	5	4	1		0	
0										0										7		2		2		0		Reset

AES_PSEUDO_EN 配置是否启用 AES 伪轮函数。

0: 禁用

1: 启用

(R/W)

AES_PSEUDO_BASE 配置伪轮次的基本次数。(R/W)

AES_PSEUDO_INC 配置伪轮次的随机增量次数。(R/W)

AES_PSEUDO_RNG_CNT 配置伪轮函数中随机密钥更新的频率。(R/W)

Register 16.13. AES_INT_CLR_REG (0x00AC)

(reserved)															AES_INT_CLR					
31																			1	0
0x00000000																			x	Reset

AES_INT_CLR 写 1 清除 AES 中断。(WT)

Register 16.14. AES_INT_ENA_REG (0x00B0)

(reserved)																															AES_INT_ENA	
31																															1	0
0x00000000																															0	Reset

AES_INT_ENA 写 1 使能 AES 中断。(R/W)

Register 16.15. AES_DATE_REG (0x00B4)

(reserved)															AES_DATE	
31				28	27											0
0	0	0	0	0x2312070											Reset	

AES_DATE 版本控制寄存器。(R/W)

第 17 章

ECC 加速器 (ECC)

17.1 概述

椭圆曲线密码学 (Elliptic Curve Cryptography, ECC) 是一种基于椭圆曲线数学的公开密钥加密演算法，其优势在于相对于 RSA 算法，使用较小长度的密钥就能够提供相当等级的加密安全性。

ESP32-C5 ECC 硬件加速器支持基于可选曲线的多种基础运算，用以实现对 ECC 基本运算及其衍生算法（如 ECDSA 等算法）的加速。

17.2 主要特性

ESP32-C5 ECC 硬件加速器具备以下特性：

- 支持三种可选 ECC 曲线，即 [FIPS 186-5](#) 中定义的 P-192、P-256 和 P-384
- 提供 11 种可选工作模式
- 支持高级安全模式
- 提供计算完成的中断和中断控制

17.3 ECC 背景知识

本节介绍了与 ECC 硬件加速器相关的背景知识以及章节中会应用到的专业名词。

17.3.1 椭圆曲线与曲线上的点

ECC 是一种基于大素数的有限域椭圆曲线的算法，这一椭圆曲线的数学表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

其中，

- p 是素数，
- a 和 b 为两个小于 p 的非负整数，
- (x, y) 为满足该椭圆曲线方程的点。

17.3.2 仿射坐标系与 Jacobian 坐标系

一条椭圆曲线：

- 在仿射坐标系下的表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

- 在 Jacobian 坐标系下的表达式为：

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

一个曲线上的点在仿射坐标系下的表示 (x, y) 与在 Jacobian 坐标系下的表示 (X, Y, Z) 有如下转换关系：

- 从 Jacobian 坐标系到仿射坐标系的转换：

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- 从仿射坐标系到 Jacobian 坐标系的转换：

$$X = x$$

$$Y = y$$

$$Z = 1$$

17.3.3 内存块

ECC 硬件加速器的内存块用于存储 ECC 运算中所用到的输入数据和输出数据。

表 17.3-1. ECC 硬件加速器内存块

内存块 ¹	大小 (字节)	起始地址 ²	结束地址 ²	访问权限
ECC_MULT_Mem_k	48	0x100	0x12F	R/W
ECC_MULT_Mem_Px	48	0x130	0x15F	R/W
ECC_MULT_Mem_Py	48	0x160	0x18F	R/W
ECC_MULT_Mem_Qx	48	0x190	0x1BF	R/W
ECC_MULT_Mem_Qy	48	0x1C0	0x1EF	R/W
ECC_MULT_Mem_Qz	48	0x1F0	0x21F	R/W

¹ 内存块在各个工作模式中储存的数据内容不同，详见章节 17.4.2。

² 采用相对于 ECC 加速器基地址的偏移量。详见章节 4 系统和存储器 中的表 4.3-2。

17.3.4 数据与数据块

在 ECC 硬件加速器模块中会用到的数据位宽可能为 192, 256 或 384 位，这取决于进行运算的曲线。

假设一个数据为 $D[255:0]$ ，则其可以被划分为八个 32 位的数据块 $D[n][31:0]$ ($n = 0, 1, \dots, 7$)，序号数低的数据块对应二进制低位，即：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

17.3.5 数据存储

数据存储即将数据存储进一个内存块的操作，该数据为 ECC 算法的输入数据。具体而言，将数据写入一个 ECC 内存块相当于将该数据 $D[n][31:0]$ ($n = 0, 1, \dots, 11$) 依次写入“该内存块起始地址 + $4 \times n$ ”：

- 写入 $D[0]$ 至“起始地址”
- 写入 $D[1]$ 至“起始地址 + 4”
- ...
- 写入 $D[11]$ 至“起始地址 + 44”

说明：

进行数据存储操作时，只需要写入数据长度的数据块即可，不需要在高位补 0。例如，在写入 192 位长度的数据时，只需要存储 192 位的数据，不需要在数据的高位补 0。

17.3.6 数据读取

数据读取即将一个数据从一个内存块读出的操作，该数据为 ECC 算法的输出数据。具体来说，从一个 ECC 内存块读数据相当于从“该内存块起始地址 + $4 \times n$ ”依次读出 $D[n][31:0]$ ($n = 0, 1, \dots, 11$)：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- ...
- 从“起始地址 + 44”读出 $D[11]$

说明：

进行数据读取操作时，只需要读取数据长度的数据块即可。例如，在读取 192 位长度的数据时，只需要读取低 192 位（即 6 个数据块）的数据。

17.3.7 标准运算与 Jacobian 运算

ESP32-C5 ECC 硬件加速器中，所有标准运算（包括标准点验证、标准点加和标准点乘）的输入数据以及输出数据中的点均在仿射坐标系中；相对应地，所有 Jacobian 运算（包括 Jacobian 点验证、Jacobian 点加和 Jacobian 点乘）的输入数据以及输出数据中的点均在 Jacobian 坐标系中。

17.4 功能描述

17.4.1 曲线模式

ESP32-C5 ECC 硬件加速器共支持 3 种曲线模式，每种曲线模式唯一对应一条椭圆曲线。用户通过配置 `ECC_MULT_CURVE_MODE` 可选定密钥长度模式，其与椭圆曲线的对应关系如表 17.4-1 所示。

表 17.4-1. ECC 加速器曲线模式控制

ECC_MULT_CURVE_MODE	对应椭圆曲线*
0	FIPS P-192
1	FIPS P-256
2	FIPS P-384

* FIPS P-192/P-256/P-384 的曲线定义，请参考 [FIPS 186-5](#)。

17.4.2 工作模式

ESP32-C5 ECC 硬件加速器共有 11 种工作模式，每种工作模式基于选定曲线进行不同操作。用户通过配置 `ECC_MULT_WORK_MODE` 可选定工作模式，其值与工作模式的对应关系如表 17.4-2 所示。

表 17.4-2. ECC 硬件加速器工作模式控制

ECC_MULT_WORK_MODE	对应模式
0	标准点乘
1	保留项，不可用
2	标准点验证
3	标准点验证 + 标准点乘
4	Jacobian 点乘
5	点加
6	Jacobian 点验证
7	标准点验证 + Jacobian 点乘
8	模加
9	模减
10	模乘
11	模除

说明：
请注意，Jacobian 点乘模式的运算速度比标准点乘模式约快 10%。

每种工作模式的具体计算和输入/输出数据，请参照下述子章节。

17.4.2.1 标准点乘模式

标准点乘模式计算公式如下：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据： P_x 、 P_y 和 k 对应的内存块为 `ECC_MULT_Mem_Px`、 `ECC_MULT_Mem_Py` 和 `ECC_MULT_Mem_k`。

- 输出数据: Q_x 和 Q_y 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。

17.4.2.2 标准点验证模式

标准点验证模式用于计算点 (P_x, P_y) 是否在选定的椭圆曲线上。

- 输入数据: P_x 和 P_y 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
- 输出数据: 点验证的结果存储在 `ECC_MULT_VERIFICATION_RESULT` 中。

17.4.2.3 标准点验证 + 标准点乘模式

标准点验证 + 标准点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上, 如果其在选定的椭圆曲线上, 则继续计算如下公式:

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中,

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据: P_x 、 P_y 和 k 对应的内存块为 `ECC_MULT_Mem_Px`、`ECC_MULT_Mem_Py` 和 `ECC_MULT_Mem_k`。
- 输出数据:
 - 点验证的结果存储在 `ECC_MULT_VERIFICATION_RESULT` 中。
 - Q_x 和 Q_y 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - J_x 、 J_y 和 J_z 对应的内存块为 `ECC_MULT_Mem_Qx`、`ECC_MULT_Mem_Qy` 和 `ECC_MULT_Mem_Qz`。

17.4.2.4 Jacobian 点乘模式

Jacobian 点乘模式计算公式如下:

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中,

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全, 不需要输入。
- 输入数据: P_x 、 P_y 和 k 对应的内存块为 `ECC_MULT_Mem_Px`、`ECC_MULT_Mem_Py` 和 `ECC_MULT_Mem_k`。
- 输出数据: Q_x 、 Q_y 和 Q_z 对应的内存块为 `ECC_MULT_Mem_Qx`、`ECC_MULT_Mem_Qy` 和 `ECC_MULT_Mem_Qz`。

17.4.2.5 点加模式

点加模式计算公式如下:

$$R = (R_x, R_y) = (J_x, J_y, J_z) = (P_x, P_y, 1) + (Q_x, Q_y, Q_z)$$

其中,

- (R_x, R_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：
 - P_x 和 P_y 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - Q_x 、 Q_y 和 Q_z 对应的内存块为 `ECC_MULT_Mem_Qx`、`ECC_MULT_Mem_Qy` 和 `ECC_MULT_Mem_Qz`。
- 输出数据：
 - R_x 和 R_y 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - J_x 、 J_y 和 J_z 对应的内存块为 `ECC_MULT_Mem_Qx`、`ECC_MULT_Mem_Qy` 和 `ECC_MULT_Mem_Qz`。

17.4.2.6 Jacobian 点验证模式

Jacobian 点验证模式用于计算点 (Q_x, Q_y, Q_z) 是否在选定的椭圆曲线上。

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入数据： Q_x 、 Q_y 和 Q_z 对应的内存块为 `ECC_MULT_Mem_Qx`、`ECC_MULT_Mem_Qy` 和 `ECC_MULT_Mem_Qz`。
- 输出数据：点验证的结果存储在 `ECC_MULT_VERIFICATION_RESULT` 中。

17.4.2.7 标准点验证 + Jacobian 点乘模式

标准点验证 + Jacobian 点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据： P_x 、 P_y 和 k 对应的内存块为 `ECC_MULT_Mem_Px`、`ECC_MULT_Mem_Py` 和 `ECC_MULT_Mem_k`。
- 输出数据：
 - 点验证的结果存储在 `ECC_MULT_VERIFICATION_RESULT` 中。
 - Q_x 、 Q_y 和 Q_z 对应的内存块为 `ECC_MULT_Mem_Qx`、`ECC_MULT_Mem_Qy` 和 `ECC_MULT_Mem_Qz`。

17.4.2.8 模加模式

模加模式计算公式如下：

$$R = A + B \bmod N$$

其中，

- 输入数据：
 - A 和 B 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - N 的值可通过以下寄存器字段配置：
 - * 配置 `ECC_MULT_CURVE_MODE` 来选择曲线。
 - * 配置 `ECC_MULT_MOD_BASE` 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 `ECC_MULT_Mem_Px`。

17.4.2.9 模减模式

模减模式计算公式如下：

$$R = A - B \bmod N$$

其中，

- 输入数据：
 - A 和 B 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - N 的值可通过以下寄存器字段配置：
 - * 配置 `ECC_MULT_CURVE_MODE` 来选择曲线。
 - * 配置 `ECC_MULT_MOD_BASE` 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 `ECC_MULT_Mem_Px`。

17.4.2.10 模乘模式

模乘模式计算公式如下：

$$R = A \cdot B \bmod N$$

其中，

- 输入数据：
 - A 和 B 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - N 的值可通过以下寄存器字段配置：
 - * 配置 `ECC_MULT_CURVE_MODE` 来选择曲线。
 - * 配置 `ECC_MULT_MOD_BASE` 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 `ECC_MULT_Mem_Py`。

17.4.2.11 模除模式

模除模式计算公式如下：

$$R = A \cdot B^{-1} \bmod N$$

其中，

- 输入数据：
 - A 和 B 对应的内存块为 `ECC_MULT_Mem_Px` 和 `ECC_MULT_Mem_Py`。
 - N 的值可通过以下寄存器字段配置：

- * 配置 `ECC_MULT_CURVE_MODE` 来选择曲线。
- * 配置 `ECC_MULT_MOD_BASE` 来选择使用曲线模数或基点阶数。
- 输出数据：R 对应的内存块为 `ECC_MULT_Mem_Py`。

17.4.3 提升抗攻击性能

在执行不同类型的点乘计算（即模式 0、3、4 和 7）时，ESP32-C5 的 ECC 加速器提供了一个选项，可保证所有点乘运算都维持相同的最长运行时间，从而进一步提升其抗攻击性能。

要启用强抗攻击性能，请将 `ECC_MULT_SECURITY_MODE` 置 1。启用此设置后，ECC 加速器在每次进行点乘计算时：

- 都会消耗相同的时间；
- 都会消耗相同的功耗。

17.4.4 时钟

ECC 模块有两种时钟：

- `clk_ecc`：ECC 模块运行所使用的功能时钟
- `clk_apb_ecc`：用于配置 ECC 模块寄存器的总线时钟

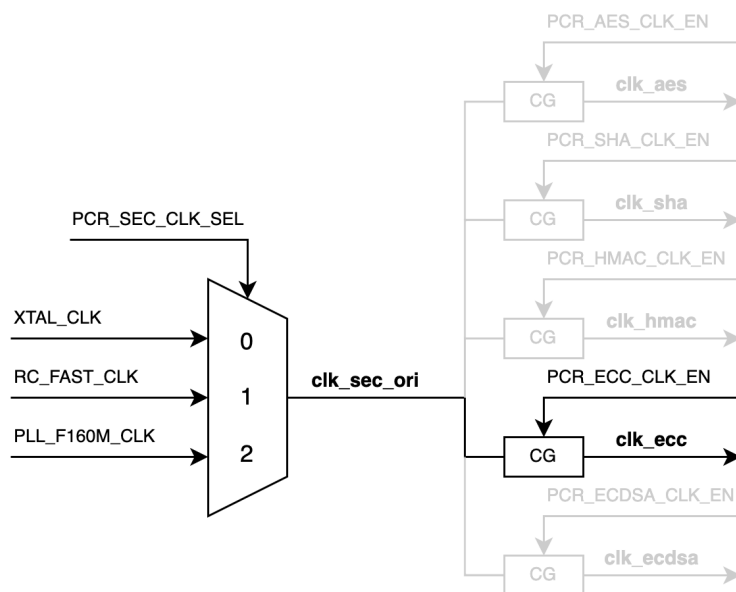


图 17.4-1. `clk_ecc` 时钟图

如图 17.4-1 `clk_ecc` 时钟图 所示，安全功能源时钟 `clk_sec_ori` 有以下时钟源：

- `XTAL_CLK`
- `RC_FAST_CLK`
- `PLL_F160M_CLK`

ECC 模块功能时钟 `clk_ecc` 由时钟门控寄存器 `PCR_ECC_CLK_EN` 控制，当该寄存器值为 1 时，时钟门控开启，使能 `clk_ecc`。

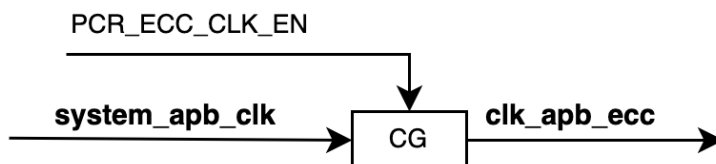


图 17.4-2. clk_apb_ecc 时钟图

如图 17.4-2 *clk_apb_ecc 时钟图* 所示，ECC 模块配置时钟 clk_apb_ecc 的时钟源为 SYSTEM_APB_CLK。

ECC 模块配置时钟 clk_apb_ecc 由时钟门控寄存器 PCR_ECC_CLK_EN 控制，当该寄存器值为 1 时，时钟门控开启，使能 clk_apb_ecc。

有关时钟的更多信息，请查看章节 6 *复位和时钟*。

17.4.5 复位

ECC 模块支持两种复位方式：

- 自身完全复位：向 PCR_ECC_RST_EN 先写 1 后写 0 使能，可复位整个 ECC 模块。
- 密码学保护复位：向 PCR_ECDSA_RST_EN 先写 1 后写 0 使能，可复位整个 ECC 模块。

说明：

由于密码学加速器之间的资源复用，ECDSA 模块在运算时会复用 ECC 模块中的资源。为了保护 ECDSA 模块的运算数据，在 ECDSA 模块复位时，也会复位 ECC 模块。

17.5 中断

ESP32-C5 ECC 硬件加速器可产生一个中断信号 ECC_INTR，并将其发送给 *中断矩阵*。

中断信号 ECC_INTR 由 ECC 硬件加速器的内部中断源 ECC_MULT_CALC_DONE_INT 生成。ECC 硬件加速器运算完成即触发该中断源。

每个中断源都有一组通用的配置寄存器，详见章节 *中断配置寄存器*。中断源 ECC_MULT_CALC_DONE_INT 由以下寄存器位控制（更多信息请参考 17.7 *寄存器列表*）：

- ECC_MULT_CALC_DONE_INT_RAW：ECC 硬件加速器运算完成时置 1。
- ECC_MULT_CALC_DONE_INT_ST：表示 ECC 硬件加速器运算完成中断的状态，用 ECC_MULT_CALC_DONE_INT_ENA 使能或屏蔽 ECC_MULT_CALC_DONE_INT_RAW 来生成该位。
- ECC_MULT_CALC_DONE_INT_ENA：用于使能或屏蔽 ECC 硬件加速器运算完成中断状态位。
- ECC_MULT_CALC_DONE_INT_CLR：置 1 此位清除 ECC 硬件加速器运算完成中断，对应的 ECC_MULT_CALC_DONE_INT_RAW 和 ECC_MULT_CALC_DONE_INT_ST 位会清零。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 *中断矩阵 [to be added later]* > 3.1 *中断术语*。

17.6 配置流程

软件配置 ECC 硬件加速器的流程如下：

1. 配置 ECC 硬件加速器的时钟与复位。配置细节请参考小节 [17.4.4](#) 和 [17.4.5](#)。
2. 根据 [17.4](#) 小节的描述，按照需求配置 ECC 加速器密钥长度模式和工作模式。
3. 根据 [17.5](#) 小节的描述，使能 `ECC_MULT_CALC_DONE_INT` 中断。
4. 置位 `ECC_MULT_START` 以启动 ECC 运算。
5. 等待 `ECC_MULT_CALC_DONE_INT` 中断的产生，即 ECC 运算结束。
6. 根据 [17.4](#) 小节的描述，查看运算结果。

17.7 寄存器列表

本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 [系统和存储器](#) 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
中断寄存器			
ECC_MULT_INT_RAW_REG	原始中断状态寄存器	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	中断屏蔽状态寄存器	0x0010	RO
ECC_MULT_INT_ENA_REG	中断使能寄存器	0x0014	R/W
ECC_MULT_INT_CLR_REG	中断清除寄存器	0x0018	WT
配置寄存器			
ECC_MULT_CONF_REG	ECC 加速器配置寄存器	0x001C	varies
版本寄存器			
ECC_MULT_DATE_REG	版本控制寄存器	0x00FC	R/W

17.8 寄存器

本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 17.1. ECC_MULT_INT_RAW_REG (0x000C)

(reserved)																														ECC_MULT_CALC_DONE_INT_RAW	
31																														1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

ECC_MULT_CALC_DONE_INT_RAW ECC_MULT_CALC_DONE_INT 的原始中断状态。(R/SS/WTC)

Register 17.2. ECC_MULT_INT_ST_REG (0x0010)

(reserved)																														ECC_MULT_CALC_DONE_INT_ST	
31																														1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

ECC_MULT_CALC_DONE_INT_ST ECC_MULT_CALC_DONE_INT 的屏蔽中断状态。(RO)

Register 17.3. ECC_MULT_INT_ENA_REG (0x0014)

(reserved)																														ECC_MULT_CALC_DONE_INT_ENA	
31																														1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

ECC_MULT_CALC_DONE_INT_ENA 写 1 使能 ECC_MULT_CALC_DONE_INT 中断。(R/W)

Register 17.4. ECC_MULT_INT_CLR_REG (0x0018)

Diagram of the ECC_MULT_DONE_INT_CLR register structure. The register is 32 bits wide, with bit 31 labeled '31' and bit 0 labeled '0'. The entire register is labeled '(reserved)'. The bit 0 is also labeled 'Reset'.

ECC_MULT_CALC_DONE_INT_CLR 写1清除**ECC_MULT_CALC_DONE_INT**中断。(WT)

Register 17.5. ECC_MULT_CONF_REG (0x001C)

ECC_MULT_MEM_CLOCK_GATE_FORCE_ON																												(reserved)																												ECC_MULT_SECURITY_MODE																ECC_MULT_WORK_MODE																ECC_MULT_MOD_BASE																ECC_MULT_CURVE_MODE																ECC_MULT_RESET																ECC_MULT_START																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
ECC_MULT_CLK_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																</			

ECC_MULT_START 配置是否启动 ECC 加速器。此位运算结束后自动清 0。

- 0: 无效
- 1: 启动 ECC 加速器

(R/W/SC)

ECC_MULT_RESET 配置是否复位 ECC 加速器。

- 0: 无效
- 1: 复位

(WT)

ECC_MULT_CURVE_MODE 配置 ECC 加速器的曲线模式。

- 0: P-192
- 1: P-256
- 2: P-384
- 3: 无效

(R/W)

ECC_MULT_MOD_BASE 配置在模运算操作中选择使用曲线模数或基点阶数。仅在工作模式 8-11 中有效。

- 0: n（基点阶数）
- 1: p（曲线模数）

(R/W)

见下页...

Register 17.5. ECC_MULT_CONF_REG (0x001C)

[接上页...](#)

ECC_MULT_WORK_MODE 配置 ECC 加速器的工作模式。

- 0: 标准点乘
 - 1: 保留项, 不可用
 - 2: 标准点验证
 - 3: 标准点验证 + 标准点乘
 - 4: Jacobian 点乘
 - 5: 点加
 - 6: Jacobian 点验证
 - 7: 标准点验证 + Jacobian 点乘
 - 8: 模加
 - 9: 模减
 - 10: 模乘
 - 11: 模除
- (R/W)

ECC_MULT_SECURITY_MODE 配置是否针对点乘计算启用高安全性模式。仅在工作模式 0、3、4 和 7 中有效。

- 0: 禁用
 - 1: 启用
- (R/W)

ECC_MULT_VERIFICATION_RESULT 表示 ECC 加速器的验证结果, 仅在运算完成时有效。

- 0: 验证失败
 - 1: 验证成功
- (R/SS)

ECC_MULT_CLK_EN 配置是否强制打开寄存器门控。

- 0: 无效
 - 1: 强制打开
- (R/W)

ECC_MULT_MEM_CLOCK_GATE_FORCE_ON 配置是否强制打开 ECC 内存时钟门控。

- 0: 无效
 - 1: 强制打开
- (R/W)

Register 17.6. ECC_MULT_DATE_REG (0x00FC)

(reserved)				ECC_MULT_DATE																										
31	28	27	0																											
0	0	0	0	0x2408120																										Reset

ECC_MULT_DATE 版本控制寄存器。(R/W)

第 18 章

HMAC 加速器 (HMAC)

18.1 概述

HMAC 模块通过 hash 算法 SHA-256 和密钥计算得到数据信息的信息认证码 (MAC)，采用 RFC 2104 标准。长度为 256 位的 HMAC 密钥存储在 eFuse 的密钥块中，可配置为读保护，即用户无法直接从 HMAC 加速器外部访问密钥。

18.2 主要特性

- 使用标准 HMAC-SHA-256 算法
- 基于 eFuse 密钥的 HMAC-SHA-256 算法计算，
 - 为提高安全性，软件在下行模式下无法访问其计算结果
 - 软件在上行模式下可访问其计算结果
- 下行模式下，支持生成数字签名算法 (DSA) 外设所需的密钥
- 下行模式下，支持重启软禁用的 JTAG

18.3 功能描述

HMAC 模块有两种工作模式，即上行模式和下行模式。上行模式中，由用户提供 HMAC 信息并回读计算结果；下行模式中，HMAC 模块为其他两个可能的内部硬件模块提供输入：一方面，如 JTAG 被暂时禁用，HMAC 可使用 JTAG；另一方面，HMAC 可用作存储在 DSA 外设 flash 中数字签名参数的解密密钥。此外，这些计算在下行模式下自动进行，只要配置正确，就可以确保密钥及派生密钥材料的保密性。

说明：

芯片释放复位信号后，HMAC 模块将检查 eFuse 中是否烧写有 DSA 模块所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行数字签名算法模式，完成相应密钥计算。

18.3.1 上行模式

在上行模式下计算 HMAC 值，用户应执行以下步骤：

1. 初始化 HMAC 模块，进入上行模式。
2. 将正确填充的信息写入 HMAC 中，一次写入一个块。
3. 从 HMAC 回读计算结果。

有关此过程的详细步骤，可参见章节 18.5。

说明：

使用上行模式的较多为支持 HMAC-SHA-256 算法的挑战-应答协议。假设挑战-应答协议中的通讯双方分别称为 A 和 B，想要交换的数据信息为 M，则协议的一般验证流程为：

- A 计算出一个特殊的随机数 M 。
- A 将 M 发送给 B。
- B（通过 M 和密钥）计算 HMAC 结果，并将结果发送给 A。
- A（通过 M 和密钥）内部计算 HMAC 结果。
- A 比较两次计算结果。如结果相同，则验证通过 B 的身份。

18.3.2 下行模式 - JTAG 启动功能

JTAG 可被 eFuse 暂时关闭，关闭后，用户可使用 HMAC 模块重启 JTAG 功能。（详细步骤可参见章节 2 [eFuse 控制器 \(eFuse\) \[to be added later\]](#)。）该模式下，用户应提供某 eFuse 密钥的 HMAC 计算结果，HMAC 模块将检查用户提供的数值与计算的数值是否匹配。如果二者相同，JTAG 将被启动，直到用户再次调用 HMAC 模块清除计算结果并关闭 JTAG。

用户可执行以下步骤重启 JTAG：

1. 通过初始化时钟和 HMAC 复位信号启动 HMAC 模块，并通过配置 [HMAC_SET_PARA_PURPOSE_REG](#) 位进入下行 JTAG 启动模式，等待计算完成。详细步骤可参见章节 18.5。
2. 将 1 写入 [HMAC_SOFT_JTAG_CTRL_REG](#) 寄存器，进入 JTAG 重启模式。
3. 将 256 位的 HMAC 值写入寄存器 [HMAC_WR_JTAG_REG](#)。该值是预先在本地使用 SHA-256 和已被写入 eFuse 的密钥对 32 字节的 0x00 进行 HMAC 计算得到的。需以大端序分 8 次写入，每次 32 位。
4. 如果由 eFuse 中密钥计算的 HMAC 结果与用户在步骤 3 中写入的数值一致，则 JTAG 重启。否则，JTAG 保持关闭状态。
5. 在用户将 1 写入寄存器 [HMAC_SET_INVALIDATE_JTAG_REG](#) 或复位重启之后，JTAG 将保持关闭状态。如需再次重启 JTAG，请重复上述步骤。

18.3.3 下行模式 - 数字签名算法密钥派生功能

数字签名算法 (DSA) 模块使用 AES-CBC 算法加密其参数。HMAC 模块作为密钥导出函数 (KDF) 导出解密上述参数的 AES 密钥。

在启用 DSA 模块之前，用户必须先通过 HMAC 模块计算得到 DSA 模块工作时所需的密钥。详细信息可参见章节 21 [数字签名算法 \(DSA\)](#)。芯片上电后，HMAC 模块将检查 eFuse 中是否烧写有 DSA 模块所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行数字签名算法模式，完成相应密钥计算。

18.4 HMAC eFuse 配置

每个烧写入 eFuse 密钥块的 HMAC 密钥都有其功能，指定该密钥用于实现哪一功能。HMAC 中，未配置匹配的功能数值的密钥无法执行计算。当前 HMAC 模块共支持三种功能：下行模式下的 JTAG 重启功能和 DSA 密钥导

出功能，以及上行模式下的 HMAC 计算功能。表 18.4-1 列出了上述各项功能，以及一种可指定密钥同时用于 JTAG 重启和 DSA 密钥导出的功能。

启动 HMAC 模块进行计算之前，用户应先读取章节 2 *eFuse 控制器 (eFuse) [to be added later]* 中 EFUSE_KEY_PURPOSE_x 的值（基于 eFuse 中共存在 6 个密钥，故 x 的值为 0 ~ 5，其中 EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 属于寄存器 EFUSE_RD_REPEAT_DATA1_REG，EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 属于寄存器 EFUSE_RD_REPEAT_DATA2_REG），检查 eFuse 中是否已烧写密钥。例如，在上行模式下，如果 EFUSE_KEY_PURPOSE_0~5 中没有 EFUSE_KEY_PURPOSE_HMAC_UP，则说明 eFuse 中没有上行模式可用的密钥。此时，可按照下述步骤将密钥烧写入 eFuse 中：

- 1. 准备一个 256 位 HMAC 密钥，将其烧写到空的 eFuse 密钥块 *y* 中。eFuse 中共有 6 个密钥块，分别为 4 ~ 9，因此 *y* 的值为 4 ~ 9。也就是说，若密钥为 key0，则对应的密钥块为 block4。接着，将功能对应数值烧写至 EFUSE_KEY_PURPOSE_(*y* - 4)。例如在上行模式下，烧写密钥后，应将 EFUSE_KEY_PURPOSE_HMAC_UP（对应值为 8）写入 EFUSE_KEY_PURPOSE_(*y* - 4)。更多烧写 eFuse 密钥的详细信息，可参见章节 2 *eFuse 控制器 (eFuse) [to be added later]*。
- 2. 如有需要，配置 eFuse 密钥块读保护功能，使用户无法读取密钥值。用户应保留步骤中生成的随机密钥副本以便验证。

请注意，功能为 EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL 的密钥既可用于 JTAG 重启，也可用于 DSA 密钥导出。

表 18.4-1. HMAC 功能及配置数值

功能 (<i>m</i>)	模式	数值	描述
JTAG 重启	下行模式	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
DSA 密钥导出	下行模式	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC 计算	上行模式	8	EFUSE_KEY_PURPOSE_HMAC_UP
JTAG 重启和 DSA 密钥导出	下行模式	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

选取 eFuse 的密钥块与 HMAC 功能

eFuse 控制器共有 6 个密钥块，即 KEY0 ~ 5。用户将编号 *n* 写入寄存器 HMAC_SET_PARA_KEY_REG，表示选择 KEY*n* 作为本次 HMAC 模块运行时使用的密钥。

将所使用功能对应的数值写入寄存器 HMAC_SET_PARA_PURPOSE_REG 完成配置（可参见章节 18.5）。需要注意的是，eFuse 存储器中的密钥在烧写时都定义了功能用途，只有当 HMAC 的配置功能与 KEY*n* 定义的功能用途相匹配时，HMAC 模块才会执行配置好的计算任务。否则，HMAC 模块将返回匹配错误结果并结束当前计算任务。

比如，如果用户选择了 KEY3 作为本次计算的密钥，且烧写入 EFUSE_KEY_PURPOSE_3 中的数值为 6，即功能 EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG。参照表 18.4-1 可知，KEY3 是用于 JTAG 重启的密钥。如果此时寄存器 HMAC_SET_PARA_PURPOSE_REG 中配置的数值也是 6，HMAC 模块才会启动 JTAG 重启功能的计算。

从密钥管理器中选取密钥

用户可以在密钥管理器中部署 HMAC 的密钥。要选择密钥管理器中的该密钥，请将密钥编号 7 写入注册表 `HMAC_SET_PARA_KEY_REG`。用户从密钥管理器中选取密钥时，注册表中 HMAC 操作的预定目标会自动设置为“HMAC Calculation”，其他设置都不会生效。

18.5 配置流程

用户调用 ESP32-C5 中 HMAC 流程如下：

18.5.1 启动 HMAC 模块

1. 置位寄存器 `HP_SYS_CLKRST_REG_CRYPTO_HMAC_CLK_EN` 中 HMAC 和 SHA 的外设时钟位，清除寄存器 `HP_SYS_CLKRST_REG_RST_EN_HMAC` 中相应的外设复位位。关于如何配置时钟和重启信号，请参见章节 6 复位和时钟。
2. 将 1 写入寄存器 `HMAC_SET_START_REG`。

18.5.2 配置 HMAC 密钥和密钥功能

1. 将密钥功能 m 写入寄存器 `HMAC_SET_PARA_PURPOSE_REG`。表 18.4-1 列出了 m 对应的数值，可参见章节 18.4。
2. 将数值 n 写入寄存器 `HMAC_SET_PARA_KEY_REG`，选择 eFuse 存储器中的 `KEY $n$` 作为本次计算的密钥 (n 的取值范围为 0 ~ 5, 7)，可参见章节 18.4。
3. 将 1 写入寄存器 `HMAC_SET_PARA_FINISH_REG`，完成配置。
4. 读取寄存器 `HMAC_QUERY_ERROR_REG`。如果返回值为 1，表明选取的密钥块与配置的密钥功能不匹配，本次计算任务中止。如果返回值为 0，表明选取的密钥块与配置的密钥功能匹配，可以执行计算流程。
5. 如果 `HMAC_SET_PARA_PURPOSE_REG` 的数值不为 8，表明 HMAC 模块当前处于下行模式，请参考章节 18.5.3 进行下一步配置。如果数值为 8，表明 HMAC 模块当前处于上行模式，请参考章节 18.5.4 进行下一步配置。

18.5.3 下行模式流程

1. 轮询状态寄存器 `HMAC_QUERY_BUSY_REG`，当读取到该寄存器的值为 0 时，继续下一步骤。
2. 如需清除计算结果，并确保后续不再使用相关硬件 (JTAG 或 DSA)，用户可将 1 写入寄存器 `HMAC_SET_INVALIDATE_JTAG_REG`，清除 JTAG 密钥生成的结果；也可将 1 写入寄存器 `HMAC_SET_INVALIDATE_DS_REG`，清除数字签名算法密钥生成的结果。此后，如需重启 JTAG 或 DSA 模块，需重新进行 HMAC 调用流程。

18.5.4 上行模式流程

1. 上行模式下传输数据：
 - (a) 轮询状态寄存器 `HMAC_QUERY_BUSY_REG`，当读取到该寄存器的值为 0 时，继续下一步骤。
 - (b) 用户根据章节 18.6.1 的描述对数据进行 SHA 附加填充并获取数据块 `Block $n$` ($n \geq 0$)
 - (c) 根据数据块 n 的数量，选择其中一种操作：

- 如果 Block_ *n* 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位：
 - i. 将 512 位的数据块 Block_ *n* 写入寄存器 HMAC_WR_MESSAGE_ *n* _REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
 - ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
 - iii. 写 1 到寄存器 HMAC_ONE_BLOCK_REG 中，完成数据传输。
- 如果 Block_ *n* 是最后一个填充数据块：
 - i. 将 512 位的数据块 Block_0 写入寄存器 HMAC_WR_MESSAGE_ *n* _REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
 - ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
 - iii. 自动完成数据传输。
- 如果 Block_ *n* 是倒数第二个填充数据块：
 - i. 将 512 位的数据块 Block_ *n* 写入寄存器 HMAC_WR_MESSAGE_ *n* _REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
 - ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
 - iii. 写 1 到寄存器 HMAC_SET_MESSAGE_PAD_REG 中，随后跳转到步骤 c 继续数据传输。
- 如果 Block_ *n* 既不是最后一个也不是倒数第二个数据块：
 - i. 将 512 位的数据块 Block_ *n* 写入寄存器 HMAC_WR_MESSAGE_ *n* _REG (*n*: 0-15) 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG 中，HMAC 模块将计算该数据块。
 - ii. 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
 - iii. 写 1 到寄存器 HMAC_SET_MESSAGE_ING_REG 中，随后跳转到步骤 c 继续数据传输。

2. 读取上行模式下的结果 hash 值：

- (a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。
- (b) 从寄存器 HMAC_RD_RESULT_ *n* _REG (*n*: 0-7) 中读取 hash 结果数值。
- (c) 将 1 写入寄存器 HMAC_SET_RESULT_FINISH_REG，结束当次计算，并同时清除计算结果。
- (d) 上行模式下的操作完成。

说明：

DSA 模块和 HMAC 模块可直接调用或在内部使用 SHA 加速器，但两者不能同时共享硬件资源。因此在 HMAC 模块运行过程中，SHA 模块无法被 CPU 和 DSA 模块调用。

18.6 HMAC 算法细节

18.6.1 附加填充位

HMAC 模块中采用 SHA-256 作为加密 hash 算法。该算法中，若待输入数据的总位数不是 512 的倍数，用户须在软件中应用 SHA-256 附加填充算法。SHA-256 附加填充算法与 [FIPS PUB 180-4](#) 中章节 *Padding the*

Message 所描述相同。下行模式下，用户无需输入数据或进行数据填充，HMAC 模块默认使用 32 字节 0x00 方式重启 JTAG，以及 32-byte 0xff 方式生成 DSA 模块的 AES 密钥。

为方便阅读，以下将简要介绍数据填充的步骤。如图 18.6-1 所示，假设待处理数据长度为 m 个位，填充步骤如下：

- 1. 在待处理数据末尾附加一个“1”。
- 2. 附加 k 个“0”。其中， k 为满足 $m + 1 + k \equiv 448(mod512)$ 的最小非负数。
- 3. 附加一个 64 位的整数值作为二进制块。该二进制块的内容为待填充数据作为一个大端二进制整数值 m 的长度。

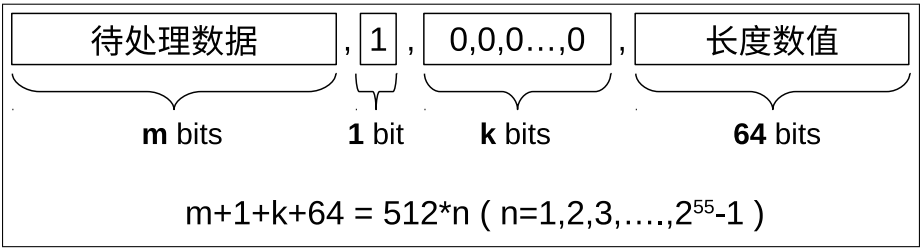


图 18.6-1. HMAC 附加填充位示意图

上行模式下，若待填充数据总位数是 512 的整数倍，则用户可通过将 1 写入 `HMAC_SET_MESSAGE_END_REG` 配置由硬件完成 SHA 附加填充操作，也可通过将 1 写入 `HMAC_SET_MESSAGE_PAD_REG` 自行完成填充操作。若待填充数据总位数不是 512 的整数倍，则用户只能自行完成 SHA 附加填充操作。数据填充操作完成后，用户应参照章节 18.5 完成后续配置。

18.6.2 HMAC 算法结构

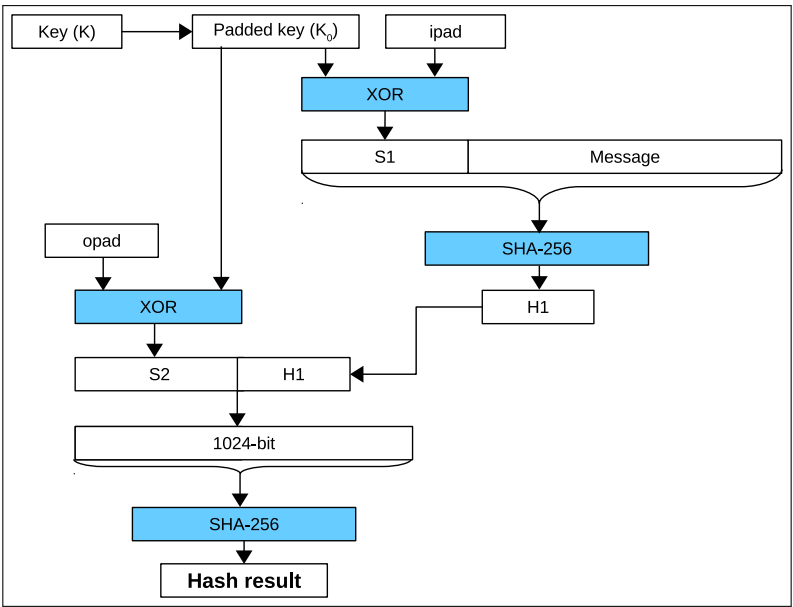


图 18.6-2. HMAC 结构示意图

HMAC 模块中应用的算法结构示意图如 18.6-2 所示。这是 RFC 2104 中描述的标准 HMAC 算法。

图 18.6-2 中，

- 1. ipad 是由 64 个 0x36 字节组成的 512 位数据块。

2. opad 是由 64 个 0x5c 字节组成的 512 位数据块。

首先，HMAC 模块在 256 位的密钥 K 的位序列后附加上 256 位的 0 序列，得到 512 位的 K_0 。再对 K_0 和 ipad 进行异或运算，得到 512 位的 S1。将总位数为 512 倍数的待输入数据附加到 512 位的 S1 数值后，使用 SHA-256 加密算法计算得到 256 位的 H1。

HMAC 模块通过对 K_0 和 opad 进行异或运算得到 S2，将 256 位的 hash 计算结果附加到 512 位的 S2 数值后，得到 768 位长度的序列，使用章节 18.6.1 中描述的 SHA 附加填充算法将该序列填充成 1024 位的序列，最后使用 SHA-256 加密算法计算得到的最终 hash 结果（256 位）。

18.7 寄存器列表

本小节的所有地址均为相对于 HMAC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

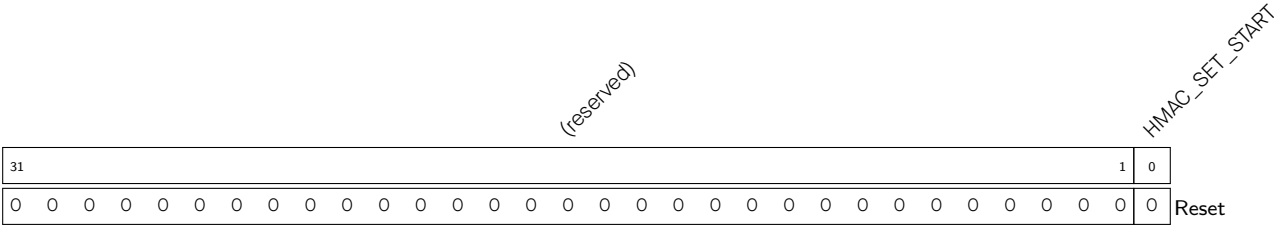
名称	描述	地址	访问
控制/状态寄存器			
HMAC_SET_START_REG	HMAC 开始控制寄存器	0x0040	WS
HMAC_SET_PARA_FINISH_REG	HMAC 配置完成寄存器	0x004C	WS
HMAC_SET_MESSAGE_ONE_REG	HMAC 信息控制寄存器	0x0050	WS
HMAC_SET_MESSAGE_ING_REG	HMAC 信息继续寄存器	0x0054	WS
HMAC_SET_MESSAGE_END_REG	HMAC 信息终止寄存器	0x0058	WS
HMAC_SET_RESULT_FINISH_REG	HMAC 读取结果完成寄存器	0x005C	WS
HMAC_SET_INVALIDATE_JTAG_REG	注销 JTAG 结果寄存器	0x0060	WS
HMAC_SET_INVALIDATE_DS_REG	注销数字签名结果寄存器	0x0064	WS
HMAC_QUERY_ERROR_REG	存储用户配置的密钥和功能的配对结果	0x0068	RO
HMAC_QUERY_BUSY_REG	存储 HMAC 模块的忙碌状态	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	软件填充寄存器	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block 信息寄存器	0x00F4	WS
配置寄存器			
HMAC_SET_PARA_PURPOSE_REG	HMAC 参数配置寄存器	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC 密钥配置寄存器	0x0048	WO
HMAC_WR_JTAG_REG	重启 JTAG 寄存器 1	0x00FC	WO
HMAC_SOFT_JTAG_CTRL_REG	重启 JTAG 寄存器 0	0x00F8	WS
版本寄存器			
HMAC_DATE_REG	版本控制寄存器	0x01FC	R/W

18.8 寄存器

本小节的所有地址均为相对于 HMAC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 18.1. HMAC_SET_START_REG (0x0040)

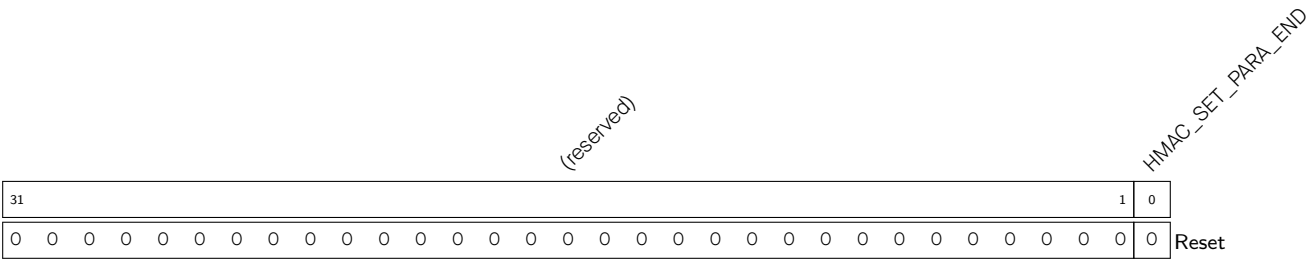


HMAC_SET_START 配置是否启动 HMAC。

- 0: 关闭 HMAC
- 1: 启动 HMAC

(WS)

Register 18.2. HMAC_SET_PARA_FINISH_REG (0x004C)

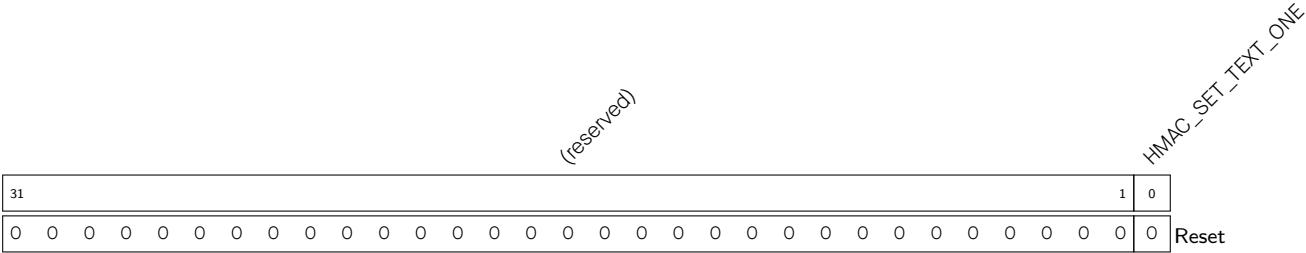


HMAC_SET_PARA_END 配置是否完成 HMAC 配置。

- 0: 无效
- 1: 完成配置

(WS)

Register 18.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



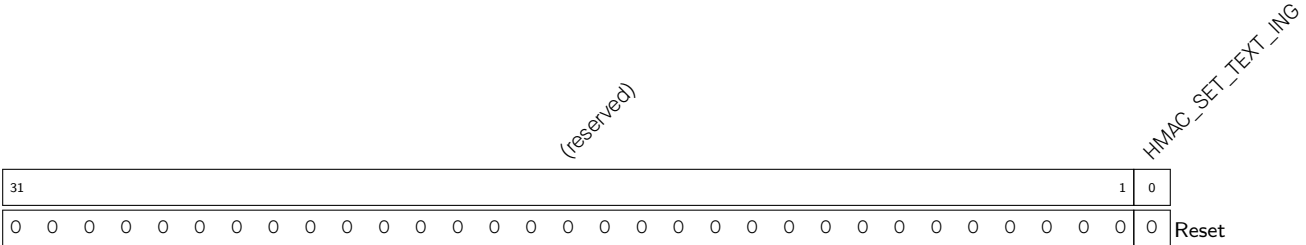
HMAC_SET_TEXT_ONE 调用 SHA 计算一个数据块。

0: 无效

1: 调用

(WS)

Register 18.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



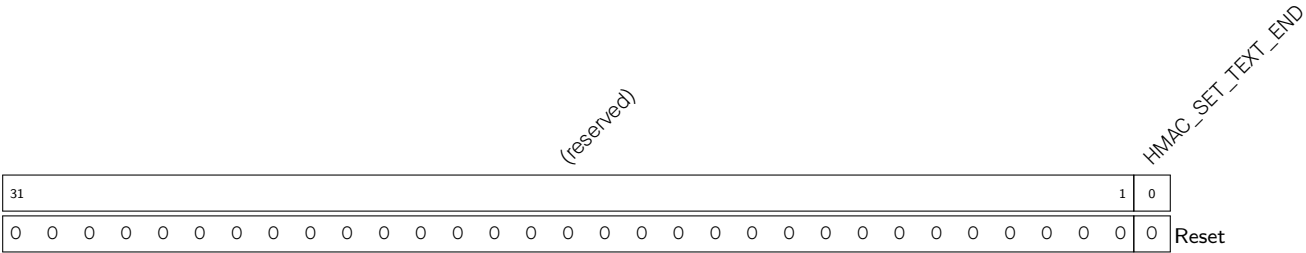
HMAC_SET_TEXT_ING 配置是否存在未处理的数据块。

0: 不存在

1: 存在

(WS)

Register 18.5. HMAC_SET_MESSAGE_END_REG (0x0058)



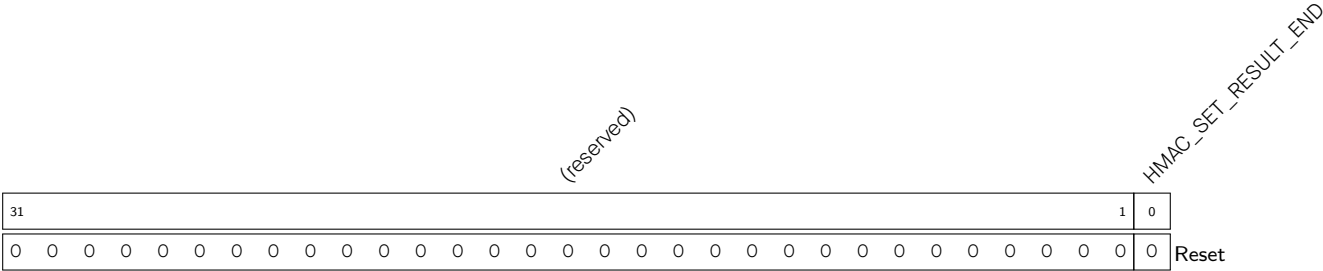
HMAC_SET_TEXT_END 配置是否开始硬件填充。

0: 无效

1: 开始硬件填充

(WS)

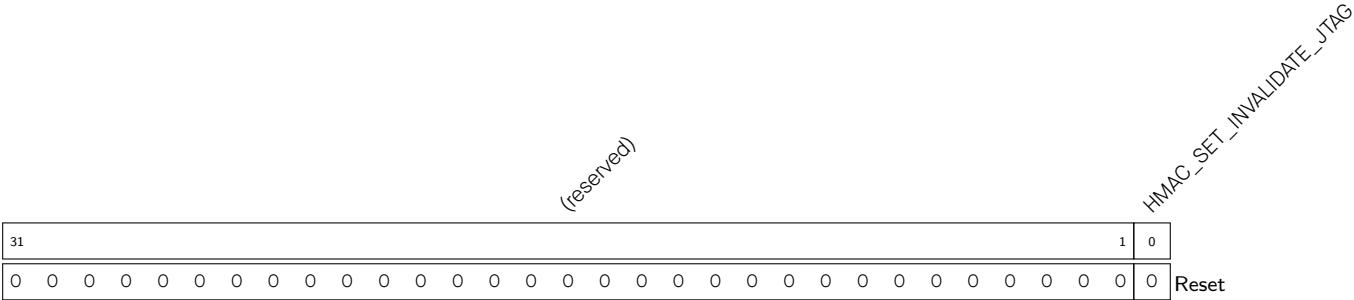
Register 18.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



HMAC_SET_RESULT_END 配置是否退出上行模式并清空计算结果。

- 0: 无效
- 1: 退出上行模式并清空计算结果 (WS)

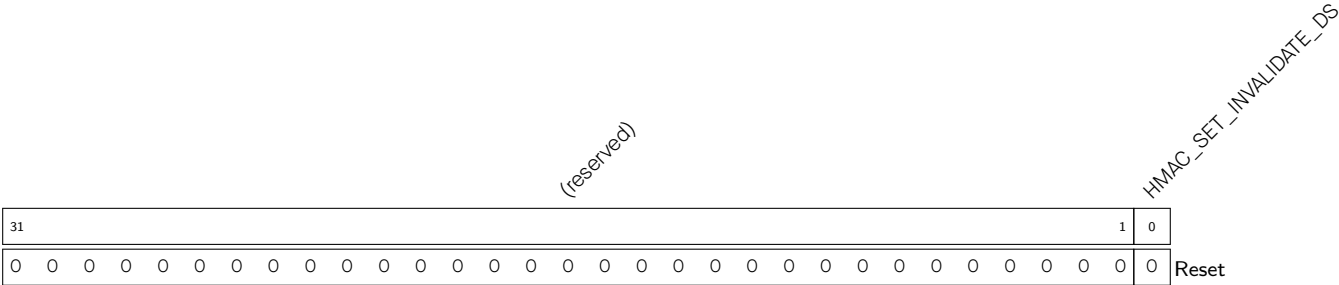
Register 18.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



HMAC_SET_INVALIDATE_JTAG 配置是否清空下行模式下 JTAG 重启功能的计算结果。

- 0: 无效
- 1: 清空计算结果 (WS)

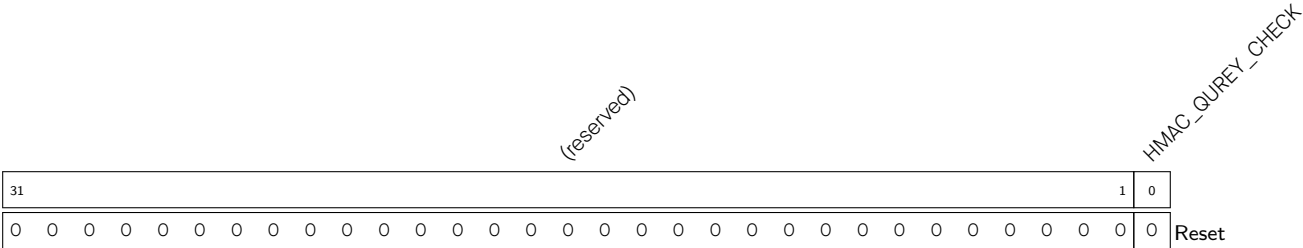
Register 18.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



HMAC_SET_INVALIDATE_DS 配置是否清空下行模式下 DSA 功能的计算结果。

- 0: 无效
- 1: 清空计算结果 (WS)

Register 18.9. HMAC_QUERY_ERROR_REG (0x0068)

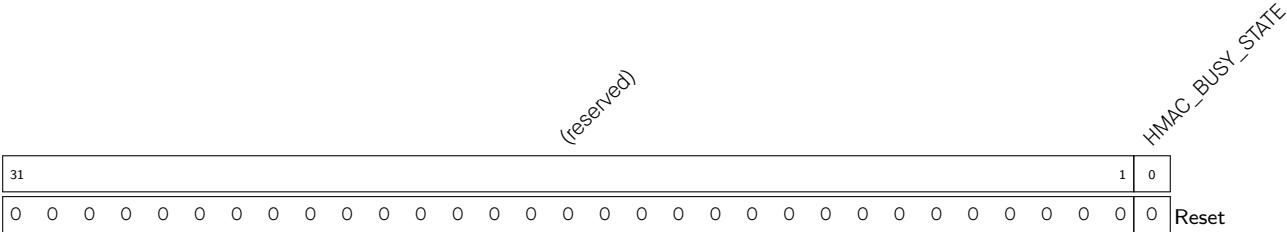


HMAC_QUREY_CHECK 表示 HMAC 密钥与功能是否匹配。

- 0: HMAC 密钥和功能匹配。
- 1: 不匹配

(RO)

Register 18.10. HMAC_QUERY_BUSY_REG (0x006C)

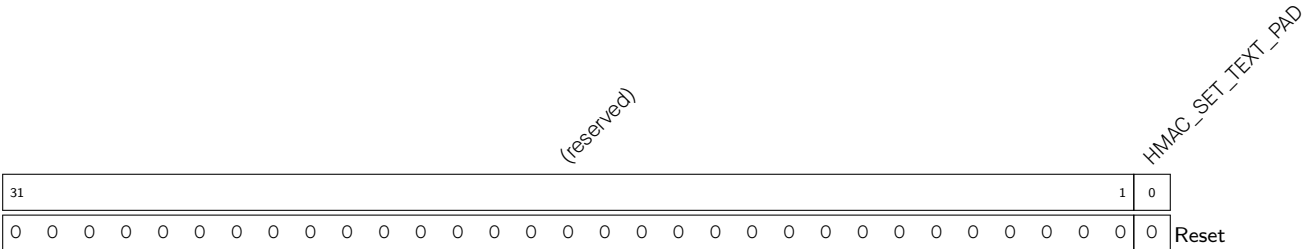


HMAC_BUSY_STATE 表示 HMAC 是否处于忙碌状态。执行计算任务之前，请确保 HMAC 已空闲。

- 0: 空闲
- 1: HMAC 仍处于工作状态

(RO)

Register 18.11. HMAC_SET_MESSAGE_PAD_REG (0x00F0)

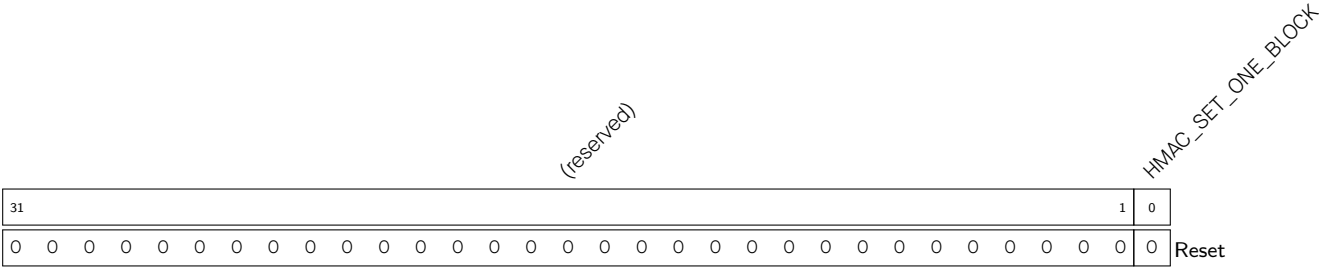


HMAC_SET_TEXT_PAD 表示是否由软件执行填充操作。

- 0: 不由软件执行
- 1: 由软件执行

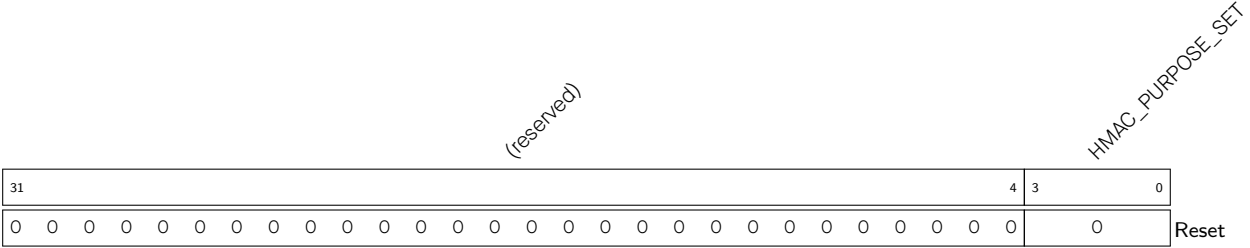
(WO)

Register 18.12. HMAC_ONE_BLOCK_REG (0x00F4)



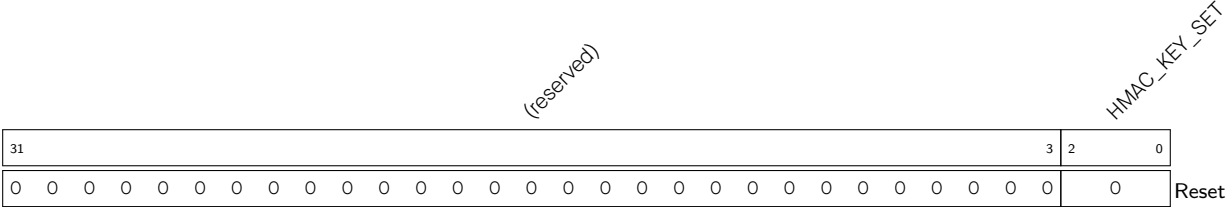
HMAC_SET_ONE_BLOCK 置 1 表示 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位，无需填充。(WS)

Register 18.13. HMAC_SET_PARA_PURPOSE_REG (0x0044)



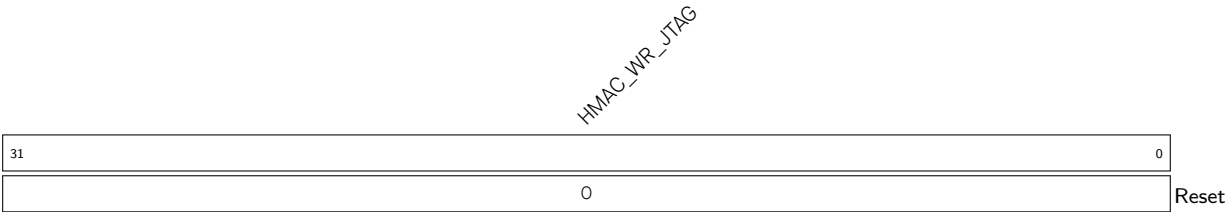
HMAC_PURPOSE_SET 设置 HMAC 功能，请参阅表 18.4-1。(WO)

Register 18.14. HMAC_SET_PARA_KEY_REG (0x0048)



HMAC_KEY_SET 配置 HMAC 密钥。共有 6 个 eFuse 密钥和 1 个 key manager 密钥，编号 0 ~ 5 对应 6 个 eFuse 密钥，编号 7 对应 key manager 密钥，将选择的密钥编号写入该字段即可。(WO)

Register 18.15. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG 写入用于重启 JTAG 的输入比较数值。(WO)

Register 18.16. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)

(reserved)																																HMAC_SOFT_JTAG_CTRL			
31																															1	0			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

HMAC_SOFT_JTAG_CTRL 配置是否开启 JTAG 验证模式。

0: 关闭

1: 开启

(WS)

Register 18.17. HMAC_DATE_REG (0x01FC)

(reserved)		HMAC_DATE																										
31	30	29																										0
0	0	0x20230901																									Reset	

HMAC_DATE 版本控制寄存器。(R/W)

第 19 章

RSA 加速器 (RSA)

19.1 概述

RSA 加速器可为多种运用于“RSA 非对称式加密演算法”的高精度计算提供硬件支持，能够极大地降低此类运算的运行时间和软件复杂度。与纯软件 RSA 算法相比，硬件 RSA 加速器的运算速度更快。RSA 加速器还支持多种“运算子长度”，具有很高的灵活性。

19.2 主要特性

RSA 加速器支持以下功能：

- 大数模幂运算（支持两个加速选项）
- 大数模乘运算
- 大数乘法运算
- 多种运算子长度
- 支持在运算完成后触发中断

19.3 功能描述

RSA 加速器的激活需使能 `PCR_RSA_CONF_REG` 寄存器中的 `PCR_RSA_CLK_EN` 位，并同时清零 `PCR_RSA_RST_EN` 位。此外，还需要清零 `PCR_DS_RST_EN` 和 `PCR_ECDSA_RST_EN` 以复位 [数字签名算法 \(DSA\)](#) 和 [椭圆曲线数字签名算法 \(ECDSA\)](#)。

不过，RSA 加速器激活后还须等待 [RSA 相关存储器](#) 初始化完成后才能开始工作。具体来说，`RSA_QUERY_CLEAN_REG` 读 0 时初始化开始，读 1 时初始化完成。因此，在复位后首次使用 RSA 加速器时，软件需要先查询 `RSA_QUERY_CLEAN_REG` 的值是否为 1，以确保 RSA 加速器可正常工作。

此外，RSA 加速器支持中断功能，可对寄存器 `RSA_INT_ENA_REG` 写 1 或 0 以开启或关闭中断。RSA 加速器的中断功能默认开启。

注意：

ESP32-C5 的 [数字签名算法 \(DSA\)](#) 模块在工作期间会调用 RSA 加速器。此时，用户无法正常访问 RSA 加速器。

19.3.1 大数模幂运算

大数模幂运算的算法是 $Z = X^Y \bmod M$ ，它是基于蒙哥马利乘法 (Montgomery Multiplication) 实现的。因此，对于大数模幂运算，除了需要运算子 X 、 Y 、 M 外，还需要额外两个运算子，即参数 $\bar{r}$ 和 M' 。这两个参数需要通过软件提前运算得到。

RSA 加速器支持运算子长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \dots, 96\}$) 的大数模幂运算。 Z 、 X 、 Y 、 M 和 $\bar{r}$ 的位宽为这 96 种中的任意一种，要求它们的位宽必须相同，而 M' 的位宽始终是 32。

设进制数

$$b = 2^{32}$$

则运算子可以由若干个 b 进制数来表示：

$$n = \frac{N}{32}$$

$$Z = (Z_{n-1}Z_{n-2} \cdots Z_0)_b$$

$$X = (X_{n-1}X_{n-2} \cdots X_0)_b$$

$$Y = (Y_{n-1}Y_{n-2} \cdots Y_0)_b$$

$$M = (M_{n-1}M_{n-2} \cdots M_0)_b$$

$$\bar{r} = (\bar{r}_{n-1}\bar{r}_{n-2} \cdots \bar{r}_0)_b$$

其中 $Z_{n-1} \cdots Z_0$ 、 $X_{n-1} \cdots X_0$ 、 $Y_{n-1} \cdots Y_0$ 、 $M_{n-1} \cdots M_0$ 、 $\bar{r}_{n-1} \cdots \bar{r}_0$ 分别表示一个 b 进制数，位宽皆为 32。且 Z_{n-1} 、 X_{n-1} 、 Y_{n-1} 、 M_{n-1} 、 $\bar{r}_{n-1}$ 分别为 Z 、 X 、 Y 、 M 、 $\bar{r}$ 最高位的 b 进制数，而 Z_0 、 X_0 、 Y_0 、 M_0 、 $\bar{r}_0$ 分别为 Z 、 X 、 Y 、 M 、 $\bar{r}$ 最低位的 b 进制数。

另设 $R = b^n$ ，则计算得参数 $\bar{r} = R^2 \bmod M$ 。

另外 M' 可使用下方公式计算：

$$M' = -M^{-1} \bmod b$$

其中， M^{-1} 为 M 对 R 的[模逆元](#)，可使用扩展二进制 GCD 算法计算。

大数模幂运算的软件流程为：

1. 对寄存器 [RSA_INT_ENA_REG](#) 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。
 - (a) 对寄存器 [RSA_MODE_REG](#) 写入 $(\frac{N}{32} - 1)$ 。
 - (b) 对寄存器 [RSA_M_PRIME_REG](#) 写入 M' 。
 - (c) 根据需要配置加速选项相关寄存器。请参照章节 [19.3.4](#) 获取详细信息。
3. 将 X_i 、 Y_i 、 M_i 、 $\bar{r}_i$ ($i \in \{0, 1, \dots, n-1\}$) 分别写入存储器 [RSA_X_MEM](#)、[RSA_Y_MEM](#)、[RSA_M_MEM](#)、[RSA_Z_MEM](#)。每块存储器的容量都是 96 字 (word)。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。
只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 [RSA_SET_START_MODEXP](#) 写入 1 启动计算。
5. 等待运算结束。轮询 [RSA_QUERY_IDLE](#) 直到读到 1，或者等待 RSA 中断产生。

6. 从存储器 `RSA_Z_MEM` 读出运算结果 $Z_i (i \in \{0, 1, \dots, n-1\})$ 。
7. 若中断功能已开启，对 `RSA_CLEAR_INTERRUPT` 写入 1 以清除中断。

运算结束后，寄存器 `RSA_MODE_REG` 中存储的运算子长度信息以及存储器 `RSA_Y_MEM` 中的 Y_i 、存储器 `RSA_M_MEM` 中的 M_i 、寄存器 `RSA_M_PRIME_REG` 中的 M' 都不会变化。但是，存储器 `RSA_X_MEM` 中的 X_i 与存储器 `RSA_Z_MEM` 中的 $\bar{r}_i$ 都被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

19.3.2 大数模乘运算

大数模乘运算 $Z = X \times Y \bmod M$ 也是基于蒙哥马利乘法实现的。因此，与大数模幂运算类似，也需要预先通过软件计算额外的两个运算子 $\bar{r}$ 和 M' 。

RSA 加速器也支持 96 种运算子长度的大数模乘运算。

大数模乘运算的软件流程为：

1. 对寄存器 `RSA_INT_ENA_REG` 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。
 - (a) 对寄存器 `RSA_MODE_REG` 写入 $(\frac{N}{32} - 1)$ 。
 - (b) 对寄存器 `RSA_M_PRIME_REG` 写入 M' 。
3. 将 X_i 、 Y_i 、 M_i 、 $\bar{r}_i (i \in \{0, 1, \dots, n-1\})$ 分别写入存储器 `RSA_X_MEM`、`RSA_Y_MEM`、`RSA_M_MEM`、`RSA_Z_MEM`。每块存储器的容量都是 96 字 (word)。

每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。

4. 对 `RSA_SET_START_MODMULT` 写入 1。
5. 等待运算结束。轮询 `RSA_QUERY_IDLE` 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 `RSA_Z_MEM` 读出运算结果 $Z_i (i \in \{0, 1, \dots, n-1\})$ 。
7. 若中断功能已开启，对寄存器 `RSA_CLEAR_INTERRUPT` 写入 1 以清除中断。

运算结束后，寄存器 `RSA_MODE_REG` 中存储的运算子长度信息以及存储器 `RSA_X_MEM` 中的 X_i 、存储器 `RSA_Y_MEM` 中的 Y_i 、存储器 `RSA_M_MEM` 中的 M_i 、寄存器 `RSA_M_PRIME_REG` 中的 M' 都不会变化。但是，存储器 `RSA_Z_MEM` 中的 $\bar{r}_i$ 已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

19.3.3 大数乘法运算

大数乘法运算实现了 $Z = X \times Y$ 。其中 Z 的长度是运算子 X 、 Y 长度的两倍，所以 RSA 加速器只支持运算子 X 、 Y 长度为 $N = 32 \times x (x \in \{1, 2, 3, \dots, 48\})$ 的大数乘法运算。运算子 Z 的长度 $\hat{N}$ 为 $2 \times N$ 。

大数乘法运算的软件流程为：

1. 对寄存器 `RSA_INT_ENA_REG` 写 1 或 0 以开启或关闭中断。
2. 配置相关寄存器。对寄存器 `RSA_MODE_REG` 写入 $(\frac{\hat{N}}{32} - 1)$ ，即 $(\frac{N}{16} - 1)$ 。

3. 将 X_i 、 Y_i ($i \in \{0, 1, \dots, n-1\}$) 分别写入存储器 `RSA_X_MEM`、`RSA_Z_MEM`。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。 n 为 $\frac{N}{32}$ 。

X_i ($i \in \{0, 1, \dots, n-1\}$) 要填充到存储器 `RSA_X_MEM` 中的第 i 个字对应的地址中，但需要注意的是， Y_i ($i \in \{0, 1, \dots, n-1\}$) 并不是要填充到存储器 `RSA_Z_MEM` 中的第 i 个字对应的地址中，而是需要填充到存储器 `RSA_Z_MEM` 中的第 $n+i$ 个字对应的地址中，即存储器 `RSA_Z_MEM` 的基地址加上偏移量 $4 \times (n+i)$ 。

只需要根据运算子长度，将这两个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。

4. 对 `RSA_SET_START_MULT` 写入 1。
5. 等待运算结束。轮询 `RSA_QUERY_IDLE` 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 `RSA_Z_MEM` 读出运算结果 Z_i ($i \in \{0, 1, \dots, \hat{n}-1\}$)。 $\hat{n}$ 为 $2 \times n$ 。
7. 若中断功能已开启，对 `RSA_CLEAR_INTERRUPT` 写入 1 以清除中断。

运算结束后，寄存器 `RSA_MODE_REG` 中存储的运算子长度信息以及存储器 `RSA_X_MEM` 中的 X_i 都不会变化。但是，存储器 `RSA_Z_MEM` 中的 Y_i 已经被覆盖。所以当需要连续运算时，只需要更新必需的寄存器与存储器即可。

19.3.4 控制加速

对于大数模幂运算，ESP32-C5 的 RSA 加速器还特别提供 `SEARCH` 和 `CONSTANT_TIME` 两个选项，可进一步提高运算速度。默认情况下，这两个选项均处于不额外加速状态，可以单独使用，也可以同时使用。值得注意的是，即使这两个选项均处于不额外加速状态，硬件 RSA 加速器也比纯软件实现的 RSA 算法快很多。

具体来说，当这两个选项均处于不额外加速状态时，求解 $Z = X^Y \bmod M$ 的时间开销完全由运算子长度决定。否则，只要有某个选项携带有额外加速效果，那么运算的时间开销还与 Y 的 O/1 分布有关。

为了更清楚地说明问题，首先假设 Y 的二进制表示为：

$$Y = (\tilde{Y}_{N-1}\tilde{Y}_{N-2}\cdots\tilde{Y}_{t+1}\tilde{Y}_t\tilde{Y}_{t-1}\cdots\tilde{Y}_0)_2$$

其中，

- N 代表 Y 的长度，
- $\tilde{Y}_t$ 的值为 1，
- $\tilde{Y}_{N-1}, \tilde{Y}_{N-2}, \dots, \tilde{Y}_{t+1}$ 的值均为 0，
- 且 $\tilde{Y}_{t-1}, \tilde{Y}_{t-2}, \dots, \tilde{Y}_0$ 中包括 m 个 0，其余 $t-m$ 全部为 1，即 $\tilde{Y}_{t-1}\tilde{Y}_{t-2}, \dots, \tilde{Y}_0$ 的汉明重量 (Hamming weight) 为 $t-m$ 。

此时，当启动任一额外加速选项时：

- `SEARCH` 选项 (`RSA_SEARCH_ENABLE` 置 1 开启额外加速)
 - RSA 加速器将忽略所有 $\tilde{Y}_i$ ($i > \alpha$) 位。其中，加速位置 α 可通过 `RSA_SEARCH_POS_REG` 寄存器配置。 α 在配置时需小于 $N-1$ ，否则相当于没有额外加速；且不建议小于 t ，否则无法正确求解 $Z = X^Y \bmod M$ 。当设置 α 为 t 时，加速效果最佳。此时， $\tilde{Y}_{N-1}, \tilde{Y}_{N-2}, \dots, \tilde{Y}_{t+1}$ 中的 0 位将在运算中全部被忽略。

- 注意，这个选项会忽略一些位，进而导致密钥长度变短，使安全性降低。因此，不应使用在安全要求高的应用中。
- CONSTANT_TIME 选项 (RSA_CONSTANT_TIME_REG 置 0 开启额外加速)
 - RSA 加速器在运算过程中将简化对 Y 中 0 位的处理。因此， Y 中的 0 越多，加速效果越明显。
 - 注意，这个选项的加速效果与密钥中的 0/1 分布密切相关，这可能会被侧信道攻击 (SCA) 等攻击加以利用，使安全性降低。因此，不应使用在安全要求高的应用中。

为了直观地展示这两个选项带来的额外加速效果，下面通过一个典型实例加以说明。

本实例中：

- $Z = X^Y \bmod M$
- N 等于 3072
- Y 等于 65537
- X 和 M 取任意值
- 当 SEARCH 选项开启时，设定 α 为 16，即 RSA_SEARCH_POS_REG 配置为 16。

表 19.3-1 展示了在进行上述 $Z = X^Y \bmod M$ 运算时，开启不同加速选项组合分别对应的时间开销。

表 19.3-1. 加速效果

SEARCH 选项	CONSTANT_TIME 选项	时间开销 (时钟周期)
不加速	不加速	174.7×10^6
加速	不加速	1.023×10^6
不加速	加速	0.546×10^6
加速	加速	0.540×10^6

如图 19.3-1 所示：

- 当两个选项均处于不额外加速状态时，时间开销最大。
- 当两个选项均处于额外加速状态时，时间开销最小。
- 相比于不额外加速状态，任一选项处于额外加速状态时的时间开销明显大幅度降低。

19.4 中断

ESP32-C5 的 RSA 加速器可以生成以下中断信号并发送给中断矩阵。

- RSA_INTR

上述中断信号由 RSA 加速器的内部中断源生成。表 19.4-1 列出了 RSA 加速器的内部中断源、中断触发条件、以及生成的中断信号。

表 19.4-1. RSA 的内部中断源

内部中断源	触发条件	中断信号
RSA_CALC_DONE_INT	一次 RSA 运算结束	RSA_INTR

说明:

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 *[to be added later]* > 3.1 中断术语。

19.5 存储器列表

请注意，这里的地址都是相对于 RSA 加速器基地址的地址偏移量（相对地址），详见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	大小（字节）	起始地址	结束地址	访问
RSA_M_MEM	存储器 <i>M</i>	384	0x0000	0x017F	R/W
RSA_Z_MEM	存储器 <i>Z</i>	384	0x0200	0x037F	R/W
RSA_Y_MEM	存储器 <i>Y</i>	384	0x0400	0x057F	R/W
RSA_X_MEM	存储器 <i>X</i>	384	0x0600	0x077F	R/W

19.6 寄存器列表

本小节的所有地址均为相对于 RSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

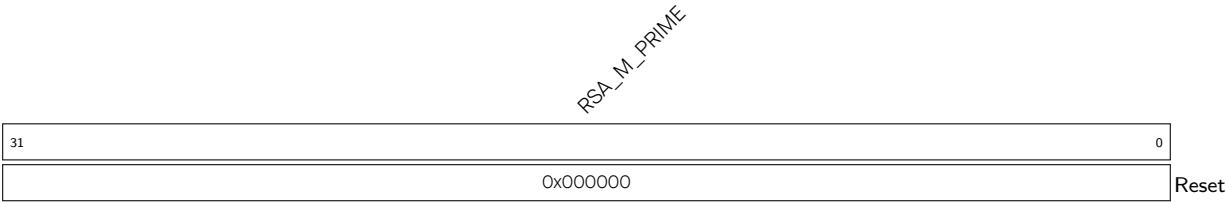
名称	描述	地址	访问
控制/配置寄存器			
RSA_M_PRIME_REG	配置 <i>M'</i>	0x0800	R/W
RSA_MODE_REG	配置 RSA 长度模式	0x0804	R/W
RSA_SET_START_MODEXP_REG	启动模幂运算	0x080C	WT
RSA_SET_START_MODMULT_REG	启动模乘运算	0x0810	WT
RSA_SET_START_MULT_REG	启动乘法运算	0x0814	WT
RSA_QUERY_IDLE_REG	代表 RSA 状态	0x0818	RO
RSA_CONSTANT_TIME_REG	配置 CONSTANT_TIME 选项	0x0820	R/W
RSA_SEARCH_ENABLE_REG	配置 SEARCH 选项	0x0824	R/W
RSA_SEARCH_POS_REG	配置 SEARCH 开始位置	0x0828	R/W
状态寄存器			
RSA_QUERY_CLEAN_REG	表示 RSA 内存初始化状态	0x0808	RO
中断寄存器			
RSA_INT_CLR_REG	RSA 中断清除寄存器	0x081C	WT
RSA_INT_ENA_REG	RSA 中断使能寄存器	0x082C	R/W
版本寄存器			
RSA_DATE_REG	版本控制寄存器	0x0830	R/W

19.7 寄存器

本小节的所有地址均为相对于 RSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

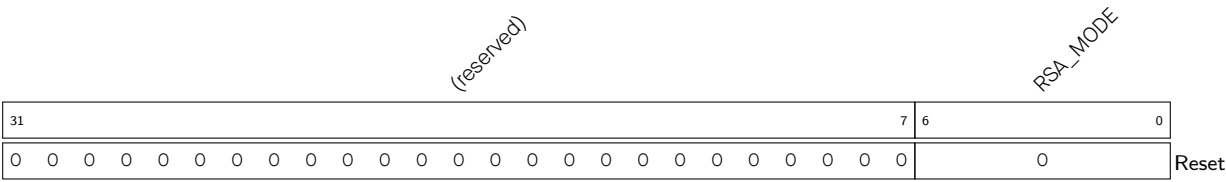
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 19.1. RSA_M_PRIME_REG (0x0800)



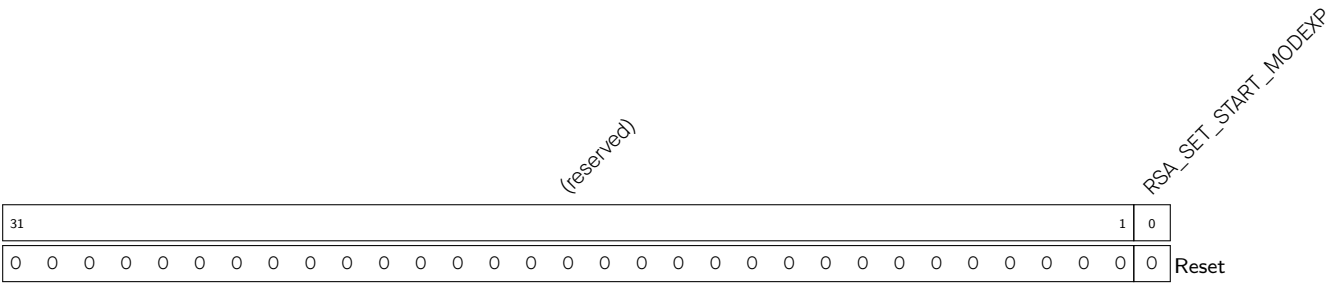
RSA_M_PRIME 配置 M' 。
(R/W)

Register 19.2. RSA_MODE_REG (0x0804)



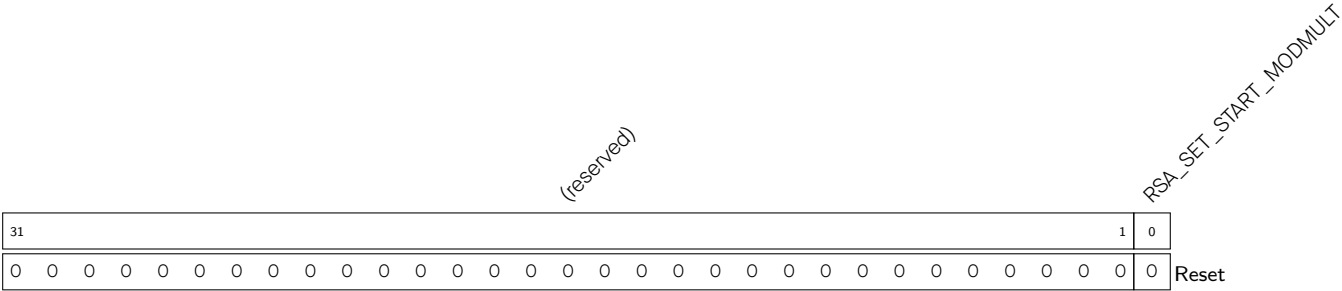
RSA_MODE 配置 RSA 运算子长度。
(R/W)

Register 19.3. RSA_SET_START_MODEXP_REG (0x080C)



RSA_SET_START_MODEXP 配置是否启动模幂运算。
0: 无效果
1: 启动
(WT)

Register 19.4. RSA_SET_START_MODMULT_REG (0x0810)



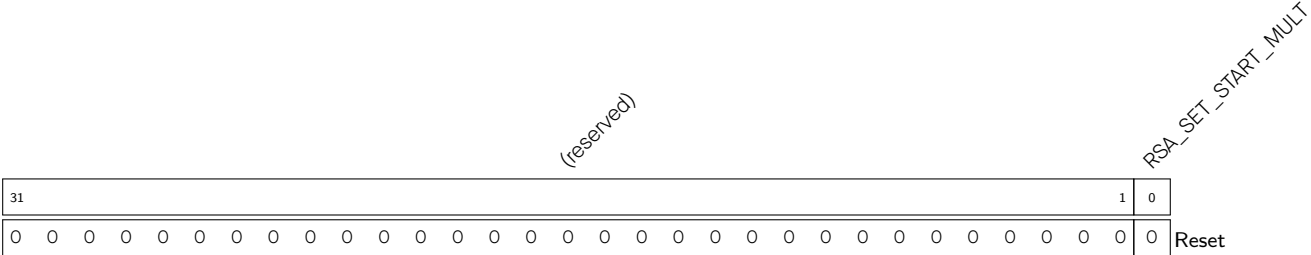
RSA_SET_START_MODMULT 配置是否启动模乘运算。

0: 无效果

1: 启动

(WT)

Register 19.5. RSA_SET_START_MULT_REG (0x0814)



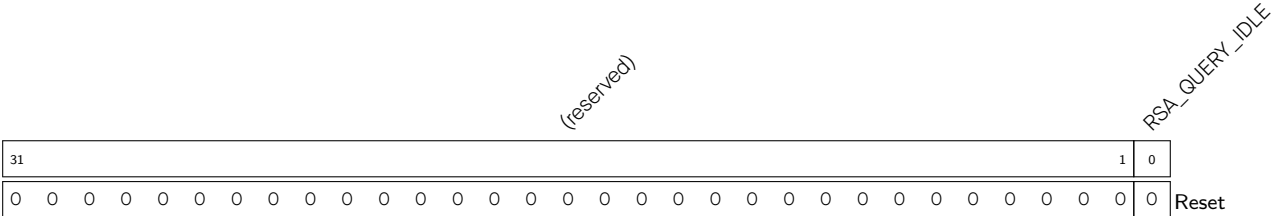
RSA_SET_START_MULT 配置是否启动乘法运算。

0: 无效果

1: 启动

(WT)

Register 19.6. RSA_QUERY_IDLE_REG (0x0818)



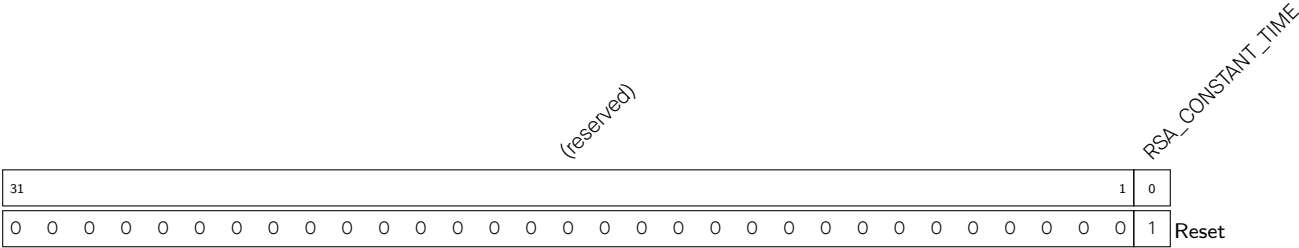
RSA_QUERY_IDLE 代表 RSA 状态。

0: 忙碌

1: 空闲

(RO)

Register 19.7. RSA_CONSTANT_TIME_REG (0x0820)



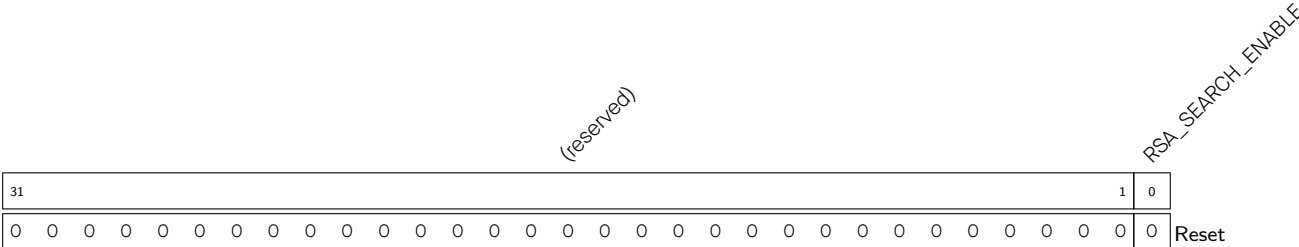
RSA_CONSTANT_TIME 配置 CONSTANT_TIME 选项。

0: 加速

1: 不加速（默认）

(R/W)

Register 19.8. RSA_SEARCH_ENABLE_REG (0x0824)



RSA_SEARCH_ENABLE 配置 SEARCH 选项。

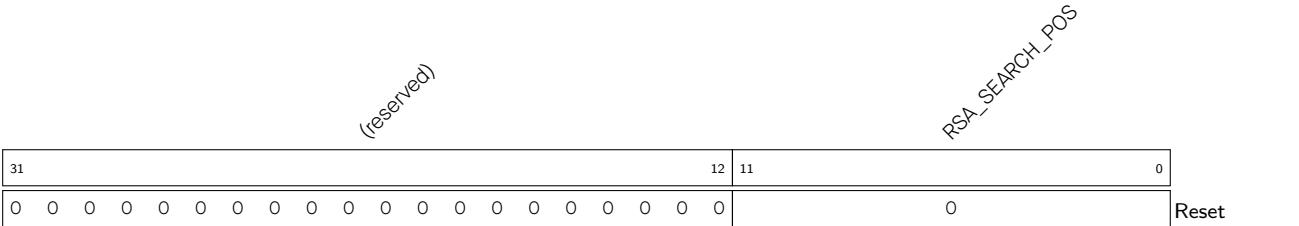
0: 不加速（默认）

1: 加速

此选项需配合 [RSA_SEARCH_POS_REG](#) 使用。

(R/W)

Register 19.9. RSA_SEARCH_POS_REG (0x0828)

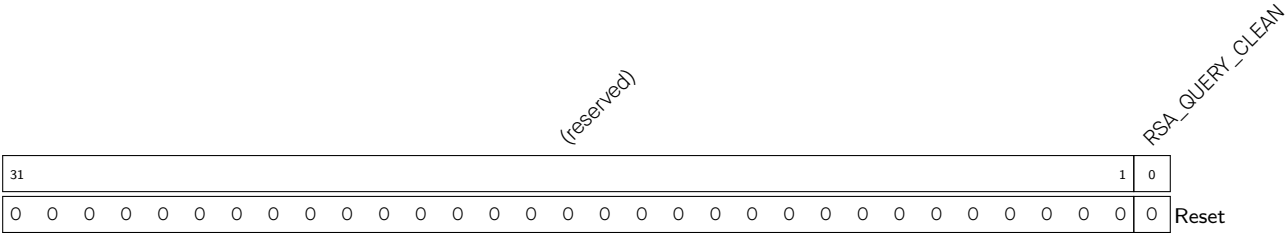


RSA_SEARCH_POS 配置 SEARCH 的开始地址。

此选项需配合 [RSA_SEARCH_ENABLE_REG](#) 使用，仅在 [RSA_SEARCH_ENABLE](#) 为 1 时有效。

(R/W)

Register 19.10. RSA_QUERY_CLEAN_REG (0x0808)



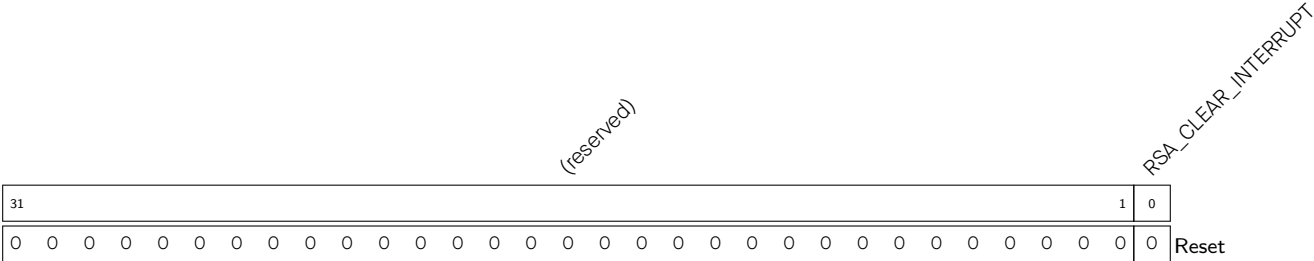
RSA_QUERY_CLEAN 代表 RSA 内存初始化的状态。

0: 没有完成

1: 已经完成

(RO)

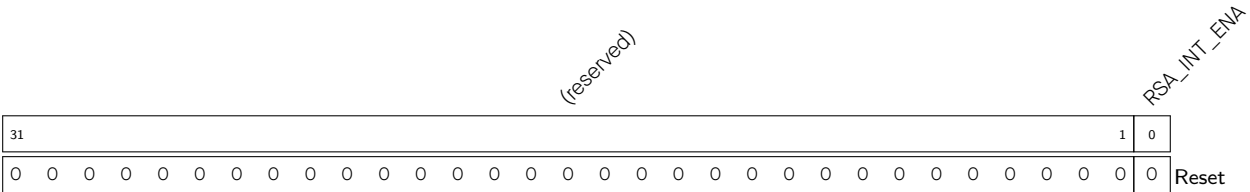
Register 19.11. RSA_INT_CLR_REG (0x081C)



RSA_CLEAR_INTERRUPT 写 1 清除 RSA 中断。

(WT)

Register 19.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA 写 1 使能 RSA 中断。

(R/W)

Register 19.13. RSA_DATE_REG (0x0830)

(reserved)		RSA_DATE	
31	30	29	0
0	0	0x20200618	
			Reset

RSA_DATE 版本控制寄存器。
(R/W)

第 20 章

SHA 加速器 (SHA)

20.1 概述

ESP32-C5 内置 SHA（安全哈希算法）硬件加速器可完成 SHA 运算，具有 [Typical SHA](#) 和 [DMA-SHA](#) 两种工作模式。整体而言，相比基于纯软件的 SHA 运算，SHA 硬件加速器能够极大地提高运算速度。

20.2 主要特性

ESP32-C5 的 SHA 硬件加速器：

- 支持 [FIPS PUB 180-4 规范](#) 中的以下运算标准
 - SHA-1 运算
 - SHA-224 运算
 - SHA-256 运算
- 提供两种工作模式
 - Typical SHA 工作模式
 - DMA-SHA 工作模式
- 允许插入 (interleaved) 功能（仅限 Typical SHA 工作模式）
- 允许中断功能（仅限 DMA-SHA 工作模式）

20.3 工作模式简介

ESP32-C5 内置的 SHA 加速器支持两种工作模式。

- [Typical SHA 工作模式](#)：所有数据读写统一通过 CPU 访问完成。
- [DMA-SHA 工作模式](#)：所有读数据通过硬件上的 DMA 完成。具体来说，用户可配置 DMA 控制器，由 DMA 控制器提供 SHA 运算过程中所需的数据信息。因此，可以释放 CPU 执行其他任务。

SHA 加速器的激活需要使能 [PCR_SHA_CONF_REG](#) 外围时钟的 [PCR_SHA_CLK_EN](#) 位，并同时清零 [PCR_SHA_RST_EN](#) 位。此外，还需要清零 [PCR_DS_RST_EN](#), [PCR_HMAC_RST_EN](#) 和 [PCR_ECDSA_RST_EN](#) 复位 [数字签名算法 \(DSA\)](#)，[HMAC 加速器 \(HMAC\)](#) 和 [椭圆曲线数字签名算法 \(ECDSA\)](#)。

用户可通过配置 [SHA_START_REG](#) 或 [SHA_DMA_START_REG](#) 选择 SHA 加速器的工作模式，具体请见表 20.3-1。

表 20.3-1. 工作模式选择

工作模式	选择方式
Typical SHA	SHA_START_REG 置 1
DMA-SHA	SHA_DMA_START_REG 置 1

用户可通过配置 SHA_MODE_REG 寄存器选择 SHA 加速器的运算标准，具体请见表 20.3-2。

表 20.3-2. 运算标准选择

哈希运算标准	SHA_MODE_REG 的配置
SHA-1	0
SHA-224	1
SHA-256	2

注意：
ESP32-C5 的 数字签名算法 (DSA) 和 HMAC 加速器 (HMAC) 模块在工作时也会调用 SHA 加速器。此时，用户无法正常访问 SHA 加速器。

20.4 功能描述

SHA 加速器可以提取信息摘要 (message digest)，其主要工作流程分为两步：[信息预处理](#) 和 [哈希运算](#)。

20.4.1 信息预处理

信息预处理分为三个主要步骤：[附加填充比特](#)、[信息解析](#) 和 [设置初始哈希值](#)。

20.4.1.1 附加填充比特

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件操作将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 m 位，则填充步骤如下：

- 1. 首先，在待处理信息后填充 1 个“1”；
- 2. 随后，再填充 k 个“0”。其中， k 为满足 $m + 1 + k \equiv 448 \text{ mod } 512$ 的最小非负数解；
- 3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 m 的值。

更多详情，请参考 [《FIPS PUB 180-4 规范》](#) > 章节“Padding the Message”。

20.4.1.2 信息解析

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$ 、 $M^{(2)}$ 、...、 $M^{(N)}$ 。一个 512 位信息块包括 16 个 32 位的字 (word)，则第 i 个信息块的第一个 32 位字表示为 $M_0^{(i)}$ ，第二个 32 位字表示为 $M_1^{(i)}$ ，...，第 16 个 32 位字表示为 $M_{15}^{(i)}$ 。

SHA 加速器在工作时，每次处理的信息块数据均将按照如下规则写入相应的寄存器中：将 $M_0^{(i)}$ 存放在 `SHA_M_0_REG` 中， $M_1^{(i)}$ 存放在 `SHA_M_1_REG`，...， $M_{15}^{(i)}$ 存放在 `SHA_M_15_REG` 中。

说明：

有关“信息块”及相关概念的描述，请参考 [《FIPS PUB 180-4 规范》](#) > 章节“Glossary of Terms and Acronyms”。

20.4.1.3 哈希初始值 (Initial Hash Value)

在进行哈希运算前，首先必须设置哈希初始值 $H^{(0)}$ ，其中 SHA-1、SHA-224 和 SHA-256 运算的哈希初始值为常量 C，且已经固定在硬件中，无需额外配置。

20.4.2 哈希运算流程

在完成信息预处理后，ESP32-C5 SHA 加速器将正式开始哈希运算，最终根据不同运算标准得到不同长度的信息摘要。正如上文所述，ESP32-C5 SHA 加速器支持 [Typical SHA](#) 和 [DMA-SHA](#) 两种工作模式，下面将对这两种工作模式的具体流程进行介绍。

20.4.2.1 Typical SHA 模式下的运算流程

通常情况下，ESP32-C5 的 SHA 会先处理完当前信息的所有信息块并生成该信息的信息摘要，之后再开始计算新的信息摘要。

不过，ESP32-C5 SHA 加速器还支持插入 (interleaved) 运算（仅 Typical SHA 支持），即在完成当前信息的所有运算前，用户在每个信息块计算完成后均可插入新的运算。

具体来说，用户可以将存储在 `SHA_H_n_REG` 寄存器中的信息摘要暂时保存到其他地方，然后让 SHA 加速器来完成其他优先级更高的运算任务。当插入的运算结束后，用户再将之前暂存的信息摘要重新写入 `SHA_H_n_REG` 中，并继续完成之前中断的计算。

Typical SHA 的具体运算流程

1. 选择运算标准。
 - 配置 `SHA_MODE_REG` 寄存器，设置运算标准。具体配置，请参考表 20.3-2。
2. 处理当前信息块。
 - 将当前信息块写入 `SHA_M_n_REG` 寄存器。
3. 启动 SHA 加速器¹。
 - 如果为首次运算，则对 `SHA_START_REG` 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器按照步骤 1 中选定的运算标准，使用硬件中固定的哈希初始值进行运算；
 - 如果非首次运算²，则对 `SHA_CONTINUE_REG` 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器使用 `SHA_H_n_REG` 寄存器中的值作为哈希初始值进行运算。
4. 查询当前信息块的处理进度。
 - 轮询寄存器 `SHA_BUSY_REG` 一直到读回的值为 0，代表 SHA 硬件加速器已完成对当前信息块的计算，进入“空闲”状态³。
5. 选择是否有后续的待处理信息块。

- 如果存在后续待处理信息块，则跳回执行步骤 2。
- 否则，继续执行。

6. 获取信息摘要：

- 从寄存器堆 `SHA_H_n_REG` 取出信息摘要。

说明：

1. 这里，在 SHA 加速器进行硬件运算时，如果存在后续待处理信息块，软件还可以同时将后续信息块写入 `SHA_M_n_REG` 寄存器，以节省时间。
2. 比如重新启动 SHA 加速器完成之前暂停任务的情况。
3. 这里，你可以选择是否需要插入其他任务。如需插入，请前往 [插入任务工作流程](#) 具体查看。

如上文所述，ESP32-C5 SHA 加速器支持在 **Typical SHA 模式** 下“插入”任务。

具体工作流程如下。

1. 保存插入前任务的以下数据，准备将 SHA 加速器的使用权移交给插入的任务。
 - 读取并保存寄存器 `SHA_MODE_REG` 中的运算标准类型。
 - 读取并保存寄存器堆 `SHA_H_n_REG` 中的信息摘要。
2. 执行插入的任务。具体按照插入运算类型的不同，请见 [Typical SHA](#) 或 [DMA-SHA 工作流程](#)。
3. 恢复插入前任务的以下数据，准备将 SHA 加速器的使用权交还给插入前的任务。
 - 将获得使用权前保存的运算标准类型重新写入寄存器 `SHA_MODE_REG`；
 - 将获得使用权前保存的信息摘要写入寄存器堆 `SHA_H_n_REG`。
4. 将之前任务的下一个待处理信息块写入 `SHA_M_n_REG` 寄存器，并对 `SHA_CONTINUE_REG` 寄存器置 1，重新启动 SHA 加速器，完成之前暂停的任务。

20.4.2.2 DMA-SHA 模式下的运算流程

ESP32-C5 SHA 加速器在 DMA-SHA 工作模式下不支持插入运算，即用户必须在每次 DMA 运算（可能包括 1 个或多个信息块）全部结束后才能插入新的运算。这种情况下，用户如有插入运算需求，可将较大信息块进行拆分，并进行多次 DMA 运算。每次 DMA 运算之间允许插入其他运算标准的计算任务。单次 DMA 运算最多可以处理 63 个数据块。

与 Typical SHA 不同，SHA 在 DMA-SHA 工作模式下，运算过程中的数据搬运过程均由 DMA 完成。具体配置可见章节 3 [通用 DMA 控制器 \(GDMA\)](#)。

DMA-SHA 的具体工作流程

1. 选择运算标准。
 - 配置 `SHA_MODE_REG` 寄存器，设置运算标准。具体配置，请参考表 20.3-2。
2. 选择是否启用中断。请将 `SHA_INT_ENA_REG` 寄存器配置为 1 以启动中断。
3. 配置块个数。
 - 将待加密数据的总块数 M 写入 `SHA_DMA_BLOCK_NUM_REG` 寄存器。
4. 开始 DMA-SHA 运算。

- 如果当前 DMA-SHA 运算为接着上一次 DMA-SHA 的运算，需要提前将上一次计算得到的信息摘要写入寄存器堆 `SHA_H_n_REG` 中，随后将 1 写入寄存器 `SHA_DMA_CONTINUE_REG`；
 - 否则，只需要将 1 写入寄存器 `SHA_DMA_START_REG`。
5. 等待 DMA-SHA 运算结束。判断 DMA-SHA 运算结束有以下两种方法：
- 轮询寄存器 `SHA_BUSY_REG` 结果为 0。
 - 等待中断信号产生。此时，应及时通过软件将 `SHA_INT_CLEAR_REG` 寄存器置为 1 以清除中断。
6. 获取信息摘要
- 从寄存器堆 `SHA_H_n_REG` 取出信息摘要。

20.4.3 信息摘要存储

哈希运算完成之后，计算得到的信息摘要被 SHA 加速器更新至对应的 `SHA_H_n_REG` ($n: 0 \sim 7$) 寄存器中。不同运算标准得到的信息摘要长度也不同，详情见表 20.4-1：

表 20.4-1. 不同运算标准信息摘要的寄存器占用情况

哈希运算标准	信息摘要长度（位）	寄存器占用情况 ¹
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG

¹ 信息摘要从最高有效位至最低有效位存放，第一个 word 存放在寄存器 `SHA_H_0_REG` 中，第二个 word 存放在寄存器 `SHA_H_1_REG` 中，以此类推。

20.4.4 中断

ESP32-C5 的 SHA 加速器可以生成以下中断信号并发送给中断矩阵。

- SHA_INTR

该中断信号由 SHA 加速器的内部中断源生成。表 20.4-2 列出了 SHA 加速器的内部中断源、中断触发条件、以及生成的中断信号。

表 20.4-2. SHA 加速器的内部中断源

内部中断源	触发条件	中断信号
SHA_CALC_DONE_INT	DMA-SHA 模式下完成信息摘要的计算	SHA_INTR

说明：

- 中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 [to be added later] > 3.1 中断术语。
- 与标准中断寄存器组不同，SHA 的中断寄存器组仅包含 `INT_ENA` (`SHA_INT_ENA_REG`) 和 `INT_CLR` (`SHA_INT_CLEAR_REG`) 字段，并不支持用户读取 `INT_RAW` 和 `INT_ST` 字段。

20.5 寄存器列表

本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

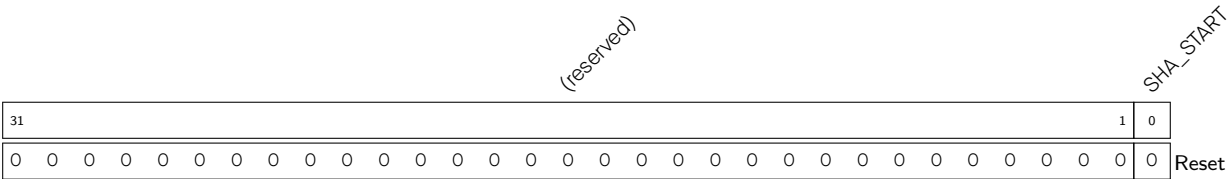
名称	描述	地址	权限
控制与配置寄存器			
SHA_MODE_REG	配置 SHA 加速器的运算标准	0x0000	R/W
SHA_CONTINUE_REG	继续 SHA 运算（仅用于 Typical SHA 模式）	0x0014	WO
SHA_DMA_START_REG	启动 SHA 加速器的 DMA-SHA 模式	0x001C	WO
SHA_START_REG	启动 SHA 加速器的 Typical SHA 模式	0x0010	WO
SHA_DMA_CONTINUE_REG	继续 SHA 运算（仅用于 DMA-SHA 模式）	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	信息块个数寄存器（仅用于 DMA-SHA 模式）	0x000C	R/W
状态寄存器			
SHA_BUSY_REG	表示 SHA 加速器是否处于“忙碌”状态	0x0018	RO
中断寄存器			
SHA_INT_CLEAR_REG	DMA-SHA 中断清除寄存器	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA 中断使能寄存器	0x0028	R/W
数据寄存器			
SHA_H_0_REG	哈希值	0x0040	R/W
SHA_H_1_REG	哈希值	0x0044	R/W
SHA_H_2_REG	哈希值	0x0048	R/W
SHA_H_3_REG	哈希值	0x004C	R/W
SHA_H_4_REG	哈希值	0x0050	R/W
SHA_H_5_REG	哈希值	0x0054	R/W
SHA_H_6_REG	哈希值	0x0058	R/W
SHA_H_7_REG	哈希值	0x005C	R/W
SHA_M_1_REG	输入信息	0x0084	R/W
SHA_M_2_REG	输入信息	0x0088	R/W
SHA_M_3_REG	输入信息	0x008C	R/W
SHA_M_4_REG	输入信息	0x0090	R/W
SHA_M_5_REG	输入信息	0x0094	R/W
SHA_M_6_REG	输入信息	0x0098	R/W
SHA_M_7_REG	输入信息	0x009C	R/W
SHA_M_8_REG	输入信息	0x00A0	R/W
SHA_M_9_REG	输入信息	0x00A4	R/W
SHA_M_10_REG	输入信息	0x00A8	R/W
SHA_M_11_REG	输入信息	0x00AC	R/W
SHA_M_12_REG	输入信息	0x00B0	R/W
SHA_M_13_REG	输入信息	0x00B4	R/W
SHA_M_14_REG	输入信息	0x00B8	R/W
SHA_M_15_REG	输入信息	0x00BC	R/W
版本寄存器			

名称	描述	地址	权限
SHA_DATE_REG	版本控制寄存器	0x002C	R/W

20.6 寄存器

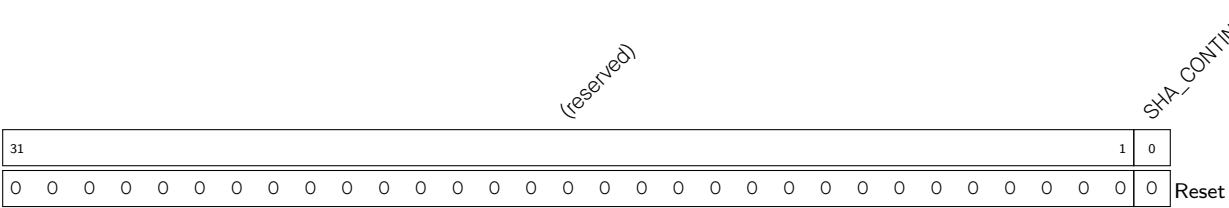
本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

Register 20.1. SHA_START_REG (0x0010)



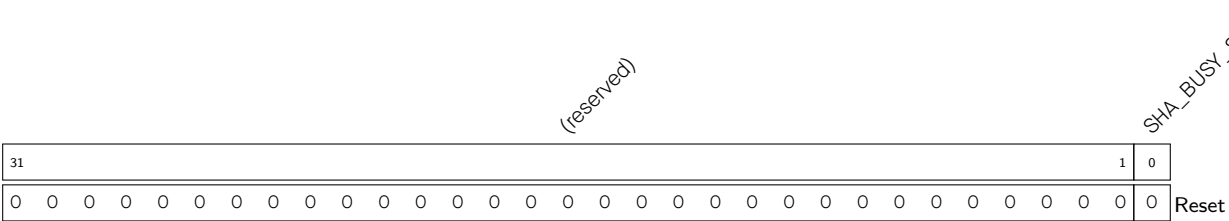
SHA_START 写 1 启动 SHA 加速器的 Typical SHA 模式。(WO)

Register 20.2. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE 写 1 继续 SHA 加速器的 Typical SHA 运算。(WO)

Register 20.3. SHA_BUSY_REG (0x0018)



SHA_BUSY_STATE 表示 SHA 是否处于“忙碌”状态。

- 0: 空闲
- 1: 忙碌

(RO)

Register 20.4. SHA_DMA_START_REG (0x001C)

(reserved)																															SHA_DMA_START																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Reset

SHA_DMA_START 写 1 启动 SHA 加速器的 DMA-SHA 模式。(WO)

Register 20.5. SHA_DMA_CONTINUE_REG (0x0020)

(reserved)																															SHA_DMA_CONTINUE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SHA_DMA_CONTINUE 写 1 继续 SHA 加速器的 DMA-SHA 运算。(WO)

Register 20.6. SHA_INT_CLEAR_REG (0x0024)

(reserved)																															SHA_CLEAR_INTERRUPT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

SHA_CLEAR_INTERRUPT 写 1 清除 DMA-SHA 中断。(WO)

Register 20.7. SHA_INT_ENA_REG (0x0028)

(reserved)																															SHA_INTERRUPT_ENA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

Reset

SHA_INTERRUPT_ENA 写 1 使能 DMA-SHA 中断。(R/W)

Register 20.8. SHA_DATE_REG (0x002C)

(reserved)										SHA_DATE										
31	30	29																		0
0	0	0x20201229																	Reset	

SHA_DATE 版本控制寄存器。(R/W)

Register 20.9. SHA_MODE_REG (0x0000)

[illegible]

SHA_MODE 配置 SHA 加速器的运算标准。

0: SHA-1

1: SHA-224

2: SHA-256

3 ~ 7: 无效值

(R/W)

Register 20.10. SHA_DMA_BLOCK_NUM_REG (0x000C)

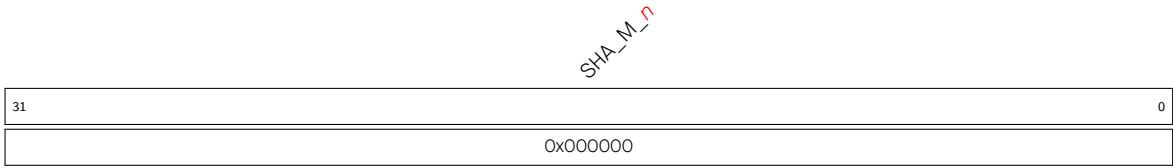
(reserved)																										SHA_DMA_BLOCK_NUM											
31																										5						0					
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																										0x0						Reset					

SHA_DMA_BLOCK_NUM 配置 DMA-SHA 工作模式下的信息块个数。(R/W)

Register 20.11. SHA_H_ *n*_REG (*n*: 0-7) (0x0040+4**n*)

SHA_H_*n* 表示第 *n* 个 32 位哈希值。(R/W)

Register 20.12. SHA_M_ *n* _REG (*n*: 0-15) (0x0080+4**n*)



SHA_M_ *n* 表示第 *n* 个 32 位输入信息。(R/W)

第 21 章

数字签名算法 (DSA)

21.1 概述

数字签名算法 (DSA) 使用密码学算法，用于验证消息的真实性和完整性。该算法也可用于向服务器验证设备身份，或验证消息是否经过篡改。

ESP32-C5 包含一个数字签名算法 (Digital Signature Algorithm, DSA) 模块，可提供硬件加速，高效生成基于 RSA 的数字签名。HMAC 作为密钥导出函数 (Key Derivation Function, KDF)，使用存储在 eFuse 或密钥管理器生成的密钥作为输入密钥，输出 *DSA_KEY* 密钥。随后，数字签名算法模块使用 *DSA_KEY* 解密预先加密的参数，计算出签名。上述过程都发生在硬件层面，因此，在计算过程中，不论是解密 RSA 参数的密钥，还是用于 HMAC 密钥导出函数的输入密钥，都对用户不可见。

21.2 主要特性

- 支持长度最大为 3072 位的 RSA 数字签名密钥
- 支持仅限 DSA 读取的加密私钥数据
- 支持 SHA-256 摘要，保护私钥数据免遭攻击者篡改

21.3 功能描述

21.3.1 概述

DSA 模块计算 RSA 签名的方式为 $Z = X^Y \bmod M$ ，其中 Z 是签名， X 是输入消息， Y 和 M 是 RSA 私钥参数。

私钥参数以密文形式存储在 flash 中。用于解密私钥参数的密钥 (*DSA_KEY*) 可以来自于 HMAC 模块或密钥管理器模块。当该私钥：

- 由 DSA 模块通过 HMAC 模块计算获取时，生成密钥所需的 HMAC 输入 *HMAC_KEY* 存放在 eFuse 中，只允许被 HMAC 模块访问。也就是说，只有 DSA 硬件才能解密私钥密文，软件无法获取私钥明文。关于 eFuse 和 HMAC 的相关细节请参照章节 2 [eFuse 控制器 \(eFuse\) \[to be added later\]](#) 和章节 18 [HMAC 加速器 \(HMAC\)](#)。
- 由 DSA 模块通过密钥管理器模块获取时，该密钥必须以安全的方式由密钥管理器部署而来。

需要签名时，软件直接将输入消息 X 发送到 DSA 外设。RSA 签名计算完成后，软件将读取签名结果 Z 。

为方便描述，此处约定几个符号，它们的作用域仅限于本章。

- 1^s : 表示一个完全由 “1” 组成的长度为 s 位的位串。

- $[x]_s$: 一个长度为 s 位的位串, 要求 s 为 8 的整数倍。如果 x 是一个数 ($x < 2^s$), 那么其在位串中遵循小端字节序。 x 可以是一个变量, 例如 $[Y]_{4096}$, 或一个十六进制的常数, 比如 $[0x0C]_8$ 。根据需要, $[x]_t$ 右边可以加上 $(s - t)$ 个 0, 使字符串长度扩展成 s 位, 得到 $[x]_s$ 。例如: $[0x05]_8 = 00000101$, $[0x05]_{16} = 0000010100000000$, $[0x0005]_{16} = 0000000000000101$, $[0x13]_8 = 00010011$, $[0x13]_{16} = 0001001100000000$, $[0x0013]_{16} = 0000000000010011$ 。
- $||$: 表示位串粘接操作符, 用于将两个位串前后粘成一个较长的位串。

21.3.2 私钥运算符

私钥运算符 Y (私钥指数) 和 M (密钥模数) 由用户生成。它们具有特定的 RSA 密钥长度 (最大为 3072 位)。RSA 签名操作还额外需要两个运算符, 即参数 $\bar{r}$ 和 M' , 由 Y 和 M 运算得到。

用户加密运算符 Y 、 M 、 $\bar{r}$ 、 M' 与验证摘要, 并以密文 C 的形式存储。密文 C 输入到 DSA 模块之后, 先由硬件解密, 然后参与 RSA 签名运算。有关生成 C 的方法, 请参考章节 21.3.3。

DSA 模块支持运算符长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \dots, 96\}$) 的 RSA 签名运算 $Z = X^Y \bmod M$ 。其中, 参数的位宽需要满足 RSA 计算的运算符长度要求, 即 Z 、 X 、 Y 、 M 和 $\bar{r}$ 的位宽必须相同, 且为 N 中的任意值, 而 M' 的位宽则始终是 32。更多 RSA 计算相关信息, 请参考章节 19 RSA 加速器 (RSA) 中的 19.3.1 大数模幂运算部分。

21.3.3 软件准备工作

如果用户想使用 DSA 模块进行数字签名, 则需要提前做好软件准备工作, 如图 21.3-1 所示。图中左半部分列出了在硬件开始 RSA 签名计算之前软件需要做的准备工作, 右半部分则列出了硬件在整个签名计算过程中的计算流程。

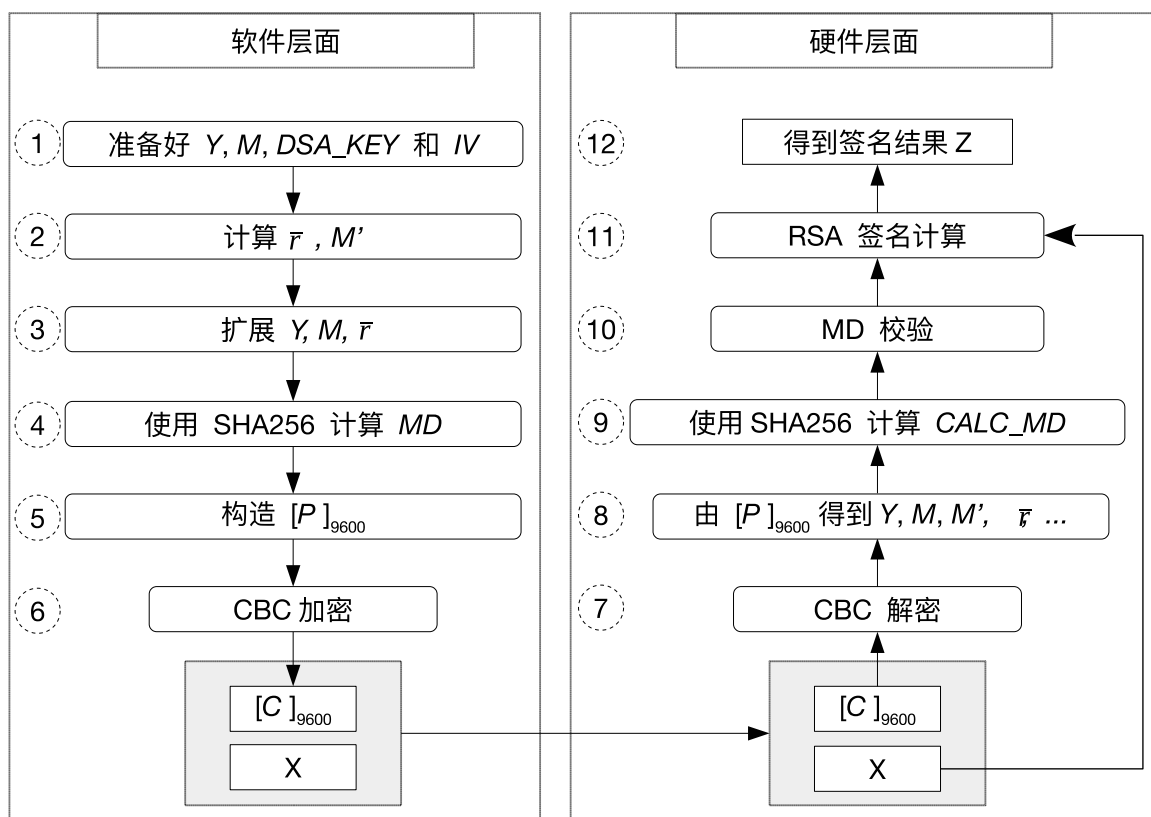


图 21.3-1. 软件准备工作与硬件工作流程

说明:

1. 即便有多次签名计算, 也只需进行一次软件准备工作 (图 21.3-1 左侧), 但每一次签名计算都需要重新进行硬件运算 (图 21.3-1 右侧)。
2. 软件准备需要配置时钟复位, 相关信息请参考章节 6 复位和时钟。

用户需要依照图 21.3-1 左侧中指定的步骤来计算 C 。详细过程如下:

- **步骤 1:** 按照章节 21.3.2 所述, 准备好符合运算子长度要求的 RSA 私钥运算子 Y 和 M 。定义 $[L]_{32} = \frac{N}{32} - 1$ (比如, 对于 RSA 3072, $[L]_{32} = [0x60-1]_{32}$)。根据私钥的生成方法准备 $[DSA\_KEY]_{256}$:
 - 当使用 HMAC 作为密钥来源时, 准备好 $[HMAC\_KEY]_{256}$, 并计算出 $[DSA\_KEY]_{256}$, 即 $DSA\_KEY = \text{HMAC-SHA256}([HMAC\_KEY]_{256}, 1^{256})$ 。
 - 当使用密钥管理器作为密钥来源时, 准备好 256 位的 DSA_KEY 。
- 接着, 生成一个符合 AES-CBC 块加密算法要求的随机 $[IV]_{128}$ 。有关 AES 的更多信息, 请参考章节 16 AES 加速器 (AES)。
- **步骤 2:** 根据 M 求解 $\bar{r}$ 和 M' 。
- **步骤 3:** 扩展 Y 、 M 和 $\bar{r}$, 得到 $[Y]_{3072}$ 、 $[M]_{3072}$ 和 $[\bar{r}]_{3072}$ 。此步骤是由于 Y 、 M 和 $\bar{r}$ 的最大位宽为 3072, 故位宽小于 3072 的运算子需要扩展至 3072, 而位宽等于 3072 的则不需要扩展。
- **步骤 4:** 使用 SHA-256 计算 MD 校验码: $[MD]_{256} = \text{SHA256}([Y]_{3072} || [M]_{3072} || [\bar{r}]_{3072} || [M']_{32} || [L]_{32} || [IV]_{128})$
- **步骤 5:** 构造 $[P]_{9600} = ([Y]_{3072} || [M]_{3072} || [\bar{r}]_{3072} || [Box]_{384})$ 。其中, $[Box]_{384} = ([MD]_{256} || [M']_{32} || [L]_{32} || [\beta]_{64})$, $[\beta]_{64}$ 是符合 PKCS#7 封装方式的追加码, 由 8 个字节码 0x80 组成, 即 $[0x0808080808080808]_{64}$, 目的在于使 P 的长度为 128 位的整数倍。
- **步骤 6:** 计算密文形式的私钥参数 $C = [C]_{9600} = \text{AES-CBC-ENC}([P]_{9600}, [DSA\_KEY]_{256}, [IV]_{128})$, 长度为 1200 字节。 C 也可以表示为 $C = [C]_{9600} = ([\hat{Y}]_{3072} || [\hat{M}]_{3072} || [\hat{\bar{r}}]_{3072} || [\hat{Box}]_{384})$ 。其中, $[\hat{Y}]_{3072}$ 、 $[\hat{M}]_{3072}$ 、 $[\hat{\bar{r}}]_{3072}$ 、 $[\hat{Box}]_{384}$ 是 C 的四个子参数, 分别对应 $[Y]_{3072}$ 、 $[M]_{3072}$ 、 $[\bar{r}]_{3072}$ 、 $[Box]_{384}$ 的密文形式。

21.3.4 硬件工作流程

每次需要计算数字签名时, 都会触发硬件操作。硬件需要三个输入信息: 预先生成的私钥密文 C 、唯一消息 X 以及 IV 。

DSA 模块的工作流程可以分为三个阶段:

1. 解析阶段, 即图 21.3-1 中的步骤 7 和步骤 8

解析过程是图 21.3-1 中步骤 6 的逆过程。DSA 模块将调用 AES 硬件加速器以 CBC 块模式对输入的密文信息 C 进行解密, 获取明文信息。该过程可以表示为 $P = \text{AES-CBC-DEC}(C, DSA\_KEY, IV)$ 。其中, IV 就是 $[IV]_{128}$, 由用户直接指定; $[DSA\_KEY]_{256}$ 由:

- 硬件 HMAC 提供, 通过存储在 eFuse 中的 $HMAC\_KEY$ 得到, 软件无法获取。
- 硬件密钥管理器提供时, 通过部署在密钥管理器中的 DSA_KEY 得到, 软件无法获取。

通过 P , DSA 模块能够解析出 $[Y]_{3072}$ 、 $[M]_{3072}$ 、 $[\bar{r}]_{3072}$ 、 $[M']_{32}$ 、 $[L]_{32}$ 、MD 校验码和追加码 $[\beta]_{64}$, 这相当于步骤 5 的逆过程。

2. 校验阶段，即图 21.3-1 中的步骤 9 和步骤 10

DSA 模块会执行两种校验操作：MD 校验和填充 (padding) 校验。由于填充校验和 MD 校验同步进行，因此填充校验未在图 21.3-1 中体现。

- MD 校验：DSA 模块调用 SHA-256 进行哈希计算获取哈希值 $[CALC\_MD]_{256}$ （即步骤 4）。然后，将 $[CALC\_MD]_{256}$ 与 MD 校验码 $[MD]_{256}$ 作比较，当且仅当二者相同时，MD 校验通过。
- 填充校验：DSA 模块检查解析阶段解析出的追加码 $[\beta]_{64}$ 是否符合 PKCS#7 标准，当且仅当符合标准时，填充校验通过。

仅当 MD 校验通过后，DSA 模块才会执行后续计算，否则 DSA 模块将拒绝执行。如果填充校验失败，将生成警告信息，但不会影响 DSA 模块的后续操作。

3. 计算阶段，即图 21.3-1 中的步骤 11 和步骤 12

DSA 模块将把用户输入的 X ，以及解析得到的 Y 、 M 和 $\bar{r}$ 都视为大数，结合解析得到的 M' ，构成了大数模幂运算 $X^Y \bmod M$ 的所有必要输入参数。大数模幂运算的运算长度由 L 的值唯一指定。DSA 模块调用 RSA 完成大数模幂运算 $Z = X^Y \bmod M$ ， Z 为签名结果。

21.3.5 软件工作流程

每次需要计算数字签名时，都应执行以下软件操作。输入内容为预先生成的私钥密文 C 、唯一的消息 X 、 IV 。本节中的软件步骤会触发章节 21.3.4 中描述的硬件工作流程。

下述流程基于一个假设：软件已经调用了 HMAC 模块，HMAC 已经根据 $HMAC\_KEY$ 计算出了 DSA_KEY 。

1. **准备工作**：准备好 C 、 X 、 IV 。具体方法请参考章节 21.3.3。
2. **启动 DSA**：对寄存器 `DS_SET_START_REG` 写 1。
3. **配置 `DS_KEY_SOURCE_REG` 选择 DSA_KEY 的来源，并检查是否已经准备好**：
 - 配置 `DS_KEY_SOURCE_REG` 为 1，即选择密钥管理器中部署的 DSA_KEY 。
检查 DSA_KEY 已经成功部署，检查方式为：读取寄存器域 `KEYMNG_KEY_DS_VLD`，当其值为 1 时，代表 DSA_KEY 已经成功部署并准备好。
 - 配置 `DS_KEY_SOURCE_REG` 为 0，即选择从 HMAC 模块计算 DSA_KEY 。
轮询寄存器 `DS_QUERY_BUSY_REG` 直到读到 0。
如果 `DS_QUERY_BUSY_REG` 超过 1 ms 还没读到 0，则说明 HMAC 初始化出现了问题。此时，软件应当读寄存器 `DS_QUERY_KEY_WRONG_REG`，根据返回值判断具体是下述哪一种情况：
 - 如果读到零值，说明 HMAC 未被调用。
 - 如果读到非零值 (1 ~ 15)，则说明 HMAC 被调用过，但是 DSA 模块没有成功从 HMAC 模块获取到 DSA_KEY 。此种情况可能是因为受到了其他程序的干扰。
4. **将 IV 写入存储器 `DS_IV_MEM`**：将 IV block 中的内容写入存储器 `DS_IV_MEM`（大小为 16 字节）。有关 IV block 的更多信息，请参考章节 16 AES 加速器 (AES)。
5. **将 X 写入存储器 `DS_X_MEM`**：将 $X_i (i \in \{0, 1, \dots, n-1\})$ 写入存储器 `DS_X_MEM`，容量为 96 个字 (word)，其中 $n = \frac{N}{32}$ 。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。当 X 的长度小于 96 个字时，存储器 `DS_X_MEM` 中有一部分空间未使用，该部分空间中的数据可以是任意值。
6. **将 C 写入存储器**：即将 C 中的四个子参数分别写入对应的存储器。

- 将 $\widehat{Y}_i$ ($i \in \{0, 1, \dots, 95\}$) 写入存储器 DS_Y_MEM。
- 将 $\widehat{M}_i$ ($i \in \{0, 1, \dots, 95\}$) 写入存储器 DS_M_MEM。
- 将 $\widehat{r}_i$ ($i \in \{0, 1, \dots, 95\}$) 写入存储器 DS_RB_MEM。
- 将 $\widehat{Box}_i$ ($i \in \{0, 1, \dots, 11\}$) 写入存储器 DS_BOX_MEM。

其中存储器 DS_Y_MEM、DS_M_MEM、DS_RB_MEM 的容量均为 96 个字，而存储器 DS_BOX_MEM 容量只有 12 个字。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算器的低位进制数，高地址存放运算器的高位进制数。

7. **启动计算**：对寄存器 DS_SET_CONTINUE_REG 写入 1。
8. **等待运算结束**：轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。
9. **检查校验结果**：读寄存器 DS_QUERY_CHECK_REG，根据返回值决定后续操作。
 - 如果返回值为 0，则说明填充校验通过，MD 校验通过，可以继续读取 Z 结果值。
 - 如果返回值为 1，则说明填充校验通过，但 MD 校验失败。 Z 结果值无效，跳至步骤 11。
 - 如果返回值为 2，则说明填充校验失败，但 MD 校验通过，用户可以继续读取 Z 结果值。但仍需注意，此时数据填充不符合 PKCS#7 封装方式。
 - 如果返回值为 3，则说明填充校验失败，且 MD 校验失败。这种情况说明有致命错误发生。 Z 结果值无效。跳至步骤 11。
10. **读出运算结果**：从存储器 DS_Z_MEM 读出运算结果 Z_i ($i \in \{0, 1, \dots, n-1\}$)，其中 $n = \frac{N}{32}$ 。 Z 以小端字节序存储在存储器中。
11. **退出计算环境**：对寄存器 DS_SET_FINISH_REG 写入 1，然后轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。

DSA 退出计算环境后，所有输入/输出寄存器和存储器中的数据都已经被抹除（清零）。

21.4 存储器列表

请注意，这里的起始地址和结束地址都是相对于数字签名算法基地址的地址偏移量（相对地址），具体基地址请见章节 4 [系统和存储器](#) 中的表 4.3-2。

名称	描述	大小（字节）	起始地址	结束地址	访问
DS_Y_MEM	存储器 Y	512	0x0000	0x01FF	R/W
DS_M_MEM	存储器 M	512	0x0200	0x03FF	R/W
DS_RB_MEM	存储器 $\bar{r}$	512	0x0400	0x05FF	R/W
DS_BOX_MEM	存储器 Box	48	0x0600	0x062F	R/W
DS_IV_MEM	存储器 IV	16	0x0630	0x063F	R/W
DS_X_MEM	存储器 X	512	0x0800	0x09FF	R/W
DS_Z_MEM	存储器 Box Z	512	0x0A00	0x0BFF	R/W

21.5 寄存器列表

本小节的所有地址均为相对于数字签名算法基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

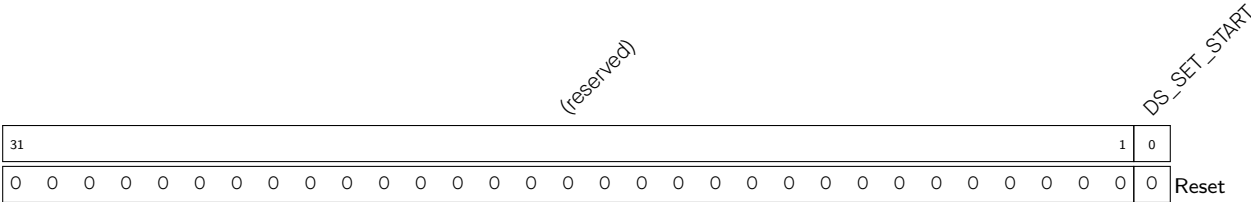
请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
状态/控制寄存器			
DS_SET_START_REG	启动 DSA 模块	0x0E00	WT
DS_SET_CONTINUE_REG	继续 DSA 计算	0x0E04	WT
DS_SET_FINISH_REG	结束计算	0x0E08	WT
DS_QUERY_BUSY_REG	DSA 模块状态	0x0E0C	RO
DS_QUERY_KEY_WRONG_REG	查询 <i>DS_KEY</i> 未准备好的原因	0x0E10	RO
DS_QUERY_CHECK_REG	查询校验结果	0x0E14	RO
配置寄存器			
DS_KEY_SOURCE_REG	配置 DSA 密钥来源	0x0E18	R/W
版本控制寄存器			
DS_DATE_REG	版本控制寄存器	0x0E20	R/W

21.6 寄存器

本小节的所有地址均为相对于数字签名算法基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

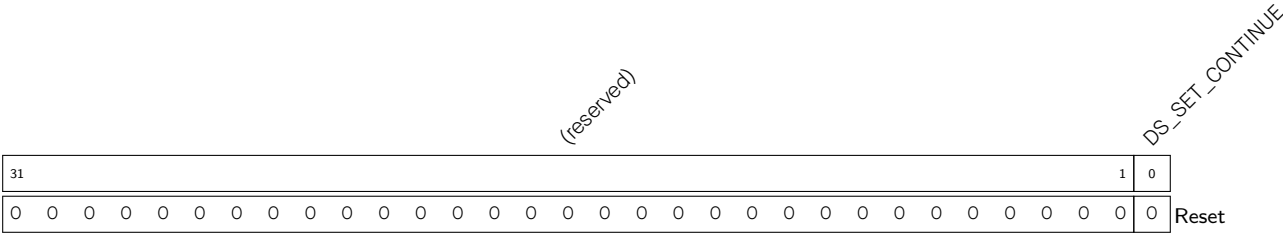
Register 21.1. DS_SET_START_REG (0x0E00)



DS_SET_START 配置是否启动 DSA 模块。

- 0: 无效
- 1: 启动 DSA 模块 (WT)

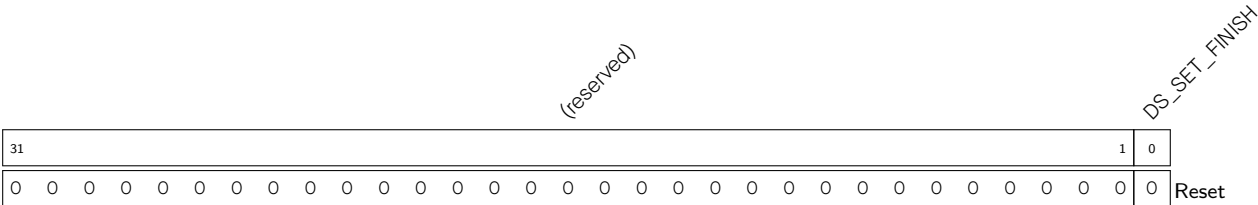
Register 21.2. DS_SET_CONTINUE_REG (0x0E04)



DS_SET_CONTINUE 配置是否继续 DSA 计算。

- 0: 无效
- 1: 继续 DSA 计算 (WT)

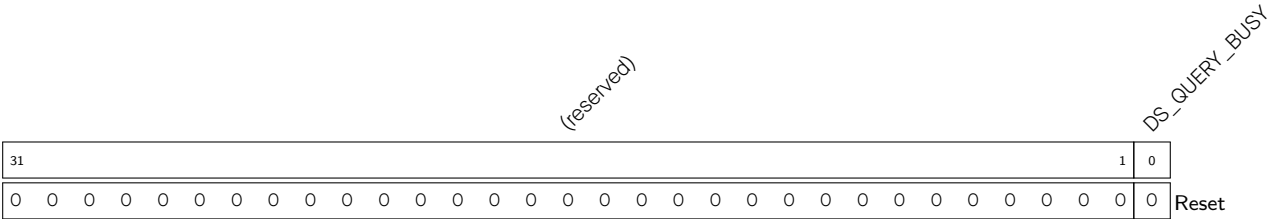
Register 21.3. DS_SET_FINISH_REG (0x0E08)



DS_SET_FINISH 配置是否结束 DSA 计算。

- 0: 无效
- 1: 结束 DSA 计算 (WT)

Register 21.4. DS_QUERY_BUSY_REG (0x0E0C)

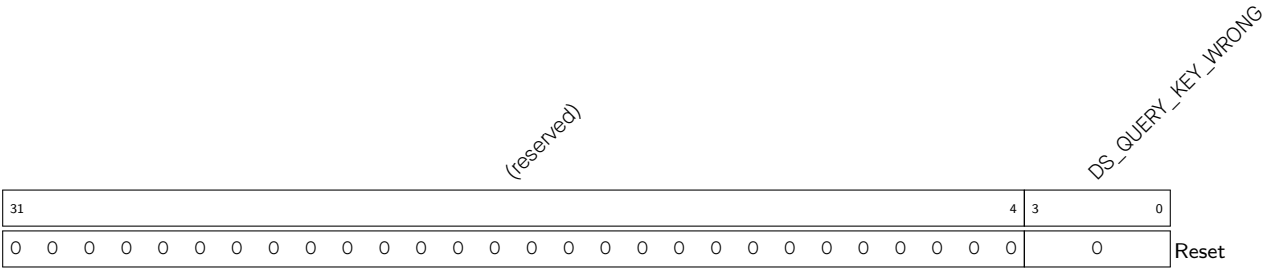


DS_QUERY_BUSY 表示 DSA 模块是否空闲。

- 0: DSA 模块空闲
- 1: DSA 模块正忙

(RO)

Register 21.5. DS_QUERY_KEY_WRONG_REG (0x0E10)



DS_QUERY_KEY_WRONG 表示 HMAC 模块的错误情况。

- 0: HMAC 未被调用
- 1-15: HMAC 曾被调用，DSA 模块没有成功从 HMAC 模块获取到 *DS_KEY*
- 15 以上: 无效

(RO)

Register 21.6. DS_QUERY_CHECK_REG (0x0E14)



DS_MD_ERROR 表示 MD 校验是否通过。

0: MD 校验通过

1: MD 校验失败

(RO)

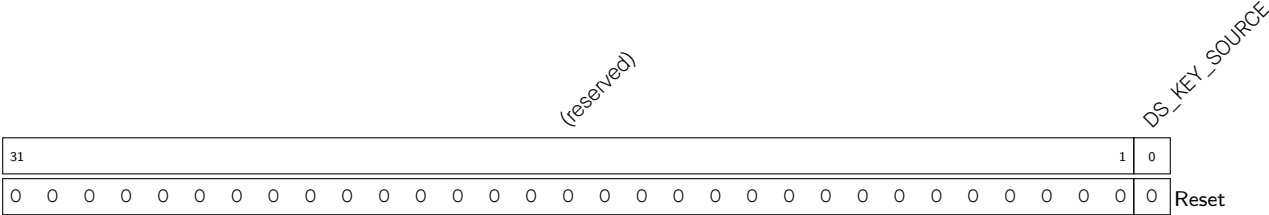
DS_PADDING_BAD 表示填充校验是否通过。

0: 填充校验通过

1: 填充校验失败

(RO)

Register 21.7. DS_KEY_SOURCE_REG (0x0E18)



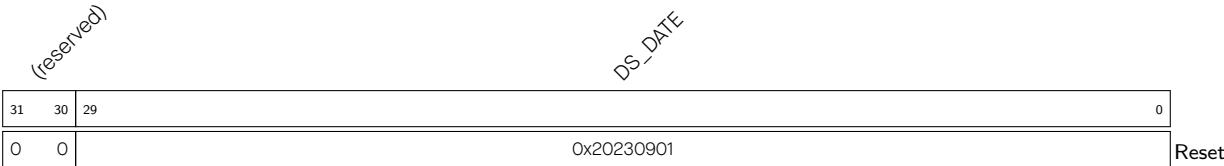
DS_KEY_SOURCE 表示 DSA_KEY 的来源。

0: HMAC

1: 密钥管理器

(R/W)

Register 21.8. DS_DATE_REG (0x0E20)



DS_DATE 版本控制寄存器。(R/W)

第 22 章

椭圆曲线数字签名算法 (ECDSA)

22.1 概述

在密码学中，椭圆曲线数字签名算法 (ECDSA) 是数字签名算法 (DSA) 基于椭圆曲线密码学的一种变体。

ESP32-C5 的 ECDSA 加速器可在安全高效的环境中计算 ECDSA 签名，不仅可保证签名过程的机密性，还实现了快速计算，非常适合对加密安全性和加密速度有高要求的应用。通过 ECDSA 加速器，用户可以在不牺牲性能的情况下保证数据得到保护。

22.2 主要特性

ESP32-C5 的 ECDSA 加速器支持：

- 数字签名的生成和验证
- 三种椭圆曲线，分别是 P-192、P-256 和 P-384（具体定义见 [FIPS 186-5 规范](#)）
- 多种哈希算法，包括 SHA-224、SHA-256、SHA-384、SHA-512、SHA-512-224 和 SHA-512-256（具体定义见 [FIPS PUB 180-4 规范](#)）
- 不同工作状态下的动态访问权限，以确保信息安全

22.3 ECDSA 背景知识

22.3.1 域参数

ECDSA 使用的参数通常被称为域参数，可用于生成密钥、生成签名和验证签名等运算，主要包括有限域中定义的椭圆曲线参数、曲线上的基点和基点阶数等。

ECDSA 加速器中使用的域参数主要包括两大类：

- 椭圆曲线域参数：
 - 质数模数 p ，指定定义椭圆曲线的有限域的大小。
 - 曲线上的基点 G ，用于生成公钥。
 - 基点的阶数 n ，代表在椭圆曲线上对基点 G 进行重复点加的次数。
- 哈希函数参数：
 - 哈希算法，用于将要签名的消息生成固定长度的哈希值。
 - 哈希值的长度，决定签名的大小。

这些参数共同定义了 ECDSA 域。

22.3.2 密钥生成

ECDSA 中的密钥生成过程如下所述：

1. 选择一个椭圆曲线域，定义如下参数：
 - 质数模数 p
 - 系数 a 和 b
 - 基点 G 和其阶数 n
2. 生成私钥：
 - 私钥 d 为在 1 到 $n-1$ 之间随机选择的整数。注意，这里必须确保私钥是真正随机的（不可预测或复制）。因此，使用一个安全的[随机数发生器 \(RNG\)](#)非常重要。
 - 私钥用于生成签名，需妥善保密保管。
 - 私钥选择完成后，应将其烧录到 ESP32-C5 中的 eFuse 中。（请参考第 2 章 [eFuse 控制器 \(eFuse\)](#) [\[to be added later\]](#)）。
3. 计算公钥：
 - 公钥 Q 的计算方式为 $Q = dG$ 。
4. 导出公钥：
 - 公钥 Q 通常以 (Qx, Qy) 表示，可公开给他人。

22.3.3 签名生成

ECDSA 签名生成过程如下所述：

1. 选择一条消息 m ，用于签名。
2. 计算消息 m 的哈希值 e ： $e = \text{HASH}(m)$ ，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息哈希值 e 的摘要 z ：令 z 为 e 的左端 L_n 位，其中 L_n 是基点阶数 n 的位长度。
4. 选择一个随机数 k ：范围在 1 到 $n-1$ 之间，其中 n 是椭圆曲线上基点的阶数。
5. 计算签名：签名计算如下：
 - (a) 计算点 $(x, y) = kG$
 - (b) 计算 $r = x \bmod n$ 。如果 r 等于零，返回上一步并选择新的 k 值。
 - (c) 计算 $s = k^{-1} * (z + d * r) \bmod n$ 。如果 s 等于零，返回步骤 4 并选择新的 k 值。
 - (d) 签名为 (r, s) 。
6. 发送消息 m 和签名 (r, s) 给接收者。

22.3.4 签名验证

接收方可以使用密钥生成阶段导出的公钥，来验证对应私钥生成的签名。验证过程涉及检查签名是否由对应的私钥生成，以及签名是否对给定的消息有效。

ECDSA 签名验证过程如下所述：

1. 获得公钥 Q ，接收消息 m 和签名 (r, s) 。
2. 计算消息的哈希值： e 等于 $\text{HASH}(m)$ ，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息的摘要：令 z 为 e 的左端 L_n 位，其中 L_n 是基点阶数 n 的位长度。
4. 验证签名：签名验证如下所示：
 - (a) 验证 r 和 s 是否是介于 1 到 $n-1$ 之间的整数，其中 n 是椭圆曲线上基点的阶数。如果 r 或 s 超出此范围，则签名无效。
 - (b) 计算 $u_1 = z * s^{-1} \bmod n$ 和 $u_2 = r * s^{-1} \bmod n$ 。
 - (c) 计算点 $(x_1, y_1) = u_1 * G + u_2 * Q$ ，其中 G 是椭圆曲线上的基点， Q 是与用于签名生成的私钥相关联的公钥。
 - (d) 验证 $r = x_1 \bmod n$ 。如果 r 不等于 $x_1 \bmod n$ ，则签名无效。
5. 接受或拒绝签名：如果签名有效，接收方可以确信消息没有被篡改，且消息来自预期的发送方，可以接收。否则，接收方将拒绝并判定为无效消息。

22.4 功能描述

本节描述了 ESP32-C5 的 ECDSA 加速器的详细信息。

22.4.1 ECDSA 工作模式

ESP32-C5 内置的 ECDSA 加速器具有三种工作模式，即签名生成、签名验证和公钥导出模式。

用户可以通过配置 `ECDSA_WORK_MODE`，根据下表 22.4-1 选择 ECDSA 加速器的工作模式。

表 22.4-1. ECDSA 工作模式

ECDSA_WORK_MODE	工作模式
0	签名验证
1	签名生成
2	公钥导出

用户可以通过配置 `ECDSA_ECC_CURVE`，根据下表 22.4-2 选择使用的椭圆曲线。

表 22.4-2. ECDSA 椭圆曲线选择

ECDSA_ECC_CURVE	椭圆曲线
0	P-192
1	P-256
2	P-384

用户可以通过配置 `ECDSA_SHA_MODE`，根据下表 22.4-3 选择用于消息哈希的 SHA 算法。

表 22.4-3. ECDSA SHA 算法

ECDSA_SHA_MODE	SHA 算法
1	SHA-224
2	SHA-256
3	SHA-384
4	SHA-512
5	SHA-512-224
6	SHA-512-256
其他	无效

此外，用户可以通过查询 `ECDSA_STATE_REG` 寄存器并将返回值与下表 22.4-4 进行比较，来确认 ECDSA 加速器的工作状态。

表 22.4-4. ECDSA 工作状态

ECDSA_STATE_REG	状态	描述
0	IDLE	空闲或已完成的运算，对应于 IDLE 阶段。
1	LOAD	等待用户将信息加载到 ECDSA 中，对应于 LOAD 阶段。
2	GAIN	等待用户从 ECDSA 获取信息，对应于 GAIN 阶段。
3	BUSY	硬件正在运行中，对应于 PREP、PROC 和 POST 阶段。

22.4.2 数据和数据块

ESP32-C5 的 ECDSA 加速器会用到的数据位宽为 192, 256, 384 或 512 位。

假设一个数据为 $D[255:0]$ ，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n = 0, 1, \dots, 7)$ ，序号数低的数据块对应二进制低位。以 256 位的数据为例：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

22.4.2.1 数据写入

数据写入即将数据写入进一个内存块的过程，该数据为 ECDSA 算法的输入数据。具体而言，将数据写入一个 ECDSA 内存块相当于将该数据 $D[n][31:0]$ 依次写入“该内存块起始地址 + $4 \times n$ ”。以 256 位的数据为例：

- 将 $D[0]$ 写入“起始地址”
- 将 $D[1]$ 写入“起始地址 + 4”
- ...
- 将 $D[7]$ 写入“起始地址 + 28”

说明：

进行数据写入操作时，只需要写入数据长度的数据块即可，不需要在数据的高位补 0。例如，在写入 192 位长度的数据时，不需要在数据的高位补 0，只需要存储 192 位的数据。

22.4.2.2 数据读取

数据读取即将一个数据从一个内存块读出的过程，该数据为 ECDSA 算法的输出数据。具体来说，从一个 ECDSA 内存块读数据相当于从“该内存块起始地址 + $4 \times n$ ”依次读出 $D[n][31:0]$ 。以 256 位的数据为例：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- ...
- 从“起始地址 + 28”读出 $D[7]$

说明：

进行数据读取操作时，只需要读取数据长度的数据块即可，例如读取 192 位长度的数据时，只需要读取低 192 位（即 6 个数据块）的数据。

22.4.2.3 消息填充

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件运算将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 L_M 位，则填充步骤如下：

1. 首先，在待处理信息后填充 1 个“1”；
2. 随后，再填充 L_A 个“0”。其中， L_A 为满足 $L_M + 1 + L_A \equiv 448 \pmod{512}$ 的最小非负数解；
3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 L_M 的值。

更多详情，请参考 [《FIPS PUB 180-4 规范》](#) > 章节“Padding the Message”。

22.4.2.4 解析消息

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$ 、 $M^{(2)}$ 、...、 $M^{(N)}$ 。

说明：

1. 有关“解析消息”的详细信息，请参考 [FIPS PUB 180-4 规范](#) > 章节“Parsing the Message”。
2. 有关“信息块”及相关概念的描述，请参考 [FIPS PUB 180-4 规范](#) > 章节“Glossary of Terms and Acronyms”。

22.4.3 安全功能

为确保 ECDSA 运算过程的安全性，ECDSA 加速器实现了多种安全功能。

22.4.3.1 抗攻击性能

ESP32-C5 ECDSA 加速器每次执行操作时，都会利用 ECC 加速器的强抗攻击性能（参考小节 17.4.3 提升抗攻击性能）。这意味着 ECDSA 在每次生成和验证签名时：

- 都会消耗相同的时间；
- 都会消耗相同的功耗。

这为 ESP32-C5 的 ECDSA 加速器提供了强大的抗攻击性能。

22.4.3.2 动态访问权限

ESP32-C5 的 ECDSA 加速器实施了动态访问权限机制，可有效防止运行过程中可能发生的非法配置篡改或数据访问，从而保证密钥的安全性。

具体来说，ECDSA 的寄存器在不同工作状态下的访问权限是不同的。例如，`ECDSA_CONF_REG` 寄存器仅在加速器处于 IDLE 状态时允许读写运算。这样一来，当加速器处于其他状态（如 LOAD、GAIN 和 BUSY）时，该寄存器中的配置信息则不允许读取或写入，从而得到很好的保护。有关 ECDSA 所有工作状态的详细信息，请见表 22.4-4。

有关每个 ECDSA 寄存器动态访问权限的详细信息，请参阅[寄存器列表](#)。

22.4.3.3 硬件占用

在 ECDSA 运算过程中，ESP32-C5 的 ECDSA 加速器将占用以下硬件模块：

- SHA 加速器
- ECC 加速器

其中，SHA 加速器在触发 `ECDSA_SHA_RELEASE_INT` 时将释放，而 ECC 加速器在整个 ECDSA 运算中将被占用。

说明：

硬件占用是一种保护复用模块和存储空间机制。当一个模块被硬件占用时，用户将无法执行以下运算：

- 读取或写入模块的寄存器或存储器数据。
- 关闭模块时钟。
- 复位模块。

在硬件占用结束时，占用的模块将自动复位。此外，当用户对主模块执行软件复位时，所有占用的模块将同时复位。

22.5 配置流程

22.5.1 ECDSA 流程

ECDSA 的完整工作流程包括以下六个阶段。

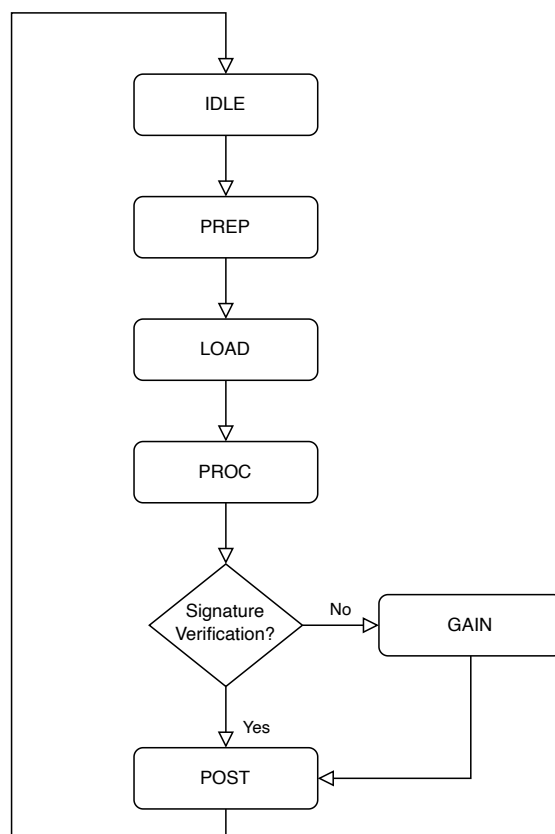


图 22.5-1. ECDSA 流程

每个阶段的详细配置流程在下文中描述。

22.5.1.1 IDLE 阶段

在 IDLE 阶段：

1. 配置静态参数，包括 eFuse 位：
 - (a) ECDSA_KEY: ECDSA 中的私钥 d 的值。为了正确配置 eFuse 中的密钥值，用户需要将密钥值写入 KEY n ($n = 0-5$)，并将相应的 EFUSE_KEY_PURPOSE_ n 设置为 ECDSA_KEY。有关更详细的配置步骤，请参阅第 2 章 [eFuse 控制器 \(eFuse\) \[to be added later\]](#)。
2. 配置 ECDSA_CONF_REG，包括以下字段：
 - (a) ECDSA_WORK_MODE: 选择 ECDSA 加速器的工作模式。
 - (b) ECDSA_ECC_CURVE: 选择 ECDSA 加速器的椭圆曲线。
 - (c) ECDSA_SOFTWARE_SET_Z: 选择是否使用直接输入的 z 。
 - (d) ECDSA_DETERMINISTIC_K: 选择是否使用确定性的 k 。
3. 配置寄存器字段 ECDSA_START 以进入 PREP 阶段。

说明：

更多有关“确定性 ECDSA” (Deterministic ECDSA) 的信息，请参考 [RFC6979](#)。

22.5.1.2 PREP 阶段

在 PREP 阶段, `ECDSA_STATE_REG` 为 BUSY, ECDSA 加速器执行准备工作。

通过轮询 `ECDSA_BUSY` 的状态直到其不为 BUSY, 表示 PREP 阶段已结束。然后 ECDSA 加速器将自动进入 LOAD 阶段。

22.5.1.3 LOAD 阶段

在 LOAD 阶段:

1. 使用以下选项之一将输入 z 提供给 ECDSA 加速器:
 - 直接输入 z : 将 z 写入 `ECDSA_MEM_Z`。
 - ECDSA-SHA 接口: 从消息生成 z 。有关详细信息, 请参阅第 22.5.1.7 节。
2. 根据所选的 `ECDSA_WORK_MODE`, 进行以下配置:
 - 签名验证:
 - (a) 将签名 (r, s) 写入 `ECDSA_MEM_R` 和 `ECDSA_MEM_S`。
 - (b) 将公钥 (Q_x, Q_y) 写入 `ECDSA_MEM_Qx` 和 `ECDSA_MEM_Qy`。
 - 签名生成:
 - (a) 不需要其他配置。
 - 公钥导出:
 - (a) 不需要其他配置。
3. 将 1 写入 `ECDSA_LOAD_DONE`, 表示配置完成。然后加速器将自动进入 PROC 阶段。

22.5.1.4 PROC 阶段

在 PROC 阶段, `ECDSA_STATE_REG` 为 BUSY, ECDSA 加速器根据所选的工作模式执行 ECDSA 运算。

通过轮询 `ECDSA_BUSY` 直到其不为 BUSY, 表示 PROC 阶段已结束。然后 ECDSA 加速器将根据所选的工作模式自动进入 GAIN 阶段或 POST 阶段:

- 签名生成: GAIN 阶段
- 签名验证: POST 阶段

22.5.1.5 GAIN 阶段

当选择了签名生成或公钥导出模式时, ECDSA 加速器在 PROC 阶段之后进入 GAIN 阶段:

1. 从 ECDSA 内存中读取数据:
 - 从 `ECDSA_MEM_R` 和 `ECDSA_MEM_S` 读取签名 (r, s) , 仅在签名生成模式下。
 - 从 `ECDSA_MEM_Qx` 和 `ECDSA_MEM_Qy` 读取公钥 (Q_x, Q_y) , 在签名生成或公钥导出模式下。
2. 向 `ECDSA_GET_DONE` 写入 1, 表示 GAIN 阶段已完成。然后加速器将自动进入 POST 阶段。

22.5.1.6 POST 阶段

在 POST 阶段，`ECDSA_STATE_REG` 为 BUSY，ECDSA 加速器执行 ECDSA 运算的一些收尾工作。

通过轮询 `ECDSA_BUSY` 直到不再为 BUSY，表示 POST 阶段已结束。然后 ECDSA 加速器将自动返回 IDLE 阶段。

22.5.1.7 ECDSA SHA 接口

ESP32-C5 的 ECDSA 加速器可以自动执行哈希运算并基于直接输入的消息生成 z 。

对于消息哈希，ECDSA 加速器支持 SHA 算法 SHA-224（仅当选择 P-192 作为椭圆曲线时有效）和 SHA-256。

要使用 ECDSA SHA 接口，请完成以下步骤：

1. 根据第 22.4.2.3 节中的描述，对消息进行填充。
2. 将消息及其填充解析为消息块。详细信息请参见第 22.4.2.4 节。
3. 处理当前消息块。
 - 将当前消息块写入 `ECDSA_MEM_M`。
4. 启动 ECDSA SHA 接口¹。
 - 如果这是第一次执行此步骤，请将 1 写入 `ECDSA_SHA_START` 以启动 ECDSA SHA 接口；
 - 如果不是第一次执行此步骤²，请将 1 写入 `ECDSA_SHA_CONTINUE` 以继续运算。
5. 查询当前信息块的处理进度。
 - 轮询寄存器 `ECDSA_SHA_BUSY`，直到其为 IDLE，表示接口已完成当前消息块的运算。
6. 处理下一个消息块：
 - 如有下一个消息块，则返回到步骤 3。
 - 如没有更多的消息块，退出。

说明：

1. 在此步骤中，软件还可以在接口启动 SHA 运算时将下一个要处理的消息块（如有）写入寄存器 `ECDSA_MEM_M`，以节省时间。
2. 即正在继续之前暂停的 ECDSA SHA 接口运算。

22.5.2 时钟

ECDSA 模块有两种时钟：

- `clk_ecdsa`：ECDS 模块运行所使用的功能时钟
- `clk_apb_ecdsa`：用于配置 ECDSA 模块寄存器的总线时钟

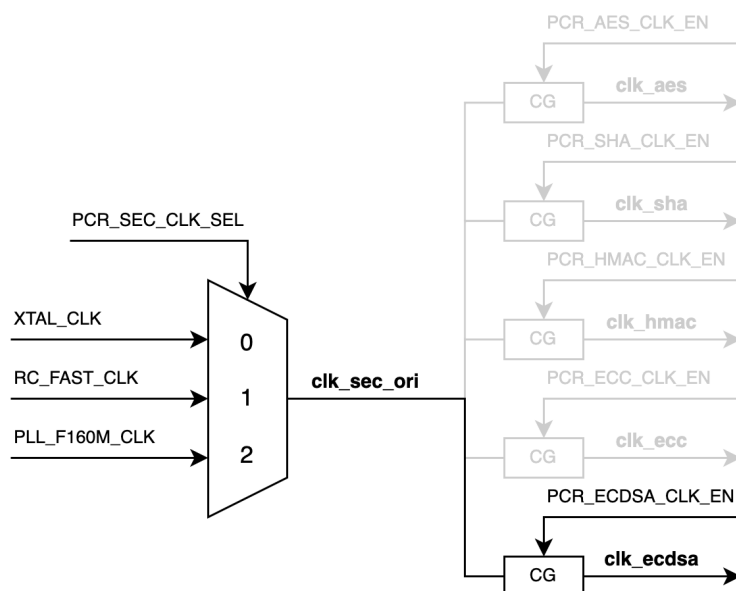


图 22.5-2. clk_ecdsa 时钟图

如图 22.5-2 *clk_ecdsa* 时钟图 所示，安全功能源时钟 *clk_sec_ori* 有以下时钟源：

- XTAL_CLK
- RC_FAST_CLK
- PLL_F160M_CLK

ECDSA 模块功能时钟 *clk_ecdsa* 由时钟门控寄存器 *PCR_ECDSA_CLK_EN* 控制，当该寄存器值为 1 时，时钟门控开启，使能 *clk_ecdsa*。

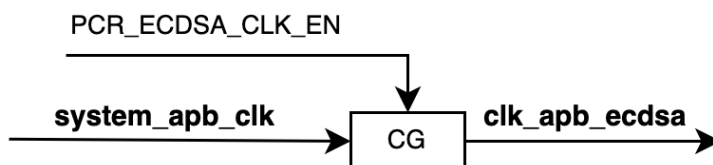


图 22.5-3. clk_apb_ecdsa 时钟图

如图 22.5-3 *clk_apb_ecdsa* 时钟图 所示，ECDSA 模块配置时钟 *clk_apb_ecdsa* 的时钟源为 SYSTEM_APB_CLK。

ECDSA 模块配置时钟 *clk_apb_ecdsa* 由时钟门控寄存器 *PCR_ECDSA_CLK_EN* 控制，当该寄存器值为 1 时，时钟门控开启，使能 *clk_apb_ecdsa*。

有关时钟的更多信息，请查看章节 6 [复位和时钟](#)。

22.5.3 复位

ECDSA 模块支持完全复位，向 *PCR_ECDSA_RST_EN* 先写 1 后写 0 使能，可复位整个 ECDSA 模块。

22.6 中断

ESP32-C5 的 ECDSA 加速器可以生成一个中断信号 ECDSA_INTR 并发送给 [中断矩阵](#)。

ECDSA 加速器有四个中断源可产生中断信号 ECDSA_INTR，如下所示。每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。

- ECDSA_PREP_DONE_INT：在 PREP 阶段结束时触发。可以通过以下寄存器配置此中断源：
 - ECDSA_PREP_DONE_INT_RAW：存储 ECDSA_PREP_DONE_INT 的原始中断状态。
 - ECDSA_PREP_DONE_INT_ST：代表 ECDSA_PREP_DONE_INT 中断的状态。通过 ECDSA_PREP_DONE_INT_RAW 字段启用/禁用此字段生成。
 - ECDSA_PREP_DONE_INT_ENA：启用/禁用 ECDSA_PREP_DONE_INT 中断。
 - ECDSA_PREP_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_PREP_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_PREP_DONE_INT_RAW 和 ECDSA_PREP_DONE_INT_ST 将被清除。
- ECDSA_PROC_DONE_INT：在 PROC 阶段结束时触发。可以通过以下寄存器配置此中断源：
 - ECDSA_PROC_DONE_INT_RAW：存储 ECDSA_PROC_DONE_INT 的原始中断状态。
 - ECDSA_PROC_DONE_INT_ST：代表 ECDSA_PROC_DONE_INT 中断的状态。通过 ECDSA_PROC_DONE_INT_RAW 字段启用/禁用此字段生成。
 - ECDSA_PROC_DONE_INT_ENA：启用/禁用 ECDSA_PROC_DONE_INT 中断。
 - ECDSA_PROC_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_PROC_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_PROC_DONE_INT_RAW 和 ECDSA_PROC_DONE_INT_ST 将被清除。
- ECDSA_POST_DONE_INT：在 POST 阶段结束时触发。可以通过以下寄存器配置此中断源：
 - ECDSA_POST_DONE_INT_RAW：存储 ECDSA_POST_DONE_INT 的原始中断状态。
 - ECDSA_POST_DONE_INT_ST：代表 ECDSA_POST_DONE_INT 中断的状态。通过 ECDSA_POST_DONE_INT_RAW 字段启用/禁用此字段生成。
 - ECDSA_POST_DONE_INT_ENA：启用/禁用 ECDSA_POST_DONE_INT 中断。
 - ECDSA_POST_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_POST_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_POST_DONE_INT_RAW 和 ECDSA_POST_DONE_INT_ST 将被清除。
- ECDSA_SHA_RELEASE_INT：在 SHA 释放时触发。可以通过以下寄存器配置此中断源：
 - ECDSA_SHA_RELEASE_INT_RAW：存储 ECDSA_SHA_RELEASE_INT 的原始中断状态。
 - ECDSA_SHA_RELEASE_INT_ST：代表 ECDSA_SHA_RELEASE_INT 中断的状态。通过 ECDSA_SHA_RELEASE_INT_RAW 字段启用/禁用此字段生成。
 - ECDSA_SHA_RELEASE_INT_ENA：启用/禁用 ECDSA_SHA_RELEASE_INT 中断。
 - ECDSA_SHA_RELEASE_INT_CLR：将此位设置为 1 以清除 ECDSA_SHA_RELEASE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_SHA_RELEASE_INT_RAW 和 ECDSA_SHA_RELEASE_INT_ST 将被清除。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 [3 中断矩阵 \[to be added later\]](#) > 3.1 中断术语。

22.7 存储器块

表 22.7-1. ECDSA 存储器块

存储器块	大小 (字节)	起始地址 [*]	结束地址 [*]	访问权限
ECDSA_MEM_M	128	0x280	0x2FF	R/W
ECDSA_MEM_R	48	0x3E0	0x40F	R/W
ECDSA_MEM_S	48	0x410	0x43F	R/W
ECDSA_MEM_Z	48	0x440	0x46F	R/W
ECDSA_MEM_Qx	48	0x470	0x49F	R/W
ECDSA_MEM_Qy	48	0x4A0	0x4CF	R/W

^{*} 相对于 ECDSA 加速器基地址的地址偏移在第 4 章[系统和存储器](#)中的表 4.3-2 中提供。

22.8 寄存器列表

本小节的所有地址均为相对于 ECDSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

为了提高安全性，在不同工作状态下，ECDSA 寄存器的读写权限不同。以下是每个工作状态的缩写及其与阶段的对应关系：

- PI: IDLE 状态，对应 IDLE 阶段。
- PL: LOAD 状态，对应 LOAD 阶段。
- PG: GAIN 状态，对应 GAIN 阶段。
- PB: BUSY 状态，对应 PREP、PROC 和 POST 阶段。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限			
			PI	PL	PG	PB
数据存储	见表 22.7-1。					
配置寄存器						
ECDSA_CONF_REG	ECDSA 配置寄存器	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA 启动寄存器	0x001C	WT			N/A
时钟和复位寄存器						
ECDSA_CLK_REG	ECDSA 时钟门控寄存器	0x0008	R/W	N/A		
中断寄存器						
ECDSA_INT_RAW_REG	ECDSA 中断原始状态寄存器	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA 中断状态寄存器	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA 中断使能寄存器	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA 中断清除寄存器	0x0018	WT			
状态寄存器						
ECDSA_STATE_REG	ECDSA 状态寄存器	0x0020	RO			
结果寄存器						
ECDSA_RESULT_REG	ECDSA 结果寄存器	0x0024	RO/SS			N/A
SHA 寄存器						
ECDSA_SHA_MODE_REG	ECDSA 控制 SHA 寄存器（哈希算法）	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA 控制 SHA 寄存器（运算）	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA 控制 SHA 寄存器（运算）	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA 控制 SHA 状态寄存器	0x0218	N/A	RO	N/A	
版本寄存器						
ECDSA_DATE_REG	版本控制寄存器	0x00FC	R/W	N/A		

22.9 寄存器

本小节的所有地址均为相对于 ECDSA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 22.1. ECDSA_CONF_REG (0x0004)

(reserved)															ECDSA_DETERMINISTIC_K			
															ECDSA_SOFTWARE_SET_Z			
															(reserved)			
															ECDSA_ECC_CURVE			
															ECDSA_WORK_MODE			
31							7	6	5	4	3	2	1	0				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

ECDSA_WORK_MODE 配置 ECDSA 加速器的工作模式。

- 0: 签名验证模式
- 1: 签名生成模式
- 2: 公钥导出模式
- 3: 无效

(R/W)

ECDSA_ECC_CURVE 配置使用的椭圆曲线。

- 0: P-192
- 1: P-256
- 2: P-384
- 3: 无效

(R/W)

ECDSA_SOFTWARE_SET_Z 配置参数 z 的配置方式。

- 0: SHA 结果生成
- 1: 软件写入

(R/W)

ECDSA_DETERMINISTIC_K 配置参数 k 的配置方式。

- 0: TRNG 自动生成
- 1: 由确定性推导算法生成

(R/W)

Register 22.2. ECDSA_START_REG (0x001C)

(reserved)																																ECDSA_GET_DONE ECDSA_LOAD_DONE ECDSA_START																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ECDSA_START 配置是否启动 ECDSA 运算。此位配置后自动清 0。

- 0: 无效
- 1: 启动 ECDSA 运算
- (WT)

ECDSA_LOAD_DONE 写 1 生成中断信号，表示 ECDSA 加速器的 LOAD 阶段已完成。此位配置后自动清 0。(WT)

ECDSA_GET_DONE 写 1 生成中断信号，表示 ECDSA 加速器的 GAIN 阶段已完成。此位配置后自动清 0。(WT)

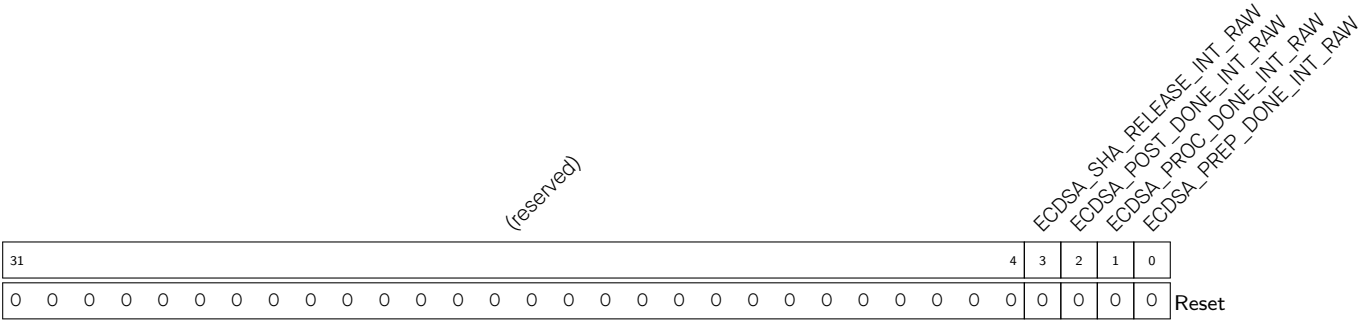
Register 22.3. ECDSA_CLK_REG (0x0008)

(reserved)																																ECDSA_CLK_GATE_FORCE_ON																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ECDSA_CLK_GATE_FORCE_ON 配置是否强制启动 ECDSA 内存的时钟门控。

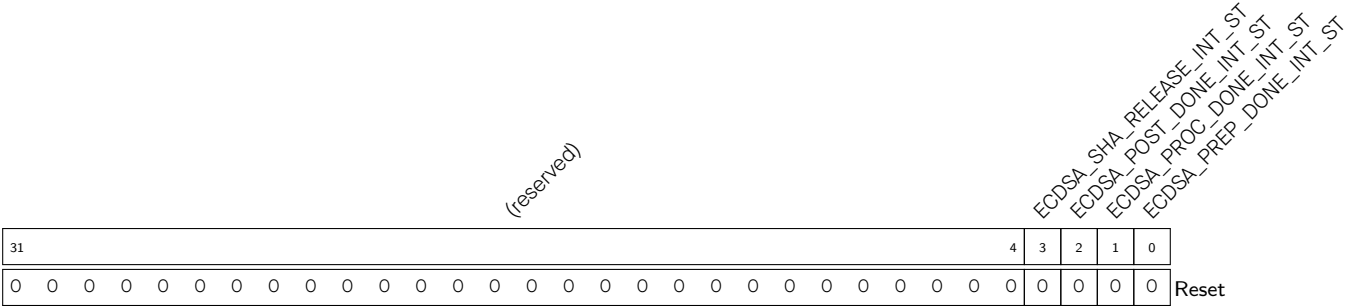
- 0: 无效
- 1: 强制启动
- (R/W)

Register 22.4. ECDSA_INT_RAW_REG (0x000C)



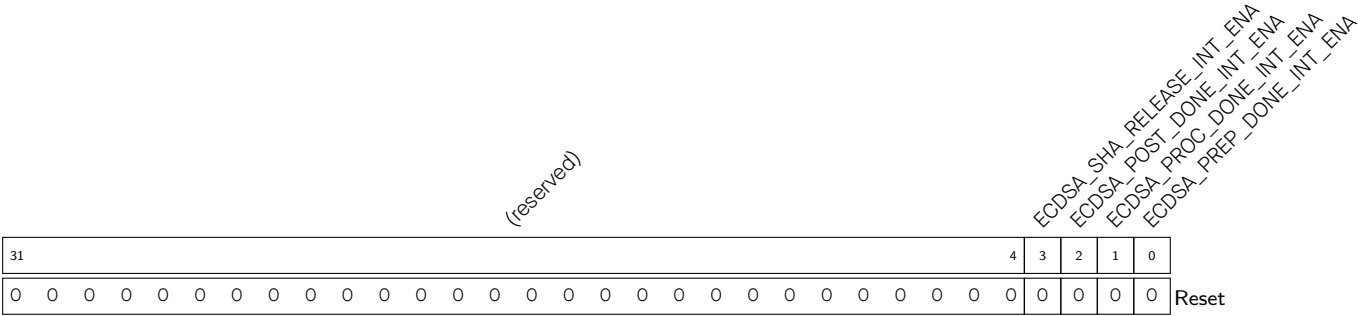
- ECDSA_PREP_DONE_INT_RAW ECDSA_PREP_DONE_INT 的原始中断状态。(R/SS/WTC)
- ECDSA_PROC_DONE_INT_RAW ECDSA_PROC_DONE_INT 的原始中断状态。(R/SS/WTC)
- ECDSA_POST_DONE_INT_RAW ECDSA_POST_DONE_INT 的原始中断状态。(R/SS/WTC)
- ECDSA_SHA_RELEASE_INT_RAW ECDSA_SHA_RELEASE_INT 的原始中断状态。(R/SS/WTC)

Register 22.5. ECDSA_INT_ST_REG (0x0010)



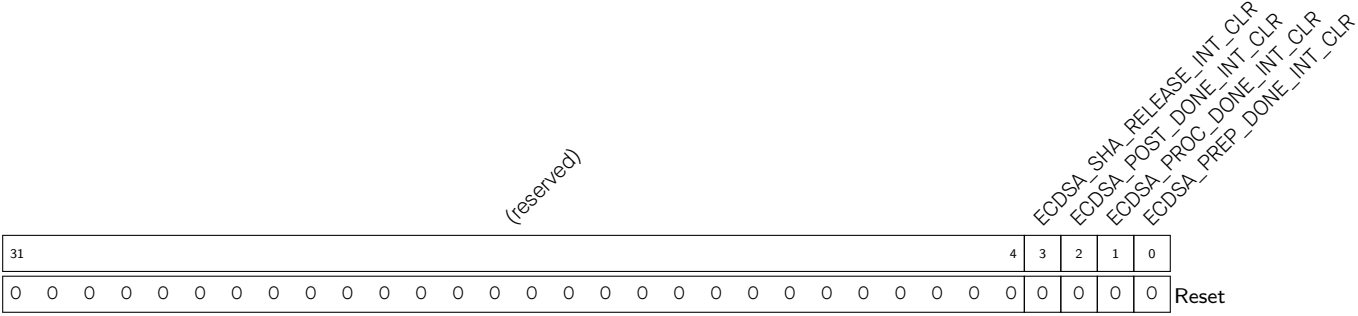
- ECDSA_PREP_DONE_INT_ST ECDSA_PREP_DONE_INT 的屏蔽中断状态。(RO)
- ECDSA_PROC_DONE_INT_ST ECDSA_PROC_DONE_INT 的屏蔽中断状态。(RO)
- ECDSA_POST_DONE_INT_ST ECDSA_POST_DONE_INT 的屏蔽中断状态。(RO)
- ECDSA_SHA_RELEASE_INT_ST ECDSA_SHA_RELEASE_INT 的屏蔽中断状态。(RO)

Register 22.6. ECDSA_INT_ENA_REG (0x0014)



- ECDSA_PREP_DONE_INT_ENA 写 1 使能 ECDSA_PREP_DONE_INT 中断。(R/W)
- ECDSA_PROC_DONE_INT_ENA 写 1 使能 ECDSA_PROC_DONE_INT 中断。(R/W)
- ECDSA_POST_DONE_INT_ENA 写 1 使能 ECDSA_POST_DONE_INT 中断。(R/W)
- ECDSA_SHA_RELEASE_INT_ENA 写 1 使能 ECDSA_SHA_RELEASE_INT 中断。(R/W)

Register 22.7. ECDSA_INT_CLR_REG (0x0018)



- ECDSA_PREP_DONE_INT_CLR 写 1 清除 ECDSA_PREP_DONE_INT 中断。(WT)
- ECDSA_PROC_DONE_INT_CLR 写 1 清除 ECDSA_PROC_DONE_INT 中断。(WT)
- ECDSA_POST_DONE_INT_CLR 写 1 清除 ECDSA_POST_DONE_INT 中断。(WT)
- ECDSA_SHA_RELEASE_INT_CLR 写 1 清除 ECDSA_SHA_RELEASE_INT 中断。(WT)

Register 22.8. ECDSA_STATE_REG (0x0020)

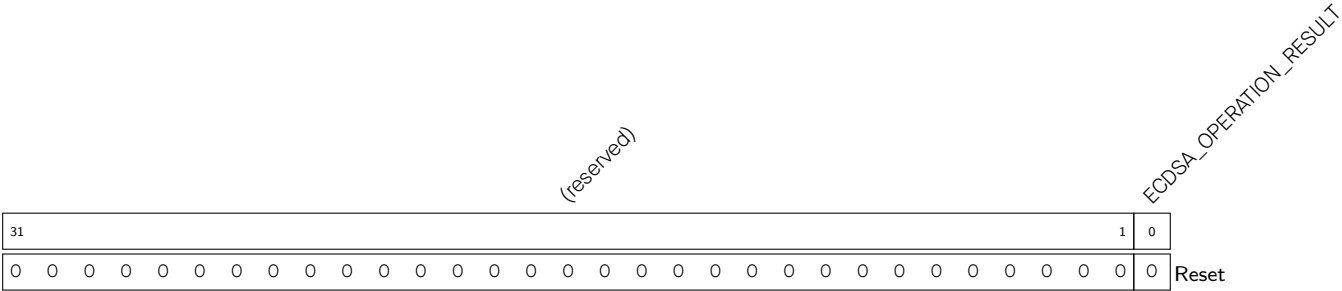


ECDSA_BUSY 表示 ECDSA 加速器的工作状态。

- 0: IDLE
- 1: LOAD
- 2: GAIN
- 3: BUSY

(RO)

Register 22.9. ECDSA_RESULT_REG (0x0024)



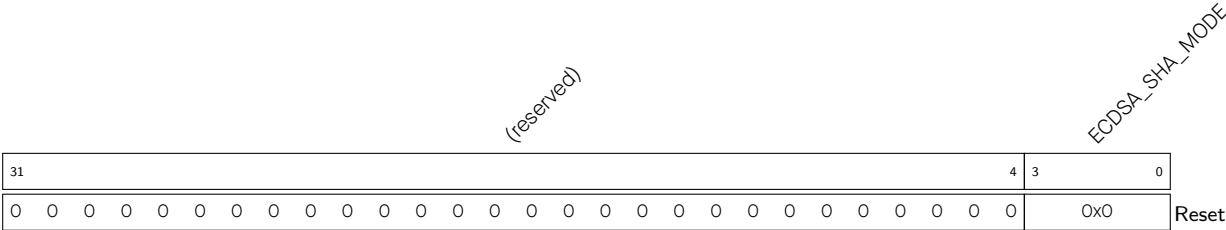
ECDSA_OPERATION_RESULT 表示 ECDSA 运算是否成功。

- 0: 失败
- 1: 成功

仅在 ECDSA 运算完成后有效。

(RO/SS)

Register 22.10. ECDSA_SHA_MODE_REG (0x0200)

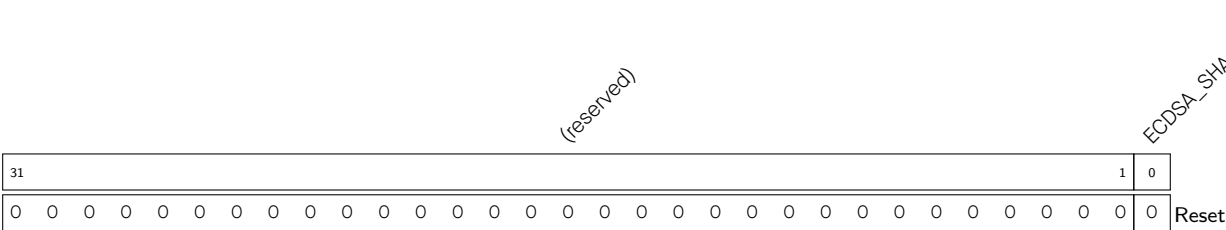


ECDSA_SHA_MODE 配置哈希运算的 SHA 算法。

- 1: SHA-224
- 2: SHA-256
- 3: SHA-384
- 4: SHA-512
- 5: SHA-512-224
- 6: SHA-512-256
- 其他: 无效

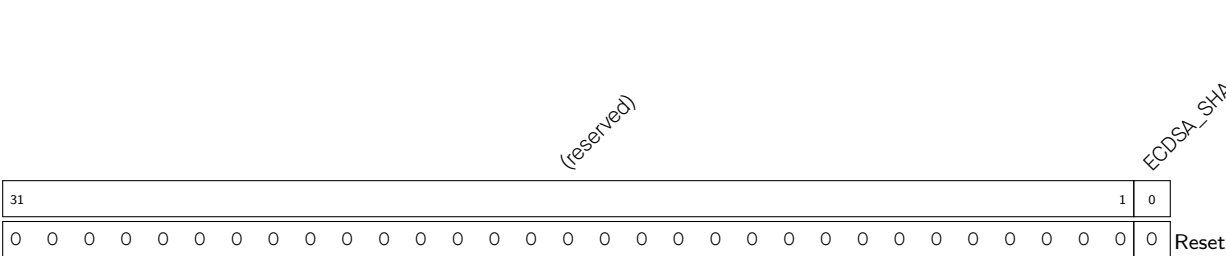
(R/W)

Register 22.11. ECDSA_SHA_START_REG (0x0210)



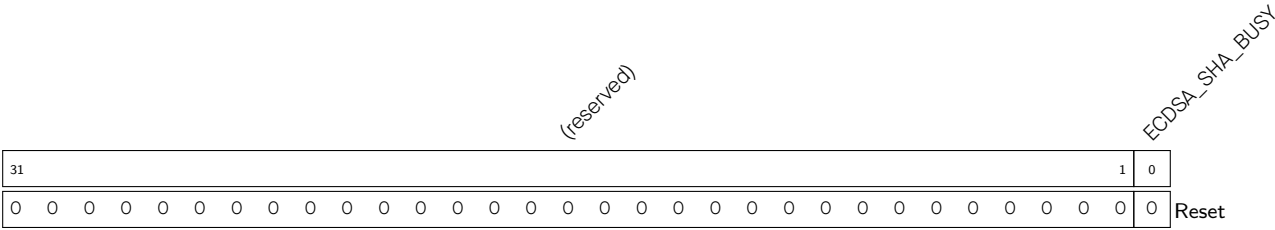
ECDSA_SHA_START 写 1 启动 ECDSA 运算中的首个 SHA 运算。此位配置后自动清 0。(WT)

Register 22.12. ECDSA_SHA_CONTINUE_REG (0x0214)



ECDSA_SHA_CONTINUE 写 1 启动 ECDSA 运算中的后续 SHA 运算。此位配置后自动清 0。(WT)

Register 22.13. ECDSA_SHA_BUSY_REG (0x0218)



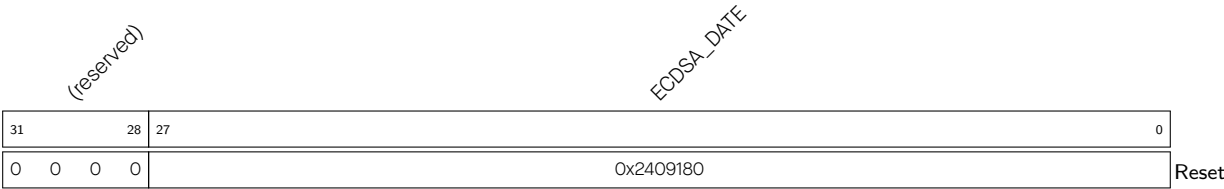
ECDSA_SHA_BUSY 表示 ECDSA 运算过程中 SHA 加速器的工作状态。

0: IDLE

1: BUSY

(RO)

Register 22.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE ECDSA 版本控制寄存器。(R/W)

第 23 章

片外存储器加密与解密 (XTS_AES)

23.1 概述

ESP32-C5 芯片集成了片外存储器加密与解密模块，使用 [IEEE Std 1619-2007](#) 指定的 XTS-AES 标准算法，为用户存放在片外存储器（flash 和 RAM）的应用代码和数据提供了安全保障。用户可以将专有固件、敏感的用户数据（如用来访问私有网络的证书）存放在片外 flash 中，或将一般数据存放在片外 RAM 中。

23.2 主要特性

- 使用通用 XTS-AES 算法，符合 [IEEE Std 1619-2007](#)
- 支持手动加密，需要软件参与
- 支持高速自动加解密，无需软件参与
- 由寄存器配置、eFuse 参数、启动 (boot) 模式共同决定开启/关闭加解密功能
- 支持可配置的抗 DPA 攻击功能

23.3 模块结构

片外存储器加解密模组包含三个模块：手动加密 (Manual Encryption) 模块、自动加密 (Auto Encryption) 模块和自动解密 (Auto Decryption) 模块。模块结构如图 23.3-1 所示。

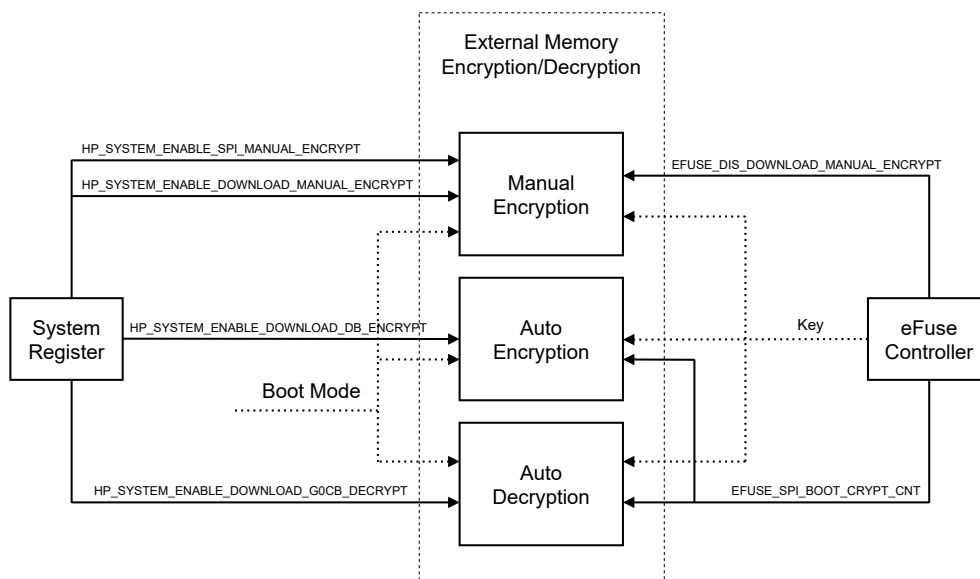


图 23.3-1. 片外存储器加解密结构

手动加密模块能够对指令和数据进行加密，加密后的指令/数据随即以密文状态由 SPI1 写入片外 flash。

当 CPU 通过 cache 向片外 RAM 写入数据时，自动加密模块会首先自动加密数据，然后将数据作为密文写入片外 RAM。

当 CPU 通过高速 cache 从片外 flash 或片外 RAM 读取数据时，自动解密模块将自动解密密文，从而检索指令和数据。

系统寄存器外设中（请参见章节 5 系统寄存器 [\[to be added later\]](#)），寄存器

HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 内与片外存储器加解密相关的有以下 3 位：

- HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT
- HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT
- HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

此外，片外存储器加解密模组还会从外设 eFuse 控制器中获取两个参数：

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 和 EFUSE_SPI_BOOT_CRYPT_CNT。更多详细信息，请参考章节 2 eFuse 控制器 (eFuse) [\[to be added later\]](#)。

23.4 功能描述

23.4.1 XTS 算法

手动加密模块和自动解密模块均使用 XTS 算法。具体实现中，XTS 算法使用 1024 位为一个数据单元 (data unit)，此处的“数据单元”由 XTS-AES Tweakable Block Cipher 标准中的章节 XTS-AES encryption procedure 定义。更多关于 XTS-AES 算法的信息，请参考 [IEEE Std 1619-2007](#)。

23.4.2 密钥

执行 XTS 算法时，手动加密模块、自动加密模块和自动解密模块共用同一密钥 Key 。密钥 Key 来自硬件 eFuse，软件无法访问获取。

Key 的长度为 256 位。 Key 的值由 BLOCK4 ~ BLOCK9 中一个 eFuse 块的参数信息决定。为方便阐述如何通过 eFuse 参数信息推导出 Key 的值，现定义：

- Block_A：BLOCK4 ~ BLOCK9 中密钥用途为 EFUSE_KEY_PURPOSE_XTS_AES_128_KEY 的 BLOCK（请参考章节 2 eFuse 控制器 (eFuse) [to be added later] 中的表 2 eFuse 控制器 (eFuse) [to be added later]）。如果 Block_A 存在，那么 Block_A 中存放着 256 位的 Key_A 。

根据 Block_A 是否存在，有两种对应的生成 Key 的方式。任意情况下， Key 值都唯一确定，如表 23.4-1 所示。

表 23.4-1. 根据 Key_A 生成的 Key 值

Block _A 是否存在	Key	Key 长度 (位)
是	Key_A	256
否	0^{256}	256

说明：

- “ 0^{256} ” 表示由 256 个位 0 组成的位串。
- 使用 0^{256} 作为密钥 Key 存在风险，建议配置有效密钥。

更多有关密钥用途的信息，请参考章节 2 eFuse 控制器 (eFuse) [to be added later] 中的表 2 eFuse 控制器 (eFuse) [to be added later]。

23.4.3 目标空间

目标空间指的是片外存储器中存放首次加密密文的一段连续地址空间。目标空间可由类型、目标大小和目标基地址唯一确定。这三个参数的定义如下：

- 目标类型：目标空间的类型 (*type*)，可为片外 flash 或片外 RAM。值为 0 时代表片外 flash，值为 1 则代表片外 RAM。
- 目标大小：目标空间的大小 (*size*)，即单次加密的数据量。以字节为单位，支持 16 字节、32 字节或 64 字节。
- 目标基地址：目标空间的基地址 (*base_addr*)。目标基地址为 24 位的物理地址，取值范围为 $0x0000\_0000 \sim 0x00FF\_FFFF$ ，但要求以 *size* 为单位对齐，即 $base\_addr \% size == 0$ 。

例如，在一次加密操作中，需要将 16 字节的指令数据加密后写入片外 flash 中的地址段 $0x130 \sim 0x13F$ ，则目标空间为 $0x130 \sim 0x13F$ ，目标类型为 0（外部 flash），目标大小为 16（字节），目标基地址为 $0x130$ 。

对于任意长度（必须是 16 字节的整数倍）的明文指令或数据的加密，整个加密过程可拆分成多次进行，每次加密均具备相应的目标空间和相应参数。

对于自动加解密模块，目标空间等参数由硬件自动调节。对于手动加密模块，目标空间等参数需由用户主动配置。

说明：

[IEEE Std 1619-2007](#) 中的章节 5.1 Data units and tweaks 中定义的扭曲值是一个 128-bit 的非负整数 (*tweak*)，其

值可以通过公式求出： $tweak = type * 2^{30} + (base\_addr \& 0x3FFFFF80)$ 。 $tweak$ 中低 7 位和高 97 位恒为零。

23.4.4 数据写入

对于自动加解密模块，数据写入由硬件自动完成。对于手动加密模块，数据写入需由用户主动配置。手动加密模块中包含一个由 16 个寄存器组成的寄存器块 `XTS_AES_PLAIN_n_REG` ($n: 0 \sim 15$)，专用于数据写入，一次可以存放最多 512 位明文指令/数据。

实际上，在手动加密模块中，相对于明文的来源，密文将要存放的位置更为重要。基于明文和密文之间严格的对应关系，为更好地描述明文在寄存器块中的存放方式，现假设明文从一开始就存放在目标空间中，在加密完成后替换为密文。因此，本节中将不再出现“明文”的概念，而用“目标空间”来代替。

目标空间映射到寄存器块的方法：

假设目标空间中某个字的存放地址为 $address$ ，记 $offset = address \% 64$ ， $n = offset / 4$ ，那么该字将被存放在寄存器 `XTS_AES_PLAIN_n_REG` 中。

例如，当目标大小为 64 时，寄存器块中的所有寄存器都将被使用，目标空间中的地址与寄存器块之间的映射关系如表 23.4-2 所示。

表 23.4-2. 目标空间与寄存器堆的映射关系

<i>offset</i>	寄存器	<i>offset</i>	寄存器
0x00	<code>XTS_AES_PLAIN_0_REG</code>	0x20	<code>XTS_AES_PLAIN_8_REG</code>
0x04	<code>XTS_AES_PLAIN_1_REG</code>	0x24	<code>XTS_AES_PLAIN_9_REG</code>
0x08	<code>XTS_AES_PLAIN_2_REG</code>	0x28	<code>XTS_AES_PLAIN_10_REG</code>
0x0C	<code>XTS_AES_PLAIN_3_REG</code>	0x2C	<code>XTS_AES_PLAIN_11_REG</code>
0x10	<code>XTS_AES_PLAIN_4_REG</code>	0x30	<code>XTS_AES_PLAIN_12_REG</code>
0x14	<code>XTS_AES_PLAIN_5_REG</code>	0x34	<code>XTS_AES_PLAIN_13_REG</code>
0x18	<code>XTS_AES_PLAIN_6_REG</code>	0x38	<code>XTS_AES_PLAIN_14_REG</code>
0x1C	<code>XTS_AES_PLAIN_7_REG</code>	0x3C	<code>XTS_AES_PLAIN_15_REG</code>

23.4.5 手动加密模块

手动加密模块是一个外设模块，自身带有寄存器，可以被 CPU 直接访问。模块内的寄存器、系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，手动加密模块才允许手动加密：

- SPI Boot 模式下：

当寄存器 `HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` 的 `HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT` 位为 1 时，手动加密模块拥有工作权限，否则无法工作。

- Download Boot 模式下：

当寄存器 `HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` 的 `HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT` 位为 1，且 eFuse 参数 `EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT` 为 0 时，手动加密模块拥有工作权限，否则无法工作。

说明:

- 即使 CPU 可以越过 cache，直接读取片外存储器从而得到加密指令/数据，但用户依旧无法获取密钥 *Key*。

23.4.6 自动加密模块

自动加密模块并非常规外设模块，不带有寄存器，也无法被 CPU 直接访问。由系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动加密模块才允许自动加密：

- SPI Boot 模式下：

当参数 EFUSE_SPI_BOOT_CRYPT_CNT（共 3 位）为 1，2，4 或 7 时（即二进制表示有奇数个 1 时），自动加密模块拥有工作权限，否则无法工作。

- Joint Download Boot 模式下：

当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT 位为 1 时，自动加密模块拥有工作权限，否则无法工作。

说明:

- 当自动加密模块拥有工作权限时，如果 CPU 向片外 RAM 写入数据，自动加密模块将自动加密该数据，随后将密文写入片外 RAM。加密的整个过程无需软件参与，且对 cache 透明。加密算法过程中，密钥 *Key* 无法被用户获取。
- 当自动加密模块没有工作权限时，将忽略 CPU 对 cache 的访问请求，不处理数据。此时，数据将以明文形式直接写入片外 RAM。

23.4.7 自动解密模块

自动解密并非传统外设模块，自身不带寄存器，不能被 CPU 直接访问。系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动解密模块才允许自动解密：

- SPI Boot 模式下

当参数 EFUSE_SPI_BOOT_CRYPT_CNT（共 3 位）为 1，2，4 或 7 时（即二进制表示有奇数个 1 时），自动解密模块拥有工作权限，否则无法工作。

- Download Boot 模式下

当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 位为 1 时，自动解密模块拥有工作权限，否则无法工作。

说明:

1. 当自动解密模块拥有工作权限时，如果 CPU 通过 cache 读取片外存储器中的指令/数据，自动解密将自动对读取到的密文进行解密以恢复指令/数据。解密的整个过程无需软件参与，且对 cache 透明。解密算法过程中，密

钥 *Key* 无法被软件获取。

2. 当自动解密模块没有工作权限时，无论数据加密与否，自动解密模块不对片外存储器中的数据产生作用。此时，CPU 通过 cache 读取到的是片外存储器中的原始内容。

23.5 软件流程

手动加密模块工作时需要软件参与，软件流程为：

1. 配置 XTS_AES：

- 将寄存器 `XTS_AES_DESTINATION_REG` 配置为 0。
- 将寄存器 `XTS_AES_PHYSICAL_ADDRESS_REG` 的值设置为 *base_addr*。
- 将寄存器 `XTS_AES_LINESIZE_REG` 的值设置为 $\frac{size}{32}$ 。

关于 *base_addr* 和 *size* 的定义，请参考章节 23.4.3。

2. 将明文指令/数据写入至寄存器块 `XTS_AES_PLAIN_n_REG` (*n*: 0 ~ 15)。更多详细信息，请参考章节 23.4.4。

请根据实际需求写入寄存器，未使用的寄存器可为任意值。

3. 等待手动加密模块进入空闲状态。轮询寄存器 `XTS_AES_STATE_REG` 直到软件读取到 0，表明手动加密模块已进入空闲状态。

4. 向寄存器 `XTS_AES_TRIGGER_REG` 写入 1，启动手动加密。

5. 等待加密完成。轮询寄存器 `XTS_AES_STATE_REG`，直到软件读取到 2。
上述步骤为使用 *Key* 操作手动加密模块对明文指令/数据进行加密的过程。

6. 向寄存器 `XTS_AES_RELEASE_REG` 写入 1，使 SPI1 获得密文的访问权限。该操作完成后，寄存器 `XTS_AES_STATE_REG` 的值将为 3。

7. 调用 SPI1，将密文写入片外 flash（请参阅 [ESP-IDF 编程指南](#) 中的 [Flash 加密 API 参考](#)）。

8. 向寄存器 `XTS_AES_DESTROY_REG` 写入 1，销毁密文。该操作完成后，寄存器 `XTS_AES_STATE_REG` 的值将为 0。

请根据所需加密的明文指令/数据的数量，重复上述步骤。

23.6 抗 DPA 攻击

DPA (Differential Power Analysis, 差分能量分析) 是密码学中的一种旁路攻击手段，攻击者可以统计分析从多个加密操作收集的数据，从而计算加密计算中的中间值。ESP32-C5 XTS_AES 模块支持抗 DPA 攻击功能，以抵御来自外界的 DPA 攻击。

根据 [IEEE Std 1619-2007](#)，XTS-AES 算法可分为两步：

- 第一步：计算扭曲值。
- 第二步：计算密文/明文。下文中，将此步骤称为“计算数据单元”。

ESP32-C5 允许用户配置在上述两个过程中启用抗 DPA 功能，以增加安全性，详见下文。

23.6.1 时钟抗 DPA 攻击

时钟抗 DPA 攻击是 ESP32-C5 抵御外部 DPA 攻击的首要手段。

用户可以通过寄存器配置实现不同安全等级：

- 首先，为更好地描述各步骤，定义以下参数：
 - $select\_reg = XTS\_AES\_CRYPT\_DPA\_SELECT\_REGISTER$
 - $reg\_d\_dpa\_en = XTS\_AES\_CRYPT\_CALC\_D\_DPA\_EN$
 - $efuse\_dpa\_en = EFUSE\_CRYPT\_DPA\_ENABLE$
 - $reg\_anti\_dpa\_level = XTS\_AES\_CRYPT\_SECURITY\_LEVEL$
 - $efuse\_anti\_dpa\_level = 3$
- 配置 XTS_AES 模块抗 DPA 攻击的安全等级：

$$Anti\_DPA\_level = select\_reg ? (reg\_anti\_dpa\_level) : (efuse\_dpa\_en * efuse\_anti\_dpa\_level)$$

当 $Anti\_DPA\_level$ 为 0 时，关闭抗 DPA 攻击功能。 $Anti\_DPA\_level$ 的值越大，抗 DPA 攻击的能力就越强。

- 配置是否在 XTS-AES 算法计算数据单元时开启抗 DPA 功能：

$$Anti\_DPA\_enabled\_in\_calc\_D = select\_reg ? reg\_d\_dpa\_en : efuse\_dpa\_en$$

在 $Anti\_DPA\_level$ 不为 0 的情况下，当 $Anti\_DPA\_enabled\_in\_calc\_D$ 等于 1 时，XTS-AES 算法在计算数据单元时开启抗 DPA 功能。

在 $Anti\_DPA\_level$ 不为 0 的情况下，XTS-AES 算法计算扭曲值时总是开启抗 DPA 功能。

说明：

- 在 $efuse\_dpa\_en$ 置 1 后，用户仍可通过配置 $select\_reg = 1$ 及 $reg\_anti\_dpa\_level = 0$ 关闭抗 DPA 功能。
- 配置是否开启抗 DPA 功能会对外存访问带宽造成影响：
 - 在计算数据单元过程中配置开启抗 DPA 功能时，当抗攻击等级 ≥ 4 ，读写带宽将降低超过 50%。
 - 在计算数据单元过程中配置关闭抗 DPA 功能时，当抗攻击等级 ≥ 6 ，读写带宽将降低超过 50%。

23.6.2 伪轮次抗 DPA 功能

ESP32-C5 XTS-AES 模块具有伪轮次抗 DPA 功能，可以增加芯片的抵御攻击的性能。当开启伪轮次功能时，XTS-AES 模块将在原始操作轮次之前和之后随机添加伪轮次。在一个伪轮次中，XTS-AES 将使用由硬件自动生成的伪密钥执行伪计算。伪轮次中的计算不会影响原始 XTS-AES 操作的结果。

伪轮次抗 DPA 功能的主要配置参数包括：

伪轮次功能的模式

伪轮次功能的模式 $pseudo\_mode$ 可用来选择增加伪轮次计算的具体 XTS-AES 工作步骤：

- 0：计算过程中不增加伪轮次；
- 1：在计算扭曲值时增加伪轮次；
- 2：在计算扭曲值和计算数据单元时的某些轮次增加伪轮次；

- 3: 在计算扭曲值和计算数据单元时的所有轮次中增加伪轮次。

伪轮次的总数

在每次 XTS-AES 计算中随机插入伪轮次的具体数量可通过以下配置：

- pseudo_base：该值配置伪轮次功能的基本轮数。
- pseudo_inc：该值配置伪轮次功能的随机递增轮数。

伪轮次的总数将在

$$[pseudo\_base, pseudo\_base + (2^{pseudo\_inc} - 1)]$$

范围内随机生成。

随机数更新频率

pseudo_rnd_cnt 表示伪轮次功能中随机密钥的更新频率。配置的值越大，代表更新频率越高。通常建议将此值配置为最大值 7。

配置

上述参数可通过 eFuse 位或 XTS_AES 相关寄存器完成配置，具体取决于 EFUSE_XTS_DPA_PSEUDO_LEVEL 的值：

- 0：用户可通过 XTS_AES 相关寄存器进行配置
- 1, 2 或 3：不允许用户配置

详见表 23.6-1。

表 23.6-1. XTS_AES 伪轮次功能配置

EFUSE_XTS_DPA_PSEUDO_LEVEL	pseudo_mode	pseudo_base	pseudo_inc	pseudo_rnd_cnt
0	XTS_AES_PSEUDO_MODE	XTS_AES_PSEUDO_BASE	XTS_AES_PSEUDO_INC	XTS_AES_PSEUDO_RNG_CNT
1	1	4	2	7
2	2	4	2	7
3	3	4	2	7

23.7 寄存器列表

本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

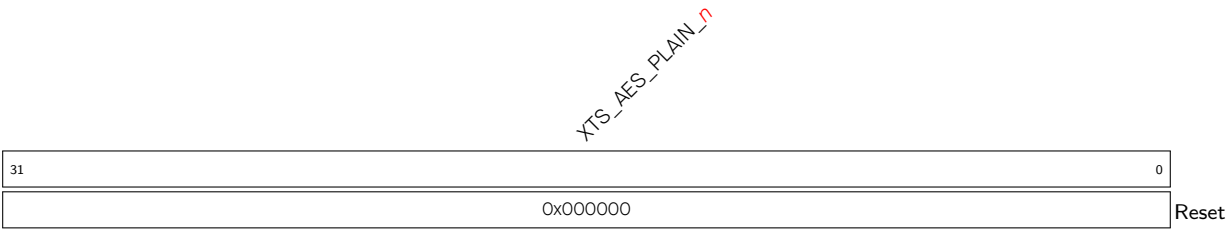
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
明文寄存器堆			
XTS_AES_PLAIN_0_REG	明文寄存器 0	0x0300	R/W
XTS_AES_PLAIN_1_REG	明文寄存器 1	0x0304	R/W
XTS_AES_PLAIN_2_REG	明文寄存器 2	0x0308	R/W
XTS_AES_PLAIN_3_REG	明文寄存器 3	0x030C	R/W
XTS_AES_PLAIN_4_REG	明文寄存器 4	0x0310	R/W
XTS_AES_PLAIN_5_REG	明文寄存器 5	0x0314	R/W
XTS_AES_PLAIN_6_REG	明文寄存器 6	0x0318	R/W
XTS_AES_PLAIN_7_REG	明文寄存器 7	0x031C	R/W
XTS_AES_PLAIN_8_REG	明文寄存器 8	0x0320	R/W
XTS_AES_PLAIN_9_REG	明文寄存器 9	0x0324	R/W
XTS_AES_PLAIN_10_REG	明文寄存器 10	0x0328	R/W
XTS_AES_PLAIN_11_REG	明文寄存器 11	0x032C	R/W
XTS_AES_PLAIN_12_REG	明文寄存器 12	0x0330	R/W
XTS_AES_PLAIN_13_REG	明文寄存器 13	0x0334	R/W
XTS_AES_PLAIN_14_REG	明文寄存器 14	0x0338	R/W
XTS_AES_PLAIN_15_REG	明文寄存器 15	0x033C	R/W
配置寄存器			
XTS_AES_LINESIZE_REG	配置目标空间的大小	0x0340	R/W
XTS_AES_DESTINATION_REG	配置片外存储器的类型	0x0344	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	配置物理地址	0x0348	R/W
XTS_AES_DPA_CTRL_REG	配置抗 DPA 功能	0x0388	R/W
XTS_AES_PSEUDO_ROUND_CONF_REG	配置 XTS-AES 伪轮次功能	0x038C	R/W
控制/状态寄存器			
XTS_AES_TRIGGER_REG	启动 AES 算法	0x034C	WT
XTS_AES_RELEASE_REG	释放控制寄存器	0x0350	WT
XTS_AES_DESTROY_REG	销毁控制寄存器	0x0354	WT
XTS_AES_STATE_REG	状态寄存器	0x0358	RO
版本控制寄存器			
XTS_AES_DATE_REG	版本控制寄存器	0x035C	R/W

23.8 寄存器

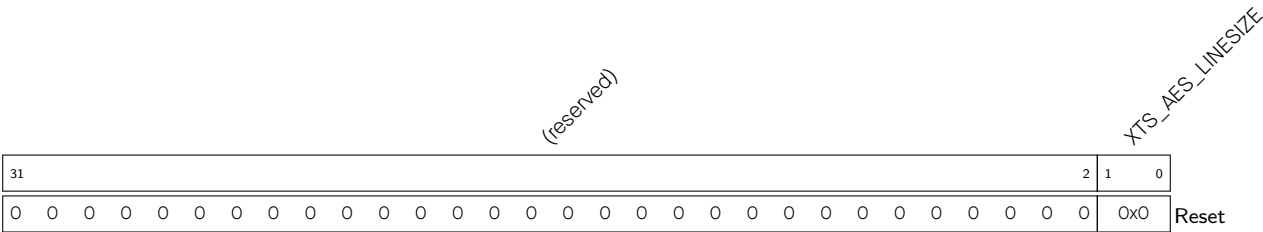
本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

Register 23.1. XTS_AES_PLAIN_ *n* _REG (*n*: 0-15) (0x0300+4**n*)



XTS_AES_PLAIN_ *n* 配置明文的第 *n* 个 32 位部分。(R/W)

Register 23.2. XTS_AES_LINESIZE_REG (0x0340)

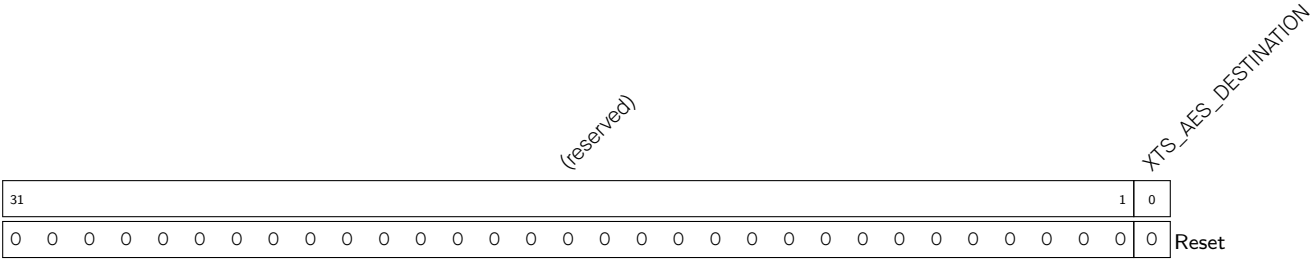


XTS_AES_LINESIZE 配置单次加密的数据大小。

- 0：加密 16 字节
- 1：加密 32 字节
- 2：加密 64 字节
- 3：无效

(R/W)

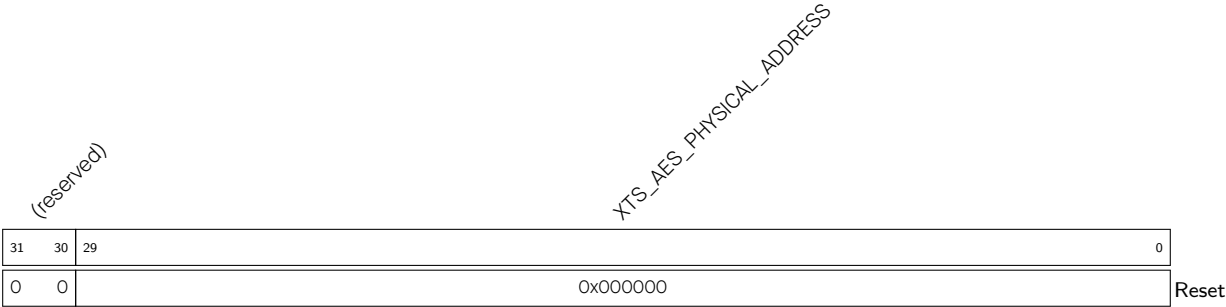
Register 23.3. XTS_AES_DESTINATION_REG (0x0344)



XTS_AES_DESTINATION 配置手动加密的存储器类型。目前只能手动加密 flash，所以只能为 0。用户不能写入 1，否则将发生错误。

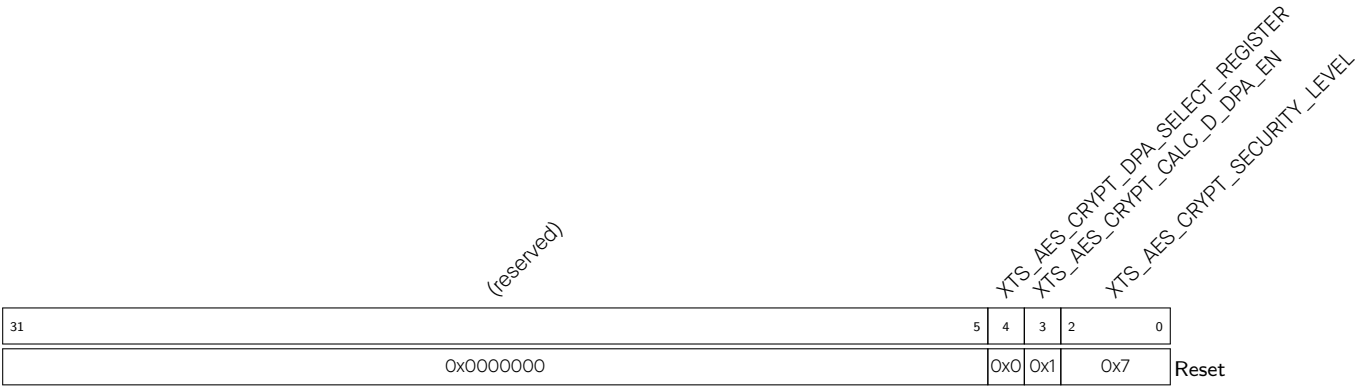
0: 加密 flash
1: 加密片外 RAM
(R/W)

Register 23.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0348)



XTS_AES_PHYSICAL_ADDRESS 配置手动加密使用的物理地址。注意，该值必须按照 [XTS_AES_LINESIZE](#) 定义的大小对齐。(R/W)

Register 23.5. XTS_AES_DPA_CTRL_REG (0x0388)



XTS_AES_CRYPT_DPA_SELECT_REGISTER 配置由 eFuse 或寄存器控制时钟抗 DPA 功能。

- 0: 由寄存器控制时钟抗 DPA 功能
- 1: 由 eFuse 控制时钟抗 DPA 功能

(R/W)

XTS_AES_CRYPT_CALC_D_DPA_EN 配置是否在 XTS_AES 算法中启用时钟抗 DPA 功能。

- 0: 仅在计算扭曲值时启用时钟抗 DPA 功能
- 1: 计算扭曲值或是计算数据单元时都启用时钟抗 DPA 功能

请注意，此字段仅在 [XTS_AES_CRYPT_SECURITY_LEVEL](#) 不为 0 时有效。

(R/W)

XTS_AES_CRYPT_SECURITY_LEVEL 配置外部加解密的安全等级。

- 0: 时钟抗 DPA 功能已关闭
- 1-7: 数字越大，则加解密安全性更高

(R/W)

Register 23.6. XTS_AES_PSEUDO_ROUND_CONF_REG (0x038C)

(reserved)											XTS_AES_PSEUDO_INC		XTS_AES_PSEUDO_BASE		XTS_AES_PSEUDO_RNG_CNT		XTS_AES_PSEUDO_MODE			
31											11		10	9	8	5	4	2	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0											2		2		7		0		Reset	

XTS_AES_MODE_PSEUDO 配置增加伪轮次抗 DPA 计算的具体 XTS-AES 工作步骤。

- 0: 计算过程中不增加伪轮次
- 1: 在计算扭曲值时增加伪轮次
- 2: 在计算扭曲值时和计算数据单元的某些轮次时增加伪轮次
- 3: 在计算扭曲值时和计算数据单元的所有轮次时增加伪轮次

(R/W)

XTS_AES_PSEUDO_RNG_CNT 配置在伪轮次抗 DPA 计算中伪密钥的更新频率。(R/W)

XTS_AES_PSEUDO_BASE 配置伪轮次抗 DPA 计算中的基本轮数。(R/W)

XTS_AES_PSEUDO_INC 配置伪轮次抗 DPA 计算中的递增轮数。(R/W)

Register 23.7. XTS_AES_TRIGGER_REG (0x034C)

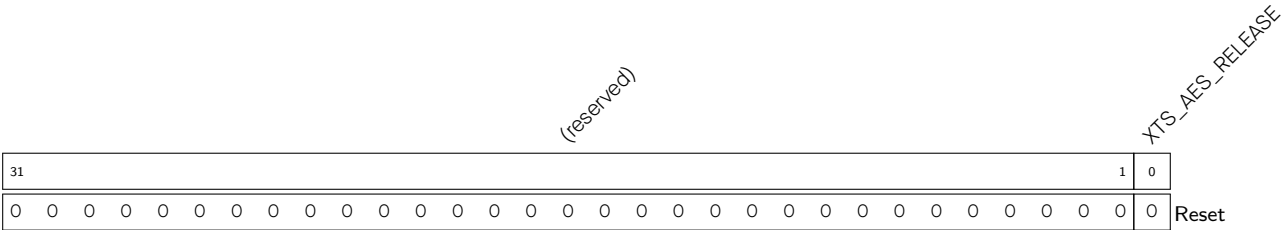
(reserved)																															XTS_AES_TRIGGER																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

XTS_AES_TRIGGER 配置是否使能手动加密运算。

- 0: 关闭手动加密运算
- 1: 使能手动加密运算

注意，此配置尽在手动加密状态为 0 时有效。此配置生效后，手动加密状态变为 1，计算完成后，手动加密状态变为 2。(WT)

Register 23.8. XTS_AES_RELEASE_REG (0x0350)



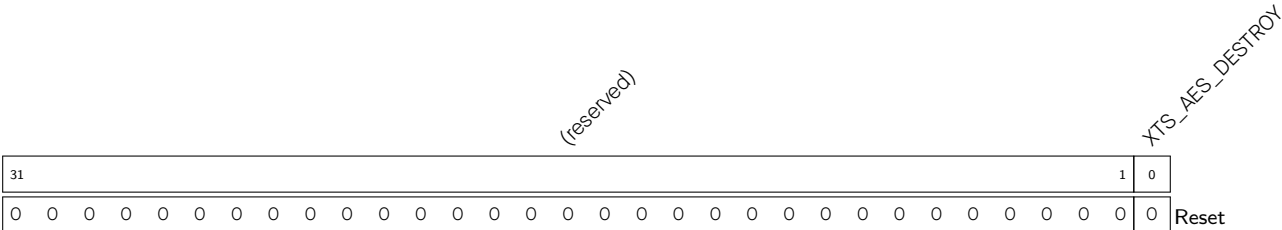
XTS_AES_RELEASE 配置是否向 MSPI 发布加密结果。

0：不发布

1：发布

此配置仅在手动加密状态为 2 时有效。此配置生效后，手动加密状态变为 3。(WT)

Register 23.9. XTS_AES_DESTROY_REG (0x0354)



XTS_AES_DESTROY 配置是否销毁加密结果。

0：无效

1：销毁加密结果

此配置仅在手动加密状态为 3 时有效。此配置生效后，手动加密状态变为 0。(WT)

Register 23.10. XTS_AES_STATE_REG (0x0358)



XTS_AES_STATE 表示手动加密模块状态。

0 (XTS_AES_IDLE)：空闲

1 (XTS_AES_BUSY)：计算中

2 (XTS_AES_DONE)：计算完成，但手动加密结果数据对 MSPI 不可见

3 (XTS_AES_RELEASE)：手动加密结果对 MSPI 可见 (RO)

Register 23.11. XTS_AES_DATE_REG (0x035C)

(reserved)		XTS_AES_DATE	
31	30	29	0
0	0	0x20210907	
			Reset

XTS_AES_DATE 版本控制寄存器。(R/W)

第 24 章

随机数发生器 (RNG)

24.1 概述

ESP32-C5 内置一个真随机数发生器，其生成的 32 位随机数可作为加密等操作的基础。

24.2 主要特性

ESP32-C5 的随机数发生器可通过物理过程而非算法生成真随机数，所有生成的随机数在特定范围内出现的概率完全一样。

24.3 功能描述

系统可以从随机数发生器的寄存器 `LPPERI_RNG_DATA_SYNC_REG` 中读取随机数，每个读到的 32 位随机数都是真随机数，噪声源为系统中的**热噪声**、**异步时钟**和**异步计数器**。

- **热噪声**可以来自 SAR ADC 或高速 ADC 或两者兼有。当芯片的 SAR ADC 或高速 ADC 工作时，就会产生比特流，并通过异或 (XOR) 逻辑运算作为随机数种子进入随机数生成器。
- `RC_FAST_CLK` 是一种**异步时钟源**，会产生电路亚稳态。这种亚稳态也可以作为熵，进入随机数生成器。
- **异步计数器**是利用异步时钟源进行计数，异步时钟源会产生电路亚稳态，可以作为随机数种子进入随机数生成器。异步计数器分为 2 种：
 - 第一种是直接利用片上的 **实时时钟定时器 (RTC_TIMER)** 模块产生的计数值，并通过异或 (XOR) 逻辑运算作为随机数种子进入随机数生成器；
 - 第二种是使用 **环形缓冲区 (BUF_CHAIN)** 作为噪声源。

这两种异步计数器可以各自独立工作，也可以多个一起工作。

说明：

关于噪声源的限制，请查看 [章节编程指南的说明](#)。

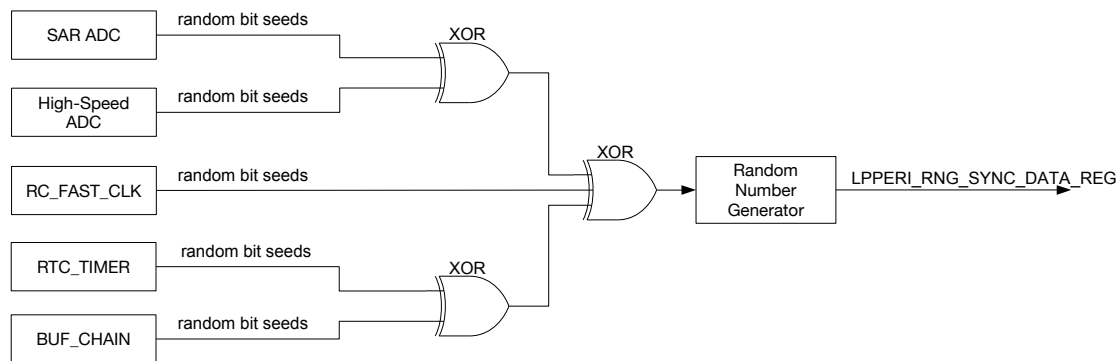


图 24.3-1. 噪声源

当 SAR ADC 打开时，每个 RC_FAST_CLK 时钟周期内（来自内部 RC 振荡器，详见 [6 复位和时钟](#) 章节），随机数发生器将获得 2 位的熵。因此，为了获得最大的熵值，建议读取 [LPPERI_RNG_DATA_SYNC_REG](#) 寄存器时的速率不超过 1 MHz。

当高速 ADC 打开时，每个 APB 时钟周期（通常为 80 MHz）内，随机数发生器将获得 2 位的熵。因此，为了获得最大的熵值，建议读取 [LPPERI_RNG_DATA_SYNC_REG](#) 寄存器时的速率不超过 5 MHz。

我们在仅打开高速 ADC 的状态下，以 5 MHz 的速率从 [LPPERI_RNG_DATA_SYNC_REG](#) 读取了 2 GB 的数据样本，并使用 Dieharder 随机数测试套件（版本 3.31.1）对样本进行了测试。最终，样本通过了所有测试。

24.4 编程指南

要开启随机数生成器，需向 [LPPERI_RNG_CK_EN](#) 写 1 开启随机数生成器时钟。使用 [椭圆曲线数字签名算法 \(ECDSA\)](#) 时，无论 [LPPERI_RNG_CK_EN](#) 配置如何，随机数生成器时钟都会自动启动。

在使用 ESP32-C5 的随机数生成器时，如果熵源没有开启，可能会导致产生伪随机数，应注意避免。其中，

- SAR ADC 受控于 DIG ADC 控制器。详见章节 [36 ADC 控制器](#)。
- 高速 ADC 在 Wi-Fi 或蓝牙开启时自动打开。
- RC_FAST_CLK 处于常开状态。
- 异步计数器
 - RTC 定时器 (RTC_TIMER) 通过设置 [LPPERI_RTC_TIMER_EN](#) 开启。
 - 环形缓冲区 (BUF_CHAIN) 通过置位 [LPPERI_RNG_SAMPLE_ENABLE](#) 开启。

说明：

1. 在 Wi-Fi 开启时，极端情况下高速 ADC 有读值饱和的可能，这会降低熵值。因此，建议在 Wi-Fi 开启时，同时打开 SAR ADC 产生随机数。
2. RC_FAST_CLK 时钟和异步计数器仅可以提高随机数发生器的熵值。然而，为了保证随机数发生器可以获得足够的熵值，仍建议在使用随机数发生器时至少保证 SAR ADC 或高速 ADC 处于工作状态。

请多次读取 `LPPERI_RNG_DATA_SYNC_REG` 寄存器的值，直至获得足够多的随机数。在读取寄存器时，注意控制速率不要超过章节 24.3 的介绍。

24.5 寄存器列表

本小节的所有地址均为相对于随机数生成器基地址的地址偏移量（相对地址），具体基地址请见章节 [4 系统和存储器](#) 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
LPPERI_RNG_CFG_REG	RNG 配置寄存器	0x0024	varies
LPPERI_RNG_DATA_SYNC_REG	RNG 数据同步寄存器	0x0028	RO

24.6 寄存器

本小节的所有地址均为相对于随机数生成器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 24.1. LPPERI_RNG_CFG_REG (0x0024)

(reserved)										(reserved)										LPPERI_RTC_TIMER_EN										(reserved)										LPPERI_RNG_SAMPLE_ENABLE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
31										24										23										12										11										10										9										1										0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
0x0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0										0									

LPPERI_RNG_SAMPLE_ENABLE 配置是否使能 BUF_CHAIN。

0: 关闭

1: 使能

(R/W)

LPPERI_RTC_TIMER_EN Bit[0]: 配置是否在 CRC 前使能 RTC 定时器。

Bit[1]: 配置是否在 CRC 后使能 RTC 定时器。

0: 关闭

1: 使能

(R/W)

Register 24.2. LPPERI_RNG_DATA_SYNC_REG (0x0028)

LPPERI_RNG_SYNC_DATA																															
31																															0
0																															
Reset																															

LPPERI_RNG_SYNC_DATA 表示 RNG 同步数据。(RO)

第 V 卷

通讯接口

该部分涉及系统的连接性，介绍各种带有通讯接口的模块，如 I2C、I2S、SPI、UART、USB 等，同时还包括其他用于遥控、电机控制、LED 控制等功能的信号生成接口。

第 25 章

UART 控制器 (UART)

25.1 概述

UART 是一种以字符为导向、用于实现设备间通信的异步传输接口。异步传输的意思是不需要在发送数据上添加时钟信息。这也要求发送端和接收端的速率、停止位、奇偶校验位等都要相同，通信才能成功。

一个典型的 UART 帧开始于一个起始位，紧接着是有效数据，然后是奇偶校验位（可有可无），最后是停止位。ESP32-C5 上的 UART 控制器支持多种字符长度和停止位。另外，控制器还支持软硬件流控和 GDMA，可以实现高速的数据传输。开发者可以使用多个 UART 端口，同时又能保证很少的软件开销。

ESP32-C5 芯片中共有三个 UART 控制器，包含两个正常功能的 UART 和一个满足低功耗需求的 LP UART。UART 控制器可以兼容不同的 UART 设备。另外，UART 还可以用于红外数据交换 (IrDA) 或 RS485 调制解调器。本文以 UART n 指代两个 UART 控制器， n 为 0、1。LP UART 是 UART 的功能剪裁版本，且单独控制一组寄存器，通过表 25.2-1 可以区分 UART 和 LP UART 的特性。

25.2 主要特性

UART 控制器的特性以及与 LP UART 的特性区分见下表：

表 25.2-1. UART 和 LP UART 特性区分

UART 特性	LP UART 特性
可编程收发波特率，最大为 5 MBaud	
一个 UART 的发送通道和接收通道分别 有 128 x 8 bit RAM 存储空间	LP UART 的发送 FIFO 和接收 FIFO 共用 20 x 8 bit 存储空间
全双工异步通信	
数据位（5 到 8 位）	
停止位（1、1.5 或 2 位）	
奇偶校验位	
AT_CMD 特殊字符检测	
RS485 协议	—
IrDA 协议	—
GDMA 高速数据通信	—
接收超时	
UART 唤醒模式	
软件流控和硬件流控	

见下页

表 25.2-1 – 接上页

UART 特性	LP_UART 特性
三个可分频的时钟源： 1. XTAL_CLK 2. RC_FAST_CLK 3. PLL_F80M_CLK	三个可分频的时钟源： 1. RC_FAST_CLK 2. XTAL_DIV_CLK 3. PLL_F8M_CLK

下文主要针对正常功能的 UART 进行叙述。

25.3 UART 架构

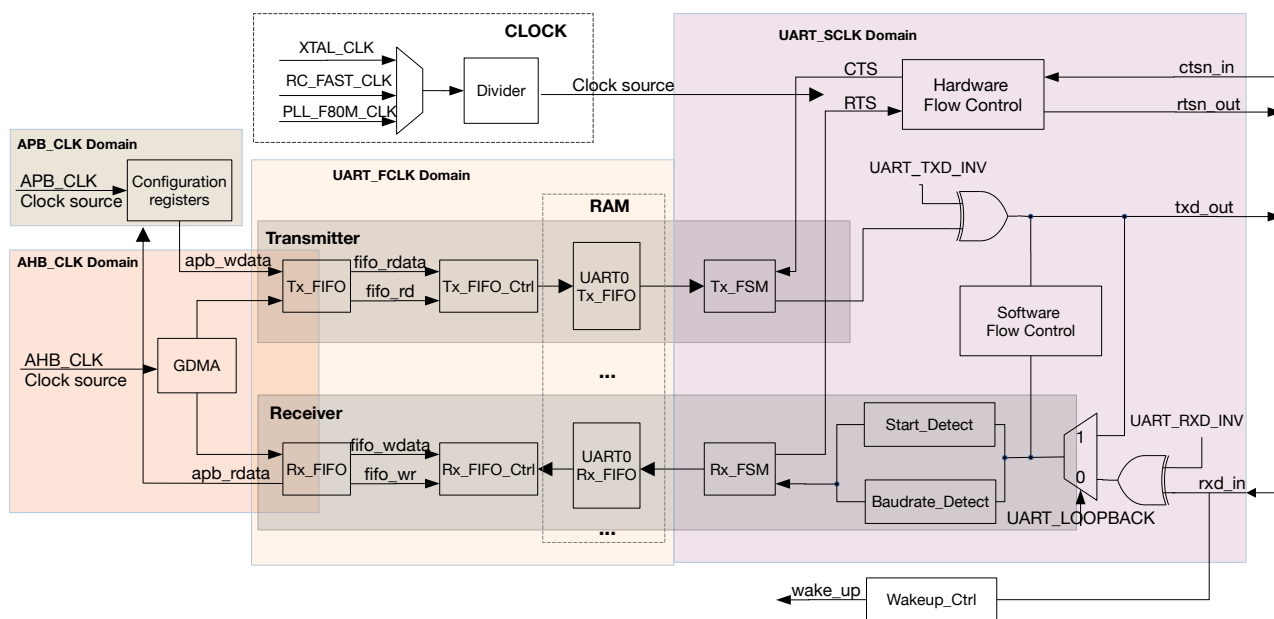


图 25.3-1. UART 基本架构图

图 25.3-1 为 UART 基本架构图。UART 模块工作在 4 个时钟域，分别为 APB_CLK、AHB_CLK、UART_SCLK、UART_FCLK。其中 APB_CLK 与 AHB_CLK 时钟关系为同步不同频（APB_CLK 由 AHB_CLK 分频得到），UART_SCLK 与 UART_FCLK 时钟关系为同步不同频（UART_SCLK 由 UART_FCLK 分频得到）。UART_FCLK 有三个时钟源：PLL_F80M_CLK、RC_FAST_CLK 以及晶振时钟 XTAL_CLK（详情请参考章节 6 复位和时钟）。可以通过配置 `PCR_UARTn_SCLK_SEL` 来选择 UART_FCLK 时钟源，之后使用分频器对时钟源进行分频，然后产生时钟信号 UART_SCLK。分频系数的整数部分由 `PCR_UARTn_SCLK_DIV_NUM` 配置，小数部分的分母由 `PCR_UARTn_SCLK_DIV_A` 配置，小数部分的分子由 `PCR_UARTn_SCLK_DIV_B` 配置。支持的分频范围为 1 ~ 256。只有正常功能的 UART 包含此分频器，LP UART 不包含。

UART 控制器可以分为两个功能块：发送块和接收块。

发送块包含一个发送 FIFO 用于缓存待发送的数据。软件可以通过 APB 总线向 Tx_FIFO 写数据，也可以通过 GDMA 将数据搬入 Tx_FIFO。Tx_FIFO_Ctrl 用于控制 Tx_FIFO 的读写过程，当 Tx_FIFO 非空时，Tx_FSM 通过 Tx_FIFO_Ctrl 读取数据，并将数据按照配置的帧格式转化成比特流。比特流输出信号 txd_out 可以通过配置 `UART_TXD_INV` 寄存器实现取反功能。

接收块包含一个接收 FIFO 用于缓存待处理的数据。输入比特流 rxd_in 可以输入到 UART 控制器。可以通过

`UART_RXD_INV` 寄存器实现取反。Baudrate_Detect 通过检测最小比特流输入信号的脉宽来测量输入信号的波特率。Start_Detect 用于检测数据的起始位，当检测到起始位之后，Rx_FSM 通过 Rx_FIFO_Ctrl 将帧解析后的数据存入 Rx_FIFO 中。软件可以通过 APB 总线读取 Rx_FIFO 中的数据，也可以使用 GDMA 进行数据接收。

HW_Flow_Ctrl 通过标准 UART RTS 和 CTS (rtsn_out 和 ctsn_in) 流控信号来控制 rxd_in 和 txd_out 的数据流。SW_Flow_Ctrl 通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来进行数据流的控制。当 UART 处于 Light-sleep 模式（详情请参考章节 4 低功耗管理 [to be added later]）时，可以通过四种不同的方式产生 wake_up 信号给 PMU 模块，由 PMU 来唤醒 ESP32-C5 芯片。具体唤醒方式请参考章节 25.4.8。

25.4 功能描述

25.4.1 时钟与复位

UART 为异步外设。其寄存器配置模块工作在 APB_CLK 时钟域，TX/RX FIFO 作为跨时钟域模块工作在 AHB_CLK 和 UART_FCLK 时钟域。UART RAM 控制单元工作在 UART_FCLK 时钟域，控制 UART 发送与接收的模块工作在 UART_SCLK 时钟域，即 UART Core 模块的时钟域。

在 UART_SCLK 时钟频率满足生成波特率的需求下，可通过预分频使 UART Core 模块工作在较小的时钟频率，从而减小 UART 外设的功耗。通常情况下，UART Core 模块时钟小于 APB_CLK 时钟，并且在满足 UART 波特率的情况下，UART Core 时钟分频系数可以配置到最大值。UART 也支持 UART Core 模块时钟大于 APB_CLK 时钟，此时，UART Core 模块时钟最大为 APB_CLK 的 3 倍。另外，UART Core 的 TX/RX 时钟可以被单独控制。置位 `UART_TX_SCLK_EN` 使能 UART Core 的 TX 时钟；置位 `UART_RX_SCLK_EN` 使能 UART Core 的 RX 时钟。

为确保配置寄存器的值成功从 APB_CLK 时钟域同步到 UART_SCLK 时钟域，寄存器配置需要遵循一定的流程，详情请参考章节 25.6。

对整个 UART 的复位，需要遵循如下配置流程：

- 将 `PCR_UARTn_CLK_EN` 置 1 打开 UART n Core 时钟。
- 向 `PCR_UARTn_RST_EN` 位写 1。
- 将 `PCR_UARTn_RST_EN` 位清 0。

25.4.2 UART FIFO

UART 控制器的发送块和接收块分别使用一块 128 x 8 位 RAM 存储空间，并且分别通过一个 4 x 8 位的异步 FIFO 接口访问对应的 RAM。发送块和接收块的存储空间和异步 FIFO 接口相互独立，不可复用。

UART 的 Tx_FIFO 可以通过置位 `UART_TXFIFO_RST` 来复位，Rx_FIFO 可以通过置位 `UART_RXFIFO_RST` 来复位。

对于 TX FIFO，可以通过 APB 总线或 GDMA 向其写入数据，硬件 Tx_FSM 自动从其中读取数据，数据将按照配置的帧格式转换成比特流；对于 RX FIFO，可以通过 APB 总线或 GDMA 读取其中的数据，并存储到内存，硬件 Rx_FSM 将接收到的比特流转换成字节并写入 RX FIFO。两个 UART 共享同一个 GDMA 通道。

配置 `UART_TXFIFO_EMPTY_THRHD` 可以设置 Tx_FIFO 空信号阈值，当存储在 Tx_FIFO 中的数据量小于 `UART_TXFIFO_EMPTY_THRHD` 时会产生中断 `UART_TXFIFO_EMPTY_INT`；配置 `UART_RXFIFO_FULL_THRHD` 可以设置 Rx_FIFO 满信号阈值，当储存在 Rx_FIFO 中的数据量大于或等于 `UART_RXFIFO_FULL_THRHD` 会产生中断 `UART_RXFIFO_FULL_INT`。另外，当 Rx_FIFO 中储存的数据量超过其能存储的最大值时，会产生 `UART_RXFIFO_OVF_INT` 中断。

UART_n 可以通过寄存器 `UART_FIFO_REG` 访问 FIFO。您可以写 `UART_RXFIFO_RD_BYTE` 将数据存入 TX FIFO，也可以读 `UART_RXFIFO_RD_BYTE` 获取 RX FIFO 中的数据。

25.4.3 波特率产生与检测

25.4.3.1 波特率产生

在 UART 发送或接收数据之前，需要配置寄存器来设置波特率。波特率发生器主要通过对输入时钟源的分频来实现，支持小数分频。`UART_CLKDIV_SYNC_REG` 将分频系数分成两个部分：`UART_CLKDIV` 用于配置整数部分，`UART_CLKDIV_FRAG` 用于配置小数部分。在输入时钟为 80 MHz 的情况下，UART 能支持的最大波特率为 5 MBaud。

波特率分频器系数为

$$UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}$$

也就是说，最终波特率为

$$\frac{INPUT\_FREQ}{UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}}$$

其中，INPUT_FREQ 为 UART Core 时钟。例如，若 `UART_CLKDIV` = 694，`UART_CLKDIV_FRAG` = 7，则分频系数为

$$694 + \frac{7}{16} = 694.4375$$

`UART_CLKDIV_FRAG` 为 0 时，分频器为整数分频，每 `UART_CLKDIV` 个输入脉冲都会产生一个输出脉冲。

`UART_CLKDIV_FRAG` 不为 0 时，分频器为小数分频，输出波特率脉冲不完全统一。如图 25.4-1 所示，每 16 个输出脉冲，波特率发生器分频 (`UART_CLKDIV` + 1) 个输入脉冲或 `UART_CLKDIV` 个输入脉冲。分频 (`UART_CLKDIV` + 1) 个输入脉冲产生 `UART_CLKDIV_FRAG` 个输出脉冲，分频 `UART_CLKDIV` 个输入脉冲产生剩余的 (16 - `UART_CLKDIV_FRAG`) 个输出脉冲。

如图 25.4-1 所示，输出脉冲相互交错，使得输出时序更加统一。

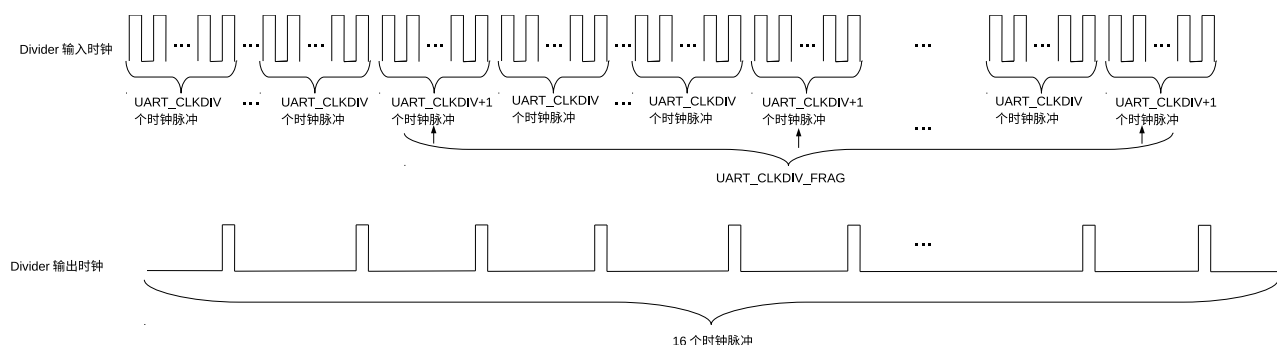


图 25.4-1. UART 控制器分频

为了支持 IrDA（详情见章节 25.4.7），IrDA 小数分频器会产生 $16 \times \text{UART\_CLKDIV\_SYNC\_REG}$ 分频的时钟用于 IrDA 数据传输。产生 IrDA 数据传输时钟的小数分频器原理与上述小数分频器一样，取 `UART_CLKDIV`/16 作为分频值的整数部分，取 `UART_CLKDIV` 的低 4 比特作为小数部分。

25.4.3.2 波特率检测

置位 `UART_AUTOBAUD_EN` 可以开启 UART 波特率自检测功能。图 25.4-2 中的 Baudrate_Detect 可以滤除信号脉宽小于 `UART_GLITCH_FILT` 的噪声。

在 UART 双方进行通信之前，可以通过发送随机数据，让具有波特率检测功能的数据接收方进行波特率分析。
`UART_LOWPULSE_MIN_CNT` 存储了最小低电平脉冲宽度，`UART_HIGHPULSE_MIN_CNT` 存储了最小高电平脉冲宽度，`UART_POSEDGE_MIN_CNT` 存储了两个上升沿之间的最小脉冲宽度，`UART_NEGEDGE_MIN_CNT` 存储了两个下降沿之间最小的脉冲宽度。软件可以通过读取这四个寄存器获取发送方的波特率。

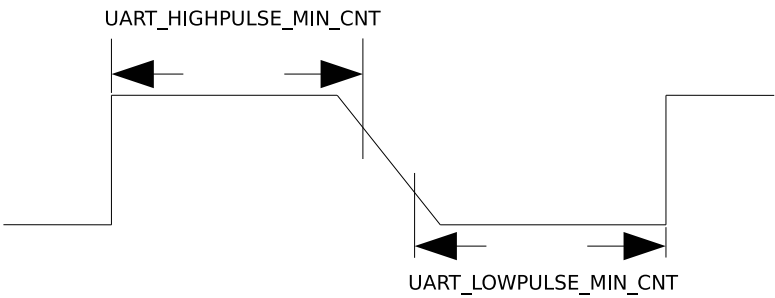


图 25.4-2. UART 信号下降沿较差时序图

波特率的计算分为三种情况：

- 1. 正常情况下，为防止因亚稳态在上升沿或下降沿附近采样数据错误而导致 `UART_LOWPULSE_MIN_CNT` 或者 `UART_HIGHPULSE_MIN_CNT` 不准确，单比特脉冲宽度可以通过将这两个值相加取平均消除误差。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

- 2. 对于 UART 信号的下降沿信号比较差的情况，如图25.4-2 所示，这时通过取 `UART_LOWPULSE_MIN_CNT` 与 `UART_HIGHPULSE_MIN_CNT` 的和平均得到的值不准确，可以通过 `UART_POSEDGE_MIN_CNT` 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

- 3. 对于 UART 信号的上升沿信号比较差的情况，可以通过 `UART_NEGEDGE_MIN_CNT` 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

25.4.4 UART 数据帧

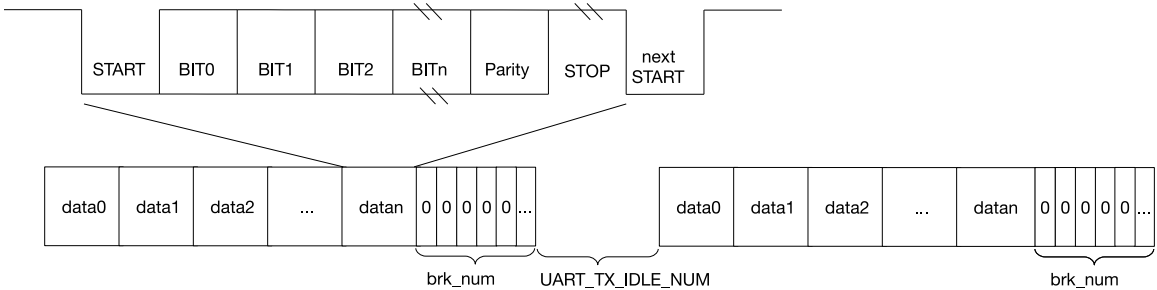


图 25.4-3. UART 数据帧结构

图 25.4-3 所示为基本数据帧格式，数据帧从起始位开始，以停止位结束。起始位占用 1 位，停止位可以通过配置 `UART_STOP_BIT_NUM` 实现 1、1.5、2 位宽（RS485 模式下可增加转换延时，详见章节 25.4.6.2）。起始位

为低电平，停止位为高电平。

数据位宽 (BIT0 ~ BITn) 为 5 ~ 8 位，可以通过 `UART_BIT_NUM` 进行配置。当置位 `UART_PARITY_EN` 时，数据帧会在数据之后添加一位奇偶校验位。`UART_PARITY` 用于选择奇校验或是偶校验。当接收器检测到输入数据的校验位错误时会产生 `UART_PARITY_ERR_INT` 中断，输入数据仍会存入 `Rx_FIFO`。当接收器检测到数据数据帧格式错误时会产生 `UART_FRM_ERR_INT` 中断，默认情况下，输入数据会被存入 `Rx_FIFO`。

`Tx_FIFO` 中数据都发送完成后会产生 `UART_TX_DONE_INT` 中断。置位 `UART_TXD_BRK` 时，`Tx_FIFO` 中数据发送完成后，发送端会进入终止状态 (break condition)，继续发送几个连续的特殊数据帧 `NULL`。在 `NULL` 数据帧发送完成后，`TX` 数据线输出为低电平。`NULL` 数据帧的数量可由 `UART_TX_BRK_NUM` 进行配置。发送器发送完所有的 `NULL` 数据帧之后会产生 `UART_TX_BRK_DONE_INT` 中断。数据帧之间可以通过配置 `UART_TX_IDLE_NUM` 保持最小间隔时间。当一帧数据之后的空闲时间大于等于 `UART_TX_IDLE_NUM` 寄存器的配置值时则产生 `UART_TX_BRK_IDLE_DONE_INT` 中断。

在传输一个 `NULL` 数据帧所需的时间内，接收端 `RX` 数据线若一直检测到低电平，接收端会检测为终止状态，并触发 `UART_BRK_DET_INT` 中断表示终止状态已结束。

接收端通过 `UART_RXFIFO_TOUT_INT` 中断检测总线状态。接收端接收到至少一个字节数据后，总线处于空闲状态超过 `UART_RX_TOUT_THRHD` 位时间时，触发 `UART_RXFIFO_TOUT_INT` 中断。您可用此中断检测发送端是否已经发送所有数据。

25.4.5 AT_CMD 字符格式

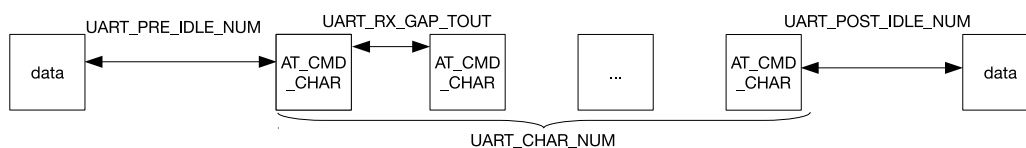


图 25.4-4. AT_CMD 字符格式

图 25.4-4 为一种特殊的 `AT_CMD` 字符格式。当接收器连续收到 `AT_CMD_CHAR` 字符且字符之间满足如下条件时将会产生 `UART_AT_CMD_CHAR_DET_INT` 中断。

- 接收到的第一个 `AT_CMD_CHAR` 与上一个非 `AT_CMD_CHAR` 之间间隔至少 `UART_PRE_IDLE_NUM` 个波特率周期。
- `AT_CMD_CHAR` 字符之间间隔小于 `UART_RX_GAP_TOUT` 个波特率周期。
- 接收的 `AT_CMD_CHAR` 字符个数必须大于等于 `UART_CHAR_NUM`。
- 接收到的最后一个 `AT_CMD_CHAR` 字符与下一个非 `AT_CMD_CHAR` 之间间隔至少 `UART_POST_IDLE_NUM` 个波特率周期。

注意：由于 `AT_CMD_CHAR` 字符之间间隔小于 `UART_RX_GAP_TOUT` 个波特率周期的限制，建议用户使用该功能时，配置的 APB 时钟频率不要低于 8 MHz。

25.4.6 RS485

两个 UART 控制器支持 RS485 通讯模式，RS485 因使用差分信号传输数据，相比于 RS232 具有更远的传输距离及更高的传输速率。RS485 有两线半双工及四线全双工两种选择，UART 模块采用两线半双工模式，并支持侦听总线的功能。

25.4.6.1 驱动控制

如图 25.4-5 所示，RS485 两线 multidrop 系统中，需要一个外部 RS485 收发器实现单端信号与差分信号的转换。RS485 收发器包括一个驱动器与一个接收器。当 UART 不作为发送器时，通过关闭驱动器来断开与差分传输线的连接。DE 为 1 时，使能驱动器；DE 为 0 关闭驱动器。

UART 接收端通过外部接收器将差分信号转为单端信号。RE 作为接收器的使能控制信号，RE 为 0，使能接收器；RE 为 1，关闭接收器。如果 RE 被配置为 0，从而允许 UART 保持侦听总线上的数据，包括 UART 发送的数据。

DE 信号的控制分为软件控制和硬件控制两种方法。为减少软件的开销，DE 信号采用硬件来控制。图 25.4-5 所示，DE 与 UART 的 dtrn_out 相连（详见 25.4.9.1 小节）。

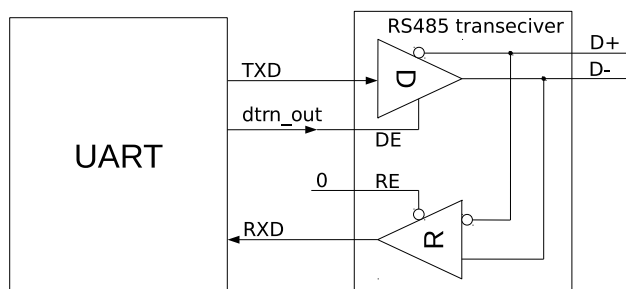


图 25.4-5. RS485 模式驱动控制结构图

25.4.6.2 转换延时

默认情况下，UART 处于接收状态。当从发送转为接收状态时，为保证发送数据被稳定接收，RS485 协议要求在发送停止位之后增加一个波特率的转换延时。UART 发送模块支持在起始位之前或在停止位之后增加一个波特率的延时。置位 `UART_DLO_EN`，在起始位之前增加一个波特率周期延时；置位 `UART_DL1_EN`，在停止位之后增加一个波特率周期延时。

25.4.6.3 总线侦听

RS485 两线 multidrop 系统中，当外部 RS485 收发器的 RE 被配置为 0 时，UART 支持侦听总线。默认情况下，不允许 UART 在发送数据时接收数据。置位 `UART_RS485TX_RX_EN`，允许在发送数据时接收数据，配合图 25.4-5 中外部 RS485 收发器的配置，UART 保持侦听传输总线。另外，默认情况下，不允许 UART 在接收数据时发送数据。置位 `UART_RS485RXBY_TX_EN`，允许在接收数据时发送数据。

UART 支持侦听 UART 发送的数据。UART 处于发送状态下，当侦听到 UART 发送的数据与 UART 接收的数据冲突时，触发 `UART_RS485_CLASH_INT` 中断；侦听到发送的数据帧错误时，触发 `UART_RS485_FRM_ERR_INT` 中断；侦听到发送数据校验位错误时，触发 `UART_RS485_PARITY_ERR_INT` 中断。

25.4.7 IrDA

IrDA 数据协议由物理层，链路接入层和链路管理层三个基本层协议组成。UART 实现了其物理层协议。在 IrDA 编码模式下，支持最大信号速率到 115.2 Kbit/s，即 SIR 模式。如图 25.4-6 所示，IrDA 编码器将来自 UART 的非归零编码 (NRZ) 信号采用反向归零编码 (RZI) 并输出给外部驱动和红外 LED，用 3/16 Bit Time 的脉宽调制信号表示逻辑“0”，用低电平表示逻辑“1”。IrDA 解码器接收来自红外接收器的信号并输出为 UART 的 NRZ 编码。一般情况下，接收端信号空闲时为高电平，编码器输出校验位与解码器输入校验位相反。当检测到低脉冲表示接收到开始信号。

IrDA 使能时，一个位被划分为 16 个时钟周期，在其第 9、10、11 个时钟周期中，当需要发送的比特为 0 时，IrDA 输出为高。

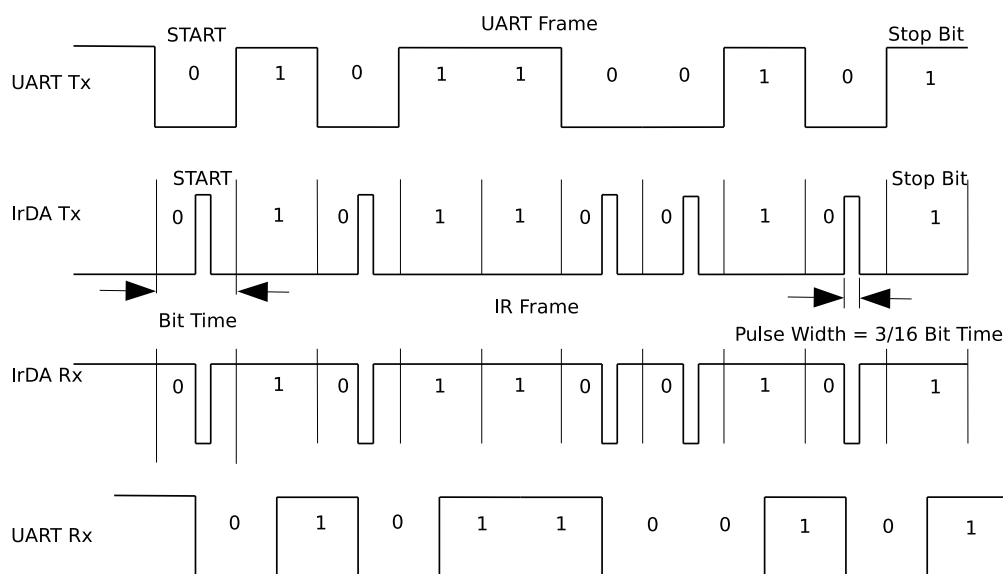


图 25.4-6. SIR 模式编解码时序图

IrDA 为半双工传输协议，不允许同时进行收发。如图 25.4-7 所示，置位 `UART_IRDA_EN` 使能 IrDA 功能。置位 `UART_IRDA_TX_EN`（置 1）使能 IrDA 发送数据，这时不允许 IrDA 接收数据；复位 `UART_IRDA_TX_EN`（清 0）使能 IrDA 接收数据，这时不允许 IrDA 发送数据。

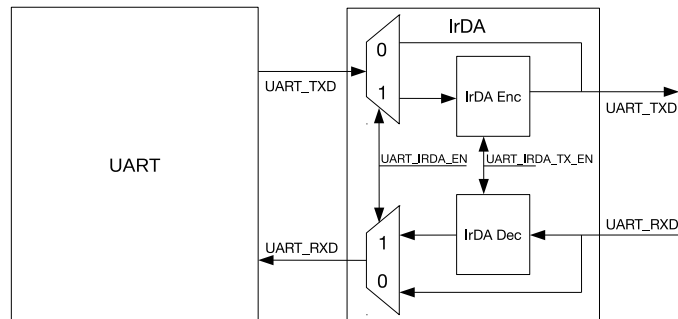


图 25.4-7. IrDA 编解码结构图

25.4.8 唤醒

UART 支持唤醒功能。当 UART 处于 Light-sleep 模式时，可以通过四种不同的方式产生 wake_up 信号给 PMU 模块，由 PMU 来唤醒芯片。

- `UART_WK_MODE_SEL=0`，当所有的时钟都关闭时，此时可以通过使 RXD 翻转若干周期，当上升沿个数大于等于 `UART_ACTIVE_THRESHOLD + 3` 时唤醒芯片。
- `UART_WK_MODE_SEL=1`，由于 UART Core 时钟保持工作，因此 UART RX 仍然可以接收数据并将数据暂存在 Rx FIFO 中。当 Rx FIFO 中的数据量超过 `UART_RX_WAKE_UP_THRHD` 时，可以将芯片从 Light-sleep 中唤醒。
- `UART_WK_MODE_SEL=2`，当 UART RX 监测到起始位后，唤醒芯片。

- `UART_WK_MODE_SEL=3`，当 UART RX 接收到特定字符序列后，唤醒芯片。用户通过配置 `UART_WK_CHAR0`、`UART_WK_CHAR1`、`UART_WK_CHAR2`、`UART_WK_CHAR3`、`UART_WK_CHAR4` 自定义唤醒字符。UART 监测到由 `CHAR0/CHAR1/CHAR2/CHAR3/CHAR4` 组成的字符序列流之后唤醒芯片。可以通过配置 `UART_CHAR_NUM` 和 `UART_WK_CHAR_MASK` 来设置不同的字符序列。如下表所示，对于最后一个配置，UART 将依次检测 `CHAR0 ~ CHAR4`。

表 25.4-1. UART_CHAR_WAKEUP 模式配置

UART_CHAR_NAME	UART_WP_CHAR_MASK	检测字符序列
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

使用 UART 唤醒之后，需要通过在 Active 模式下向 UART 传输数据或是复位整个 UART 模块清除 `wake_up` 信号，否则下一次唤醒所需的上升沿个数将减少。

25.4.9 流控

UART 控制器有两种数据流控方式：硬件流控和软件流控。硬件流控主要通过输出信号 `rtsn_out` 以及输入信号 `ctsn_in` 进行数据流控制。软件流控主要通过发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来实现数据流控功能。

25.4.9.1 硬件流控

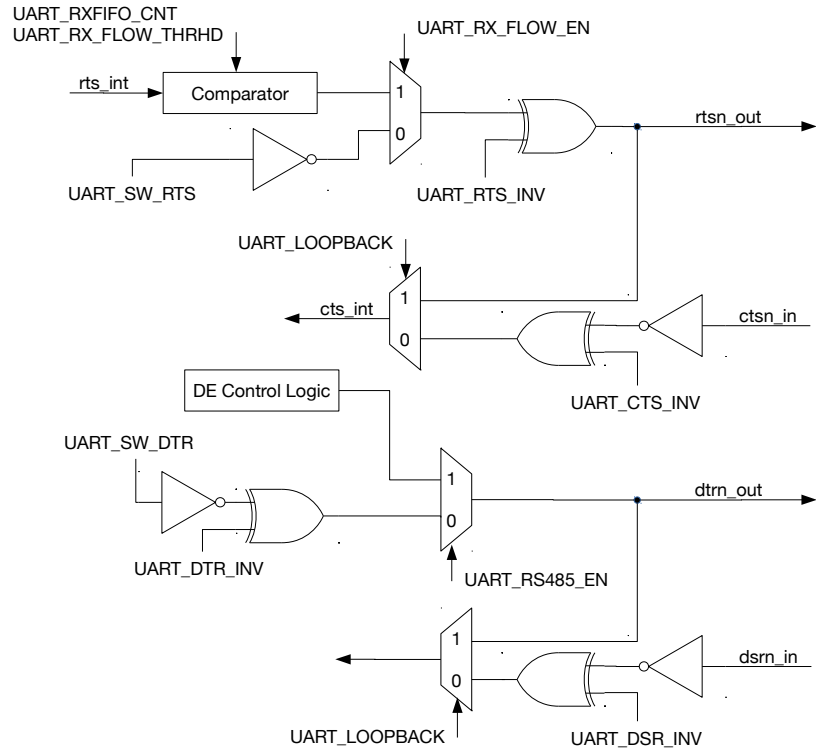


图 25.4-8. 硬件流控图

图 25.4-8 为 UART 硬件流控图。硬件流控的控制信号为输出信号 `rtsn_out` 及输入信号 `ctsn_in`。图 25.4-9 为两个 UART 之间硬件流控信号连接图。记 ESP32-C5 UART 为 IUO，External UART 为 EUO，下文将使用这两个标记来区分两个 UART。输出信号 `rtsn_out` (IUO) 为低电平表示允许对方 (EUO) 发送数据，`rtsn_out` (IUO) 为高电平表示通知对方 (EUO) 中止数据发送直到 `rtsn_out` (IUO) 恢复低电平。`rtsn_out` 输出信号的控制有两种方式。

- 软件控制：将 `UART_RX_FLOW_EN` 置 0 进入该模式。该模式下通过软件配置 `UART_SW_RTS` 改变 `rtsn_out` 的电平。
- 硬件控制：将 `UART_RX_FLOW_EN` 置 1 进入该模式。该模式下硬件会当 Rx_FIFO 中的数据大于 `UART_RX_FLOW_THRHD` 时拉高 `rtsn_out` 的电平。

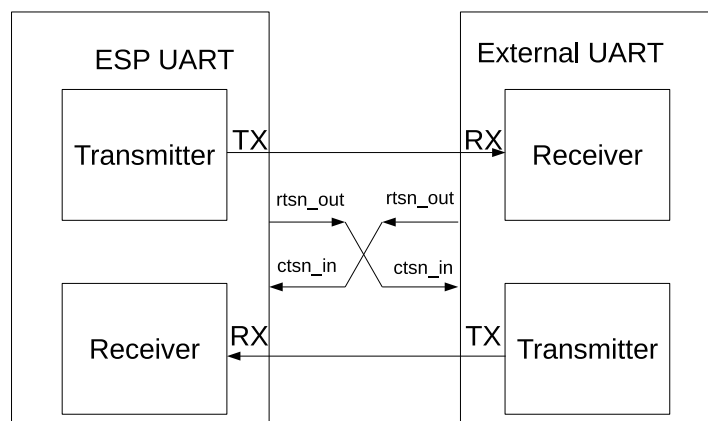


图 25.4-9. 硬件流控信号连接图

输入信号 `ctsn_in` (IUO) 为低电平表示允许发送端 (IUO) 发送数据；`ctsn_in` (IUO) 为高电平表示禁止发送端 (IUO) 发送数据。当 UART 检测到输入信号 `ctsn_in` (IUO) 的沿变化时会产生 `UART_CTS_CHG_INT` 中断。

UART 发送设备 (IUO) 输出信号 `dtrn_out` 为高电平表示发送数据已经准备完毕，处于可用状态。`dtrn_out` 通过配置寄存器 `UART_SW_DTR` 产生。UART 接收设备 (IUO) 在检测到输入信号 `dstrn_in` 的沿变化时会产生 `UART_DSR_CHG_INT` 中断。软件在检测到中断后，通过读取 `UART_DSRN` 可以获取 `dstrn_in` 的输入信号电平，`UART_DSRN` 为高电平时，表示对方设备 (EUO) 处于可用状态。

对于 RS485 两线 multidrop 系统，使用 `dtrn_out` 来收发转换。置位 `UART_RS485_EN` 使能 RS485 功能，`dtrn_out` 由硬件产生。数据开始发送时，`dtrn_out` 拉高，使能外部驱动器；数据最后一位发送完成后，`dtrn_out` 拉低，关闭外部驱动器。注意，当使能停止位之后增加一个波特率延时，`dtrn_out` 会在延时结束后才拉低。

置位 `UART_LOOPBACK` 即开启 UART 的回环测试功能。此时 UART 的输出信号 `txd_out` 和其输入信号 `rxn_in` 相连，`rtsn_out` 和 `ctsn_in` 相连，`dtrn_out` 和 `dstrn_in` 相连。当接收的数据与发送的数据相同时表明 UART 能够正常发送和接收数据。

25.4.9.2 软件流控

软件流控不使用硬件的 CTS/RTS 控制线，而是在发送数据流中嵌入 XON/XOFF 字符来通知对方是否可以使用数据发送来实现流控。将 `UART_SW_FLOW_CON_EN` 置 1 使能软件流控。

在使用软件流控后，硬件会自动检测接收数据流中是否有 XON/XOFF 字符，在检测到相应的字符后会产生 `UART_SW_XOFF_INT` 或 `UART_SW_XON_INT` 中断。在检测到接收数据流中有 XOFF 字符后，发送器将会在发送完当前数据后停止发送；在检测到接收数据流中有 XON 字符后，将会使能发送器发送数据。另外，软件可以

通过置位 `UART_FORCE_XOFF` 来强制发送器停止发送数据，发送器会在发送完当前字节后停止发送；也可以通过置位 `UART_FORCE_XON` 来使能发送器发送数据。

软件可以根据 Rx_FIFO 中剩余空间大小决定流控字符的发送。置位 `UART_SEND_XOFF`，发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 `UART_XOFF_CHAR` 配置；置位 `UART_SEND_XON`，发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 `UART_XON_CHAR` 配置。另外，当 UART 接收 FIFO 中的数据量超过 `UART_XOFF_THRESHOLD` 时，硬件会置位 `UART_SEND_XOFF`，UART 发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 `UART_XOFF_CHAR` 配置。当 UART 接收 FIFO 中的数据量小于 `UART_XON_THRESHOLD` 时，硬件会置位 `UART_SEND_XON`，UART 发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 `UART_XON_CHAR` 配置。

当 UART 工作在全双工模式时，如果 UART RX 接收到 XOFF，此时尽管 UART RX 接受的数据超过阈值，UART TX 将被禁止发送包括 XOFF 在内的任何数据。为了避免该种情况导致的软流控死锁或者 overflow，可以使能 `UART_XON_XOFF_STILL_SEND`，此时允许 UART TX 在被禁止发送时，仍然可以发送一个 XOFF 字符。

25.4.10 GDMA 模式

ESP32-C5 中的两个 UART 接口通过通用主机控制器接口 (UHCI) 共用 1 组 GDMA TX/RX 通道。在 GDMA 模式下，支持对 HCI 协议数据包的解析 (decoder) 及数据包封装 (encoder)。`UHCI_UART_SEL` 字段用于选择哪个串口占用 GDMA 通道。

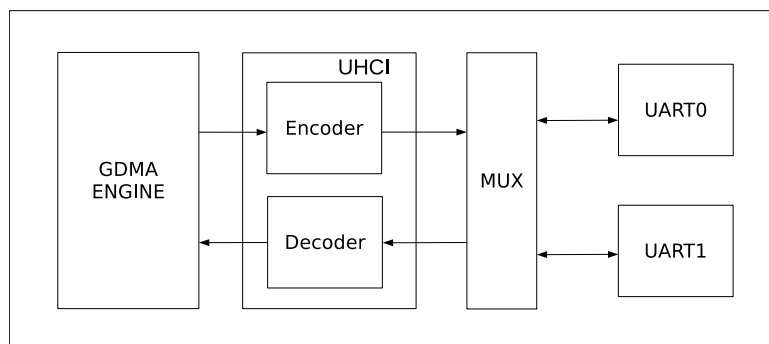


图 25.4-10. GDMA 模式数据传输

图 25.4-10 为 GDMA 方式数据传输图。在 GDMA Rx 通道接收数据前，软件将接收链表准备好。

`GDMA_INLINK_ADDR_CHn` 用于指向第一个接收链表描述符。置位 `GDMA_INLINK_START_CHn` 之后，通用主机控制器接口 (UHCI) 会将 UART 接收到的数据传送给 decoder。经过 decoder 解析之后的数据在 GDMA 通道的控制下存入接收链表指定的 RAM 空间。

在 GDMA Tx 通道发送数据前，软件需要将发送链表和发送数据准备好，`GDMA_OUTLINK_ADDR_CHn` 用于指向第一个发送链表描述符。置位 `GDMA_OUTLINK_START_CHn` 之后，GDMA 引擎即从链表中指定的 RAM 地址读取数据，并通过 encoder 进行数据包封装，然后经 UART 的发送模块串行发送出去。

HCI 的数据包格式为 (分隔符 + 数据 + 分隔符)。encoder 用于在数据前后加上分隔符，并将数据中和分隔符一样的数据替换为特殊字符。decoder 用于去除数据包前后分隔符，并将数据中的特殊字符替换为分隔符。数据前后的分隔符可以有连续多个。分隔符可由 `UHCI_SEPER_CHAR` 进行配置，默认值为 `0xC0`。数据中与分隔符一样的数据可以用 `UHCI_ESC_SEQ0_CHAR0` (默认为 `0xDB`) 和 `UHCI_ESC_SEQ0_CHAR1` (默认为 `0xDD`) 进行替换。当数据全部发送完成后，会产生 `GDMA_OUT_TOTAL_EOF_CHn_INT` 中断。当数据接收完成后，会产生 `GDMA_IN_SUC_EOF_CHn_INT` 中断。

25.5 中断

ESP32-C5 的 UART_n 和 UHCI 可以生成以下中断信号并发送给中断矩阵。

- UART_n_INTR
- UHCI_INTR

这些中断信号分别由 UART_n 和 UHCI 的内部中断源生成。

UART_n 中断源如下所列：

- UART_AT_CMD_CHAR_DET_INT：当接收器检测到 AT_CMD 字符时触发此中断。
- UART_RS485_CLASH_INT：在 RS485 模式下检测到发送器和接收器之间的冲突时触发此中断。
- UART_RS485_FRM_ERR_INT：在 RS485 模式下检测到发送块发送的数据帧错误时触发此中断。
- UART_RS485_PARITY_ERR_INT：在 RS485 模式下检测到发送块发送的数据校验位错误时触发此中断。
- UART_TX_DONE_INT：当发送器发送完 FIFO 中的所有数据时触发此中断。
- UART_TX_BRK_IDLE_DONE_INT：当发送器在最后一个数据发送后保持了最短的间隔时间时触发此中断。
- UART_TX_BRK_DONE_INT：当发送 FIFO 中的数据发送完之后发送器完成了发送 NULL 则触发此中断。
- UART_GLITCH_DET_INT：当接收器在起始位的中点处检测到毛刺时触发此中断。
- UART_SW_XOFF_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XOFF 字符时触发此中断。
- UART_SW_XON_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XON 字符时触发此中断。
- UART_RXFIFO_TOUT_INT：当接收器至少已经接收一个字节的数，且之后在 UART_RX_TOUT_THRHD 位的时间内，总线一直处于 idle 状态将触发此中断。
- UART_BRK_DET_INT：当接收器在停止位之后检测到一个 NULL（即传输一个 NULL 的时间内保持逻辑低电平）时触发此中断。
- UART_CTS_CHG_INT：当接收器检测到 CTS_n 信号的沿变化时触发此中断。
- UART_DSR_CHG_INT：当接收器检测到 DSR_n 信号的沿变化时触发此中断。
- UART_RXFIFO_OVF_INT：当接收器接收到的数据量多于 FIFO 的存储量时触发此中断。
- UART_FRM_ERR_INT：当接收器检测到数据帧错误时触发此中断。
- UART_PARITY_ERR_INT：当接收器检测到校验位错误时触发此中断。
- UART_TXFIFO_EMPTY_INT：当发送 FIFO 中的数据量少于 UART_TXFIFO_EMPTY_THRHD 所指定的值时触发此中断。
- UART_RXFIFO_FULL_INT：当接收器接收到的数据多于 UART_RXFIFO_FULL_THRHD 所指定的值时触发此中断。
- UART_WAKEUP_INT：UART 被唤醒时产生此中断。

UHCI 中断源如下所列：

- UHCI_APP_CTRL1_INT：软件置位 UHCI_APP_CTRL1_INT_RAW 时触发此中断。
- UHCI_APP_CTRL0_INT：软件置位 UHCI_APP_CTRL0_INT_RAW 时触发此中断。
- UHCI_OUTLINK_EOF_ERR_INT：当检测到发送链表描述符中的 EOF 有错误时触发此中断。

- UHCI_SEND_A_REG_Q_INT: 当使用 `always_send` 发送一串短包, UHCI 发送了短包后触发此中断。
- UHCI_SEND_S_REG_Q_INT: 当使用 `single_send` 发送一串短包, UHCI 发送了短包后触发此中断。
- UHCI_TX_HUNG_INT: 当 UHCI 利用 GDMA Tx 通道从 RAM 中读取数据的时间过长时触发此中断。
- UHCI_RX_HUNG_INT: 当 UHCI 利用 GDMA Rx 通道接收数据的时间过长时触发此中断。
- UHCI_TX_START_INT: 当检测到分隔符时触发此中断。
- UHCI_RX_START_INT: 当分隔符已发送时触发此中断。

每个中断源都有一组通用的配置寄存器, 详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [25.7 寄存器列表](#)。

25.6 编程流程

25.6.1 寄存器类型

UART 的所有寄存器都处于 APB_CLK 时钟域。

UART 上的配置寄存器分为两类: 第一类寄存器作用在 APB/AHB 时钟域, 该类寄存器配置后不需要额外的操作; 第二类寄存器作用在 UART_SCLK 时钟域, 因此需要进行跨时钟域处理。在配置完该类寄存器后, 需要通过写寄存器 `UART_REG_UPDATE` 使配置的值同步到 Core 时钟域。当同步完成后, 硬件会将 `UART_REG_UPDATE` 自动清零。推荐用户在配置需要同步的寄存器之后, 检查 `UART_REG_UPDATE` 是否为 0, 确保上一次的同步操作完成之后再执行接下来的寄存器配置。

为了方便用户区分这两类寄存器, 章节 [25.7](#) 中将需要做跨时钟域操作的寄存器放在一起, 该类寄存器名称附带 `_SYNC` 后缀。不需要做跨时钟域操作的寄存器放在一起, 该类寄存器名称不带 `_SYNC` 后缀。

25.6.2 具体步骤

图25.6-1 显示了 UART 模块的编程流程。主要包括：初始化、寄存器配置、启动 UART TX/RX 和数据传输结束。

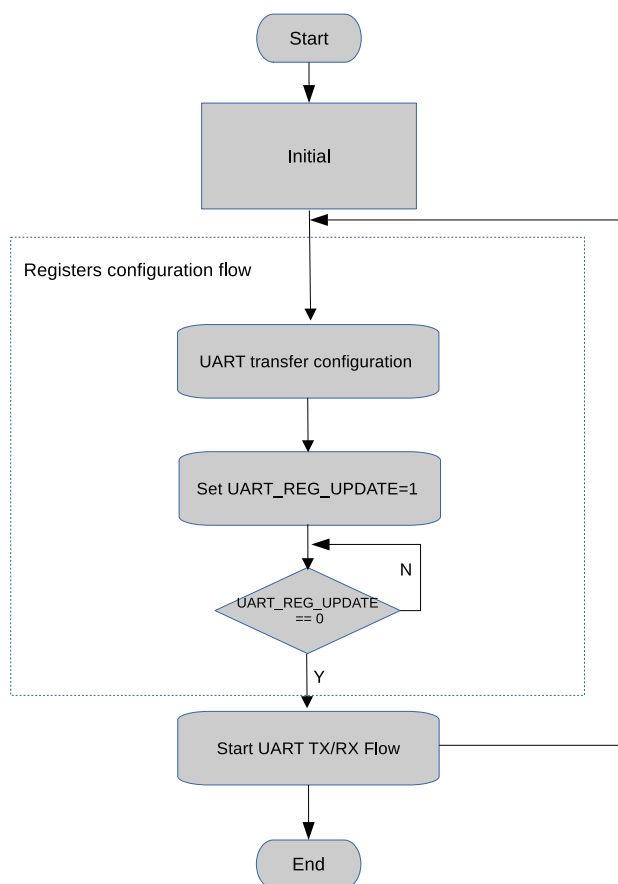


图 25.6-1. UART 编程流程

25.6.2.1 UART_n 模块初始化

UART_n 模块初始化流程如下：

- HP_SYS_CLKRST_RST_EN_UART_n_APB 位写 1；
- HP_SYS_CLKRST_RST_EN_UART_n_APB 位清零。
- HP_SYS_CLKRST_RST_EN_UART_n_CORE 位写 1；
- HP_SYS_CLKRST_RST_EN_UART_n_CORE 位清零。

25.6.2.2 UART_n 通信配置

UART_n 通信配置流程如下：

- 等待 `UART_REG_UPDATE` 为 0，确保上一次同步已经完成；
- 配置 `HP_SYS_CLKRST_UARTn_CLK_SRC_SEL` 选择时钟源；
- 配置 `HP_SYS_CLKRST_UARTn_SCLK_DIV_NUM`、`HP_SYS_CLKRST_UARTn_SCLK_DIV_DENOMINATOR`、`HP_SYS_CLKRST_UARTn_SCLK_DIV_NUMERATOR` 设置预分频器系数；

- 配置 `UART_CLKDIV`、`UART_CLKDIV_FRAG` 设置发送波特率；
- 配置 `UART_BIT_NUM` 设置数据长度；
- 配置 `UART_PARITY_EN`、`UART_PARITY` 设置奇偶校验；
- 可选步骤，根据应用不同存在差异；
- 向 `UART_REG_UPDATE` 写 1，将配置的值同步到 Core 时钟域。

25.6.2.3 启动 UART_n

启动 UART_n TX 发送数据：

- 配置 `UART_TXFIFO_EMPTY_THRHD`，设置 TX FIFO 空阈值；
- 对 `UART_TXFIFO_EMPTY_INT_ENA` 置 0，关闭 `UART_TXFIFO_EMPTY_INT` 中断；
- 向 `UART_RXFIFO_RD_BYTE` 写入需要发送的数据；
- 置位 `UART_TXFIFO_EMPTY_INT_CLR`，清除 `UART_TXFIFO_EMPTY_INT` 中断；
- 置位 `UART_TXFIFO_EMPTY_INT_ENA`，使能 `UART_TXFIFO_EMPTY_INT` 中断；
- 检测 `UART_TXFIFO_EMPTY_INT_ST`，等待发送数据结束。

启动 UART_n RX 数据接收：

- 配置 `UART_RXFIFO_FULL_THRHD`，设置 RX FIFO 满阈值；
- 置位 `UART_RXFIFO_FULL_INT_ENA`，使能 `UART_RXFIFO_FULL_INT` 中断；
- 检测 `UART_RXFIFO_FULL_INT_ST`，等待 RX FIFO 接收数据满；
- 通过读 `UART_RXFIFO_RD_BYTE`，从 RX FIFO 中读出数据，并可通过 `UART_RXFIFO_CNT` 获得当前 RX FIFO 中的接收数据量。

25.7 寄存器列表

25.7.1 UART 寄存器列表

本小节的所有地址均为相对于 UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置寄存器			
UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART 阈值和分配配置	0x0064	R/W
UART 中断寄存器			
UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
UART_INT_ENA_REG	中断使能位	0x000C	R/W
UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
UART_CONFO_SYNC_REG	配置寄存器 0	0x0020	R/W
UART_CONF1_REG	配置寄存器 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
UART_SLEEP_CONFO_REG	UART 睡眠配置寄存器 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART 睡眠配置寄存器 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART 睡眠配置寄存器 2	0x0038	R/W
UART_SWFC_CONFO_SYNC_REG	软件流控字符配置	0x003C	varies
UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 模式配置	0x004C	R/W
UART_CLK_CONF_REG	UART core 时钟配置	0x0088	R/W
UART_REG_UPDATE_REG	UART 寄存器配置更新	0x0098	R/W/SC
UART_ID_REG	UART ID 寄存器	0x009C	R/W
状态寄存器			
UART_STATUS_REG	UART 状态寄存器	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
UART_FSM_STATUS_REG	UART 发送和接收状态	0x0070	RO
UART_AFIFO_STATUS_REG	UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置寄存器			
UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	序列发送后的时序配置	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	超时配置	0x0058	R/W

名称	描述	地址	访问
UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置	0x005C	R/W
自动波特率检测寄存器			
UART_POSPULSE_REG	自动波特率检测高电平脉冲寄存器	0x0074	RO
UART_NEGPULSE_REG	自动波特率检测低电平脉冲寄存器	0x0078	RO
UART_LOWPULSE_REG	自动波特率检测最低电平脉冲持续时间寄存器	0x007C	RO
UART_HIGHPULSE_REG	自动波特率检测最短高电平脉冲持续时间寄存器	0x0080	RO
UART_RXD_CNT_REG	自动波特率检测沿变化计数寄存器	0x0084	RO
版本寄存器			
UART_DATE_REG	UART 版本控制寄存器	0x008C	R/W

25.7.2 LP UART 寄存器列表

本小节的所有地址均为相对于 LP UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置寄存器			
LP_UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
LP_UART_TOUT_CONF_SYNC_REG	LP UART 阈值和分配配置	0x0064	R/W
LP UART 中断寄存器			
LP_UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
LP_UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
LP_UART_INT_ENA_REG	中断使能位	0x000C	R/W
LP_UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
LP_UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
LP_UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
LP_UART_CONFO_SYNC_REG	配置寄存器 0	0x0020	R/W
LP_UART_CONF1_REG	配置寄存器 1	0x0024	R/W
LP_UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
LP_UART_SLEEP_CONFO_REG	LP UART 睡眠配置寄存器 0	0x0030	R/W
LP_UART_SLEEP_CONF1_REG	LP UART 睡眠配置寄存器 1	0x0034	R/W
LP_UART_SLEEP_CONF2_REG	LP UART 睡眠配置寄存器 2	0x0038	R/W
LP_UART_SWFC_CONFO_SYNC_REG	软件流控字符配置	0x003C	varies
LP_UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
LP_UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
LP_UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
LP_UART_DELAY_CONF_SYNC_REG	RS485 模式配置	0x004C	R/W
LP_UART_CLK_CONF_REG	LP UART core 时钟配置	0x0088	R/W
LP_UART_REG_UPDATE_REG	LP UART 寄存器配置更新	0x0098	R/W/SC

名称	描述	地址	访问
LP_UART_ID_REG	LP UART ID 寄存器	0x009C	R/W
状态寄存器			
LP_UART_STATUS_REG	LP UART 状态寄存器	0x001C	RO
LP_UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
LP_UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
LP_UART_FSM_STATUS_REG	LP UART 发送和接收状态	0x0070	RO
LP_UART_AFIFO_STATUS_REG	LP UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置寄存器			
LP_UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置	0x0050	R/W
LP_UART_AT_CMD_POSTCNT_SYNC_REG	序列发送前后的时序配置	0x0054	R/W
LP_UART_AT_CMD_GAPTOOUT_SYNC_REG	超时配置	0x0058	R/W
LP_UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置	0x005C	R/W
版本寄存器			
LP_UART_DATE_REG	LP UART 版本控制寄存器	0x008C	R/W

25.7.3 UHCI 寄存器列表

本小节的所有地址均为相对于 UHCI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
UHCI_CONFO_REG	UHCI 配置寄存器	0x0000	R/W
UHCI_CONF1_REG	UHCI 配置寄存器	0x0014	varies
UHCI_ESCAPE_CONF_REG	转义符配置	0x0020	R/W
UHCI_HUNG_CONF_REG	超时配置	0x0024	R/W
UHCI_ACK_NUM_REG	配置 UHCI ACK 值	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI 快速发送配置寄存器	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 快速发送寄存器	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 快速发送寄存器	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 快速发送寄存器	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 快速发送寄存器	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 快速发送寄存器	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 快速发送寄存器	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 快速发送寄存器	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 快速发送寄存器	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 快速发送寄存器	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 快速发送寄存器	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 快速发送寄存器	0x005C	R/W
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 快速发送寄存器	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 快速发送寄存器	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 快速发送寄存器	0x0068	R/W

名称	描述	地址	访问
UHCI_ESC_CONFO_REG	转义序列配置寄存器 0	0x006C	R/W
UHCI_ESC_CONF1_REG	转义序列配置寄存器 1	0x0070	R/W
UHCI_ESC_CONF2_REG	转义序列配置寄存器 2	0x0074	R/W
UHCI_ESC_CONF3_REG	转义序列配置寄存器 3	0x0078	R/W
UHCI_PKT_THRES_REG	包长度配置寄存器	0x007C	R/W
UHCI 中断寄存器			
UHCI_INT_RAW_REG	原始中断状态	0x0004	varies
UHCI_INT_ST_REG	屏蔽中断状态	0x0008	RO
UHCI_INT_ENA_REG	中断使能位	0x000C	R/W
UHCI_INT_CLR_REG	中断清除位	0x0010	WT
UHCI 状态寄存器			
UHCI_STATE0_REG	UHCI 接收状态	0x0018	RO
UHCI_STATE1_REG	UHCI 发送状态	0x001C	RO
UHCI_RX_HEAD_REG	UHCI 包报头寄存器	0x002C	RO
版本寄存器			
UHCI_DATE_REG	UHCI 版本控制寄存器	0x0080	R/W

25.8 寄存器

25.8.1 UART 寄存器

本小节的所有地址均为相对于 UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 25.1. UART_FIFO_REG (0x0000)

(reserved)																UART_RXFIFO_RD_BYTE																
31																8	7								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0								Reset								

UART_RXFIFO_RD_BYTE 表示 UART_n 从 FIFO 读取的数据。
单位：字节。(RO)

Register 25.2. UART_TOUT_CONF_SYNC_REG (0x0064)

(reserved)																UART_RX_TOUT_THRHD						UART_RX_TOUT_FLOW_DIS UART_RX_TOUT_EN			
31													12	11							2	1	0	Reset	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0xa						0	0		

UART_RX_TOUT_EN 配置是否开启 UART 接收器的超时功能。
0：关闭
1：开启
(R/W)

UART_RX_TOUT_FLOW_DIS 配置开启硬件流控时是否关闭 idle 状态计数器计数。
0：开启
1：关闭
(R/W)

UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.3. UART_INT_RAW_REG (0x0004)

(reserved)																															UART_WAKEUP_INT_RAW UART_AT_CMD_CHAR_DET_INT_RAW UART_RS485_CLASH_INT_RAW UART_RS485_FRM_ERR_INT_RAW UART_TX_BRK_IDLE_DONE_INT_RAW UART_TX_DONE_INT_RAW UART_GLITCH_DET_INT_RAW UART_SW_XOFF_INT_RAW UART_SW_XON_INT_RAW UART_RXFIFO_TOUT_INT_RAW UART_CTS_CHG_INT_RAW UART_DSR_CHG_INT_RAW UART_RXFIFO_OVF_INT_RAW UART_FRM_ERR_INT_RAW UART_PARITY_ERR_INT_RAW UART_TXFIFO_EMPTY_INT_RAW UART_RXFIFO_FULL_INT_RAW															
31																								20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset	
0												0												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0		

UART_RXFIFO_FULL_INT_RAW UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/WTC/SS)

UART_PARITY_ERR_INT_RAW UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_FRM_ERR_INT_RAW UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

UART_DSR_CHG_INT_RAW UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_CTS_CHG_INT_RAW UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_BRK_DET_INT_RAW UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XON_INT_RAW UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

UART_GLITCH_DET_INT_RAW UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_DONE_INT_RAW UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW UART_RS485_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_FRM_ERR_INT_RAW UART_RS485_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_CLASH_INT_RAW UART_RS485_CLASH_INT 的原始中断状态。(R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW UART_AT_CMD_CHAR_DET_INT 的原始中断状态。(R/WTC/SS)

UART_WAKEUP_INT_RAW UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 25.4. UART_INT_ST_REG (0x0008)

(reserved)																				UART_WAKEUP_INT_ST UART_AT_CMD_CHAR_DET_INT_ST UART_RS485_CLASH_INT_ST UART_RS485_FRM_ERR_INT_ST UART_TX_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT_ST UART_GLITCH_DET_INT_ST UART_SW_XON_INT_ST UART_SW_XOFF_INT_ST UART_RXFIFO_TOUT_INT_ST UART_CTS_DET_INT_ST UART_DSR_CHG_INT_ST UART_RXFIFO_OVF_INT_ST UART_FRM_ERR_INT_ST UART_PARITY_ERR_INT_ST UART_TXFIFO_EMPTY_INT_ST UART_RXFIFO_FULL_INT_ST																				
31																				20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																				Reset

UART_RXFIFO_FULL_INT_ST UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

UART_TXFIFO_EMPTY_INT_ST UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

UART_PARITY_ERR_INT_ST UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

UART_FRM_ERR_INT_ST UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_OVF_INT_ST UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

UART_DSR_CHG_INT_ST UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

UART_CTS_CHG_INT_ST UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

UART_BRK_DET_INT_ST UART_BRK_DET_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_TOUT_INT_ST UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

UART_SW_XON_INT_ST UART_SW_XON_INT 的屏蔽中断状态。(RO)

UART_SW_XOFF_INT_ST UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

UART_GLITCH_DET_INT_ST UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_DONE_INT_ST UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_IDLE_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_DONE_INT_ST UART_TX_DONE_INT 的屏蔽中断状态。(RO)

UART_RS485_PARITY_ERR_INT_ST UART_RS485_PARITY_ERR_INT 的屏蔽中断状态。(RO)

UART_RS485_FRM_ERR_INT_ST UART_RS485_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RS485_CLASH_INT_ST UART_RS485_CLASH_INT 的屏蔽中断状态。(RO)

UART_AT_CMD_CHAR_DET_INT_ST UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。(RO)

UART_WAKEUP_INT_ST UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 25.5. UART_INT_ENA_REG (0x000C)

(reserved)																															UART_WAKEUP_INT_ENA UART_AT_CMD_CHAR_DET_INT_ENA UART_RS485_CLASH_INT_ENA UART_RS485_FRM_ERR_INT_ENA UART_RS485_PARITY_ERR_INT_ENA UART_TX_DONE_INT_ENA UART_TX_BRK_IDLE_DONE_INT_ENA UART_GLITCH_DET_INT_ENA UART_SW_XOFF_INT_ENA UART_SW_XON_INT_ENA UART_RXFIFO_TOUT_INT_ENA UART_CTS_DET_INT_ENA UART_DSR_CHG_INT_ENA UART_RXFIFO_OVF_INT_ENA UART_FRM_ERR_INT_ENA UART_PARITY_ERR_INT_ENA UART_TXFIFO_EMPTY_INT_ENA UART_RXFIFO_FULL_INT_ENA															
31																								20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0												0												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

UART_RXFIFO_FULL_INT_ENA 写 1 使能 UART_RXFIFO_FULL_INT。(R/W)

UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 UART_TXFIFO_EMPTY_INT。(R/W)

UART_PARITY_ERR_INT_ENA 写 1 使能 UART_PARITY_ERR_INT。(R/W)

UART_FRM_ERR_INT_ENA 写 1 使能 UART_FRM_ERR_INT。(R/W)

UART_RXFIFO_OVF_INT_ENA 写 1 使能 UART_RXFIFO_OVF_INT。(R/W)

UART_DSR_CHG_INT_ENA 写 1 使能 UART_DSR_CHG_INT。(R/W)

UART_CTS_CHG_INT_ENA 写 1 使能 UART_CTS_CHG_INT。(R/W)

UART_BRK_DET_INT_ENA 写 1 使能 UART_BRK_DET_INT。(R/W)

UART_RXFIFO_TOUT_INT_ENA 写 1 使能 UART_RXFIFO_TOUT_INT。(R/W)

UART_SW_XON_INT_ENA 写 1 使能 UART_SW_XON_INT。(R/W)

UART_SW_XOFF_INT_ENA 写 1 使能 UART_SW_XOFF_INT。(R/W)

UART_GLITCH_DET_INT_ENA 写 1 使能 UART_GLITCH_DET_INT。(R/W)

UART_TX_BRK_DONE_INT_ENA 写 1 使能 UART_TX_BRK_DONE_INT。(R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 UART_TX_BRK_IDLE_DONE_INT。(R/W)

UART_TX_DONE_INT_ENA 写 1 使能 UART_TX_DONE_INT。(R/W)

UART_RS485_PARITY_ERR_INT_ENA 写 1 使能 UART_RS485_PARITY_ERR_INT。(R/W)

UART_RS485_FRM_ERR_INT_ENA 写 1 使能 UART_RS485_FRM_ERR_INT。(R/W)

UART_RS485_CLASH_INT_ENA 写 1 使能 UART_RS485_CLASH_INT。(R/W)

UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 UART_AT_CMD_CHAR_DET_INT。(R/W)

UART_WAKEUP_INT_ENA 写 1 使能 UART_WAKEUP_INT。(R/W)

Register 25.6. UART_INT_CLR_REG (0x0010)

(reserved)																UART_WAKEUP_INT_CLR UART_AT_CMD_CHAR_DET_INT_CLR UART_RS485_CLASH_INT_CLR UART_RS485_FRM_ERR_INT_CLR UART_TX_DONE_INT_CLR UART_TX_BRK_IDLE_DONE_INT_CLR UART_GLITCH_DET_INT_CLR UART_SW_XOFF_INT_CLR UART_RXFIFO_TOUT_INT_CLR UART_BRK_DET_INT_CLR UART_CTS_CHG_INT_CLR UART_DSR_CHG_INT_CLR UART_RXFIFO_OVF_INT_CLR UART_FRM_ERR_INT_CLR UART_PARITY_ERR_INT_CLR UART_TXFIFO_EMPTY_INT_CLR UART_RXFIFO_FULL_INT_CLR																	
31													20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset				

UART_RXFIFO_FULL_INT_CLR 写 1 清除 UART_RXFIFO_FULL_INT。(WT)

UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 UART_TXFIFO_EMPTY_INT。(WT)

UART_PARITY_ERR_INT_CLR 写 1 清除 UART_PARITY_ERR_INT。(WT)

UART_FRM_ERR_INT_CLR 写 1 清除 UART_FRM_ERR_INT。(WT)

UART_RXFIFO_OVF_INT_CLR 写 1 清除 UART_RXFIFO_OVF_INT。(WT)

UART_DSR_CHG_INT_CLR 写 1 清除 UART_DSR_CHG_INT。(WT)

UART_CTS_CHG_INT_CLR 写 1 清除 UART_CTS_CHG_INT。(WT)

UART_BRK_DET_INT_CLR 写 1 清除 UART_BRK_DET_INT。(WT)

UART_RXFIFO_TOUT_INT_CLR 写 1 清除 UART_RXFIFO_TOUT_INT。(WT)

UART_SW_XON_INT_CLR 写 1 清除 UART_SW_XON_INT。(WT)

UART_SW_XOFF_INT_CLR 写 1 清除 UART_SW_XOFF_INT。(WT)

UART_GLITCH_DET_INT_CLR 写 1 清除 UART_GLITCH_DET_INT。(WT)

UART_TX_BRK_DONE_INT_CLR 写 1 清除 UART_TX_BRK_DONE_INT。(WT)

UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 UART_TX_BRK_IDLE_DONE_INT。(WT)

UART_TX_DONE_INT_CLR 写 1 清除 UART_TX_DONE_INT。(WT)

UART_RS485_PARITY_ERR_INT_CLR 写 1 清除 UART_RS485_PARITY_ERR_INT。(WT)

UART_RS485_FRM_ERR_INT_CLR 写 1 清除 UART_RS485_FRM_ERR_INT。(WT)

UART_RS485_CLASH_INT_CLR 写 1 清除 UART_RS485_CLASH_INT。(WT)

UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 UART_AT_CMD_CHAR_DET_INT。(WT)

UART_WAKEUP_INT_CLR 写 1 清除 UART_WAKEUP_INT。(WT)

Register 25.7. UART_CLKDIV_SYNC_REG (0x0014)

(reserved)								UART_CLKDIV_FRAG												(reserved)								UART_CLKDIV																								
31								24								23				20				19								12				11								0								
0								0								0				0				0								0				0								0x2b6								Reset

- UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)
- UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 25.8. UART_RX_FILT_REG (0x0018)

(reserved)																								UART_GLITCH_FILT_EN		UART_GLITCH_FILT												
31																								9	8	7	0											
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0	0x8												Reset	

- UART_GLITCH_FILT 配置滤波长度。
单位：UART Core 时钟周期。
宽度小于该数值的输入脉冲会被忽略。(R/W)
- UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。
0：关闭
1：开启
(R/W)

Register 25.9. UART_CONF0_SYNC_REG (0x0020)

(reserved)								UART_TXFIFO_RST		UART_RXFIFO_RST		UART_SW_RST		UART_MEM_CLK_EN		UART_AUTOBAUD_EN		UART_ERR_WR_MASK		UART_TXD_INV		UART_RXD_INV		UART_IRDA_EN		UART_TX_FLOW_EN		UART_IRDA_RX_INV		UART_IRDA_TX_INV		UART_IRDA_WCTL		UART_IRDA_TX_EN		UART_TXD_DPLX		UART_STOP_BRK		UART_BIT_NUM		UART_PARITY_EN		UART_PARITY	
31								24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0													
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		3	0	0	Reset													

UART_PARITY 配置奇偶检验方式。

- 0: 偶校验
 - 1: 奇校验
- (R/W)

UART_PARITY_EN 配置是否开启 UART 奇偶校验。

- 0: 关闭
 - 1: 开启
- (R/W)

UART_BIT_NUM 配置数据位长度。

- 0: 5 位
 - 1: 6 位
 - 2: 7 位
 - 3: 8 位
- (R/W)

UART_STOP_BIT_NUM 配置停止位长度。

- 0: 无效值，没有作用
 - 1: 1 位
 - 2: 1.5 位
 - 3: 2 位
- (R/W)

UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

- 0: 不发送
 - 1: 发送
- (R/W)

UART_IRDA_DPLX 配置是否开启 IrDA 回环测试模式。

- 0: 关闭
 - 1: 开启
- (R/W)

UART_IRDA_TX_EN 配置是否开启 IrDA 发送器。

- 0: 关闭
 - 1: 开启
- (R/W)

见下页...

Register 25.9. UART_CONFO_SYNC_REG (0x0020)

接上页...

UART_IRDA_WCTL 配置 IrDA 发送器的第 11 位。

0: 该位为 0

1: 该位与第 10 位相同

(R/W)

UART_IRDA_TX_INV 配置是否翻转 IrDA 发送器的电平。

0: 不翻转

1: 翻转

(R/W)

UART_IRDA_RX_INV 配置是否翻转 IrDA 接收器的电平。

0: 不翻转

1: 翻转

(R/W)

UART_LOOPBACK 配置是否开启 UART 回环测试模式。

0: 关闭

1: 开启

(R/W)

UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

0: 关闭

1: 开启

(R/W)

UART_IRDA_EN 配置是否开启 IrDA。

0: 关闭

1: 开启

(R/W)

UART_RXD_INV 配置是否翻转 UART RXD 信号电平。

0: 不翻转

1: 翻转

(R/W)

UART_TXD_INV 配置是否翻转 UART TXD 信号电平。

0: 不翻转

1: 翻转

(R/W)

UART_DIS_RX_DAT_OVF 配置是否关闭 UART 接收器数据溢出检测。

0: 开启

1: 关闭

(R/W)

见下页...

Register 25.9. UART_CONFO_SYNC_REG (0x0020)

[接上页...](#)

UART_ERR_WR_MASK 配置接收数据有错误时，是否仍将其存储至 FIFO。

- 0: 存储
 - 1: 不存储
- (R/W)

UART_AUTOBAUD_EN 配置是否开启波特率检测。

- 0: 关闭
 - 1: 开启
- (R/W)

UART_MEM_CLK_EN 配置是否开启 UART 存储器门控。

- 0: 关闭
 - 1: 开启
- (R/W)

UART_SW_RTS 配置软件流控使用的 RTS 信号。

- 0: 允许发送端发送数据
 - 1: 禁止发送端发送数据
- (R/W)

UART_RXFIFO_RST 配置是否复位 UART RX FIFO。

- 0: 不复位
 - 1: 复位
- (R/W)

UART_TXFIFO_RST 配置是否复位 UART TX FIFO。

- 0: 不复位
 - 1: 复位
- (R/W)

Register 25.10. UART_CONF1_REG (0x0024)

(reserved)										UART_CLK_EN UART_SW_DTR UART_DTR_INV UART_RTS_INV UART_DSR_INV UART_CTS_INV							UART_TXFIFO_EMPTY_THRHD							UART_RXFIFO_FULL_THRHD										
31										22		21	20	19	18	17	16	15							8		7	0						
0 0 0 0 0 0 0 0 0 0										0		0	0	0	0	0	0x60									0x60							Reset	

UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。
单位：字节。(R/W)

UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。
单位：字节。(R/W)

UART_CTS_INV 配置是否翻转 UART CTS 信号电平。
0：不翻转
1：翻转
(R/W)

UART_DSR_INV 配置是否翻转 UART DSR 信号电平。
0：不翻转
1：翻转
(R/W)

UART_RTS_INV 配置是否翻转 UART RTS 信号电平。
0：不翻转
1：翻转
(R/W)

UART_DTR_INV 配置是否翻转 UART DTR 信号电平。
0：不翻转
1：翻转
(R/W)

UART_SW_DTR 配置软件流控使用的 DTR 信号。
0：发送数据未准备完毕，处于不可用状态
1：发送数据准备完毕，处于可用状态
(R/W)

UART_CLK_EN 配置时钟门控。
0：仅在应用写寄存器时开启时钟
1：一直强制为寄存器开启时钟
(R/W)

Register 25.11. UART_HWFC_CONF_SYNC_REG (0x002C)

(reserved)																UART_RX_FLOW_EN		UART_RX_FLOW_THRHD		
31																	9	8	7	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	0x0		Reset	

UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。
单位：字节。(R/W)

UART_RX_FLOW_EN 配置是否开启 UART 接收器。
0：关闭
1：开启
(R/W)

Register 25.12. UART_SLEEP_CONFO_REG (0x0030)

UART_WK_CHAR4								UART_WK_CHAR3								UART_WK_CHAR2								UART_WK_CHAR1								
3124								2316								158								70								
0x0								0x0								0x0								0x0								Reset

UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 25.13. UART_SLEEP_CONF1_REG (0x0034)

(reserved)																UART_WK_CHAR0			
31																8	7	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x0		Reset	

UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 25.14. UART_SLEEP_CONF2_REG (0x0038)

(reserved)				UART_WK_MODE_SEL				UART_WK_CHAR_MASK				UART_WK_CHAR_NUM				UART_RX_WAKE_UP_THRHD				UART_ACTIVE_THRESHOLD			
31	28	27	26	25	21	20	18	17	10	9													0
0	0	0	0	0	0	0x0				0x5				1				0xf0				Reset	

UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。
单位：字节。(R/W)

UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。
0：不屏蔽
1：屏蔽
(R/W)

UART_WK_MODE_SEL 配置选取哪个唤醒模式。不同模式的含义详见 25.4.8 唤醒 小节。
0：唤醒模式 0
1：唤醒模式 1
2：唤醒模式 2
3：唤醒模式 3
(R/W)

Register 25.15. UART_SWFC_CONFO_SYNC_REG (0x003C)

(reserved)																UART_SEND_XOFF				UART_SEND_XON				UART_FORCE_XOFF				UART_FORCE_XON				UART_XONOFF_DEL				UART_SW_FLOW_CON_EN				UART_XON_XOFF_STILL_SEND				UART_XOFF_CHAR				UART_XON_CHAR							
31								23								22		21		20		19		18		17		16		15								8								7		0							
0		0		0		0		0		0		0		0		0		0		0		0		0		0x13								0x11								Reset													

- UART_XON_CHAR 配置 XON 流控字符。(R/W)
- UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)
- UART_XON_XOFF_STILL_SEND 配置 UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。
0: 不能发送
1: 可以发送
(R/W)
- UART_SW_FLOW_CON_EN 配置是否开启软件流控。
0: 关闭
1: 开启
(R/W)
- UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。
0: 不移除
1: 移除
(R/W)
- UART_FORCE_XON 配置是否让发送器继续发送数据。
0: 不发送
1: 继续发送
(R/W)
- UART_FORCE_XOFF 配置是否让发送器停止发送数据。
0: 不停止
1: 停止
(R/W)
- UART_SEND_XON 配置是否发送 XON 字符。
0: 不发送
1: 发送
(R/W/SS/SC)
- UART_SEND_XOFF 配置是否发送 XOFF 字符。
0: 不发送
1: 发送
(R/W/SS/SC)

Register 25.16. UART_SWFC_CONF1_REG (0x0040)

(reserved)																UART_XOFF_THRESHOLD								UART_XON_THRESHOLD																															
31																16								15								8								7								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0xe0								0x0								Reset																							

UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。
单位：字节。(R/W)

UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。
单位：字节。(R/W)

Register 25.17. UART_TXBRK_CONF_SYNC_REG (0x0044)

(reserved)																UART_TX_BRK_NUM																	
31																8	7	0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0xa																Reset	

UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。
仅在 UART_TXD_BRK 为 1 时有效。(R/W)

Register 25.18. UART_IDLE_CONF_SYNC_REG (0x0048)

(reserved)																UART_TX_IDLE_NUM										UART_RX_IDLE_THRHD													
31													20			19													10			9			0				
0 0 0 0 0 0 0 0 0 0 0 0 0													0x100													0x100													Reset

UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。
单位：位时间（传输一位所需的时间）。(R/W)

UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.19. UART_RS485_CONF_SYNC_REG (0x004C)

(reserved)																				UART_RS485_TX_DLY_NUM		UART_RS485_RX_DLY_NUM		UART_RS485RXBY_TX_EN		UART_RS485TX_RX_EN		UART_DLI_EN		UART_DLO_EN		UART_RS485_EN	
31										10										9	6	5	4	3	2	1	0						
0										0										0	0	0	0	0	0	0	0	Reset					

UART_RS485_EN 配置是否开启 RS485 模式。

0: 关闭

1: 开启

(R/W)

UART_DLO_EN 配置是否在起始位之前增加一位延时。

0: 不增加

1: 增加

(R/W)

UART_DL1_EN 配置是否在停止位之后增加一位延时。

0: 不增加

1: 增加

(R/W)

UART_RS485TX_RX_EN 配置发送器在 RS485 模式下发送数据时，是否开启接收器接收数据。

0: 关闭

1: 开启

(R/W)

UART_RS485RXBY_TX_EN 配置 RS484 接收器线路繁忙时，是否开启 RS485 发送器发送数据。

0: 关闭

1: 开启

(R/W)

UART_RS485_RX_DLY_NUM 配置接收器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

UART_RS485_TX_DLY_NUM 配置发送器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

Register 25.20. UART_CLK_CONF_REG (0x0088)

(reserved)																UART_RX_RST_CORE																UART_TX_RST_CORE																UART_RX_SCLK_EN																UART_TX_SCLK_EN																(reserved)															
31				28				27		26		25		24		23																																		0																																													
0				0				0		0		1		1		0																																Reset																																															

UART_TX_SCLK_EN 配置是否开启 UART TX 时钟。

0: 关闭

1: 开启

(R/W)

UART RX SCLK EN 配置是否开启 UART RX 时钟。

0: 关闭

1: 开启

(R/W)

UART_TX_RST_CORE 向此位先写 1 后写 0，复位 UART TX。(R/W)

UART_RX_RST_CORE 向此位先写1后写0，复位 UART RX。(R/W)

Register 25.21. UART_STATUS_REG (0x001C)

UART_TXD UART_RTSN UART_DTRN (reserved)								UART_TXFIFO_CNT								UART_RXD UART_CTSN UART_DSRRN (reserved)								UART_RXFIFO_CNT							
31	30	29	28	24				23	16						15	14	13	12	8				7	0							
1	1	1	0	0	0	0	0	0						1	1	0	0	0	0	0	0	0				Reset					

UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

UART_DSRN 表示内部 UART DSR 信号的电平值。(RO)

UART_CTSN 表示内部 UART CTS 信号的电平值。(RO)

UART_RXD 表示内部 UART RXD 信号的电平值。(RO)

UART TXFIFO CNT 表示 TX FIFO 中的数据量。

单位：字节。(RO)

UART_DTRN 表示内部 UART DTR 信号的电平。(RO)

UART RTSN 表示内部 UART RTS 信号的电平。(RO)

UART_TXD 表示内部 UART TXD 信号的电平。(RO)

Register 25.22. UART_MEM_TX_STATUS_REG (0x0068)

(reserved)																UART_TX_SRAM_RADDR				(reserved)		UART_TX_SRAM_WADDR			
31																17	16	9	8	7					0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0	0	0x0	Reset						

- UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)
- UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

Register 25.23. UART_MEM_RX_STATUS_REG (0x006C)

(reserved)																UART_RX_SRAM_WADDR				(reserved)		UART_RX_SRAM_RADDR						
31																17	16				9	8	7					0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x80			0	0x80			Reset					

- UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)
- UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

Register 25.24. UART_FSM_STATUS_REG (0x0070)

(reserved)																								UART_ST_UTX_OUT				UART_ST_URX_OUT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
31																								8	7	4		3	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- UART_ST_URX_OUT 表示 UART 接收器的状态。(RO)
- UART_ST_UTX_OUT 表示 UART 发送器的状态。(RO)

Register 25.25. UART_AFIFO_STATUS_REG (0x0090)

(reserved)																UART_RX_AFIFO_EMPTY UART_RX_AFIFO_FULL UART_TX_AFIFO_EMPTY UART_TX_AFIFO_FULL			
31													4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
																1	0	1	0
Reset																			

UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0: 未
1: 已
(RO)

UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0: 不为
1: 为
(RO)

UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0: 未
1: 已
(RO)

UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0: 不为
1: 为
(RO)

Register 25.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)

(reserved)																UART_PRE_IDLE_NUM																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x901																Reset															

UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 25.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)

(reserved)																UART_POST_IDLE_NUM																
31																16	15															0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x901																Reset

UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.28. UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)

(reserved)																UART_RX_GAP_TOUT																
31																0																
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																11																Reset

UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。
单位：位时间（传输一位所需的时间）。(R/W)

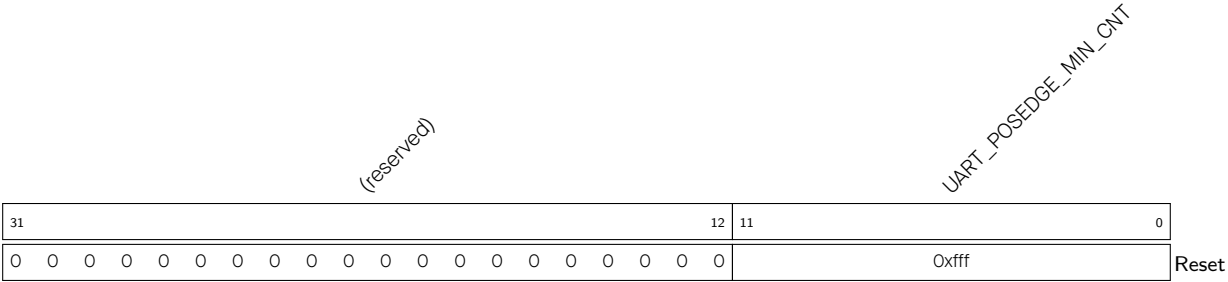
Register 25.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)

(reserved)																UART_CHAR_NUM								UART_AT_CMD_CHAR								
3116																158								70								
0000000000000000																0x3								0x2b								Reset

UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

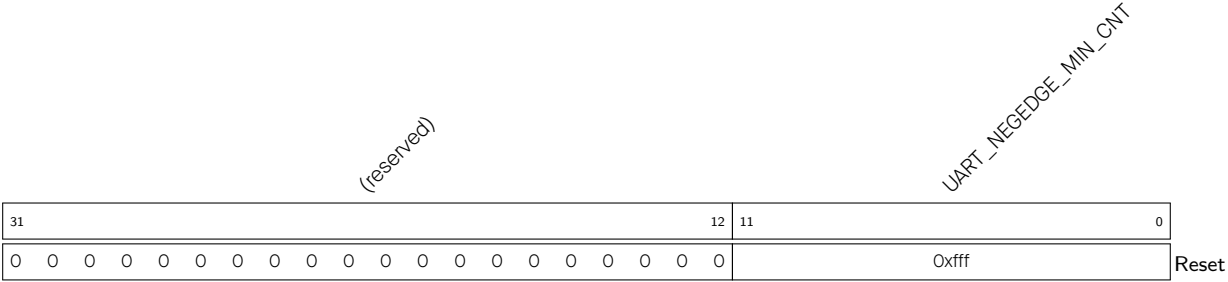
UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 25.30. UART_POSPULSE_REG (0x0074)



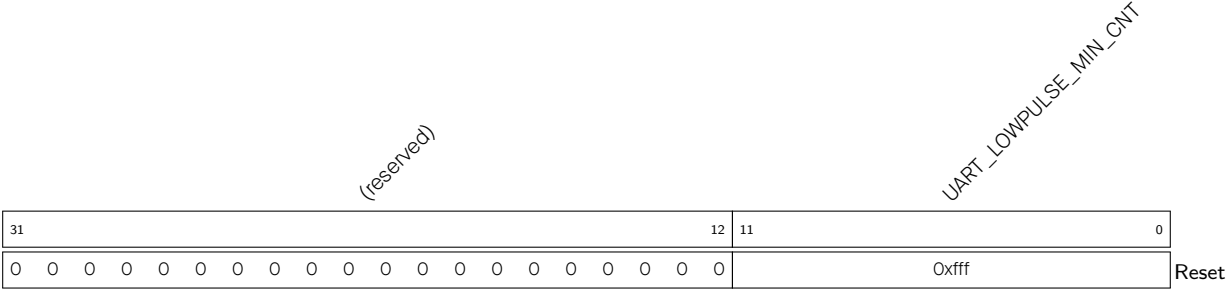
UART_POSEDGE_MIN_CNT 表示两个上升沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 25.31. UART_NEGPULSE_REG (0x0078)



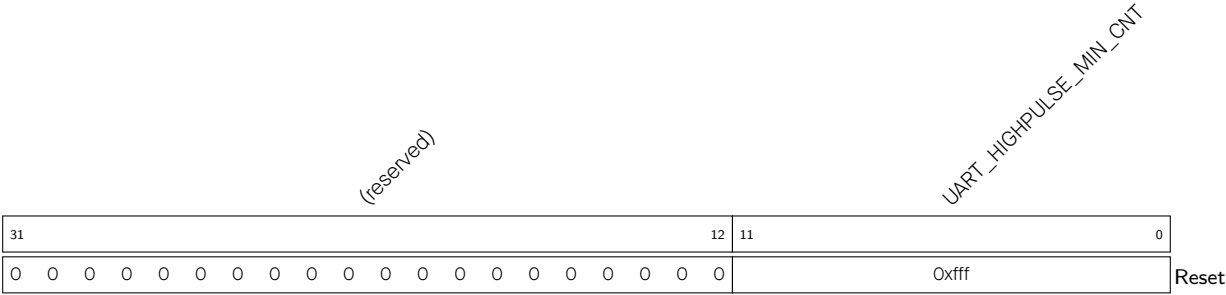
UART_NEGEDGE_MIN_CNT 两个下降沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 25.32. UART_LOWPULSE_REG (0x007C)



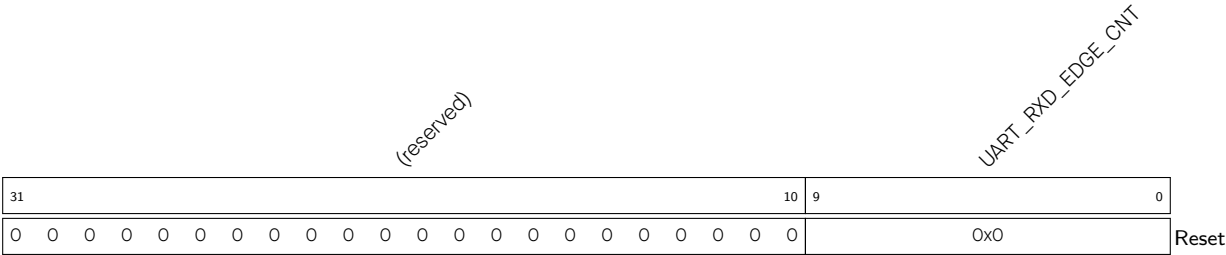
UART_LOWPULSE_MIN_CNT 表示低电平脉冲的最短持续时间，用于波特率检测。
单位：APB_CLK 时钟周期。(RO)

Register 25.33. UART_HIGHPULSE_REG (0x0080)



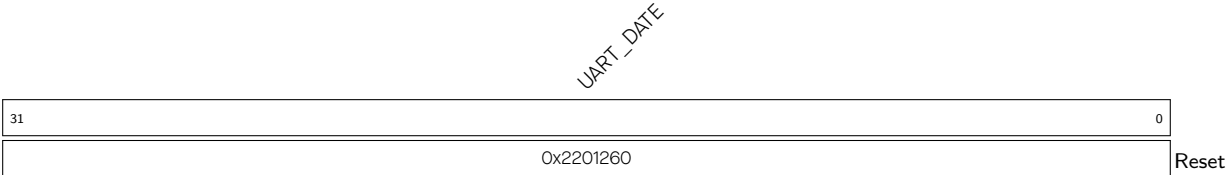
UART_HIGHPULSE_MIN_CNT 表示高电平脉冲最长持续时间，用于波特率检测。
单位：APB_CLK 时钟周期。(RO)

Register 25.34. UART_RXD_CNT_REG (0x0084)



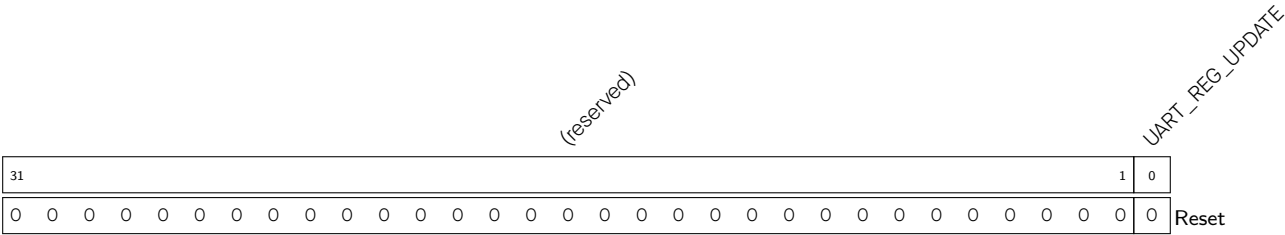
UART_RXD_EDGE_CNT 表示 RXD 沿变化的次数。用于波特率检测。(RO)

Register 25.35. UART_DATE_REG (0x008C)



UART_DATE 版本控制寄存器。(R/W)

Register 25.36. UART_REG_UPDATE_REG (0x0098)



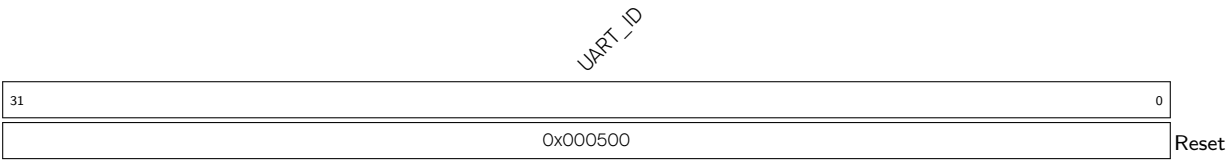
UART_REG_UPDATE 配置是否同步寄存器

0: 不同步

1: 同步

(R/W/SC)

Register 25.37. UART_ID_REG (0x009C)



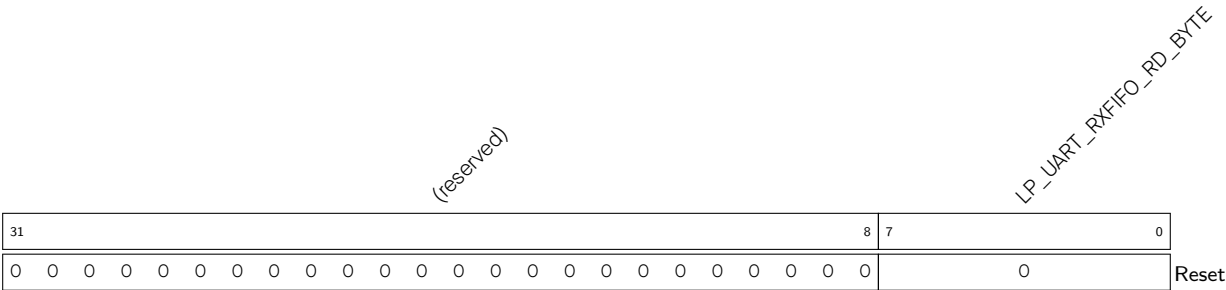
UART_ID 配置 UART ID。(R/W)

25.8.2 LP UART 寄存器

本小节的所有地址均为相对于 LP UART 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

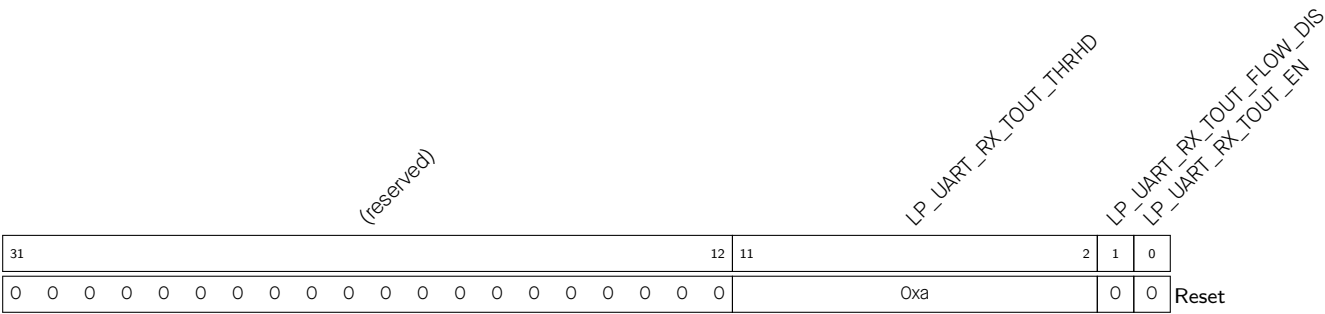
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 25.38. LP_UART_FIFO_REG (0x0000)



LP_UART_RXFIFO_RD_BYTE 表示 LP UART *n* 从 FIFO 读取的数据。
单位：字节。(RO)

Register 25.39. LP_UART_TOUT_CONF_SYNC_REG (0x0064)



LP_UART_RX_TOUT_EN 配置是否开启 LP UART 接收器的超时功能。
0：关闭
1：开启
(R/W)

LP_UART_RX_TOUT_FLOW_DIS 配置开启硬件流控时是否关闭 idle 状态计数器计数。
0：无效值，没有作用
1：关闭
(R/W)

LP_UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.40. LP_UART_INT_RAW_REG (0x0004)

(reserved)																LP_UART_WAKEUP_INT_RAW LP_UART_AT_CMD_CHAR_DET_INT_RAW (reserved) LP_UART_TX_DONE_INT_RAW LP_UART_TX_BRK_IDLE_DONE_INT_RAW LP_UART_TX_BRK_DONE_INT_RAW LP_UART_GLITCH_DET_INT_RAW LP_UART_SW_XOFF_INT_RAW LP_UART_SW_XON_INT_RAW LP_UART_RXFIFO_TOUT_INT_RAW LP_UART_CTS_DET_INT_RAW LP_UART_DSR_CHG_INT_RAW LP_UART_RXFIFO_OVF_INT_RAW LP_UART_FRM_ERR_INT_RAW LP_UART_PARITY_ERR_INT_RAW LP_UART_TXFIFO_EMPTY_INT_RAW LP_UART_RXFIFO_FULL_INT_RAW																	
31													20	19	18	17	15		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Reset				

Reset

LP_UART_RXFIFO_FULL_INT_RAW LP_UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TXFIFO_EMPTY_INT_RAW LP_UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/WTC/SS)

LP_UART_PARITY_ERR_INT_RAW LP_UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

LP_UART_FRM_ERR_INT_RAW LP_UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

LP_UART_RXFIFO_OVF_INT_RAW LP_UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

LP_UART_DSR_CHG_INT_RAW LP_UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

LP_UART_CTS_CHG_INT_RAW LP_UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

LP_UART_BRK_DET_INT_RAW LP_UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_RXFIFO_TOUT_INT_RAW LP_UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

LP_UART_SW_XON_INT_RAW LP_UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

LP_UART_SW_XOFF_INT_RAW LP_UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

LP_UART_GLITCH_DET_INT_RAW LP_UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TX_BRK_DONE_INT_RAW LP_UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

LP_UART_TX_BRK_IDLE_DONE_INT_RAW LP_UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。
(R/WTC/SS)

LP_UART_TX_DONE_INT_RAW LP_UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

LP_UART_AT_CMD_CHAR_DET_INT_RAW LP_UART_AT_CMD_CHAR_DET_INT 的原始中断状态。
(R/WTC/SS)

LP_UART_WAKEUP_INT_RAW LP_UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 25.41. LP_UART_INT_ST_REG (0x0008)

(reserved)																LP_UART_WAKEUP_INT_ST LP_UART_AT_CMD_CHAR_DET_INT_ST (reserved) LP_UART_TX_DONE_INT_ST LP_UART_TX_BRK_DONE_INT_ST LP_UART_TX_BRK_IDLE_DONE_INT_ST LP_UART_GLITCH_DET_INT_ST LP_UART_SW_XOFF_INT_ST LP_UART_SW_XON_INT_ST LP_UART_RXFIFO_TOUT_INT_ST LP_UART_CTS_CHG_INT_ST LP_UART_DSR_CHG_INT_ST LP_UART_RXFIFO_OVF_INT_ST LP_UART_FRM_ERR_INT_ST LP_UART_PARITY_ERR_INT_ST LP_UART_TXFIFO_EMPTY_INT_ST LP_UART_RXFIFO_FULL_INT_ST																
31													20	19	18	17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset				

LP_UART_RXFIFO_FULL_INT_ST LP_UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

LP_UART_TXFIFO_EMPTY_INT_ST LP_UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

LP_UART_PARITY_ERR_INT_ST LP_UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

LP_UART_FRM_ERR_INT_ST LP_UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

LP_UART_RXFIFO_OVF_INT_ST LP_UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

LP_UART_DSR_CHG_INT_ST LP_UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

LP_UART_CTS_CHG_INT_ST LP_UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

LP_UART_BRK_DET_INT_ST LP_UART_BRK_DET_INT 的屏蔽中断状态。(RO)

LP_UART_RXFIFO_TOUT_INT_ST LP_UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

LP_UART_SW_XON_INT_ST LP_UART_SW_XON_INT 的屏蔽中断状态。(RO)

LP_UART_SW_XOFF_INT_ST LP_UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

LP_UART_GLITCH_DET_INT_ST LP_UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

LP_UART_TX_BRK_DONE_INT_ST LP_UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_TX_BRK_IDLE_DONE_INT_ST LP_UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。
(RO)

LP_UART_TX_DONE_INT_ST LP_UART_TX_DONE_INT 的屏蔽中断状态。(RO)

LP_UART_AT_CMD_CHAR_DET_INT_ST LP_UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。
(RO)

LP_UART_WAKEUP_INT_ST LP_UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 25.42. LP_UART_INT_ENA_REG (0x000C)

(reserved)																LP_UART_WAKEUP_INT_ENA LP_UART_AT_CMD_CHAR_DET_INT_ENA (reserved) LP_UART_TX_DONE_INT_ENA LP_UART_TX_BRK_IDLE_DONE_INT_ENA LP_UART_TX_BRK_DONE_INT_ENA LP_UART_GLITCH_DET_INT_ENA LP_UART_SW_XOFF_INT_ENA LP_UART_SW_XON_INT_ENA LP_UART_RXFIFO_TOUT_INT_ENA LP_UART_CTS_CHG_INT_ENA LP_UART_DSR_CHG_INT_ENA LP_UART_FRM_ERR_INT_ENA LP_UART_PARITY_ERR_INT_ENA LP_UART_TXFIFO_EMPTY_INT_ENA LP_UART_RXFIFO_FULL_INT_ENA																	
31													20	19	18	17	15		14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset				

LP_UART_RXFIFO_FULL_INT_ENA 写 1 使能 LP_UART_RXFIFO_FULL_INT。(R/W)

LP_UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 LP_UART_TXFIFO_EMPTY_INT。(R/W)

LP_UART_PARITY_ERR_INT_ENA 写 1 使能 LP_UART_PARITY_ERR_INT。(R/W)

LP_UART_FRM_ERR_INT_ENA 写 1 使能 LP_UART_FRM_ERR_INT。(R/W)

LP_UART_RXFIFO_OVF_INT_ENA 写 1 使能 LP_UART_RXFIFO_OVF_INT。(R/W)

LP_UART_DSR_CHG_INT_ENA 写 1 使能 LP_UART_DSR_CHG_INT。(R/W)

LP_UART_CTS_CHG_INT_ENA 写 1 使能 LP_UART_CTS_CHG_INT。(R/W)

LP_UART_BRK_DET_INT_ENA 写 1 使能 LP_UART_BRK_DET_INT。(R/W)

LP_UART_RXFIFO_TOUT_INT_ENA 写 1 使能 LP_UART_RXFIFO_TOUT_INT。(R/W)

LP_UART_SW_XON_INT_ENA 写 1 使能 LP_UART_SW_XON_INT。(R/W)

LP_UART_SW_XOFF_INT_ENA 写 1 使能 LP_UART_SW_XOFF_INT。(R/W)

LP_UART_GLITCH_DET_INT_ENA 写 1 使能 LP_UART_GLITCH_DET_INT。(R/W)

LP_UART_TX_BRK_DONE_INT_ENA 写 1 使能 LP_UART_TX_BRK_DONE_INT。(R/W)

LP_UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 LP_UART_TX_BRK_IDLE_DONE_INT。(R/W)

LP_UART_TX_DONE_INT_ENA 写 1 使能 LP_UART_TX_DONE_INT。(R/W)

LP_UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 LP_UART_AT_CMD_CHAR_DET_INT。(R/W)

LP_UART_WAKEUP_INT_ENA 写 1 使能 LP_UART_WAKEUP_INT。(R/W)

Register 25.43. LP_UART_INT_CLR_REG (0x0010)

(reserved)																LP_UART_WAKEUP_INT_CLR LP_UART_AT_CMD_CHAR_DET_INT_CLR (reserved) LP_UART_TX_DONE_INT_CLR LP_UART_TX_BRK_IDLE_DONE_INT_CLR LP_UART_TX_BRK_DONE_INT_CLR LP_UART_GLITCH_DET_INT_CLR LP_UART_SW_XOFF_INT_CLR LP_UART_SW_XON_INT_CLR LP_UART_RXFIFO_TOUT_INT_CLR LP_UART_CTS_CHG_INT_CLR LP_UART_DSR_CHG_INT_CLR LP_UART_RXFIFO_OVF_INT_CLR LP_UART_FRM_ERR_INT_CLR LP_UART_PARITY_ERR_INT_CLR LP_UART_TXFIFO_EMPTY_INT_CLR LP_UART_RXFIFO_FULL_INT_CLR																																								
31																20																19	18	17			15			14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0																0																0	0	0			0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_UART_RXFIFO_FULL_INT_CLR 写 1 清除 LP_UART_RXFIFO_FULL_INT。(WT)

LP_UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 LP_UART_TXFIFO_EMPTY_INT。(WT)

LP_UART_PARITY_ERR_INT_CLR 写 1 清除 LP_UART_PARITY_ERR_INT。(WT)

LP_UART_FRM_ERR_INT_CLR 写 1 清除 LP_UART_FRM_ERR_INT。(WT)

LP_UART_RXFIFO_OVF_INT_CLR 写 1 清除 LP_UART_RXFIFO_OVF_INT。(WT)

LP_UART_DSR_CHG_INT_CLR 写 1 清除 LP_UART_DSR_CHG_INT。(WT)

LP_UART_CTS_CHG_INT_CLR 写 1 清除 LP_UART_CTS_CHG_INT。(WT)

LP_UART_BRK_DET_INT_CLR 写 1 清除 LP_UART_BRK_DET_INT。(WT)

LP_UART_RXFIFO_TOUT_INT_CLR 写 1 清除 LP_UART_RXFIFO_TOUT_INT。(WT)

LP_UART_SW_XON_INT_CLR 写 1 清除 LP_UART_SW_XON_INT。(WT)

LP_UART_SW_XOFF_INT_CLR 写 1 清除 LP_UART_SW_XOFF_INT。(WT)

LP_UART_GLITCH_DET_INT_CLR 写 1 清除 LP_UART_GLITCH_DET_INT。(WT)

LP_UART_TX_BRK_DONE_INT_CLR 写 1 清除 LP_UART_TX_BRK_DONE_INT。(WT)

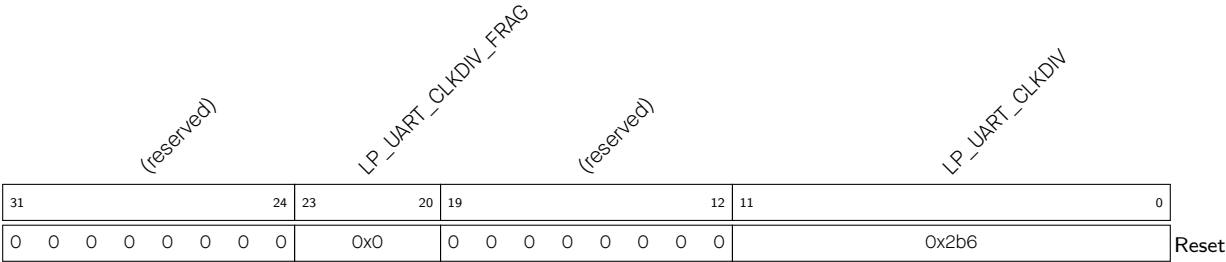
LP_UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 LP_UART_TX_BRK_IDLE_DONE_INT。(WT)

LP_UART_TX_DONE_INT_CLR 写 1 清除 LP_UART_TX_DONE_INT。(WT)

LP_UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 LP_UART_AT_CMD_CHAR_DET_INT。(WT)

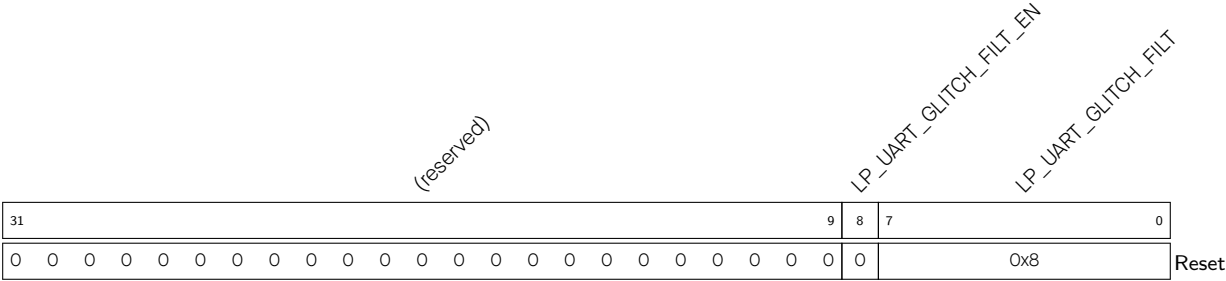
LP_UART_WAKEUP_INT_CLR 写 1 清除 LP_UART_WAKEUP_INT。(WT)

Register 25.44. LP_UART_CLKDIV_SYNC_REG (0x0014)



- LP_UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)
- LP_UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 25.45. LP_UART_RX_FILT_REG (0x0018)



- LP_UART_GLITCH_FILT 配置滤波长度。
单位：UART Core 时钟周期。
宽度小于该数值的输入脉冲会被忽略。(R/W)
- LP_UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。
0：关闭
1：开启
(R/W)

Register 25.46. LP_UART_CONFO_SYNC_REG (0x0020)

(reserved)										LP_UART_TXFIFO_RST LP_UART_RXFIFO_RST LP_UART_SW_RTS (reserved) LP_UART_MEM_CLK_EN LP_UART_ERR_WR_MASK LP_UART_DIS_RX_DAT_OVF (reserved) LP_UART_TXD_INV LP_UART_RXD_INV LP_UART_TX_FLOW_EN LP_UART_LOOPBACK										(reserved)										LP_UART_TXD_BRK LP_UART_STOP_BIT_NUM LP_UART_BIT_NUM LP_UART_PARITY_EN LP_UART_PARITY						
31								24	23	22	21	20	19	18	17	16	15	14	13	12	11								7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							0	1	3	0	0	Reset		

LP_UART_PARITY 配置奇偶检验方式。

- 0: 偶校验
 - 1: 奇校验
- (R/W)

LP_UART_PARITY_EN 配置是否开启 LP UART 奇偶校验。

- 0: 关闭
 - 1: 开启
- (R/W)

LP_UART_BIT_NUM 配置数据位长度。

- 0: 5 位
 - 1: 6 位
 - 2: 7 位
 - 3: 8 位
- (R/W)

LP_UART_STOP_BIT_NUM 配置停止位长度。

- 0: 无效值，没有作用
 - 1: 1 位
 - 2: 1.5 位
 - 3: 2 位
- (R/W)

LP_UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

- 0: 不发送
 - 1: 发送
- (R/W)

LP_UART_LOOPBACK 配置是否开启 LP UART 回环测试模式。

- 0: 关闭
 - 1: 开启
- (R/W)

LP_UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

- 0: 关闭
 - 1: 开启
- (R/W)

见下页...

Register 25.46. LP_UART_CONFO_SYNC_REG (0x0020)

接上页...

LP_UART_RXD_INV 配置是否翻转 LP UART RXD 信号电平。

0: 不翻转

1: 翻转

(R/W)

LP_UART_TXD_INV 配置是否翻转 LP UART TXD 信号电平。

0: 不翻转

1: 翻转

(R/W)

LP_UART_DIS_RX_DAT_OVF 配置是否关闭 LP UART 接收器数据溢出检测。

0: 开启

1: 关闭

(R/W)

LP_UART_ERR_WR_MASK 配置接收数据有错误时，是否仍将其存储至 FIFO。

0: 存储

1: 不存储

(R/W)

LP_UART_MEM_CLK_EN 配置是否开启 LP UART 存储器门控。

0: 关闭

1: 开启

(R/W)

LP_UART_SW_RTS 配置软件流控使用的 RTS 信号。

0: 允许发送端发送数据

1: 禁止发送端发送数据

(R/W)

LP_UART_RXFIFO_RST 配置是否复位 LP UART RX FIFO。

0: 不复位

1: 复位

(R/W)

LP_UART_TXFIFO_RST 配置是否复位 LP UART TX FIFO。

0: 不复位

1: 复位

(R/W)

Register 25.47. LP_UART_CONF1_REG (0x0024)

(reserved)										LP_UART_CLK_EN		LP_UART_SW_DTR		LP_UART_DTR_INV		LP_UART_RTS_INV		LP_UART_DSR_INV		LP_UART_CTS_INV		LP_UART_TXFIFO_EMPTY_THRHD				(reserved)				LP_UART_RXFIFO_FULL_THRHD				(reserved)			
31											22	21	20	19	18	17	16	15	11		10	8	7			3	2	0									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0xc	0		0	0	0xc	0		0	0	0	0	0	0	0	0	Reset					

LP_UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。
单位：字节。(R/W)

LP_UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。
单位：字节。(R/W)

LP_UART_CTS_INV 配置是否翻转 LP UART CTS 信号电平。
0：不翻转
1：翻转
(R/W)

LP_UART_DSR_INV 配置是否翻转 LP UART DSR 信号电平。
0：不翻转
1：翻转
(R/W)

LP_UART_RTS_INV 配置是否翻转 LP UART RTS 信号电平。
0：不翻转
1：翻转
(R/W)

LP_UART_DTR_INV 配置是否翻转 LP UART DTR 信号电平。
0：不翻转
1：翻转
(R/W)

LP_UART_SW_DTR 配置软件流控使用的 DTR 信号。
0：发送数据未准备完毕，处于不可用状态
1：发送数据准备完毕，处于可用状态
(R/W)

LP_UART_CLK_EN 配置时钟门控。
0：仅在应用写寄存器时开启时钟
1：一直强制为寄存器开启时钟
(R/W)

Register 25.48. LP_UART_HWFC_CONF_SYNC_REG (0x002C)

(reserved)																								9	8	7	3			2	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0	0x0			0	0	0	Reset	

LP_UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。

单位：字节。(R/W)

LP_UART_RX_FLOW_EN 配置是否开启 LP UART 接收器。

0: 关闭

1: 开启

(R/W)

Register 25.49. LP_UART_SLEEP_CONFO_REG (0x0030)

31	24	23	16	15	8	7	0
LP_UART_WK_CHAR4		LP_UART_WK_CHAR3		LP_UART_WK_CHAR2		LP_UART_WK_CHAR1	
0x0		0x0		0x0		0x0	

Reset

LP_UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

LP_UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

LP_UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

LP_UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 25.50. LP_UART_SLEEP_CONF1_REG (0x0034)

(reserved)																												LP_UART_WK_CHARO					
31																												8	7	0			
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																												0x0				Reset	

LP_UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 25.51. LP_UART_SLEEP_CONF2_REG (0x0038)

(reserved)				LP_UART_WK_MODE_SEL				LP_UART_WK_CHAR_MASK				LP_UART_WK_CHAR_NUM				LP_UART_RX_WAKE_UP_THRHD				(reserved)				LP_UART_ACTIVE_THRESHOLD			
31	28	27	26	25	21	20	18	17	13	12	10	9	0														
0	0	0	0	0	0x0				0x5				1				0	0	0	0xf0				Reset			

LP_UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

LP_UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。
单位：字节。(R/W)

LP_UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

LP_UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。
0：不屏蔽
1：屏蔽
(R/W)

LP_UART_WK_MODE_SEL 配置选取哪个唤醒模式。
0：唤醒模式 0
1：唤醒模式 1
2：唤醒模式 2
3：唤醒模式 3
(R/W)

Register 25.52. LP_UART_SWFC_CONFO_SYNC_REG (0x003C)

(reserved)										LP_UART_SEND_XOFF LP_UART_SEND_XON LP_UART_FORCE_XOFF LP_UART_FORCE_XON LP_UART_XONOFF_DEL LP_UART_SW_FLOW_CON_EN LP_UART_XON_XOFF_STILL_SEND								LP_UART_XOFF_CHAR								LP_UART_XON_CHAR										
31									23	22	21	20	19	18	17	16	15									8	7									0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x13								0x11								Reset				

LP_UART_XON_CHAR 配置 XON 流控字符。(R/W)

LP_UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)

LP_UART_XON_XOFF_STILL_SEND 配置 LP UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。

0: 不能发送

1: 可以发送

(R/W)

LP UART SW FLOW CON EN 配置是否开启软件流控。

0: 关闭

1: 开启

(R/W)

LP_UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。

0: 不移除

1: 移除

(R/W)

LP_UART_FORCE_XON 配置是否让发送器继续发送数据。

0: 不发送

1: 继续发送

(R/W)

LP_UART_FORCE_XOFF 配置是否让发送器停止发送数据。

0: 不停止

1: 停止

(R/W)

LP_UART_SEND_XON 配置是否发送 XON 字符。

0: 不发送

1: 发送

(R/W/SS/SC)

LP_UART_SEND_XOFF 配置是否发送 XOFF 字符。

0: 不发送

1: 发送

(R/W/SS/SC)

Register 25.53. LP_UART_SWFC_CONF1_REG (0x0040)

(reserved)																LP_UART_XOFF_THRESHOLD				(reserved)				LP_UART_XON_THRESHOLD				(reserved)			
311615111087320																Reset															
0000000000000000																0xc				000				0x0				000			

LP_UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。
单位：字节。(R/W)

LP_UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。
单位：字节。(R/W)

Register 25.54. LP_UART_TXBRK_CONF_SYNC_REG (0x0044)

(reserved)																										LP_UART_TX_BRK_NUM									
31																										8	7	0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																										0xa								Reset	

LP_UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。
仅在 LP_UART_TXD_BRK 为 1 时有效。(R/W)

Register 25.55. LP_UART_IDLE_CONF_SYNC_REG (0x0048)

(reserved)												LP_UART_TX_IDLE_NUM								LP_UART_RX_IDLE_THRHD								
31												20	19						10	9								0
0 0 0 0 0 0 0 0 0 0 0 0												0x100								0x100								Reset

LP_UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。
单位：位时间（传输一位所需的时间）。(R/W)

LP_UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.56. LP_UART_DELAY_CONF_SYNC_REG (0x004C)

(reserved)																															LP_UART_DL1_EN LP_UART_DLO_EN (reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
31																															3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LP_UART_DLO_EN 配置是否在起始位之前增加一位停止位。

0: 不增加

1: 增加

(R/W)

LP_UART_DL1_EN 配置是否在停止位之后增加一位停止位。

0: 不增加

1: 增加

(R/W)

Register 25.57. LP_UART_CLK_CONF_REG (0x0088)

(reserved)				LP_UART_RX_RST_CORE LP_UART_TX_RST_CORE LP_UART_RX_SCLK_EN LP_UART_TX_SCLK_EN				(reserved)																								
31			28	27	26	25	24	23																								0
0	0	0	0	0	0	1	1	0																							Reset	

LP_UART_TX_SCLK_EN 配置是否开启 LP UART TX 时钟。

0: 关闭

1: 开启

(R/W)

LP_UART_RX_SCLK_EN 配置是否开启 LP UART RX 时钟。

0: 关闭

1: 开启

(R/W)

LP_UART_TX_RST_CORE 向此位先写 1 后写 0，复位 LP UART TX。(R/W)

LP_UART_RX_RST_CORE 向此位先写 1 后写 0，复位 LP UART RX。(R/W)

Register 25.58. LP_UART_STATUS_REG (0x001C)

LP_UART_TXD				LP_UART_TXFIFO_CNT				LP_UART_RXD				LP_UART_RXFIFO_CNT					
LP_UART_RTSN				(reserved)				LP_UART_CTSN				(reserved)					
LP_UART_DTRN				(reserved)				LP_UART_DSRN				(reserved)					
31	30	29	28	24	23	19	18	16	15	14	13	12	8	7	3	2	0
1	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Reset

LP_UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

LP_UART_DSRN 表示内部 LP UART DSR 信号的电平值。(RO)

LP_UART_CTSN 表示内部 LP UART CTS 信号的电平值。(RO)

LP_UART_RXD 表示内部 LP UART RXD 信号的电平值。(RO)

LP_UART_TXFIFO_CNT 表示 TX FIFO 中的数据量。

单位：字节。(RO)

LP_UART_DTRN 表示内部 LP UART DTR 信号的电平。(RO)

LP_UART_RTSN 表示内部 LP UART RTS 信号的电平。(RO)

LP_UART_TXD 表示内部 LP UART TXD 信号的电平。(RO)

Register 25.59. LP_UART_MEM_TX_STATUS_REG (0x0068)

(reserved)																LP_UART_TX_SRAM_RADDR				(reserved)				LP_UART_TX_SRAM_WADDR				(reserved)				
311716121187320																																
0000000000000000																0x0				0000				0x0				0000				Reset

LP_UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)

LP_UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

Register 25.60. LP_UART_MEM_RX_STATUS_REG (0x006C)

(reserved)																LP_UART_RX_SRAM_WADDR				(reserved)				LP_UART_RX_SRAM_RADDR				(reserved)				
311716121187320																																
0000000000000000																0x10				0000				0x10				000				Reset

LP_UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)

LP_UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

Register 25.61. LP_UART_FSM_STATUS_REG (0x0070)

(reserved)																								LP_UART_ST_UTX_OUT		LP_UART_ST_URX_OUT		
31																								8	7	4	3	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0		0		Reset

LP_UART_ST_URX_OUT 表示 LP UART 接收器的状态。(RO)

LP_UART_ST_UTX_OUT 表示 LP UART 发送器的状态。(RO)

Register 25.62. LP_UART_AFIFO_STATUS_REG (0x0090)

(reserved)																												LP_UART_RX_AFIFO_EMPTY LP_UART_RX_AFIFO_FULL LP_UART_TX_AFIFO_EMPTY LP_UART_TX_AFIFO_FULL				
31																												4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	Reset	

LP_UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0: 未
1: 已
(RO)

LP_UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0: 不为
1: 为
(RO)

LP_UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0: 未
1: 已
(RO)

LP_UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0: 不为
1: 为
(RO)

Register 25.63. LP_UART_AT_CMD_PRECNT_SYNC_REG (0x0050)

(reserved)																LP_UART_PRE_IDLE_NUM																															
31																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x901																Reset															

LP_UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位: 位时间 (传输一位所需的时间)。(R/W)

Register 25.64. LP_UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)

(reserved)																LP_UART_POST_IDLE_NUM																
31																16	15															0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x901															Reset	

LP_UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.65. LP_UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)

(reserved)																LP_UART_RX_GAP_TOUT																
3116																150																
0000000000000000																11																Reset

LP_UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。
单位：位时间（传输一位所需的时间）。(R/W)

Register 25.66. LP_UART_AT_CMD_CHAR_SYNC_REG (0x005C)

(reserved)																LP_UART_CHAR_NUM								LP_UART_AT_CMD_CHAR																																							
31																16																15								8								7								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x3								0x2b								Reset																															

LP_UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

LP_UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 25.67. LP_UART_DATE_REG (0x008C)

LP_UART_DATE																															
31																															0
0x2305050																															
Reset																															

LP_UART_DATE 版本控制寄存器。(R/W)

Register 25.68. LP_UART_REG_UPDATE_REG (0x0098)

(reserved)																															LP_UART_REG_UPDATE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																															1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

LP_UART_REG_UPDATE 配置是否同步寄存器

0: 不同步

1: 同步

(R/W/SC)

Register 25.69. LP_UART_ID_REG (0x009C)

LP_UART_ID																															
31																															0
0x000500																															
Reset																															

LP_UART_ID 配置 LP UART ID。(R/W)

25.8.3 UHCI 寄存器

本小节的所有地址均为相对于 UHCI 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 25.70. UHCI_CONFO_REG (0x0000)

[illegible]

UHCI_TX_RST 先写1再写0复位解码器状态机。(R/W)

UHCI_RX_RST 先写1再写0复位编码器状态机。(R/W)

UHCI_UART_SEL 从 UART 0 ~ 1 中选择其一连接到 UHCI。

0: 选择 UART0

1: 选择 UART1

2 ~ 7: 无效值, 没有 UART 连接到 UHCI

(R/W)

UHCI_SEPER_EN 配置是否使用特殊字符分隔数据帧。

0: 不分隔

1: 分隔

(R/W)

UHCI_HEAD_EN 配置是否用格式报头编码数据包。

0: 不使用报头

1: 使用报头

(R/W)

UHCI_CRC_REC_EN 配置 UHCI 是否开启接收 16 位 CRC。

0: 关闭

1: 开启

(R/W)

见下页...

Register 25.70. UHCI_CONFO_REG (0x0000)

[接上页...](#)

UHCI_UART_IDLE_EOF_EN 配置 UART 空闲时 UHCI 是否停止接收数据。

0: 不停止

1: 停止

(R/W)

UHCI_LEN_EOF_EN 配置 UHCI 解码器停止接收数据的条件。

0: 接收到 0xCO 后停止

1: 接收字节数达到指定值时停止。UHCI_HEAD_EN 为 1 时, 指定值是 UCHI 数据包报头指定的数据长度。UHCI_HEAD_EN 为 0 时, 指定值为配置值。

(R/W)

UHCI_ENCODE_CRC_EN 配置是否开启数据完整性检测, 在数据末尾加 16 位 CCITT-CRC。

0: 关闭

1: 开启

(R/W)

UHCI_CLK_EN 配置时钟门控。

0: 仅在应用写寄存器时开启时钟

1: 一直强制为寄存器开启时钟

(R/W)

UHCI_UART_RX_BRK_EOF_EN 配置 UART 收到 NULL 帧后, UHCI 是否停止接收数据。

0: 不停止

1: 停止

(R/W)

Register 25.71. UHCI_CONF1_REG (0x0014)

[illegible]

UHCI_CHECK_SUM_EN 配置 UHCI 接收数据包时，是否开启报头校验和检查。

0: 关闭

1: 开启

(R/W)

UHCI_CHECK_SEQ_EN 配置 UHCI 接收数据包时，是否开启序列号检查。

0: 关闭

1: 开启

(R/W)

UHCI_CRC_DISABLE 配置是否开启 CRC 计算。

0: 关闭

1: 开启

仅在 UHCI 包中的数据完整性检测位为 1 时有效。

(R/W)

UHCI_SAVE_HEAD 配置 UHCI 接收数据包时，是否保存数据包报头。

0: 不保存

1: 保存

(R/W)

UHCI_TX_CHECK_SUM_RE 配置用校验和编码数据包。

0: 不使用

1: 使用

(R/W)

UHCI_TX_ACK_NUM_RE 配置准备发送可靠数据包时，是否用 ACK 编码该数据包。

0: 不使用

1: 使用

(R/W)

UHCI_WAIT_SW_START 配置是否将 UHCI 编码器的状态机调至 ST_SW_WAIT 状态。

0: 沓

1: 是

(R/W)

UHCI_SW_START 编码器的状态机为 ST_SW_WAIT 状态时，配置 UHCI 是否开始发送数据包。

0: 不发送

1: 开始发送

(R/W/SC)

Register 25.72. UHCI_ESCAPE_CONF_REG (0x0020)

[illegible]

UHCI_TX_CO_ESC_EN 配置 DMA 接收数据时，是否开启 0xC0 字符解码。

0: 关闭

1: 开启

(R/W)

UHCI TX DB ESC EN 配置 DMA 接收数据时，是否开启 0xDB 字符解码。

0: 关闭

1: 开启

(R/W)

UHCI_TX_11_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x11 解码。

0: 关闭

1: 开启

(R/W)

UHCI_TX_13_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x13 解码。

0: 关闭

1: 开启

(R/W)

UHCI_RX_CO_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xC0。

0: 不替换

1: 替换

(R/W)

UHCI RX DB ESC EN 配置 DMA 接收数据时，是否用特殊字符替换 0xDB。

0: 不替换

1: 替换

(R/W)

UHCI_RX_11_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x11。

0: 不替换

1: 替换

(R/W)

UHCI RX 13 ESC EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x13。

0: 不替换

1: 替换

(R/W)

Register 25.73. UHCI_HUNG_CONF_REG (0x0024)

(reserved)								UHCI_RXFIFO_TIMEOUT_ENA				UHCI_RXFIFO_TIMEOUT_SHIFT				UHCI_RXFIFO_TIMEOUT				UHCI_TXFIFO_TIMEOUT_ENA				UHCI_TXFIFO_TIMEOUT_SHIFT				UHCI_TXFIFO_TIMEOUT			
31								24	23	22	20	19					12	11	10	8	7								0		
0	0	0	0	0	0	0	0	0	1	0	0x10				1	0	0x10											Reset			

UHCI_TXFIFO_TIMEOUT 配置 DMA 接收数据的超时值。
单位：毫秒。(R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT 配置 TX FIFO 超时计数器的阈值上限。(R/W)

UHCI_TXFIFO_TIMEOUT_ENA 配置是否开启 TX FIFO 接收数据超时。
0：关闭
1：开启
(R/W)

UHCI_RXFIFO_TIMEOUT 配置 DMA 从 RAM 读取数据的超时值。单位：毫秒。(R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT 配置 RX FIFO 超时计数器的阈值上限。(R/W)

UHCI_RXFIFO_TIMEOUT_ENA 配置是否开启 DMA 发送数据超时。
0：关闭
1：开启
(R/W)

Register 25.74. UHCI_ACK_NUM_REG (0x0028)

(reserved)																												UHCI_ACK_NUM_LOAD		UHCI_ACK_NUM		
31																												4	3	2	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																												0	0x0		Reset	

UHCI_ACK_NUM 配置软件流控中使用的 ACK 数量。(R/W)

UHCI_ACK_NUM_LOAD 配置是否加载 ACK。
0：不加载
1：加载
(WT)

Register 25.75. UHCI_QUICK_SENT_REG (0x0030)

(reserved)																												UHCI_ALWAYS_SEND_EN				UHCI_ALWAYS_SEND_NUM				UHCI_SINGLE_SEND_EN				UHCI_SINGLE_SEND_NUM																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
31																												8	7	6	4		3	2	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

UHCI_SINGLE_SEND_NUM 配置 single_send 模式下发送哪个寄存器中的数据。

- 0: Q0 寄存器
 - 1: Q1 寄存器
 - 2: Q2 寄存器
 - 3: Q3 寄存器
 - 4: Q4 寄存器
 - 5: Q5 寄存器
 - 6: Q6 寄存器
 - 7: 无效值，没有作用
- (R/W)

UHCI_SINGLE_SEND_EN 配置是否开启 single_send 模式。

- 0: 关闭
 - 1: 开启
- (R/W/SC)

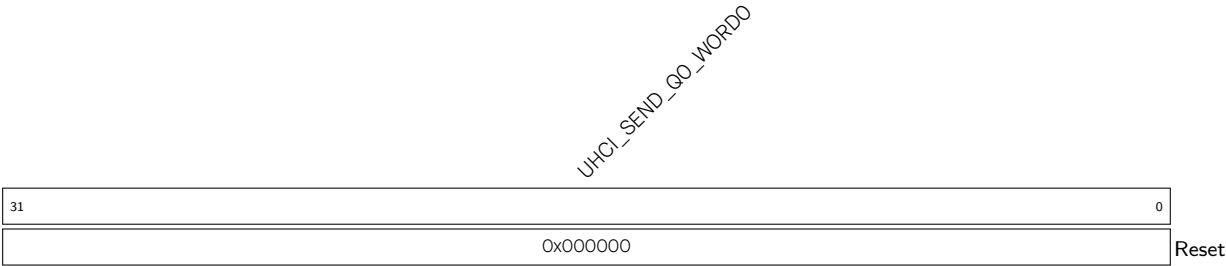
UHCI_ALWAYS_SEND_NUM 配置 always_send 模式下发送哪个寄存器中的数据。

- 0: Q0 寄存器
 - 1: Q1 寄存器
 - 2: Q2 寄存器
 - 3: Q3 寄存器
 - 4: Q4 寄存器
 - 5: Q5 寄存器
 - 6: Q6 寄存器
 - 7: 无效值，没有作用
- (R/W)

UHCI_ALWAYS_SEND_EN 配置是否开启 always_send 模式。

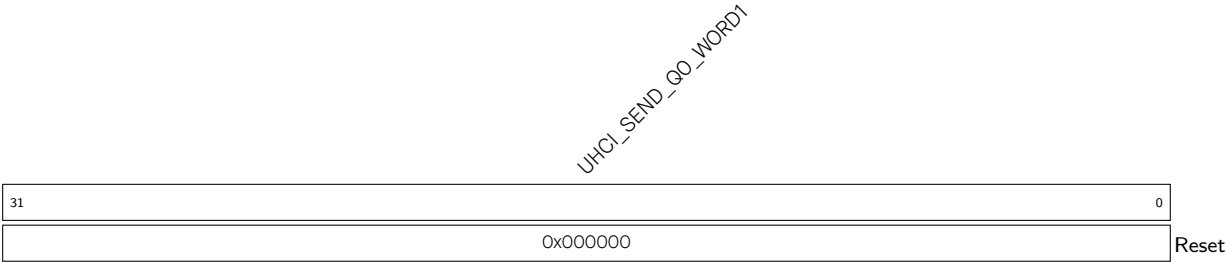
- 0: 关闭
 - 1: 开启
- (R/W)

Register 25.76. UHCI_REG_Q0_WORD0_REG (0x0034)



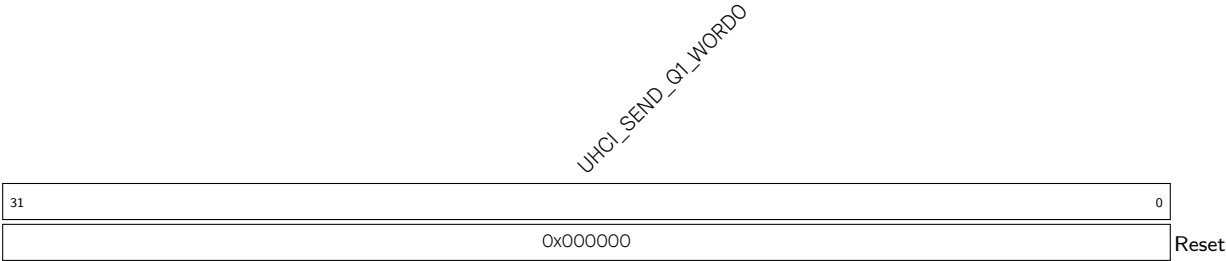
UHCI_SEND_Q0_WORD0 Q0 寄存器待发送数据。(R/W)

Register 25.77. UHCI_REG_Q0_WORD1_REG (0x0038)



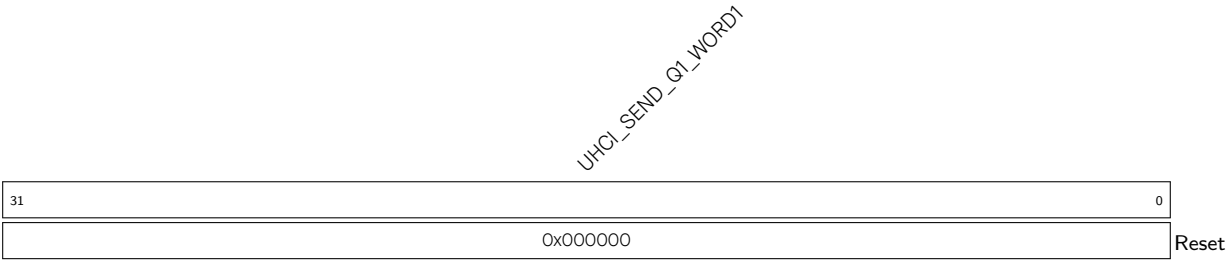
UHCI_SEND_Q0_WORD1 Q0 寄存器待发送数据。(R/W)

Register 25.78. UHCI_REG_Q1_WORD0_REG (0x003C)



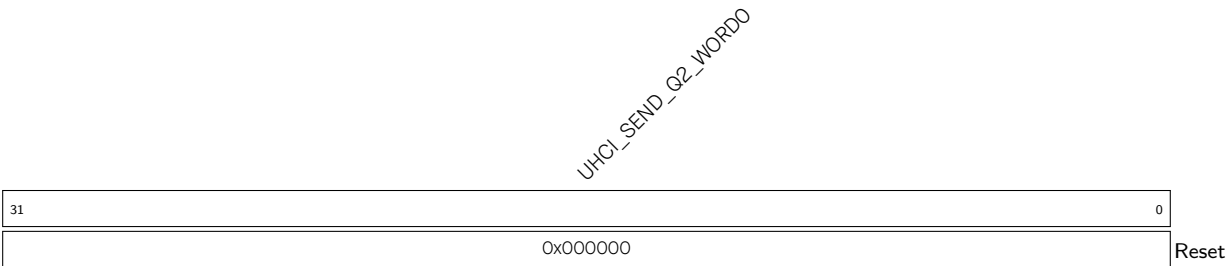
UHCI_SEND_Q1_WORD0 Q1 寄存器待发送数据。(R/W)

Register 25.79. UHCI_REG_Q1_WORD1_REG (0x0040)



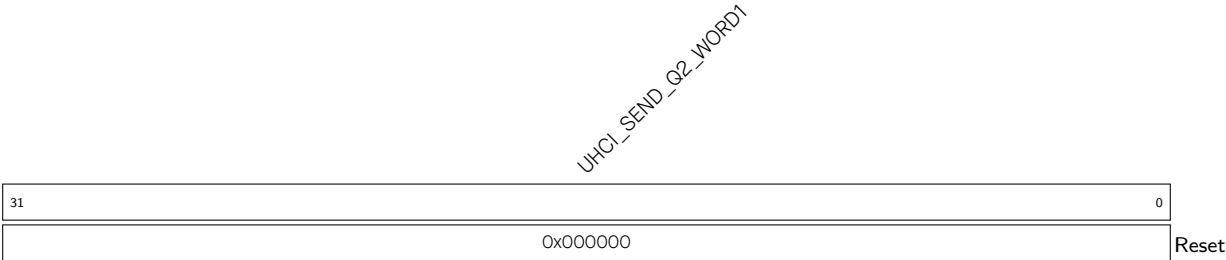
UHCI_SEND_Q1_WORD1 Q1 寄存器待发送数据。(R/W)

Register 25.80. UHCI_REG_Q2_WORD0_REG (0x0044)



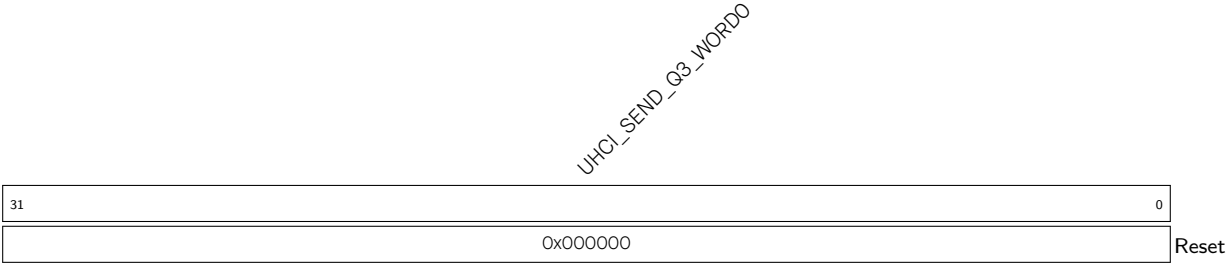
UHCI_SEND_Q2_WORD0 Q2 寄存器待发送数据。(R/W)

Register 25.81. UHCI_REG_Q2_WORD1_REG (0x0048)



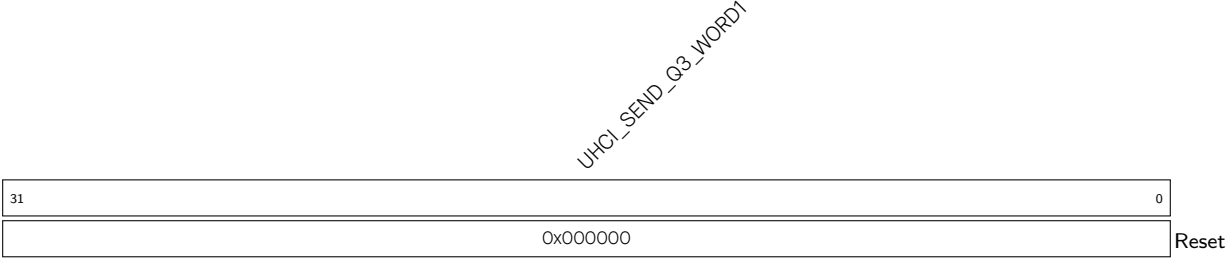
UHCI_SEND_Q2_WORD1 Q2 寄存器待发送数据。(R/W)

Register 25.82. UHCI_REG_Q3_WORD0_REG (0x004C)



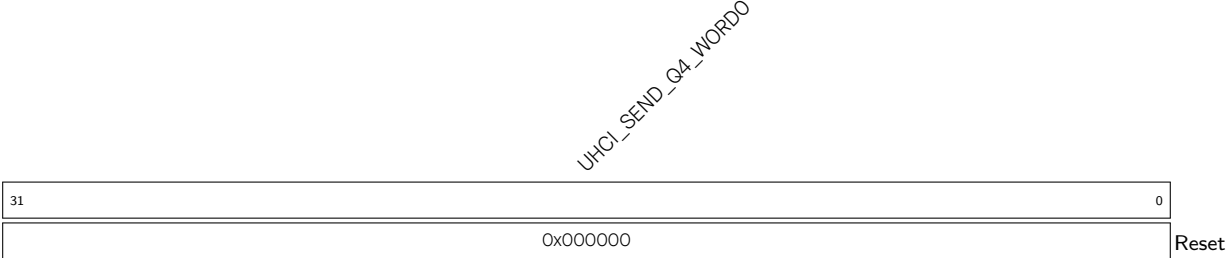
UHCI_SEND_Q3_WORD0 Q3 寄存器待发送数据。(R/W)

Register 25.83. UHCI_REG_Q3_WORD1_REG (0x0050)



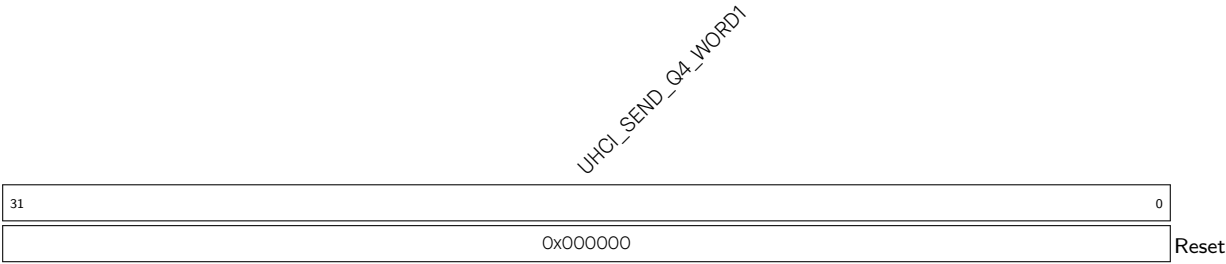
UHCI_SEND_Q3_WORD1 Q3 寄存器待发送数据。(R/W)

Register 25.84. UHCI_REG_Q4_WORD0_REG (0x0054)



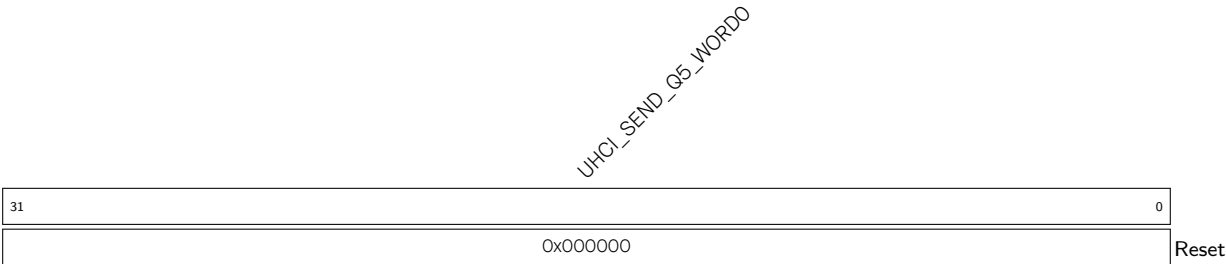
UHCI_SEND_Q4_WORD0 Q4 寄存器待发送数据。(R/W)

Register 25.85. UHCI_REG_Q4_WORD1_REG (0x0058)



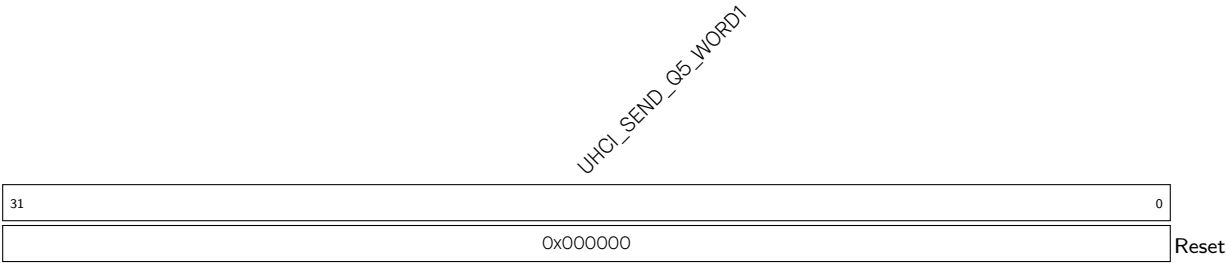
UHCI_SEND_Q4_WORD1 Q4 寄存器待发送数据。(R/W)

Register 25.86. UHCI_REG_Q5_WORD0_REG (0x005C)



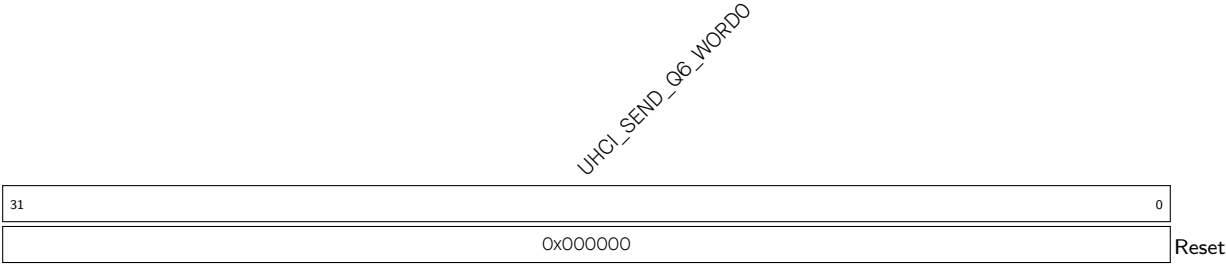
UHCI_SEND_Q5_WORD0 Q5 寄存器待发送数据。(R/W)

Register 25.87. UHCI_REG_Q5_WORD1_REG (0x0060)



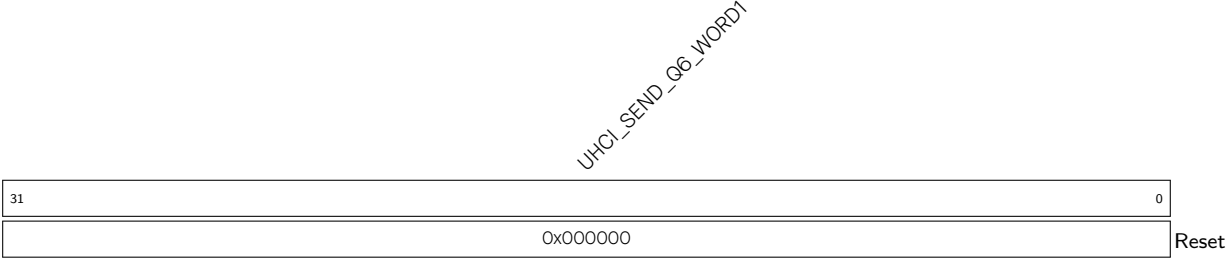
UHCI_SEND_Q5_WORD1 Q5 寄存器待发送数据。(R/W)

Register 25.88. UHCI_REG_Q6_WORD0_REG (0x0064)



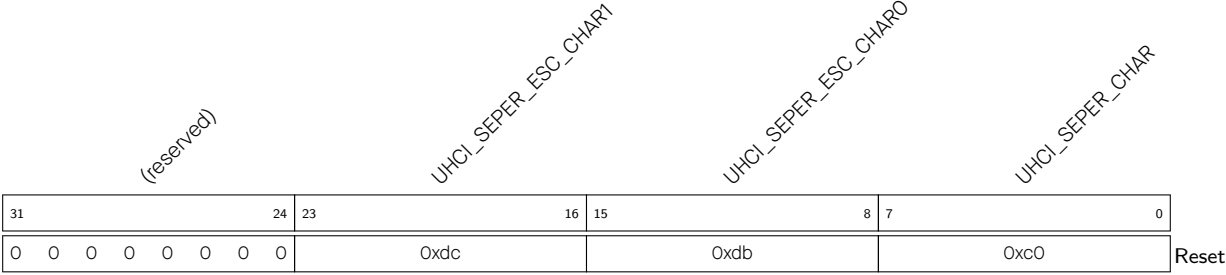
UHCI_SEND_Q6_WORD0 Q6 寄存器待发送数据。(R/W)

Register 25.89. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Q6 寄存器待发送数据。(R/W)

Register 25.90. UHCI_ESC_CONFO_REG (0x006C)



- UHCI_SEPER_CHAR 配置编码数据包的分隔符，默认为 0xc0。(R/W)
- UHCI_SEPER_ESC_CHAR0 配置 SLIP 转义序列的第一个字符，默认为 0xdb。(R/W)
- UHCI_SEPER_ESC_CHAR1 配置 SLIP 转义序列的第二个字符，默认为 0xdc。(R/W)

Register 25.91. UHCI_ESC_CONF1_REG (0x0070)

(reserved)								UHCI_ESC_SEQ0_CHAR1								UHCI_ESC_SEQ0_CHAR0								UHCI_ESC_SEQ0										
31								24								16	15								8	7								0
0	0	0	0	0	0	0	0	0xdd								0xdb								0xdb								Reset		

UHCI_ESC_SEQ0 配置需编码的字符，默认为用作 SLIP 转义序列第一个字符的 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ0 默认为 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ0 默认为 0xDD。(R/W)

Register 25.92. UHCI_ESC_CONF2_REG (0x0074)

(reserved)								UHCI_ESC_SEQ1_CHAR1								UHCI_ESC_SEQ1_CHAR0								UHCI_ESC_SEQ1											
31								24	23								16	15								8	7								0
0	0	0	0	0	0	0	0	0xde								0xdb								0x11								Reset			

UHCI_ESC_SEQ1 配置需编码的字符，默认为流控字符的 0x11。(R/W)

UHCI_ESC_SEQ1_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ1 默认为 0xDB。(R/W)

UHCI_ESC_SEQ1_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ1 默认为 0xDE。(R/W)

Register 25.93. UHCI_ESC_CONF3_REG (0x0078)

(reserved)								UHCI_ESC_SEQ2_CHAR1								UHCI_ESC_SEQ2_CHAR0								UHCI_ESC_SEQ2								
31								24							16							8							0			
0	0	0	0	0	0	0	0	0xdf								0xdb								0x13								Reset

- UHCI_ESC_SEQ2** 配置需编码的字符，默认为流控字符的 0x13。(R/W)
- UHCI_ESC_SEQ2_CHAR0** 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ2 默认为 0xDB。(R/W)
- UHCI_ESC_SEQ2_CHAR1** 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ2 默认为 0xDF。(R/W)

Register 25.94. UHCI_PKT_THRES_REG (0x007C)

(reserved)																UHCL_PKT_THRS																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
31													13	12																	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- UHCI_PKT_THRS** 配置数据包的最大长度。
单位：字节。仅在 UHCI_HEAD_EN 为 0 时有效。(R/W)

Register 25.95. UHCI_INT_RAW_REG (0x0004)

(reserved)																								UHCI_APP_CTRL1_INT_RAW UHCI_APP_CTRL0_INT_RAW UHCI_OUT_EOF_INT_RAW UHCI_SEND_A_REG_Q_INT_RAW UHCI_TX_HUNG_Q_INT_RAW UHCI_RX_HUNG_INT_RAW UHCI_TX_START_INT_RAW UHCI_RX_START_INT_RAW																		
31																								9		8	7	6	5	4	3	2	1	0								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset													

- UHCI_RX_START_INT_RAW UHCI_RX_START_INT 的原始中断状态。(R/WTC/SS)
- UHCI_TX_START_INT_RAW UHCI_TX_START_INT 的原始中断状态。(R/WTC/SS)
- UHCI_RX_HUNG_INT_RAW UHCI_RX_HUNG_INT 的原始中断状态。(R/WTC/SS)
- UHCI_TX_HUNG_INT_RAW UHCI_TX_HUNG_INT 的原始中断状态。(R/WTC/SS)
- UHCI_SEND_S_REG_Q_INT_RAW UHCI_SEND_S_REG_Q_INT 的原始中断状态。(R/WTC/SS)
- UHCI_SEND_A_REG_Q_INT_RAW UHCI_SEND_A_REG_Q_INT 的原始中断状态。(R/WTC/SS)
- UHCI_OUT_EOF_INT_RAW UHCI_OUT_EOF_INT 的原始中断状态。(R/WTC/SS)
- UHCI_APP_CTRL0_INT_RAW UHCI_APP_CTRL0_INT 的原始中断状态。(R/W)
- UHCI_APP_CTRL1_INT_RAW UHCI_APP_CTRL1_INT 的原始中断状态。(R/W)

Register 25.96. UHCI_INT_ST_REG (0x0008)

(reserved)																								UHCI_APP_CTRL1_INT_ST UHCI_APP_CTRL0_INT_ST UHCI_OUTLINK_EOF_ERR_INT_ST UHCI_SEND_A_REG_Q_INT_ST UHCI_SEND_S_REG_Q_INT_ST UHCI_TX_HUNG_INT_ST UHCI_RX_HUNG_INT_ST UHCI_TX_START_INT_ST UHCI_RX_START_INT_ST												
31																								9		8	7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset								

- UHCI_RX_START_INT_ST UHCI_RX_START_INT 的屏蔽中断状态。(RO)
- UHCI_TX_START_INT_ST UHCI_TX_START_INT 的屏蔽中断状态。(RO)
- UHCI_RX_HUNG_INT_ST UHCI_RX_HUNG_INT 的屏蔽中断状态。(RO)
- UHCI_TX_HUNG_INT_ST UHCI_TX_HUNG_INT 的屏蔽中断状态。(RO)
- UHCI_SEND_S_REG_Q_INT_ST UHCI_SEND_S_REG_Q_INT 的屏蔽中断状态。(RO)
- UHCI_SEND_A_REG_Q_INT_ST UHCI_SEND_A_REG_Q_INT 的屏蔽中断状态。(RO)
- UHCI_OUTLINK_EOF_ERR_INT_ST UHCI_OUTLINK_EOF_ERR_INT 的屏蔽中断状态。(RO)
- UHCI_APP_CTRL0_INT_ST UHCI_APP_CTRL0_INT 的屏蔽中断状态。(RO)
- UHCI_APP_CTRL1_INT_ST UHCI_APP_CTRL1_INT 的屏蔽中断状态。(RO)

Register 25.97. UHCI_INT_ENA_REG (0x000C)

(reserved)																								UHCI_APP_CTRL1_INT_ENA UHCI_APP_CTRL0_INT_ENA UHCI_OUTLINK_EOF_ERR_INT_ENA UHCI_SEND_A_REG_Q_INT_ENA UHCI_SEND_S_REG_Q_INT_ENA UHCI_TX_HUNG_INT_ENA UHCI_RX_HUNG_INT_ENA UHCI_RX_START_INT_ENA														
31																								9		8	7	6	5	4	3	2	1	0				
0																								0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

UHCI_RX_START_INT_ENA 写 1 使能 UHCI_RX_START_INT。(R/W)

UHCI_TX_START_INT_ENA 写 1 使能 UHCI_TX_START_INT。(R/W)

UHCI_RX_HUNG_INT_ENA 写 1 使能 UHCI_RX_HUNG_INT。(R/W)

UHCI_TX_HUNG_INT_ENA 写 1 使能 UHCI_TX_HUNG_INT。(R/W)

UHCI_SEND_S_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_S_REG_Q_INT。(R/W)

UHCI_SEND_A_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_A_REG_Q_INT。(R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA 写 1 使能 UHCI_OUTLINK_EOF_ERR_INT。(R/W)

UHCI_APP_CTRL0_INT_ENA 写 1 使能 UHCI_APP_CTRL0_INT。(R/W)

UHCI_APP_CTRL1_INT_ENA 写 1 使能 UHCI_APP_CTRL1_INT。(R/W)

Register 25.98. UHCI_INT_CLR_REG (0x0010)

(reserved)																				UHCL_APP_CTRL1_INT_CLR								UHCL_APP_CTRL0_INT_CLR								UHCL_OUTLINK_EOF_ERR_INT_CLR								UHCL_SEND_A_REG_Q_INT_CLR								UHCL_SEND_S_REG_Q_INT_CLR								UHCL_TX_HUNG_INT_CLR								UHCL_RX_HUNG_INT_CLR								UHCL_TX_START_INT_CLR								UHCL_RX_START_INT_CLR																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31																				9		8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		

UHCI_RX_START_INT_CLR 写1清除 UHCI_RX_START_INT。(WT)

UHCI_TX_START_INT_CLR 写1清除 UHCI_TX_START_INT。(WT)

UHCI_RX_HUNG_INT_CLR 写1清除 UHCI_RX_HUNG_INT。(WT)

UHCI_TX_HUNG_INT_CLR 写1清除 UHCI_TX_HUNG_INT。(WT)

UHCI_SEND_S_REG_Q_INT_CLR 写1清除 UHCI_SEND_S_REG_Q_INT。(WT)

UHCI_SEND_A_REG_Q_INT_CLR 写1清除 UHCI_SEND_A_REG_Q_INT。(WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR 写1清除 UHCI_OUTLINK_EOF_ERR_INT。(WT)

UHCI_APP_CTRL0_INT_CLR 写1清除 UHCI_APP_CTRL0_INT。(WT)

UHCI_APP_CTRL1_INT_CLR 写 1 清除 UHCI_APP_CTRL1_INT。(WT)

Register 25.99. UHCI_STATE0_REG (0x0018)

(reserved)																UHCI_DECODE_STATE		UHCI_RX_ERR_CAUSE		
31																6	5	3	2	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0		0		Reset

UHCI_RX_ERR_CAUSE 表示 DMA 接收数据错误的原因。

- 0: 无效，没有作用
- 1: HCI 包校验和错误
- 2: HCI 包序列号错误
- 3: HCI 包 CRC 位错误
- 4: 找到 0xC0 但接收的 HCI 包不完整
- 5: 未找到 0xC0 但接收的 HCI 包完整
- 6: CRC 检测错误
- 7: 无效，没有作用

(RO)

UHCI_DECODE_STATE 表示 UHCI 解码器状态。(RO)

Register 25.100. UHCI_STATE1_REG (0x001C)

(reserved)																												UHCI_ENCODE_STATE		
31																												3	2	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																												0		Reset

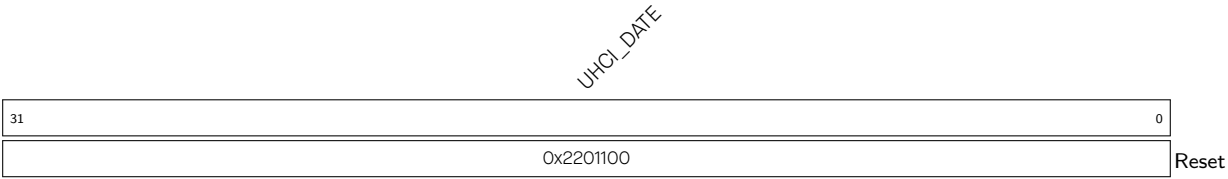
UHCI_ENCODE_STATE 表示 UHCI 编码器状态。(RO)

Register 25.101. UHCI_RX_HEAD_REG (0x002C)

UHCI_RX_HEAD																														
0																														
0x000000																														
Reset																														

UHCI_RX_HEAD 表示当前接收数据包的报头。(RO)

Register 25.102. UHCI_DATE_REG (0x0080)



UHCI_DATE 版本控制寄存器。(R/W)

第 26 章

I2C 控制器 (I2C)

I2C (Inter-Integrated Circuit) 总线用于使 ESP32-C5 和多个外部设备进行通信。多个外部设备可以共用一个 I2C 总线。

ESP32-C5 在主系统有一个 I2C 控制器，在低功耗系统有一个 I2C 控制器。其中，主系统中的 I2C 控制器既可作为主机又可作为从机（下文以 I2C 指代），低功耗系统中的 I2C 控制器则只可作为主机使用，在主系统休眠时仍能工作（下文以 LP_I2C 指代）。

26.1 概述

I2C 是一个两线总线，由 SDA 线和 SCL 线构成。这些线设置为漏极开漏 (open-drain) 输出。因此，I2C 总线上可以挂载多个外设，通常是和一个或多个主机以及一个或多个从机。但同一时刻只有一个主机能占用总线访问一个从机。

主机发出开始信号，即主机在 SCL 为高电平时拉低 SDA 线，代表通讯开始。随后主机通过 SCL 线发出 9 个时钟脉冲，前 8 个脉冲用于传输 7 位地址和 1 个读 / 写位。如果从机地址与该 7 位地址一致，那么从机可以通过在第 9 个脉冲拉低 SDA 线来应答。接下来，根据读 / 写标志位，主机和从机之间可以传输更多的数据。根据应答位 (ACK) 的逻辑电平决定是否停止发送数据。在数据传输中，SDA 线仅在 SCL 线为低电平时才发生变化。当主机完成通讯，发送一个停止信号：主机在 SCL 为高电平时，拉高 SDA 线。如果一次通信中主机既有写操作又有读操作，则主机需在读写操作变化前，发送一个重新开始信号、从机地址和读 / 写标志位。重新开始信号不仅用于一次通信中切换方向，也用于切换设备模式（主机或从机模式）。

26.2 主要特性

ESP32-C5 I2C 控制器具有以下几个特点：

- 支持主机模式和从机模式
- 支持多主机和从机通信
- 支持标准模式 (100 Kbit/s)
- 支持快速模式 (400 Kbit/s)
- 支持 7 位以及 10 位地址寻址
- 从机模式下支持拉低 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能
- 支持从机地址和从机内存或寄存器地址的双寻址模式

26.3 结构概览

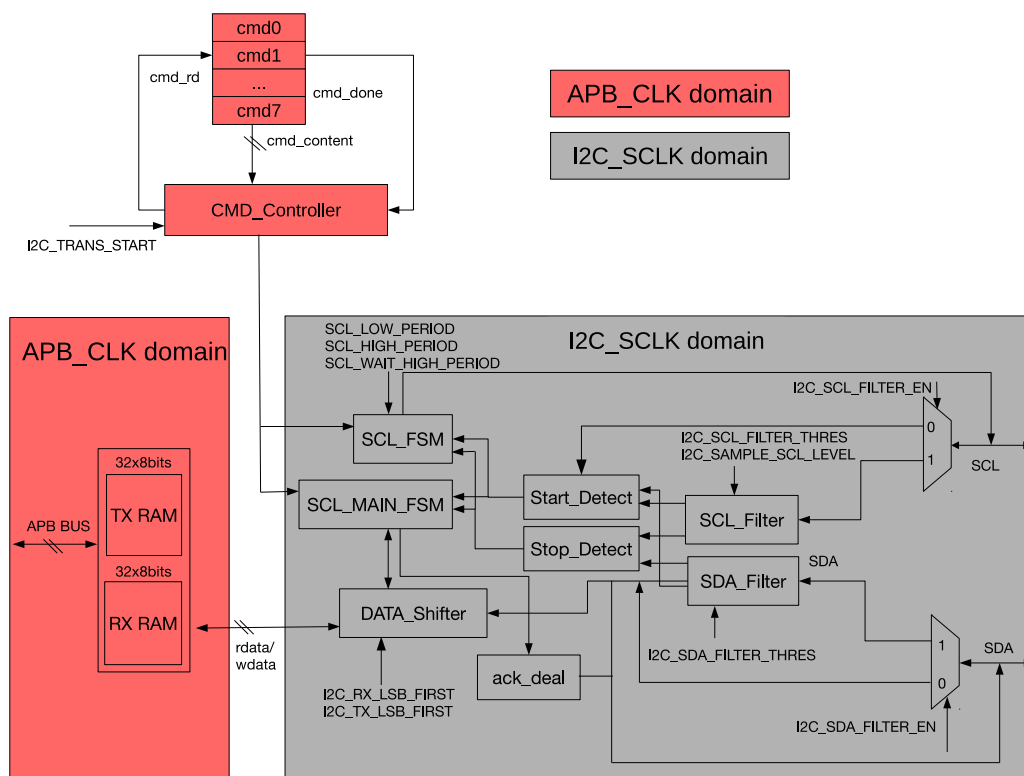


图 26.3-1. I2C 主机基本架构

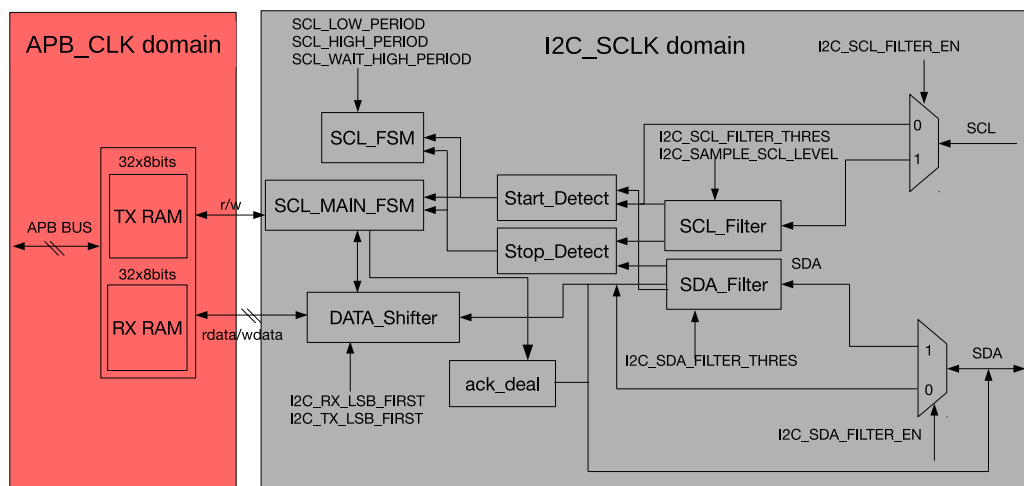


图 26.3-2. I2C 从机基本架构

I2C 控制器可以工作于主机模式或者从机模式，`I2C_MS_MODE` 寄存器用于模式选择。图 26.3-1 为 I2C 主机基本架构图，图 26.3-2 为 I2C 从机基本架构图。I2C 控制器内部包括的模块主要有：

- 接收和发送存储器 TX/RX RAM：分别用来存储 I2C 要发送和接收到的数据
- 命令控制器 CMD_Controller：产生 RSTART、STOP、WRITE、READ 和 END 指令
- SCL 时钟控制器 SCL_FSM：用来产生满足 I2C 协议的 SCL 时钟。图 26.3-3 和表 26.3-1 是 I2C 协议的时序图和对应的参数表。

- SDA 数据控制器 SCL_MAIN_FSM：用来控制 I2C 指令的执行，和 SDA 线的数据序列，还控制 ack_deal 模块生成 ACK 位并检测 SDA 线上 ACK 位的电平
- 串并转换器 DATA_Shifter：用来完成串行数据和并行数据之间的转换
- SCL 滤波器 SCL_Filter：用于消除 SCL 输入信号上的噪声
- SDA 滤波器 SDA_Filter：用于消除 SDA 输入信号上的噪声
- ACK 位控制器 ack_deal：在 SCL_MAIN_FSM 控制下生成 ACK 位并检测 SDA 线上 ACK 位的电平。

另外，还有产生 I2C 内部时钟的时钟模块，以及在 APB 总线和 I2C 模块之间同步的同步模块。

时钟模块的作用是进行时钟源选择、时钟开关和时钟分频。同步模块用来同步不同时钟域之间信号的传输。

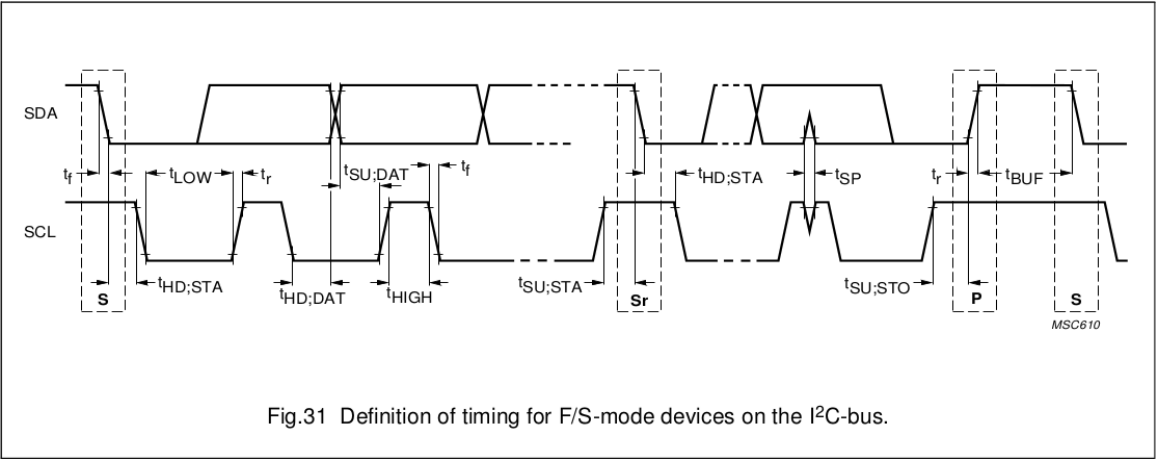


图 26.3-3. I2C 协议时序（引自 [The I2C-bus specification](#) Version 2.1 Fig.31）

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f _{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	t _{HD;STA}	4.0	—	0.6	—	μs
LOW period of the SCL clock	t _{LOW}	4.7	—	1.3	—	μs
HIGH period of the SCL clock	t _{HIGH}	4.0	—	0.6	—	μs
Set-up time for a repeated START condition	t _{SU;STA}	4.7	—	0.6	—	μs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I ² C-bus devices	t _{HD;DAT}	5.0 0 ⁽²⁾	— 3.45 ⁽³⁾	— 0 ⁽²⁾	— 0.9 ⁽³⁾	μs μs
Data set-up time	t _{SU;DAT}	250	—	100 ⁽⁴⁾	—	ns
Rise time of both SDA and SCL signals	t _r	—	1000	20 + 0.1C _b ⁽⁵⁾	300	ns
Fall time of both SDA and SCL signals	t _f	—	300	20 + 0.1C _b ⁽⁵⁾	300	ns
Set-up time for STOP condition	t _{SU;STO}	4.0	—	0.6	—	μs
Bus free time between a STOP and START condition	t _{BUF}	4.7	—	1.3	—	μs

表 26.3-1. I2C 时序参数（引自 [The I2C-bus specification](#) Version 2.1 Table 5）

26.4 功能描述

正如前文介绍，I2C 总线上可以挂载一个或多个主机以及一个或多个从机。下文中介绍的是 ESP32-C5 I2C 控制器的操作方法，请注意 I2C 总线上其他主机或者从机的操作方法可能与 ESP32-C5 I2C 操作方法不同，具体请参考各个 I2C 控制器的技术规格书。

26.4.1 时钟配置

寄存器配置和 TX/RX RAM 部分的时钟域为 APB_CLK。I2C 主要逻辑部分，包括 SCL_FSM、SCL_MAIN_FSM、SCL_FILTER、SDA_FILTER 和 DATA_SHIFTER 都为 I2C_SCLK 时钟域。

用户可以通过 `PCR_I2C_SCLK_SEL`，将 HP 系统中 I2C 的时钟源 I2C_SCLK 配置为 XTAL_CLK 或 RC_FAST_CLK。对于 LP 系统中 LP_I2C 的时钟源 I2C_SCLK，则可以通过 `LP_CLKRST_LP_I2C_CLK_SEL`，将其配置为 CLK_XTALD2 或 CLK_ROOT_FAST。

配置 I2C 时钟源的步骤如下：

- `PCR_I2C_SCLK_EN` 为 1 时，启用 I2C I2C_SCLK 时钟源。
- `PCR_I2C_SCLK_SEL` 为 0 时，配置时钟源为 XTAL_CLK。
- `PCR_I2C_SCLK_SEL` 为 1 时，配置时钟源为 RC_FAST_CLK。

配置 LP_I2C 时钟源的步骤如下：

- `LPPERI_LP_EXT_I2C_CK_EN` 为 1 时，启用 LP_I2C I2C_SCLK 时钟源。
- `LP_CLKRST_LP_I2C_CLK_SEL` 为 0 时，配置时钟源为 CLK_ROOT_FAST。
- `LP_CLKRST_LP_I2C_CLK_SEL` 为 1 时，配置时钟源为 CLK_XTALD2。

选择后的时钟经过小数分频，得到 I2C 的工作时钟 I2C_SCLK：

$$I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

式中 I2C_SCLK_DIV_NUM 代表分频系数的整数部分，I2C_SCLK_DIV_A 代表分频系数小数部分的分子，I2C_SCLK_DIV_B 代表分频系数小数部分的分母。根据时序参数的限制，分频后的 I2C_SCLK 的频率需至少大于 SCL 频率 20 倍。

在 I2C 中：

- 配置 `PCR_I2C_SCLK_DIV_NUM` 调整 I2C_SCLK_DIV_NUM
- 配置 `PCR_I2C_SCLK_DIV_A` 调整 I2C_SCLK_DIV_A
- 配置 `PCR_I2C_SCLK_DIV_B` 调整 I2C_SCLK_DIV_B

26.4.2 滤除 SCL 和 SDA 噪声

SCL_Filter 和 SDA_Filter 滤波器模块实现方式相同，用于滤除 SCL 及 SDA 输入信号上的噪声。通过配置 `I2C_SCL_FILTER_EN` 以及 `I2C_SDA_FILTER_EN` 寄存器可以开启或关闭滤波器。

以 SCL_Filter 为例，当使能 SCL_Filter 功能时，滤波器会连续采样输入信号 SCL，如果输入信号在连续

`I2C_SCL_FILTER_THRES` 个 `I2C_SCLK` 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，`SCL_Filter` 和 `SDA_Filter` 滤波器分别会过滤脉冲宽度小于 `I2C_SCL_FILTER_THRES` 和 `I2C_SDA_FILTER_THRES` 个 `I2C_SCLK` 时钟周期的线路毛刺。

26.4.3 SCL 时钟拉伸

从机模式下，可以通过拉低 SCL 线实现时钟拉伸，在此期间数据传输暂停，可以用来给软件足够的时间处理数据。置位 `I2C_SLAVE_SCL_STRETCH_EN` 位使能 SCL 时钟拉伸功能，配置 `I2C_STRETCH_PROTECT_NUM` 字段来控制 SCL 拉伸后释放的时长以防出现时序错误。

出现以下四种情况时从机可以选择拉低 SCL 线实现时钟拉伸：

1. 地址命中：从机模式下，从机地址与主机发送到 SDA 线上的地址相匹配，且读写标志位为 1。
2. 写满：从机模式下，I2C 控制器的 RX RAM 为满。注意，从机在接收少于 FIFO 深度个字节时（ESP32-C5 I2C 的 FIFO 深度为 32 字节），可以不开启时钟拉伸功能；当要接收大于等于 FIFO 深度个字节时，可以通过 FIFO 阈值中断写 RAM 的乒乓操作，或者开始时钟拉伸功能，给软件提供处理时间。开启时钟拉伸功能时，必须将 `I2C_RX_FULL_ACK_LEVEL` 置 0，来保证功能正确，否则可能会出现不可预计的后果。
3. 读空：从机模式下，I2C 控制器要发送数据，但 TX RAM 为空。
4. 发送 ACK 时：从机模式下置位 `I2C_SLAVE_BYTE_ACK_CTL_EN`，从机会在发送 ACK 时拉低 SCL。软件在此阶段进行一些操作，如数据校验，并通过配置 `I2C_SLAVE_BYTE_ACK_LVL` 控制将要发送的 ACK 的电平高低。要注意的是，当出现从机接收的 RX RAM 满时，要发送的 ACK 电平将由 `I2C_RX_FULL_ACK_LEVEL` 而不是 `I2C_SLAVE_BYTE_ACK_LVL` 决定。此时同样需要将 `I2C_RX_FULL_ACK_LEVEL` 置 0，以保证 SCL 时钟拉伸功能的正常产生。

产生时钟拉伸时，软件可读取 `I2C_STRETCH_CAUSE` 位获取 SCL 时钟拉伸的原因。置位 `I2C_SLAVE_SCL_STRETCH_CLR` 位关闭 SCL 时钟拉伸。

26.4.4 SCL 空闲时产生 SCL 脉冲

通常情况下，在 I2C 总线空闲时，SCL 线一直为高。ESP32-C5 I2C 支持在 I2C 主机处于空闲状态时，可编程配置产生 SCL 脉冲的功能。这个功能仅在 I2C 控制器作为主机时有效。置位 `I2C_SCL_RST_SLV_EN`，硬件会发送 `I2C_SCL_RST_SLV_NUM` 个 SCL 脉冲，之后 `I2C_SCL_RST_SLV_EN` 位会自动清零。

26.4.5 同步

I2C 的寄存器配置用 APB 时钟，I2C 主模块用 `I2C_SCLK`，这之间存在异步处理，需要增加同步的步骤将配置寄存器的值更新进入 I2C 主模块。步骤为先写配置寄存器，再向 `I2C_CONF_UPGATE` 位写 1。需要通过这种方法更新的配置寄存器详见表 26.4-1。

表 26.4-1. I2C 同步寄存器

配置寄存器	寄存器域	地址
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_ADDR_10BIT_RW_CHECK_EN	
	I2C_ADDR_BROADCASTING_EN	
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SLAVE_ADDR_REG	I2C_ADDR_10BIT_EN	0x0010
	I2C_SLAVE_ADDR	
I2C_FIFO_CONF_REG	I2C_FIFO_ADDR_CFG_EN	0x0018
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_STRETCH_CONF_REG	I2C_SLAVE_BYTE_ACK_CTL_EN	0x0084
	I2C_SLAVE_BYTE_ACK_LVL	
	I2C_SLAVE_SCL_STRETCH_EN	
	I2C_STRETCH_PROTECT_NUM	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038
	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

26.4.6 漏级开路输出

SCL 及 SDA 线采用漏级开路的驱动方式。I2C 控制器有两种配置方式实现漏级开路驱动方式：

1. 置位 `I2C_SCL_FORCE_OUT`、`I2C_SDA_FORCE_OUT` 并配置相应 SCL 及 SDA PAD 的 `GPIO_PIN $n$ _PAD_DRIVER` 寄存器为漏级开路驱动。
2. 清零 `I2C_SCL_FORCE_OUT` 以及 `I2C_SDA_FORCE_OUT`。

SCL 和 SDA 配置成开漏方式时, 从低电平转向高电平的时间会较长, 这个转变时间由线上的上拉电阻以及电容共同决定。开漏模式下, I2C 输出频率的占空比受限于 SCL 上拉速度, 主要受 SCL 的速度限制。

另外, 在 `I2C_SCL_FORCE_OUT` 和 `I2C_SCL_PD_EN` 置 1 时, 可以强制拉低 SCL 线; 在 `I2C_SDA_FORCE_OUT` 和 `I2C_SDA_PD_EN` 置 1 时, 可以强制拉低 SDA 线。

26.4.7 时序参数配置

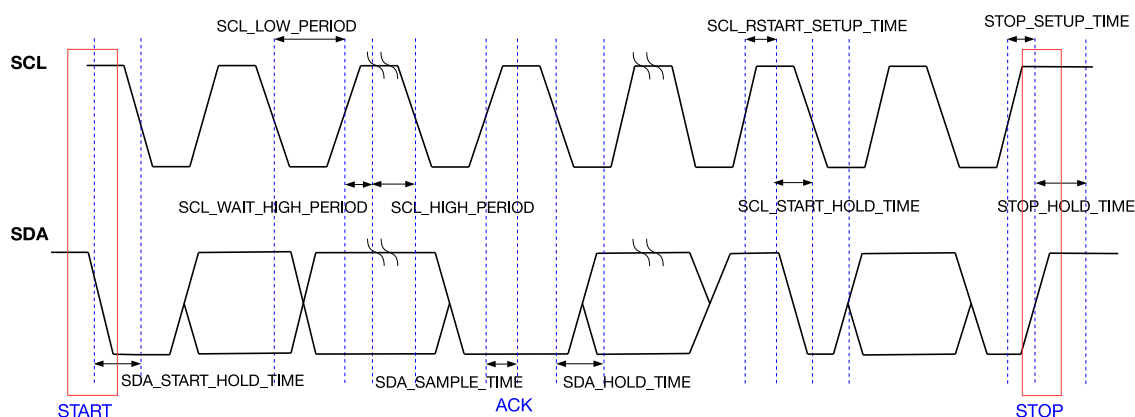


图 26.4-1. I2C 时序图

图 26.4-1 为实现 I2C 协议的 I2C 主机的时序图, 图中的寄存器均用来配置时序参数。I2C 控制器的 START 位、STOP 位、数据保持时间、数据采样时间、SCL 上升沿等待时间等时序均可以通过图 26.4-1 中所示的寄存器进行配置。这些寄存器以模块时钟 (`I2C_SCLK`) 为单位, 与各时序参数的对应关系为:

1. $t_{LOW} = (I2C\_SCL\_LOW\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
2. $t_{HIGH} = (I2C\_SCL\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
3. $t_{SU:STA} = (I2C\_SCL\_RSTART\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
4. $t_{HD:STA} = (I2C\_SCL\_START\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
5. $t_r = (I2C\_SCL\_WAIT\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
6. $t_{SU:STO} = (I2C\_SCL\_STOP\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
7. $t_{BUF} = (I2C\_SCL\_STOP\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
8. $t_{HD:DAT} = (I2C\_SDA\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
9. $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

根据在何种模式下有意义, 下列时序寄存器可分为两组:

- 主机模式:

1. `I2C_SCL_START_HOLD_TIME`: 生成 I2C 协议中的 START 位时, SDA 信号拉低到 SCL 信号拉低的时间间隔。该时间间隔为 (`I2C_SCL_START_HOLD_TIME` + 1) 个模块时钟周期。仅控制器工作在主机模式时有意义。
2. `I2C_SCL_LOW_PERIOD`: SCL 低电平持续时间。SCL 低电平时间为 (`I2C_SCL_LOW_PERIOD` + 1) 个模块时钟周期。该寄存器仅在控制器工作在主机模式时有意义。

SCL 低电平时间会在以下情况下延长:

- 当 I2C 为主机时, 外设拉低 SCL
 - I2C 控制器执行 END 命令拉低 SCL
 - I2C 为从机时, 发生 SCL 时钟拉伸
3. `I2C_SCL_WAIT_HIGH_PERIOD`: 等待 SCL 线拉高的模块时钟周期数。请确保在该时间内 SCL 线可以完成上拉。否则会导致 SCL 高电平持续时间不可预测。仅控制器工作在主机模式时有意义。
 4. `I2C_SCL_HIGH_PERIOD`: SCL 线拉高后维持高电平的模块时钟周期数。仅控制器工作在主机模式时有意义。当 SCL 线在 `I2C_SCL_WAIT_HIGH_PERIOD` + 1 个模块时钟内完成上拉, 则 SCL 线的频率为:

$$f_{scl} = \frac{f_{I2C\_SCLK}}{I2C\_SCL\_LOW\_PERIOD + I2C\_SCL\_HIGH\_PERIOD + I2C\_SCL\_WAIT\_HIGH\_PERIOD + 3 + I2C\_SCL\_FILTER\_THRES}$$

其中 3 个时钟周期是对 SCL 进行同步所需的时间, 如果打开了 SCL 滤波功能, 则还需要加上滤波带来的延迟 `I2C_SCL_FILTER_THRES`。用 `I2C_SCL_WAIT_HIGH_PERIOD` + 1 个模块时钟模拟的 SCL 上拉时间受到上拉电阻、IO 驱动能力、SCL 线上电容等等的的影响, 可能出现测试的实际频率和理论频率有偏差的情况。此时可以通过调整 `I2C_SCL_WAIT_HIGH_PERIOD` 的值来使实际频率和理论频率接近。

- 主机模式和从机模式:

1. `I2C_SDA_SAMPLE_TIME`: SCL 上升沿到采样 SDA 线电平值的时间间隔。推荐设置在 SCL 高电平持续时间的中间值, 以保证能够正确采样到 SDA 线上电平。控制器工作在主机模式及从机模式时都有意义。
2. `I2C_SDA_HOLD_TIME`: SDA 输出数据变化与 SCL 下降沿的时间间隔。控制器工作在主机模式及从机模式时都有意义。

根据时序参数的限制, 对时序寄存器的配置范围也有约束。

1. $\frac{f_{I2C\_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C\_SCLK} \leq (I2C\_SDA\_HOLD\_TIME - 4) \times f_{APB\_CLK}$
3. `I2C_SDA_HOLD_TIME` + `I2C_SCL_START_HOLD_TIME` > `SDA_FILTER_THRES` + 3
4. `I2C_SCL_WAIT_HIGH_PERIOD` < `I2C_SDA_SAMPLE_TIME` < `I2C_SCL_HIGH_PERIOD`
5. `I2C_SDA_SAMPLE_TIME` < `I2C_SCL_WAIT_HIGH_PERIOD` + `I2C_SCL_START_HOLD_TIME` + `I2C_SCL_RSTART_SETUP_TIME`
6. `I2C_STRETCH_PROTECT_NUM` + `I2C_SDA_HOLD_TIME` > `I2C_SCL_LOW_PERIOD`

26.4.8 超时控制

I2C 内部有三种超时控制, 分别是对 SCL_FSM 状态的超时控制、SCL_MAIN_FSM 状态的超时控制和对 SCL 线状态的超时控制。其中前两种是一直使能的, 第三种是可编程配置的。

当 SCL_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_ST\_TO\_I2C+1}$ 个时钟周期后，会触发 I2C_SCL_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_ST_TO_I2C 的配置值最大为 23，即最大在时间超过 2^{24} 个 I2C_SCLK 时钟周期后会产生超时中断。

当 SCL_MAIN_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C+1}$ 个 I2C_SCLK 时钟周期后，会触发 I2C_SCL_MAIN_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_MAIN_ST_TO_I2C 的配置值最大为 23，即最大在时间超过 2^{24} 个模块时钟后会产生超时中断。

使能 I2C_TIME_OUT_EN 打开 SCL 线状态的超时控制。当 SCL 线状态长时间维持同一电平不变，且时间超过 $2^{I2C\_TIME\_OUT\_VALUE+1}$ 后，会触发 I2C_TIME_OUT_INT 中断，I2C 总线回到空闲状态。

26.4.9 指令配置

I2C 控制器工作于主机模式时，CMD_Controller 会依次从八个命令寄存器中读出命令并按照命令来控制 SCL_FSM 及 SCL_MAIN_FSM。

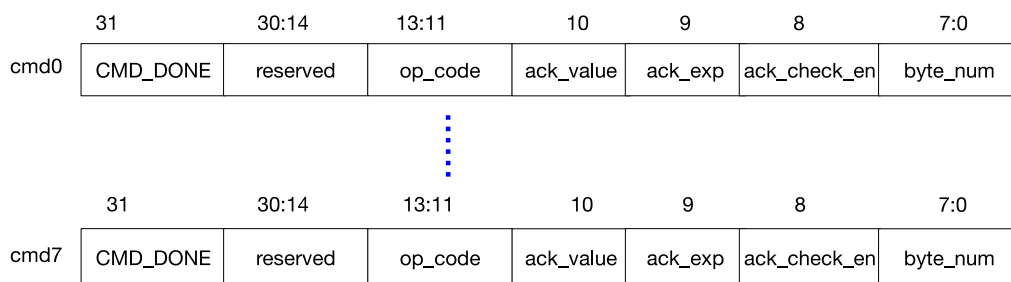


图 26.4-2. I2C 命令寄存器结构

命令寄存器只在 I2C 控制器工作于主机模式时才有效，其内部结构如图 26.4-2 所示。命令寄存器的参数为：

1. CMD_DONE: 命令执行完成标识。每条命令执行完硬件会将对应命令寄存器中的 CMD_DONE 置 1。软件可以通过读取每条命令的 CMD_DONE 位来判断该命令是否执行完毕。每次更新命令时，软件需要将 CMD_DONE 位清零。
2. op_code: 命令编码，共有五种命令：
 - WRITE: op_code 等于 1 时为 WRITE 命令，该命令指示 I2C 控制器向从机发送从机地址、被访问的寄存器地址（仅限双寻址模式）、数据。
 - STOP: op_code 等于 2 时为 STOP 命令，该命令指示 I2C 控制器发送 I2C 协议中的 STOP 位。此条命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件再次启动 CMD_Controller 后，会重新从命令寄存器 0 开始去取指令。
 - READ: op_code 等于 3 时为 READ 命令，该命令指示 I2C 控制器从从机读取数据。
 - END: op_code 等于 4 时为 END 命令，该命令指示 I2C 控制器将 SCL 信号拉低，暂停 I2C 通信。该命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件在更新命令寄存器和 RAM 数据后可重新启动 CMD_Controller，继续进行 I2C 协议传输。再次启动后 CMD_Controller 会重新从命令寄存器 0 开始去取指令。
 - RSTART: op_code 等于 6 时为 RSTART 命令，该命令指示 I2C 控制器发送 I2C 协议中的 START 位或 RESTART 位。

3. `ack_value`: 该位设置读操作时 I2C 控制器在 I2C 协议中的 ACK 位发送的电平值。RSTART、STOP、END、WRITE 命令中该位没有意义。
4. `ack_exp`: 该位用于设置写操作时 I2C 控制器在 I2C 协议中的 ACK 位期望接收的电平值。RSTART、STOP、END、READ 命令中该位没有意义。
5. `ack_check_en`: 该位使能写操作中 I2C 控制器检查从机发送的 ACK 位电平与命令中的 `ack_exp` 是否一致。如果接收的 ACK 值与 WRITE 命令中的 `ack_exp` 电平不一致时, I2C 主机会产生 `I2C_NACK_INT` 中断, 停止发送数据并产生 STOP。此位为 1 时, 检测从机发送的 ACK 位电平; 此位为 0 时, 不检测从机发送的 ACK 位电平。RSTART、STOP、END、READ 命令中该位没有意义。
6. `byte_num`: 读写数据的长度 (单位字节), 最大为 255, 最小为 1。RSTART、STOP、END 命令中 `byte_num` 无意义。

每次命令序列的执行都是从命令寄存器 0 开始, 到 STOP 或 END 命令结束。所以需要保证每个命令序列中必须有 STOP 或 END 命令。

一次完整的 I2C 协议传输应该起始于 START 命令, 结束于 STOP 命令。可通过 END 命令将一次 I2C 协议传输分为多个命令序列来完成。每个命令序列可以改变数据传输的方向、时钟频率、从机地址和数据长度等。这样可以弥补 RAM 大小不足的问题, 也可以实现更灵活的 I2C 通信。

26.4.10 TX/RX RAM 数据存储

TX/RX RAM 大小均为 32 x 8 位。TX RAM 和 RX RAM 均可以通过 FIFO 和直接地址 (non-FIFO) 两种方式访问。将 `I2C_NONFIFO_EN` 位设置成 0, 为 FIFO 方式; `I2C_NONFIFO_EN` 位设置成 1 时为直接地址方式。

TX RAM 用于存储 I2C 控制器需要发送的数据。在 I2C 通信的过程中, 当 I2C 控制器需要发送数据时 (不包括 ACK 位响应), 会依次读出 TX RAM 中的数据并串行输出到 SDA 线上。当 I2C 控制器工作于主机模式时, 所有需要发送给从机的数据都必须按照发送顺序依次存储在 TX RAM 中。包括被访问的从机地址、读写标志位、被访问的寄存器地址 (仅限双地址寻址模式下)、写数据。当 I2C 控制器工作于从机模式时, TX RAM 中只存放写数据。

TX RAM 可被 CPU 读写。CPU 可通过两种方式写 TX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 `I2C_DATA_REG` 写 TX RAM, 硬件自动进行 TX RAM 写地址自增。直接地址访问是通过地址段 (`I2C 基地址 + 0x100`) ~ (`I2C 基地址 + 0x17C`) 直接访问 TX RAM。TX RAM 的每一个字节占据一个字 (word) 的地址。因此, 第一个字节访问地址为 `I2C 基地址 + 0x100`, 第二字节访问地址为 `I2C 基地址 + 0x104`, 第三字节访问地址为 `I2C 基地址 + 0x108`, 以此类推。CPU 只可通过直接地址访问方式读 TX RAM, 且 CPU 可以通过直接地址读回写入到 TX RAM 的字节。读 TX RAM 的地址和写 TX RAM 的地址相同。

RX RAM 存储的是 I2C 通信过程中, I2C 控制器接收到的数据。当 I2C 控制器工作于从机模式时, 主机发送的从机地址及被访问的寄存器地址 (仅限双地址寻址模式下) 都不会存储在 RX RAM 中。软件可以在 I2C 通信结束后, 读出 RX RAM 的值。

RX RAM 只可被 CPU 读。CPU 可通过两种方式读 RX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 `I2C_DATA_REG` 读 RX RAM, 硬件自动完成 RX RAM 读地址自增。直接地址访问是通过地址段 (`I2C 基地址 + 0x180`) ~ (`I2C 基地址 + 0x1FC`) 直接访问 RX RAM。RX RAM 的每一个字节占据一个字的地址。因此, 第一个字节访问地址为 `I2C 基地址 + 0x180`, 第二字节访问地址为 `I2C 基地址 + 0x184`, 第三字节访问地址为 `I2C 基地址 + 0x188`, 以此类推。

在 FIFO 模式下可以对 TX RAM 进行乒乓操作, 来实现发送大于 FIFO 深度的数据。置位 `I2C_FIFO_PRT_EN`, 当 RAM 中剩下的待发送数据字节数小于 `I2C_TXFIFO_WM_THRHD` 时, 会产生 `I2C_TXFIFO_WM_INT` 中断。软件收

到中断后，继续向 `I2C_DATA_REG`（主机）中写数。需要保证向 TX RAM 写数或更新数据的操作提前于 I2C 发送数据的动作，否则会产生不可预计的后果。

在 FIFO 模式下也可以对 RX RAM 进行乒乓操作，来实现接收大于 FIFO 深度的数据。置位 `I2C_FIFO_PRT_EN`，将 `I2C_RX_FULL_ACK_LEVEL` 置 0。当 RAM 中收到的数据字节数大于等于 `I2C_RXFIFO_WM_THRHD`（从机）时，会产生 `I2C_RXFIFO_WM_INT` 中断。软件收到中断后，从 `I2C_DATA_REG`（从机）中读数。

26.4.11 数据转换

DATA_Shifter 模块用于串并转换，当 I2C 发送数据时，将 TX RAM 中的字节数据转化成比特流；当 I2C 接收数据时，将比特流转化成字节数据并存入 RX RAM。`I2C_RX_LSB_FIRST` 和 `I2C_TX_LSB_FIRST` 用于配置最高有效位或最低有效位的优先储存或传输。

26.4.12 寻址模式

ESP32-C5 I2C 支持 7 位寻址模式和 10 位寻址模式，7 位寻址和 10 位寻址可结合使用。除此之外，ESP32-C5 I2C 还支持双地址寻址模式。

假设从机地址为 `SLV_ADDR`。ESP32-C5 I2C 控制器可以使用 7 位寻址（`SLV_ADDR[6:0]`），也可以使用 10 位寻址（`SLV_ADDR[9:0]`）。

对于主机模式而言，7 位寻址下只要发送一个字节地址段 `SLV_ADDR[6:0]` 和读写标志。7 位寻址模式下，有种特殊情况是广播寻址。在从机中，将 `I2C_ADDR_BROADCASTING_EN` 置 1，开启广播寻址模式。当接收到主机发送的地址为广播地址（0x00）且读写标志位为 0 时，此时无论从机自己的地址是多少，都会响应主机。

10 位寻址需要发送两字节地址段。第一个要发送的数是从机地址的第一个 7 位 `slave_addr_first_7bits` 和读写标志，`slave_addr_first_7bits` 的值应该配置为（`0x78 | SLV_ADDR[9:8]`）。第二个要发送的数是 `slave_addr_second_byte`，`slave_addr_second_byte` 的值为 `SLV_ADDR[7:0]`。

在从机中，可以通过配置 `I2C_ADDR_10BIT_EN` 寄存器开启 10 位寻址模式。`I2C_SLAVE_ADDR` 用于配置 I2C 从机地址。`I2C_SLAVE_ADDR[14:7]` 的值应配置为 `SLV_ADDR[7:0]`，`I2C_SLAVE_ADDR[6:0]` 的值应配置为（`0x78 | SLV_ADDR[9:8]`）。由于 10 位从机地址比 7 位地址多一个字节，所以 WRITE 命令对应的 `byte_num` 以及 RAM 中数据数量都相应增加 1。详细说明可参考 [编程示例](#)。

控制器处于从机模式时，还支持双地址寻址模式。双地址的第一个地址是 I2C 从机地址，第二个地址是 I2C 从机的内存地址。双地址模式下，RAM 必须采用直接地址（non-FIFO）的方式访问。

通过置位 `I2C_FIFO_ADDR_CFG_EN` 来使能双地址访问功能。当从机接收到的从机地址和内部配置的从机地址不一致时，会产生 `I2C_SLAVE_ADDR_UNMATCH` 中断。

26.4.13 10 位寻址的读写标志位检查

在 10 位寻址模式下，将 `I2C_ADDR_10BIT_RW_CHECK_EN` 置 1，会对发送的第一个数 `slave_addr_first_7bits` 和读写标志做检查。当读写标志不是写，此时与协议不符合，会结束传输。若不打开此功能，当读写标志不是写，还能支持继续传输，但可能出现传输错误。

26.4.14 启动控制器

对于主机，配置成主机模式和命令寄存器等相关配置后，通过向 `I2C_TRANS_START` 写 1，启动主机解析，执行命令序列。一组命令总是从命令寄存器 0 开始，顺序执行到 STOP 或者 END 命令。当另一组命令需要从命令寄存器 0 开始执行时，需要重新向 `I2C_TRANS_START` 写 1 来更新命令。

对于从机，配置成从机模式并配置其他相关寄存器之后，向 `I2C_TRANS_START` 写 1。从机将会在与主机所发送的地址匹配时，响应主机。

对于从机，请注意：

- 每次接收传输前必须置位 `I2C_TRANS_START`，否则即使地址匹配，从机也不会响应主机。
- 如果主机发起 STOP 提前中止了和从机的传输，那么在下一次传输前，从机必须先置位 `I2C_SLV_TX_AUTO_START_EN`，再置位 `I2C_TRANS_START`，最后清零 `I2C_SLV_TX_AUTO_START_EN`，否则从机的 TX FIFO 将会出现指针异常。

26.5 LP_I2C 和 I2C 的功能差异

LP_I2C 可以作为主机在主系统休眠时和外部设备通信，它包含所有 ESP32-C5 I2C 主机部分的功能，但不包含任何 ESP32-C5 I2C 从机部分的功能。因此，LP_I2C 不包含任何与从机功能相关的寄存器，详细的寄存器列表见 [26.8.2 LP_I2C 寄存器列表](#)。

LP_I2C 和 I2C 主机在设计上的区别有以下几点：

- LP_I2C 中 TX/RX RAM 为 16x8 位大小，即 TX/RX FIFO 深度变为 16 字节。
- LP_I2C 的 APB_CLK 的时钟源为 `CLK_AON_FAST`。通过配置 `LP_CLKRST_LP_I2C_CLK_SEL` 选择 I2C_SCLK 的时钟源。具体时钟源配置为：
 - 0: `CLK_ROOT_FAST`
 - 1: `CLK_XTALD2`

配置 `LPPERI_LP_EXT_I2C_CK_EN` 为高电平来使能 I2C_SCLK 的时钟源。时钟频率发生变化时要注意对时序寄存器做出相应调整。

LP_I2C 的编程示例参见 [26.7](#) 中的主机部分。

26.6 中断

ESP32-C5 的 `I2Cn` 可以生成 `I2Cn_INTR` 中断信号并发送给 [中断矩阵](#)。这一中断信号由以下 `I2Cn` 的内部中断源生成：

- `I2C_SLAVE_STRETCH_INT`：从机模式下，当从机时钟拉伸时，产生此中断。
- `I2C_DET_START_INT`：主机或从机检测到 I2C START 信号，触发此中断。
- `I2C_SCL_MAIN_ST_TO_INT`：当 I2C 主状态机 `SCL_MAIN_FSM` 保持某个状态超过 $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C+1}$ 个模块时钟周期时，触发此中断。
- `I2C_SCL_ST_TO_INT`：当 I2C 状态机 `SCL_FSM` 保持某个状态超过 $2^{I2C\_SCL\_ST\_TO\_I2C+1}$ 个模块时钟周期时，触发此中断。
- `I2C_RXFIFO_UDF_INT`：当 I2C 通过 APB 总线读取 RX FIFO，但 RX FIFO 为空时，触发该中断。
- `I2C_TXFIFO_OVF_INT`：当 I2C 通过 APB 总线写 TX FIFO，但 TX FIFO 为满时，触发该中断。
- `I2C_NACK_INT`：当 I2C 配置为主机时，接收到的 ACK 与命令中期望的 ACK 值不一致时，即触发该中断；当 I2C 配置为从机时，接收到的 ACK 值为 1 时即触发该中断。
- `I2C_TRANS_START_INT`：当 I2C 发送一个 START 位时，即触发该中断。

- I2C_TIME_OUT_INT: 在传输过程中, 当 I2C SCL 保持为高或为低电平的时间超过 $2 \times \text{I2C\_TIME\_OUT\_VALUE} + 1$ 个模块时钟后, 即触发该中断。
- I2C_TRANS_COMPLETE_INT: 当 I2C 检测到 STOP 位时, 即触发该中断。
- I2C_MST_TXFIFO_UDF_INT: 当 I2C 主机的 TX FIFO 下溢时, 触发此中断。
- I2C_ARBITRATION_LOST_INT: 当 I2C 主机的 SCL 为高电平, SDA 输出值与输入值不相等时, 即触发该中断。
- I2C_BYTE_TRANS_DONE_INT: 当 I2C 发送或接收一个字节, 即触发该中断。
- I2C_END_DETECT_INT: 当 I2C 主机命令的 op_code 为 END, 且检测到 I2C END 状态时, 触发此中断。
- I2C_RXFIFO_OVF_INT: 当 I2C RX FIFO 上溢时, 触发此中断。
- I2C_TXFIFO_WM_INT: I2C TX FIFO 水标中断。当 I2C_FIFO_PRT_EN 为 1, 且 TX FIFO 指针小于 I2C_TXFIFO_WM_THRHD[4:0] 时, 触发此中断。
- I2C_RXFIFO_WM_INT: I2C RX FIFO 水标中断。当 I2C_FIFO_PRT_EN 为 1, 且 RX FIFO 指针大于 I2C_RXFIFO_WM_THRHD[4:0] 时, 触发此中断。
- I2C_GENERAL_CALL_INT: 从机模式下, 接收到通用广播地址时, 触发此中断。
- I2C_SLAVE_ADDR_UNMATCH_INT: 从机模式下, 接收到的从机地址和内部配置的从机地址不一致时, 触发此中断。

说明:

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 [to be added later] > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器, 详见章节 中断配置寄存器。具体寄存器名称请查看 31.5 寄存器列表。

26.7 配置流程

本节提供一些典型通信场景的编程示例。ESP32-C5 中有两个 I2C 控制器, 为了便于描述, 下文所有图示中的 I2C 主机和从机都假定为 ESP32-C5 I2C 外设控制器。

26.7.1 I2C 主机写入从机, 7 位寻址, 单次命令序列

26.7.1.1 场景介绍

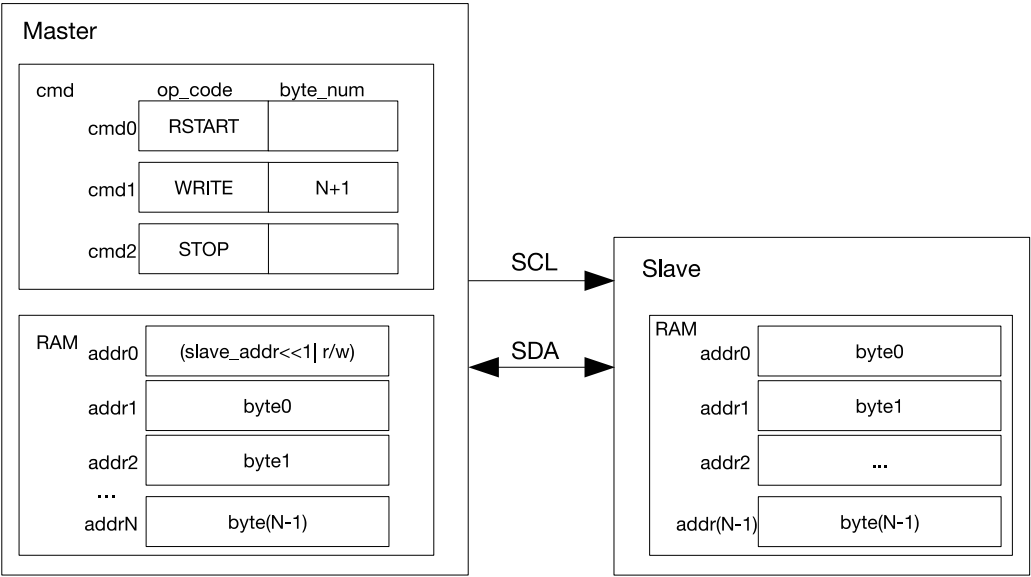


图 26.7-1. I2C 主机写 7 位寻址的从机

图 26.7-1 为 I2C 主机采用 7 位寻址写 N 个字节数据到 I2C 从机的寄存器或 RAM。如图 26.7-1 所示，主机 RAM 中第一个字节数据为 7 位从机地址 + 1 位读写标志位，其中读写标志位为 0 时表示写操作，接下来的连续空间存储待发送的数据。cmd 框中包含了相应的命令序列。

对于主机，在软件配置好命令序列以及 RAM 数据后，置位 I2C_TRANS_START 寄存器启动控制器进行数据传输。控制器的行为可分为四步：

- 1. 等待 SCL 线为高电平，以避免 SCL 线被其他主机或者从机占用。
- 2. 通过发送 START 位来执行 RSTART 命令。
- 3. 执行 WRITE 命令从 RAM 的首地址开始取出 N+1 个字节并依次发送给从机，其中第一个字节为地址。
- 4. 发送 STOP。当 I2C 主机完成 STOP 位的传输后，会产生 I2C_TRANS_COMPLETE_INT 中断。

26.7.1.2 配置示例

- 1. 参照章节 26.4.7，配置 I2C 主机和 I2C 从机的时序参数寄存器。
- 2. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
- 3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
- 4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2（主机）	STOP	—	—	—	—

- 5. 参考章节 26.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
- 6. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。

7. 向 I2C_CONF_UPGATE (主机) 和 I2C_CONF_UPGATE (从机) 写 1 来同步寄存器。
8. 向 I2C_TRANS_START (主机) 和 I2C_TRANS_START (从机) 位写 1 开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR (从机)，当 I2C 主机 WRITE 命令中的 ack_check_en (主机) 配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
 - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平一致，传输继续。
 - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平不一致，I2C 主机产生 I2C_NACK_INT (主机) 中断，停止发送数据并且产生 STOP。
10. I2C 主机发送数据，并根据 ack_check_en (主机) 的配置决定是否对 ACK 信号进行检测。
11. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 26.4.10。
12. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 26.4.10。

若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN (从机) 使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT (从机) 中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR (从机) 置 1 清除中断，并将 I2C_SLAVE_SCL_STRETCH_CLR (从机) 置 1 释放 SCL 总线。
13. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT (主机) 中断。

26.7.2 I2C 主机写入从机，10 位寻址，单次命令序列

26.7.2.1 场景介绍

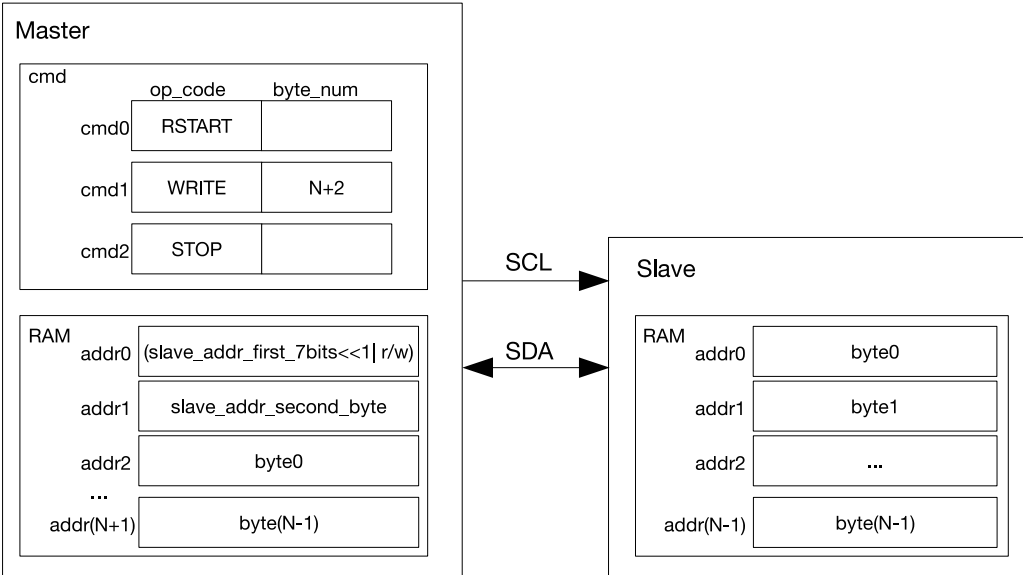


图 26.7-2. I2C 主机写 10 位寻址的从机

图 26.7-2 为 I2C 主机写 N 个字节到 10 位地址 I2C 从机的配置图。整个配置和传输过程都和 26.7.1 中类似，除了在传输的开始主机在 10 位寻址模式下需要发送两字节地址段。由于 10 位从机地址比 7 位地址多一个字节，

所以 WRITE 命令对应的 byte_num 以及 TX RAM 中数据数量都相应增加 1。

26.7.2.2 配置示例

1. 设置 I2C_MS_MODE (主机) 为 1, I2C_MS_MODE (从机) 为 0。
2. 向 I2C_CONF_UPGATE (主机) 和 I2C_CONF_UPGATE (从机) 写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0 (主机)	RSTART	—	—	—	—
I2C_COMMAND1 (主机)	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2 (主机)	STOP	—	—	—	—

4. 在 I2C_SLAVE_ADDR_REG (从机) 的 I2C_SLAVE_ADDR (从机) 设置 I2C 从机的 10 位从机地址, 并将 I2C_ADDR_10BIT_EN (从机) 置 1 使能 10 位寻址模式。
5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据, 第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W, 第二个地址字节是 I2C_SLAVE_ADDR[7:0]。之后就是要发送的数据, 可选 FIFO 方式和直接访问方式。
6. 向 I2C_CONF_UPGATE (主机) 和 I2C_CONF_UPGATE (从机) 写 1 来同步寄存器。
7. 向 I2C_TRANS_START (主机) 和 I2C_TRANS_START (从机) 位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR (从机), 当 I2C 主机 WRITE 命令中的 ack_check_en (主机) 配置为 1 时, I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0, 则不会对 ACK 检测, 会默认为匹配。
 - 匹配: 接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平一致, 传输继续。
 - 不匹配: 接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平不一致, I2C 主机产生 I2C_NACK_INT (主机) 中断, 停止发送数据并且产生 STOP。
9. I2C 主机发送数据, 并根据 ack_check_en (主机) 的配置判断是否进行 ACK 检测。
10. 若发送数据 N 大于 TX FIFO 深度, 在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作, 具体做法参照章节 26.4.10。
11. 若接收数据 N 大于 RX FIFO 深度, 在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作, 具体做法参照章节 26.4.10。
 若接收数据 N 大于 RX FIFO 深度, 另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN (从机) 使能 SCL 时钟拉伸, 同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT (从机) 中断。此时 I2C 从机会将 SCL 拉低, 软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR (从机) 置 1 清除中断, 并将 I2C_SLAVE_SCL_STRETCH_CLR (从机) 置 1 释放 SCL 总线。
12. 当整个传输正常结束, I2C 主机执行 STOP 命令, 并产生 I2C_TRANS_COMPLETE_INT (主机) 中断。

26.7.3 I2C 主机写入从机, 7 位双地址寻址, 单次命令序列

26.7.3.1 场景介绍

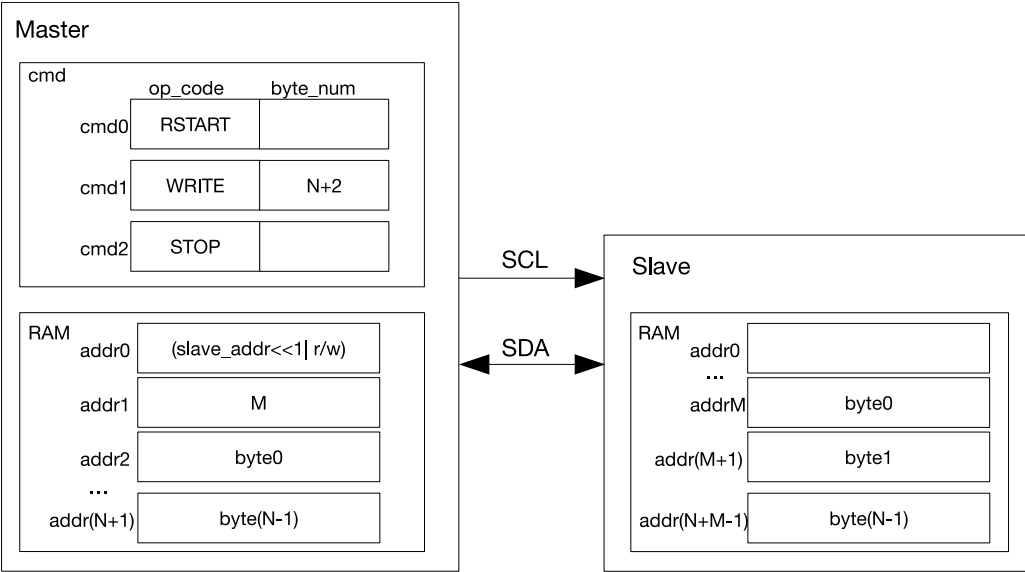


图 26.7-3. I2C 主机写 7 位双地址寻址从机

图26.7-3 为 I2C 主机采用 7 位双寻址模式写 N 个字节数据到 I2C 从机的寄存器或 RAM 的值。整个配置和传输过程都和章节 26.7.1 中类似，区别是传输的开始主机在 7 位双寻址模式下需要发送两个字节。双地址的第一个地址是 7 位从机地址，第二个地址是 I2C 从机的内存地址（即图 26.7-3 中的 addrM）。双地址模式下，RX RAM 必须采用 non-FIFO 方式访问。另一个区别是，I2C 从机将接收到的数据 byte0 ~ byte (N-1) 从 RX RAM 中的 addrM 开始依次存储，addrM 就是主机发送的 I2C 内存地址。当超出地址 31 后会从地址 0 开始继续存储。

26.7.3.2 配置示例

- 1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
- 2. 设置I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
- 3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
- 4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

- 5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，可以用 FIFO 方式或直接访问方式。
- 6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
- 7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
- 8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
- 9. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。

- 匹配：接收的 ACK 值与 WRITE 命令中的 `ack_exp`（主机）电平一致，传输继续。
 - 不匹配：接收的 ACK 值与 WRITE 命令中的 `ack_exp`（主机）电平不一致，I2C 主机产生 `I2C_NACK_INT`（主机）中断，停止发送数据并且产生 STOP。
10. I2C 从机接收到 I2C 主机发送的内存地址，完成 RX RAM 的地址偏移。
 11. I2C 主机发送数据，并根据 `ack_check_en`（主机）的配置判断是否进行 ACK 检测。
 12. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 26.4.10。
 13. 若接收数据 N 大于 RX FIFO 深度，置位 `I2C_SLAVE_SCL_STRETCH_EN`（从机）使能 SCL 时钟拉伸，同时将 `I2C_RX_FULL_ACK_LEVEL` 置 0。RX RAM 满时会产生 `I2C_SLAVE_STRETCH_INT`（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 `I2C_SLAVE_STRETCH_INT_CLR`（从机）置 1 清除中断，并将 `I2C_SLAVE_SCL_STRETCH_CLR`（从机）置 1 释放 SCL 总线。
 14. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 `I2C_TRANS_COMPLETE_INT`（主机）中断。

26.7.4 I2C 主机写入从机，7 位寻址，多次命令序列

26.7.4.1 场景介绍

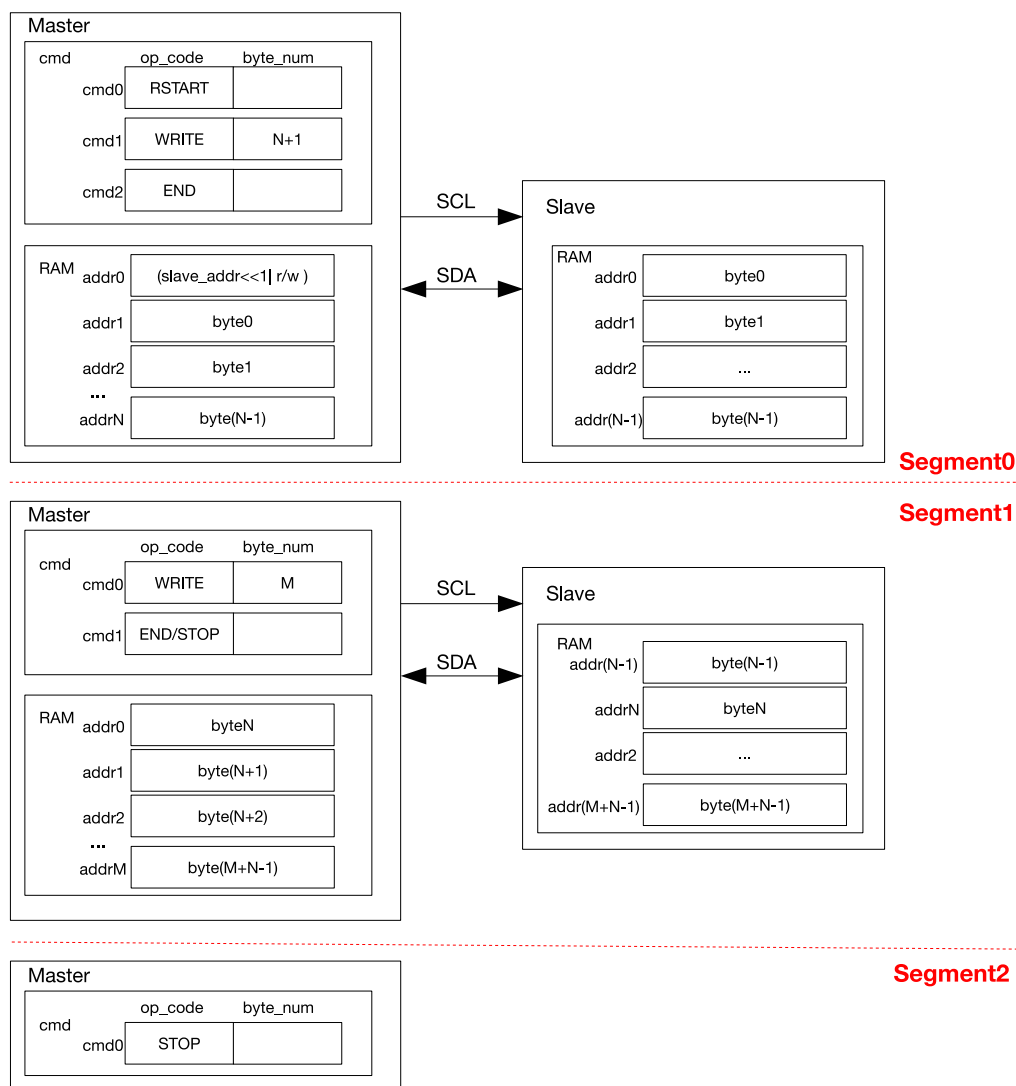


图 26.7-4. I2C 主机分段写 7 位寻址的从机

RAM 的大小只有 TX/RX FIFO 深度，当进行大量的数据传输时，建议使用多次命令序列进行分段传输。每次命令序列以 END 命令结束。当控制器执行 END 命令时，将拉低 SCL 线，软件此时可以更新命令序列寄存器和 RAM 的内容以用于下一次命令序列的传输。

以两段和三段传输为例，如图 26.7-4 所示为 I2C 主机分成三段或者两段写从机。配置 I2C 主机的命令序列如第一段所示，并且在主机的 RAM 中准备好数据，置位 `I2C_TRANS_START`，I2C 主机即开始数据传输。在执行到 END 命令后，I2C 主机会关闭 SCL 时钟，并将 SCL 线拉低来防止其他设备占用 I2C 总线。此时控制器产生 `I2C_END_DETECT_INT` 中断。

在检测到 `I2C_END_DETECT_INT` 中断后，软件可以更新命令序列以及 RAM 中的内容如第二段所示，并清除 `I2C_END_DETECT_INT` 中断。当第二段中 `cmd1` 为 STOP 时，不需要第三段，即为两段写从机。置位 `I2C_TRANS_START` 后，I2C 主机继续发送数据，并在最后发送 STOP 位。

当为三段写从机时，I2C 主机在第二段发送完数据，并检测到 I2C 主机的 `I2C_END_DETECT_INT` 中断后，即可配置 `cmd` 如第三段所示。置位 `I2C_TRANS_START` 后，I2C 主机即产生 STOP 位，从而停止传输。

请注意，在两个分段之间，I2C 总线上的其他主机设备不会占用总线。只有在发送了 STOP 命令后总线才会被

释放。任何情况下，置位 `I2C_FSM_RST` 可复位 I2C 控制器，该寄存器硬件自清 `I2C_FSM_RST`。

26.7.4.2 配置示例

- 1. 设置 `I2C_MS_MODE`（主机）为 1，`I2C_MS_MODE`（从机）为 0。
- 2. 向 `I2C_CONF_UPGATE`（主机）和 `I2C_CONF_UPGATE`（从机）写 1 来同步寄存器。
- 3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
<code>I2C_COMMAND0</code> （主机）	RSTART	—	—	—	—
<code>I2C_COMMAND1</code> （主机）	WRITE	ack_value	ack_exp	1	N+1
<code>I2C_COMMAND2</code> （主机）	END	—	—	—	—

- 4. 参考章节 26.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
- 5. 在 `I2C_SLAVE_ADDR_REG`（从机）的 `I2C_SLAVE_ADDR`（从机）设置 I2C 从机的地址。
- 6. 向 `I2C_CONF_UPGATE`（主机）和 `I2C_CONF_UPGATE`（从机）写 1 来同步寄存器。
- 7. 向 `I2C_TRANS_START`（主机）和 `I2C_TRANS_START`（从机）位写 1 开始传输。
- 8. I2C 从机比较 I2C 主机发送的从机地址和自己的 `I2C_SLAVE_ADDR`（从机），当 I2C 主机 WRITE 命令中的 `ack_check_en`（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 `ack_check_en` 配置为 0，则不会对 ACK 检测，会默认为匹配。
 - 匹配：接收的 ACK 值与 WRITE 命令中的 `ack_exp`（主机）电平一致，传输继续。
 - 不匹配：接收的 ACK 值与 WRITE 命令中的 `ack_exp`（主机）电平不一致，I2C 主机产生 `I2C_NACK_INT`（主机）中断，停止发送数据并且产生 STOP。
- 9. I2C 主机发送数据，并根据 `ack_check_en`（主机）配置的不同进行或不进行 ACK 检测。
- 10. 等到 `I2C_END_DETECT_INT`（主机）中断产生后，设置 `I2C_END_DETECT_INT_CLR`（主机）为 1 来清除中断。
- 11. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
<code>I2C_COMMAND0</code> （主机）	WRITE	ack_value	ack_exp	1	M
<code>I2C_COMMAND1</code> （主机）	END/STOP	—	—	—	—

- 12. 向 I2C 主机的 TX RAM 写入 M 个要发送的数据，可以用 FIFO 方式或直接访问方式。
- 13. 向 `I2C_TRANS_START`（主机）位写 1 开始传输，并重复步骤 9 的流程。
- 14. 若指令为 STOP，I2C 主机执行 STOP 命令结束传输，并产生 `I2C_TRANS_COMPLETE_INT`（主机）中断。
- 15. 若 `I2C_COMMAND1`（主机）的指令为 END，则重复步骤 10。
- 16. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
<code>I2C_COMMAND1</code> （主机）	STOP	—	—	—	—

17. 向 I2C_TRANS_START（主机）位写 1 开始传输。
18. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

26.7.5 I2C 主机读取从机，7 位寻址，单次命令序列

26.7.5.1 场景介绍

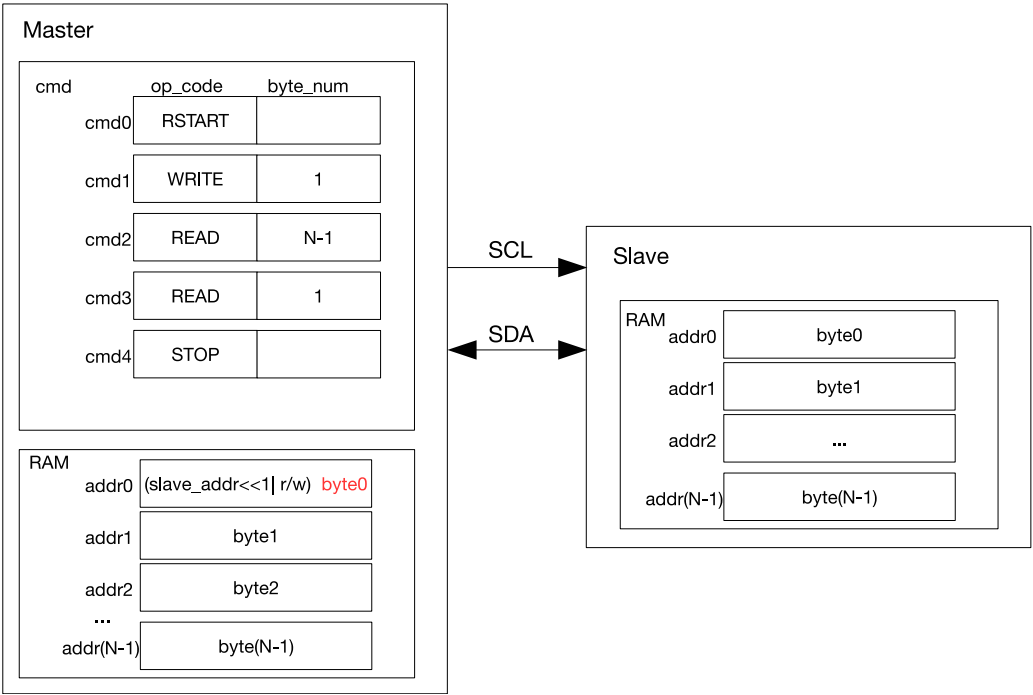


图 26.7-5. I2C 主机读 7 位寻址的从机

图 26.7-5 I2C 主机从 7 位寻址 I2C 从机读取 N 个字节数据的寄存器或 RAM 的值。cmd1 为 WRITE 命令，I2C 主机会将 I2C 从机的地址发送出去。该命令发送的字节是 7 位 I2C 从机地址以及读写标志位。读写标志位为 1 表示读操作。I2C 从机在地址匹配成功之后即开始发送数据给 I2C 主机。I2C 主机根据 READ 命令中设置的 ack_value，在接收完一个字节的的数据之后回复 ACK。

图 26.7-5 中 READ 分成两次，I2C 主机对 cmd2 中 N-1 个数据均回复 ACK，对 cmd3 中的数据即传输的最后一个数据回复 NACK，实际使用时可以根据需要进行配置。在存储接收的数据时，I2C 主机从 RX RAM 的首地址开始存储，即为图中红色 byte0 存储的位置。

26.7.5.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。

4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0 (主机)	RSTART	—	—	—	—
I2C_COMMAND1 (主机)	WRITE	0	0	1	1
I2C_COMMAND2 (主机)	READ	0	0	1	N-1
I2C_COMMAND3 (主机)	READ	1	0	1	1
I2C_COMMAND4 (主机)	STOP	—	—	—	—

- 参考章节 26.4.10，向 I2C 主机的 TX RAM 写入从机地址。可选 FIFO 方式和直接访问方式。
- 在 I2C_SLAVE_ADDR_REG (从机) 的 I2C_SLAVE_ADDR (从机) 设置 I2C 从机的地址。
- 向 I2C_CONF_UPGATE (主机) 和 I2C_CONF_UPGATE (从机) 写 1 来同步寄存器。
- 向 I2C_TRANS_START (主机) 写 1 开始主机的传输。
- 参考章节 26.4.14，启动从机的传输。
- I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR (从机)，当 I2C 主机 WRITE 命令中的 ack_check_en (主机) 配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
 - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平一致，传输继续。
 - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平不一致，I2C 主机产生 I2C_NACK_INT (主机) 中断，停止发送数据并且产生 STOP。
- 等待 I2C_SLAVE_STRETCH_INT (从机)，读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
- 参考章节 26.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
- 将 I2C_SLAVE_SCL_STRETCH_CLR (从机) 置 1，释放 SCL 总线。
- I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en (主机) 配置的不同进行或不进行 ACK 检测。
- 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT (从机) 中断。此时 I2C 从机将会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR (从机) 置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR (从机) 置 1 释放 SCL 总线。
- 当 I2C 主机接收最后一个数据时，将 ack_value (主机) 设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。
- 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT (主机) 中断。

26.7.6 I2C 主机读取从机，10 位寻址，单次命令序列

26.7.6.1 场景介绍

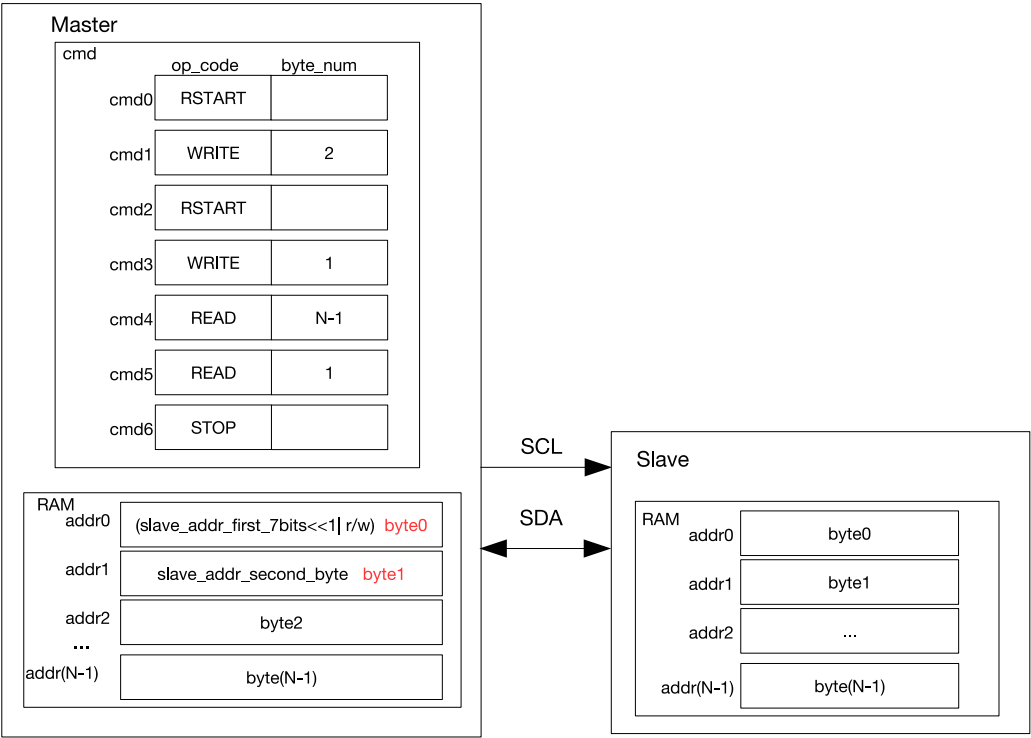


图 26.7-6. I2C 主机读 10 位寻址的从机

图 26.7-6 为 I2C 主机从 10 位寻址的 I2C 从机中读取数据的寄存器或 RAM 的值。相比于 7 位寻址，I2C 主机的第一写命令的字节数为两个字节，相应 TX RAM 中存储两个字节的 I2C 从机 10 位地址，且第一个地址字节的 R/W 位为 W（主机）。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位为 R（从机）。之后主机从从机中读取数据。两个字节地址的配置方式与章节 26.7.2 的相同。

26.7.6.2 配置示例

- 1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
- 2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
- 3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
- 4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1
I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

5. 在 `I2C_SLAVE_ADDR_REG` (从机) 的 `I2C_SLAVE_ADDR` (从机) 设置 I2C 从机的 10 位从机地址, 并将 `I2C_ADDR_10BIT_EN` (从机) 置 1 使能 10 位寻址模式。
6. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据, 第一个地址字节是 $((0x78 | I2C\_SLAVE\_ADDR[9:8]) \ll 1) | R/W$, R/W 位为 W (主机); 第二个地址字节是 `I2C_SLAVE_ADDR[7:0]`。第三个字节是重复的第一个地址字节加上 R/W 位, 其中 R/W 位为 R (从机)。可选 FIFO 方式和直接访问方式。
7. 向 `I2C_CONF_UPGATE` (主机) 和 `I2C_CONF_UPGATE` (从机) 写 1 来同步寄存器。
8. 向 `I2C_TRANS_START` (主机) 写 1 开始主机的传输。
9. 参考章节 26.4.14, 启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的 `I2C_SLAVE_ADDR` (从机), 当 I2C 主机 WRITE 命令中的 `ack_check_en` (主机) 配置为 1 时, I2C 主机会在发送完每个字节之后进行 ACK 检测。若 `ack_check_en` 配置为 0, 则不会对 ACK 检测, 会默认为匹配。
 - 匹配: 接收的 ACK 值与 WRITE 命令中的 `ack_exp` (主机) 电平一致, 传输继续。
 - 不匹配: 接收的 ACK 值与 WRITE 命令中的 `ack_exp` (主机) 电平不一致, I2C 主机产生 `I2C_NACK_INT` (主机) 中断, 停止发送数据并且产生 STOP。
11. I2C 主机发送 RSTART 命令, 并发送 TX RAM 里的第三个字节, 即为重复的地址字节和 R 位。
12. I2C 从机重复执行步骤 10, 若地址匹配, 继续后面的步骤。
13. 等待 `I2C_SLAVE_STRETCH_INT` (从机), 读取 `I2C_STRETCH_CAUSE` 的值为 0, 此时从机地址与 SDA 线上发送的地址相匹配, 且从机要发送数据。
14. 参考章节 26.4.10, 向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
15. 将 `I2C_SLAVE_SCL_STRETCH_CLR` (从机) 置 1, 释放 SCL 总线。
16. I2C 从机发送数据, I2C 主机会根据当前 READ 指令对应的 `ack_check_en` (主机) 配置的不同进行或不进行 ACK 检测。
17. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度, 当发送数据全部发完, 在 I2C 从机的 TX RAM 空时产生 `I2C_SLAVE_STRETCH_INT` (从机) 中断。此时 I2C 从机会将 SCL 拉低, 软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据, 也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 `I2C_SLAVE_STRETCH_INT_CLR` (从机) 置 1 清除中断, 将 `I2C_SLAVE_SCL_STRETCH_CLR` (从机) 置 1 释放 SCL 总线。
18. 当 I2C 主机接收最后一个数据时, 将 `ack_value` (主机) 设成 1, I2C 从机接收到 `I2C_NACK_INT` 中断, 停止发送。
19. 当整个传输正常结束, I2C 主机执行 STOP 命令, 并产生 `I2C_TRANS_COMPLETE_INT` (主机) 中断。

26.7.7 I2C 主机读取从机, 7 位双寻址, 单次命令序列

26.7.7.1 场景介绍

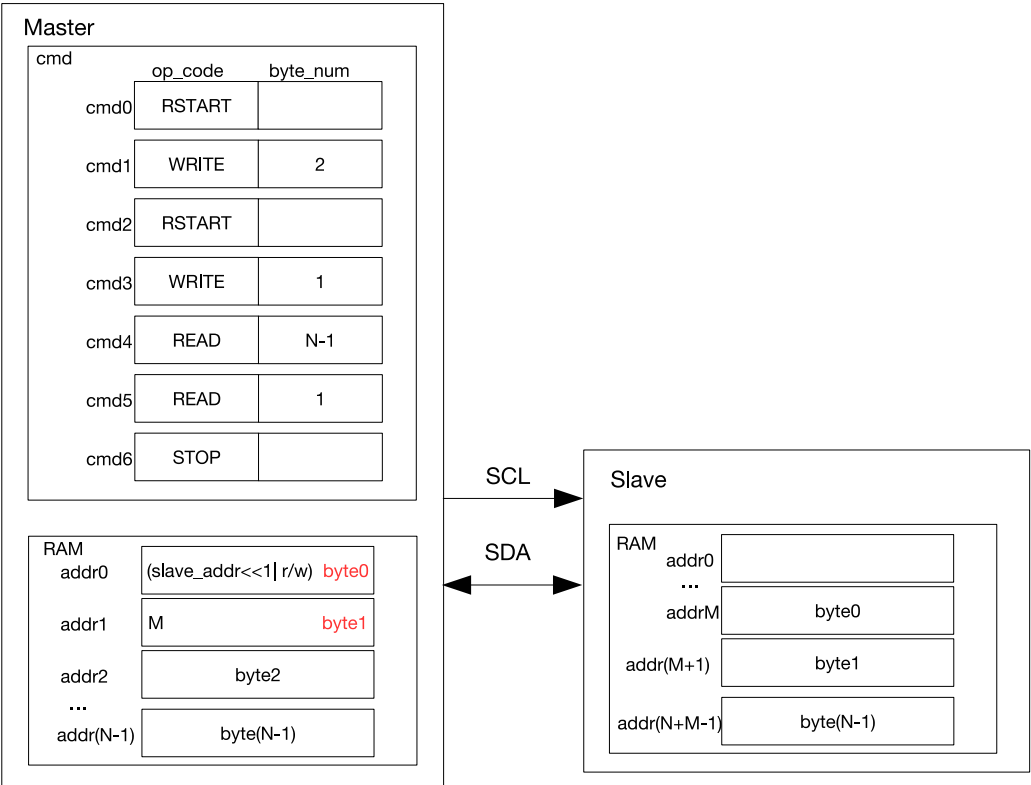


图 26.7-7. I2C 主机从 7 位寻址从机的 M 地址读取 N 个数据

图 26.7-7 为 I2C 主机从 I2C 从机中指定地址读取数据的寄存器或 RAM 的值。主机在传输开始发送 2 个地址字节，第一个地址字节是从机的 7 位地址加 R/W 位，R/W 位为 W（主机）；第二个地址字节是从机的内存地址 M。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位变为 R（从机）。之后主机从从机的 AddrM 地址开始读取数据。

26.7.7.2 配置示例

- 1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
- 2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
- 3. 设置 I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
- 4. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
- 5. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1

I2C_COMMAND5 (主机)	READ	1	0	1	1
I2C_COMMAND6 (主机)	STOP	—	—	—	—

6. 在 I2C_SLAVE_ADDR_REG (从机) 的 I2C_SLAVE_ADDR (从机) 设置 I2C 从机的 7 位地址, I2C_ADDR_10BIT_EN (从机) 置 0 使能 7 位寻址模式。
7. 参考章节 26.4.10, 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据, 第一个地址字节是 (I2C_SLAVE_ADDR[6:0])«1|R/W, R/W 位为 W (主机); 第二个地址字节是 I2C 从机的内存地址 M。第三个字节是重复的第一个地址字节加上 R/W 位, 其中 R/W 位为 R (从机)。可选 FIFO 方式和直接访问方式。
8. 向 I2C_CONF_UPGATE (主机) 和 I2C_CONF_UPGATE (从机) 写 1 来同步寄存器。
9. 向 I2C_TRANS_START (主机) 写 1 开始主机的传输。
10. 参考章节 26.4.14, 启动 I2C 从机的传输。
11. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR (从机), 当 I2C 主机 WRITE 命令中的 ack_check_en (主机) 配置为 1 时, I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0, 则不会对 ACK 检测, 会默认为匹配。
 - 匹配: 接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平一致, 传输继续。
 - 不匹配: 接收的 ACK 值与 WRITE 命令中的 ack_exp (主机) 电平不一致, I2C 主机产生 I2C_NACK_INT (主机) 中断, 停止发送数据并且产生 STOP。
12. I2C 从机接收到 I2C 主机发送的内存地址, 完成 TX RAM 的地址偏移。
13. I2C 主机发送 RSTART 命令, 并发送 TX RAM 里的第三个字节, 即为重复的地址字节和 R 位。
14. I2C 从机重复执行步骤 11, 若地址匹配, 继续后面的步骤。
15. 等待 I2C_SLAVE_STRETCH_INT (从机), 读取 I2C_STRETCH_CAUSE 的值为 0, 此时从机地址与 SDA 线上发送的地址相匹配, 且从机要发送数据。
16. 参考章节 26.4.10, 向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
17. 将 I2C_SLAVE_SCL_STRETCH_CLR (从机) 置 1, 释放 SCL 总线。
18. I2C 从机发送数据, I2C 主机会根据当前 READ 指令对应的 ack_check_en (主机) 配置的不同进行或不进行 ACK 检测。
19. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度, 当发送数据全部发完, 在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT (从机) 中断。此时 I2C 从机会将 SCL 拉低, 软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据, 也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR (从机) 置 1, 清除中断; I2C_SLAVE_SCL_STRETCH_CLR (从机) 置 1, 释放 SCL 总线。
20. 当 I2C 主机接收最后一个数据时, 将 ack_value (主机) 设成 1, I2C 从机接收到 NACK 中断, 停止发送。
21. 当整个传输正常结束, I2C 主机执行 STOP 命令, 并产生 I2C_TRANS_COMPLETE_INT (主机) 中断。

26.7.8 I2C 主机读取从机, 7 位寻址, 多次命令序列

26.7.8.1 场景介绍

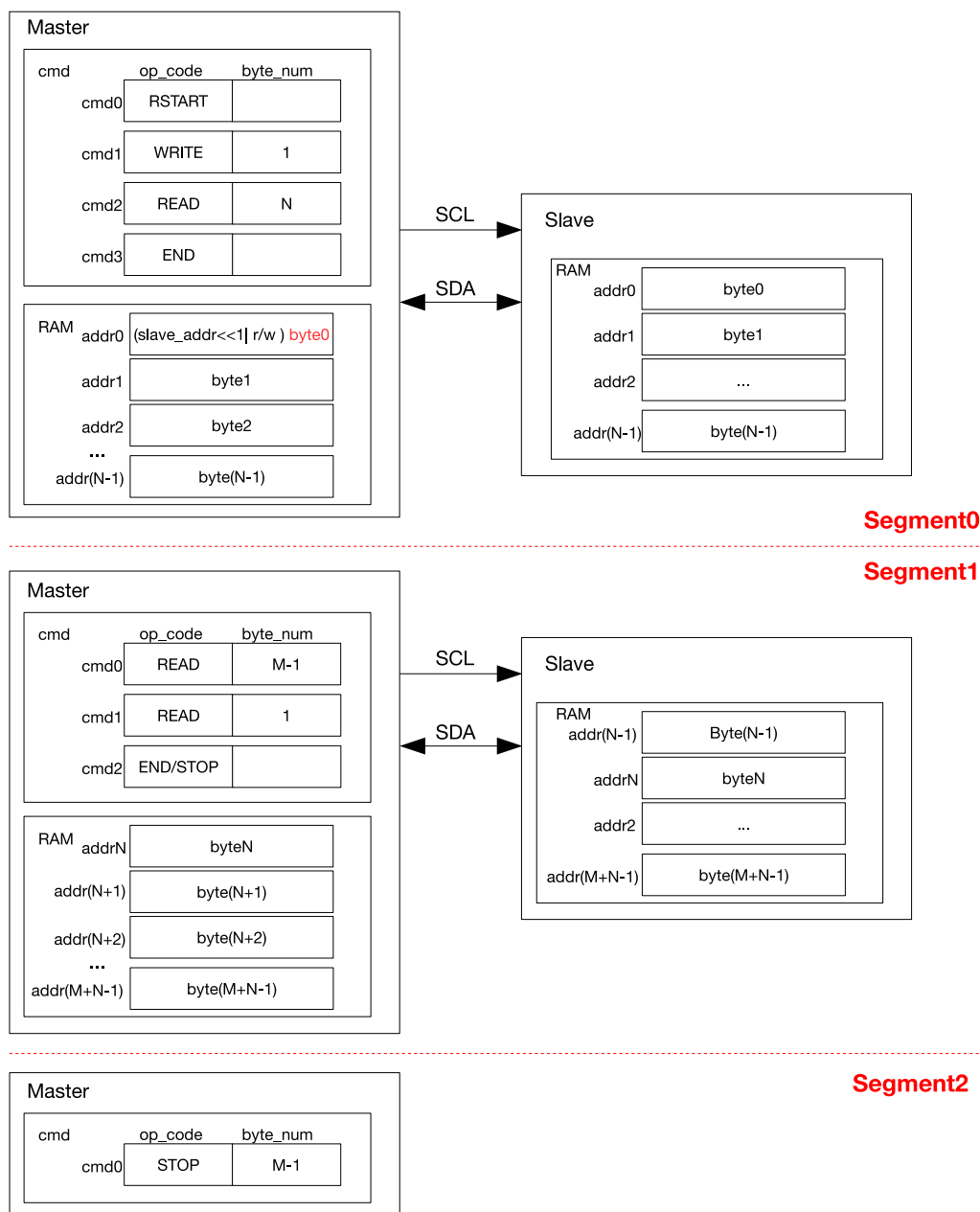


图 26.7-8. I2C 主机分段读 7 位寻址的从机

图 26.7-8 为 I2C 主机通过 END 命令分三段或者分两段，从 I2C 从机读取 N+M 个数据的流程图。

1. 第一段流程和 26.7-5 类似，只是最后一个命令变为 END。
2. 接着在从机的 TX RAM 中准备好数据，置位 `I2C_TRANS_START`，当执行到 END 命令时，I2C 主机可以更新命令寄存器和 RAM 的内容，如第二段所示，并且清零其对应的 `I2C_END_DETECT_INT` 中断。当第二段中 cmd2 为 STOP 时，即两段读 I2C 从机，置位 `I2C_TRANS_START`，I2C 主机继续传输数据，最后发送 STOP 位来停止传输。
3. 当第二段中 cmd2 为 END 时，在 I2C 主机完成第二次数据传输，并检测到 I2C 主机的 `I2C_END_DETECT_INT` 中断后，配置 cmd 如第三段所示。置位 `I2C_TRANS_START`，I2C 主机发送 STOP 位停止传输。

26.7.8.2 配置示例

1. 设置 `I2C_MS_MODE` (主机) 为 1, `I2C_MS_MODE` (从机) 为 0。
2. 推荐将 `I2C_SLAVE_SCL_STRETCH_EN` (从机) 置 1, 以便 I2C 从机在需要发送数据时可以把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间, 否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照 `I2C_SLAVE_SCL_STRETCH_EN` (从机) 为 1 的情况进行。
3. 向 `I2C_CONF_UPGATE` (主机) 和 `I2C_CONF_UPGATE` (从机) 写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
<code>I2C_COMMAND0</code> (主机)	RSTART	—	—	—	—
<code>I2C_COMMAND1</code> (主机)	WRITE	0	0	1	1
<code>I2C_COMMAND2</code> (主机)	READ	0	0	1	N
<code>I2C_COMMAND3</code> (主机)	END	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址, 可选 FIFO 方式和直接访问方式。
6. 在 `I2C_SLAVE_ADDR_REG` (从机) 的 `I2C_SLAVE_ADDR` (从机) 设置 I2C 从机的地址。
7. 向 `I2C_CONF_UPGATE` (主机) 和 `I2C_CONF_UPGATE` (从机) 写 1 来同步寄存器。
8. 向 `I2C_TRANS_START` (主机) 写 1 开始主机的传输。
9. 参考章节 26.4.14, 启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的 `I2C_SLAVE_ADDR` (从机), 当 I2C 主机 WRITE 命令中的 `ack_check_en` (主机) 配置为 1 时, I2C 主机在发送完每个字节之后进行 ACK 检测。若 `ack_check_en` 配置为 0, 则不会对 ACK 检测, 会默认为匹配。
 - 匹配: 接收的 ACK 值与 WRITE 命令中的 `ack_exp` (主机) 电平一致, 传输继续。
 - 不匹配: 接收的 ACK 值与 WRITE 命令中的 `ack_exp` (主机) 电平不一致, I2C 主机产生 `I2C_NACK_INT` (主机) 中断, 停止发送数据并且产生 STOP。
11. 等待 `I2C_SLAVE_STRETCH_INT` (从机), 读取 `I2C_STRETCH_CAUSE` 的值为 0, 此时从机地址与 SDA 线上发送的地址相匹配, 且从机要发送数据。
12. 参考章节 26.4.10, 向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将 `I2C_SLAVE_SCL_STRETCH_CLR` (从机) 置 1, 释放 SCL 总线。
14. I2C 从机发送数据, I2C 主机会根据当前 READ 指令对应的 `ack_check_en` (主机) 配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机一个 READ 指令要读取的数 N 或 M 超过 I2C 从机的 TX FIFO 深度个字节, 当 I2C 从机的 TX RAM 中发送数据全部发完, 为空时产生 `I2C_SLAVE_STRETCH_INT` (从机) 中断。此时 I2C 从机会将 SCL 拉低, 软件在此期间可以继续向 I2C RAM 填充数据, 也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 `I2C_SLAVE_STRETCH_INT_CLR` (从机) 置 1 清除中断, 将 `I2C_SLAVE_SCL_STRETCH_CLR` (从机) 置 1 释放 SCL 总线。
16. 等到一次 READ 指令完成, I2C 主机执行 END 指令, `I2C_END_DETECT_INT` (主机) 中断产生后, 设置 `I2C_END_DETECT_INT_CLR` (主机) 为 1 来清除中断。

17. 更新 I2C 主机的指令寄存器，有两种设置方式：

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0 (主机)	READ	ack_value	ack_exp	1	M
I2C_COMMAND1 (主机)	END	—	—	—	—

或者

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0 (主机)	READ	0	0	1	M-1
I2C_COMMAND0 (主机)	READ	1	0	1	1
I2C_COMMAND1 (主机)	STOP	—	—	—	—

18. 向 I2C 从机的 TX RAM 写入 M 个字节要发送的数据，若 M 大于 TX FIFO 深度，重复步骤 12，可以用 FIFO 方式或直接访问方式。

19. 向 I2C_TRANS_START (主机) 位写 1 开始传输，并重复步骤 14 的流程。

20. 若最后一个指令为 STOP，则当 I2C 主机接收最后一个数据时，将 ack_value (主机) 设成 1，I2C 从机接收到 NACK 中断，停止发送。I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT (主机) 中断。

21. 若最后一个指令为 END，则重复步骤 16，并在完成后继续下面的步骤。

22. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND1 (主机)	STOP	—	—	—	—

23. 向 I2C_TRANS_START (主机) 位写 1 开始传输。

24. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT (主机) 中断。

26.8 寄存器列表

26.8.1 I2C 寄存器列表

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时序寄存器			
I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
I2C_CTR_REG	传输配置寄存器	0x0004	varies
I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
I2C_SLAVE_ADDR_REG	从机地址配置寄存器	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
I2C_SCL_STRETCH_CONF_REG	配置 I2C 从机 SCL 时钟拉伸	0x0084	varies
状态寄存器			
I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
I2C_DATA_REG	Rx FIFO 读取数据寄存器	0x001C	HRO
中断寄存器			
I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
I2C_INT_CLR_REG	中断清除位	0x0024	WT
I2C_INT_ENA_REG	中断使能位	0x0028	R/W
I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
I2C_COMDO_REG	I2C 命令寄存器 0	0x0058	varies
I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies

名称	描述	地址	访问
I2C_COMMD2_REG	I2C 命令寄存器 2	0x0060	varies
I2C_COMMD3_REG	I2C 命令寄存器 3	0x0064	varies
I2C_COMMD4_REG	I2C 命令寄存器 4	0x0068	varies
I2C_COMMD5_REG	I2C 命令寄存器 5	0x006C	varies
I2C_COMMD6_REG	I2C 命令寄存器 6	0x0070	varies
I2C_COMMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
I2C_DATE_REG	版本寄存器	0x00F8	R/W
地址寄存器			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO

26.8.2 LP_I2C 寄存器列表

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时序寄存器			
LP_I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
LP_I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
LP_I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
LP_I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
LP_I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
LP_I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
LP_I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
LP_I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
LP_I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
LP_I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
LP_I2C_CTR_REG	传输配置寄存器	0x0004	varies
LP_I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
LP_I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
LP_I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
LP_I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
状态寄存器			
LP_I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
LP_I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
LP_I2C_DATA_REG	RX FIFO 读取数据寄存器	0x001C	RO

名称	描述	地址	访问
中断寄存器			
LP_I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
LP_I2C_INT_CLR_REG	中断清除位	0x0024	WT
LP_I2C_INT_ENA_REG	中断使能位	0x0028	R/W
LP_I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
LP_I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	varies
LP_I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies
LP_I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	varies
LP_I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	varies
LP_I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	varies
LP_I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
LP_I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies
LP_I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
LP_I2C_DATE_REG	版本寄存器	0x00F8	R/W
地址寄存器			
LP_I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
LP_I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO

26.9 寄存器

26.9.1 I2C 寄存器

本小节的所有地址均为相对于 I2C 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 26.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

(reserved)																								I2C_SCL_LOW_PERIOD									
31																								9	8	0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0								Reset	

I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。
单位：I2C_SCLK 时钟周期数
(R/W)

Register 26.2. I2C_SDA_HOLD_REG (0x0030)

(reserved)																								I2C_SDA_HOLD_TIME																							
31																								9								8								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0								Reset															

I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。
单位：I2C_SCLK 时钟周期数
(R/W)

Register 26.3. I2C_SDA_SAMPLE_REG (0x0034)

(reserved)																I2C_SDA_SAMPLE_TIME																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																9																8																0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0																0</															

Reset

I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。

单位: I2C_SCLK 时钟周期数

(R/W)

Register 26.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)

(reserved)																I2C_SCL_WAIT_HIGH_PERIOD																I2C_SCL_HIGH_PERIOD																							
31																15																9								8								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0																0								Reset															

Reset

I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位: I2C_SCLK 时钟周期数

(R/W)

I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位: I2C_SCLK 时钟周期数

(R/W)

Register 26.5. I2C_SCL_START_HOLD_REG (0x0040)

(reserved)																I2C_SCL_START_HOLD_TIME																
31																	9	8									0					
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																8																Reset

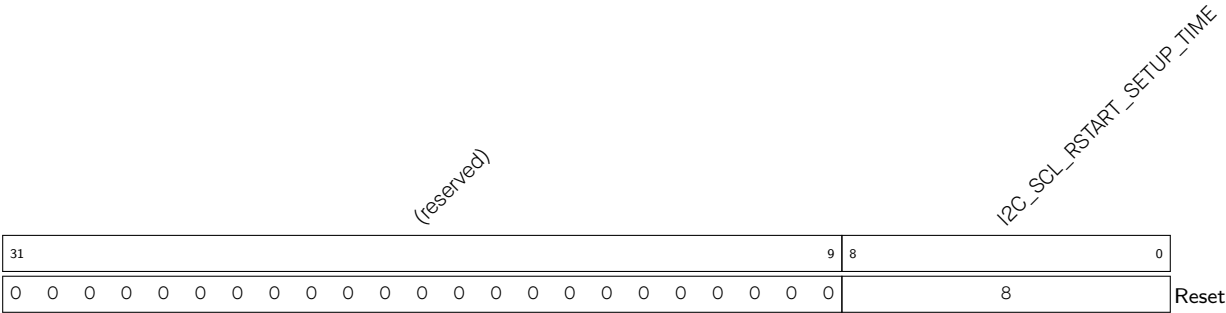
Reset

I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位: I2C_SCLK 时钟周期数

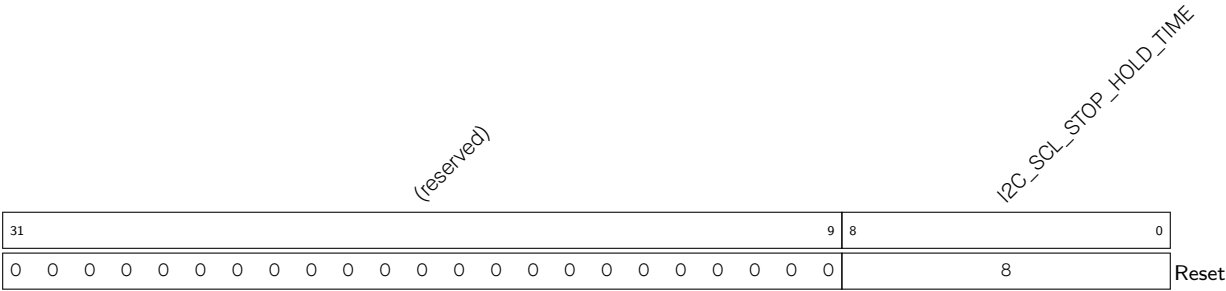
(R/W)

Register 26.6. I2C_SCL_RSTART_SETUP_REG (0x0044)



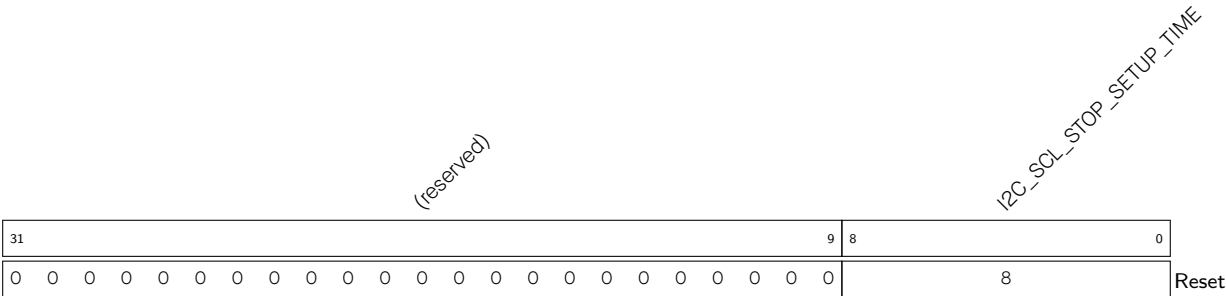
I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.7. I2C_SCL_STOP_HOLD_REG (0x0048)



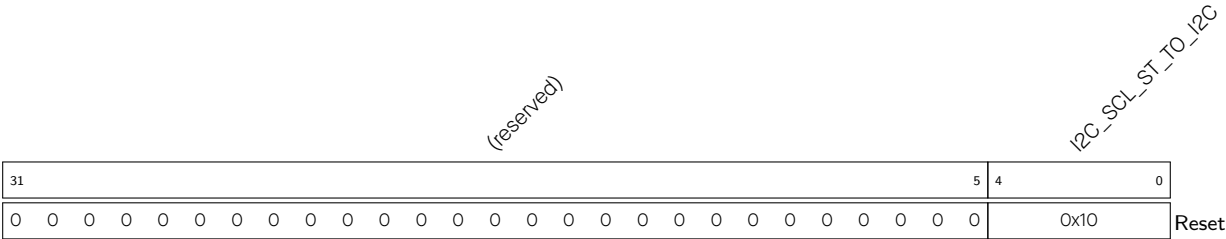
I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.8. I2C_SCL_STOP_SETUP_REG (0x004C)



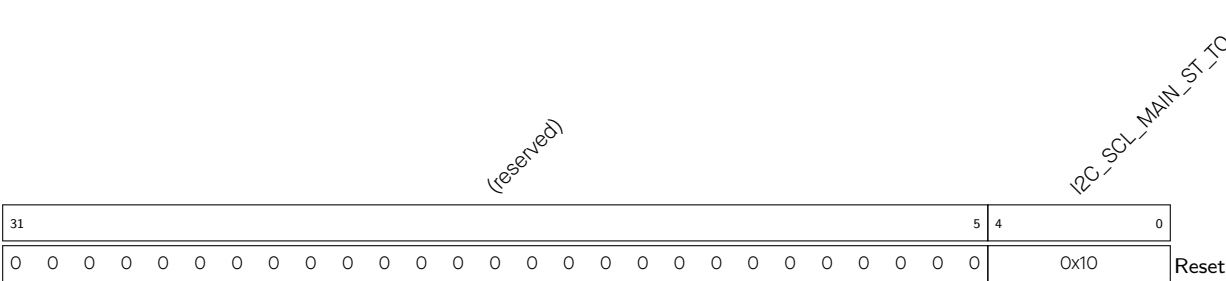
I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间，不能大于 23。实际周期是 $2^{(reg\_time\_out\_value+1)}$ 。
单位：I2C_SCLK 时钟周期数
(R/W)

Register 26.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)



I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。实际周期是 $2^{(I2C\_SCL\_MAIN\_ST\_TO\_I2C+1)}$ 。
单位：I2C_SCLK 时钟周期数
(R/W)

Register 26.11. I2C_CTR_REG (0x0004)

(reserved)																I2C_ADDR_BROADCASTING_EN I2C_ADDR_10BIT_RW_CHECK_EN I2C_SLV_TX_AUTO_START_EN I2C_CONF_UPGATE I2C_FSM_RST I2C_ARBITRATION_EN I2C_RX_LSB_FIRST I2C_TX_LSB_FIRST I2C_TRANS_START I2C_MS_MODE I2C_SAMPLE_FULL_ACK_LEVEL I2C_SCL_FORCE_OUT I2C_SDA_FORCE_OUT															
31															15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	Reset				

I2C_SDA_FORCE_OUT 配置 SDA 输出模式。

- 0: 开漏输出
- 1: 直接输出
- (R/W)

I2C_SCL_FORCE_OUT 配置 SCL 输出模式。

- 0: 开漏输出
- 1: 直接输出
- (R/W)

I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。

- 0: SCL 为高电平时采样 SDA 数据
- 1: SCL 为低电平时采样 SDA 数据
- (R/W)

I2C_RX_FULL_ACK_LEVEL 用于配置主机在接收的 RX RAM 达到阈值时需发送的 ACK 电平值。(R/W)

I2C_MS_MODE 配置该模块为 I2C 主机或从机。

- 0: 从机
- 1: 主机
- (R/W)

I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。

- 0: 无效
- 1: 开始
- (WT)

I2C_TX_LSB_FIRST 控制待发送数据的发送顺序。

- 0: 从最高有效位开始发送数据
- 1: 从最低有效位开始发送数据
- (R/W)

I2C_RX_LSB_FIRST 控制接收数据的存储顺序。

- 0: 从最高有效位开始接收数据
- 1: 从最低有效位开始接收数据
- (R/W)

见下页...

Register 26.11. I2C_CTR_REG (0x0004)

... 接上页

I2C_CLK_EN 门控寄存器的时钟信号。

0: 仅当通过软件读写寄存器时支持时钟

1: 强制寄存器时钟开启

(R/W)

I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0: 无效

1: 使能

(R/W)

I2C_FSM_RST 配置复位 SCL_FSM。

0: 无效

1: 复位

(WT)

I2C_CONF_UPGATE 配置此位实现同步。

0: 无效

1: 同步

(WT)

I2C_SLV_TX_AUTO_START_EN 配置从机自动发送数据的使能位。

0: 禁用

1: 使能

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN 配置使能 10 位寻址模式的读写标志位检查功能，检查读写标志是否符合协议。

0: 不检查

1: 检查

(R/W)

I2C_ADDR_BROADCASTING_EN 配置是否支持 7 位寻址模式的广播功能。

0: 不支持

1: 支持

(R/W)

Register 26.12. I2C_TO_REG (0x000C)

(reserved)																										I2C_TIME_OUT_EN		I2C_TIME_OUT_VALUE		
31																										6	5	4	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																										0	0x10		Reset	

I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期是 2 的 I2C_TIME_OUT_VALUE+1 次方。
单位: I2C_SCLK 时钟周期数
(R/W)

I2C_TIME_OUT_EN 配置使能超时控制。
0: 无效
1: 使能
(R/W)

Register 26.13. I2C_SLAVE_ADDR_REG (0x0010)

I2C_ADDR_10BIT_EN																(reserved)																I2C_SLAVE_ADDR															
31	30														15	14	0																														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0														Reset																

I2C_SLAVE_ADDR 配置 I2C 从机的从机地址。(R/W)

I2C_ADDR_10BIT_EN 配置在主机模式下使能从机的 10 位寻址模式。
0: 无效
1: 使能
(R/W)

Register 26.14. I2C_FIFO_CONF_REG (0x0018)

(reserved)																I2C_FIFO_PRT_EN		I2C_TX_FIFO_RST	I2C_RX_FIFO_RST	I2C_FIFO_ADDR_CFG_EN	I2C_NONFIFO_EN	I2C_TXFIFO_WM_THRHD		I2C_RXFIFO_WM_THRHD		
31																15	14	13	12	11	10	9	5		4	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																1	0	0	0	0	0x4		0xb		Reset	

I2C_RXFIFO_WM_THRHD 配置在直接访问模式下，RX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 I2C_RXFIFO_WM_THRHD[4:0] 时，I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_TXFIFO_WM_THRHD 配置在直接访问模式下，TX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 TX FIFO 计数值大于 I2C_TXFIFO_WM_THRHD[4:0] 时，I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

I2C_FIFO_ADDR_CFG_EN 配置从机是否使能双寻址模式。当该模式使能时，在 I2C 地址字节之后接收的字节代表 I2C 从机 RAM 中的偏移地址。

0: 禁用

1: 使能

(R/W)

I2C_RX_FIFO_RST 配置复位 RX FIFO。

0: 无效

1: 复位

(R/W)

I2C_TX_FIFO_RST 配置复位 TX FIFO。

0: 无效

1: 复位

(R/W)

I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0: 无效

1: 使能

(R/W)

Register 26.15. I2C_FILTER_CFG_REG (0x0050)

(reserved)										I2C_SDA_FILTER_EN		I2C_SCL_FILTER_EN		I2C_SDA_FILTER_THRES		I2C_SCL_FILTER_THRES			
31										10		9	8	7	4		3	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0										1		1	0		0		Reset		

I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。
单位：I2C_SCLK 时钟周期数
(R/W)

I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。
单位：I2C_SCLK 时钟周期数
(R/W)

I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。
0：无效
1：复位
(R/W)

I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。
0：无效
1：复位
(R/W)

Register 26.16. I2C_SCL_SP_CONF_REG (0x0080)

(reserved)																								I2C_SDA_PD_EN		I2C_SCL_PD_EN		I2C_SCL_RST_SLV_NUM		I2C_SCL_RST_SLV_EN	
31																								8	7	6	5	1		0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0	0	0		0	Reset		

I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 I2C_SCL_RST_SLV_NUM。

- 0：无效
- 1：发送 SCL 脉冲
- (R/W/SC)

I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。

- 仅在 I2C_SCL_RST_SLV_EN 为 1 时有效。
- 单位：I2C_SCLK 时钟周期数
- (R/W)

I2C_SCL_PD_EN 配置降低 I2C SCL 输出功耗。

- 0：正常功耗
- 1：不工作，降低功耗
- 仅在 I2C_SCL_FORCE_OUT 为 1 时有效。
- (R/W)

I2C_SDA_PD_EN 配置降低 I2C SDA 输出功耗。

- 0：正常功耗
- 1：不工作，降低功耗
- 仅在 I2C_SDA_FORCE_OUT 为 1 时有效。(R/W)

Register 26.17. I2C_SCL_STRETCH_CONF_REG (0x0084)

(reserved)																I2C_SLAVE_BYTE_ACK_LVL I2C_SLAVE_BYTE_ACK_CTL_EN I2C_SLAVE_SCL_STRETCH_CLR I2C_SLAVE_SCL_STRETCH_EN				I2C_STRETCH_PROTECT_NUM			
31													14	13	12	11	10	9					0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				Reset

I2C_STRETCH_PROTECT_NUM 配置 SCL 时钟拉伸释放后的保护时间，通常设置为大于 SDA 的建立时间。

单位：I2C_SCLK 时钟周期数

(R/W)

I2C_SLAVE_SCL_STRETCH_EN 配置使能从机的 SCL 时钟拉伸。I2C_SLAVE_SCL_STRETCH_EN 为 1，且出现可触发时钟拉伸的事件时，拉伸 SCL 时钟。SCL 时钟拉伸的原因可见 [I2C_STRETCH_CAUSE](#)。

0：禁用

1：使能

(R/W)

I2C_SLAVE_SCL_STRETCH_CLR 配置清除从机 SCL 时钟拉伸。

0：无效

1：清除

(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN 配置使能从机控制 ACK 电平。

0：禁用

1：使能

(R/W)

I2C_SLAVE_BYTE_ACK_LVL 从机控制 ACK 电平使能时，通过此位设置 ACK 的电平值。

0：低电平

1：高电平

(R/W)

Register 26.18. I2C_SR_REG (0x0008)

(reserved)		I2C_SCL_STATE_LAST		(reserved)		I2C_SCL_MAIN_STATE_LAST		I2C_TXFIFO_CNT		(reserved)		I2C_STRETCH_CAUSE		I2C_RXFIFO_CNT		(reserved)		I2C_SLAVE_ADDRESSED		I2C_BUS_BUSY		I2C_ARB_LOST		(reserved)		I2C_SLAVE_RW		I2C_RESP_REC	
31	30	28	27	26	24	23	18		17	16	15	14	13	8		7	6	5	4	3	2	1	0	Reset					
0	0	0	0	0	0	0	0	0	0x3	0	0	0	0	0	0	0	0	0	0	0	0	0	0					0	0

I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

- 0: ACK
- 1: NACK

(RO)

I2C_SLAVE_RW 表示在从机模式下数据传输方向。

- 0: 主机向从机写入数据
- 1: 主机读取从机数据

(RO)

I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

- 0: 仲裁未丢失
- 1: 仲裁丢失

(RO)

I2C_BUS_BUSY 表示 I2C 总线状态。

- 0: I2C 总线处于空闲状态
- 1: I2C 总线正在传输数据

(RO)

I2C_SLAVE_ADDRESSED 表示主机发送的地址是否与从机的地址相同。

仅当模块被配置为 I2C 从机时有效。

- 0: 不相同
- 1: 相同

(RO)

I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

I2C_STRETCH_CAUSE 表示从机模式下 SCL 时钟拉伸的原因。

- 0: 主机开始读取数据时拉伸 SCL 时钟
- 1: 从机模式下 I2C TX FIFO 读空时拉伸 SCL 时钟
- 2: 从机模式下 I2C RX FIFO 写满时拉伸 SCL 时钟
- 3: 无效

(RO)

I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

见下页...

Register 26.18. I2C_SR_REG (0x0008)

... 接上页

I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

- 0: 空闲
 - 1: 地址偏移
 - 2: ACK 地址
 - 3: 接收数据
 - 4: 发送数据
 - 5: 发送 ACK
 - 6: 等待 ACK
 - 7: 无效
- (RO)

I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

- 0: 空闲
 - 1: 开始
 - 2: 下降沿
 - 3: 低电平
 - 4: 上升沿
 - 5: 高电平
 - 6: 停止
 - 7: 无效
- (RO)

Register 26.19. I2C_FIFO_ST_REG (0x0014)

(reserved)				I2C_SLAVE_RW_POINT				(reserved)				I2C_TXFIFO_WADDR				I2C_TXFIFO_RADDR				I2C_RXFIFO_WADDR				I2C_RXFIFO_RADDR			
31	30	29				22	21	20	19			15	14			10	9			5	4						0
0	0					0	0			0			0			0				0				0			Reset

I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)

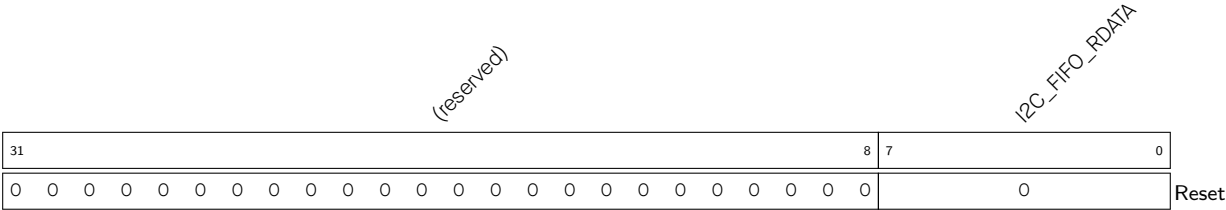
I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)

I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)

I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

I2C_SLAVE_RW_POINT 表示在 I2C 从机模式下由 I2C 主机寻址的 I2C 从机 RAM 的偏移地址。(RO)

Register 26.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 26.21. I2C_INT_RAW_REG (0x0020)

(reserved)																I2C_SLAVE_ADDR_UNMATCH_INT_RAW I2C_GENERAL_CALL_INT_RAW I2C_SLAVE_STRETCH_INT_RAW I2C_DET_START_INT_RAW I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_ST_TO_INT_RAW I2C_RXFIFO_UDF_INT_RAW I2C_TXFIFO_UDF_INT_RAW I2C_NACK_INT_RAW I2C_TRANS_START_INT_RAW I2C_TIME_OUT_INT_RAW I2C_TRANS_COMPLETE_INT_RAW I2C_ARBTRATION_LOST_INT_RAW I2C_BYTE_TRANS_DONE_INT_RAW I2C_END_DETECT_INT_RAW I2C_RXFIFO_OVF_INT_RAW I2C_RXFIFO_WM_INT_RAW																					
31																19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Reset									

I2C_RXFIFO_WM_INT_RAW I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_END_DETECT_INT_RAW I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW I2C_BYTE_TRANS_DONE_INT 的原始中断状态。(R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW I2C_ARBITRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW I2C_MST_TXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

I2C_TIME_OUT_INT_RAW I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_START_INT_RAW I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

I2C_NACK_INT_RAW I2C_NACK_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_DET_START_INT_RAW I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW I2C_GENARAL_CALL_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW I2C_SLAVE_ADDR_UNMATCH_INT 的原始中断状态。(R/SS/WTC)

Register 26.22. I2C_INT_CLR_REG (0x0024)

(reserved)																I2C_SLAVE_ADDR_UNMATCH_INT_CLR I2C_GENERAL_CALL_INT_CLR I2C_SLAVE_STRETCH_INT_CLR I2C_DET_START_INT_CLR I2C_SCL_MAIN_ST_TO_INT_CLR I2C_SCL_ST_TO_INT_CLR I2C_RXFIFO_UDF_INT_CLR I2C_NACK_INT_CLR I2C_TRANS_START_INT_CLR I2C_TIME_OUT_INT_CLR I2C_TRANS_COMPLETE_INT_CLR I2C_MST_TXFIFO_UDF_INT_CLR I2C_ARBITRATION_LOST_INT_CLR I2C_BYTE_TRANS_DONE_INT_CLR I2C_END_DETECT_INT_CLR I2C_RXFIFO_OVF_INT_CLR I2C_RXFIFO_WM_INT_CLR																	
31													19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset					

I2C_RXFIFO_WM_INT_CLR 写 1 清除 I2C_RXFIFO_WM_INT 中断。(WT)

I2C_TXFIFO_WM_INT_CLR 写 1 清除 I2C_TXFIFO_WM_INT 中断。(WT)

I2C_RXFIFO_OVF_INT_CLR 写 1 清除 I2C_RXFIFO_OVF_INT 中断。(WT)

I2C_END_DETECT_INT_CLR 写 1 清除 I2C_END_DETECT_INT 中断。(WT)

I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 I2C_BYTE_TRANS_DONE_INT 中断。(WT)

I2C_ARBITRATION_LOST_INT_CLR 写 1 清除 I2C_ARBITRATION_LOST_INT 中断。(WT)

I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 I2C_MST_TXFIFO_UDF_INT 中断。(WT)

I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 I2C_TRANS_COMPLETE_INT 中断。(WT)

I2C_TIME_OUT_INT_CLR 写 1 清除 I2C_TIME_OUT_INT 中断。(WT)

I2C_TRANS_START_INT_CLR 写 1 清除 I2C_TRANS_START_INT 中断。(WT)

I2C_NACK_INT_CLR 写 1 清除 I2C_NACK_INT 中断。(WT)

I2C_TXFIFO_OVF_INT_CLR 写 1 清除 I2C_TXFIFO_OVF_INT 中断。(WT)

I2C_RXFIFO_UDF_INT_CLR 写 1 清除 I2C_RXFIFO_UDF_INT 中断。(WT)

I2C_SCL_ST_TO_INT_CLR 写 1 清除 I2C_SCL_ST_TO_INT 中断。(WT)

I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

I2C_DET_START_INT_CLR 写 1 清除 I2C_DET_START_INT 中断。(WT)

I2C_SLAVE_STRETCH_INT_CLR 写 1 清除 I2C_SLAVE_STRETCH_INT 中断。(WT)

I2C_GENERAL_CALL_INT_CLR 写 1 清除 I2C_GENARAL_CALL_INT 中断。(WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR 写 1 清除 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(WT)

Register 26.23. I2C_INT_ENA_REG (0x0028)

(reserved)																I2C_SLAVE_ADDR_UNMATCH_INT_ENA I2C_GENERAL_CALL_INT_ENA I2C_SLAVE_STRETCH_INT_ENA I2C_DET_START_INT_ENA I2C_SCL_MAIN_ST_TO_INT_ENA I2C_SCL_ST_TO_INT_ENA I2C_RXFIFO_UDF_INT_ENA I2C_TXFIFO_UDF_INT_ENA I2C_NACK_INT_ENA I2C_TRANS_START_INT_ENA I2C_TIME_OUT_INT_ENA I2C_TRANS_COMPLETE_INT_ENA I2C_MST_TXFIFO_UDF_INT_ENA I2C_ARBITRATION_LOST_INT_ENA I2C_BYTE_TRANS_DONE_INT_ENA I2C_END_DETECT_INT_ENA I2C_RXFIFO_OVF_INT_ENA I2C_RXFIFO_WM_INT_ENA																	
31													19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

I2C_RXFIFO_WM_INT_ENA 写 1 使能 I2C_RXFIFO_WM_INT 中断。(R/W)

I2C_TXFIFO_WM_INT_ENA 写 1 使能 I2C_TXFIFO_WM_INT 中断。(R/W)

I2C_RXFIFO_OVF_INT_ENA 写 1 使能 I2C_RXFIFO_OVF_INT 中断。(R/W)

I2C_END_DETECT_INT_ENA 写 1 使能 I2C_END_DETECT_INT 中断。(R/W)

I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 I2C_BYTE_TRANS_DONE_INT 中断。(R/W)

I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 I2C_ARBITRATION_LOST_INT 中断。(R/W)

I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 I2C_MST_TXFIFO_UDF_INT 中断。(R/W)

I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 I2C_TRANS_COMPLETE_INT 中断。(R/W)

I2C_TIME_OUT_INT_ENA 写 1 使能 I2C_TIME_OUT_INT 中断。(R/W)

I2C_TRANS_START_INT_ENA 写 1 使能 I2C_TRANS_START_INT 中断。(R/W)

I2C_NACK_INT_ENA 写 1 使能 I2C_NACK_INT 中断。(R/W)

I2C_TXFIFO_OVF_INT_ENA 写 1 使能 I2C_TXFIFO_OVF_INT 中断。(R/W)

I2C_RXFIFO_UDF_INT_ENA 写 1 使能 I2C_RXFIFO_UDF_INT 中断。(R/W)

I2C_SCL_ST_TO_INT_ENA 写 1 使能 I2C_SCL_ST_TO_INT 中断。(R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

I2C_DET_START_INT_ENA 写 1 使能 I2C_DET_START_INT 中断。(R/W)

I2C_SLAVE_STRETCH_INT_ENA 写 1 使能 I2C_SLAVE_STRETCH_INT 中断。(R/W)

I2C_GENERAL_CALL_INT_ENA 写 1 使能 I2C_GENARAL_CALL_INT 中断。(R/W)

I2C_SLAVE_ADDR_UNMATCH_INT_ENA 写 1 使能 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(R/W)

Register 26.24. I2C_INT_STATUS_REG (0x002C)

[illegible]

I2C_RXFIFO_WM_INT_ST I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_WM_INT_ST I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_RXFIFO_OVF_INT_ST I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

I2C_END_DETECT_INT_ST I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

I2C_BYTE_TRANS_DONE_INT_ST I2C_BYTE_TRANS_DONE_INT 的屏蔽中断状态。(RO)

I2C_ARBITRATION_LOST_INT_ST I2C_ARBITRATION_LOST_INT 的屏蔽中断状态。(RO)

I2C_MST_TXFIFO_UDF_INT_ST I2C_MST_TXFIFO_UDF_INT 的屏蔽中断状态。(RO)

I2C_TRANS_COMPLETE_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

I2C_TIME_OUT_INT_ST I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

I2C TRANS START INT ST I2C TRANS START INT 的屏蔽中断状态。(RO)

I2C_NACK_INT_ST I2C_NACK_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_OVF_INT_ST I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

I2C RXFIFO UDF INT ST I2C RXFIFO UDF INT 的屏蔽中断状态。(RO)

I2C_SCL_ST_TO_INT_ST I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

I2C_SCL_MAIN_ST_TO_INT_ST I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

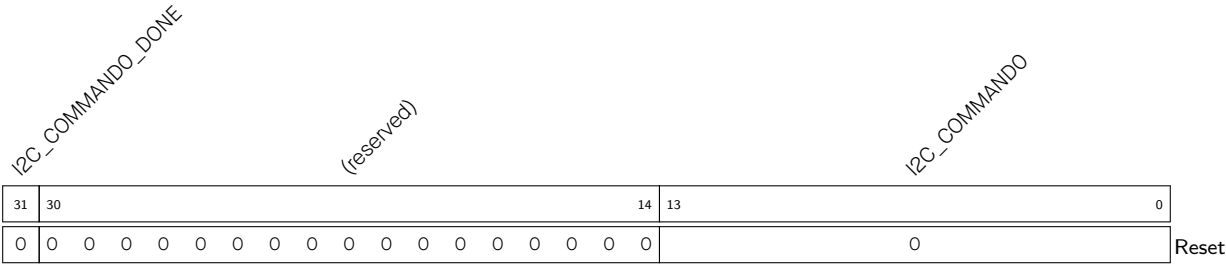
I2C_DET START INT ST I2C_DET START INT 的屏蔽中断状态。(RO)

I2C_SLAVE_STRETCH_INT_ST **I2C_SLAVE_STRETCH_INT** 的屏蔽中断状态。(RO)

I2C_GENERAL_CALL_INT_ST I2C_GENARAL_CALL_INT 的屏蔽中断状态。(RO)

I2C SLAVE ADDR UNMATCH INT ST I2C SLAVE ADDR UNMATCH INT 的屏蔽中断状态。(RO)

Register 26.25. I2C_COMDO_REG (0x0058)



I2C_COMMANDO 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

- 1: WRITE
- 2: STOP
- 3: READ
- 4: END
- 6: RSTART

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构 26.4-2 。

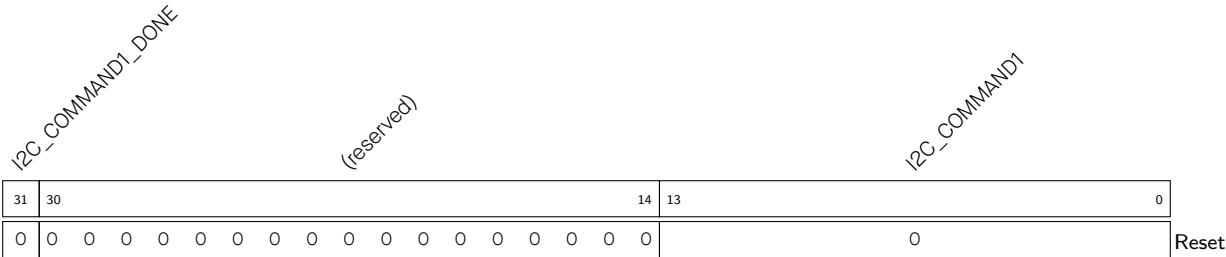
(R/W)

I2C_COMMANDO_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

- 0: 未完成
- 1: 已完成

(R/W/SS)

Register 26.26. I2C_COMD1_REG (0x005C)



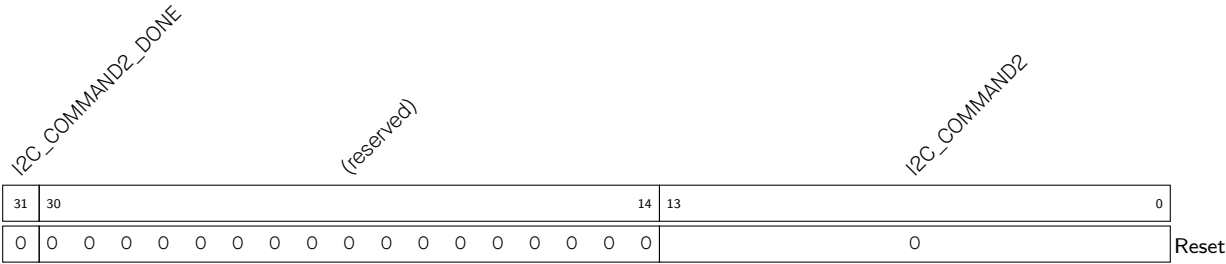
I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 I2C_COMMANDO。(R/W)

I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

- 0: 未完成
- 1: 已完成

(R/W/SS)

Register 26.27. I2C_COMD2_REG (0x0060)

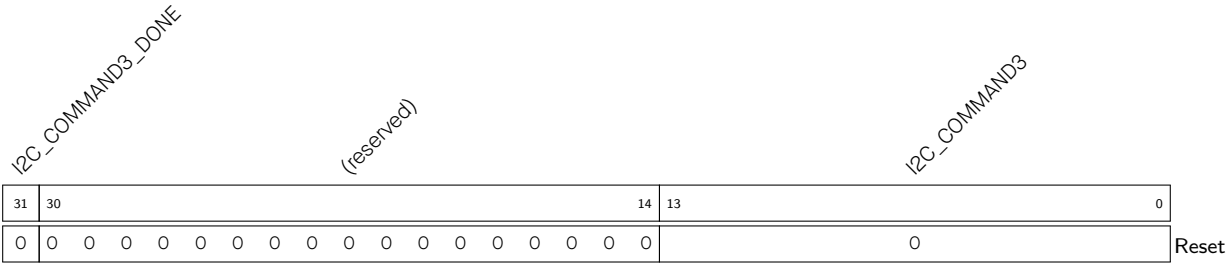


I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 I2C_COMMAND0。(R/W)

I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.28. I2C_COMD3_REG (0x0064)

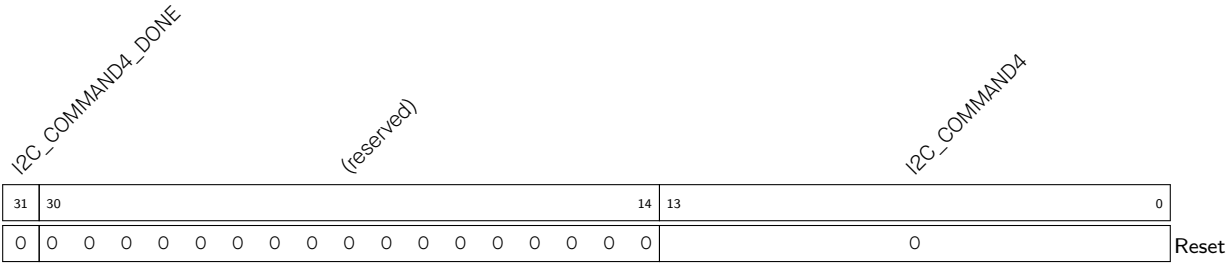


I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 I2C_COMMAND0。(R/W)

I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.29. I2C_COMD4_REG (0x0068)

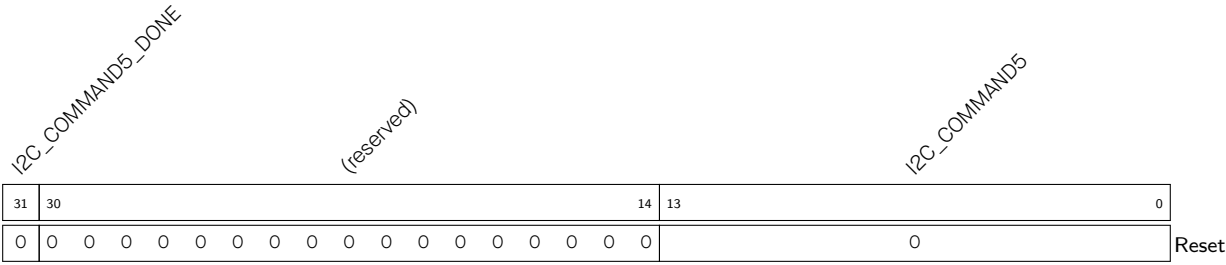


I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 [I2C_COMMAND0](#)。(R/W)

I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

- 0: 未完成
 - 1: 已完成
- (R/W/SS)

Register 26.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 [I2C_COMMAND0](#)。(R/W)

I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

- 0: 未完成
 - 1: 已完成
- (R/W/SS)

Register 26.31. I2C_COMMD6_REG (0x0070)

31	30		14	13		0
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			0	0

Reset

I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 **I2C_COMMAND0**。(R/W)

I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 26.32. I2C_CMD7_REG (0x0074)

Diagram illustrating the structure of the I2C_COMMAND7 register:

- I2C_COMMAND7_DONE** (bits 31-30): 2 bits
- (reserved)** (bits 29-14): 16 bits
- I2C_COMMAND7** (bits 13-0): 14 bits

Bit values shown below the register map:

- Bits 31-14: 0
- Bit 0: 0

I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 [I2C_COMMAND0](#)。(R/W)

I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 26.33. I2C_DATE_REG (0x00F8)

Diagram illustrating the I2C address field structure:

- The address field is represented by a horizontal bar.
- The address value is `0x2401040`.
- The address is associated with the `Reset` signal.

I2C_DATE 版本控制寄存器。(R/W)

Register 26.34. I2C_TXFIFO_START_ADDR_REG (0x0100)

Diagram of the I2C_TXFIFO_START_ADDR register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. The label "I2C_TXFIFO_START_ADDR" is positioned diagonally above the register box. A "Reset" label is located at the bottom right corner of the diagram.

I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

Register 26.35. I2C_RXFIFO_START_ADDR_REG (0x0180)

Diagram illustrating the structure of the I2C_RXFIFO_START_ADDR register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. Bit 0 is labeled "Reset".

I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

26.9.2 LP_I2C 寄存器

Register 26.36. LP_I2C_SCL_LOW_PERIOD_REG (0x0000)

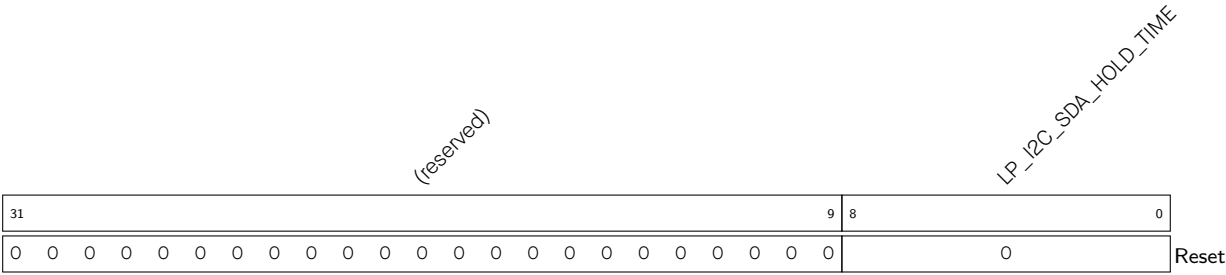
(reserved)																								LP_I2C_SCL_LOW_PERIOD																							
31																								9								8								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0								Reset															

LP_I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。

单位: I2C_SCLK 时钟周期数

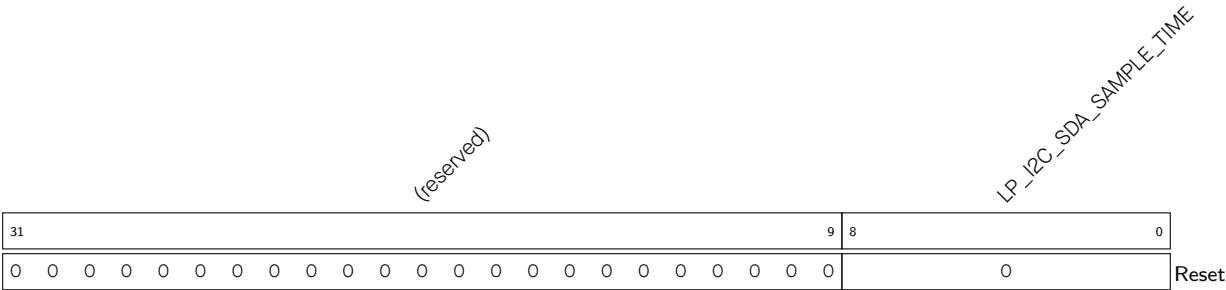
(R/W)

Register 26.37. LP_I2C_SDA_HOLD_REG (0x0030)



LP_I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.38. LP_I2C_SDA_SAMPLE_REG (0x0034)



LP_I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.39. LP_I2C_SCL_HIGH_PERIOD_REG (0x0038)

(reserved)																LP_I2C_SCL_WAIT_HIGH_PERIOD								LP_I2C_SCL_HIGH_PERIOD								Reset
31																15								7								0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0								0								

LP_I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位: I2C_SCLK 时钟周期数
(R/W)

LP_I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.40. LP_I2C_SCL_START_HOLD_REG (0x0040)

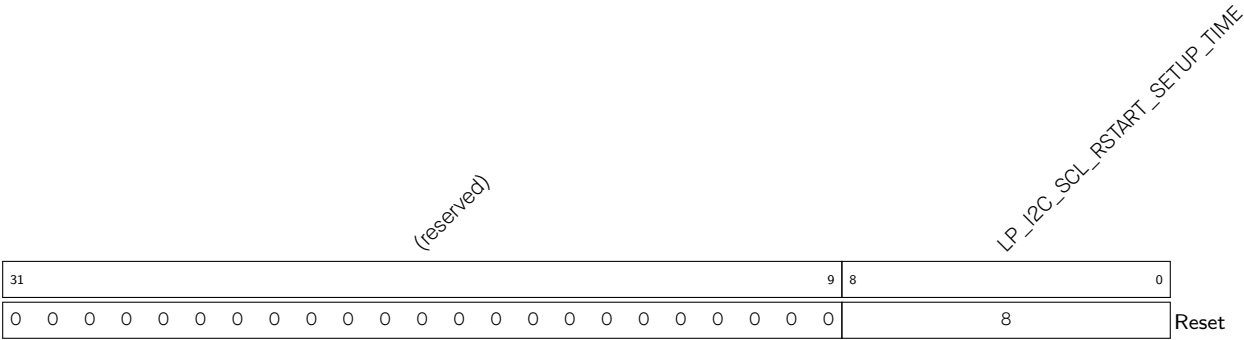
Register 0x00: LP_I2C_SCL_START_HOLD_TIME

31	9	8	0
(reserved)		LP_I2C_SCL_START_HOLD_TIME	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
		8	
Reset			

LP_I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.41. LP_I2C_SCL_RSTART_SETUP_REG (0x0044)

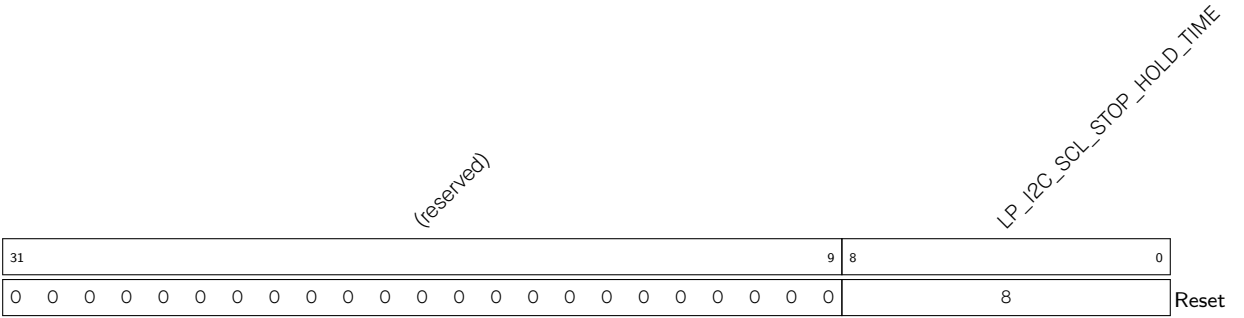


LP_I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。

单位: I2C_SCLK 时钟周期数

(R/W)

Register 26.42. LP_I2C_SCL_STOP_HOLD_REG (0x0048)

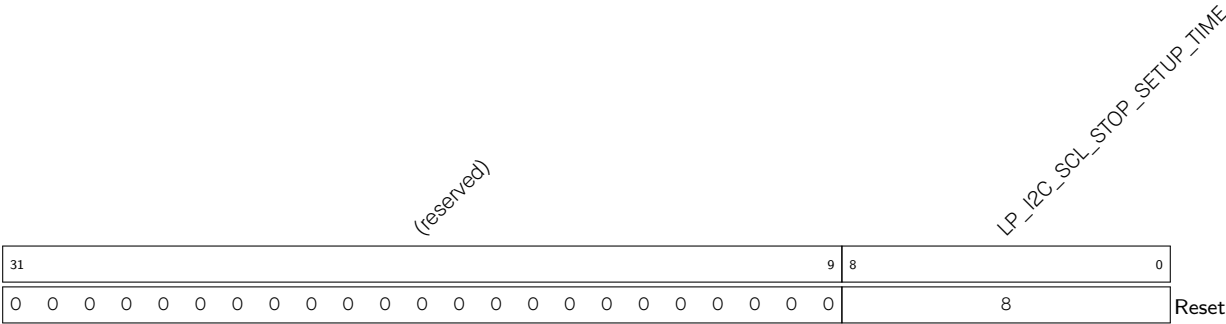


LP_I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。

单位: I2C_SCLK 时钟周期数

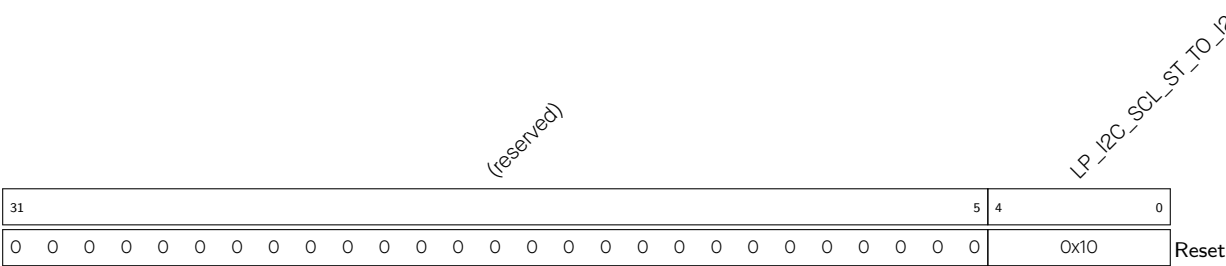
(R/W)

Register 26.43. LP_I2C_SCL_STOP_SETUP_REG (0x004C)



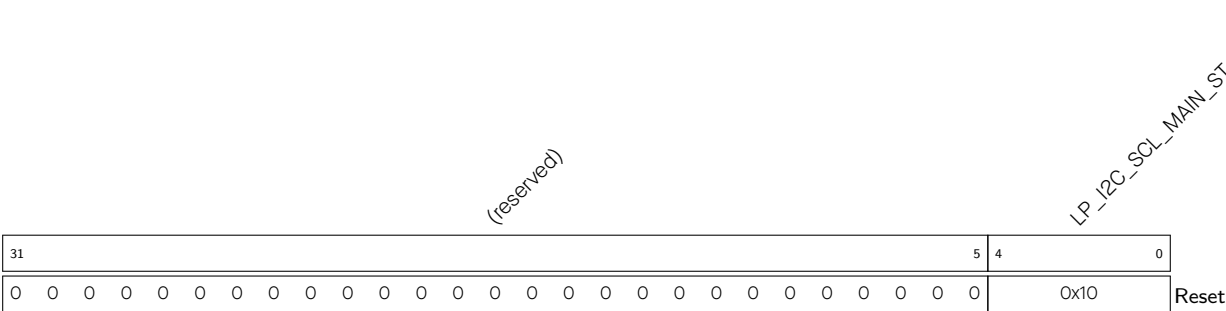
LP_I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.44. LP_I2C_SCL_ST_TIME_OUT_REG (0x0078)



LP_I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间，不能大于 23。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.45. LP_I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)



LP_I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。
单位: I2C_SCLK 时钟周期数
(R/W)

Register 26.46. LP_I2C_CTR_REG (0x0004)

(reserved)												LP_I2C_CONF_UPGATE LP_I2C_FSM_RST LP_I2C_ARBITRATION_EN LP_I2C_CLK_EN LP_I2C_RX_LSB_FIRST LP_I2C_TX_LSB_FIRST (reserved) LP_I2C_TRANS_START LP_I2C_RX_FULL_ACK_LEVEL LP_I2C_SAMPLE_SCL_LEVEL (reserved)												
31												12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	Reset	

- LP_I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。
0: SCL 为高电平时采样 SDA 数据
1: SCL 为低电平时采样 SDA 数据
(R/W)
- LP_I2C_RX_FULL_ACK_LEVEL 用于配置主机在 rx_fifo_cnt 达到阈值时需发送的 ACK 电平值。(R/W)
- LP_I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。
0: 无效
1: 开始
(WT)
- LP_I2C_TX_LSB_FIRST 控制待发送数据的发送顺序。
0: 从最高有效位开始发送数据
1: 从最低有效位开始发送数据
(R/W)
- LP_I2C_RX_LSB_FIRST 控制接收数据的存储顺序。
0: 从最高有效位开始接收数据
1: 从最低有效位开始接收数据
(R/W)
- LP_I2C_CLK_EN 门控寄存器的时钟信号。
0: 仅当通过软件读写寄存器时支持时钟
1: 强制寄存器时钟开启。
(R/W)

Continued on the next page...

Register 26.46. I2C_CTR_REG (0x0004)

Continued from the previous page...

LP_I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0: 无效

1: 使能

(R/W)

LP_I2C_FSM_RST 配置复位 SCL_FSM。

0: 无效

1: 复位

(WT)

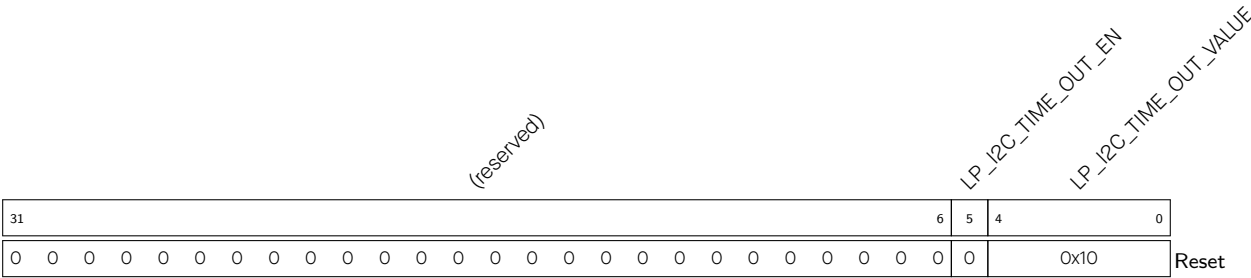
LP_I2C_CONF_UPGATE 配置此位实现同步。

0: 无效

1: 同步

(WT)

Register 26.47. LP_I2C_TO_REG (0x000C)



LP_I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期是 $2^{(LP\_I2C\_TIME\_OUT\_VALUE+1)}$ 。

单位: I2C_SCLK 时钟周期数

(R/W)

LP_I2C_TIME_OUT_EN 配置使能超时控制。

0: 无效

1: 使能

(R/W)

Register 26.48. LP_I2C_FIFO_CONF_REG (0x0018)

(reserved)																LP_I2C_FIFO_PRT_EN				LP_I2C_TX_FIFO_RST				LP_I2C_RX_FIFO_RST				(reserved)				LP_I2C_NONFIFO_EN				(reserved)				LP_I2C_TXFIFO_WM_THRHD				(reserved)				LP_I2C_RXFIFO_WM_THRHD			
31																15	14	13	12	11	10	9	8	5				4	3	0																					
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																1	0	0	0	0	0	0x2				0	0x6				Reset																				

LP_I2C_RXFIFO_WM_THRHD 配置在直接访问模式下, RX FIFO 的水标阈值。当 LP_I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 LP_I2C_RXFIFO_WM_THRHD[4:0], LP_I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

LP_I2C_TXFIFO_WM_THRHD 配置在直接访问模式下, TX FIFO 的水标阈值。当 LP_I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 LP_I2C_TXFIFO_WM_THRHD[4:0] 时, LP_I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

LP_I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

LP_I2C_RX_FIFO_RST 配置复位 RX FIFO。

0: 无效

1: 复位

(R/W)

LP_I2C_TX_FIFO_RST 配置复位 TX FIFO。

0: 无效

1: 复位

(R/W)

LP_I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0: 无效

1: 使能

(R/W)

Register 26.49. LP_I2C_FILTER_CFG_REG (0x0050)

(reserved)																		LP_I2C_SDA_FILTER_EN		LP_I2C_SCL_FILTER_EN		LP_I2C_SDA_FILTER_THRES		LP_I2C_SCL_FILTER_THRES		
31																		10	9	8	7	4		3	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0		0		Reset	

LP_I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。
单位：I2C_SCLK 时钟周期数
(R/W)

LP_I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。
单位：I2C_SCLK 时钟周期数
(R/W)

LP_I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。
0：无效
1：复位
(R/W)

LP_I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。
0：无效
1：复位
(R/W)

Register 26.50. LP_I2C_SCL_SP_CONF_REG (0x0080)

(reserved)																								LP_I2C_SDA_PD_EN		LP_I2C_SCL_PD_EN		LP_I2C_SCL_RST_SIV_NUM		LP_I2C_SCL_RST_SIV_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																								8	7	6	5	1		0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

LP_I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 [LP_I2C_SCL_RST_SLV_NUM](#). (R/W/SC)

LP_I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。
仅在 [LP_I2C_SCL_RST_SLV_EN](#) 为 1 时有效。
单位：I2C_SCLK 时钟周期数
(R/W)

LP_I2C_SCL_PD_EN 配置降低 I2C SCL 输出功耗。
0：正常功耗
1：不工作，降低功耗
仅在 LP_I2C_SCL_FORCE_OUT 为 1 时有效。
(R/W)

LP_I2C_SDA_PD_EN 配置降低 I2C SDA 输出功耗。
0：正常功耗
1：不工作，降低功耗
仅在 LP_I2C_SDA_FORCE_OUT 为 1 时有效。
(R/W) deleted these two registers based on JLW's request.

Register 26.51. LP_I2C_SR_REG (0x0008)

(reserved)		LP_I2C_SCL_STATE_LAST		(reserved)		LP_I2C_SCL_MAIN_STATE_LAST		(reserved)		LP_I2C_TXFIFO_CNT		(reserved)		LP_I2C_RXFIFO_CNT		(reserved)		LP_I2C_BUS_BUSY		LP_I2C_ARB_LOST		(reserved)		LP_I2C_RESP_REC	
31	30	28	27	26	24	23	22	18	17	13	12	8	7	5	4	3	2	1	0						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

- 0: ACK
- 1: NACK

(RO)

LP_I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

- 0: 仲裁未丢失
- 1: 仲裁丢失

(RO)

LP_I2C_BUS_BUSY 表示 I2C 总线状态。

- 0: I2C 总线处于空闲状态
- 1: I2C 总线正在传输数据

(RO)

LP_I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

LP_I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

LP_I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

- 0: 空闲
- 1: 地址偏移
- 2: ACK 地址
- 3: 接收数据
- 4: 发送数据
- 5: 发送 ACK
- 6: 等待 ACK

(RO)

LP_I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

- 0: 空闲
- 1: 开始
- 2: 下降沿
- 3: 低电平
- 4: 上升沿
- 5: 高电平
- 6: 停止

(RO)

Register 26.52. LP_I2C_FIFO_ST_REG (0x0014)

(reserved)																LP_I2C_TXFIFO_WADDR				(reserved)				LP_I2C_TXFIFO_RADDR				(reserved)				LP_I2C_RXFIFO_WADDR				(reserved)				LP_I2C_RXFIFO_RADDR			
31																		19	18			15	14	13			10	9	8			5	4	3			0						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset										

- LP_I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)
- LP_I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)
- LP_I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)
- LP_I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

Register 26.53. LP_I2C_DATA_REG (0x001C)

(reserved)																								LP_I2C_FIFO_RDATA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
31																								8	7	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- LP_I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 26.54. LP_I2C_INT_RAW_REG (0x0020)

(reserved)																LP_I2C_DET_START_INT_RAW LP_I2C_SCL_MAIN_ST_TO_INT_RAW LP_I2C_SCL_ST_TO_INT_RAW LP_I2C_RXFIFO_UDF_INT_RAW LP_I2C_TXFIFO_UDF_INT_RAW LP_I2C_NACK_INT_RAW LP_I2C_TRANS_START_INT_RAW LP_I2C_TIME_OUT_INT_RAW LP_I2C_TRANS_COMPLETE_INT_RAW LP_I2C_ARBTRATION_LOST_INT_RAW LP_I2C_BYTE_TRANS_DONE_INT_RAW LP_I2C_END_DETECT_INT_RAW LP_I2C_RXFIFO_OVF_INT_RAW LP_I2C_RXFIFO_WM_INT_RAW																																	
31																16																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0																0																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	Reset

LP_I2C_RXFIFO_WM_INT_RAW LP_I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TXFIFO_WM_INT_RAW LP_I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_RXFIFO_OVF_INT_RAW LP_I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_END_DETECT_INT_RAW LP_I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_BYTE_TRANS_DONE_INT_RAW LP_I2C_BYTE_TRANS_DONE_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_ARBTRATION_LOST_INT_RAW LP_I2C_ARBTRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_MST_TXFIFO_UDF_INT_RAW LP_I2C_MST_TXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TRANS_COMPLETE_INT_RAW LP_I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TIME_OUT_INT_RAW LP_I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TRANS_START_INT_RAW LP_I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_NACK_INT_RAW LP_I2C_NACK_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_TXFIFO_OVF_INT_RAW LP_I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_RXFIFO_UDF_INT_RAW LP_I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_SCL_ST_TO_INT_RAW LP_I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_SCL_MAIN_ST_TO_INT_RAW LP_I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

LP_I2C_DET_START_INT_RAW LP_I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

Register 26.55. LP_I2C_INT_CLR_REG (0x0024)

(reserved)																LP_I2C_DET_START_INT_CLR LP_I2C_SCL_MAIN_ST_TO_INT_CLR LP_I2C_SCL_ST_TO_INT_CLR LP_I2C_RXFIFO_UDF_INT_CLR LP_I2C_TXFIFO_UDF_INT_CLR LP_I2C_NACK_INT_CLR LP_I2C_TRANS_START_INT_CLR LP_I2C_TIME_OUT_INT_CLR LP_I2C_TRANS_COMPLETE_INT_CLR LP_I2C_ARBTRATION_LOST_INT_CLR LP_I2C_BYTE_TRANS_DONE_INT_CLR LP_I2C_END_DETECT_INT_CLR LP_I2C_RXFIFO_OVF_INT_CLR LP_I2C_RXFIFO_WM_INT_CLR																
31																16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

LP_I2C_RXFIFO_WM_INT_CLR 写 1 清除 LP_I2C_RXFIFO_WM_INT 中断。(WT)

LP_I2C_TXFIFO_WM_INT_CLR 写 1 清除 LP_I2C_TXFIFO_WM_INT 中断。(WT)

LP_I2C_RXFIFO_OVF_INT_CLR 写 1 清除 LP_I2C_RXFIFO_OVF_INT 中断。(WT)

LP_I2C_END_DETECT_INT_CLR 写 1 清除 LP_I2C_END_DETECT_INT 中断。(WT)

LP_I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 LP_I2C_BYTE_TRANS_DONE_INT 中断。(WT)

LP_I2C_ARBTRATION_LOST_INT_CLR 写 1 清除 LP_I2C_ARBTRATION_LOST_INT 中断。(WT)

LP_I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 LP_I2C_MST_TXFIFO_UDF_INT 中断。(WT)

LP_I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 LP_I2C_TRANS_COMPLETE_INT 中断。(WT)

LP_I2C_TIME_OUT_INT_CLR 写 1 清除 LP_I2C_TIME_OUT_INT 中断。(WT)

LP_I2C_TRANS_START_INT_CLR 写 1 清除 LP_I2C_TRANS_START_INT 中断。(WT)

LP_I2C_NACK_INT_CLR 写 1 清除 LP_I2C_NACK_INT 中断。(WT)

LP_I2C_TXFIFO_OVF_INT_CLR 写 1 清除 LP_I2C_TXFIFO_OVF_INT 中断。(WT)

LP_I2C_RXFIFO_UDF_INT_CLR 写 1 清除 LP_I2C_RXFIFO_UDF_INT 中断。(WT)

LP_I2C_SCL_ST_TO_INT_CLR 写 1 清除 LP_I2C_SCL_ST_TO_INT 中断。(WT)

LP_I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 LP_I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

LP_I2C_DET_START_INT_CLR 写 1 清除 LP_I2C_DET_START_INT 中断。(WT)

Register 26.56. LP_I2C_INT_ENA_REG (0x0028)

(reserved)																LP_I2C_DET_START_INT_ENA LP_I2C_SCL_MAIN_ST_TO_INT_ENA LP_I2C_SCL_ST_TO_INT_ENA LP_I2C_RXFIFO_UDF_INT_ENA LP_I2C_TXFIFO_UDF_INT_ENA LP_I2C_NACK_INT_ENA LP_I2C_TRANS_START_INT_ENA LP_I2C_TIME_OUT_INT_ENA LP_I2C_TRANS_COMPLETE_INT_ENA LP_I2C_ARBTRATION_LOST_INT_ENA LP_I2C_BYTE_TRANS_DONE_INT_ENA LP_I2C_END_DETECT_INT_ENA LP_I2C_RXFIFO_OVF_INT_ENA LP_I2C_TXFIFO_OVF_INT_ENA LP_I2C_RXFIFO_WM_INT_ENA LP_I2C_TXFIFO_WM_INT_ENA																																	
31																16																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0																0																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_I2C_RXFIFO_WM_INT_ENA 写 1 使能 LP_I2C_RXFIFO_WM_INT 中断。(R/W)

LP_I2C_TXFIFO_WM_INT_ENA 写 1 使能 LP_I2C_TXFIFO_WM_INT 中断。(R/W)

LP_I2C_RXFIFO_OVF_INT_ENA 写 1 使能 LP_I2C_RXFIFO_OVF_INT 中断。(R/W)

LP_I2C_END_DETECT_INT_ENA 写 1 使能 LP_I2C_END_DETECT_INT 中断。(R/W)

LP_I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 LP_I2C_BYTE_TRANS_DONE_INT 中断。(R/W)

LP_I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 LP_I2C_ARBITRATION_LOST_INT 中断。(R/W)

LP_I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 LP_I2C_MST_TXFIFO_UDF_INT 中断。(R/W)

LP_I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 LP_I2C_TRANS_COMPLETE_INT 中断。(R/W)

LP_I2C_TIME_OUT_INT_ENA 写 1 使能 LP_I2C_TIME_OUT_INT 中断。(R/W)

LP_I2C_TRANS_START_INT_ENA 写 1 使能 LP_I2C_TRANS_START_INT 中断。(R/W)

LP_I2C_NACK_INT_ENA 写 1 使能 LP_I2C_NACK_INT 中断。(R/W)

LP_I2C_TXFIFO_OVF_INT_ENA 写 1 使能 LP_I2C_TXFIFO_OVF_INT 中断。(R/W)

LP_I2C_RXFIFO_UDF_INT_ENA 写 1 使能 LP_I2C_RXFIFO_UDF_INT 中断。(R/W)

LP_I2C_SCL_ST_TO_INT_ENA 写 1 使能 LP_I2C_SCL_ST_TO_INT 中断。(R/W)

LP_I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 LP_I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

LP_I2C_DET_START_INT_ENA 写 1 使能 LP_I2C_DET_START_INT 中断。(R/W)

Register 26.57. LP_I2C_INT_STATUS_REG (0x002C)

(reserved)																LP_I2C_DET_START_INT_ST LP_I2C_SCL_MAIN_ST_TO_INT_ST LP_I2C_SCL_ST_TO_INT_ST LP_I2C_RXFIFO_UDF_INT_ST LP_I2C_TXFIFO_UDF_INT_ST LP_I2C_NACK_INT_ST LP_I2C_TRANS_START_INT_ST LP_I2C_TIME_OUT_INT_ST LP_I2C_TRANS_COMPLETE_INT_ST LP_I2C_ARBTRATION_LOST_INT_ST LP_I2C_BYTE_TRANS_DONE_INT_ST LP_I2C_END_DETECT_INT_ST LP_I2C_RXFIFO_OVF_INT_ST LP_I2C_RXFIFO_WM_INT_ST																																	
31																16																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0																0																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LP_I2C_RXFIFO_WM_INT_ST LP_I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

LP_I2C_TXFIFO_WM_INT_ST LP_I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

LP_I2C_RXFIFO_OVF_INT_ST LP_I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

LP_I2C_END_DETECT_INT_ST LP_I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

LP_I2C_BYTE_TRANS_DONE_INT_ST LP_I2C_BYTE_TRANS_DONE_INT 的屏蔽中断状态。(RO)

LP_I2C_ARBTRATION_LOST_INT_ST LP_I2C_ARBTRATION_LOST_INT 的屏蔽中断状态。(RO)

LP_I2C_MST_TXFIFO_UDF_INT_ST LP_I2C_MST_TXFIFO_UDF_INT 的屏蔽中断状态。(RO)

LP_I2C_TRANS_COMPLETE_INT_ST LP_I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

LP_I2C_TIME_OUT_INT_ST LP_I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

LP_I2C_TRANS_START_INT_ST LP_I2C_TRANS_START_INT 的屏蔽中断状态。(RO)

LP_I2C_NACK_INT_ST LP_I2C_NACK_INT 的屏蔽中断状态。(RO)

LP_I2C_TXFIFO_OVF_INT_ST LP_I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

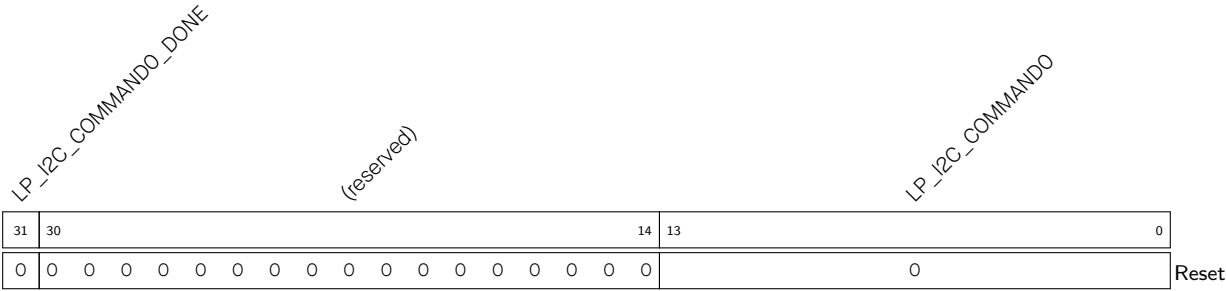
LP_I2C_RXFIFO_UDF_INT_ST LP_I2C_RXFIFO_UDF_INT 的屏蔽中断状态。(RO)

LP_I2C_SCL_ST_TO_INT_ST LP_I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

LP_I2C_SCL_MAIN_ST_TO_INT_ST LP_I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

LP_I2C_DET_START_INT_ST LP_I2C_DET_START_INT 的屏蔽中断状态。(RO)

Register 26.58. LP_I2C_COMDO_REG (0x0058)



LP_I2C_COMMAND0 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

- 1: WRITE
- 2: STOP
- 3: READ
- 4: END
- 6: RSTART

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构 26.4-2。

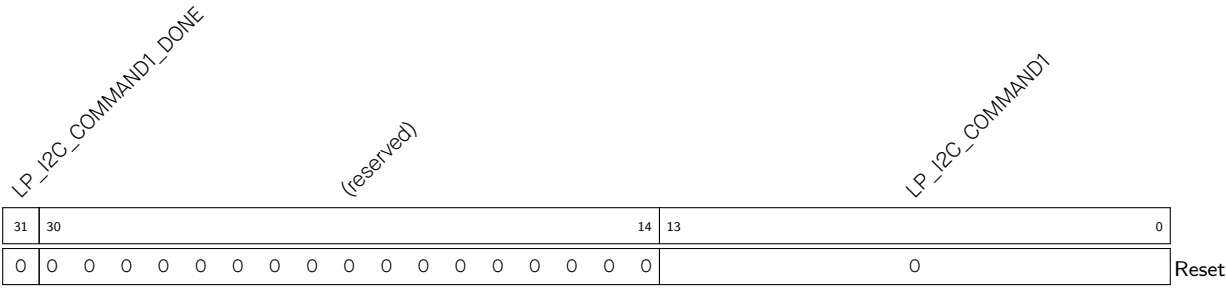
(R/W)

LP_I2C_COMMAND0_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

- 0: 未完成
- 1: 已完成

(R/W/SS)

Register 26.59. LP_I2C_COMD1_REG (0x005C)



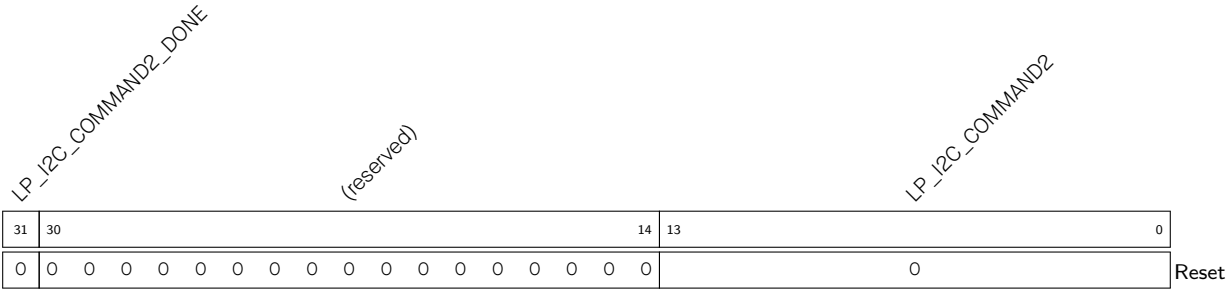
LP_I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

LP_I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

- 0: 未完成
- 1: 已完成

(R/W/SS)

Register 26.60. LP_I2C_COMD2_REG (0x0060)

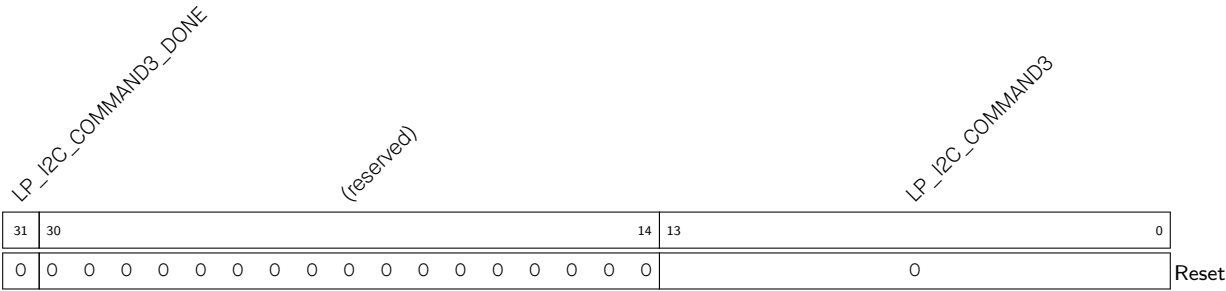


LP_I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 [LP_I2C_COMMAND0](#)。(R/W)

LP_I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.61. LP_I2C_COMD3_REG (0x0064)

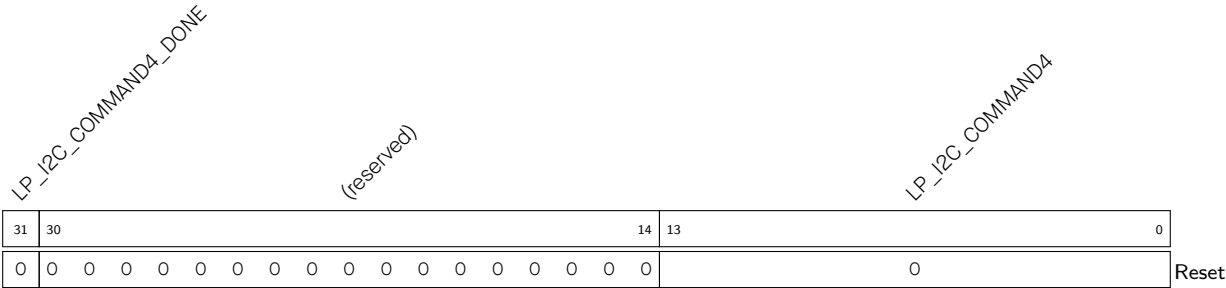


LP_I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 [LP_I2C_COMMAND0](#)。(R/W)

LP_I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.62. LP_I2C_COMD4_REG (0x0068)

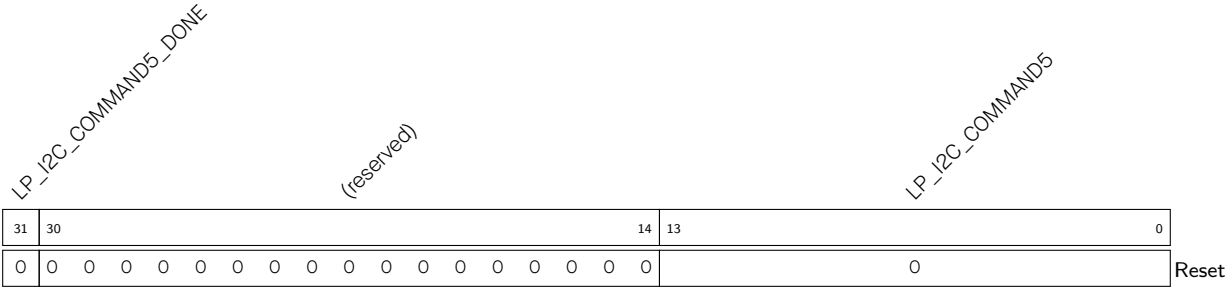


LP_I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

LP_I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.63. LP_I2C_COMD5_REG (0x006C)

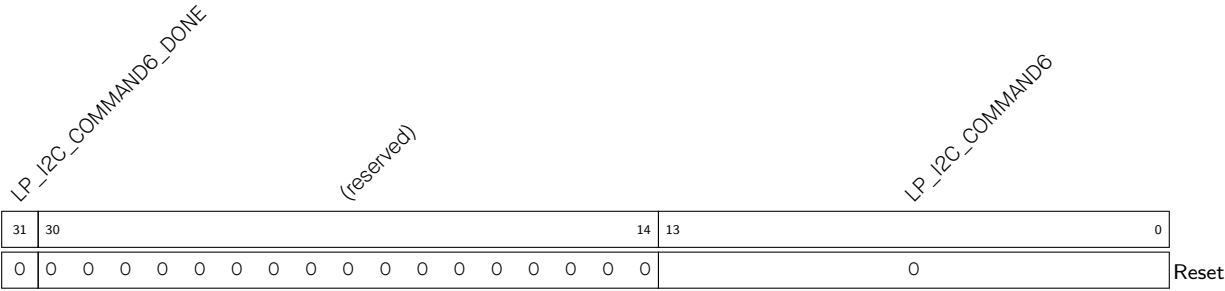


LP_I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

LP_I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.64. LP_I2C_COMD6_REG (0x0070)

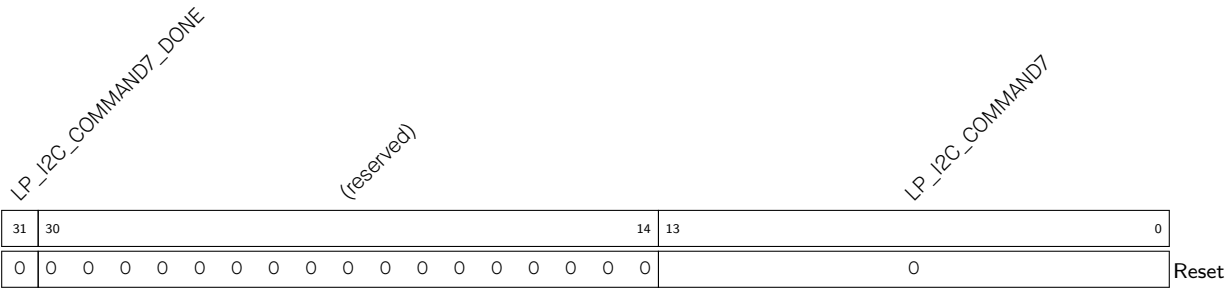


LP_I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

LP_I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.65. LP_I2C_COMD7_REG (0x0074)

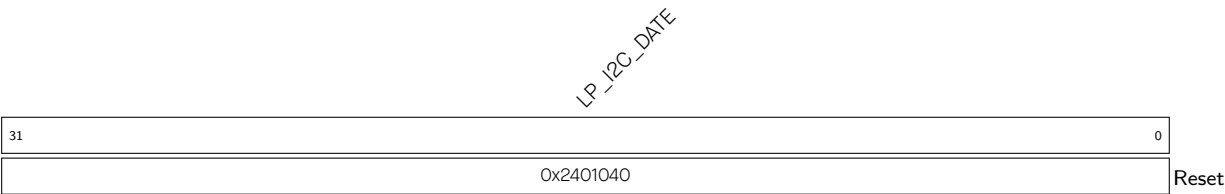


LP_I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 LP_I2C_COMMAND0。(R/W)

LP_I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

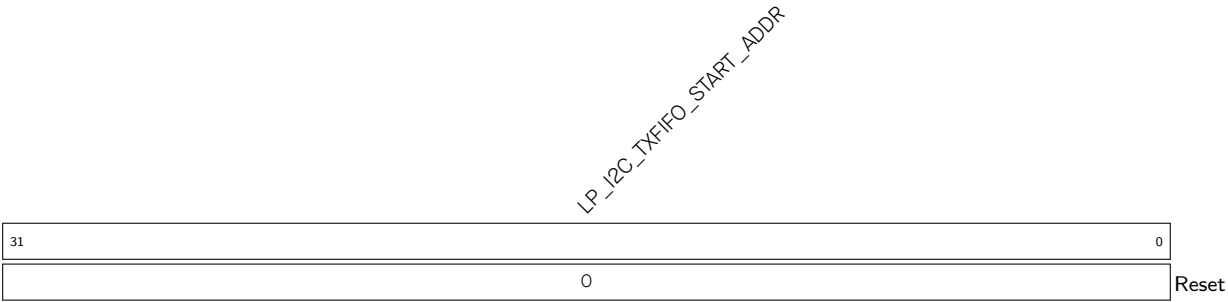
- 0: 未完成
- 1: 已完成
- (R/W/SS)

Register 26.66. LP_I2C_DATE_REG (0x00F8)



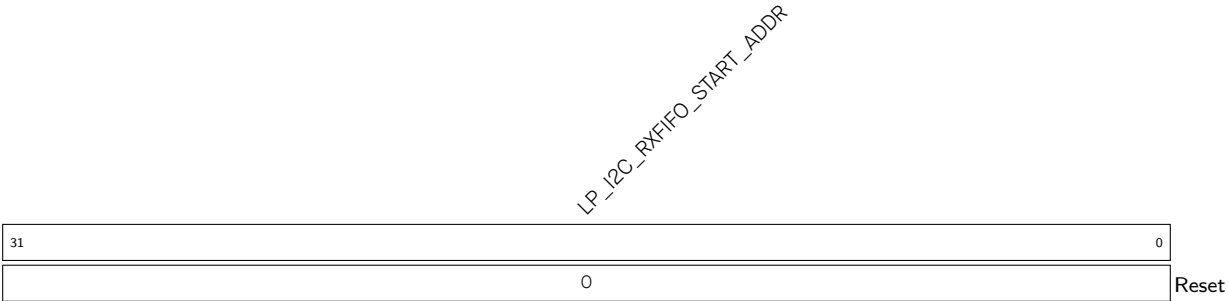
LP_I2C_DATE 版本控制寄存器。(R/W)

Register 26.67. LP_I2C_TXFIFO_START_ADDR_REG (0x0100)



LP_I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

Register 26.68. LP_I2C_RXFIFO_START_ADDR_REG (0x0180)



LP_I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

第 27 章

I2S 控制器 (I2S)

27.1 概述

ESP32-C5 内置一个 I2S 接口，为多媒体应用，尤其是为数字音频应用提供了灵活的数据通信接口。

I2S 标准总线定义了三种信号——串行时钟信号 BCK、字选择信号 WS 和串行数据信号 SD。一个基础的 I2S 数据总线包含一个主机和一个从机。主机和从机的角色在通信过程中保持不变。ESP32-C5 的 I2S 模块包含独立的 TX 模块和 RX 模块，能够保证优良的通信性能。

27.2 术语

为了更好地说明 I2S 的功能，本章使用了以下术语。

主机模式	I2S 作为主机驱动 BCK/WS，向从机发送或从其接收数据。
从机模式	I2S 作为从机由 BCK/WS 驱动，从主机接收或向其发送数据。
全双工	主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。
半双工	主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。
A 律和 μ 律	A 律和 μ 律是数字脉冲编码调制 (PCM) 非均匀量化中的压缩/解压缩算法，可以有效地改善信号的量化信噪比。
TDM RX 模式	即 TDM 输入模式，该模式利用时分复用方式接收脉冲编码调制 (PCM) 数据。信号线包括 BCK、WS 和 DATA。可以接收最多 16 个通道的数据。用户可配置数据格式，支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。
PDM RX 模式	即 PDM 输入模式，该模式接收脉冲密度调制 (PDM) 数据。信号线包括 WS 和 DATA。用户可配置数据格式，支持 PDM 标准格式等。
TDM TX 模式	即 TDM 输出模式，该模式利用时分复用方式发送脉冲编码调制 (PCM) 数据。信号线包括 BCK、WS 和 DATA，可以发送最多 16 个通道的数据。用户可配置数据格式，支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。
PDM TX 模式	即 PDM 输出模式，该模式发送脉冲密度调制 (PDM) 数据。信号线包括 WS 和 DATA。用户可配置数据格式，支持 PDM 标准格式等。
PCM-to-PDM 数据格式转换器	将 PCM 数据转换为 PDM 数据的格式转换器（后文简称为 PCM-to-PDM 转换器），可以在 PDM TX 主机模式下启用。启用后，I2S 通过 GDMA 从存储器中取得脉冲编码调制 (PCM) 数据，将其转换为脉冲密度调制 (PDM) 数据并发送。

27.3 特性

I2S 模块具有以下特性：

- 支持主机模式和从机模式
- 支持全双工和半双工通信
- 支持 TX 模块和 RX 模块独立工作或同时工作
- 支持多种音频标准：
 - TDM Philips 标准
 - TDM MSB 对齐标准
 - TDM PCM 标准
 - PDM 标准
- 支持多种 TX/RX 模式
 - TDM TX 模式，最多支持 16 通道
 - TDM RX 模式，最多支持 16 通道
 - PDM TX 模式
 - * 支持原始 PDM 数据发送
 - * 支持将 PCM 数据转换为 PDM 数据发送，最多支持 2 通道
 - PDM RX 模式
 - * 支持原始 PDM 数据接收
- 可配置时钟源，支持最高频率为 240 MHz
- 可配置高精度采样时钟，支持多种采样频率（详见下文注释）
- 支持 8/16/24/32 位的数据位宽
- TX 模式支持同步计数器
- 支持 ETM 功能
- 支持 GDMA
- 支持 I2S 接口中断

说明：

从机模式下，为了保障采样的准确性，模块时钟频率必须大于等于 BCK 时钟频率的 8 倍，因此 I2S 支持的采样频率受到数据位宽和声道数的限制。例如，时钟源频率最高为 240 MHz，即模块时钟最高能配置为 120 MHz，能支持的 BCK 时钟频率最高为 15 MHz，因此传输双声道 32 位宽数据时，能够支持 234.375 kHz 以下的采样频率，例如 8 kHz、16 kHz、32 kHz、44.1 kHz、48 kHz、88.2 kHz、96 kHz、128 kHz、192 kHz。具体请参考章节 [27.6 I2S TX/RX 模块时钟](#)。

27.4 系统架构

图 27.4-1 为 ESP32-C5 I2S 模块的结构框图。ESP32-C5 I2S 模块包含：

- 独立的 TX 模块
- 独立的 RX 模块
- 输入输出时序调节单元 (I/O 同步)
- 时钟分频器 (时钟发生器)
- 64 x 32 位 TX FIFO
- 64 x 32 位 RX FIFO
- 压缩/解压缩模块

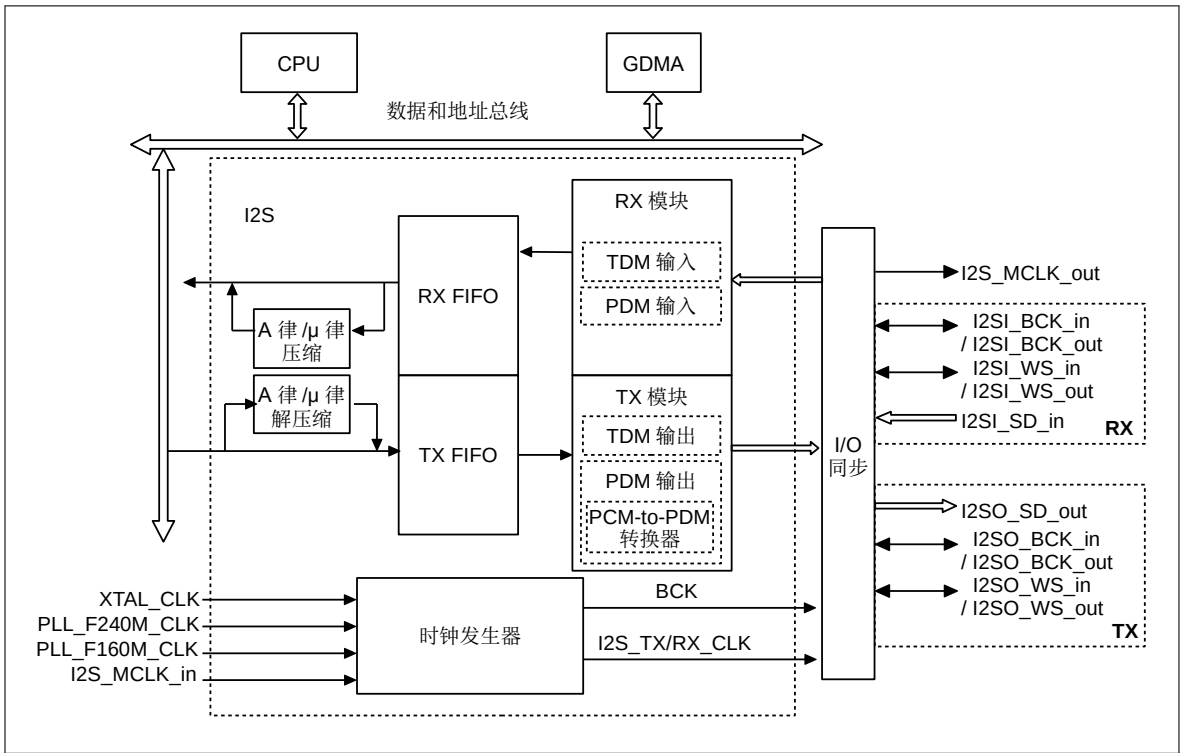


图 27.4-1. ESP32-C5 I2S 系统框图

ESP32-C5 I2S 模块支持 GDMA，可访问内部存储器，更多信息见章节 3 通用 DMA 控制器 (GDMA)。

TX 模块和 RX 模块各自有一组三线接口，分别为串行时钟线 BCK、字选择线 WS 和串行数据线 SD。其中，TX 模块的 SD 线固定为输出，RX 模块的 SD 线固定为接收。主机模式下，TX 模块和 RX 模块的 BCK 和 WS 信号线为输出，从机模式下，TX 模块和 RX 模块的 BCK 和 WS 信号线为输入。

图 27.4-1 中的右半部分展示了 I2S 模块的信号总线。RX 和 TX 模块的信号命名规则为：I2SA_B_C，例如 I2SI_BCK_in。其中：

- “A” 表示信号属于 I2S TX 或 RX 模块，包括：
 - “I” 表示信号输入至/输出自 RX 模块
 - “O” 表示信号输入至/输出自 TX 模块

- “B” 表示信号功能，包括：
 - BCK
 - WS
 - SD
- “C” 表示该信号的方向，包括：
 - “in” 表示该信号输入 I2S 模块
 - “out” 表示该信号自 I2S 模块输出

I2S 各信号的具体描述见表 27.4-1。

表 27.4-1. 模块信号描述

信号	方向	功能
I2SI_BCK_in	输入	I2S 从机模式下，输入 BCK 信号，用于 RX 模块
I2SI_BCK_out	输出	I2S 主机模式下，输出 BCK 信号，用于 RX 模块
I2SI_WS_in	输入	I2S 从机模式下，输入 WS 信号，用于 RX 模块
I2SI_WS_out	输出	I2S 主机模式下，输出 WS 信号，用于 RX 模块
I2SI_SD_in	输入	I2S RX 模块的串行输入数据线
I2SO_SD_out	输出	I2S TX 模块的串行输出数据线
I2SO_BCK_in	输入	I2S 从机模式下，输入 BCK 信号，用于 TX 模块
I2SO_BCK_out	输出	I2S 主机模式下，输出 BCK 信号，用于 TX 模块
I2SO_WS_in	输入	I2S 从机模式下，输入 WS 信号，用于 TX 模块
I2SO_WS_out	输出	I2S 主机模式下，输出 WS 信号，用于 TX 模块
I2S_MCLK_in	输入	I2S 从机模式下，来自外部芯片的时钟源
I2S_MCLK_out	输出	I2S 主机模式下，作为外部芯片的时钟源
I2SO_SD1_out	输出	TX 模块在启用 PCM-to-PDM 转换器时的串行输出数据线

* I2S 的所有信号均需要经过 GPIO 交换矩阵映射到芯片的管脚。更多信息请参考章节 5 [GPIO 交换矩阵和 IO MUX \(GPIO, IO MUX\)](#)。

27.5 I2S 模块支持的音频协议

ESP32-C5 I2S 模块支持多种音频标准，包括 TDM Philips 标准、TDM MSB 对齐标准、TDM PCM 标准以及 PDM 标准。

用户可通过配置以下寄存器，选择所需的音频标准：

- [I2S_TX/RX_TDM_EN](#)
 - 0：禁用 TDM 模式
 - 1：选择 TDM 模式
- [I2S_TX/RX_PDM_EN](#)
 - 0：禁用 PDM 模式
 - 1：选择 PDM 模式

- **I2S_TX/RX_MSB_SHIFT**

- 0: 配置 WS 信号和 SD 信号同时开始变化，即选择 MSB 对齐标准
- 1: 配置 WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即选择 Philips 标准或 PCM 标准

说明:

不可将 **I2S_TX/RX_TDM_EN** 和 **I2S_TX/RX_PDM_EN** 同时配置为 1 或同时配置为 0，否则会导致 ESP32-C5 I2S 以既非 TDM 模式也非 PDM 模式的状态错误发送数据。

27.5.1 TDM Philips 标准模式

在 Philips 标准下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并在当前通道数据发送结束前一个 BCK 时钟周期开始变化。SD 信号线上首先传输音频数据的最高位。

与普通 Philips 标准相比，TDM Philips 标准支持的通道更多，见图 27.5-1。

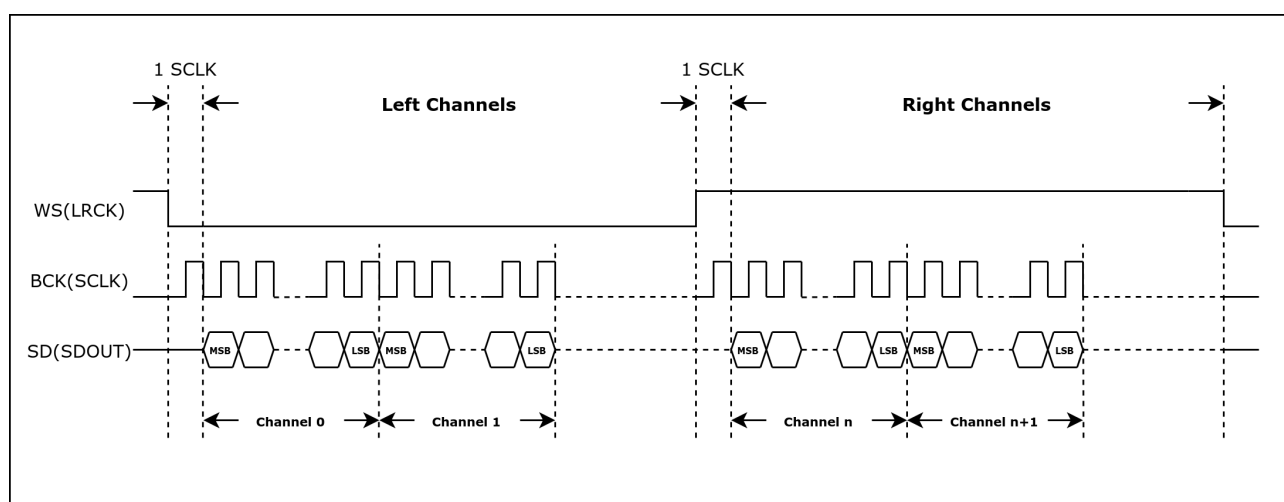


图 27.5-1. 时序图 – TDM Philips 标准

27.5.2 TDM MSB 对齐标准模式

MSB 对齐标准下，在 BCK 下降沿，WS 信号和 SD 信号同时变化。WS 持续到当前通道数据发送结束，SD 信号线上首先传输音频数据的最高位。

与普通 MSB 对齐标准相比，TDM MSB 对齐标准支持更多通道，见图 27.5-2。

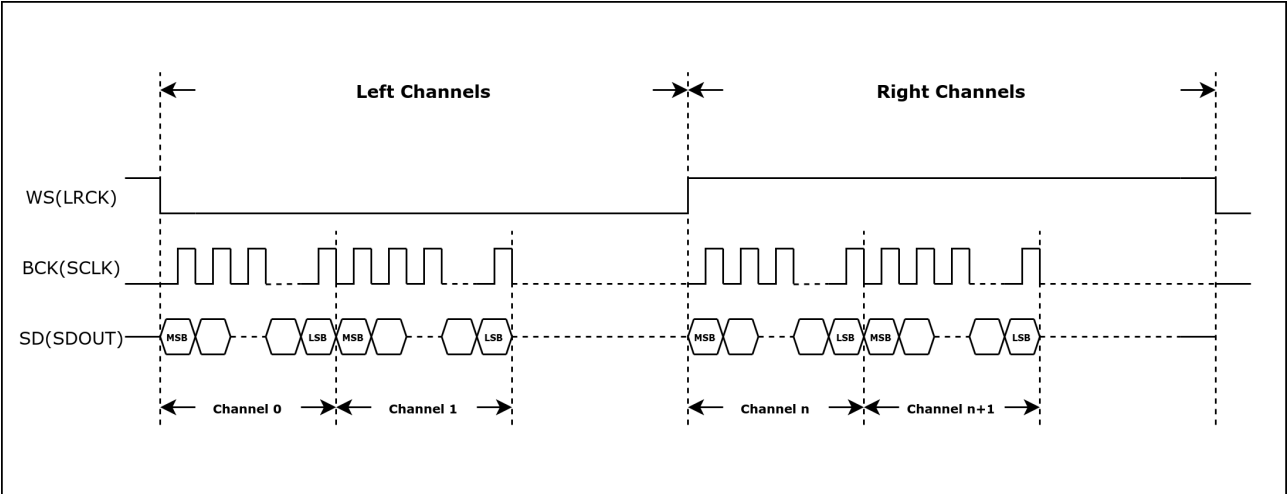


图 27.5-2. 时序图 – TDM MSB 对齐标准

27.5.3 TDM PCM 标准模式

在 PCM 标准的短帧同步模式下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并持续一个 BCK 时钟周期。SD 信号线上首先传输音频数据的最高位。

与普通 PCM 标准相比，TDM PCM 标准支持更多通道，见图 27.5-3 所示。

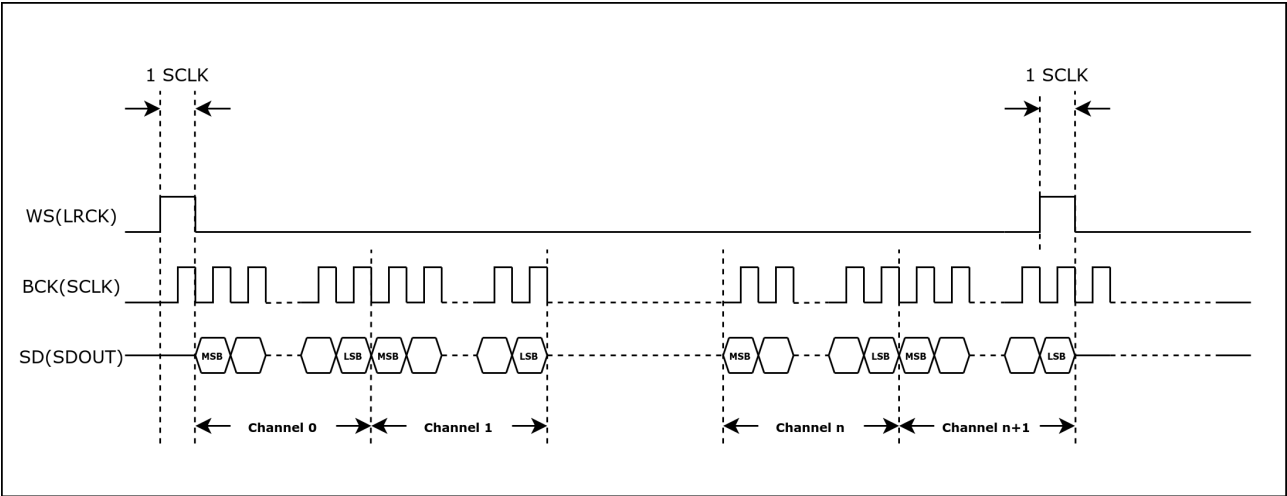


图 27.5-3. 时序图 – TDM PCM 标准

27.5.4 PDM 标准模式

如图 27.5-4 所示，在 PDM 标准下，WS 代表左/右声道，WS 与 SD 同时变化。WS 在数据发送过程中持续变化，WS 的高低对应两个声道。请参考图 27.5-4。

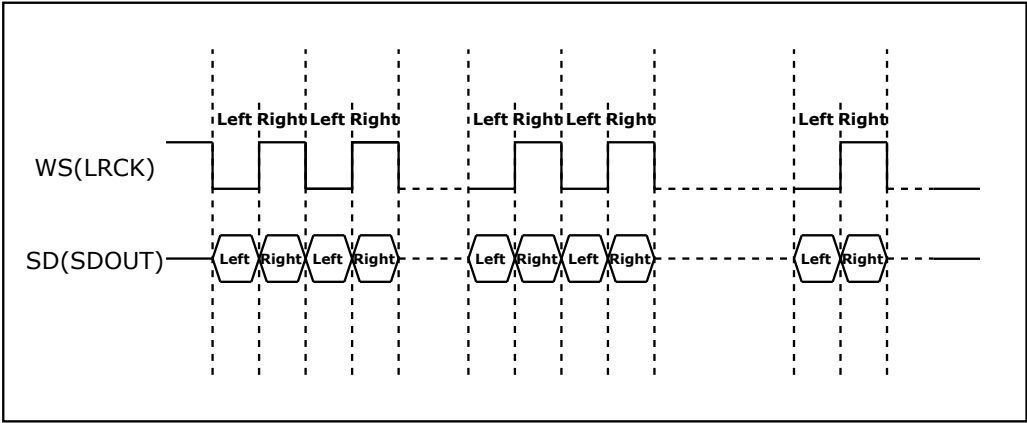


图 27.5-4. 时序图 – PDM 标准

27.6 I2S TX/RX 模块时钟

I2S_TX/RX_CLK 作为 I2S TX/RX 模块的主时钟，可由以下时钟分频所得：

- 40 MHz XTAL_CLK
- 240 MHz PLL_F240M_CLK
- 160 MHz PLL_F160M_CLK
- 外部输入时钟 I2S_MCLK_in

I2S TX/RX 模块的串行时钟 BCK 再由 I2S_TX/RX_CLK 分频获得，如图 27.6-1 所示。PCR_I2S_TX/RX_CLKM_SEL 用于选择 I2S TX/RX 的时钟源，PCR_I2S_TX/RX_CLKM_EN 用于使能或者关闭 I2S TX/RX 模块的时钟源。

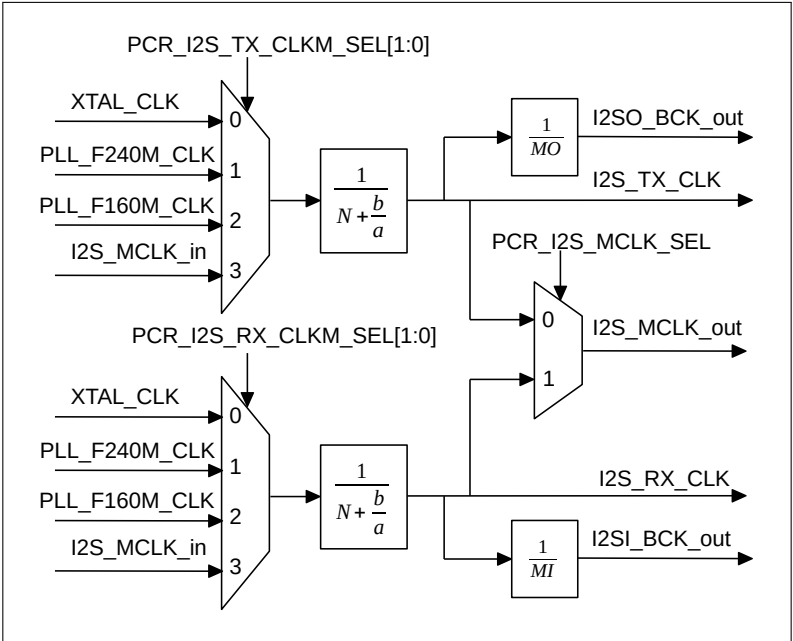


图 27.6-1. I2S 时钟分频器

I2S_TX/RX_CLK 的频率 f_{I2S_TX/RX_CLK} 与分频器时钟源频率 $f_{I2S\_CLK\_S}$ 间的关系如下：

$$f_{I2S\_TX/RX\_CLK} = \frac{f_{I2S\_CLK\_S}}{N + \frac{b}{a}}$$

其中， $2 \leq N \leq 256$ (N 为整数)，N 对应寄存器域 `PCR_I2S_TX/RX_CLKM_DIV_NUM` 的值，具体为：

- `PCR_I2S_TX/RX_CLKM_DIV_NUM` = 0 时，N = 256；
- `PCR_I2S_TX/RX_CLKM_DIV_NUM` = 1 时，N = 2；
- `PCR_I2S_TX/RX_CLKM_DIV_NUM` 为其他值时，N = `PCR_I2S_TX/RX_CLKM_DIV_NUM` 的值。

分数部分分频系数 a、b 唯一对应系数 x、y、z、yn1。系数对应公式为：

- 当 $b \leq \frac{a}{2}$ 时，yn1 = 0， $x = \text{floor}(\lceil \frac{a}{b} \rceil) - 1$ ， $y = a \% b$ ， $z = b$ ；
- 当 $b > \frac{a}{2}$ 时，yn1 = 1， $x = \text{floor}(\lceil \frac{a}{a-b} \rceil) - 1$ ， $y = a \% (a - b)$ ， $z = a - b$ 。

系数 x、y、z、yn1 在 `PCR_I2S_TX/RX_CLKM_DIV_X`、`PCR_I2S_TX/RX_CLKM_DIV_Y`、`PCR_I2S_TX/RX_CLKM_DIV_Z` 和 `PCR_I2S_TX/RX_CLKM_DIV_YN1` 中配置。

对于整数分频，`PCR_I2S_TX/RX_CLKM_DIV_X` 和 `PCR_I2S_TX/RX_CLKM_DIV_Z` 清零，`PCR_I2S_TX/RX_CLKM_DIV_Y` 置为 1。

说明：

使用小数分频功能可能会产生时钟抖动。

在主机发送模式下，I2S TX 模块的串行时钟 BCK 为 `I2SO_BCK_out` 信号，由 `I2S_TX_CLK` 分频获得。即：

$$f_{I2SO\_BCK\_out} = \frac{f_{I2S\_TX\_CLK}}{MO}$$

其中，MO 的值为：

$$MO = \text{I2S\_TX\_BCK\_DIV\_NUM} + 1$$

说明：

注意，`I2S_TX_BCK_DIV_NUM` 不可配置为 1。

在主机接收模式下，I2S RX 模块的串行时钟 BCK 为 `I2SI_BCK_out` 信号，由 `I2S_RX_CLK` 分频获得。即：

$$f_{I2SI\_BCK\_out} = \frac{f_{I2S\_RX\_CLK}}{MI}$$

其中 MI 的值为：

$$MI = \text{I2S\_RX\_BCK\_DIV\_NUM} + 1$$

说明：

- `I2S_RX_BCK_DIV_NUM` 不可配置为 1；
- 当模块处于 I2S 从机模式时，必须保证 $f_{I2S\_TX/RX\_CLK} \geq 8 * f_{BCK}$ 。另外模块可以输出 `I2S_MCLK_out` 作为外部设备的主时钟。

27.7 I2S 模块复位

I2S 模块中各个单元以及 FIFO 可通过配置相关位进行复位：

- I2S TX/RX 单元：可配置 `I2S_TX_RESET` 和 `I2S_RX_RESET` 进行复位；
- I2S TX/RX FIFO：可配置 `I2S_TX_FIFO_RESET` 和 `I2S_RX_FIFO_RESET` 进行复位。

说明：

在模块和 FIFO 复位之前，需要先配置 I2S 模块时钟。

27.8 I2S 主/从机模式

ESP32-C5 I2S 模块可作为主机或从机，两种模式均支持半双工通信或全双工通信。用户可配置 `I2S_RX_SLAVE_MOD` 和 `I2S_TX_SLAVE_MOD` 选择需要的模式。

- `I2S_TX_SLAVE_MOD`
 - 0：主机发送模式
 - 1：从机发送模式
- `I2S_RX_SLAVE_MOD`
 - 0：主机接收模式
 - 1：从机接收模式

27.8.1 主/从机发送模式

- 主机发送模式
 - 置位 `I2S_TX_START` 启动一次发送操作。置位该位，TX 模块会一直输出时钟信号和串行数据。
 - 置位 `I2S_TX_STOP_EN` 时，如果 FIFO 中的数据全部发送完毕，则主机停止发送数据和时钟信号。
 - 清零 `I2S_TX_STOP_EN` 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，发送模块将一直发送最后一帧数据和时钟信号。
 - 当 `I2S_TX_START` 被清零时，主机停止发送数据。
- 从机发送模式
 - 置位 `I2S_TX_START`。TX 模块等待主机 BCK 时钟，来启动发送操作。
 - 置位 `I2S_TX_STOP_EN` 时，如果 FIFO 中的数据全部发送完毕，则从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。
 - 清零 `I2S_TX_STOP_EN` 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，TX 模块将一直发送最后一帧数据。
 - 当 `I2S_TX_START` 被清零时，从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。

27.8.2 主/从机接收模式

- 主机接收模式

- 置位 `I2S_RX_START` 启动一次接收操作。RX 模块会一直输出时钟信号，并对输入数据进行采样。
- 配置 `I2S_RX_STOP_MODE` 来控制接收模式何时暂停接收：
 - * 0: 只有 `I2S_RX_START` 被清除，RX 模块才会暂停接收。
 - * 1: RX 模块会在 `I2S_RX_START` 被清除，或者接收字节数大于 `I2S_RX_EOF_NUM_REG` 配置的接收长度值时暂停接收。当 `I2S_RX_START` 未被清除时，可通过置位 `I2S_RX_RESET` 来重新启动接收。
 - * 2: RX 模块会在 `I2S_RX_START` 被清除或 GDMA RX FIFO 满时暂停接收。当 `I2S_RX_START` 未被清除且 GDMA RX FIFO 不再满时，可通过置位 `I2S_RX_RESET` 重新启动接收。
- 清零 `I2S_RX_START`，RX 模块停止接收数据。

- 从机接收模式

- 置位 `I2S_RX_START`。等待主机 BCK 时钟，来启动接收操作。
- 配置 `I2S_RX_STOP_MODE` 来控制接收模式何时暂停接收：
 - * 0: 只有 `I2S_RX_START` 被清除，RX 模块才会暂停接收；
 - * 1: RX 模块会在 `I2S_RX_START` 被清除，或者接收字节数大于 `I2S_RX_EOF_NUM_REG` 配置的接收长度值时暂停接收。当 `I2S_RX_START` 未被清除时，可通过置位 `I2S_RX_RESET` 来重新开始接收；
 - * 2: RX 模块会在 `I2S_RX_START` 被清除或 GDMA RX FIFO 满时暂停接收。当 `I2S_RX_START` 未被清除且 GDMA RX FIFO 不再满时，可通过置位 `I2S_RX_RESET` 恢复接收。
- 清零 `I2S_RX_START`，RX 模块停止接收数据。

27.9 发送数据

说明:

本小节以及后续小节所述的配置，均需要通过置位 `I2S_TX_UPDATE` 的方式来进行更新，从而将 I2S TX 寄存器数据从 APB 时钟域同步到 I2S TX 时钟域。详细配置见第 27.13.1 小节。

I2S 发送数据时，从 GDMA 读取数据，经过数据格式控制和通道模式控制，从外设输出信号输出对应数据。

27.9.1 数据格式控制

数据格式控制分为三个阶段：

- 第一阶段：从内存中读出有效数据并写入 TX FIFO；
- 第二阶段：将待发送数据从 TX FIFO 中读出，并进行输出数据模式转换；
- 第三阶段：将待发送数据转换为串行数据流输出。

27.9.1.1 通道有效数据位宽

I2S_TX_BITS_MOD 和 I2S_TX_24_FILL_EN 决定了每个通道的有效数据位宽，其可取值和对应的有效数据位宽如下表：

表 27.9-1. 通道有效数据位宽控制

通道有效数据位宽	I2S_TX_BITS_MOD	I2S_TX_24_FILL_EN
32	31	x [*]
	23	1
24	23	0
16	15	x
8	7	x

^{*}“x” 表示该值被忽略（适用于本章所有表格）。

27.9.1.2 通道有效数据字节序

I2S 通过 GDMA 读取数据之后，I2S_TX_BIG_ENDIAN 用于控制从 GDMA 读取数据的字节序。表 27.9-2 描述了不同通道有效数据位宽下，该寄存器对读取数据的控制。

表 27.9-2. 通道有效数据字节序控制

通道有效数据位宽	原始数据	控制后数据字节序	I2S_TX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

说明：

B0、B1、B2、B3 各代表一个 8 位数据，符号 {} 表示这些字节被组合到一起。例如，{B3, B2, B1, B0} 代表一个 32 位的数据，其中 B0 代表 0-7 位，B1 代表 8-15 位，B2 代表 16-23 位，B3 代表 24-31 位。

27.9.1.3 A 律/ μ 律压缩/解压缩

ESP32-C5 I2S 按照 32 位（缺省高位补 0^{*}）的方式对排列好字节序的有效数据进行 A 律/ μ 律压缩/解压缩。

说明：

缺省高位补 0，即如果有效数据位宽小于 32，则待压缩/解压缩数据的 [31: 有效位宽] 位自动填充 0。

配置 I2S_TX_PCM_BYPASS 为：

- 0：进行压缩/解压缩
- 1：不进行压缩/解压缩

配置 `I2S_TX_PCM_CONF` 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2： μ 律解压缩
- 3： μ 律压缩

至此，数据格式控制的第一阶段完成。

27.9.1.4 通道发送数据位宽

ESP32-C5 I2S 中，`I2S_TX_TDM_CHAN_BITS` 决定了每个通道发送数据的位宽。

- 当每个通道发送数据的位宽大于有效数据位宽时，会在剩余的位置补充 0。此时，配置 `I2S_TX_LEFT_ALIGN` 为：
 - 0：有效数据位于发送数据低位，在数据高位补零；
 - 1：有效数据位于发送数据高位，在数据低位补零。
- 当每个通道发送数据的位宽小于有效数据位宽时，每次发送数据仅发送低位有效数据部分，高位部分将被舍弃。

至此，数据格式控制的第二阶段完成。

27.9.1.5 通道数据比特顺序

ESP32-C5 I2S 通道数据比特顺序由 `I2S_TX_BIT_ORDER` 控制：

- 0：不调换有效数据比特顺序；
- 1：调换有效数据比特顺序。

至此，数据格式控制部分全部完成。经过数据格式控制的数据将从高位向低位依次发送。图 27.9-1 所示即为一次完整的 TX 数据格式控制过程。

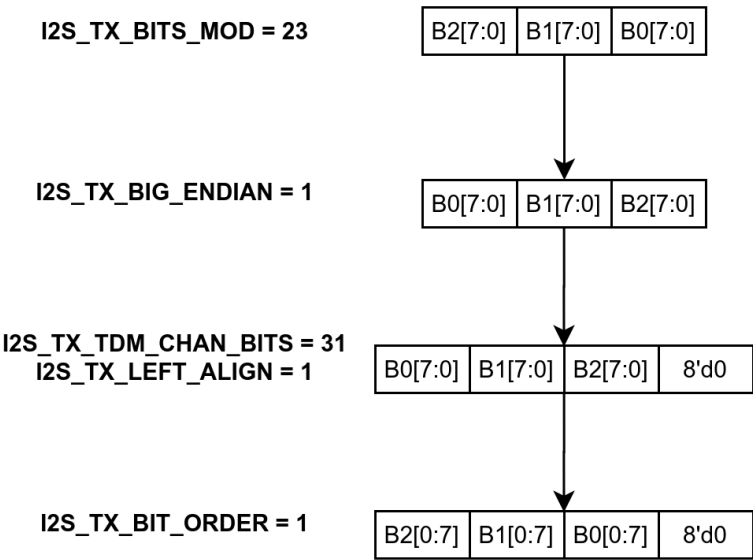


图 27.9-1. TX 数据格式控制

27.9.2 通道模式控制

ESP32-C5 I2S 支持 TDM 和 PDM 两种发送模式。置位 `I2S_TX_TDM_EN` 使能 TDM 输出模式，置位 `I2S_TX_PDM_EN` 使能 PDM 输出模式。

说明：
`I2S_TX_TDM_EN` 和 `I2S_TX_PDM_EN` 不能同时置位或同时清零。

27.9.2.1 TDM 输出模式

在 TDM 输出模式下，I2S 最多支持 16 个发送通道。发送通道数由 `I2S_TX_TDM_TOT_CHAN_NUM` 控制。例如，配置 `I2S_TX_TDM_TOT_CHAN_NUM` 为 5，则六个通道（通道 0 ~ 5）将用于发送数据，见图 27.9-2。

说明：
将 I2S 模块设置为 TDM 双通道模式，可实现控制大多数 I2S 双声道编解码器。

在发送数据的通道中，如果其对应的 `I2S_TX_TDM_CHANn_EN` 为：

- 1：则该通道发送通道数据；
- 0：则该通道发送数据由 `I2S_TX_CHAN_EQUAL` 控制，当该值为：
 - 1：则发送前一个通道的数据；
 - 0：则发送 `I2S_SINGLE_DATA` 的值。

当 I2S 处于主机 TDM 输出模式下，WS 信号由 `I2S_TX_WS_IDLE_POL` 和 `I2S_TX_TDM_WS_WIDTH` 控制：

- `I2S_TX_WS_IDLE_POL` 的值为 WS 信号的默认电平；
- `I2S_TX_TDM_WS_WIDTH` 的值为在发送所有通道数据的过程中，WS 为默认电平的周期数。

另外，`I2S_TX_HALF_SAMPLE_BITS` 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

TDM 通道配置示例

在本示例中，寄存器的配置如下：

- 配置 `I2S_TX_TDM_CHAN_NUM` 为 5，即选择使用通道 0 ~ 5 进行数据发送。
- 配置 `I2S_TX_CHAN_EQUAL` 为 1，即如果相应通道的 `I2S_TX_TDM_CHAN $n$ _EN` ($n = 0 \sim 5$) 被清除，则该通道将发送前一通道的数据。
- 配置 `I2S_TX_TDM_CHAN0/2/5_EN` 为 1，即这些通道将用于发送通道数据。
- 配置 `I2S_TX_TDM_CHAN1/3/4_EN` 为 0，则这些通道将发送前一个通道的数据。

配置完成后，则数据将按下图方式进行发送。

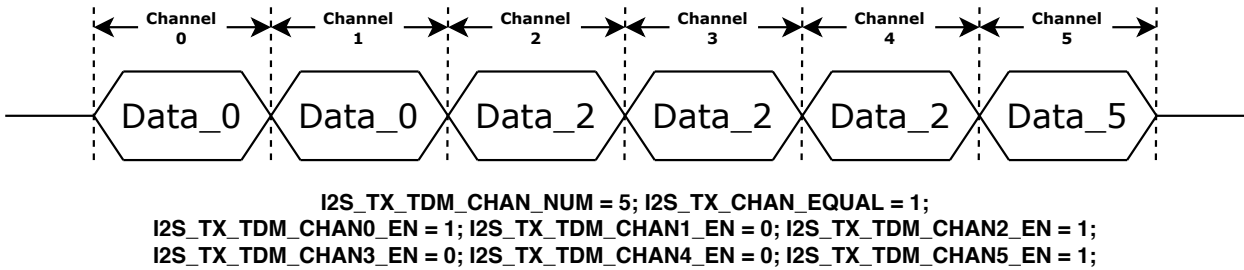


图 27.9-2. TDM 通道控制

27.9.2.2 PDM 输出模式

在 PDM 输出模式下，I2S 可以发送原始 PDM 数据，或者将 PCM 数据转换为 PDM 数据后发送。

在 PDM 输出模式下，从 GDMA 取得数据的过程由 `I2S_TX_MONO` 和 `I2S_TX_MONO_FST_VLD` 控制，具体如下表。请根据内存中存储的数据为单/双声道数据来配置该寄存器。

表 27.9-3. PDM 输出模式下 I2S 取数逻辑

取数逻辑	模式	<code>I2S_TX_MONO</code>	<code>I2S_TX_MONO_FST_VLD</code>
每个 WS 沿都向 GDMA 发起取数请求	双声道	0	x
只在 WS 后半周期向 GDMA 发起取数请求	单声道	1	0
只在 WS 前半周期向 GDMA 发起取数请求	单声道	1	1

当 I2S 处于主机 PDM 输出模式时，WS 信号的默认电平由 `I2S_TX_WS_IDLE_POL` 控制，WS 信号频率为 BCK 信号频率的一半。请参考 27.6 小节配置 BCK 信号的方式来配置 WS 信号，见图 27.9-3。

在发送原始 PDM 数据时，I2S 通道数据由 `I2S_TX_CHAN_MOD` 和 `I2S_TX_WS_IDLE_POL` 控制，具体如下表。

表 27.9-4. 原始 PDM 数据的 I2S 通道模式控制

模式	左声道	右声道	模式控制字 段 ¹	声道选择位 ²
双声道	发送左通道数据	发送右通道数据	0	x
单声道	发送左通道数据	发送左通道数据	1	0
	发送右通道数据	发送右通道数据	1	1
	发送右通道数据	发送右通道数据	2	0
	发送左通道数据	发送左通道数据	2	1
	发送“single”值 ³	发送右通道数据	3	0
	发送左通道数据	发送“single”值	3	1
	发送左通道数据	发送“single”值	4	0
	发送“single”值	发送右通道数据	4	1

¹ I2S_TX_CHAN_MOD
² I2S_TX_WS_IDLE_POL
³ single 值等于 I2S_SINGLE_DATA 的值。

在 启用 PCM-to-PDM 转换器 时，可以将 GDMA 中的 PCM 数据转为 PDM 数据，并按照 PDM 信号格式输出，配置 I2S_PCM2PDM_CONV_EN 启动该转换器。PCM-to-PDM 转换器的寄存器配置如下：

- 配置一线 PDM 输出格式或一/二线 DAC 输出格式，具体如下表。

表 27.9-5. I2S PCM-to-PDM 数据输出

通道输出格式	I2S_TX_PDM_DAC_MODE_EN	I2S_TX_PDM_DAC_2OUT_EN
一线 PDM 输出格式 ¹	0	x
一线 DAC 输出格式 ²	1	0
二线 DAC 输出格式	1	1

说明：

- 1. 此处定义的 PDM 输出格式是指一个 WS 周期发送两个通道的 SD 数据。
- 2. 此处定义的 DAC 输出格式是指一个 WS 周期发送一个通道的 SD 数据。
- 3. 一线 PDM/DAC 输出格式使用 I2SO_Data_out 作为数据输出线。
- 4. 二线 DAC 输出格式使用 I2SO_Data_out 和 I2SO_Data1_out 作为数据输出线。
- 5. 从应用上来说，PDM 输出格式针对的是需要时钟信号、可以解码 PDM 格式编解码器的应用场景，DAC 输出格式针对的是不依赖时钟信号、没有 PDM 格式编解码器、且通过低通滤波可以直接恢复成模拟波形的应用场景。

- 配置采样频率和上采样率。
I2S 启用 PCM-to-PDM 转换器进行数据输出时，PDM 时钟频率即为 BCK 时钟频率。采样频率（ f_{Sampling} ，即 PCM 数据采样频率）与 BCK 时钟频率（PDM 数据采样频率）的关系如下：

$$f_{\text{Sampling}} = \frac{f_{\text{BCK}}}{\text{OSR}}$$

其中，上采样率 OSR 和 `I2S_TX_PDM_SINC_OSR2` 的关系为：

$$\text{OSR} = \text{I2S\_TX\_PDM\_SINC\_OSR2} \times 64$$

采样频率 f_{Sampling} 和 `I2S_TX_PDM_FS` 的对应关系为：

$$f_{\text{Sampling}} = \text{I2S\_TX\_PDM\_FS} \times 100$$

请根据需要的采样频率、上采样率以及 PDM 时钟频率进行寄存器配置。

PDM 通道配置示例

本示例针对 I2S，示例中寄存器的配置如下：

- 配置 `I2S_PCM2PDM_CONV_EN` 为 0，即选择输出原始 PDM 数据。
- 配置 `I2S_TX_MONO` 为 0，即 I2S 在 WS 的高低电平均通过 GDMA 从存储器取数。
- 配置 `I2S_TX_CHAN_MOD` 为 2，即选择单声道模式。
- 配置 `I2S_TX_WS_IDLE_POL` 为 1，则左声道和右声道均发送左通道数据，右声道数据将被舍弃。

配置完成后，假定存储器中的数据经过数据格式控制之后为：

Left	Right	Left	Right	...	Left	Right

说明：

- 此处的数据并非存储器中的原始数据，而是经过数据格式控制之后的数据。
- 表中的“Left”和“Right”代表通道数据，其位宽为通道有效数据位宽，请参考章节 27.9.1。

则数据经过通道模式控制后，将按照下图方式进行发送。

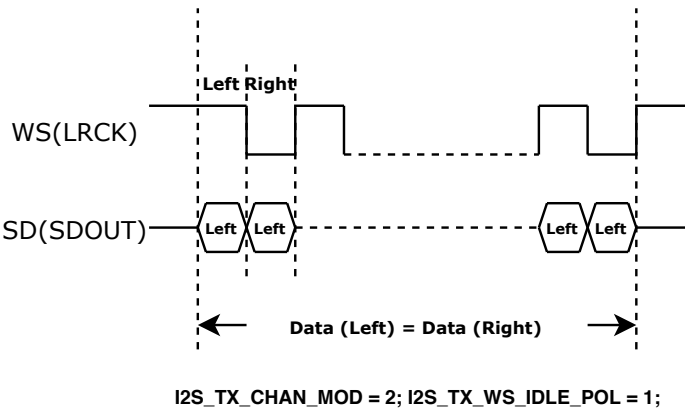


图 27.9-3. PDM 通道控制

27.9.3 同步计数器

ESP32-C5 I2S TX 模块包含同步计数器，用于反映当前 I2S 的数据发送情况。同步计数器可用于检测 I2S 发送数据错误，或在多个设备使用 I2S 发送数据时进行同步。

ESP32-C5 I2S 共包含两种类型的同步计数器：

- [I2S_TX_FIFO_CNT](#)

- 每次 I2S TX FIFO 成功读取一个数据, 该计数器加 1。
- 置位 [I2S_TX_FIFO_CNT_RST](#) 可复位该同步计数器。
- 该计数器位宽为 31 比特, 溢出时自动归零。

- [I2S_TX_BCK_CNT](#)

- 每次 I2S TX 发送一个 BCK 周期, 该计数器加 1。
- 可以通过置位 [I2S_TX_BCK_CNT_RST](#) 来复位该同步计数器。
- 该计数器位宽为 31 比特, 溢出时自动归零。

说明:

- 每次 I2S TX FIFO 发送一个数据, 均意味着发送了一个通道数据, 因此, 当 I2S 正确发送数据时, $I2S\_TX\_BCK\_CNT = I2S\_TX\_FIFO\_CNT * \text{通道发送数据位宽}$ (请参考小节 27.9.1.4)。
- 由于 I2S TX 模块发送数据时, 从 FIFO 取出数据到把数据发送至线上存在一些延迟, 因此读到 [I2S_TX_BCK_CNT](#) 的值与 [I2S_TX_FIFO_CNT](#) 的值可能略微不对应。

应用场景举例

1. 多个设备 I2S TX 同步

多个设备同时通过 I2S TX 发送数据时, 由于时钟源的微小频率差, 长时间发送数据后会出现同步问题。此时, 可以通过读取每个设备的 [I2S_TX_FIFO_CNT](#) 值来判断发送音频数据的位置, 从而进行同步操作。

2. I2S TX 稳定性检查

当设备处于 GDMA 占用较多的应用场景时, 可能会由于多路仲裁, 导致 I2S TX 出现偶尔缺数。此时, 检查 [I2S_TX_BCK_CNT](#) 与 [I2S_TX_FIFO_CNT](#) 的比例关系, 可以确认设备运行 I2S TX 时的稳定性。

27.10 接收数据

I2S 接收数据时, 从外设接口读取数据, 经过通道模式控制和数据格式控制, 通过 GDMA 输入数据至内存。

27.10.1 通道模式控制

ESP32-C5 I2S 支持 TDM 和 PDM 两种输入模式。置位 [I2S_RX_TDM_EN](#) 使能 TDM 输入模式, 置位 [I2S_RX_PDM_EN](#) 使能 PDM 输入模式。

说明:

[I2S_RX_TDM_EN](#) 和 [I2S_RX_PDM_EN](#) 不能同时置位或同时清零。

27.10.1.1 TDM 输入模式

在 TDM 输入模式下, I2S 最多支持 16 个接收通道。接收通道数由 [I2S_RX_TDM_TOT_CHAN_NUM](#) 控制。例如, 配置 [I2S_RX_TDM_TOT_CHAN_NUM](#) 为 5, 则通道 0 ~ 5 接收数据。

在接收数据的通道中, 如果其对应的 [I2S_RX_TDM_CHAN_n_EN](#) 为:

- 0: 则该通道数据无效, 不会存入 RX FIFO;

- 1: 则该通道数据有效，会被存入 RX FIFO。

当 I2S 处于主机 TDM 输入模式下，WS 信号由 `I2S_RX_WS_IDLE_POL` 和 `I2S_RX_TDM_WS_WIDTH` 控制：

- `I2S_RX_WS_IDLE_POL` 的值为 WS 信号的默认电平；
- `I2S_RX_TDM_WS_WIDTH` 的值为在接收所有通道的过程中，WS 为默认电平的周期数。

另外，`I2S_RX_HALF_SAMPLE_BITS` 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

27.10.1.2 PDM 输入模式

在 PDM 输入模式下，I2S 可以接收原始 PDM 数据，将通道中的串行数据转换为待输入数据。

当 I2S 处于主机 PDM 输入模式下，WS 信号的默认电平由 `I2S_RX_WS_IDLE_POL` 控制，WS 信号频率为 BCK 信号频率的一半，请参考 27.6 小节配置 BCK 信号的方式配置 WS 信号。注意，在 PDM 输入模式下，请配置 `I2S_RX_HALF_SAMPLE_BITS` 的值与 `I2S_RX_BITS_MOD` 的值相同。

27.10.2 数据格式控制

数据格式控制分为两个阶段：

- 第一阶段：从串行数据流输入转换为待输入数据，并存入 RX FIFO；
- 第二阶段：将待输入数据从 RX FIFO 中读出，并进行输入数据模式转换。

27.10.2.1 通道数据比特顺序

ESP32-C5 I2S 通道数据将从高位到低位依次存入待输入数据，并由 `I2S_RX_BIT_ORDER` 控制是否调换比特序，当该值为：

- 0: 不调换待输入数据的比特序；
- 1: 调换待输入数据的比特序。

至此，数据格式控制的第一阶段完成，经过比特顺序控制的待输入数据被存入 RX FIFO。

27.10.2.2 通道储存数据位宽

`I2S_RX_BITS_MOD` 和 `I2S_RX_24_FILL_EN` 决定了每个通道的储存数据位宽，其可取值和对应的储存数据位宽如下表：

表 27.10-1. 通道储存数据位宽控制

通道储存数据位宽	<code>I2S_RX_BITS_MOD</code>	<code>I2S_RX_24_FILL_EN</code>
32	31	x
	23	1
24	23	0
16	15	x
8	7	x

27.10.2.3 通道接收数据位宽

I2S_RX_TDM_CHAN_BITS 决定了每个通道接收数据的位宽。

- 当每个通道储存数据位宽小于接收数据的位宽时，接收时仅会在接收数据中取储存位宽作为储存数据。此时，配置 I2S_RX_LEFT_ALIGN 为：
 - 0：储存数据位于接收数据低位；
 - 1：储存数据位于接收数据高位。
- 当每个通道接收数据的位宽小于储存数据位宽时，会将接收数据高位补 0，作为储存数据。

27.10.2.4 通道储存数据字节序

通道接收数据经过截取/补全后，成为了待储存数据，I2S_RX_BIG_ENDIAN 用于控制待储存数据的字节序。下表描述了不同通道储存数据位宽下，该寄存器对待储存数据的控制。

表 27.10-2. 通道储存数据字节序控制

通道储存数据位宽	原始数据	控制后数据	I2S_RX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

27.10.2.5 A 律/ μ 律压缩/解压缩

I2S 对排列好字节序的待储存数据会按照 32 位（缺省高位补 0）的方式进行 A 律/ μ 律压缩/解压缩。

配置 I2S_RX_PCM_BYPASS 为：

- 0：进行压缩/解压缩
- 1：不进行压缩/解压缩

配置 I2S_RX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2： μ 律解压缩
- 3： μ 律压缩

至此，数据格式控制部分全部完成，数据通过 GDMA 存入内存。

27.11 事件任务矩阵功能

在 ESP32-C5 中，I2S 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 I2S 的 ETM 任务，或者通过 I2S 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅

介绍与 I2S 相关的 ETM 任务和 ETM 事件。

I2S 可接收的 ETM 任务有：

- I2SO_TASK_START_TX：触发时启动 I2S TX 数据传输。
- I2SO_TASK_START_RX：触发时启动 I2S RX 数据传输。
- I2SO_TASK_STOP_TX：触发时停止 I2S TX 数据传输。
- I2SO_TASK_STOP_RX：触发时停止 I2S RX 数据传输。

I2S 可产生的 ETM 事件有：

- I2SO_EVT_TX_DONE：表示 I2S TX 发送完成，即 TX FIFO 中的数据全部发送完毕。
- I2SO_EVT_RX_DONE：根据寄存器 [I2S_RX_STOP_MODE](#) 的配置值决定此事件的触发条件：
 - 0：不会触发此事件；
 - 1：触发时表示 I2S RX 模块接收字节数大于 [I2S_RX_EOF_NUM_REG](#) 配置的接收长度；
 - 2：触发时表示 GDMA RX FIFO 满。
- I2SO_EVT_X_WORDS_SENT：表示 I2S TX 发送字数大于等于 [I2S_ETM_TX_SEND_WORD_NUM](#) 的值。
- I2SO_EVT_X_WORDS_RECEIVED：表示 I2S RX 接收字数大于等于 [I2S_ETM_RX_RECEIVE_WORD_NUM](#) 的值。

在具体应用中，可以配置 I2S 的 ETM 事件来触发 I2S 的 ETM 任务，例如配置 I2SO_EVT_X_WORDS_SENT 事件触发 I2SO_TASK_STOP_TX 任务，从而以 ETM 的方式完成停止 I2S 的操作。

27.12 I2S 中断

ESP32-C5 的 I2S 可以生成 I2S_INTR 中断信号并发送给[中断矩阵](#)。这一中断信号由以下 I2S 的内部中断源生成。

- I2S_TX_HUNG_INT：当发送数据超时，触发此中断。例如，I2S 配置为从机发送模式，但主机长时间未提供 BCK 或 WS 信号，则将触发该中断。超时配置见寄存器 [I2S_LC_HUNG_CONF_REG](#)。
- I2S_RX_HUNG_INT：当接收数据超时，触发此中断。例如，I2S 配置为从机接收模式，但主机长时间未发送数据，则将触发该中断。超时配置见寄存器 [I2S_LC_HUNG_CONF_REG](#)。
- I2S_TX_DONE_INT：当发送数据完成，即 I2S TX FIFO 中的数据全部发送完毕时，触发此中断。
- I2S_RX_DONE_INT：根据寄存器 [I2S_RX_DONE_MODE](#) 的配置值决定中断触发方式：
 - 0：当 I2S RX 模块接收字节数大于 [I2S_RX_EOF_NUM_REG](#) 配置的接收长度值时，触发此中断；
 - 1：当 GDMA RX FIFO 满时，触发此中断。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 [3 中断矩阵 \[to be added later\]](#) > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [27.14 寄存器列表](#)。

27.13 软件配置流程

27.13.1 软件配置 I2S 发送流程

软件配置 I2S 发送的流程如下：

1. 根据 27.6 小节的描述，配置时钟。
2. 根据表 27.4-1 的描述，配置信号管脚。
3. 根据主从机模式配置 `I2S_TX_SLAVE_MOD`：
 - 0：主机发送模式
 - 1：从机发送模式
4. 根据 27.9 小节的描述，配置正确的发送数据模式和发送通道模式，置位 `I2S_TX_UPDATE`。
5. 根据 27.7 小节的描述，复位 TX 模块和发送 FIFO。
6. 根据 27.12 小节的描述使能相应的中断。
7. 配置 GDMA 发送链表。
8. 根据需要置位 `I2S_TX_STOP_EN`，更多信息见章节 27.8.1。
9. 开始发送数据：
 - 主机模式下，等待 I2S 从设备配置完成后，置位 `I2S_TX_START` 开始发送数据；
 - 从机模式下，置位 `I2S_TX_START`。I2S 主设备提供 BCK 和 WS 信号后，开始发送数据。
10. 等待步骤 6 设置的中断信号，或查询 `I2S_TX_IDLE` 检查传输是否结束：
 - 0：发送设备为工作状态；
 - 1：发送设备为空闲状态。
11. 清零 `I2S_TX_START` 停止发送数据。

27.13.2 软件配置 I2S 接收流程

软件配置 I2S 接收模式的流程如下：

1. 根据 27.6 小节的描述，配置时钟。
2. 根据表 27.4-1 的描述，配置信号管脚。
3. 配置 `I2S_RX_SLAVE_MOD` 选择需要的模式：
 - 0：主机接收模式
 - 1：从机接收模式
4. 根据 27.10 小节的描述，配置正确的接收通道模式和接收数据模式，置位 `I2S_RX_UPDATE`。
5. 根据 27.7 小节的描述，复位 RX 模块和接收 FIFO。
6. 根据 27.12 小节的描述使能相应的中断。
7. 配置 GDMA 接收链表，并在 `I2S_RX_EOF_NUM_REG` 中配置接收数据长度。

8. 开始接收数据：

- 在主机模式下，等待从机准备好后，置位 `I2S_RX_START` 开始接收数据；
- 在从机模式下，置位 `I2S_RX_START`，等待主机提供 BCK 和 WS 信号后开始接收数据。

9. 接收的数据由 GDMA 根据配置存到存储器的指定地址。最终产生步骤 6 中设置的中断。

27.14 寄存器列表

本小节的所有地址均为相对于 I2S 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
I2S_INT_RAW_REG	I2S 原始中断寄存器	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S 中断状态寄存器	0x0010	RO
I2S_INT_ENA_REG	I2S 中断使能寄存器	0x0014	R/W
I2S_INT_CLR_REG	I2S 中断清除寄存器	0x0018	WT
RX 控制和配置寄存器			
I2S_RX_CONF_REG	I2S RX 配置寄存器	0x0020	varies
I2S_RX_CONF1_REG	I2S RX 配置寄存器 1	0x0028	R/W
I2S_RX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0050	R/W
I2S_RXEOF_NUM_REG	I2S RX 数据大小控制寄存器	0x0064	R/W
TX 控制和配置寄存器			
I2S_TX_CONF_REG	I2S TX 配置寄存器	0x0024	varies
I2S_TX_CONF1_REG	I2S TX 配置寄存器 1	0x002C	R/W
I2S_TX_PCM2PDM_CONF_REG	I2S TX PCM-to-PDM 配置寄存器	0x0044	R/W
I2S_TX_PCM2PDM_CONF1_REG	I2S TX PCM-to-PDM 配置寄存器	0x0048	R/W
I2S_TX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0054	R/W
RX 时钟和时序寄存器			
I2S_RX_TIMING_REG	I2S RX 时序控制寄存器	0x0058	R/W
TX 时钟和时序寄存器			
I2S_TX_TIMING_REG	I2S TX 时序控制寄存器	0x005C	R/W
控制和配置寄存器			
I2S_LC_HUNG_CONF_REG	I2S 超时配置寄存器	0x0060	R/W
I2S_CONF_SIGLE_DATA_REG	I2S single 数据寄存器	0x0068	R/W
TX 状态寄存器			
I2S_STATE_REG	I2S TX 状态寄存器	0x006C	RO
ETM 寄存器			
I2S_ETM_CONF_REG	I2S ETM 配置寄存器	0x0070	R/W
同步计数器寄存器			
I2S_FIFO_CNT_REG	I2S 同步计数器寄存器	0x0074	varies
I2S_BCK_CNT_REG	I2S 同步计数器寄存器	0x0078	varies
时钟寄存器			
I2S_CLK_GATE_REG	时钟门控寄存器	0x007C	R/W
版本控制寄存器			
I2S_DATE_REG	版本控制寄存器	0x0080	R/W

27.15 寄存器

本小节的所有地址均为相对于 I2S 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 27.1. I2S_INT_RAW_REG (0x000C)

(reserved)																												I2S_TX_HUNG_INT_RAW I2S_RX_HUNG_INT_RAW I2S_TX_DONE_INT_RAW I2S_RX_DONE_INT_RAW																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
31																												4	3	2	1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- I2S_RX_DONE_INT_RAW I2S_RX_DONE_INT 中断的原始中断状态。(R/SS/WTC)
- I2S_TX_DONE_INT_RAW I2S_TX_DONE_INT 中断的原始中断状态。(R/SS/WTC)
- I2S_RX_HUNG_INT_RAW I2S_RX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)
- I2S_TX_HUNG_INT_RAW I2S_TX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

Register 27.2. I2S_INT_ST_REG (0x0010)

(reserved)																												I2S_TX_HUNG_INT_ST I2S_RX_HUNG_INT_ST I2S_TX_DONE_INT_ST I2S_RX_DONE_INT_ST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
31																												4	3	2	1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- I2S_RX_DONE_INT_ST I2S_RX_DONE_INT 中断的屏蔽中断状态。(RO)
- I2S_TX_DONE_INT_ST I2S_TX_DONE_INT 中断的屏蔽中断状态。(RO)
- I2S_RX_HUNG_INT_ST I2S_RX_HUNG_INT 中断的屏蔽中断状态。(RO)
- I2S_TX_HUNG_INT_ST I2S_TX_HUNG_INT 中断的屏蔽中断状态。(RO)

Register 27.3. I2S_INT_ENA_REG (0x0014)

(reserved)																												I2S_TX_HUNG_INT_ENA I2S_RX_HUNG_INT_ENA I2S_TX_DONE_INT_ENA I2S_RX_DONE_INT_ENA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31																											4	3	2	1	0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- I2S_RX_DONE_INT_ENA 写 1 使能 I2S_RX_DONE_INT。(R/W)
- I2S_TX_DONE_INT_ENA 写 1 使能 I2S_TX_DONE_INT。(R/W)
- I2S_RX_HUNG_INT_ENA 写 1 使能 I2S_RX_HUNG_INT。(R/W)
- I2S_TX_HUNG_INT_ENA 写 1 使能 I2S_TX_HUNG_INT。(R/W)

Register 27.4. I2S_INT_CLR_REG (0x0018)

(reserved)																												I2S_TX_HUNG_INT_CLR I2S_RX_HUNG_INT_CLR I2S_TX_DONE_INT_CLR I2S_RX_DONE_INT_CLR																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31																												4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- I2S_RX_DONE_INT_CLR 写 1 清除 I2S_RX_DONE_INT。(WT)
- I2S_TX_DONE_INT_CLR 写 1 清除 I2S_TX_DONE_INT。(WT)
- I2S_RX_HUNG_INT_CLR 写 1 清除 I2S_RX_HUNG_INT。(WT)
- I2S_TX_HUNG_INT_CLR 写 1 清除 I2S_TX_HUNG_INT。(WT)

Register 27.5. I2S RX CONF REG (0x0020)

(reserved)					I2S_RX_BCK_DIV_NUM																				I2S_RX_PDM_EN										I2S_RX_TDM_EN										I2S_RX_BIT_ORDER										I2S_RX_WS_IDLE_POL										I2S_RX_24_FILL_EN										I2S_RX_LEFT_ALIGN										I2S_RX_DONE_MODE										I2S_RX_MSB_SHIFT										I2S_RX_PCM_BYPASS										I2S_RX_PCM_CONF										I2S_RX_MONO_FST_VLD										I2S_RX_UPDATE										I2S_RX_BIG_ENDIAN										I2S_RX_STOP_MODE										I2S_RX_SLAVE_MOD										I2S_RX_START										I2S_RX_FIFO_RESET										I2S_RX_RESET																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																	
31					27					26					21					20					19					18					17					16					15					14					13					12					11					10					9					8					7					6					5					4					3					2					1					0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
0					0					0					0					0					6					0					0					0					0					0					1					0					1					1					0x1					1					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0					0				

I2S_RX_RESET 配置是否复位接收单元。

0: 无效

1: 复位

(WT)

I2S_RX_FIFO_RESET 配置是否复位 RX FIFO。

0: 无效

1: 复位

(WT)

I2S_RX_START 配置是否开始接收数据。

0: 无效

1: 开始

(R/W/SC)

I2S_RX_SLAVE_MOD 配置是否使能从机接收模式。

0: 使能主机模式

1: 使能从机模式

(R/W)

I2S_RX_STOP_MODE 配置 I2S RX 何时暂停接收。

0: 只有 I2S_RX_START 被清除时暂停接收

1: **I2S_RX_START** 被清除, 或者接收字节数大于 **I2S_RX_EOF_NUM_REG** 配置的接收长度值时
暂停接收

2: 接收单元会在 I2S_RX_START 被清除, 或者 GDMA RX FIFO 满时暂停接收

(R/W)

I2S_RX_MONO 配置是否使能接收单元的单声道模式。

0: 禁用

1: 使能

(R/W)

I2S_RX_BIG_ENDIAN 配置 I2S RX 字节序。

0: 低字节数据写入低位地址

1: 低字节数据写入高位地址

(R/W)

见下页...

Register 27.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_UPDATE 配置是否将 I2S RX 寄存器从 APB 时钟域同步到 I2S RX 时钟域。寄存器完成更新后，该位将由硬件清除。

0: 无效

1: 同步

(R/W/SC)

I2S_RX_MONO_FST_VLD 配置 I2S RX 单声道模式下的有效数据通道。

0: 第二个通道数据有效

1: 第一个通道数据有效

(R/W)

I2S_RX_PCM_CONF 配置 I2S RX 压缩/解压缩模式。

0 (atol): A 律解压缩

1 (ltoa): A 律压缩

2 (utol): μ 律解压缩

3 (ltou): μ 律压缩

(R/W)

I2S_RX_PCM_BYPASS 配置接收数据是否将绕过压缩/解压缩单元。

0: 无效

1: 绕过压缩/解压缩单元

(R/W)

I2S_RX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0: 上升沿对齐

1: 相隔一个周期

(R/W)

I2S_RX_DONE_MODE 配置触发 [I2S_RX_DONE_INT](#) 中断的条件。

0: RX FIFO 满时

1: IN_SUC_EOF 为 1 时

(R/W)

I2S_RX_LEFT_ALIGN 配置 I2S RX 对齐模式。

0: 右对齐模式

1: 左对齐模式

(R/W)

I2S_RX_24_FILL_EN 配置 24 位通道数据位的储存数据位数。

0: 将 24 位通道数据位保存到 24 位储存数据

1: 将 24 位通道数据位保存到 32 位储存数据 (多余的位填充为 0)

(R/W)

见下页...

Register 27.5. I2S_RX_CONF_REG (0x0020)

[接上页...](#)

I2S_RX_WS_IDLE_POL 配置 WS 与接收数据通道的关系。

0: WS 为 0 时, 接收左通道数据; WS 为 1 时, 接收右通道数据

1: WS 为 1 时, 接收左通道数据; WS 为 0 时, 接收右通道数据

(R/W)

I2S_RX_BIT_ORDER 配置是否调换 I2S RX 待接收数据的位顺序。

0: 不调换

1: 调换

(R/W)

I2S_RX_TDM_EN 配置是否使能 I2S TDM RX 模式。

0: 禁用

1: 使能

(R/W)

I2S_RX_PDM_EN 配置是否使能 I2S PDM RX 模式。

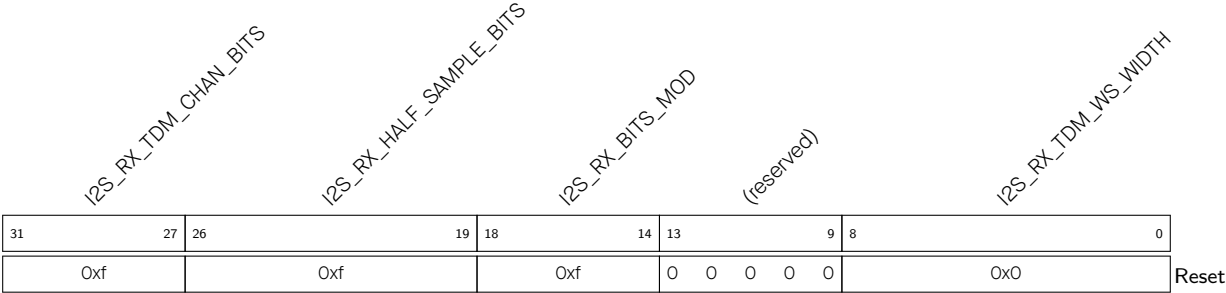
0: 禁用

1: 使能

(R/W)

I2S_RX_BCK_DIV_NUM 配置在 RX 模式下 BCK 时钟的分频系数。注意, 不可配置为 1。(R/W)

Register 27.6. I2S_RX_CONF1_REG (0x0028)



I2S_RX_TDM_WS_WIDTH 配置 I2SI_WS_out (WS 默认电平) 空闲时在 TDM 模式下的时长。
I2SI_WS_out 空闲时在 TDM 模式下的时长 = (I2S_RX_TDM_WS_WIDTH[8:0] + 1) x T_BCK。(R/W)

I2S_RX_BITS_MOD 配置在 RX 模式下接收通道的有效数据位长度。
7: 所有有效的通道数据均为 8 位模式
15: 所有有效的通道数据均为 16 位模式
23: 所有有效的通道数据均为 24 位模式
31: 所有有效的通道数据均为 32 位模式
其他值无效。
(R/W)

I2S_RX_HALF_SAMPLE_BITS 配置 I2S RX 单次采样比特数。一个 WS 信号中 BCK 持续的周期 = I2S_RX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_RX_TDM_CHAN_BITS 配置在 TDM RX 模式下每个通道的 RX 数据位长。RX 数据位长 = I2S_RX_TDM_CHAN_BITS + 1。(R/W)

Register 27.7. I2S_RX_TDM_CTRL_REG (0x0050)

[illegible]

I2S_RX_TDM_PDM_CHAN n _EN (n : 0-7) 配置是否使能 I2S TDM 或 PDM RX 通道 n 的有效输入数据。

0: 禁用该功能, 在该通道中仅输入 0

1: 使能

(R/W)

I2S_RX_TDM_CHAN n _EN (n : 8-15) 配置是否使能 I2S TDM RX 通道 n 的有效输入数据。

0: 禁用该功能, 在该通道中仅输入 0

1: 使能

(R/W)

I2S_RX_TDM_TOT_CHAN_NUM 配置在 I2S TDM RX 模式下使用的通道总数。使用的通道总数 = I2S_RX_TDM_TOT_CHAN_NUM + 1。(R/W)

Register 27.8. I2S_RXEOF_NUM_REG (0x0064)

Register 0x40 bit field diagram. The register is 32 bits wide, divided into two 16-bit halves. The top half (bits 31-16) is labeled "(reserved)". The bottom half (bits 15-0) is labeled "I2S_RX_EOF_NUM". The bottom half is further divided into a 10-bit field "I2S_RX_EOF_NUM[9:0]" and a 6-bit field "I2S_RX_EOF_NUM[5:0]". The bottom half is also labeled "Reset".

I2S_RX_EOF_NUM 配置接收数据的长度。接收数据长度 = (I2S_RX_BITS_MOD + 1) x (I2S_RX_EOF_NUM + 1)。如果接收数据的长度达到该值，则将在已配置的 GDMA RX 通道中触发 **AHB_DMA_IN_SUC_EOF_CHn_INT** 中断。(R/W)

Register 27.9. I2S_TX_CONF_REG (0x0024)

(reserved)		I2S_SIG_LOOPBACK				I2S_TX_CHAN_MOD				I2S_TX_BCK_DIV_NUM				I2S_TX_PDM_EN		I2S_TX_TDM_EN		I2S_TX_BIT_ORDER		I2S_TX_WS_IDLE_POL		I2S_TX_24_FILL_EN		I2S_TX_LEFT_ALIGN		I2S_TX_BCK_NO_DLY		I2S_TX_PCM_BYPASS		I2S_TX_PCM_CONF		I2S_TX_MONO_FST_VLD		I2S_TX_UPDATE		I2S_TX_BIG_ENDIAN		I2S_TX_CHAN_EQUAL		I2S_TX_STOP_EN		I2S_TX_SLAVE_EN		I2S_TX_START		I2S_TX_FIFO_RESET	
31	30	29	27	26	21				20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																		
0	0	0		6				0	0	0	0	0	1	1	1	1	0x0	1	0	0	0	0	0	0	1	0	0	0	0	Reset																	

I2S_TX_RESET 配置是否复位发送单元。

0: 无效

1: 复位

(WT)

I2S_TX_FIFO_RESET 配置是否复位 TX FIFO。

0: 无效

1: 复位

(WT)

I2S_TX_START 配置是否开始发送数据。

0: 无效

1: 开始发送

(R/W/SC)

I2S_TX_SLAVE_MOD 配置是否使能从机发送模式。

0: 使能主机模式

1: 使能从机模式

(R/W)

I2S_TX_STOP_EN 配置当 TX FIFO 为空时，发送单元是否停止输出 BCK 和 WS 信号。

0: 无效

1: 停止

(R/W)

I2S_TX_CHAN_EQUAL 配置在 I2S TX 单声道模式或 TDM 模式下，是否使能左声道数据等于右声道数据。

0: 在 I2S TX 单声道模式或 TDM 模式下, I2S_SINGLE_DATA 为无效的通道数据

1: 在 I2S TX 单声道模式或 TDM 模式下, 左声道数据等于右声道数据

(R/W)

I2S_TX_MONO 配置是否使能发送单元的单声道模式。

0: 禁用

1: 使能

(R/W)

I2S_TX_BIG_ENDIAN 配置 I2S TX 字节序。

0: 低字节数据写入低位地址

1: 低字节数据写入高位地址

(R/W)

见下页...

Register 27.9. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_UPDATE 配置是否将 I2S TX 寄存器从 APB 时钟域同步到 I2S TX 时钟域。寄存器更新完成后，此位将由硬件清除。

- 0: 无效
 - 1: 同步
- (R/W/SC)

I2S_TX_MONO_FST_VLD 配置在 I2S TX 单声道模式下的有效数据通道。

- 0: 第二个通道数据有效
 - 1: 第一个通道数据有效
- (R/W)

I2S_TX_PCM_CONF 配置 I2S TX 压缩/解压缩模式。

- 0 (atol): A 律解压缩
 - 1 (ltoa): A 律压缩
 - 2 (utol): μ 律解压缩
 - 3 (ltou): μ 律压缩
- (R/W)

I2S_TX_PCM_BYPASS 配置是否发送数据将绕过压缩/解压缩单元。

- 0: 无效
 - 1: 绕过
- (R/W)

I2S_TX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

- 0: 上升沿对齐
 - 1: 相隔一个周期
- (R/W)

I2S_TX_BCK_NO_DLY 配置在主机模式下，BCK 上升沿和下降沿生成来源。

- 0: 用 I2SO_BCK_in 输入的 BCK 构建上升沿和下降沿
 - 1: 用 I2S TX 模块时钟分频构建上升沿和下降沿
- (R/W)

I2S_TX_LEFT_ALIGN 配置 I2S TX 对齐模式。

- 0: 右对齐模式
 - 1: 左对齐模式
- (R/W)

I2S_TX_24_FILL_EN 配置 24 位数据的发送格式。

- 0: 将 24 位数据以 24 位的格式发送出去
 - 1: 将 24 位数据以 32 位的格式发送出去（不足的位用 0 填充）
- (R/W)

见下页...

Register 27.9. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_WS_IDLE_POL 配置 WS 与发送数据通道的关系。

0: WS 为 0 时, 发送左通道数据; WS 为 1 时, 发送右通道数据

1: WS 为 1 时, 发送左通道数据; WS 为 0 时, 发送右通道数据

I2S_TX_BIT_ORDER 配置是否调换 I2S TX 待发送有效数据的位顺序。

0: 不调换

1: 调换

(R/W)

I2S_TX_TDM_EN 配置是否使能 I2S TDM TX 模式。

0: 禁用

1: 使能

(R/W)

I2S_TX_PDM_EN 配置是否使能 I2S PDM TX 模式。

0: 禁用

1: 使能

(R/W)

I2S_TX_BCK_DIV_NUM 配置在 TX 模式下 BCK 时钟的分频系数。注意, 不可配置为 1。(R/W)

I2S_TX_CHAN_MOD 配置 I2S 发送单元通道, 更多信息见表 27.9-4。(R/W)

I2S_SIG_LOOPBACK 配置是否使能发送单元和接收单元共享 WS 和 BCK 信号。

0: 禁用

1: 使能

(R/W)

Register 27.10. I2S_TX_CONF1_REG (0x002C)

I2S_TX_TDM_CHAN_BITS					I2S_TX_HALF_SAMPLE_BITS					I2S_TX_BITS_MOD					(reserved)					I2S_TX_TDM_WS_WIDTH				
31	27	26	19	18	14	13	9	8	0															
0xf		0xf			0xf		0		0		0		0		0		0x0			Reset				

I2S_TX_TDM_WS_WIDTH 配置 I2SO_WS_out (WS 默认电平) 空闲时在 TDM 模式下的时长。
I2SO_WS_out 空闲时在 TDM 模式下的时长 = (I2S_TX_TDM_WS_WIDTH[8:0] + 1) x T_BCK。(R/W)

I2S_TX_BITS_MOD 配置 TX 模式下发送通道的有效数据位长度。
7: 所有有效的通道数据均为 8 位模式
15: 所有有效的通道数据均为 16 位模式
23: 所有有效的通道数据均为 24 位模式
31: 所有有效的通道数据均为 32 位模式
其他值无效。
(R/W)

I2S_TX_HALF_SAMPLE_BITS 配置 TX 单次采样比特数。一个 WS 信号中 BCK 持续的周期长 = I2S_TX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_TX_TDM_CHAN_BITS 配置在 TDM TX 模式下每个通道的 TX 数据位。TX 数据位 = I2S_TX_TDM_CHAN_BITS + 1。(R/W)

Register 27.11. I2S_TX_PCM2PDM_CONF_REG (0x0044)

(reserved)										I2S_PCM2PDM_CONV_EN										I2S_TX_PDM_DAC_MODE_EN										I2S_TX_PDM_DAC_2OUT_EN										(reserved)										(reserved)										(reserved)										(reserved)										I2S_TX_PDM_SINC_OSR2										(reserved)									
31						26	25	24	23	22	21	20	19	18	17	16	15	14	13	12						5	4						1	0																																																																	
0		0		0		0		0		0		0		1		0		0x1		0x1		0x1		0x1		0x0					0x2					0		Reset																																																													

I2S_TX_PDM_SINC_OSR2 配置 I2S TX PDM 的 OSR 值。(R/W)

I2S_TX_PDM_DAC_2OUT_EN 配置 DAC 输出模式。

- 0: 使能 1-line DAC 输出模式
 - 1: 使能 2-line DAC 输出模式
- 仅在 [I2S_TX_PDM_DAC_MODE_EN](#) 为 1 时有效。
(R/W)

I2S_TX_PDM_DAC_MODE_EN 配置使能 1-line PDM 输出模式或 DAC 输出模式。

- 0: 使能 1-line PDM 输出模式
 - 1: 使能 DAC 输出模式
- (R/W)

I2S_PCM2PDM_CONV_EN 配置是否使能 I2S TX PCM-to-PDM 转换器。

- 0: 禁用
 - 1: 使能
- (R/W)

Register 27.12. I2S_TX_PCM2PDM_CONF1_REG (0x0048)

(reserved)							(reserved)					(reserved)					I2S_TX_PDM_FS					(reserved)											
31						26	25						23	22						19						10	9						0
0	0	0	0	0	0	0	7					7					480					960					Reset						

I2S_TX_PDM_FS 配置 I2S TX PDM 上采样率。(R/W)

Register 27.13. I2S_TX_TDM_CTRL_REG (0x0054)

[illegible]

I2S_TX_TDM_CHAN n _EN (n : 0-15) 配置是否使能 I2S TDM TX 通道 n 的有效输出数据。

0: 通道发送数据由 I2S TX CHAN EQUAL 和 I2S SINGLE DATA 控制, 参考章节 27.9.2.1

1: 使能

(R/W)

I2S_TX_TDM_TOT_CHAN_NUM 配置在 I2S TDM TX 模式下使用的通道总数。通道总数 = I2S_TX_TDM_TOT_CHAN_NUM + 1。(R/W)

I2S_TX_TDM_SKIP_MSK_EN 配置 GDMA TX buffer 的发送数据。

0:GDMA TXbuffer 中的数据都用于已启用的通道,未启用的通道发送时不会读取 GDMA TX buffer 中的数据

1: GDMA TX buffer 中的数据在所有通道发送时均会被读取，因此在未启用的通道发送时 GDMA TX buffer 中的数据会被跳过

(R/W)

Register 27.14. I2S_RX_TIMING_REG (0x0058)

(reserved)		I2S_RX_BCK_IN_DM		(reserved)		I2S_RX_WS_IN_DM		(reserved)		I2S_RX_BCK_OUT_DM		(reserved)		I2S_RX_WS_OUT_DM		(reserved)		I2S_RX_SD3_IN_DM		(reserved)		I2S_RX_SD2_IN_DM		(reserved)		I2S_RX_SD1_IN_DM		(reserved)		I2S_RX_SD_IN_DM	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0	0x0	0	0

Reset

I2S_RX_SD_IN_DM 配置 I2S RX SD 输入信号的延迟模式。

- 0: 旁路
 - 1: 上升沿延迟
 - 2: 下降沿延迟
 - 3: 无效
- (R/W)

I2S_RX_SD1_IN_DM 配置 I2S RX SD1 输入信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。
(R/W)

I2S_RX_SD2_IN_DM 配置 I2S RX SD2 输入信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。(R/W)

I2S_RX_SD3_IN_DM 配置 I2S RX SD3 输入信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。(R/W)

I2S_RX_WS_OUT_DM 配置 I2S RX WS 输出信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。(R/W)

I2S_RX_BCK_OUT_DM 配置 I2S RX BCK 输出信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。(R/W)

I2S_RX_WS_IN_DM 配置 I2S RX WS 输入信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。
(R/W)

I2S_RX_BCK_IN_DM 配置 I2S RX BCK 输入信号的延迟模式。具体配置值请参考 [I2S_RX_SD_IN_DM](#)。(R/W)

Register 2715. I2S_TX_TIMING_REG (0x005C)

(reserved)				I2S_TX_BCK_IN_DM				(reserved)				I2S_TX_WS_IN_DM				(reserved)				I2S_TX_BCK_OUT_DM				(reserved)				I2S_TX_WS_OUT_DM				(reserved)				I2S_TX_SD1_OUT_DM				(reserved)				I2S_TX_SD_OUT_DM			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15											6	5	4	3	2	1	0														
0	0	0x0		0	0	0x0		0	0	0x0		0	0	0x0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0	0	0	0x0		Reset											

I2S_TX_SD_OUT_DM 配置 I2S TX SD 输出信号的延迟模式。

- 0：旁路
- 1：上升沿延迟
- 2：下降沿延迟
- 3：无效
- (R/W)

I2S_TX_SD1_OUT_DM 配置 I2S TX SD1 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_WS_OUT_DM 配置 I2S TX WS 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_OUT_DM 配置 I2S TX BCK 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_WS_IN_DM 配置 I2S TX WS 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_IN_DM 配置 I2S TX BCK 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

Register 27.16. I2S_LC_HUNG_CONF_REG (0x0060)

(reserved)																				I2S_LC_FIFO_TIMEOUT_ENA				I2S_LC_FIFO_TIMEOUT_SHIFT				I2S_LC_FIFO_TIMEOUT					
31																				12				11	10		8		7	0			
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																				1	0		0x10							Reset			

I2S_LC_FIFO_TIMEOUT 配置 FIFO 的超时阈值。FIFO Hung 计数器在 I2S 处于错误状态，并且每次分频滴答计数器达到滴答计数器的阈值时会加一。FIFO Hung 计数器等于 FIFO 的超时阈值时，将触发 [I2S_TX_HUNG_INT](#) 中断或 [I2S_RX_HUNG_INT](#) 中断。(R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT 配置分频滴答计数器的阈值。分频滴答计数器会在 I2S 处于错误状态，并且每个系统 (APB) 时钟上升沿加一。计数器值大于等于 $88000/2^{I2S\_LC\_FIFO\_TIMEOUT\_SHIFT}$ 时，复位滴答计数器。(R/W)

I2S_LC_FIFO_TIMEOUT_ENA 配置是否使能 FIFO 超时。

- 0: 禁用
- 1: 使能

(R/W)

Register 27.17. I2S_CONF_SIGLE_DATA_REG (0x0068)

I2S_SINGLE_DATA																														
31																														0
0																														
Reset																														

I2S_SINGLE_DATA 配置用于发送的通道常量数据。(R/W)

Register 27.18. I2S_STATE_REG (0x006C)

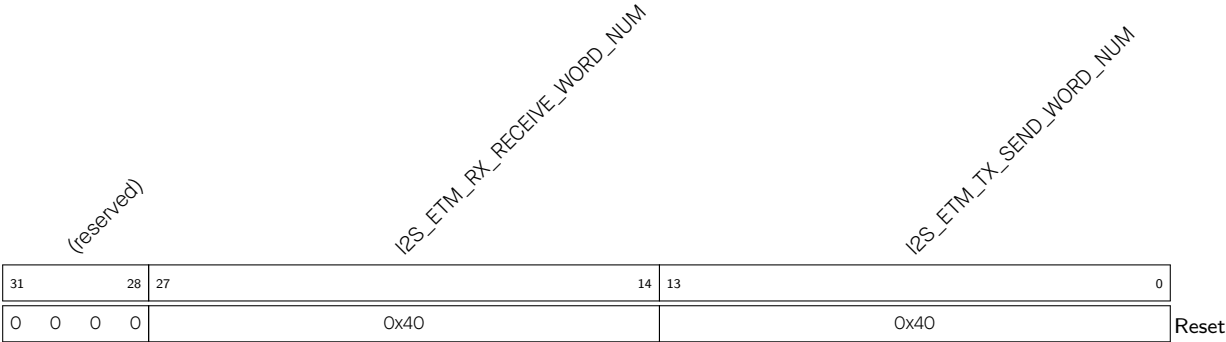
(reserved)																															I2S_TX_IDLE			
31																															1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	Reset

I2S_TX_IDLE 表示 I2S TX 单元的工作状态。

- 0: I2S TX 单元处于工作状态
- 1: I2S TX 单元处于空闲状态

(RO)

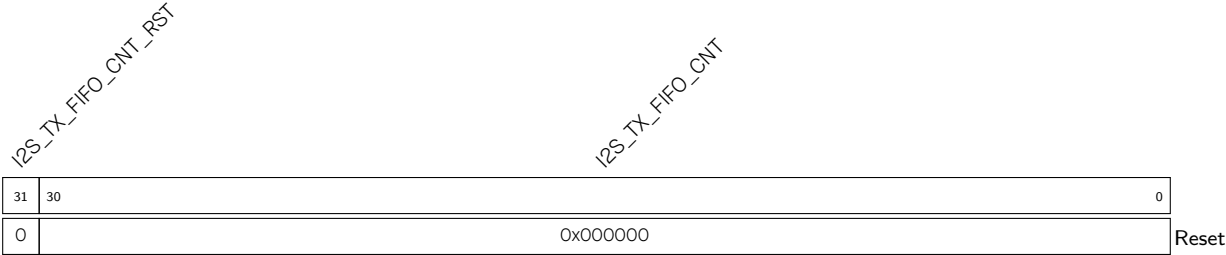
Register 27.19. I2S_ETM_CONF_REG (0x0070)



I2S_ETM_TX_SEND_WORD_NUM 配置触发 [I2S_TX_X_WORDS_SENT](#) 事件的阈值。当发送字数为 I2S_ETM_TX_SEND_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

I2S_ETM_RX_RECEIVE_WORD_NUM 配置触发 [ETM I2S_RX_X_WORDS_RECEIVED](#) 事件的阈值。当接收字数为 I2S_ETM_RX_RECEIVE_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

Register 27.20. I2S_FIFO_CNT_REG (0x0074)



I2S_TX_FIFO_CNT 配置 TX FIFO 计数值。(RO)

I2S_TX_FIFO_CNT_RST 配置是否复位 TX FIFO 计数器。

- 0: 无效
- 1: 复位

(WT)

Register 27.21. I2S_BCK_CNT_REG (0x0078)

																																				0	
31	30																																				
0		0x000000																																			Reset

I2S_TX_BCK_CNT 配置 TX BCK 计数值。(RO)

I2S_TX_BCK_CNT_RST 配置是否复位 TX BCK 计数器。

0: 无效

1: 复位

(WT)

Register 27.22. I2S_CLK_GATE_REG (0x007C)

(reserved)																																		I2S_CLK_EN	
31																																		1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_CLK_EN 配置是否启用时钟门控。

0: 禁用

1: 启用

(R/W)

Register 27.23. I2S_DATE_REG (0x0080)

(reserved)																															I2S_DATE																																						
31				28				27																															0																														
0				0				0				0				0x2307190																															Reset																						

I2S_DATE 版本控制寄存器。(R/W)

第 28 章

USB 串口/JTAG 控制器

ESP32-C5 中包含一个 USB 串口/JTAG 控制器，可用于烧录芯片的外部 flash、读取程序输出的数据以及将调试器连接到正在运行的程序中。任何带有 USB 主机（下文将简称为“主机”）的计算机都可以实现上述功能，无需其他外部组件辅助。

28.1 概述

使用串口通信和 JTAG 调试端口通信来编程和调试 ESP32-C5 项目都是可行的，但也有其限制。首先，串口通信和 JTAG 调试端口通信都占用 IO 管脚，减少了软件中可用于控制外部信号的管脚数量。此外，串口通信和 JTAG 调试端口通信都需借助外部芯片或适配器与主机连接，因此，如果需要整合这两个功能，则需额外使用外部芯片或调试适配器。

为解决上述问题，ESP32-C5 提供了一个 USB 串口/JTAG 控制器，同时集成 USB-串口转换器和 USB-JTAG 适配器功能。由于该模块仅使用 USB 2.0 所需的两条数据线直接连接外部 USB 主机，因此 ESP32-C5 仅需占用两个管脚用于调试。

28.2 主要特性

USB 串口/JTAG 控制器具有如下特性：

- 支持 USB 全速标准；集成 CDC-ACM（通信设备类抽象控制模型）和 JTAG 适配器固定功能
- CDC-ACM：
 - 配置虚拟串行功能，在大多数现代操作系统上可实现即插即用
 - 支持主机控制芯片复位和进入下载模式
- JTAG 适配器：
 - 支持使用紧凑的 JTAG 指令实现与 CPU 调试内核的快速通信
- 共配置一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点，可实现最大 64 字节的数据载荷
- 集成内部 PHY：基本无需其他外部组件连接主机计算机

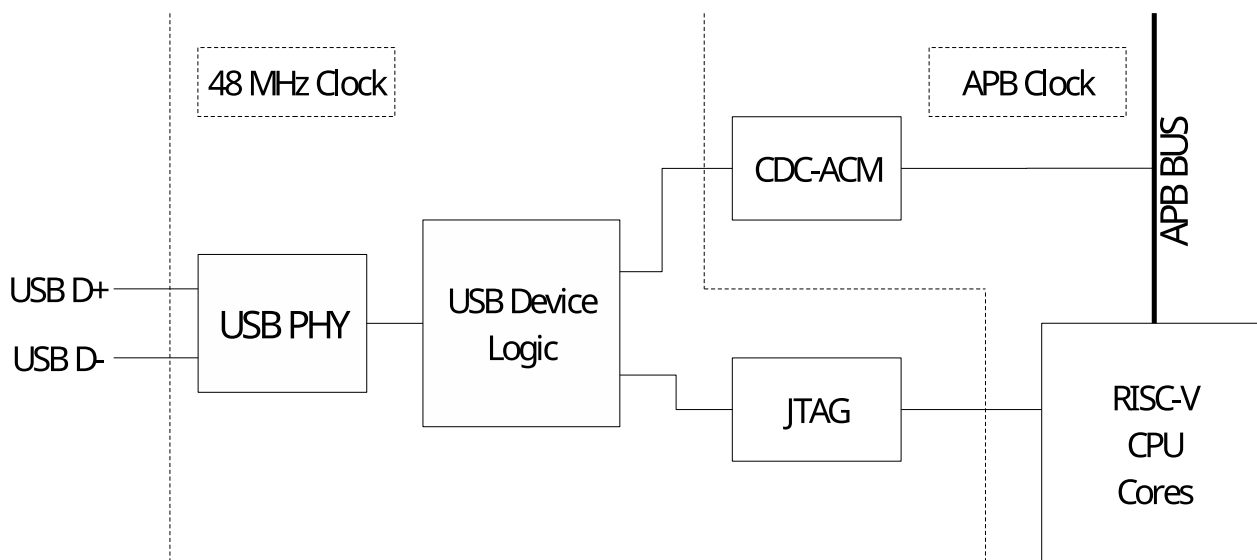


图 28.2-1. USB 串口/JTAG 控制器高层框图

如图 28.2-1 所示，USB 串口/JTAG 控制器包含 USB PHY、USB 设备接口、JTAG 命令处理器、响应捕捉单元以及若干个 CDC-ACM 寄存器。PHY 和设备接口使用主 PLL (BBPLL) 时钟产生的 48 MHz 时钟作为时钟源，CDC-ACM 块中软件可访问的部分则使用 APB_CLK 作为时钟源。JTAG 命令处理器与 ESP32-C5 主处理器中的 JTAG 调试单元相连；CDC-ACM 寄存器则连接至 APB 总线，主 CPU 上运行的软件可对其进行读写访问。

请注意，USB 串口/JTAG 控制器仅支持 USB 2.0 全速标准 (12 Mbps)，不支持 USB 2.0 标准的其他模式（如 480 Mbps 的高速模式）。

图 28.2-2 展示了 USB 串口/JTAG 控制器中 USB 部分的内部详细结构。USB 串口/JTAG 控制器由一个 USB 2.0 全速设备组成，包含一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点。这些端点共同组成该复合型 USB 设备，具体可分为 CDC-ACM USB 设备和实现 JTAG 接口功能的供应商特定设备。JTAG 接口直接与芯片的 RISC-V CPU 的调试接口相连，可对运行在该 CPU 上的程序进行调试。同时，CDC-ACM 中包含一组寄存器，CPU 上运行的程序可对其进行读写操作。此外，芯片上的 ROM 启动代码可允许用户通过使用该接口重新烧录 flash。

说明：

如需确定某个功能使用的具体端点，请解析设备返回的相关 USB 描述符。这样可以保证与功能相同但实现方式不同的设备兼容。

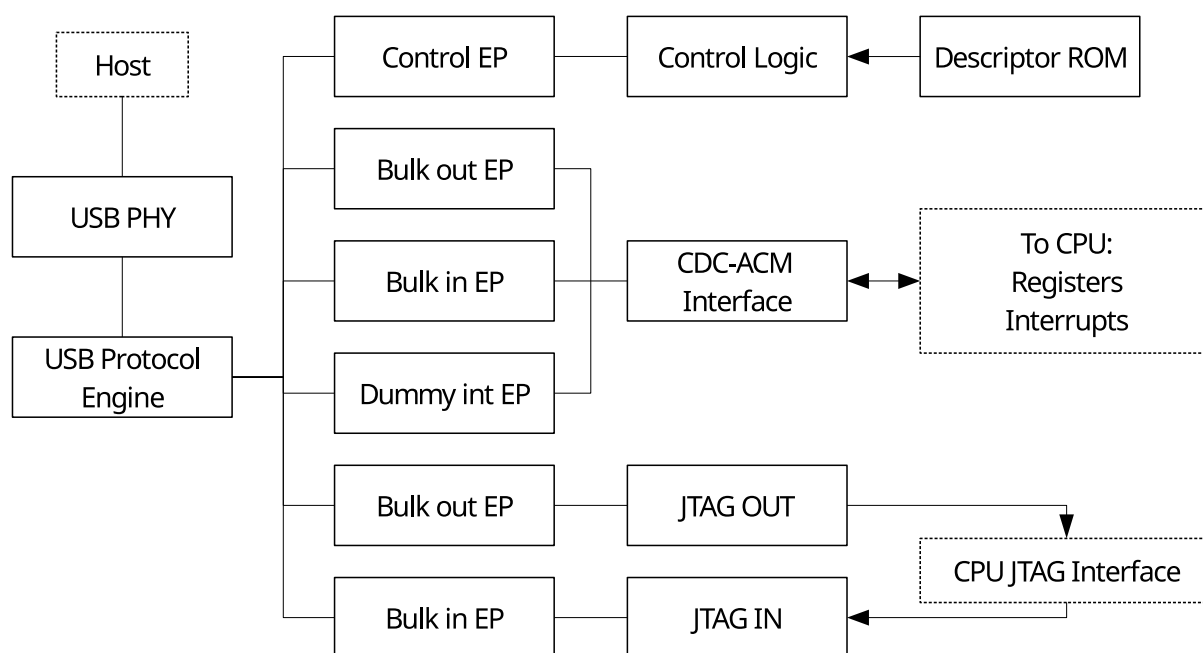


图 28.2-2. USB 串口/JTAG 控制器框图

28.3 功能描述

USB 串口/JTAG 控制器一边与 USB 主机处理器连接，另一边与 CPU 调试硬件以及通过 CDC-ACM 端口通信的软件连接。

28.3.1 CDC-ACM USB 接口描述

CDC-ACM 接口遵循标准 USB CDC-ACM 类别进行虚拟串口通信，包含一个虚拟中断端点（不会发送任何事件，无使用需求）以及一个批量输入端点 (Bulk IN) 和一个批量输出端点 (Bulk OUT) 进行数据接收和发送。这些端点一次可以处理最高 64 字节的数据包，实现高吞吐量。CDC-ACM 为标准的 USB 设备类型，主机一般无需任何特殊安装程序就能正常工作，也就是说，当 USB 调试设备正确连接至主机时，操作系统应能在片刻后显示新的串口信息。

CDC-ACM 接口可以接收以下标准 CDC-ACM 控制请求：

表 28.3-1. 标准 CDC-ACM 控制请求

命令	操作
SEND_BREAK	接收但忽略（虚拟命令）
SET_LINE_CODING	接收，发送值在软件中可读
GET_LINE_CODING	默认返回 9600 baud，无奇偶校验，8 个数据位，1 个停止位（可通过软件更改）
SET_CONTROL_LINE_STATE	设置 RTC/DTR 线的状态，如表 28.3-2 所示

除了通用的通信之外，CDC-ACM 接口还可以复位 ESP32-C5 并选择使其进入下载模式，从而烧录新的固件。这一功能可通过设置虚拟串口的 RTS 和 DTR 线来实现。

表 28.3-2. CDC-ACM 中 RTS 和 DTR 的设置

RTS	DTR	操作
0	0	清除下载模式标志
0	1	置位下载模式标志
1	0	复位 ESP32-C5
1	1	无操作

请注意，当 ESP32-C5 复位时，如果下载模式标志已置位，则 ESP32-C5 重启时将直接进入下载模式；如果下载模式标志已清除，则 ESP32-C5 将从 flash 启动。具体操作流程，请参见章节 28.5。除此之外，也可以通过烧写相应 eFuse 来禁用上述功能，详细信息请参见章节 2 eFuse 控制器 (eFuse) [to be added later]。

28.3.2 CDC-ACM 固件接口描述

由于 USB 串口/JTAG 控制器与 ESP32-C5 的内部 APB 总线相连，因此 CPU 可直接与该模块交互，主要用于对连接主机上的虚拟串口进行读写操作。

CPU 向主机发送并从主机接收 0 ~ 64 字节大小的 USB CDC-ACM 串口数据包。主机已积累足够多的 CDC-ACM 数据时，将向 CDC-ACM 的接收端点发送一个数据包。如果 USB 串行/JTAG 控制器中有空闲缓冲区，该缓冲区将接收这一数据包。反之，主机会定期检查 USB 串口/JTAG 控制器内是否有待向主机发送的数据包，如果有，主机将接收这个数据包。

固件可通过以下两种方式之一获知是否有来自主机的新数据：第一，只要缓冲区中还有来自主机的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位将保持为 1；第二，如果有新的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断将被触发。当新数据可用时，固件可通过重复从 USB_SERIAL_JTAG_EP1_REG 读取字节来获取该数据。读取每个字节后，可通过检查 USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位来看是否还有其他可读取的数据，从而确定需要读取的总字节数。读取完所有数据后，USB 设备会自动做好准备，以接收来自主机的新数据包。

一般来说，发送数据后，连接的主机会阻塞，直到固件通过上述方法之一读取数据。然而，也可能存在某些固件根本不读取数据的情况。通常来说，用于与独立 USB 串口转换器通信的主机程序不会出现这种情况，例如，程序会停止执行直到数据被读取。ESP32-C5 的 USB 串口/JTAG 转换器可以模拟独立 USB 串口/JTAG 转换器的行为：通过向 USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX 写入超时时间并向 USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN 写入 1，任何未读取的数据将在设定的超时时间后自动刷新。

当固件需要发送数据时，可将待发送数据置于发送缓冲区并触发刷写，从而使主机以 USB 数据包的形式接收该数据。在此之前，需确保发送缓冲区有可用空间以存储待发送数据。固件可通过读取 USB_REG_SERIAL_IN_EP_DATA_FREE 位检查发送缓冲区是否有可用空间；当该值为 1 时，发送缓冲区有可用空间。此时，固件可通过向 USB_SERIAL_JTAG_EP1_REG 寄存器中写入字节，从而向缓冲区中写入数据。但是，数据写入后并不会立即触发向主机发送数据，需首先对缓冲区执行刷写操作。刷写操作后，整个缓冲区将准备好被 USB 主机立即接收。可通过两种方式触发刷写：1) 将第 64 个字节写入缓冲区后，USB 硬件会自动将缓冲区刷写到主机；2) 固件可通过向 USB_REG_SERIAL_WR_DONE 写入 1 来触发刷写。

不论以何种方式触发刷写操作，在此期间固件都无法向缓冲区写入数据，直到缓冲区中的所有数据都被主机读取完成。主机读取完成后，将触发 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断，此时可向缓冲区内写入新的 64 字节数据。

在软件中可以处理一些带外串行请求，例如主机设置 DTR 和 RTS，以及改变线路状态。如果 CDC-ACM 接口收到 SET_LINE_CODING 请求，可将外设配置为触发 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断，此时可以从 USB_SERIAL_JTAG_SET_LINE_CODE_WO_REG 寄存器读取线路编码。类似的，如果

SET_CONTROL_LINE_STATE 请求导致线路状态改变，将触发 `USB_SERIAL_JTAG_RTS_CHG_INT` 和 `USB_SERIAL_JTAG_DTR_CHG_INT` 中断。此后，软件可以通过 `USB_SERIAL_JTAG_RTS` 和 `USB_SERIAL_JTAG_DTR` 位来读取具体状态。注意，如前所述，某些 RTS/DTR 序列会导致 ESP32-C5 硬件复位。软件可以通过置位 `USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS` 位来禁止硬件识别这些 DTR/RTS 序列，从而允许软件自由解读信号。

最后，主机可以通过 `GET_LINE_CODING` 读取当前线路状态。此事件将回送 `USB_SERIAL_JTAG_GET_LINE_CODE_WO_REG` 寄存器中的数据，并触发 `USB_SERIAL_JTAG_GET_LINE_CODE_INT` 中断。

28.3.3 USB-JTAG 接口：JTAG 命令处理器

USB-JTAG 接口使用一种供应商特定接口类型实现命令解析的功能。它由两个端点组成：一个用于接收命令，一个用于发送响应。此外，一些对时效性要求不高的命令也可以作为控制请求发出。

JTAG 命令处理器负责解析从主机到 JTAG 接口的命令。JTAG 命令处理器内部包含一个全四线 JTAG 总线，包括发送信号到 RISC-V CPU 的 TCK、TMS 和 TDI 输出线，以及从 CPU 返回信号至 JTAG 响应捕捉单元的 TDO 线。这些信号都符合 IEEE 1149.1 JTAG 标准。此外，还有一条 SRST 线用于复位 ESP32-C5。

另外，软件也可以置位 `USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN`，将这些信号通过 GPIO 交换矩阵连接至 ESP32-C5 的 IO 焊盘上，此操作也将允许通过 USB 串口/JTAG 控制器对外部设备进行调试。

JTAG 命令处理器会将每个接收到的半字节（4 位）解析为一条命令。由于 USB 以 8 位为一个字节接收数据，这就意味着每个字节中都包含两条命令。USB 命令处理器将先解析高 4 位字节，然后再解析低 4 位字节。这些命令用于控制内部 JTAG 总线的 TCK、TMS、TDI、SRST 线，以及向 JTAG 响应捕捉单元发出信号，说明需要捕捉 TDO 线（由 CPU 调试逻辑驱动）的状态。

JTAG 总线中，TCK、TMS、TDI 和 TDO 线直接与 RISC-V CPU 的 JTAG 调试逻辑相连。上文提到的 SRST 线则与 ESP32-C5 数字电路中的复位逻辑相连，该线电平拉高，芯片将进行芯片复位。请注意，SRST 线并不会对 USB 串口/JTAG 控制器模块产生影响。

1 个半字节中可包含以下命令：

表 28.3-3. 半字节中的命令

位	3	2	1	0
CMD_CLK	0	cap	tms	tdi
CMD_RST	1	0	0	srst
CMD_FLUSH	1	0	1	0
CMD_RSV	1	0	1	1
CMD_REP	1	1	R1	R0

- CMD_CLK：将 TDI 和 TMS 设置为指示值，并在 TCK 上发出一个时钟脉冲。如果 CAP 位为 1，将指示 JTAG 响应捕捉单元捕捉 TDO 线的状态。该指令构成了 JTAG 通信的基础。
- CMD_RST：将 SRST 线的状态设置为指示值。该命令可用于复位 ESP32-C5。
- CMD_FLUSH：指示 JTAG 响应捕捉单元对接收到的所有位的缓冲区进行刷写操作，以便主机读取这些位。请注意在某些情况下，一次 JTAG 通信会结束于第奇数个命令，即结束于第奇数个半字节。此时，可重复执行该命令直到获得偶数个半字节，使其组成整数个字节。
- CMD_RSV：该版本中保留。ESP32-C5 在接收到该命令时会自动忽略。

- CMD_REP: 将上一条指令 (非 CMD_REP) 重复一定次数, 以压缩多次重复 CMD_CLK 的命令流。因此, CMD_CLK 命令后可能跟随着多个 CMD_REP。一次 CMD_REP 命令产生的重复次数可表示为 $repetition\_count = (R1 \times 2 + R0) \times (4^{cmd\_rep\_count})$, 其中 cmd_rep_count 表示该命令之前的 CMD_REP 数量。请注意, CMD_REP 仅用于重复 CMD_CLK 命令。也就是说, 如果在 CMD_FLUSH 后使用该命令, USB 设备将无法响应, 需进行 USB 复位后才可恢复正常。

28.3.4 USB-JTAG 接口: CMD_REP 使用示例

下列命令用于演示如何使用 CMD_REP 命令。请注意, 该示例中每个命令为半字节, 命令流的每个字节为 0x0D 0x5E 0xCF。

1. 0x0 (CMD_CLK: cap=0, tdi=0, tms=0)
2. 0xD (CMD_REP: R1=0, R0=1)
3. 0x5 (CMD_CLK: cap=1, tdi=0, tms=1)
4. 0xE (CMD_REP: R1=1, R0=0)
5. 0xC (CMD_REP: R1=0, R0=0)
6. 0xF (CMD_REP: R1=1, R0=1)

每一步骤的具体操作为:

1. TCK 上发出一个时钟脉冲, TDI 和 TMS 设置为 0。未捕捉到任何数据。
2. TCK 上再次发出 $(0 \times 2 + 1) \times (4^0) = 1$ 个时钟脉冲, 其他设置与步骤 1 相同。
3. TCK 上发出一个时钟脉冲, TDI 设置为 0, TMS 设置为 1。捕捉到 TDO 线上的数据。
4. TCK 上再次发出 $(1 \times 2 + 0) \times (4^0) = 2$ 个时钟脉冲, 其他设置与步骤 3 相同。
5. 未发生任何动作: $(0 \times 2 + 0) \times (4^1) = 0$ 。请注意, 该操作将增加下一步骤中的 cmd_rep_count 数值。
6. TCK 上再次发出 $(1 \times 2 + 1) \times (4^2) = 48$ 个时钟脉冲, 其他设置与步骤 3 相同。

换言之, 该命令流示例的操作结果等同于执行两次命令 1, 然后执行 51 次命令 3 的效果。

28.3.5 USB-JTAG 接口: 响应捕捉单元

响应捕捉单元首先读取内部 JTAG 总线的 TDO 线, 并在命令处理器执行 CMD_CLK (cap=1) 命令时捕捉 TDO 线的值, 然后将这个值放入内部移位寄存器中, 且在接收到 8 位时向 USB 缓冲区写入 1 个字节。这 8 位中的最低有效位即为最先从 TDO 线读取的值。

一旦接收到 64 字节 (512 位) 数据或执行 CMD_FLUSH 命令后, 响应捕捉单元将使缓冲区可被主机接收。请注意, USB 逻辑的接口为双缓冲。因此, 只要 USB 的吞吐量充足, 响应捕捉单元就可以随时接收更多数据, 即当一个缓冲区等待发送给主机时, 另一个缓冲区可以继续接收数据。当主机从缓冲区成功接收数据且响应捕捉单元对缓冲区执行刷写操作后, 这两个缓冲区便可以交换位置。

同时, 这也意味着一个命令流可导致最多 128 字节 (若该命令流中有刷写命令, 则该数字会减小) 的捕捉数据生成, 而无需主机主动接收这些数据。如果还是生成了超过该阈值数量的捕捉数据, 则命令流将暂停, 且设备在这些数据被读取之前不会接收其他命令。

另需注意, 一般情况下, 响应捕捉单元会尽量不发送 0 字节响应。例如, 当发送一系列 CMD_FLUSH 命令后, 并不会产生一系列 0 字节 USB 响应。但在当前版本中, 一些特殊情况下也可能产生 0 字节响应。建议直接忽略这些响应信息。

28.3.6 USB-JTAG 接口：控制传输请求

除命令处理器和响应捕捉单元之外，USB-JTAG 接口也可接收一些控制请求，具体如下表所示：

表 28.3-4. USB-JTAG 控制请求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000000b	0 (VEND_JTAG_SETDIV)	[divider]	接口	0	None
01000000b	1 (VEND_JTAG_SETIO)	[iobits]	接口	0	None
11000000b	2 (VEND_JTAG_GETTDO)	0	接口	1	[iostate]
10000000b	6 (GET_DESCRIPTOR)	0x2000	0	256	[jtag cap desc]

- VEND_JTAG_SETDIV：设置使用的分频器。该命令直接影响 TCK 时钟脉冲的持续时间。TCK 时钟脉冲来自于由内部分频器向下分频得到的 48 MHz 基准时钟。该控制请求允许主机设置此内部分频器。请注意，该分频器在启动时的初始值为 2，即 TCK 时钟速率一般为 24 MHz。
- VEND_JTAG_SETIO：跳过 JTAG 命令处理器，直接将内部 TDI、TDO、TMS 和 SRST 线设置为指定值。这些指定值在 wValue 字段中以 11'b0, srst, trst, tck, tms, tdi 的格式编码。
- VEND_JTAG_GETTDO：跳过 JTAG 响应捕捉单元，直接读取内部 TDO 信号。该请求返回 1 个字节，其中的最低有效位代表 TDO 线的状态。
- GET_DESCRIPTOR：为标准 USB 请求，该请求也可使用供应商专用的 0x2000 wValue 获取 JTAG 功能描述符。该请求返回一定字节，具体字节数所代表的 USB-JTAG 适配器功能如表 28.3-5 所示。这一固定结构允许主机软件自动支持未来的硬件新版本，无需再次更新。

ESP32-C5 包含的 JTAG 功能描述符如下表所示。请注意，所有 16 位值都为小端序存储。

表 28.3-5. JTAG 功能描述符

字节	数值	描述
0	1	JTAG 协议功能结构的版本
1	10	JTAG 协议功能长度
2	1	结构类型：1 代表高速功能结构类型
3	8	该高速功能结构长度
4 ~ 5	4800	以 10 kHz 为增量的 JTAG 基准时钟速度，其最大速度为该值的一半
6 ~ 7	1	最小分频系数，可通过 VEND_JTAG_SETDIV 设置
8 ~ 9	255	最大分频系数，可通过 VEND_JTAG_SETDIV 设置

28.4 中断

ESP32-C5 的 USB 串口/JTAG 控制器的下列内部中断源可生成中断信号 USB_SERIAL_JTAG_INTR，并发送给中断矩阵。

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT：JTAG 批量输出端点接收到刷写命令时触发。
- USB_SERIAL_JTAG_SOF_INT：接收到 SOF 帧时触发。
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT：串口输出端点接收到 1 个数据包时触发。
- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT：串口输入端点为空时触发。

- USB_SERIAL_JTAG_PID_ERR_INT: 检测到 PID 错误时触发。
- USB_SERIAL_JTAG_CRC5_ERR_INT: 检测到 CRC5 错误时触发。
- USB_SERIAL_JTAG_CRC16_ERR_INT: 检测到 CRC16 错误时触发。
- USB_SERIAL_JTAG_STUFF_ERR_INT: 检测到位填充错误时触发。
- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT: 输入端点 1 接收到一个 IN 令牌时触发。
- USB_SERIAL_JTAG_USB_BUS_RESET_INT: 检测到 USB 总线复位时触发。
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT: 输出端点 1 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT: 输出端点 2 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_RTS_CHG_INT: USB 串行通道的 RTS 电平改变时触发。
- USB_SERIAL_JTAG_DTR_CHG_INT: USB 串行通道的 DTR 电平改变时触发。
- USB_SERIAL_JTAG_GET_LINE_CODE_INT: 收到 GET_LINE_CODING 请求的电平时触发。
- USB_SERIAL_JTAG_SET_LINE_CODE_INT: 收到 SET_LINE_CODING 请求的电平时触发。

28.5 配置流程

使用 USB 串口/JTAG 控制器之前, 几乎不需要多余的配置。除了主机操作系统已经完成的标准 USB 初始化之外, USB-JTAG 硬件本身不需要进行任何配置。此外, CDC-ACM 虚拟串口在主机端也支持即插即用。

固件方面也几乎不需要初始化。USB 硬件是自初始化的, 在其启动后, 如果固件连接了一台主机并在 CDC-ACM 接口上监听, 除非固件设置了中断处理程序, 否则无需任何特定设置就可以实现前文所述的数据交换。

需要注意的是, 可能会出现主机未连接或 CDC-ACM 虚拟串口未打开的情况。在这种情况下, 发送至主机的数据包永远无法被接收, 发送缓冲区也永远不会为空。因而, 必须对这种情况进行检测并执行超时操作, 以便清楚地检测主机侧的端口是否关闭。

其次, 需要注意 USB 设备依赖于产生 48 MHz USB PHY 时钟的 BBPLL。如果此 PLL 被禁用, USB 通信将停止。

上述情况可能发生在 Deep-sleep 期间。在 Deep-sleep 模式下, USB 串口/JTAG 控制器 (及其连接的 RISC-V CPU) 将完全断电。如果有设备需要在这两种模式下进行调试, 最好使用外部 JTAG 调试器和串行接口。

CDC-ACM 接口还可用于复位芯片, 使其进入或退出下载模式。产生正确的握手信号序列则有些复杂, 因为大多数操作系统仅支持分别设置或重置 DTR 和 RTS, 无法同时进行。此外, 一些驱动程序 (如 Windows 系统上的标准 CDC-ACM 驱动程序) 须先设置 RTS 后才可设置 DTR, 此时用户必须先明确设置 RTS, 才能传播 DTR 的值。推荐遵循以下步骤进行设置。

复位芯片使其进入下载模式:

表 28.5-1. 复位芯片进入下载模式

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	初始化以获取数值
清除 RTS	RTS=0, DTR=0	-
设置 DTR	RTS=0, DTR=1	设置下载模式标志
清除 RTS	RTS=0, DTR=1	传播 DTR
设置 RTS	RTS=1, DTR=1	-
清除 DTR	RTS=1, DTR=0	复位芯片
设置 RTS	RTS=1, DTR=0	传播 DTR
清除 RTS	RTS=0, DTR=0	清除下载标志

复位 SoC 使其从 flash 启动：

表 28.5-2. 复位 SoC 从 flash 启动

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	-
清除 RTS	RTS=0, DTR=0	清除下载标志
设置 RTS	RTS=1, DTR=0	复位 SoC
清除 RTS	RTS=0, DTR=0	退出复位

28.6 寄存器列表

本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
USB_SERIAL_JTAG_EP1_REG	CDC-ACM 输入/输出端点 FIFO 访问配置寄存器	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	CDC-ACM FIFO 配置寄存器	0x0004	varies
USB_SERIAL_JTAG_CONFO_REG	PHY 硬件配置寄存器	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	PHY 调试寄存器	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	时钟使能控制寄存器	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	存储器配置寄存器	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM 芯片重置寄存器	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	GET_LINE_CODING 命令的 W0	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	GET_LINE_CODING 命令的 W1	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	配置寄存器更新寄存器	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	串口 AFIFO 配置寄存器	0x0064	varies
USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG	USB UART 输出端点超时配置	0x006C	varies
USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG	USB UART 输出端点超时配置	0x0070	R/W
中断寄存器			
USB_SERIAL_JTAG_INT_RAW_REG	中断原始状态寄存器	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	中断状态寄存器	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	中断使能状态寄存器	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	中断清除状态寄存器	0x0014	WT
状态寄存器			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO 状态与控制寄存器	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	接收 SOF 帧索引寄存器	0x0024	RO
USB_SERIAL_JTAG_IN_EPO_ST_REG	输入端点状态信息寄存器	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM 输入端点状态信息寄存器	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM 中断输入端点状态信息寄存器	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG 输入端点状态信息寄存器	0x0034	RO
USB_SERIAL_JTAG_OUT_EPO_ST_REG	输出端点状态信息寄存器	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM 输出端点状态信息寄存器	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG 输出端点状态信息寄存器	0x0040	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	SET_LINE_CODING 命令的 W0	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	SET_LINE_CODING 命令的 W1	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB 总线重置状态寄存器	0x0068	RO
版本寄存器			

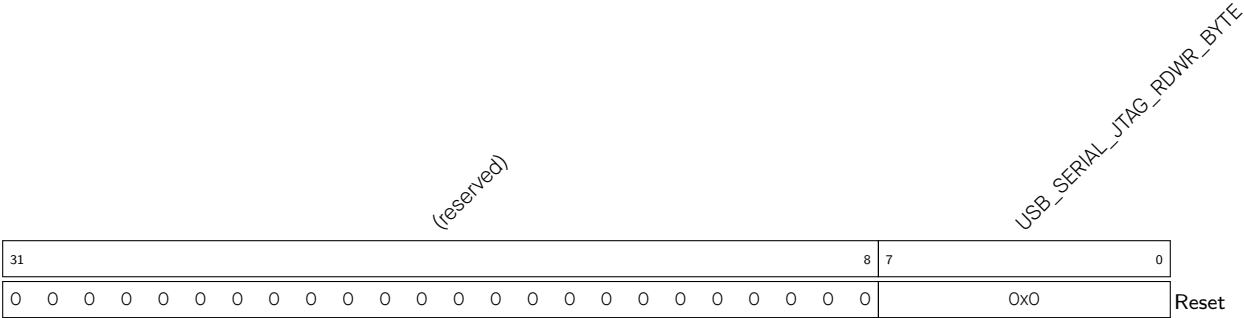
名称	描述	地址	访问
USB_SERIAL_JTAG_DATE_REG	版本寄存器	0x0080	R/W

28.7 寄存器

本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 28.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



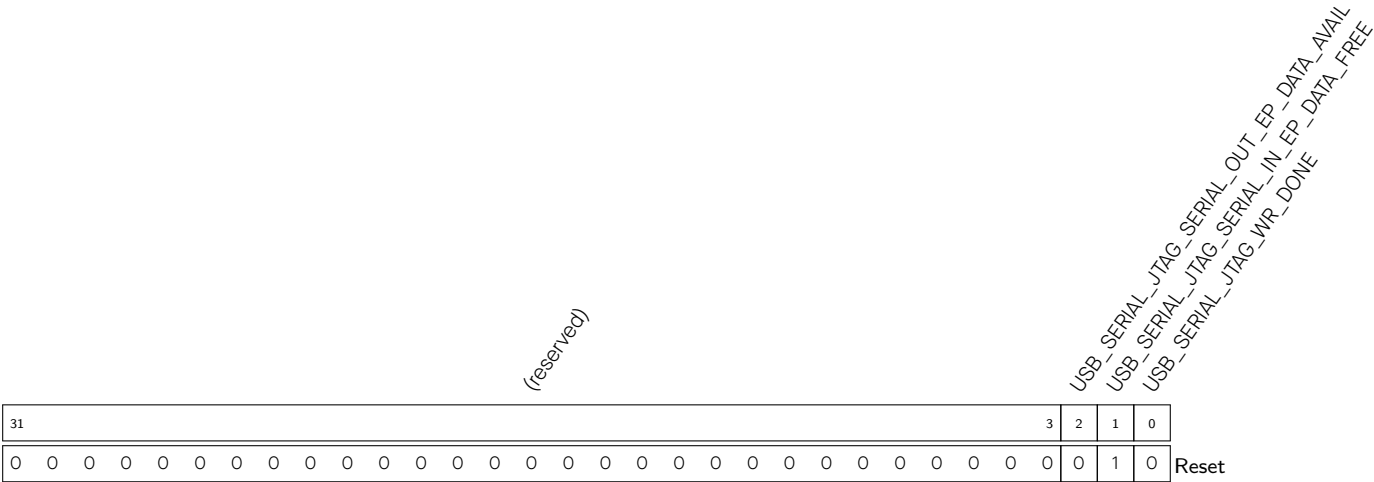
USB_SERIAL_JTAG_RDWR_BYTE 写入该位即为将数据写入 CDC-ACM TX FIFO；读取该位即为从 CDC-ACM RX FIFO 读取一个字节并返回该字节。

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 置位时，用户可向 CDC-ACM TX FIFO 中写入数据（最大 64 字节）。

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 置位时，用户可通过查看 **USB_SERIAL_JTAG_OUT_EP1_WR_ADDR** 的值获知接收到的数据量，然后从 CDC-ACM RX FIFO 中读取这些数据。

(R/W)

Register 28.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE 配置是否表明已完成向 CDC-ACM TX FIFO 写入字节数据。

0: 无效

1: 表明已完成向 CDC-ACM TX FIFO 写入字节数据

该位将保持为 0，直到 USB 主机读取 CDC-ACM TX FIFO 中的数据。

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE 表示 CDC-ACM TX FIFO 是否可以写入数据。

0: CDC-ACM TX FIFO 已写满，不可写入数据

1: CDC-ACM TX FIFO 未写满且可以写入数据

写入 [USB_SERIAL_JTAG_WR_DONE](#) 后，该位将置为 0，直到其中的数据已发送至 USB 主机。

(RO)

USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 表示 CDC-ACM RX FIFO 中是否有数据。

0: CDC-ACM RX FIFO 中没有数据

1: CDC-ACM RX FIFO 中有数据

(RO)

Register 28.3. USB_SERIAL_JTAG_CONFO_REG (0x0018)

(reserved)																31	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	Reset			
																USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN USB_SERIAL_JTAG_USB_PAD_ENABLE USB_SERIAL_JTAG_DM_PULLUP_VALUE USB_SERIAL_JTAG_DM_PULLDOWN USB_SERIAL_JTAG_DP_PULLUP USB_SERIAL_JTAG_DP_PULLDOWN USB_SERIAL_JTAG_PAD_PULLUP USB_SERIAL_JTAG_VREF_OVERRIDE USB_SERIAL_JTAG_VREFL USB_SERIAL_JTAG_VREFH USB_SERIAL_JTAG_EXCHG_PINS USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE																	

USB_SERIAL_JTAG_PHY_SEL 配置选择使用内部 PHY 或外部 PHY。

0: 内部 PHY

1: 外部 PHY

(R/W)

USB SERIAL JTAG EXCHG PINS OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚交换。

0: 不使能

1: 使能

(R/W)

USB SERIAL JTAG EXCHG PINS 配置是否交换 USB D+ 和 D- 管脚。

0: 无效

1: 交换

(R/W)

USB_SERIAL_JTAG_VREFH 配置单端输入高阈值。

0: 1.76 V

1: 1.84 V

2: 1.92 V

3: 2.00 V

(R/W)

USB SERIAL JTAG VREFL 配置单端输入低阈值。

0: 0.80 V

1: 0.88 V

2: 0.96 V

3: 1.04 V

(R/W)

USB SERIAL JTAG VREF OVERRIDE 配置是否使能软件控制输入阈值。

0: 不使能

1: 使能

(R/W)

见下页...

Register 28.3. USB_SERIAL_JTAG_CONFO_REG (0x0018)

[接上页...](#)

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚的上下拉电阻。

0: 不使能

1: 使能

(R/W)

USB_SERIAL_JTAG_DP_PULLUP 配置当 [USB_SERIAL_JTAG_PAD_PULL_OVERRIDE](#) 为 1 时, 是否使能 USB D+ 管脚的上拉电阻。

0: 不使能

1: 使能

(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN 配置当 [USB_SERIAL_JTAG_PAD_PULL_OVERRIDE](#) 为 1 时, 是否使能 USB D+ 管脚的下拉电阻。

0: 不使能

1: 使能

(R/W)

USB_SERIAL_JTAG_DM_PULLUP 配置当 [USB_SERIAL_JTAG_PAD_PULL_OVERRIDE](#) 为 1 时, 是否使能 USB D- 管脚的上拉电阻。

0: 不使能

1: 使能

(R/W)

USB_SERIAL_JTAG_DM_PULLDOWN 配置当 [USB_SERIAL_JTAG_PAD_PULL_OVERRIDE](#) 为 1 时, 是否使能 USB D- 管脚的下拉电阻。

0: 不使能

1: 使能

(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE 配置当 [USB_SERIAL_JTAG_PAD_PULL_OVERRIDE](#) 为 1 时的上拉数值。

0: 2.2 K

1: 1.1 K

(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE 配置是否使能 USB 管脚。

0: 不使能

1: 使能

(R/W)

USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN 配置是否断开 USB_JTAG 和内部 JTAG 的连接。

0: USB_JTAG 连接到 CPU 的内部 JTAG 端口

1: USB_JTAG 和内部 JTAG 之间断开连接, MTMS、MTDI 和 MTCK 为通过 GPIO 交换矩阵的输出, MTDO 为通过 GPIO 交换矩阵的输入

(R/W)

Register 28.4. USB_SERIAL_JTAG_TEST_REG (0x001C)

(reserved)																USB_SERIAL_JTAG_TEST_RX_DM							
																USB_SERIAL_JTAG_TEST_RX_DP							
																USB_SERIAL_JTAG_TEST_RX_RCV							
																USB_SERIAL_JTAG_TEST_TX_DM							
																USB_SERIAL_JTAG_TEST_TX_DP							
																USB_SERIAL_JTAG_TEST_USB_OE							
																USB_SERIAL_JTAG_TEST_ENABLE							
31															7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	

Register 28.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)

Register 0x00000000: USB_SERIAL_JTAG_CLK_EN

31	1	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

USB_SERIAL_JTAG_CLK_EN 配置是否强制打开寄存器的时钟。

0: 支持仅当应用程序向寄存器写入数据时打开时钟

1: 强制打开寄存器的时钟

(R/W)

Register 28.6. USB_SERIAL_JTAG_MEM_CONF_REG (0x0048)

Diagram illustrating the structure of the `USB_OTG_FS_RESET` register:

- Bit 31: (reserved)
- Bits 31-2: (reserved)
- Bit 1: `USB_SERIAL_JTAG_USB_MEM_EN`
- Bit 0: `USB_SERIAL_JTAG_USB_MEM_PD`

The register is labeled **Reset**.

USB_SERIAL_JTAG USB_MEM_PD 配置是否关闭 USB 存储器。

0: 无效

1: 关闭

(R/W)

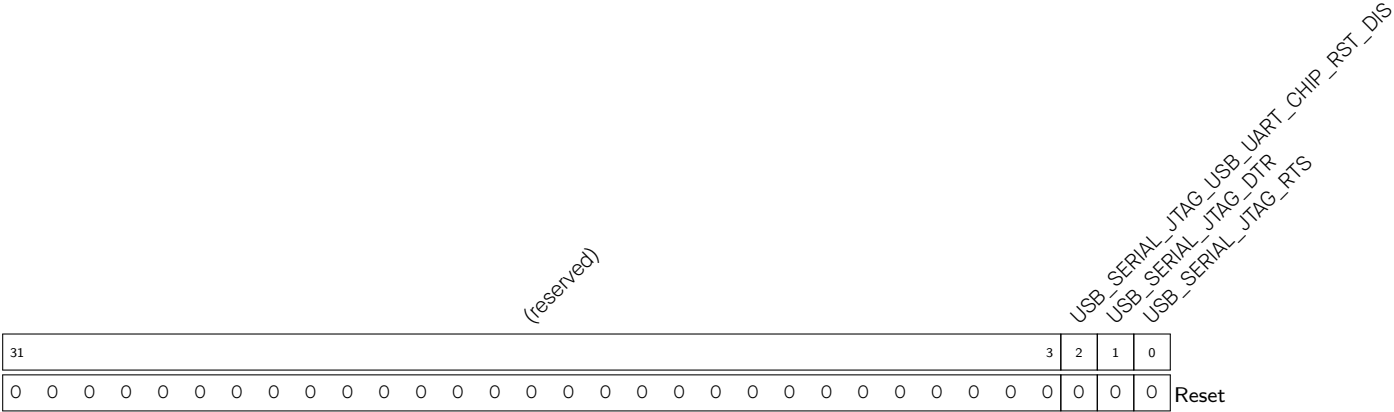
USB_SERIAL_JTAG_USB_MEM_CLK_EN 配置是否强制对 USB 存储器进行时钟分频。

0: 无效

1: 强制

(R/W)

Register 28.7. USB_SERIAL_JTAG_CHIP_RST_REG (0x004C)



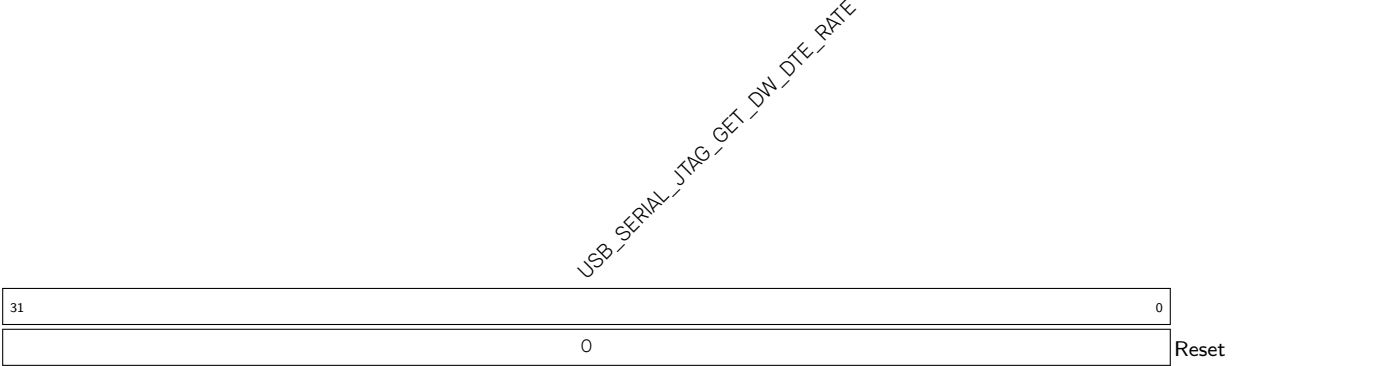
USB_SERIAL_JTAG_RTSS 表示最近的 SET_LINE_CODING 命令所设定的 RTS 信号的状态。(RO)

USB_SERIAL_JTAG_DTR 表示最近的 SET_LINE_CODING 命令所设定的 DTR 信号的状态。(RO)

USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 配置是否禁用来自 USB 串行通道的芯片复位。

- 0: 无效
- 1: 禁用
- (R/W)

Register 28.8. USB_SERIAL_JTAG_GET_LINE_CODE_WO_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE 配置软件设置的 dwDTERate 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 28.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

(reserved)								USB_SERIAL_JTAG_GET_BCHAR_FORMAT								USB_SERIAL_JTAG_GET_BPARITY_TYPE								USB_SERIAL_JTAG_GET_BDATA_BITS							
31								24	23						16	15				8	7								0		
0	0	0	0	0	0	0	0	0							0					0							Reset				

USB_SERIAL_JTAG_GET_BDATA_BITS 配置软件设置的 bDataBits 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BPARITY_TYPE 配置软件设置的 bParityType 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BCHAR_FORMAT 配置软件设置的 bCharFormat 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 28.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)

(reserved)																															USB_SERIAL_JTAG_CONFIG_UPDATE									
31																															1	0								
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																															0	Reset								

USB_SERIAL_JTAG_CONFIG_UPDATE 配置是否将配置寄存器的值从 APB 时钟域更新到 48 MHz 时钟域。
0: 无效
1: 更新
(WT)

Register 28.11. USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG (0x0064)

(reserved)																												USB_SERIAL_JTAG_SERIAL_IN_AFIFO_WFULL USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_EMPTY USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_REMPTY USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_RD USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_WR							
31																												6	5	4	3	2	1	0	Reset
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																												0	1	0	0	0	0		

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_WR 配置是否复位 CDC_ACM IN 异步 FIFO 写入时钟域。
0: 无效
1: 复位
(R/W)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_RD 配置是否复位 CDC_ACM IN 异步 FIFO 读取时钟域。
0: 无效
1: 复位
(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_WR 配置是否复位 CDC_ACM OUT 异步 FIFO 写入时钟域。
0: 无效
1: 复位
(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_RD 配置是否复位 CDC_ACM OUT 异步 FIFO 读取时钟域。

0: 无效

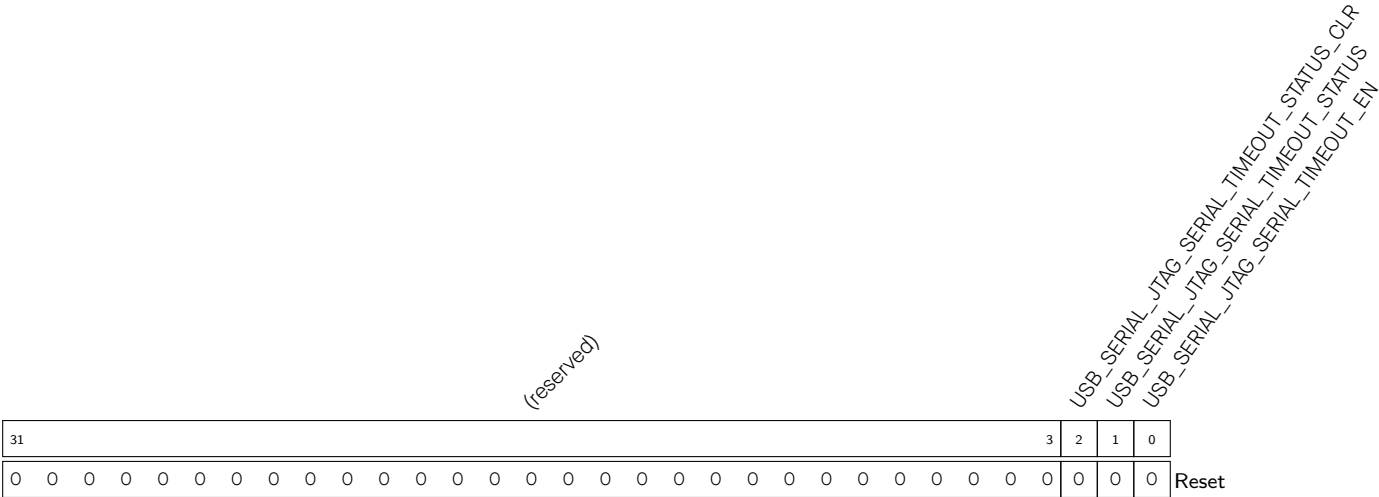
1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_EMPTY 表示 CDC_ACM_OUT 读取时钟域的异步 FIFO 空信号。(RO)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_WFULL 表示 CDC_ACM IN 写入时钟域的异步 FIFO 满信号。
(RO)

Register 28.12. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT0_REG (0x006C)



USB_SERIAL_JTAG_SERIAL_TIMEOUT_EN 使能 USB 串口输出端点超时功能。超时事件发生时，串口输出端点缓冲区自动清除，[USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS](#) 被置位。

0：禁用超时，缓冲区不会自动清除

1：使能超时

(R/W)

USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS 超时状态。

0：未发生超时

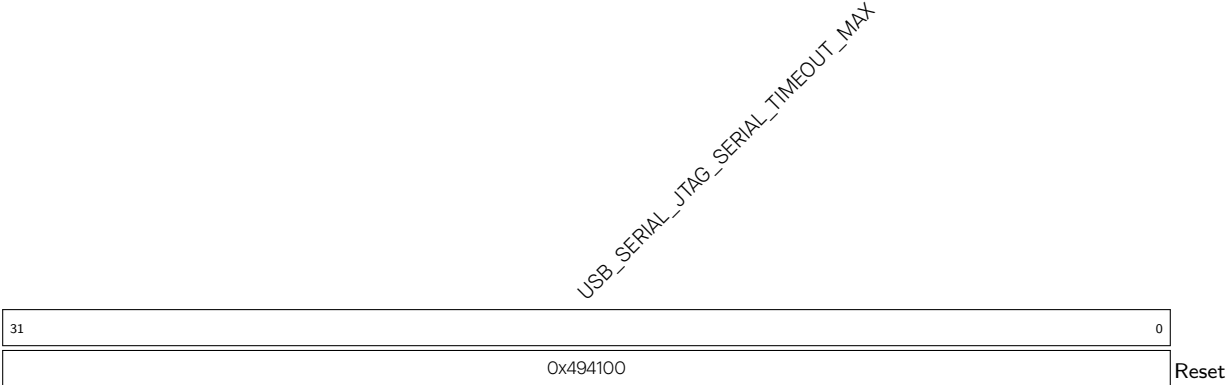
1：串口输出端点触发了超时事件

(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS_CLR 写1清除 [USB_SERIAL_JTAG_SERIAL_TIMEOUT_STATUS](#)。

(WT)

Register 28.13. USB_SERIAL_JTAG_SERIAL_EP_TIMEOUT1_REG (0x0070)



USB_SERIAL_JTAG_SERIAL_TIMEOUT_MAX USB 串口输出端点的最大超时数值，表示等待芯片读取数据的最长等待时间，单位为 20.83 ns。(R/W)

Register 28.14. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

(reserved)																USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_DTR_CHG_INT_RAW USB_SERIAL_JTAG_RTS_CHG_INT_RAW USB_SERIAL_JTAG_OUT_EP2_INT_RAW USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW USB_SERIAL_JTAG_STUFF_ERR_INT_RAW USB_SERIAL_JTAG_CRC16_ERR_INT_RAW USB_SERIAL_JTAG_PID_ERR_INT_RAW USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW USB_SERIAL_JTAG_SOF_INT_RAW USB_SERIAL_JTAG_IN_FLUSH_INT_RAW																				
31																16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	Reset											

USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW [USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW [USB_SERIAL_JTAG_SOF_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW [USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW [USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW [USB_SERIAL_JTAG_PID_ERR_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW [USB_SERIAL_JTAG_CRC5_ERR_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW [USB_SERIAL_JTAG_CRC16_ERR_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW [USB_SERIAL_JTAG_STUFF_ERR_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW [USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW [USB_SERIAL_JTAG_USB_BUS_RESET_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW [USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW [USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT](#) 的原始中断状态。(R/WTC/SS)

见下页...

Register 28.14. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

[接上页...](#)

USB_SERIAL_JTAG_RTS_CHG_INT_RAW [USB_SERIAL_JTAG_RTS_CHG_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW [USB_SERIAL_JTAG_DTR_CHG_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW [USB_SERIAL_JTAG_GET_LINE_CODE_INT](#) 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW [USB_SERIAL_JTAG_SET_LINE_CODE_INT](#) 的原始中断状态。(R/WTC/SS)

Register 28.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

(reserved)																																USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST USB_SERIAL_JTAG_DTR_CHG_INT_ST USB_SERIAL_JTAG_RTS_CHG_INT_ST USB_SERIAL_JTAG_OUT_EP2_INT_ST USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST USB_SERIAL_JTAG_STUFF_ERR_INT_ST USB_SERIAL_JTAG_CRC16_ERR_INT_ST USB_SERIAL_JTAG_PID_ERR_INT_ST USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST USB_SERIAL_JTAG_SOF_INT_ST USB_SERIAL_JTAG_IN_FLUSH_INT_ST																	
31																16																15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0																0																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SOF_INT_ST USB_SERIAL_JTAG_SOF_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST USB_SERIAL_JTAG_PID_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST USB_SERIAL_JTAG_CRC5_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST USB_SERIAL_JTAG_CRC16_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST USB_SERIAL_JTAG_STUFF_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST USB_SERIAL_JTAG_USB_BUS_RESET_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

见下页...

Register 28.15. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

[接上页...](#)

USB_SERIAL_JTAG_RTS_CHG_INT_ST [USB_SERIAL_JTAG_RTS_CHG_INT](#) 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST [USB_SERIAL_JTAG_DTR_CHG_INT](#) 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST [USB_SERIAL_JTAG_GET_LINE_CODE_INT](#) 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST [USB_SERIAL_JTAG_SET_LINE_CODE_INT](#) 的屏蔽中断状态。(RO)

Register 28.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

(reserved)																USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA USB_SERIAL_JTAG_DTR_CHG_INT_ENA USB_SERIAL_JTAG_RTS_CHG_INT_ENA USB_SERIAL_JTAG_OUT_EP2_INT_ENA USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA USB_SERIAL_JTAG_STUFF_ERR_INT_ENA USB_SERIAL_JTAG_CRC16_ERR_INT_ENA USB_SERIAL_JTAG_PID_ERR_INT_ENA USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA USB_SERIAL_JTAG_SOF_INT_ENA USB_SERIAL_JTAG_IN_FLUSH_INT_ENA																
31																16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA 写1使能 [USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_SOF_INT_ENA 写1使能 [USB_SERIAL_JTAG_SOF_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA 写1使能 [USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA 写1使能 [USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA 写1使能 [USB_SERIAL_JTAG_PID_ERR_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA 写1使能 [USB_SERIAL_JTAG_CRC5_ERR_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA 写1使能 [USB_SERIAL_JTAG_CRC16_ERR_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA 写1使能 [USB_SERIAL_JTAG_STUFF_ERR_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA 写1使能 [USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA 写1使能 [USB_SERIAL_JTAG_USB_BUS_RESET_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA 写1使能 [USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA 写1使能 [USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT](#) 中断。(R/W)

见下页...

Register 28.16. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

[接上页...](#)

USB_SERIAL_JTAG_RTS_CHG_INT_ENA 写1使能 [USB_SERIAL_JTAG_RTS_CHG_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA 写1使能 [USB_SERIAL_JTAG_DTR_CHG_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA 写1使能 [USB_SERIAL_JTAG_GET_LINE_CODE_INT](#) 中断。(R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA 写1使能 [USB_SERIAL_JTAG_SET_LINE_CODE_INT](#) 中断。(R/W)

Register 28.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

(reserved)																USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR USB_SERIAL_JTAG_DTR_CHG_INT_CLR USB_SERIAL_JTAG_RTS_CHG_INT_CLR USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR USB_SERIAL_JTAG_STUFF_ERR_INT_CLR USB_SERIAL_JTAG_CRC16_ERR_INT_CLR USB_SERIAL_JTAG_PID_ERR_INT_CLR USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR USB_SERIAL_JTAG_SOF_INT_CLR USB_SERIAL_JTAG_IN_FLUSH_INT_CLR																				
31																16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset											

USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR 写1清除 [USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT](#) 中断。(WT)

USB_SERIAL_JTAG_SOF_INT_CLR 写1清除 [USB_SERIAL_JTAG_SOF_INT](#) 中断。(WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR 写1清除 [USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT](#) 中断。(WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR 写1清除 [USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT](#) 中断。(WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR 写1清除 [USB_SERIAL_JTAG_PID_ERR_INT](#) 中断。(WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR 写1清除 [USB_SERIAL_JTAG_CRC5_ERR_INT](#) 中断。(WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR 写1清除 [USB_SERIAL_JTAG_CRC16_ERR_INT](#) 中断。(WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR 写1清除 [USB_SERIAL_JTAG_STUFF_ERR_INT](#) 中断。(WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR 写1清除 [USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT](#) 中断。(WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR 写1清除 [USB_SERIAL_JTAG_USB_BUS_RESET_INT](#) 中断。(WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR 写1清除 [USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT](#) 中断。(WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR 写1清除 [USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT](#) 中断。(WT)

见下页...

Register 28.17. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

[接上页...](#)

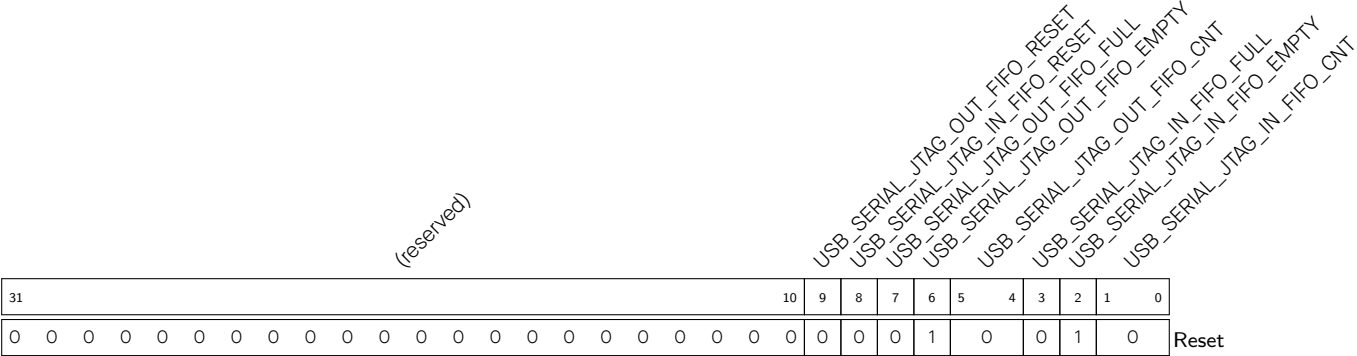
USB_SERIAL_JTAG_RTS_CHG_INT_CLR 写 1 清除 [USB_SERIAL_JTAG_RTS_CHG_INT](#) 中断。(WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR 写 1 清除 [USB_SERIAL_JTAG_DTR_CHG_INT](#) 中断。(WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR 写 1 清除 [USB_SERIAL_JTAG_GET_LINE_CODE_INT](#) 中断。(WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR 写 1 清除 [USB_SERIAL_JTAG_SET_LINE_CODE_INT](#) 中断。(WT)

Register 28.18. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT 表示 JTAG IN FIFO 计数器。(RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY 表示 JTAG IN FIFO 是否为空。

- 0: JTAG IN FIFO 不为空
 - 1: JTAG IN FIFO 为空
- (RO)

USB_SERIAL_JTAG_IN_FIFO_FULL 表示 JTAG IN FIFO 是否为满。

- 0: JTAG IN FIFO 不为满
 - 1: JTAG IN FIFO 为满
- (RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT 表示 JTAG OUT FIFO 计数器。(RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY 表示 JTAG OUT FIFO 是否为空。

- 0: JTAG OUT FIFO 不为空
 - 1: JTAG OUT FIFO 为空
- (RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL 表示 JTAG OUT FIFO 是否为满。

- 0: JTAG OUT FIFO 不为满
 - 1: JTAG OUT FIFO 为满
- (RO)

USB_SERIAL_JTAG_IN_FIFO_RESET 配置是否复位 JTAG IN FIFO。

- 0: 无效
 - 1: 复位
- (R/W)

USB_SERIAL_JTAG_OUT_FIFO_RESET 配置是否复位 JTAG OUT FIFO。

- 0: 无效
 - 1: 复位
- (R/W)

Register 28.19. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)

Diagram illustrating the structure of the `USB_SERIAL_JTAG_SOF_FRAME_INDEX` register:

- Bit 31 to 11: (reserved)
- Bit 10 to 0: `USB_SERIAL_JTAG_SOF_FRAME_INDEX`

Reset value: 0x00000000

USB_SERIAL_JTAG_SOF_FRAME_INDEX 接收 SOF 帧的帧索引。(RO)

Register 28.20. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

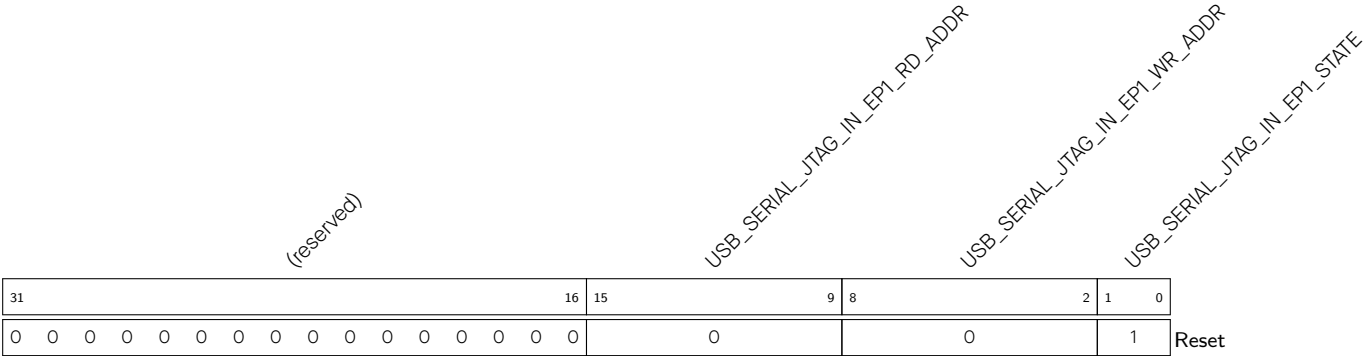
(reserved)																USB_SERIAL_JTAG_IN_EPO_RD_ADDR																USB_SERIAL_JTAG_IN_EPO_WR_ADDR																USB_SERIAL_JTAG_IN_EPO_STATE																																																																															
31																16																15																9																8																2																1																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0																0																1																Reset																																																															

USB_SERIAL_JTAG_IN_EPO_STATE 表示输入端点 0 的状态。(RO)

USB_SERIAL_JTAG_IN_EPO_WR_ADDR 表示输入端点 0 的写入数据地址。(RO)

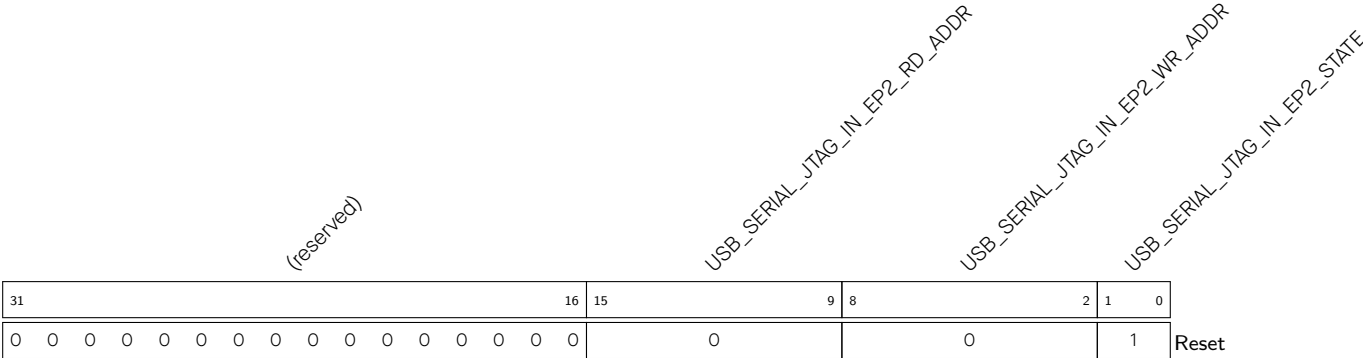
USB_SERIAL_JTAG_IN_EPO_RD_ADDR 表示输入端点 0 的读取数据地址。(RO)

Register 28.21. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)



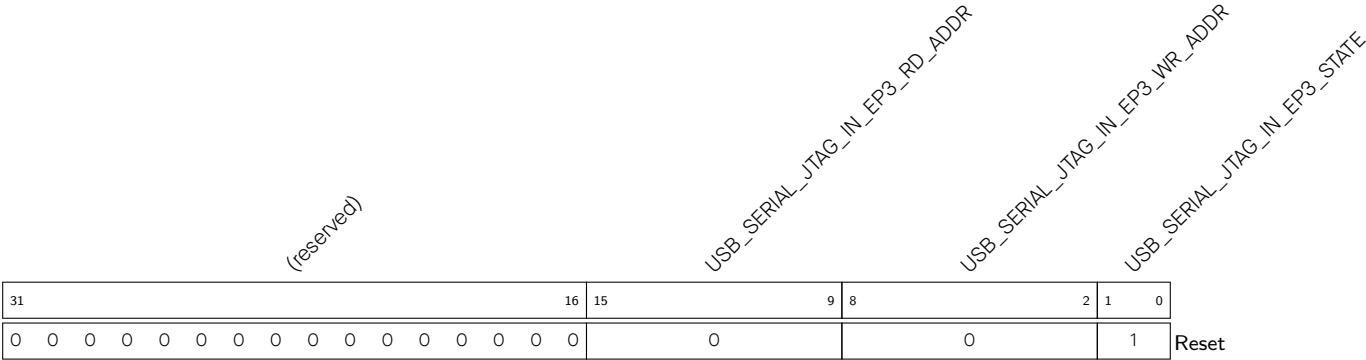
- USB_SERIAL_JTAG_IN_EP1_STATE 表示输入端点 1 的状态。(RO)
- USB_SERIAL_JTAG_IN_EP1_WR_ADDR 表示输入端点 1 的写入数据地址。(RO)
- USB_SERIAL_JTAG_IN_EP1_RD_ADDR 表示输入端点 1 的读取数据地址。(RO)

Register 28.22. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)



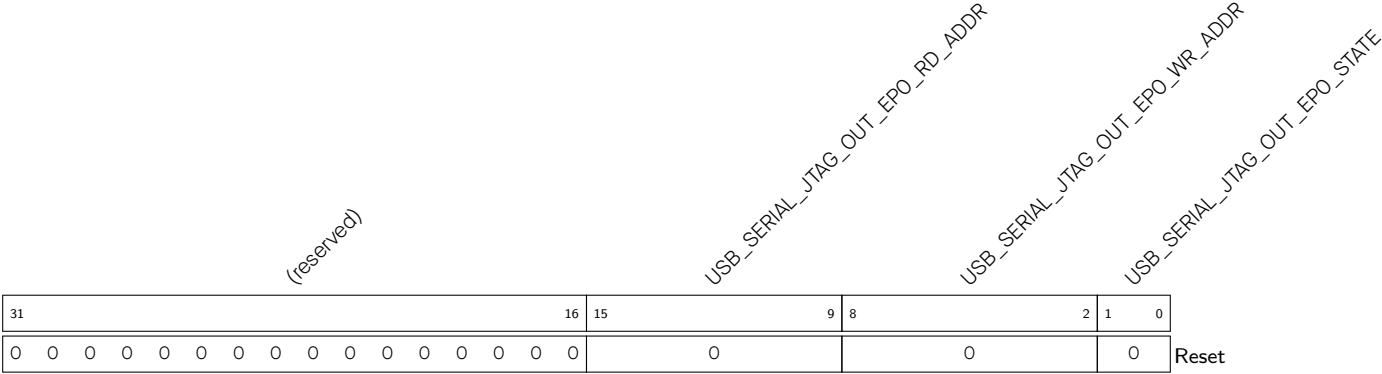
- USB_SERIAL_JTAG_IN_EP2_STATE 表示输入端点 2 的状态。(RO)
- USB_SERIAL_JTAG_IN_EP2_WR_ADDR 表示输入端点 2 的写入数据地址。(RO)
- USB_SERIAL_JTAG_IN_EP2_RD_ADDR 表示输入端点 2 的读取数据地址。(RO)

Register 28.23. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)



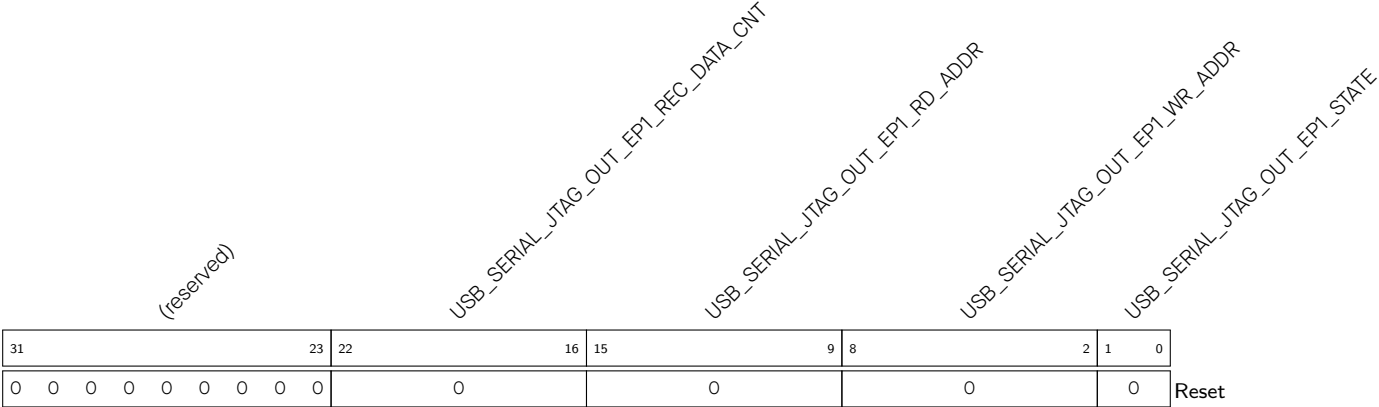
- USB_SERIAL_JTAG_IN_EP3_STATE 表示输入端点 3 的状态。(RO)
- USB_SERIAL_JTAG_IN_EP3_WR_ADDR 表示输入端点 3 的写入数据地址。(RO)
- USB_SERIAL_JTAG_IN_EP3_RD_ADDR 表示输入端点 3 的读取数据地址。(RO)

Register 28.24. USB_SERIAL_JTAG_OUT_EPO_ST_REG (0x0038)



- USB_SERIAL_JTAG_OUT_EPO_STATE 表示输出端点 0 的状态。(RO)
- USB_SERIAL_JTAG_OUT_EPO_WR_ADDR 表示输出端点 0 的写入数据地址。
当检测到 [USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT](#) 中断时，输出端点 0 中有 (USB_SERIAL_JTAG_OUT_EPO_WR_ADDR – 2) 个字节数据。
(RO)
- USB_SERIAL_JTAG_OUT_EPO_RD_ADDR 表示输出端点 0 的读取数据地址。(RO)

Register 28.25. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



USB_SERIAL_JTAG_OUT_EP1_STATE 表示输出端点 1 的状态。(RO)

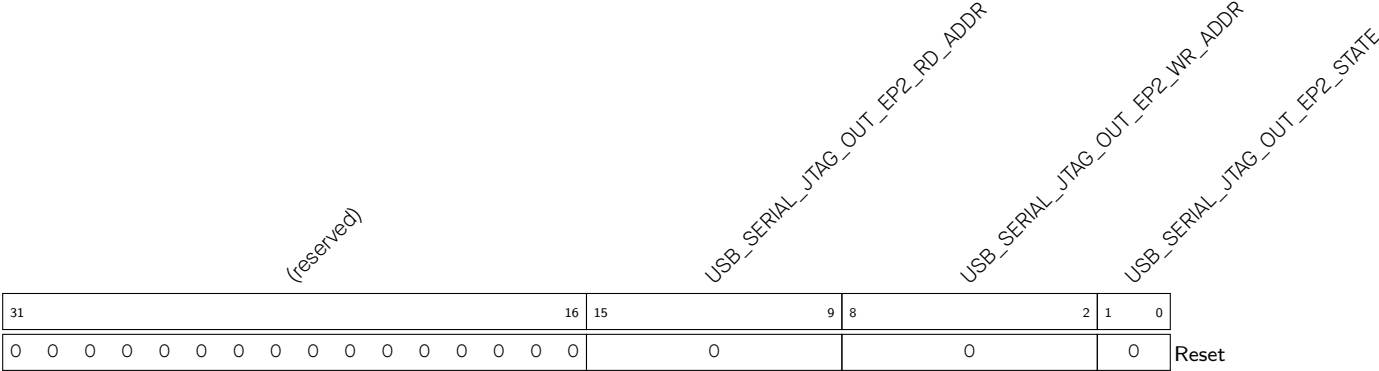
USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 表示输出端点 1 的写入数据地址。

当检测到 [USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT](#) 中断时，输出端点 1 中有 (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR – 2) 个字节数据。
(RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR 表示输出端点 1 的读取数据地址。(RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT 当接收到一个数据包时输出端点 1 中的数据计数器。(RO)

Register 28.26. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)



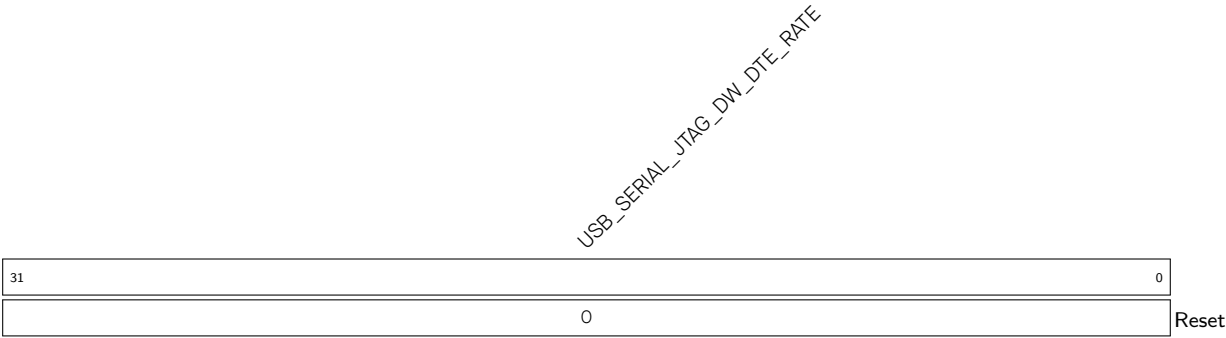
USB_SERIAL_JTAG_OUT_EP2_STATE 表示输出端点 2 的状态。(RO)

USB_SERIAL_JTAG_OUT_EP2_WR_ADDR 表示输出端点 2 的写入数据地址。

当检测到 [USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT](#) 中断时，输出端点 2 中有 (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR – 2) 个字节数据。
(RO)

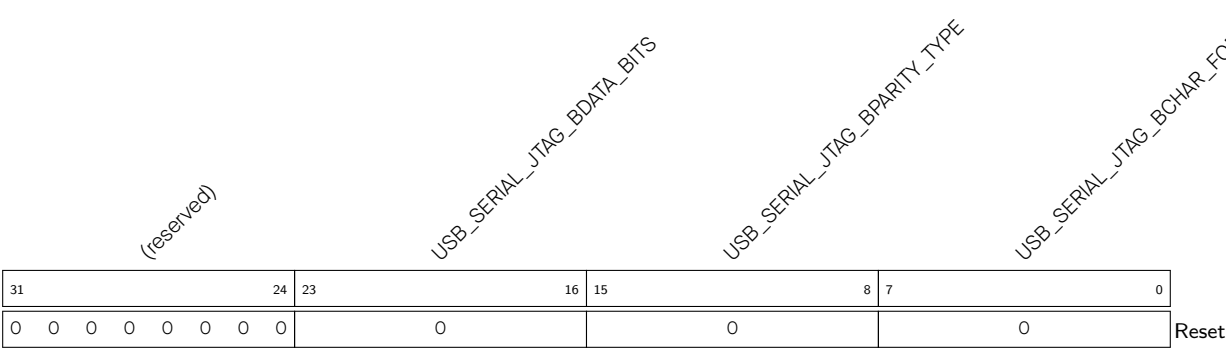
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR 表示输出端点 2 的读取数据地址。(RO)

Register 28.27. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE 表示主机通过 SET_LINE_CODING 命令设置的 dwDTERate 值。
(RO)

Register 28.28. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

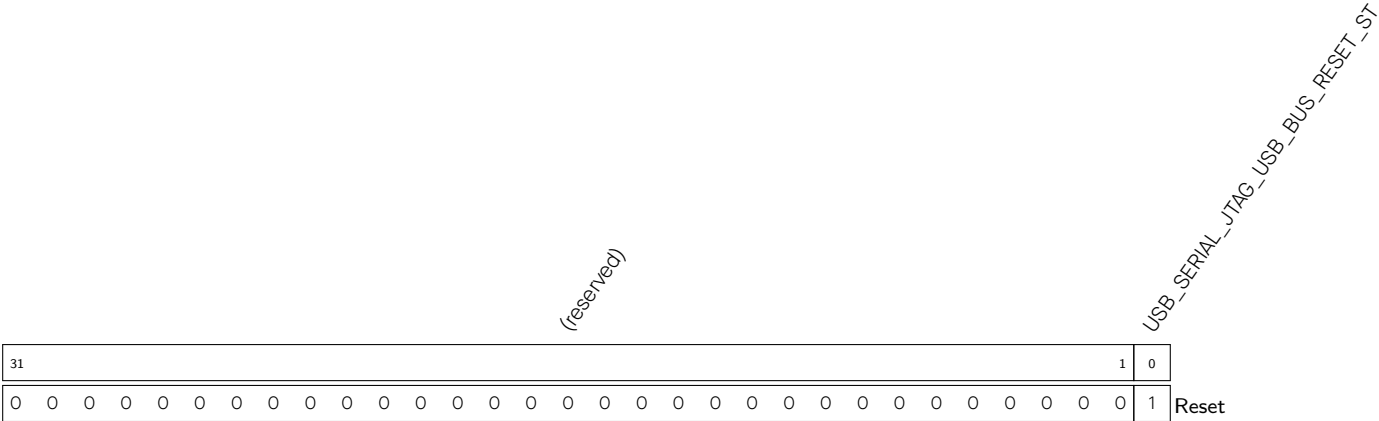


USB_SERIAL_JTAG_BCHAR_FORMAT 表示主机通过 SET_LINE_CODING 命令设置的 bCharFormat 值。(RO)

USB_SERIAL_JTAG_BPARITY_TYPE 表示主机通过 SET_LINE_CODING 命令设置的 bParityTpye 值。
(RO)

USB_SERIAL_JTAG_BDATA_BITS 表示主机通过 SET_LINE_CODING 命令设置的 bDataBits 值。(RO)

Register 28.29. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)

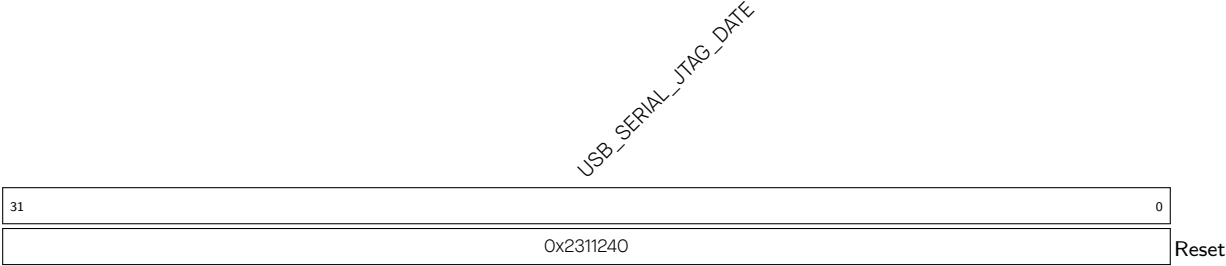


`USB_SERIAL_JTAG_USB_BUS_RESET_ST` 表示 USB 总线复位是否被释放。

- 0: USB 串口/JTAG 控制器处于 USB 总线复位状态
- 1: USB 总线复位被释放

(RO)

Register 28.30. USB_SERIAL_JTAG_DATE_REG (0x0080)



`USB_SERIAL_JTAG_DATE` 版本控制寄存器。(R/W)

第 29 章

SDIO 从机控制器 (SDIO)

29.1 概述

ESP32-C5 支持安全数字输入输出 (SDIO) 设备接口，符合 SDIO 规范 V2.00。主机 (host) 可以通过 SDIO 总线协议访问 ESP32-C5。

主机可以直接访问 ESP32-C5 SDIO 接口寄存器，或通过使用直接存储访问 (DMA) 引擎访问设备上的共享存储器，在保证性能的同时减少了处理器开销。

29.2 主要特性

SDIO 从机控制器具有以下特性：

- 符合 SDIO 物理层规范 V2.00 和 SDIO 规范 V2.00
- 支持 2 个 IO functions（除 function 0 外）
- 支持 SPI、1-bit SDIO 和 4-bit SDIO 传输模式
- 0~50 MHz 时钟范围
- 采样时钟沿或驱动时钟沿可配置
- 为信息交互设定的特定寄存器
- 支持 SDIO 中断机制
- 支持自动填充 SDIO 总线上的发送数据，同样支持自动丢弃 SDIO 总线上的填充数据
- 高达 512 字节的块大小
- 主机与从机间有中断向量可以相互中断对方
- 带有数据传输的 DMA
- 支持在保持连接的状态下进行休眠唤醒

29.3 结构概览

SDIO 从机的功能结构图如图 29.3-1 所示。

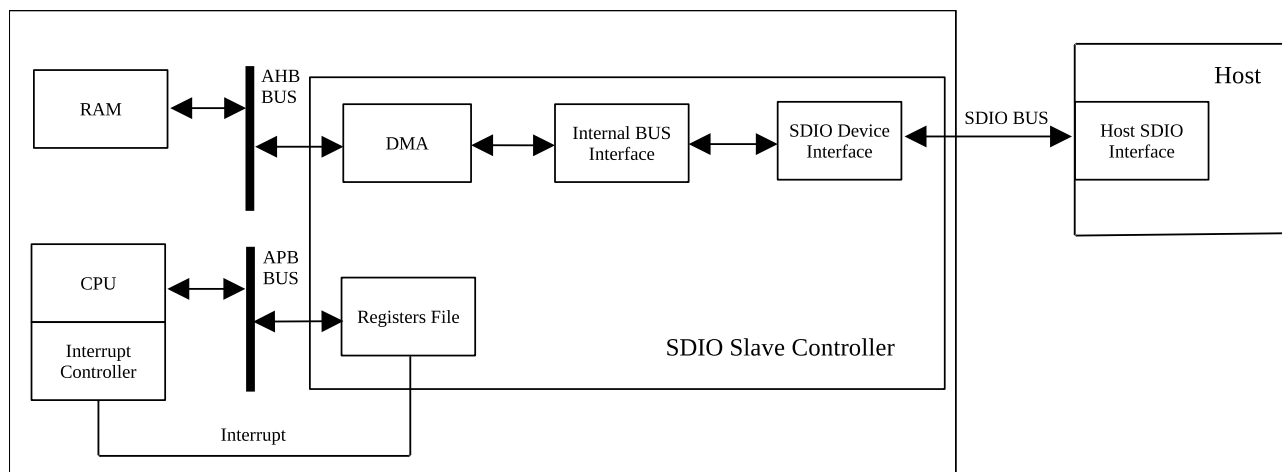


图 29.3-1. SDIO 从机功能结构图

上图中，Host（主机）代表任一符合 SDIO 规范 V2.00 的主机设备。主机通过标准 SDIO 总线与作为 SDIO 从机的 ESP32-C5 互动。SDIO 设备接口模块通过直接提供 SDIO 接口寄存器并使能 DMA 操作来与主机通信，实现高级性能总线 (AHB) 上的高速数据传输，并且不需要 CPU 的参与。

29.4 协议标准

ESP32-C5 SDIO 从机控制器符合以下协议标准：

- 《SD 卡规范第一部分：物理层规范，版本 2.00》（SD Specifications Part 1 Physical Layer Specification Version 2.00，本章中简称为“物理层规范 V2.00”）
- 《SD 卡规范 E1 部分：SDIO 规范，版本 2.00》（SD Specifications Part E1 SDIO Specification Version 2.00, January 30, 2007，本章中简称为“SDIO 规范 V2.00”）

29.5 功能描述

29.5.1 物理总线

- 总线模式：支持 SPI、1-bit SDIO 和 4-bit SDIO 传输模式。
- 总线信号：标准 SDIO 规范 V2.00 的物理总线信号。SPI 传输模式主要包括 CS/DI/SCLK/DO/IRQ 信号；1-bit SDIO 传输模式主要包括 CMD/CLK/DATA/IRQ 信号；4-bit SDIO 传输模式主要包括 CMD/CLK/DAT[3:0]。
- 总线速度模式：全速模式 0~50 MHz 时钟范围；低速模式 0~400 kHz 时钟范围。
- IO functions：除 function 0 外，还支持 2 个 IO function。其中，function 0 只用于 CCCR/FBR/CIS 操作；function 1 和 function 2 可同时用于应用数据传输（如 Wi-Fi 包和蓝牙包）和访问 SLC Host 寄存器。

详见“物理层规范 V2.00”和“SDIO 规范 V2.00”。

29.5.2 支持命令

SDIO 从机控制器支持的主要数据传输命令为 IO_RW_DIRECT (CMD52) 和 IO_RW_EXTENDED (CMD53)。

命令 IO_RW_DIRECT (CMD52) 可以用来访问寄存器和传输数据。其组成如图 29.5-1 所示，各部分具体含义请查看 SDIO 规范 V2.00。

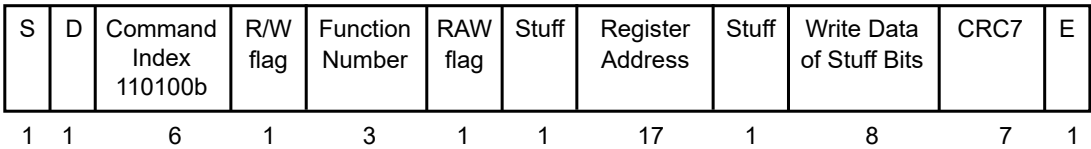


图 29.5-1. CMD52 内容

命令 IO_RW_EXTENDED (CMD53) 用来发起任意长度的包传输。其组成如图 29.5-2 所示，各部分具体含义请查看 SDIO 规范 V2.00。

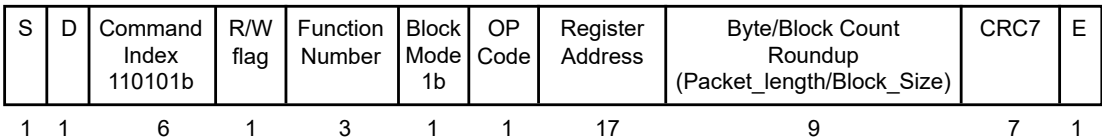


图 29.5-2. CMD53 内容

29.5.3 I/O function 0 地址空间

I/O function 0 只用于卡通用控制寄存器 (CCCR)、功能基本寄存器 (FBR)、卡信息结构寄存器 (CIS) 的操作，它的地址空间如图 29.5-3 所示。I/O function 0 的地址空间是 SDIO 规范规定的，各部分具体含义请查看 SDIO 规范 V2.00。

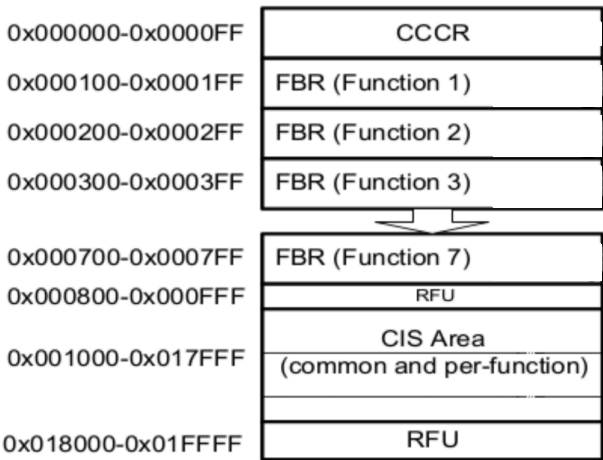


图 29.5-3. function 0 地址空间

CCCR、FBR、CIS 分别是 SDIO 协议规定的通用控制寄存器、各 function 的控制配置寄存器、以及有关版本、

功耗、制造商等卡信息的状态寄存器。这些控制或配置寄存器对应的功能是可选的，其含义详见 SDIO 规范。ESP32-C5 SDIO 从机的 CCCR 设置如表 29.5-1 所示，FBR 设置如表 29.5-2 所示。

表 29.5-1. SDIO 从机的 CCCR 设置

地址	寄存器名称	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00	CCCR/ SDIO Revision	通过 HINF_CFG_DATA1_REG 中的 HINF_SDIO_VER[7:4] 设置读取值 (SDIO bit[3:0])				通过 HINF_CFG_DATA1_REG 中的 HINF_SDIO_VER[3:0] 设置读取值 (CCCR bit[3:0])			
0x01	SD Specification Revision	0 (RFU)				通过 HINF_CFG_DATA1_REG 中的 HINF_SDIO_VER[11:8] 设置读取值 (SD bit[3:0])			
0x02	I/O Enable	0 (IOE[7:3])				读/写 (IOE[2:1])		0 (RFU)	
0x03	I/O Ready	0 (IOR[7:3])				读 (IOR[2:1])		0 (RFU)	
0x04	Int Enable	0 (IEN[7:3])				读/写 (IEN[2:1])		读/写 (IENM)	
0x05	Int Pending	0 (INT[7:3])				读 (INT[2:1])		0 (RFU)	
0x06	I/O Abort	0 (RFU)				写 (RES)	写 (AS[2:0])		
0x07	Bus Interface Control	读/写 (CD Disable)	1 (SCSI)	读/写 (ECSI)	0 (RFU)			读/写 (Bus Width[1:0])	
0x08	Card Capability	0 (4BLS)	0 (LSC)	读/写 (E4MI)	1 (S4MI)	0 (SBS)	1 (SRW)	1 (SMB)	1 (SDC)
0x09-0x0B	Common CIS Pointer	Address 0x09: 0x0; Address 0x0A: 0x10; Address 0x0B: 0x0 (Pointer to card's common CIS)							
0x0C	Bus Suspend	0 (RFU)						0 (BR)	0 (BS)
0x0D	Function Select	0 (DF)	0 (RFU)			0 (FS[3:0])			
0x0E	Exec Flags	0 (EX[7:1])							0 (EXM)
0x0F	Ready Flags	0 (RF[7:1])							0 (RFM)
0x10-0x11	FNO Block Size	读/写 (支持范围 0~512) (I/O block size for Function 0)							
0x12	Power Control	0 (RFU)						读/写 (EMPC)	1 (SMPC)
0x13	High-Speed	0 (RFU)						读/写 (EHS)	Note ^a (SHS)

^a 通过 [HINF_CFG_DATA1_REG](#) 中的 HINF_HIGH_SPEED_ENABLE 设置读取值。

表 29.5-2. SDIO 从机的 FBR 设置

地址	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x100	0 (Function 1 CSA enable)	0 (Function 1 supports CSA)	0 (RFU)		0 (Function 1 Standard SDIO Function interface code)			
0x101	0 (Function 1 Extended standard SDIO Function interface code)							
0x102	0 (RFU)						读/写 (EPS)	0 (SPS)
0x109-0x10B	Address 0x109: 0x0; Address 0x10A: 0x11; Address 0x10B: 0x0 (Pointer to Function 1 CIS)							
0x110-0x111	读/写 (支持范围 0~512) (I/O block size for Function 1)							
0x200	0 (Function 2 CSA enable)	0 (Function 2 supports CSA)	0 (RFU)		0x2 (Function 2 Standard SDIO Function interface code)			
0x201	0 (Function 2 Extended standard SDIO Function interface code)							
0x202	0 (RFU)						读/写 (EPS)	0 (SPS)
0x209-0x20B	Address 0x209: 0x0; Address 0x20A: 0x12; Address 0x20B: 0x0 (Pointer to Function 2 CIS)							
0x210-0x211	读/写 (支持范围 0~512) (I/O block size for Function 2)							

29.5.4 I/O function 1/2 地址空间

I/O function 1 和 function 2 的功能与权限完全相同，二者可同时或独立传输应用数据（例如 Wi-Fi 包和蓝牙包），都可以传输固定地址包和递增地址包。function 1 和 function 2 还可以访问同一组 SLC Host 寄存器，它们的地址空间如图 29.5-4 所示。主机可通过 I/O function 1 和 function 2 访问图中的所有地址段。

0x0	Fixed address Packet
0x1 -0x3F	Reserved
0x40 -0x3FF	SLC Host Register
0x400 - 0x1F7FF	Incr address Packet

图 29.5-4. function 1/2 地址空间

29.5.4.1 访问 SLC HOST 寄存器空间

为了方便主机与从机之间的信息交互，主机可以通过 I/O function 1/2 访问从机中的特定寄存器。这些寄存器在 0x40 -0x3FF 的连续地址段内，主机访问时只需将 CMD52 或 CMD53 中的寄存器地址设置为对应寄存器地址的低 10 位。主机可以使用 CMD53 同时访问多个寄存器，提高了数据传输的速率。

SLCHOST_CONF_WO_REG 到 SLCHOST_CONF_W15_REG 共有 52 个字节的区域可供主机和从机任意访问和修改，方便了主机与从机之间的信息交互。

主机与从机侧的软件可以同时访问 SLC Host 寄存器空间，因此需要用户设计上层机制保证主机与从机侧的软件不会同时访问相同的寄存器地址，避免两侧同时访问同一寄存器字段而造成的错误。

29.5.4.2 使用递增地址传输包

主机使用地址 0x400 - 0x1F7FF 连续传输多个应用数据包（如 Wi-Fi 包）时，CMD53 中的地址字段应设置为递增模式，OP Code 字段应固定为 1。

例如：主机希望使用 CMD53，从起始地址 0x500 开始，传输 3 个数据块。那么主机应该将 CMD53 中的字段 Block Mode 设为 1，表示以数据块为单位进行数据传输；将字段 OP Code 设为 1，表示递增地址模式；将 Register Address 字段设为 0x500，表示起始地址为 0x500；将字段 Byte/Block Count 设为 0x3，表示传输 3 个数据块；其余字段依据 SDIO 规范设置。

主机与从机间通过 CMD53 传输数据包时，从机会判定当前数据包的有效数据是否结束，从而进行无效数据的填充与丢弃。数据填充与丢弃机制详见 29.5.5.3 所述。

29.5.4.3 使用固定地址传输包

主机使用地址 0x0 传输应用数据包（如蓝牙包）时，CMD53 中的地址字段和 OP Code 字段应固定为 0。

例如：主机希望使用 CMD53，从固定地址 0x0，传输 3 个数据块。那么主机应该将 CMD53 中的字段 Block Mode 设为 1，表示以数据块为单位进行数据传输；将字段 OP Code 设为 0，表示固定地址模式；将 Register Address 字段设为 0x0，表示固定地址为 0x0；将字段 Byte/Block Count 设为 0x3，表示传输 3 个数据块；其余字段依据 SDIO 规范设置。

主机与从机间通过 CMD53 传输数据包时，从机会判定当前数据包的有效数据是否结束，从而进行无效数据的填充与丢弃。数据填充与丢弃机制详见 29.5.5.3 所述。

29.5.5 DMA

SDIO 从机有一个专门的 DMA 用于从系统存储 (RAM) 空间获取或存储传输数据，如图 29.3-1 所示，DMA 通过 AHB 总线访问 RAM。SDIO 从机控制器有权限访问的 RAM 空间详见章节 4 系统和存储器。SDIO 从机控制器一

次传输可以访问的存储地址范围，可以通过配置 SLC 寄存器中的 *SHAREMEM*_REG 相关字段控制，详见 29.8 节所述。

DMA 有 2 个通道，SLC0 和 SLC1，传输固定地址包时，会通过 DMA 的 SLC1 通道完成；传输递增地址包时，会通过 DMA 的 SLC0 通道完成。为了用户使用上的方便，SDIO 从机提供了 function 1/2 用于数据传输，I/O function 1/2 的地址范围详见 29.5.4 节所述。建议通过 function 1 传输递增地址包，使用 DMA 的 SLC0 通道；通过 function 2 传输固定地址包，使用 DMA 的 SLC1 通道。

DMA 通过 AHB 总线访问 RAM，可以通过配置 SDIO_SLCCONFO_REG 和 SDIO_SLC_BURST_LEN_REG 中的相关字段，设置 AHB 接口是否可以使用 Burst 操作，以及可以使用的 Burst 操作类型，详见 29.8 节所述。

29.5.5.1 链表

从机软件可以通过挂载链表的方式来使用 DMA 引擎。DMA 根据 RX（从机到主机）链表中的内容将相应 RAM 中的数据发送出去，也可根据 TX（主机到从机）链表中的内容将接收的数据存入指定 RAM 地址空间。每张链表由若干个描述符组成。

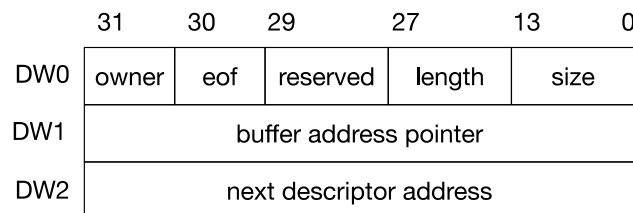


图 29.5-5. SDIO 从机 DMA 链表描述符结构

TX 链表描述符与 RX 链表描述符结构相同，图 29.5-5 所示为描述符的结构图，一个描述符由 3 个字组成。每一字段的意义如下：

- owner (DW0)[31]：表示当前描述符对应的 buffer 允许的操作者。
0：允许的操作者为 CPU；
1：允许的操作者为 DMA 引擎。
从机软件创建描述符时应将该字段置为 1；使能 DMA 回写权限并且 DMA 使用完对应 buffer 后，会将该字段清 0。
- eof (DW0)[30]：表示数据包结束标志。
0：当前描述符不是数据包的最后一个描述符；
1：当前描述符为数据包的最后一个描述符。
主机向从机发送包，从机软件创建描述符时应将该字段置为 0，DMA 会在一个包结束时将对描述符的该字段置 1；主机从从机接收包，从机软件创建描述符时根据包结束情况进行设置。
- reserved (DW0)[29:28]：reserved。
从机软件需将该字段设置为 0x0。
- length (DW0)[27:14]：表示当前描述符对应的 buffer 中的有效字节数。DMA 引擎从 buffer 中读取数据时表示能够读取的字节数；DMA 引擎向 buffer 中存储数据时表示已存数据的字节数。
主机向从机发送包，从机软件创建描述符时应将该字段置为 0x0，DMA 会在对应 buffer 使用完后回写该字段；主机从从机接收包，从机软件创建描述符时应将该字段置为对应 buffer 中能够读取的字节数。
- size (DW0)[13:0]：表示当前描述符对应的 buffer 的大小，单位为字节。
从机软件创建描述符时设置。
注意：大小必须字对齐。

- buffer address pointer (DW1): buffer 地址指针。
从机软件创建描述符时设置。
注意: 地址必须字对齐。
- next descriptor address (DW2): 下一个描述符的地址。下一个描述符不存在时, 该值为 0。
从机软件创建描述符时设置。

DMA 回写链表描述符字段详见 29.5.5.2 所述。

从机软件可以将多个描述符通过 next descriptor address (DW2) 字段组成链表, SDIO 从机 DMA 链表如图 29.5-6 所示:

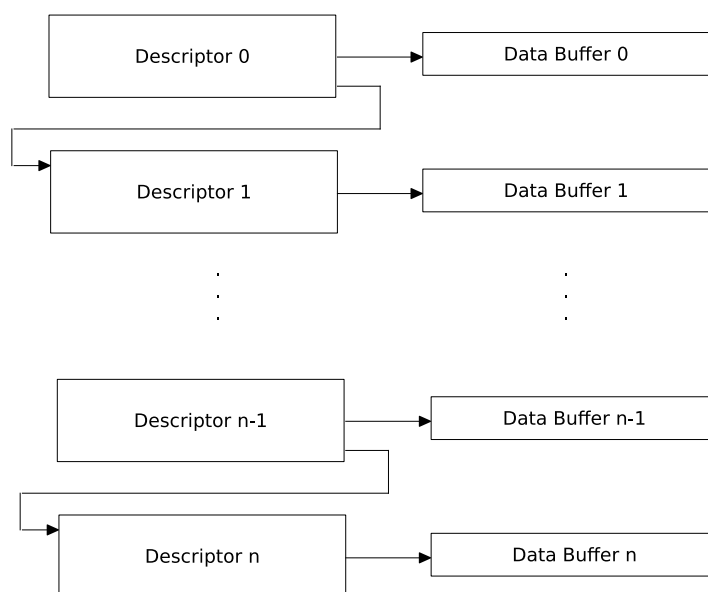


图 29.5-6. SDIO 从机 DMA 链表

为了更清晰的理解链表和 eof 位, 举例如下: 假设从机侧软件构造了链表, 包含 3 个描述符。

- 描述符 0 指向的数据大小为 500 字节, eof 位为 0。
 - 描述符 1 指向的数据大小为 200 字节, eof 位为 1。
 - 描述符 2 指向的数据大小为 200 字节, eof 位为 1。
1. 若第 1 个 CMD53 命令要求读取 400 字节的数据, 则 DMA 会将描述符 0 指向的前 400 字节数据发送给主机。
 2. 若第 2 个 CMD53 命令要求读取 400 字节的数据, 则 DMA 会先将描述符 0 指向的后 100 字节剩余数据发送给主机; 之后再将描述符 1 指向的 200 字节数据发送给主机; 之后由于描述符 1 的 eof 位为 1, DMA 判定当前命令的有效数据结束, 从而向主机发送 100 字节的无效数据 0x0。
 3. 若第 3 个 CMD53 命令要求读取 400 字节的数据, 则 DMA 会先将描述符 2 指向的 200 字节数据发送给主机; 之后由于描述符 2 的 eof 位为 1, DMA 判定当前命令的有效数据结束, 从而向主机发送 200 字节的无效数据 0x0。

29.5.5.2 链表回写

主机向从机发送包的过程中, 当一个链表描述符指定的 buffer 被写满, 或者一个包传输结束时, DMA 引擎需要跳到下一个描述符来存储后续的数据。在跳转之前, DMA 会对当前的描述符进行回写。DMA 可以回写描述符

的 DWO，将 eof 和 length 位写入最新的值，owner 位的值由 `SDIO_SLCCONFO_REG` 中的 `SDIO_SLC0/1_TX_LOOP_TEST` 决定。

主机从从机接收包的过程中，当一个链表描述符指定的 buffer 被读空时，DMA 引擎需要跳到下一个描述符来读取后续的数据。在此之前，从机侧软件可将 `SDIO_SLCCONFO_REG` 中的 `SDIO_SLC0/1_RX_AUTO_WRBACK` 置 1，DMA 会对当前的描述符进行回写。可以回写描述符的 DWO，将 owner 位写入最新的值，owner 位的值由 `SDIO_SLCCONFO_REG` 中的 `SDIO_SLC0/1_RX_LOOP_TEST` 决定，而其他位的值保持不变。

相关寄存器字段详见 29.8 节所述。

29.5.5.3 数据填充与丢弃

为了实现块传输，主机和从机都需要将 SDIO 总线上发送的数据填充为整个块。从机在发包时会自动填充数据，在收包时自动丢弃填充数据。

- 主机向从机通过 CMD53 发送数据包时，当数据量达到数据包长度时，SDIO 从机判定当前数据包有效数据结束。此时 DMA 会回写当前链表描述符，将当前描述符的 eof 位置 1，并产生 `SLC0/1_TX_SUC_EOF_INT` 中断。判定有效数据结束后，当前数据包的剩余数据会被认为是无效数据，不再经 DMA 收入 buffer。直至下一个 CMD53，从机才会重新开始将数据接收至下一个 buffer。
 - 对于递增地址包，从机判断有效数据是由地址决定的，当数据地址大于等于 `0x1F800` 后会被判定为无效数据从而丢弃。因此，主机应将 CMD53 起始地址字段设为 `0x1F800 - Packet_length` (`Packet_length` 的单位是字节)。递增地址包在 SDIO 总线上的数据流如图 29.5-7 所示。

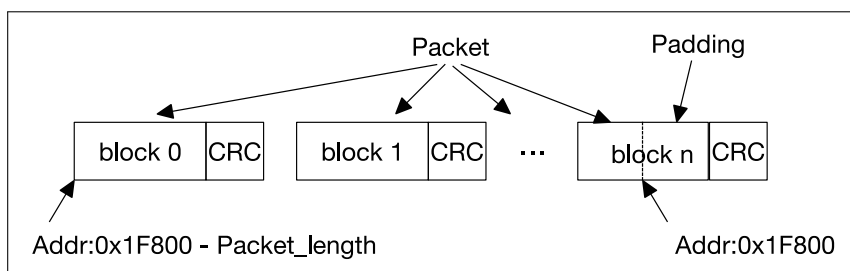


图 29.5-7. 主机向从机发送递增地址包的数据流

- 对于固定地址包，从机使用数据包的前 3 个字节作为包长度（包长度包括前 3 个字节，并且前 3 个字节也会被存入链表指定的 buffer 中），当收到的数据长度达到包长度后，之后的数据会被认为是无效数据而丢弃。
- 主机通过 CMD53 从从机获取数据包（包括递增地址包和固定地址包）时，当 DMA 读取到某个 buffer 的最后一个字节，且 DMA 链表描述符的 eof 位为 1 时，从机判定当前包的有效数据结束。此时 DMA 会回写当前描述符，并触发 `SLC0/1_RX_EOF_INT` 中断。判定有效数据结束后，当前数据包的剩余数据会被无效数据 `0x0` 填充，不再经 DMA 从 buffer 读取。直至主机发送下一个 CMD53，从机才会重新开始经 DMA 从 buffer 读取数据。

注意：主机从从机获取数据包时，无论递增地址包和固定地址包都使用 DMA 链表描述符的 eof 位作为数据结束的判断依据，而不是地址 `0x1F800`。因此，主机下发多个 CMD53 获取多个数据包，只要 DMA 没有遇到描述符 eof 位为 1，就会按照链表对应的 buffer 顺序依次获取数据，依次传输给主机；当 DMA 遇到描述符 eof 位为 1 时，会对对应的 buffer 中取完数据，然后填充无效数据完成当前 CMD53 命令，下一个 CMD53 命令将会使用下一个描述符指向的 buffer 数据。

29.5.6 SDIO 总线时序

主机与从机间是通过 PCB 走线连接，所以延迟大。为了保证总线上时序正确，从机支持调整 SDIO 总线输入的采样时钟沿和输出的驱动时钟沿。

当从主机来的数据变化的时刻靠近时钟的上升沿时，从机会选择时钟的下降沿进行采样。采样时序图如图 29.5-8 所示：

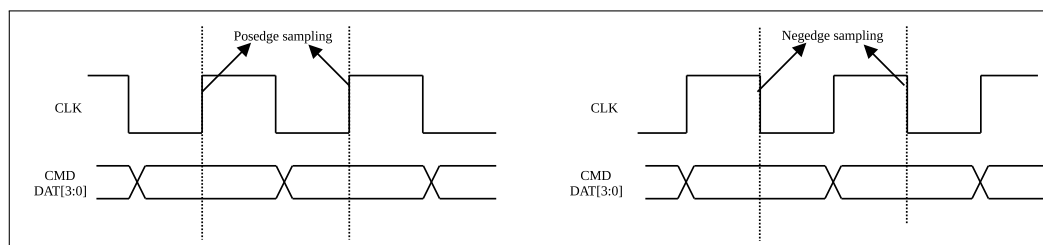


图 29.5-8. 采样时序图

从机引脚采样沿默认由 GPIO25 strapping 引脚电平决定，但通过配置 [SLCHOST_CONF_REG](#) 寄存器可以强制决定模式（优先级从高到低）：(1) 置位 [SLCHOST_FRC_POS_SAMP](#)，对应引脚强制上升沿采样；(2) 置位 [SLCHOST_FRC_NEG_SAMP](#)，对应引脚强制下降沿采样。

[SLCHOST_FRC_POS_SAMP](#) 和 [SLCHOST_FRC_NEG_SAMP](#) 的位宽均为 5 bit，5 个 bit 分别对应 CMD 线和 4 个 DATA 线 (0-3)。置位这两个域使得从机对相应的线在时钟上升沿或下降沿上进行采样。

从机还可以选择时钟的上升沿还是下降沿对数据输出线进行驱动，以适应不同的延迟。时序图如图 29.5-9 所示：

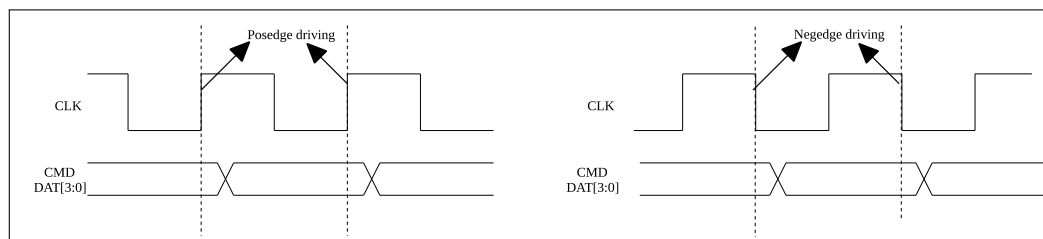


图 29.5-9. 输出时序图

从机引脚输出沿默认由 MTDI strapping 引脚电平决定，但通过配置以下寄存器可以强制决定模式（优先级从高到低）：(1) 置位 [SLCHOST_CONF_REG](#) 中的 [SLCHOST_FRC_SDIO11](#)，对应引脚强制下降沿输出；(2) 置位 [SLCHOST_CONF_REG](#) 中的 [SLCHOST_FRC_SDIO22](#)，对应引脚强制上升沿输出；(3) 置位 [HINF_CFG_DATA1_REG](#) 中的 [HINF_HIGH_SPEED_ENABLE](#) 和 [SLCHOST_CONF_REG](#) 中的 [SLCHOST_HSPEED_CON_EN](#) 后，主机通过配置 CCCR 中的 EHS (Enable High-Speed) 为 1 强制上升沿输出。

[SLCHOST_FRC_SDIO11](#) 和 [SLCHOST_FRC_SDIO22](#) 的位宽均为 5 bit，5 个 bit 分别对应 CMD 线和 4 个 DATA 线 (0-3)。置位这两个域使得从机在时钟下降沿或上升沿上对相应的线进行驱动。

关于优先级的说明：无论输入还是输出，strapping 引脚的配置也被包含在优先级范围内（优先级最低）。每一级有效的前提是比它高的优先级都不生效。例如 GPIO25 strapping 的配置决定对应引脚的采样沿，生效的前提是 [SLCHOST_FRC_POS_SAMP](#) 没有置 1，[SLCHOST_FRC_NEG_SAMP](#) 也没有置 1。

29.6 中断

主机和从机间可以通过配置中断向量灵活地中断对方。主机和从机 DMA 每个 SLC 通道之间各有 8 个中断向量可用于中断对方。在配置中断向量寄存器后就会向对方发送中断（配置相应的中断使能寄存器）。

29.6.1 主机侧中断

- `SLCHOST_SLC0/1_RX_NEW_PACKET_INT`：SDIO Slave 有数据包需要向主机发送。以下任意一种情况可触发该中断：
 - 软件配置 `SDIO_SLCORX_LINK_REG` 或 `SDIO_SLC1RX_LINK_REG` 中的 `SDIO_SLC0/1_RXLINK_START` 字段启动 DMA 时
 - DMA 处理完 eof 字段为 1 的 RX 链表描述符，之后又有新的 RX 链表描述符时
 - 数据包需要重传时
- `SLCHOST_SLC0/1_TX_OVF_INT`：从机 TX buffer 溢出中断
- `SLCHOST_SLC0/1_RX_UDF_INT`：从机 RX buffer 下溢中断
- `SLCHOST_SLC0/1_TOHOST_BITn_INT` (n : 0 ~ 7)：从机中断主机

29.6.2 从机侧中断

- `SLC0/1_RX_DSCR_ERR_INT`：从机 RX 链表描述符错误中断
- `SLC0/1_TX_DSCR_ERR_INT`：从机 TX 链表描述符错误中断
- `SLC0/1_RX_EOF_INT`：从机 RX 操作完成中断
- `SLC0/1_RX_DONE_INT`：单个 buffer 由从机 RX 完成的中断
- `SLC0/1_TX_SUC_EOF_INT`：从机 TX 操作完成中断
- `SLC0/1_TX_DONE_INT`：单个 buffer 在从机 TX 操作时填满了的中断
- `SLC0/1_TX_OVF_INT`：从机 TX buffer 溢出中断
- `SLC0/1_RX_UDF_INT`：从机 RX buffer 下溢中断
- `SLC0/1_TX_START_INT`：从机 TX 操作开始中断
- `SLC0/1_RX_START_INT`：从机 RX 操作开始中断
- `SLC_FRHOST_BITn_INT` (n : 0~15)：主机中断从机，其中中断向量 Bit[7:0] 通过 DMA SLC0 通道处理中断，中断向量 Bit[15:8] 通过 DMA SLC1 通道处理中断

29.7 包的发送和接收流程

SDIO 主机与从机间的包传输需要两者按照特定的流程配合完成。在基于 SDIO 规范协议的基础上，ESP32-C5 针对 Wi-Fi/蓝牙等更高抽象层次的数据传输应用场景，需要额外遵循如下特定的数据传输过程。

29.7.1 从机向主机发送包

包的发送是由从机发起，通过中断来通知主机（中断实现方式参看 SDIO 协议）。主机从从机读取相关信息后发起 SDIO 总线传输。整个流程如图 29.7-1 所示：

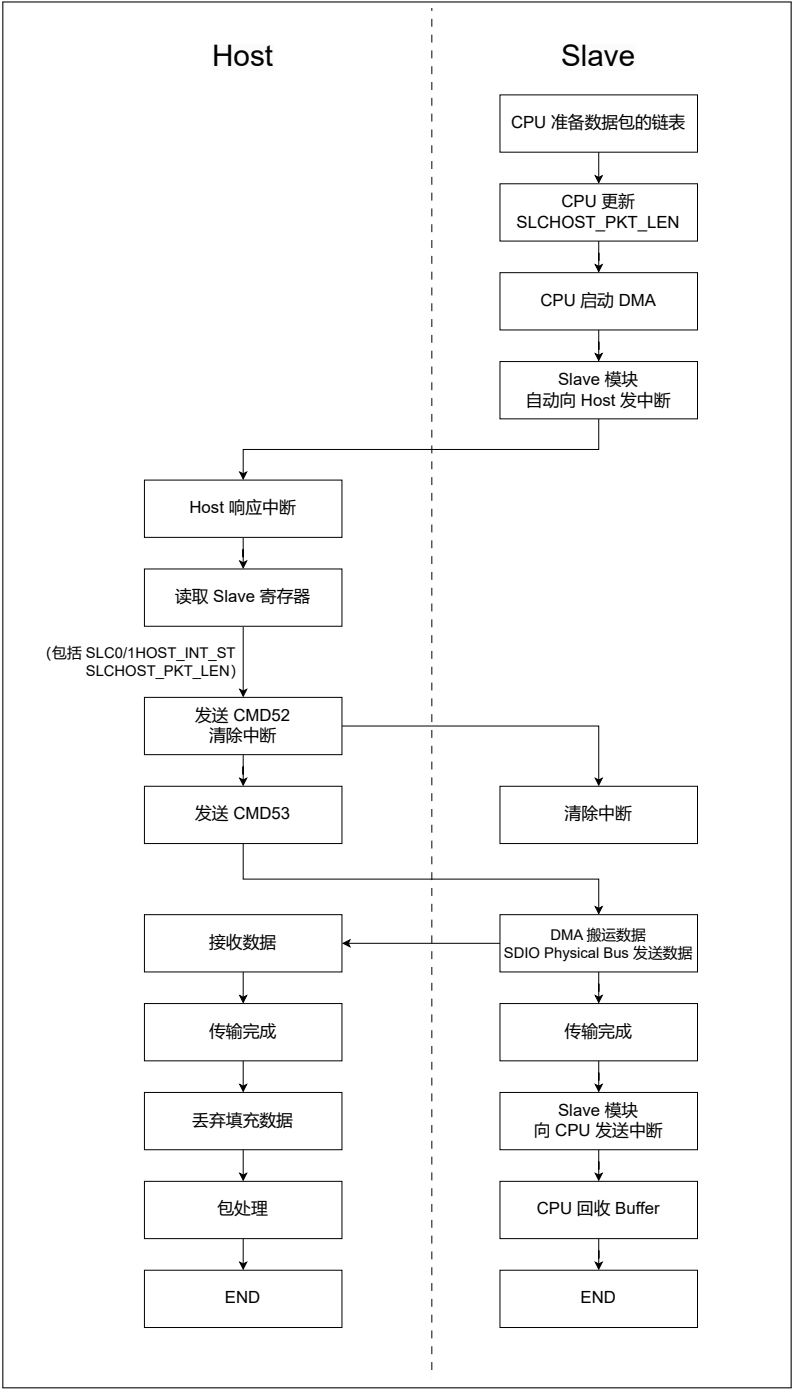


图 29.7-1. 从机向主机发送包的流程

- 1. 从机 CPU 根据传输数据包的具体情况，构建链表。更多关于构建链表的信息请参考 29.5.5.1 小节。
- 2. 从机 CPU 通过寄存器 `SDIO_SLC0_LEN_CONF_REG` 向主机更新要发送的数据长度信息。
- 3. 从机 CPU 启动 DMA。需要先将链表第一个描述符的 32 位地址写到寄存器 `SDIO_SLCORX_LINK_ADDR_REG` 或 `SDIO_SLC1RX_LINK_ADDR_REG` 中，再配置 `SDIO_SLCORX_LINK_REG` 或 `SDIO_SLC1RX_LINK_REG` 的 `SDIO_SLC0/1_RXLINK_START` 字段来启动

DMA。更多关于 DMA 的信请参考 29.5.5 小节。

4. 从机 DMA 向主机发送中断。
5. 主机收到中断后，通过访问 `SLCHOST_SLC0HOST_INT_ST_REG`、`SLCHOST_SLC1HOST_INT_ST_REG` 和 `SLCHOST_PKT_LEN_REG` 这 3 个寄存器从从机读取相关信息。
 - `SLCHOST_SLC0/1HOST_INT_ST_REG`：中断状态寄存器，其中 bit `SLCHOST_SLC0/1_RX_NEW_PACKET_INT_ST` 为 1 表明中断的原因是从机发包。
 - `SLCHOST_PKT_LEN_REG`：从机发送包长度累加寄存器，主机用本次读取值减去上次读取值就可以得到本次发送包的长度。
6. 主机通过 CMD52 命令写到寄存器 `SLCHOST_SLC0/1HOST_INT_CLR_REG` 清除中断。
7. 主机通过 CMD53 命令从从机处获取数据包。数据传输过程中，当从机判定当前包的有效数据结束时，之后的数据会使用无效数据 0x0 填充。关于从机如何判定有效数据结束的信息，请参考 29.5.5.3 小节。
8. 数据包发送完成后，从机 DMA 会向 CPU 发送中断，这时 CPU 可以回收 buffer。

注意：

- 不建议链表中 eof 位全设置为 0。如果链表中没有设置 eof 位为 1，DMA 可能会将下一个包的数据发送给当前命令，造成错误。若链表不含 eof 位为 1 的描述符，Slave 侧软件应该负责填充数据，将每个数据包的长度对齐到数据块的大小，避免 DMA 将下一个包的数据发送给当前命令。Host 发送 CMD53 读取数据时应该准确控制每个包的数据块个数，避免多取或少取数据，同时 Host 应该有识别填充数据的能力。
- 建议用户一个 CMD53 命令只传输一个数据包，每个数据包只使用一个链表，避免复杂传输造成未知异常。
- 不建议通过一个链表发送多个数据包，Host 及软件可能很难对数据包进行分割。如果软件希望通过一个链表发送多个数据包，应该在构造链表时通过 eof 字段对数据包进行分割，方便 DMA 根据链表对不同的数据包做数据填充（建议每个数据包的长度都要对齐到数据块的大小，可以避免 DMA 填充数据），同时 Host 应该有识别填充数据的能力。

29.7.2 从机从主机接收包

包的接收是由主机发起，从机通过 DMA 接收数据并存储到 RAM 中，传输完成后通过中断通知 CPU 进行数据处理。整个流程如图 29.7-2 所示：

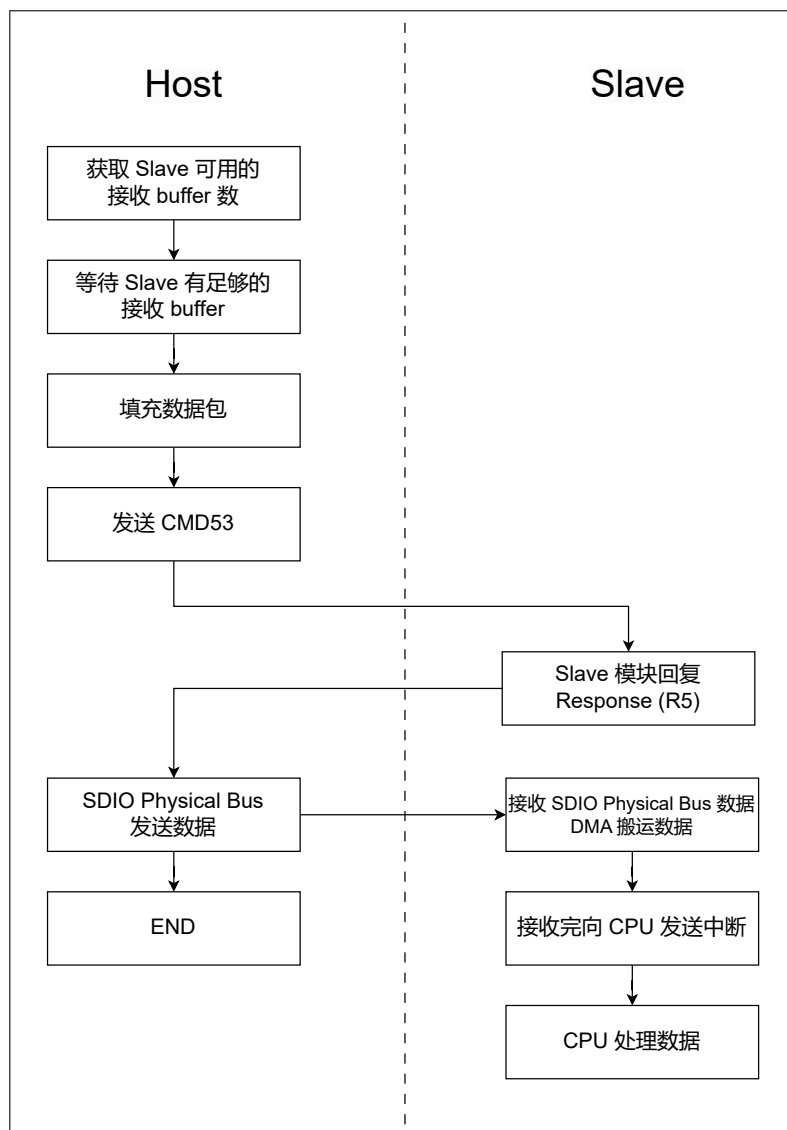


图 29.7-2. 从机从主机接收包的流程

主机通过访问寄存器 [SLCHOST_SLC0HOST_TOKEN_RDATA_REG](#) 或 [SLCHOST_SLC1HOST_TOKEN_RDATA_REG](#) 获取从机侧可用于接收包的 buffer 数。从机侧的 CPU 要在准备好用于接收包的 DMA 链表后更新这个值。

[SLCHOST_HOSTSLCHOST_SLC0_TOKEN1](#) 或 [SLCHOST_HOSTSLCHOST_SLC1_TOKEN1](#) 存放可用 buffer 的累计值。主机通过该字段的值减去已用掉的 buffer 数就得到了当前可用的 buffer 数。当 buffer 不够用时，主机需要通过不停地访问该寄存器直到 buffer 够用为止。

主机通过 CMD53 命令向从机发送数据包的过程中，当一个 DMA 链表描述符指定的区域被写满，或者一个包传输结束时，DMA 便会跳到下一个描述符来存储后续的数据。从机判定当前数据包有效数据结束时，会丢弃当前包的无效填充数据，DMA 会将当前的描述符 eof 位回写为 1，并产生 [SLC0/1_TX_SUC_EOF_INT](#) 中断。

DMA 功能、链表、以及数据丢弃详见 29.5.5 小节所述。

为了保证有充足的 buffer 来接收包，从机的 CPU 必须不停地在接收链表上挂载 buffer。挂载的流程如图 29.7-3 所示。

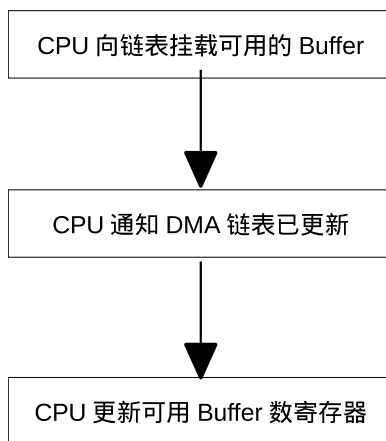


图 29.7-3. 从机 CPU 挂载 buffer 的流程

CPU 首先需要将新的 buffer 组成链表并且挂载在 DMA 正在使用的链表结尾。

然后，CPU 需要通知 DMA 链表已更新，可以通过置位 `SDIO_SLC0_TXLINK_RESTART` 或 `SDIO_SLC1_TXLINK_RESTART` 字段来实现。注意，CPU 在首次启动 DMA 用于收包时需置位 `SDIO_SLC0_TXLINK_START` 或 `SDIO_SLC1_TXLINK_START` 字段为 1。

RESTART 字段说明： DMA RESTART 只适用于 2 种情况：

- 配置相关寄存器中的 `*_STOP` 字段暂停了 DMA 的工作，之后可以通过配置 `*_RESTART` 字段使 DMA 继续工作
- 因为缺乏链表描述符导致 DMA 暂停工作，补充描述符后可以通过配置 `*_RESTART` 字段使 DMA 继续工作。

最后，CPU 通过配置 `SDIO_SLC0TOKEN1_REG` 或 `SDIO_SLC1TOKEN1_REG` 寄存器更新可用的 buffer。

29.8 寄存器列表

本小节的所有地址均为相对于 SDIO 从机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

29.8.1 HINF 寄存器列表

名称	描述	地址偏移	访问
配置寄存器			
HINF_CFG_DATA0_REG	SDIO CIS 配置	0x0000	R/W
HINF_CFG_DATA1_REG	SDIO 配置	0x0004	R/W
HINF_CFG_DATA7_REG	SDIO 配置	0x001C	varies
HINF_CIS_CONF_W n _REG(n : 0-7)	SDIO CIS 配置	0x0020+0x4* n	R/W
HINF_CFG_DATA16_REG	SDIO CIS 配置	0x0040	R/W
Status registers			
HINF_CONF_STATUS_REG	SDIO CIS function 0 config0 状态	0x0054	RO

29.8.2 SLC 寄存器列表

名称	描述	地址	访问
配置寄存器			
SDIO_SLCCONFO_REG	DMA 配置	0x0000	R/W
SDIO_SLCORX_LINK_REG	SCLO RX 链表配置	0x003C	varies
SDIO_SLCORX_LINK_ADDR_REG	SCLO RX 链表地址	0x0040	R/W
SDIO_SLCOTX_LINK_REG	SCLO TX 链表配置	0x0044	varies
SDIO_SLCOTX_LINK_ADDR_REG	SCLO RX 链表地址	0x0048	R/W
SDIO_SLC1RX_LINK_REG	SCL1 RX 链表配置	0x004C	varies
SDIO_SLC1RX_LINK_ADDR_REG	SCL1 RX 链表地址	0x0050	R/W
SDIO_SLC1TX_LINK_REG	SCL1 TX 链表配置	0x0054	varies
SDIO_SLC1TX_LINK_ADDR_REG	SCL1 TX 链表地址	0x0058	R/W
SDIO_SLCOTOKEN1_REG	SLCO 接收 buffer 配置	0x0064	varies
SDIO_SLC1TOKEN1_REG	SLC1 接收 buffer 配置	0x006C	varies
SDIO_SLCCONF1_REG	DMA 配置	0x0070	R/W
SDIO_SLC_RX_DSCR_CONF_REG	DMA slave 到主机配置寄存器	0x00A8	R/W
SDIO_SLC0_LEN_CONF_REG	传输包长度控制	0x00F4	varies
SDIO_SLC0_TX_SHAREMEM_START_REG	SLCO AHB TX 地址范围起始边界	0x0154	R/W
SDIO_SLC0_TX_SHAREMEM_END_REG	SLCO AHB TX 地址范围结束边界	0x0158	R/W
SDIO_SLC0_RX_SHAREMEM_START_REG	SLCO AHB RX 地址范围起始边界	0x015C	R/W
SDIO_SLC0_RX_SHAREMEM_END_REG	SLCO AHB RX 地址范围结束边界	0x0160	R/W
SDIO_SLC1_TX_SHAREMEM_START_REG	SLC1 AHB TX 地址范围起始边界	0x0164	R/W
SDIO_SLC1_TX_SHAREMEM_END_REG	SLC1 AHB TX 地址范围结束边界	0x0168	R/W

名称	描述	地址	访问
SDIO_SLC1_RX_SHAREMEM_START_REG	SLC1 AHB RX 地址范围起始边界	0x016C	R/W
SDIO_SLC1_RX_SHAREMEM_END_REG	SLC1 AHB RX 地址范围结束边界	0x0170	R/W
SDIO_SLC_BURST_LEN_REG	DMA AHB burst type 配置	0x017C	R/W
中断寄存器			
SDIO_SLC0INT_RAW_REG	SLC0 到从机原始中断状态	0x0004	varies
SDIO_SLC0INT_ST_REG	SLC0 到从机屏蔽中断状态	0x0008	RO
SDIO_SLC0INT_ENA_REG	SLC0 到从机中断使能	0x000C	R/W
SDIO_SLC0INT_CLR_REG	SLC0 到从机中断清除	0x0010	WT
SDIO_SLC1INT_RAW_REG	SLC1 到从机原始中断状态	0x0014	varies
SDIO_SLC1INT_CLR_REG	SLC1 到从机中断清除	0x0020	WT
SDIO_SLCINTVEC_TOHOST_REG	从机到主机中断向量设置	0x005C	WT
SDIO_SLC1INT_ST1_REG	SLC1 到从机屏蔽中断状态	0x014C	RO
SDIO_SLC1INT_ENA1_REG	SLC1 到从机中断使能	0x0150	R/W
状态寄存器			
SDIO_SLC0_LENGTH_REG	传输包长度	0x00F8	RO

29.8.3 SLC Host 寄存器列表

名称	描述	地址	访问
配置寄存器			
SLCHOST_CONF_REG	沿配置	0x01F0	R/W
中断寄存器			
SLCHOST_SLC0HOST_INT_RAW_REG	SLC0 中断主机的原始中断状态	0x0050	varies
SLCHOST_SLC1HOST_INT_RAW_REG	SLC1 中断主机的原始中断状态	0x0054	varies
SLCHOST_SLC0HOST_INT_ST_REG	SLC0 中断主机的屏蔽中断状态	0x0058	RO
SLCHOST_SLC1HOST_INT_ST_REG	SLC1 中断主机的屏蔽中断状态	0x005C	RO
SLCHOST_CONF_W7_REG	主机中断从机的中断向量集	0x008C	R/W
SLCHOST_SLC0HOST_INT_CLR_REG	SLC0 中断主机的中断清除	0x00D4	WT
SLCHOST_SLC1HOST_INT_CLR_REG	SLC1 中断主机的中断清除	0x00D8	WT
SLCHOST_SLC0HOST_FUNC1_INT_ENA_REG	SLC0 中断主机的中断使能	0x00DC	R/W
SLCHOST_SLC1HOST_FUNC1_INT_ENA_REG	SLC1 中断主机的中断清除	0x00E0	R/W
状态寄存器			
SLCHOST_SLC0HOST_TOKEN_RDATA_REG	SLC0 接收 buffer 的累计值	0x0044	RO
SLCHOST_PKT_LEN_REG	传输数据包的长度	0x0060	RO
SLCHOST_SLC1HOST_TOKEN_RDATA_REG	SLC1 接收 buffer 的累计值	0x00C4	RO
通信寄存器			
SLCHOST_CONF_W n _REG(n : 0-2)	主机从机通信寄存器	0x006C+0x4* n	R/W
SLCHOST_CONF_W3_REG	主机从机通信寄存器	0x0078	R/W
SLCHOST_CONF_W4_REG	主机从机通信寄存器	0x007C	R/W
SLCHOST_CONF_W6_REG	主机从机通信寄存器	0x0088	R/W
SLCHOST_CONF_W n _REG(n : 8-15)	主机从机通信寄存器	0x009C+0x4*(n -8)	R/W

29.9 寄存器

本小节的所有地址均为相对于 SDIO 从机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

29.9.1 HINF 寄存器

Register 29.1. HINF_CFG_DATA0_REG (0x0000)

HINF_USER_ID_FN1

HINF_DEVICE_ID_FN1

31	16	15	0
0x92		0x6666	

Reset

HINF_DEVICE_ID_FN1 配置 function 1 在 SDIO CIS 寄存器中 device ID 字段的值。(R/W)

HINF_USER_ID_FN1 配置 function 1 在 SDIO CIS 寄存器中 user ID 字段的值。(R/W)

Register 29.2. HINF_CFG_DATA1_REG (0x0004)

(reserved)																									HINF_FUNC2_EPS										HINF_SDIO_VER										HINF_IOENABLE1										HINF_FUNC1_EPS										(reserved)										HINF_SDIO_IOREADY2										HINF_HIGHSPEED_MODE										HINF_HIGHSPEED_ENABLE										(reserved)																																																																															
31																									25										24										23										12										11										10										9										8										7										6										5										4										3										2										1										0									
0																									0										0x232										0										0										0										0										0										0										0										1										0										0										0										1										Reset																			

HINF_SDIO_IORDY1 配置 SDIO CCCR 寄存器中 IOR1 字段, 以及 CIS 寄存器中 function 1 ready 字段的值。

0: function 1 未准备就绪

1: function 1 已准备就绪

详情请参考 SDIO 规范。

(R/W)

HINF_HIGHSPEED_ENABLE 配置是否支持 SDIO CCCR 寄存器中 SHS 字段的值。

0: 不支持高速模式

1: 支持高速模式

详情请参考 SDIO 规范。

(R/W)

HINF_HIGHSPEED_MODE 表示 SDIO CCCR 寄存器中 EHS 字段的值。

0: 未使能

1: 使能

详情请参考 SDIO 规范。

(RO)

HINF_SDIO_CD_ENABLE 配置是否使能 SDIO 卡检测。

0: 不使能

1: 使能

(R/W)

HINF_SDIO_IOREADY2 配置 CCCR 寄存器中的 IOR2 字段，以及 CIS 寄存器中的 function 2 ready 字段的值。

0: function 2 未准备就绪

1: function 2 已准备就绪

详情请参考 SDIO 规范。

(R/W)

HINF IOENABLE2 表示 SDIO CCCR 寄存器中的 IOE2 是否使能。

0: function 2 未使能

1: function 2 已使能

详情请参考 SDIO 规范。

(RO)

见下页……

Register 29.2. HINF_CFG_DATA1_REG (0x0004)

[接上页……](#)

HINF_CD_DISABLE 表示 SDIO CCCR 寄存器中的 CD 是否禁用。

0: 未禁用

1: 禁用

详情请参考 SDIO 规范。

(RO)

HINF_FUNC1_EPS 表示 SDIO FBR 寄存器中 function 1 的 EPS 字段的值。

0: function 1 运行在高电流模式下

1: function 1 运行在低电流模式下

详情请参考 SDIO 规范。

(RO)

HINF_EMP 表示 SDIO CCCR 寄存器中 EMPC 字段的值。

0: 主机功耗控制未使能

1: 主机功耗控制已使能

详情请参考 SDIO 规范。

(RO)

HINF_IOENABLE1 表示 SDIO CCCR 寄存器中 IOE1 字段的值。

0: function 1 未使能

1: function 1 已使能

详情请参考 SDIO 规范。

(RO)

HINF_SDIO_VER 配置 SDIO CCCR 中 SD bit[3:0]、SDIO bit[3:0]、CCCR bit[3:0] 的值。

HINF_SDIO_VER[11:8] 映射至 SD bit[3:0]

HINF_SDIO_VER[7:4] 映射至 SDIO bit[3:0]

HINF_SDIO_VER[3:0] 映射至 CCCR bit[3:0]

详情请参考 SDIO 规范。

(R/W)

HINF_FUNC2_EPS 表示 SDIO FBR 寄存器中 function 2 的 EPS 字段的值。

0: function 2 运行在高电流模式下

1: function 2 运行在低电流模式下

详情请参考 SDIO 规范。

(RO)

Register 29.3. HINF_CFG_DATA7_REG (0x001C)

(reserved)			HINF_SDIO_WAKEUP_CLR			(reserved)			HINF_ESDIO_DATA1_INT_EN			(reserved)			HINF_SDIO_RST			HINF_CHIP_STATE			HINF_PIN_STATE								
31	30	29													20	19	18	17	16	15				8	7				0
0	0	0x238												0	0x1	0	0x0			0x0			Reset						

HINF_PIN_STATE 配置 SDIO CIS 寄存器地址 318 和 574 的值，详情请参考 SDIO 规范。(R/W)

HINF_CHIP_STATE 配置 SDIO CIS 寄存器地址 312、315、568、571 的值，详情请参考 SDIO 规范。
(R/W)

HINF_SDIO_RST 配置是否重置 SDIO 从机模块。
0: 没有作用
1: 重置
(R/W)

HINF_ESDIO_DATA1_INT_EN 配置是否使能在 data1 线上的 SDIO 中断。
0: 不使能
1: 使能
(R/W)

HINF_SDIO_WAKEUP_CLR 配置是否在 SDIO 从机模块唤醒芯片系统后清除唤醒信号。
0: 没有作用
1: 清除
(WT)

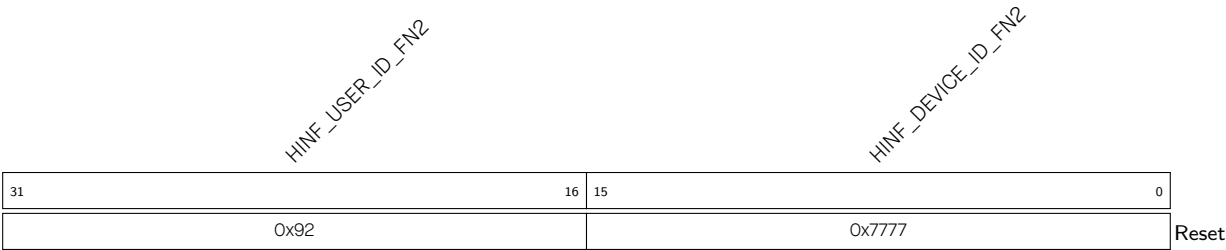
Register 29.4. HINF_CIS_CONF_Wn_REG(*n*: 0-7) (0x0020+0x4**n*)

HINF_CIS_CONF_Wn																															
31																															0
0xffffffff																															

Reset

HINF_CIS_CONF_Wn 配置 SDIO CIS 寄存器地址 (39+4**n*) ~ (36+4**n*) 的值，详情请参考 SDIO 规范。(R/W)

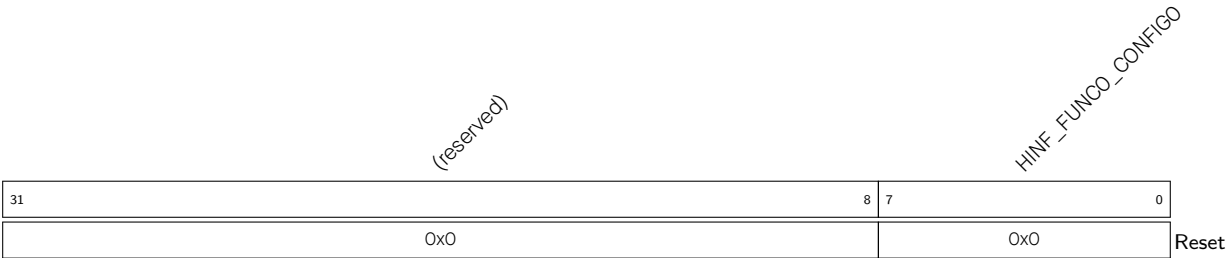
Register 29.5. HINF_CFG_DATA16_REG (0x0040)



HINF_DEVICE_ID_FN2 配置 function 2 在 SDIO CIS 寄存器中 device ID 字段的值。(R/W)

HINF_USER_ID_FN2 配置 function 2 在 SDIO CIS 寄存器中 user ID 字段的值。(R/W)

Register 29.6. HINF_CONF_STATUS_REG (0x0054)



HINF_FUNC0_CONFIG0 表示 SDIO CIS 寄存器 function 0 config0 (addr: 0x20f0) 字段状态，详情请参考 SDIO 规范。(RO)

29.9.2 SLC 寄存器

Register 29.7. SDIO_SLCCONFO_REG (0x0000)

(reserved) SDIO_SLC1_TOKEN_AUTO_CLR SDIO_SLC1_TXDATA_BURST_EN SDIO_SLC1_TXDSCR_BURST_EN (reserved) SDIO_SLC1_RXDATA_BURST_EN SDIO_SLC1_RXDSCR_BURST_EN SDIO_SLC1_RX_NO_RESTART_CLR SDIO_SLC1_RX_AUTO_WRBK SDIO_SLC1_TX_LOOP_TEST (reserved) SDIO_SLC1_RX_RST SDIO_SLC1_TX_RST (reserved) SDIO_SLC0_TOKEN_AUTO_CLR SDIO_SLC0_TXDATA_BURST_EN SDIO_SLC0_TXDSCR_BURST_EN SDIO_SLC0_RXDATA_BURST_EN SDIO_SLC0_RXDSCR_BURST_EN SDIO_SLC0_RX_NO_RESTART_CLR SDIO_SLC0_RX_AUTO_WRBK SDIO_SLC0_TX_LOOP_TEST (reserved) SDIO_SLC0_RX_RST SDIO_SLC0_TX_RST																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	Ox3	1	1	0	0	1	1	Ox3	0	0	1	1	1	1	Ox3	1	1	0	0	0	0	0	0	Ox0	0	0	Reset	

SDIO_SLC0_TX_RST 配置是否重置 SLC0 TX（主机至从机）有限状态机 (FSM)。

0: 没有作用

1: 重置

(R/W)

SDIO_SLC0_RX_RST 配置是否重置 SLC0 RX (从机至主机) FSM。

0: 没有作用

1: 重置

(R/W)

SDIO_SLC0_TX_LOOP_TEST 配置当从机 buffer 从主机接收完数据包时，SLC0 是否进行循环。

0: 不循环

1: 循环, 且硬件不改变链表中的 owner 位

(R/W)

SDIO_SLC0_RX_LOOP_TEST 配置当从机 buffer 向主机发送完数据包时，SLC0 是否进行循环。

0: 不循环

1: 循环, 且硬件不改变链表中的 owner 位

(R/W)

SDIO SLC0 RX AUTO WRBACK 配置 SCL0 是否回写 RX 链表的的 owner 位。

0: 不回写

1: 回写

(R/W)

SDIO_SLC0_RX_NO_RESTART_CLR 请初始化为1, 且不要修改。(R/W)

SDIO_SLC0_RXDSCR_BURST_EN 配置从存储器读取 RX 链表时, SLC0 是否允许使用 AHB burst 操作。

0: 仅使用单次 (single) 操作

1: 允许使用突发 (burst) 操作

(R/W)

见下页……

Register 29.7. SDIO_SLCCONFO_REG (0x0000)

[接上页……](#)

SDIO_SLC0_RXDATA_BURST_EN 配置从存储器读取数据时，SLC0 是否允许使用 AHB burst 操作。

- 0: 仅使用单次 (single) 操作
 - 1: 允许使用突发 (burst) 操作
- (R/W)

SDIO_SLC0_TXDSCR_BURST_EN 配置从存储器读取 TX 链表时，SLC0 是否允许使用 AHB burst 操作。

- 0: 仅使用单次 (single) 操作
 - 1: 允许使用突发 (burst) 操作
- (R/W)

SDIO_SLC0_TXDATA_BURST_EN 配置向存储器发送数据时，SLC0 是否允许使用 AHB burst 操作。

- 0: 仅使用单次 (single) 操作
 - 1: 允许使用突发 (burst) 操作
- (R/W)

SDIO_SLC0_TOKEN_AUTO_CLR 请初始化为 0，且不要修改。(R/W)

SDIO_SLC1_TX_RST 配置是否重置 SCL1 TX FSM。

- 0: 没有作用
 - 0: 重置
- (R/W)

SDIO_SLC1_RX_RST 配置是否重置 SCL1 RX FSM。

- 0: 没有作用
 - 0: 重置
- (R/W)

SDIO_SLC1_TX_LOOP_TEST 配置当从机 buffer 从主机接收完数据包时，SLC1 是否进行循环。

- 0: 不循环
 - 1: 循环，且硬件不改变链表中的 owner 位
- (R/W)

SDIO_SLC1_RX_LOOP_TEST 配置当从机 buffer 向主机发送完数据包时，SLC1 是否进行循环。

- 0: 不循环
 - 1: 循环，且硬件不改变链表中的 owner 位
- (R/W)

[见下页……](#)

Register 29.7. SDIO_SLCCONFO_REG (0x0000)

[接上页……](#)

SDIO_SLC1_RX_AUTO_WRBACK 配置 SCL1 是否回写 RX 链表的的 owner 位。

0: 不回写

1: 回写

(R/W)

SDIO_SLC1_RX_NO_RESTART_CLR 请初始化为 1，且不要修改。(R/W)

SDIO_SLC1_RXDSCR_BURST_EN 配置从存储器读取 RX 链表时，SLC1 是否允许使用 AHB burst 操作。

0: 仅使用单次 (single) 操作

1: 允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_RXDATA_BURST_EN 配置从存储器读取数据时，SLC1 是否允许使用 AHB burst 操作。

0: 仅使用单次 (single) 操作

1: 允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_TXDSCR_BURST_EN 配置从存储器读取 TX 链表时，SLC1 是否允许使用 AHB burst 操作。

0: 仅使用单次 (single) 操作

1: 允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_TXDATA_BURST_EN 配置向存储器发送数据时，SLC1 是否允许使用 AHB burst 操作。

0: 仅使用单次 (single) 操作

1: 允许使用突发 (burst) 操作

(R/W)

SDIO_SLC1_TOKEN_AUTO_CLR 请初始化为 0，且不要修改。(R/W)

Register 29.8. SDIO_SLCORX_LINK_REG (0x003C)

SDIO_SLC0_RXLINK_PARK					(reserved)																										
SDIO_SLC0_RXLINK_RESTART																															
SDIO_SLC0_RXLINK_START																															
SDIO_SLC0_RXLINK_STOP																															

31	30	29	28	0																											
1	0	0	0	0x0																											Reset

SDIO_SLCORXLINK_STOP 配置是否停止 SLCORX 链表操作。

0: 没有作用

1: 停止操作

(R/W/SC)

SDIO_SLCORXLINK_START 配置是否从 SDIO_SLCORXLINK_ADDR 指定的地址开始 SLCORX 链表操作。

0: 没有作用

1: 开始操作

(R/W/SC)

SDIO_SLCORXLINK_RESTART 配置是否重新启动并继续执行 SLCORX 链表操作。

0: 没有作用

1: 重新启动操作

(R/W/SC)

SDIO_SLCORXLINK_PARK 表示 SLCORX 链表 FSM 状态。

0: FSM 不处于空闲状态

1: FSM 处于空闲状态

(RO)

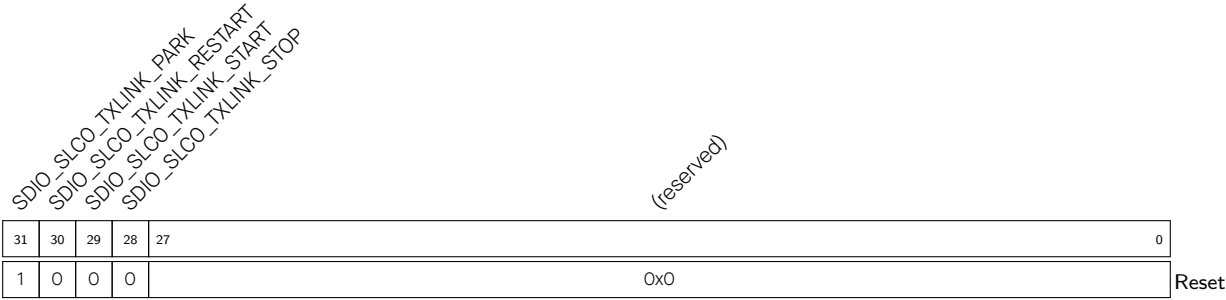
Register 29.9. SDIO_SLCORX_LINK_ADDR_REG (0x0040)

SDIO_SLC0_RXLINK_ADDR																																
31																															0	
0x0																																Reset

SDIO_SLCORXLINK_ADDR 表示 SLCORX 链表初始地址。

(R/W)

Register 29.10. SDIO_SLCOTX_LINK_REG (0x0044)



SDIO_SLCO_TXLINK_STOP 配置是否停止 SLCO TX 链表操作。

- 0: 没有作用
- 1: 停止操作
- (R/W/SC)

SDIO_SLCO_TXLINK_START 配置是否从 SDIO_SLCO_TXLINK_ADDR 指定的地址开始 SLCO TX 链表操作。

- 0: 没有作用
- 1: 开始操作
- (R/W/SC)

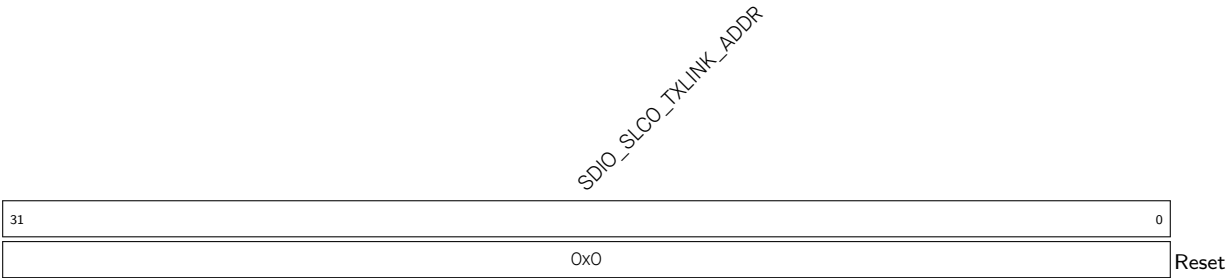
SDIO_SLCO_TXLINK_RESTART 配置是否重新启动并继续执行 SLCO TX 链表操作。

- 0: 没有作用
- 1: 重新启动操作
- (R/W/SC)

SDIO_SLCO_TXLINK_PARK 表示 SLCO TX 链表 FSM 状态。

- 0: FSM 不处于空闲状态
- 1: FSM 处于空闲状态
- (RO)

Register 29.11. SDIO_SLCOTX_LINK_ADDR_REG (0x0048)



SDIO_SLCO_TXLINK_ADDR 表示 SLCO TX 链表初始地址。

- (R/W)

Register 29.12. SDIO_SLC1RX_LINK_REG (0x004C)

31	30	29	28	27	0
1	0	0	0	0x1000000	Reset

SDIO_SLC1_RXLINK_STOP 配置是否停止 SLC1 RX 链表操作。

0: 没有作用

1: 停止操作

(R/W/SC)

SDIO_SLC1_RXLINK_START 配置是否从 SDIO_SLC1_RXLINK_ADDR 指定的地址开始 SLC1 RX 链表操作。

0: 没有作用

1: 开始操作

(R/W/SC)

SDIO_SLC1_RXLINK_RESTART 配置是否重新启动并继续执行 SLC1 RX 链表操作。

0: 没有作用

1: 重新启动操作

(R/W/SC)

SDIO_SLC1_RXLINK_PARK 表示 SLC1 RX 链表 FSM 状态。

0: FSM 不处于空闲状态

1: FSM 处于空闲状态

(RO)

Register 29.13. SDIO_SLC1RX_LINK_ADDR_REG (0x0050)

SDIO_SLC1_RXLINK_ADDR

31 0

0x0 Reset

SDIO_SLC1_RXLINK_ADDR 表示 SLC1 RX 链表初始地址。

(R/W)

Register 29.14. SDIO_SLC1TX_LINK_REG (0x0054)

31	30	29	28	27	0
1	0	0	0	0x0	Reset

SDIO_SLC1_TXLINK_STOP 配置是否停止 SLC1 TX 链表操作。

0: 没有作用

1: 停止操作

(R/W/SC)

SDIO_SLC1_TXLINK_START 配置是否从 SDIO_SLC1_TXLINK_ADDR 指定的地址开始 SLC1 TX 链表操作。

0: 没有作用

1: 开始操作

(R/W/SC)

SDIO_SLC1_TXLINK_RESTART 配置是否重新启动并继续执行 SLC1 TX 链表操作。

0: 没有作用

1: 重新启动操作

(R/W/SC)

SDIO SLC1 TXLINK PARK 表示 SLC1 TX 链表 FSM 状态。

0: FSM 不处于空闲状态

1: FSM 处于空闲状态

(RO)

Register 29.15. SDIO_SLC1TX_LINK_ADDR_REG (0x0058)

SDIO_SLC1_TYLINK_ADDR

31 0

0x0 Reset

SDIO_SLC1_TXLINK_ADDR 表示 SLC1 TX 链表初始地址。(R/W)

Register 29.16. SDIO_SLC0TOKEN1_REG (0x0064)

(reserved)				SDIO_SLC0_TOKEN1				(reserved)				SDIO_SLC0_TOKEN1_INC_MORE SDIO_SLC0_TOKEN1_INC SDIO_SLC0_TOKEN1_WR				SDIO_SLC0_TOKEN1_WDATA			
31	28	27					16	15	14	13	12	11							0
0x0				0x0				0	0	0	0	0x0							

Reset

SDIO_SLC0_TOKEN1_WDATA 配置 SLC0 token1 的值。(WT)

SDIO_SLC0_TOKEN1_WR 设置该位为 1 后, SDIO_SLC0_TOKEN1_WDATA 的值将写入 SDIO_SLC0_TOKEN1。(WT)

SDIO_SLC0_TOKEN1_INC 设置该位为 1 后, SDIO_SLC0_TOKEN1 的值将加 1。(WT)

SDIO_SLC0_TOKEN1_INC_MORE 设置该位为 1 后, SDIO_SLC0_TOKEN1_WDATA 的值将与 SDIO_SLC0_TOKEN1 的值相加。(WT)

SDIO_SLC0_TOKEN1 表示可用于通过 SLC0 通道接收数据包的 buffer 的累计值。(RO)

Register 29.17. SDIO_SLC1TOKEN1_REG (0x006C)

(reserved)				SDIO_SLC1_TOKEN1				(reserved)				SDIO_SLC1_TOKEN1_INC_MORE SDIO_SLC1_TOKEN1_INC SDIO_SLC1_TOKEN1_WR				SDIO_SLC1_TOKEN1_WDATA			
31	28	27					16	15	14	13	12	11							0
0x0				0x0				0	0	0	0	0x0							

Reset

SDIO_SLC1_TOKEN1_WDATA 配置 SLC1 token1 的值。(WT)

SDIO_SLC1_TOKEN1_WR 设置该位为 1 后, SDIO_SLC1_TOKEN1_WDATA 的值将写入 SDIO_SLC1_TOKEN1。(WT)

SDIO_SLC1_TOKEN1_INC 设置该位为 1 后, SDIO_SLC1_TOKEN1 的值将加 1。(WT)

SDIO_SLC1_TOKEN1_INC_MORE 设置该位为 1 后, SDIO_SLC1_TOKEN1_WDATA 的值将与 SDIO_SLC1_TOKEN1 的值相加。(WT)

SDIO_SLC1_TOKEN1 表示可用于通过 SLC1 通道接收数据包的 buffer 的累计值。(RO)

Register 29.18. SDIO_SLCCONF1_REG (0x0070)

(reserved)										SDIO_SLC1_RX_STITCH_EN SDIO_SLC1_TX_STITCH_EN SDIO_HOST_INT_LEVEL_SEL										(reserved)										SDIO_SLC0_RX_STITCH_EN SDIO_SLC0_TX_STITCH_EN SDIO_SLC0_LEN_AUTO_CLR SDIO_SDIO_CMD_HOLD_EN										(reserved)									
31																				7	6	5	4	3	2						0																		
0x0										1	1	0	0x0										1	1	1	1	0x0					Reset																	

SDIO_CMD_HOLD_EN 请初始化为 0，且不要修改。(R/W)

SDIO_SLC0_LEN_AUTO_CLR 请初始化为 0，且不要修改。(R/W)

SDIO_SLC0_TX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

SDIO_SLC0_RX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

SDIO_HOST_INT_LEVEL_SEL 配置发送至主机的中断的极性。

0: 低电平中断有效

1: 高电平中断有效

(R/W)

SDIO_SLC1_TX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

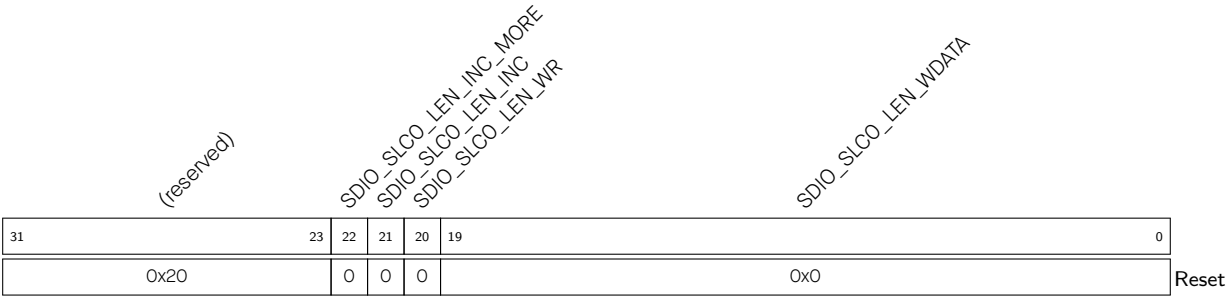
SDIO_SLC1_RX_STITCH_EN 请初始化为 0，且不要修改。(R/W)

Register 29.19. SDIO_SLC_RX_DSCR_CONF_REG (0x00A8)

(reserved)																															SDIO_SLC0_TOKEN_NO_REPLACE									
31																															1	0								
0x101b80d																															0	Reset								

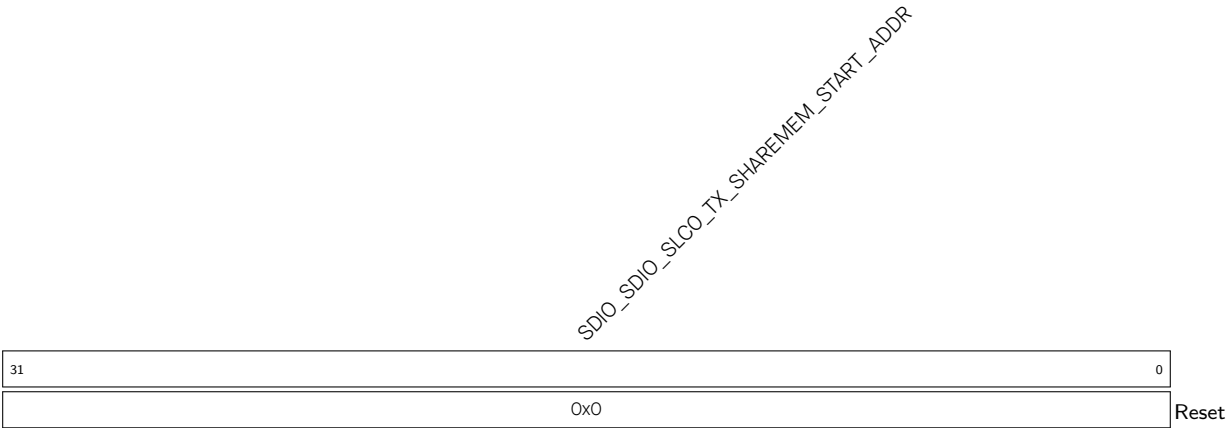
SDIO_SLC0_TOKEN_NO_REPLACE 请初始化为 1，且不要修改。(R/W)

Register 29.20. SDIO_SLCO_LEN_CONF_REG (0x00F4)



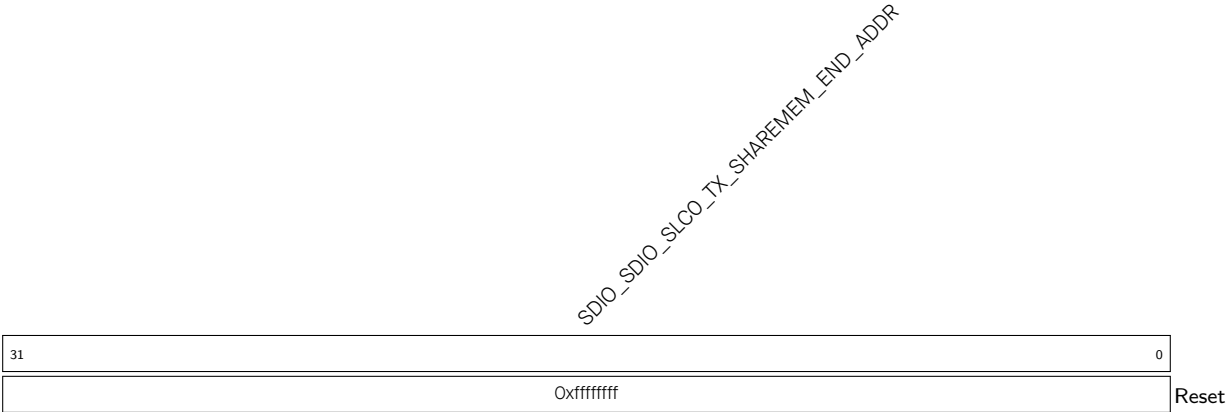
- SDIO_SLCO_LEN_WDATA** 配置从机想要发送的数据的长度。(WT)
- SDIO_SLCO_LEN_WR** 设置该位为 1 后, SDIO_SLCO_LEN_WDATA 的值将写入 SDIO_SLCO_LEN 和 SLCHOST_HOSTSLCHOST_SLCO_LEN。(WT)
- SDIO_SLCO_LEN_INC** 设置该位为 1 后, SDIO_SLCO_LEN 和 SLCHOST_HOSTSLCHOST_SLCO_LEN 的值将分别加 1。(WT)
- SDIO_SLCO_LEN_INC_MORE** 设置该位为 1 后, SDIO_SLCO_LEN 和 SLCHOST_HOSTSLCHOST_SLCO_LEN 的值将分别加上 SDIO_SLCO_LEN_WDATA 的值。(WT)

Register 29.21. SDIO_SLCO_TX_SHAREMEM_START_REG (0x0154)



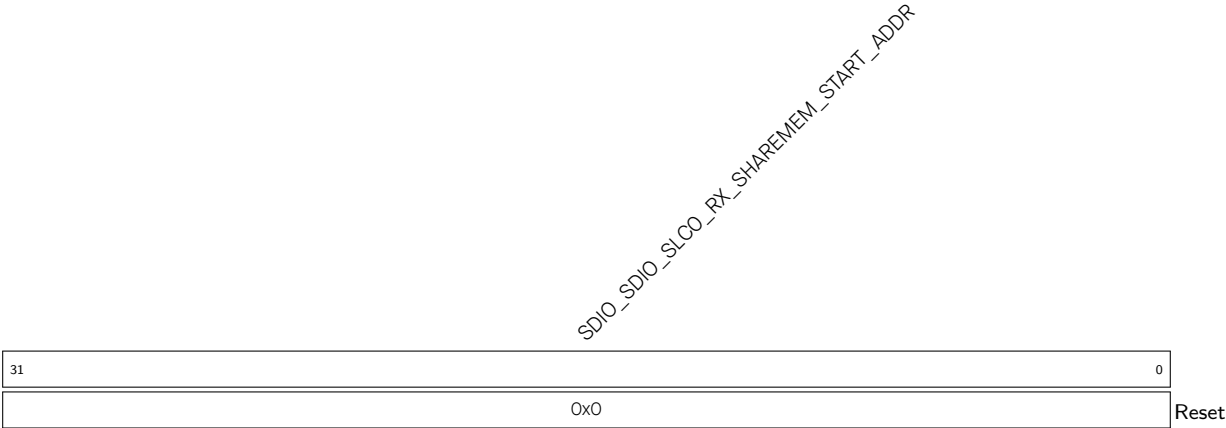
- SDIO_SDIO_SLCO_TX_SHAREMEM_START_ADDR** 配置 SLCO 主机至从机通道 AHB 起始地址边界。
(R/W)

Register 29.22. SDIO_SLC0_TX_SHAREMEM_END_REG (0x0158)



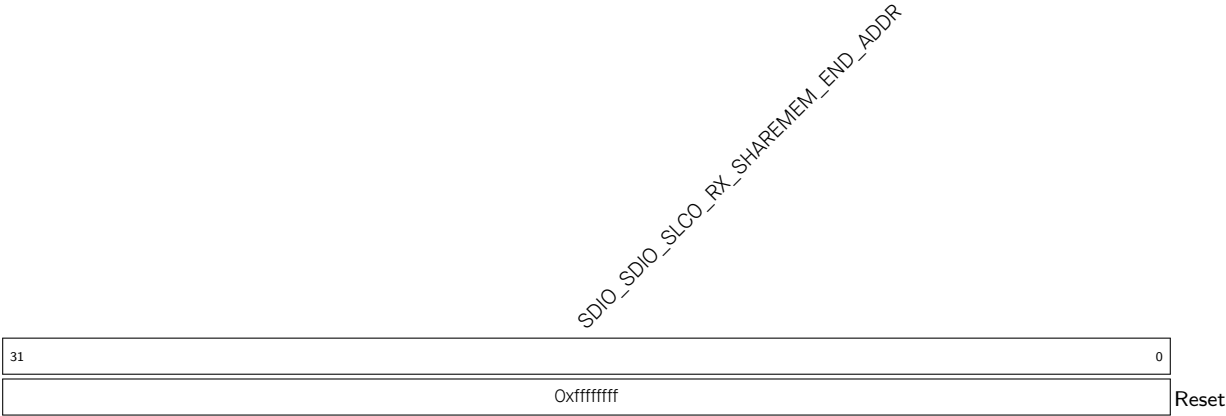
SDIO_SDIO_SLC0_TX_SHAREMEM_END_ADDR 配置 SLC0 主机至从机通道 AHB 结束地址边界。
(R/W)

Register 29.23. SDIO_SLC0_RX_SHAREMEM_START_REG (0x015C)



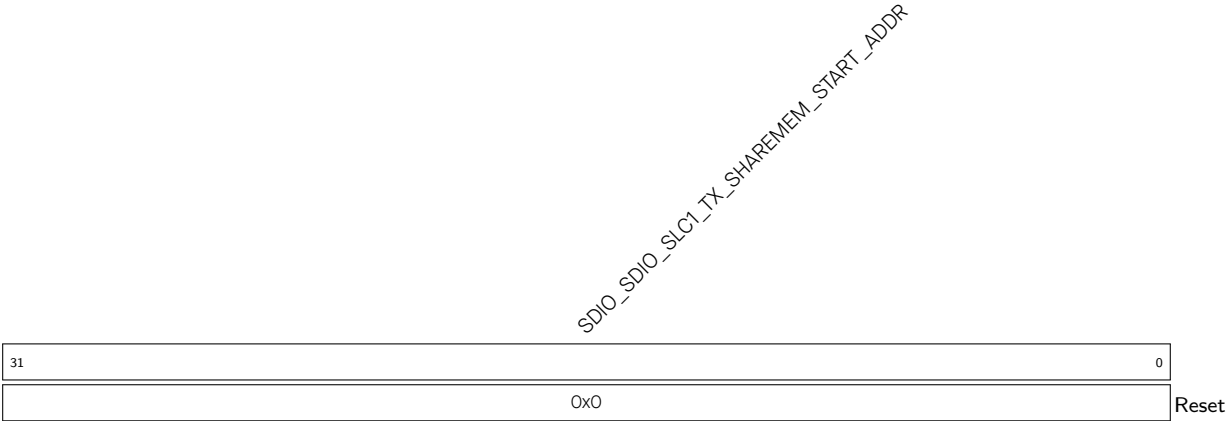
SDIO_SDIO_SLC0_RX_SHAREMEM_START_ADDR 配置 SLC0 从机至主机通道 AHB 起始地址边界。
(R/W)

Register 29.24. SDIO_SLC0_RX_SHAREMEM_END_REG (0x0160)



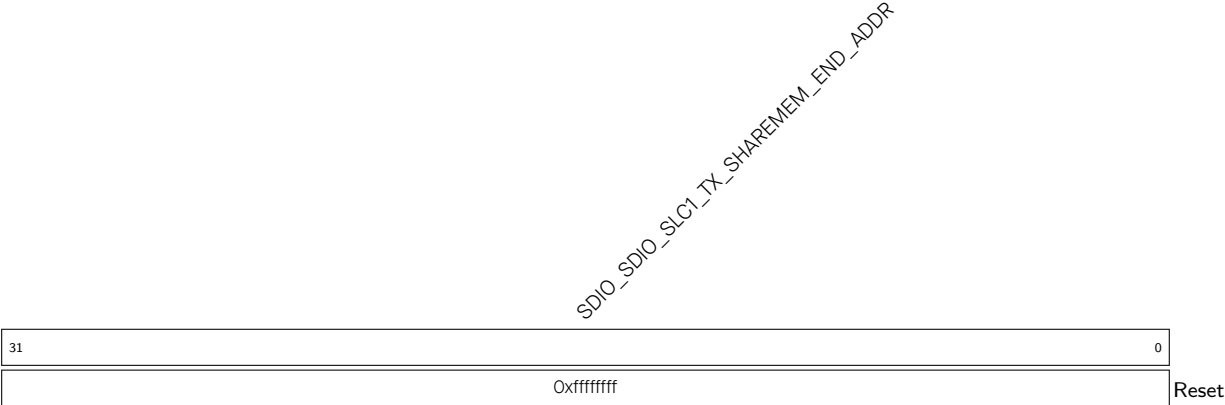
SDIO_SDIO_SLC0_RX_SHAREMEM_END_ADDR 配置 SLC0 从机至主机通道 AHB 结束地址边界。
(R/W)

Register 29.25. SDIO_SLC1_TX_SHAREMEM_START_REG (0x0164)



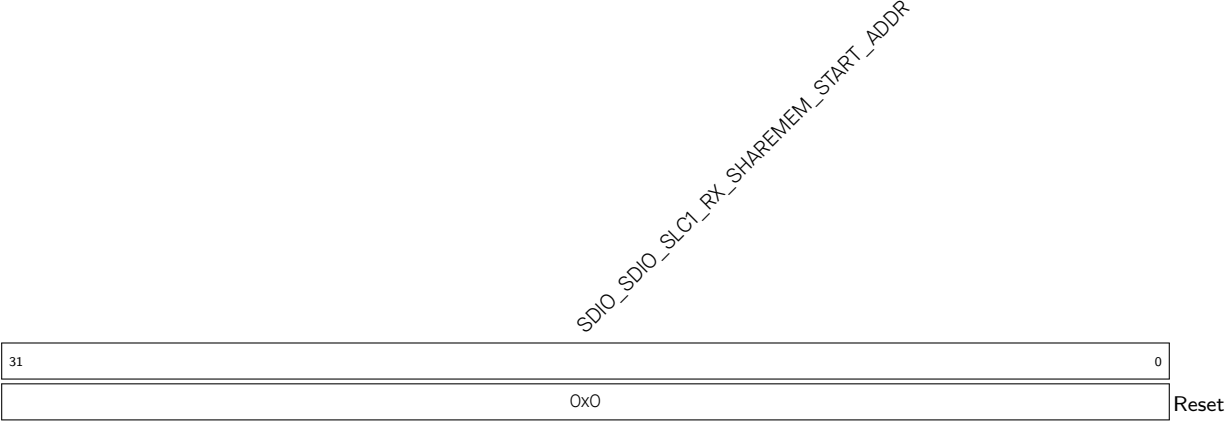
SDIO_SDIO_SLC1_TX_SHAREMEM_START_ADDR 配置 SLC1 主机至从机通道 AHB 起始地址边界。
(R/W)

Register 29.26. SDIO_SLC1_TX_SHAREMEM_END_REG (0x0168)



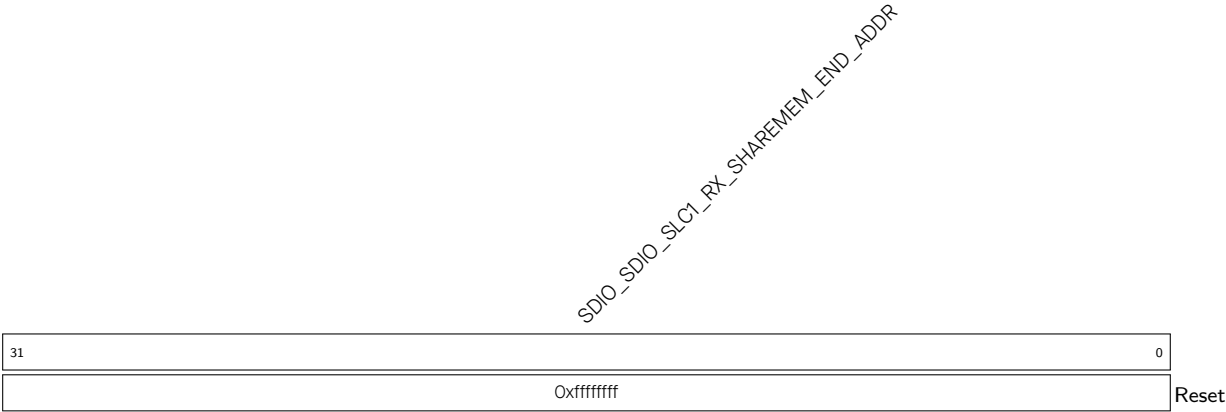
`SDIO_SDIO_SLC1_TX_SHAREMEM_END_ADDR` 配置 SLC1 主机至从机通道 AHB 结束地址边界。(R/W)

Register 29.27. SDIO_SLC1_RX_SHAREMEM_START_REG (0x016C)



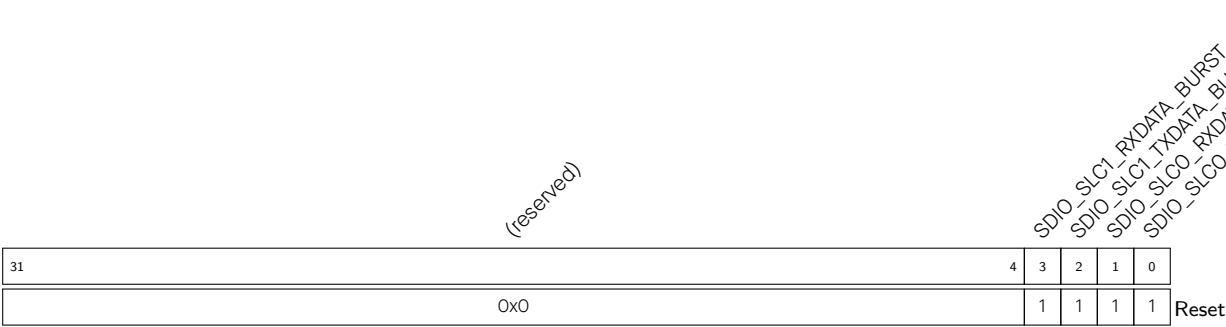
`SDIO_SDIO_SLC1_RX_SHAREMEM_START_ADDR` 配置 SLC1 从机至主机通道 AHB 起始地址边界。(R/W)

Register 29.28. SDIO_SLC1_RX_SHAREMEM_END_REG (0x0170)



SDIO_SLC1_RX_SHAREMEM_END_ADDR 配置 SLC1 从机至主机通道 AHB 结束地址边界。(R/W)

Register 29.29. SDIO_SLC_BURST_LEN_REG (0x017C)



SDIO_SLC0_TXDATA_BURST_LEN 配置 SLC0 主机至从机通道 AHB burst 类型。

0: 可以使用 incr4

1: 可以使用 incr8

(R/W)

SDIO_SLC0_RXDATA_BURST_LEN 配置 SLC0 从机至主机通道 AHB burst 类型。

0: 可以使用 incr 和 incr4

1: 可以使用 incr 和 incr8

(R/W)

SDIO_SLC1_TXDATA_BURST_LEN 配置 SLC1 主机至从机通道 AHB burst 类型。

0: 可以使用 incr4

1: 可以使用 incr8

(R/W)

SDIO_SLC1_RXDATA_BURST_LEN 配置 SLC1 从机至主机通道 AHB burst 类型。

0: 可以使用 incr 和 incr4

1: 可以使用 incr 和 incr8

(R/W)

Register 29.30. SDIO_SLC0INT_RAW_REG (0x0004)

(reserved)		SDIO_SLC0_RX_DSCR_ERR_INT_RAW		SDIO_SLC0_TX_DSCR_ERR_INT_RAW		(reserved)		SDIO_SLC0_RX_EOF_INT_RAW		SDIO_SLC0_RX_DONE_INT_RAW		SDIO_SLC0_TX_SUC_EOF_INT_RAW		SDIO_SLC0_TX_DONE_INT_RAW		(reserved)		SDIO_SLC0_TX_OVF_INT_RAW		SDIO_SLC0_RX_UDF_INT_RAW		SDIO_SLC0_TX_START_INT_RAW		SDIO_SLC_FRHOST_BIT7_INT_RAW		SDIO_SLC_FRHOST_BIT6_INT_RAW		SDIO_SLC_FRHOST_BIT5_INT_RAW		SDIO_SLC_FRHOST_BIT4_INT_RAW		SDIO_SLC_FRHOST_BIT3_INT_RAW		SDIO_SLC_FRHOST_BIT2_INT_RAW		SDIO_SLC_FRHOST_BIT1_INT_RAW		SDIO_SLC_FRHOST_BIT0_INT_RAW	
31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																	
0x0		0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset			

SDIO_SLC_FRH0ST_BIT n _INT_RAW (n : 0-7) SLC_FRH0ST_BIT n _INT (n : 0-7) 的原始中断状态。
(R/WTC/SS)

SDIO_SLC0_RX_START_INT_RAW SLC0_RX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_START_INT_RAW SLC0_TX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_UDF_INT_RAW SLC0_RX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_OVF_INT_RAW SLC0_TX_OVF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_DONE_INT_RAW SLC0_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_SUC_EOF_INT_RAW SLC0_TX_SUC_EOF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_DONE_INT_RAW SLC0_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_EOF_INT_RAW SLC0_RX_EOF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_TX_DSCR_ERR_INT_RAW SLC0_TX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC0_RX_DSCR_ERR_INT_RAW SLC0_RX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

Register 29.31. SDIO_SLC0INT_ST_REG (0x0008)

(reserved)												SDIO_SLC0_RX_DSCR_ERR_INT_ST SDIO_SLC0_TX_DSCR_ERR_INT_ST (reserved) SDIO_SLC0_RX_EOF_INT_ST SDIO_SLC0_RX_DONE_INT_ST SDIO_SLC0_TX_SUC_EOF_INT_ST (reserved) SDIO_SLC0_TX_OVF_INT_ST SDIO_SLC0_RX_UDF_INT_ST SDIO_SLC0_TX_START_INT_ST SDIO_SLC_FRHOST_BIT7_INT_ST SDIO_SLC_FRHOST_BIT6_INT_ST SDIO_SLC_FRHOST_BIT5_INT_ST SDIO_SLC_FRHOST_BIT4_INT_ST SDIO_SLC_FRHOST_BIT3_INT_ST SDIO_SLC_FRHOST_BIT2_INT_ST SDIO_SLC_FRHOST_BIT1_INT_ST SDIO_SLC_FRHOST_BIT0_INT_ST																								
31											21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x0												0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SDIO_SLC_FRHOST_BIT n _INT_ST (n : 0-7) SLC_FRHOST_BIT n _INT (n : 0-7) 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_START_INT_ST SLC0_RX_START_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_START_INT_ST SLC0_TX_START_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_UDF_INT_ST SLC0_RX_UDF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_OVF_INT_ST SLC0_TX_OVF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_DONE_INT_ST SLC0_TX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_SUC_EOF_INT_ST SLC0_TX_SUC_EOF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_DONE_INT_ST SLC0_RX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_EOF_INT_ST SLC0_RX_EOF_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_TX_DSCR_ERR_INT_ST SLC0_TX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

SDIO_SLC0_RX_DSCR_ERR_INT_ST SLC0_RX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

Register 29.32. SDIO_SLC0INT_ENA_REG (0x000C)

(reserved)																		SDIO_SLC0_RX_DSCR_ERR_INT_ENA SDIO_SLC0_TX_DSCR_ERR_INT_ENA (reserved) SDIO_SLC0_RX_EOF_INT_ENA SDIO_SLC0_RX_DONE_INT_ENA SDIO_SLC0_TX_SUC_EOF_INT_ENA SDIO_SLC0_TX_DONE_INT_ENA (reserved) SDIO_SLC0_TX_OVF_INT_ENA SDIO_SLC0_RX_UDF_INT_ENA SDIO_SLC0_RX_START_INT_ENA SDIO_SLC0_RX_FRHOST_INT_ENA SDIO_SLC0_FRHOST_BIT7_INT_ENA SDIO_SLC0_FRHOST_BIT6_INT_ENA SDIO_SLC0_FRHOST_BIT5_INT_ENA SDIO_SLC0_FRHOST_BIT4_INT_ENA SDIO_SLC0_FRHOST_BIT3_INT_ENA SDIO_SLC0_FRHOST_BIT2_INT_ENA SDIO_SLC0_FRHOST_BIT1_INT_ENA SDIO_SLC0_FRHOST_BIT0_INT_ENA																	
31											21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x0											0	0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset		

SDIO_SLC_FRHOST_BIT n _INT_ENA (n : 0-7) 写 1 使能 SLC_FRHOST_BIT n _INT (n : 0-7) 中断。(R/W)

SDIO_SLC0_RX_START_INT_ENA 写 1 使能 SLC0_RX_START_INT 中断。(R/W)

SDIO_SLC0_TX_START_INT_ENA 写 1 使能 SLC0_TX_START_INT 中断。(R/W)

SDIO_SLC0_RX_UDF_INT_ENA 写 1 使能 SLC0_RX_UDF_INT 中断。(R/W)

SDIO_SLC0_TX_OVF_INT_ENA 写 1 使能 SLC0_TX_OVF_INT 中断。(R/W)

SDIO_SLC0_TX_DONE_INT_ENA 写 1 使能 SLC0_TX_DONE_INT 中断。(R/W)

SDIO_SLC0_TX_SUC_EOF_INT_ENA 写 1 使能 SLC0_TX_SUC_EOF_INT 中断。(R/W)

SDIO_SLC0_RX_DONE_INT_ENA 写 1 使能 SLC0_RX_DONE_INT 中断。(R/W)

SDIO_SLC0_RX_EOF_INT_ENA 写 1 使能 SLC0_RX_EOF_INT 中断。(R/W)

SDIO_SLC0_TX_DSCR_ERR_INT_ENA 写 1 使能 SLC0_TX_DSCR_ERR_INT 中断。(R/W)

SDIO_SLC0_RX_DSCR_ERR_INT_ENA 写 1 使能 SLC0_RX_DSCR_ERR_INT 中断。(R/W)

Register 29.33. SDIO_SLC0INT_CLR_REG (0x0010)

(reserved)												SDIO_SLC0_RX_DSCR_ERR_INT_CLR SDIO_SLC0_TX_DSCR_ERR_INT_CLR (reserved) SDIO_SLC0_RX_EOF_INT_CLR SDIO_SLC0_RX_DONE_INT_CLR SDIO_SLC0_TX_SUC_EOF_INT_CLR SDIO_SLC0_TX_DONE_INT_CLR (reserved) SDIO_SLC0_TX_OVF_INT_CLR SDIO_SLC0_RX_UDF_INT_CLR SDIO_SLC0_RX_START_INT_CLR SDIO_SLC_FRHOST_BIT7_INT_CLR SDIO_SLC_FRHOST_BIT6_INT_CLR SDIO_SLC_FRHOST_BIT5_INT_CLR SDIO_SLC_FRHOST_BIT4_INT_CLR SDIO_SLC_FRHOST_BIT3_INT_CLR SDIO_SLC_FRHOST_BIT2_INT_CLR SDIO_SLC_FRHOST_BIT1_INT_CLR SDIO_SLC_FRHOST_BIT0_INT_CLR																								
31											21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
0x0												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SDIO_SLC_FRHOST_BIT n _INT_CLR (n : 0-7) 写 1 清除中断 SLC_FRHOST_BIT n _INT (n : 0-7)。(WT)

SDIO_SLC0_RX_START_INT_CLR 写 1 清除中断 SLC0_RX_START_INT。(WT)

SDIO_SLC0_TX_START_INT_CLR 写 1 清除中断 SLC0_TX_START_INT。(WT)

SDIO_SLC0_RX_UDF_INT_CLR 写 1 清除中断 SLC0_RX_UDF_INT。(WT)

SDIO_SLC0_TX_OVF_INT_CLR 写 1 清除中断 SLC0_TX_OVF_INT。(WT)

SDIO_SLC0_TX_DONE_INT_CLR 写 1 清除中断 SLC0_TX_DONE_INT。(WT)

SDIO_SLC0_TX_SUC_EOF_INT_CLR 写 1 清除中断 SLC0_TX_SUC_EOF_INT。(WT)

SDIO_SLC0_RX_DONE_INT_CLR 写 1 清除中断 SLC0_RX_DONE_INT。(WT)

SDIO_SLC0_RX_EOF_INT_CLR 写 1 清除中断 SLC0_RX_EOF_INT。(WT)

SDIO_SLC0_TX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC0_TX_DSCR_ERR_INT。(WT)

SDIO_SLC0_RX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC0_RX_DSCR_ERR_INT。(WT)

Register 29.34. SDIO_SLC1INT_RAW_REG (0x0014)

(reserved)												SDIO_SLC1_RX_DSCR_ERR_INT_RAW SDIO_SLC1_TX_DSCR_ERR_INT_RAW (reserved) SDIO_SLC1_RX_EOF_INT_RAW SDIO_SLC1_RX_DONE_INT_RAW SDIO_SLC1_TX_SUC_EOF_INT_RAW SDIO_SLC1_TX_DONE_INT_RAW (reserved) SDIO_SLC1_TX_OVF_INT_RAW SDIO_SLC1_RX_UDF_INT_RAW SDIO_SLC1_TX_START_INT_RAW SDIO_SLC1_RX_START_INT_RAW SDIO_SLC_FRHST_BIT15_INT_RAW SDIO_SLC_FRHST_BIT14_INT_RAW SDIO_SLC_FRHST_BIT13_INT_RAW SDIO_SLC_FRHST_BIT12_INT_RAW SDIO_SLC_FRHST_BIT11_INT_RAW SDIO_SLC_FRHST_BIT10_INT_RAW SDIO_SLC_FRHST_BIT9_INT_RAW SDIO_SLC_FRHST_BIT8_INT_RAW																							
31											21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x0												0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

SDIO_SLC_FRHOST_BIT n _INT_RAW (n : 8-15) SLC_FRHOST_BIT n _INT (n : 8-15) 的原始中断状态。
(R/WTC/SS)

SDIO_SLC1_RX_START_INT_RAW SLC1_RX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_START_INT_RAW SLC1_TX_START_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_UDF_INT_RAW SLC1_RX_UDF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_OVF_INT_RAW SLC1_TX_OVF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_DONE_INT_RAW SLC1_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_SUC_EOF_INT_RAW SLC1_TX_SUC_EOF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_DONE_INT_RAW SLC1_RX_DONE_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_EOF_INT_RAW SLC1_RX_EOF_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_TX_DSCR_ERR_INT_RAW SLC1_TX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

SDIO_SLC1_RX_DSCR_ERR_INT_RAW SLC1_RX_DSCR_ERR_INT 的原始中断状态。(R/WTC/SS)

Register 29.35. SDIO_SLC1INT_CLR_REG (0x0020)

(reserved)																SDIO_SLC1_RX_DSCR_ERR_INT_CLR SDIO_SLC1_TX_DSCR_ERR_INT_CLR (reserved) SDIO_SLC1_RX_EOF_INT_CLR SDIO_SLC1_RX_DONE_INT_CLR SDIO_SLC1_TX_SUC_EOF_INT_CLR (reserved) SDIO_SLC1_TX_OVF_INT_CLR SDIO_SLC1_RX_UDF_INT_CLR SDIO_SLC1_TX_START_INT_CLR SDIO_SLC1_RX_START_INT_CLR SDIO_SLC1_FRHOST_BIT15_INT_CLR SDIO_SLC1_FRHOST_BIT14_INT_CLR SDIO_SLC1_FRHOST_BIT13_INT_CLR SDIO_SLC1_FRHOST_BIT12_INT_CLR SDIO_SLC1_FRHOST_BIT11_INT_CLR SDIO_SLC1_FRHOST_BIT10_INT_CLR SDIO_SLC1_FRHOST_BIT9_INT_CLR SDIO_SLC1_FRHOST_BIT8_INT_CLR																					
31																21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0								0	0	0	0	0	0	0	0x0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset							

SDIO_SLC_FRHOST_BIT n _INT_CLR (n : 8-15) 写 1 清除中断 SLC_FRHOST_BIT n _INT (n : 8-15)。(WT)

SDIO_SLC1_RX_START_INT_CLR 写 1 清除中断 SLC1_RX_START_INT。(WT)

SDIO_SLC1_TX_START_INT_CLR 写 1 清除中断 SLC1_TX_START_INT。(WT)

SDIO_SLC1_RX_UDF_INT_CLR 写 1 清除中断 SLC1_RX_UDF_INT。(WT)

SDIO_SLC1_TX_OVF_INT_CLR 写 1 清除中断 SLC1_TX_OVF_INT。(WT)

SDIO_SLC1_TX_DONE_INT_CLR 写 1 清除中断 SLC1_TX_DONE_INT。(WT)

SDIO_SLC1_TX_SUC_EOF_INT_CLR 写 1 清除中断 SLC1_TX_SUC_EOF_INT。(WT)

SDIO_SLC1_RX_DONE_INT_CLR 写 1 清除中断 SLC1_RX_DONE_INT。(WT)

SDIO_SLC1_RX_EOF_INT_CLR 写 1 清除中断 SLC1_RX_EOF_INT。(WT)

SDIO_SLC1_TX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC1_TX_DSCR_ERR_INT。(WT)

SDIO_SLC1_RX_DSCR_ERR_INT_CLR 写 1 清除中断 SLC1_RX_DSCR_ERR_INT。(WT)

Register 29.36. SDIO_SLCINTVEC_TOHOST_REG (0x005C)

(reserved)																SDIO_SLC1_TOHOST_INTVEC																(reserved)																SDIO_SLC0_TOHOST_INTVEC																			
31																24	23																16	15																8	7																0
0x0																0x0																0x0																0x0																Reset			

SDIO_SLC0_TOHOST_INTVEC SLC0_TOHOST_BIT n _INT (n : 0-7) 中断设置位。这些位不会被自动清除。(WT)

SDIO_SLC1_TOHOST_INTVEC SLC1_TOHOST_BIT n _INT (n : 0-7) 中断设置位。这些位不会被自动清除。(WT)

Register 29.37. SDIO_SLC1INT_ST1_REG (0x014C)

[illegible]

SDIO_SLC_FRHOST_BIT n _INT_ST1 (n : 8-15) SLC_FRHOST_BIT n _INT (n : 8-15) 的屏蔽中断状态。
(RO)

SDIO_SLC1_RX_START_INT_ST1 **SLC1_RX_START_INT** 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_START_INT_ST1 SLC1_TX_START_INT 的屏蔽中断状态。(RO)

SDIO SLC1 RX UDF INT ST1 SLC1 RX UDF INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_OVF_INT_ST1 SLC1_TX_OVF_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_DONE_INT_ST1 SLC1_TX_DONE_INT 的屏蔽中断状态。(RO)

SDIO SLC1 TX SUC EOF INT ST1 SLC1 TX SUC EOF INT 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_DONE_INT_ST1 SLC1_RX_DONE_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_EOF_INT_ST1 SLC1_RX_EOF_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_TX_DSCR_ERR_INT_ST1 SLC1_TX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

SDIO_SLC1_RX_DSCR_ERR_INT_ST1 SLC1_RX_DSCR_ERR_INT 的屏蔽中断状态。(RO)

Register 29.38. SDIO_SLC1INT_ENA1_REG (0x0150)

(reserved)												SDIO_SLC1_RX_DSCR_ERR_INT_ENA1 SDIO_SLC1_TX_DSCR_ERR_INT_ENA1 (reserved) SDIO_SLC1_RX_EOF_INT_ENA1 SDIO_SLC1_RX_DONE_INT_ENA1 SDIO_SLC1_TX_SUC_EOF_INT_ENA1 SDIO_SLC1_TX_DONE_INT_ENA1 (reserved) SDIO_SLC1_TX_OVF_INT_ENA1 SDIO_SLC1_RX_UDF_INT_ENA1 SDIO_SLC1_TX_START_INT_ENA1 SDIO_SLC1_RX_START_INT_ENA1 SDIO_SLC_FRHST_BIT15_INT_ENA1 SDIO_SLC_FRHST_BIT14_INT_ENA1 SDIO_SLC_FRHST_BIT13_INT_ENA1 SDIO_SLC_FRHST_BIT12_INT_ENA1 SDIO_SLC_FRHST_BIT11_INT_ENA1 SDIO_SLC_FRHST_BIT10_INT_ENA1 SDIO_SLC_FRHST_BIT9_INT_ENA1 SDIO_SLC_FRHST_BIT8_INT_ENA1											
31	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0x0												0	0	0	0	0	0	0	0	0	0	0	0

SDIO_SLC_FRHST_BIT n _INT_ENA1 (n : 8-15) 写 1 使能 SLC_FRHST_BIT n _INT (n : 8-15) 中断。
(R/W)

SDIO_SLC1_RX_START_INT_ENA1 写 1 使能 SLC1_RX_START_INT 中断。(R/W)

SDIO_SLC1_TX_START_INT_ENA1 写 1 使能 SLC1_TX_START_INT 中断。(R/W)

SDIO_SLC1_RX_UDF_INT_ENA1 写 1 使能 SLC1_RX_UDF_INT 中断。(R/W)

SDIO_SLC1_TX_OVF_INT_ENA1 写 1 使能 SLC1_TX_OVF_INT 中断。(R/W)

SDIO_SLC1_TX_DONE_INT_ENA1 写 1 使能 SLC1_TX_DONE_INT 中断。(R/W)

SDIO_SLC1_TX_SUC_EOF_INT_ENA1 写 1 使能 SLC1_TX_SUC_EOF_INT 中断。(R/W)

SDIO_SLC1_RX_DONE_INT_ENA1 写 1 使能 SLC1_RX_DONE_INT 中断。(R/W)

SDIO_SLC1_RX_EOF_INT_ENA1 写 1 使能 SLC1_RX_EOF_INT 中断。(R/W)

SDIO_SLC1_TX_DSCR_ERR_INT_ENA1 写 1 使能 SLC1_TX_DSCR_ERR_INT 中断。(R/W)

SDIO_SLC1_RX_DSCR_ERR_INT_ENA1 写 1 使能 SLC1_RX_DSCR_ERR_INT 中断。(R/W)

Register 29.39. SDIO_SLC0_LENGTH_REG (0x00F8)

(reserved)												SDIO_SLC0_LEN												
3120												190												
0x0												0x0												Reset

SDIO_SLC0_LEN 表示从机希望发送的数据总长度。(RO)

29.9.3 SLC Host 寄存器

Register 29.40. SLCHOST_CONF_REG (0x01F0)

(reserved)				SLCHOST_HSPEED_CON_EN				(reserved)				SLCHOST_FRC_POS_SAMP				SLCHOST_FRC_NEG_SAMP				SLCHOST_FRC_SDIO20				SLCHOST_FRC_SDIO11				
31	28	27	26	20	19	15	14	10	9	5	4	0																
0x0				0	0x0				0x0				0x0				0x0				0x0				Reset			

- SLCHOST_FRC_SDIO11** 将 bit[4] 设置为 1，以此强制在时钟下降沿驱动 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟下降沿驱动 DAT[3:0] 信号对应 bit。(R/W)
- SLCHOST_FRC_SDIO20** 将 bit[4] 设置为 1，以此强制在时钟上升沿驱动 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟上升沿驱动 DAT[3:0] 信号对应 bit。(R/W)
- SLCHOST_FRC_NEG_SAMP** 将 bit[4] 设置为 1，以此强制在时钟下降沿采样 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟下降沿采样 DAT[3:0] 信号对应 bit。(R/W)
- SLCHOST_FRC_POS_SAMP** 将 bit[4] 设置为 1，以此强制在时钟上升沿采样 CMD 信号。将 bit[3:0] 对应 bit 设置为 1，以此强制在时钟上升沿采样 DAT[3:0] 信号对应 bit。(R/W)
- SLCHOST_HSPEED_CON_EN** 将该位和 [HINF_HIGHSPEED_ENABLE](#) 设置为 1，然后主机将 CCCR 寄存器中的 EHS 置为 1，以此强制在时钟上升沿驱动 CMD 和 DAT 信号。(R/W)

Register 29.41. SLCHOST_SLC0HOST_INT_RAW_REG (0x0050)

[illegible]

SLCHOST_SLCO_TOHOST_BIT n _INT_RAW (n : 0-7) SLCHOST_SLCO_TOHOST_BIT n _INT (n : 0-7)
的原始中断状态。(R/WTC/SS)

SLCHOST_SLC0_RX_UDF_INT_RAW **SLCHOST_SLC0_RX_UDF_INT** 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC0_TX_OVF_INT_RAW **SLCHOST_SLC0_TX_OVF_INT** 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC0_RX_NEW_PACKET_INT_RAW SLCHOST_SLC0_RX_NEW_PACKET_INT 的原始中断状态。(R/WTC/SS)

Register 29.42. SLCHOST_SLC1HOST_INT_RAW_REG (0x0054)

[illegible]

SLCHOST_SLC1_TOHOST_BIT n _INT_RAW (n : 0-7) SLCHOST_SLC1_TOHOST_BIT n _INT (n : 0-7) 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC1_RX_UDF_INT_RAW **SLCHOST_SLC1_RX_UDF_INT** 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC1_TX_OVF_INT_RAW **SLCHOST_SLC1_TX_OVF_INT** 的原始中断状态。(R/WTC/SS)

SLCHOST_SLC1_RX_NEW_PACKET_INT_RAW SLCHOST_SLC1_RX_NEW_PACKET_INT 的原始中断状态。(R/WTC/SS)

Register 29.43. SLCHOST_SLC0HOST_INT_ST_REG (0x0058)

[illegible]

SLCHOST_SLC0_TOHOST_BIT n _INT_ST (n : 0-7) SLCHOST_SLC0_TOHOST_BIT n _INT (n : 0-7) 的屏蔽中断状态。(RO)

SLCHOST_SLCO_RX_UDF_INT_ST SLCHOST_SLCO_RX_UDF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC0_TX_OVF_INT_ST SLCHOST_SLC0_TX_OVF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC0_RX_NEW_PACKET_INT_ST SLCHOST_SLC0_RX_NEW_PACKET_INT 的屏蔽中断状态。(RO)

Register 29.44. SLCHOST_SLC1HOST_INT_ST_REG (0x005C)

[illegible]

SLCHOST_SLC1_TOHOST_BIT n _INT_ST (n : 0-7) SLCHOST_SLC1_TOHOST_BIT n _INT (n : 0-7) 的屏蔽中断状态。(RO)

SLCHOST_SLC1_RX_UDF_INT_ST SLCHOST_SLC1_RX_UDF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC1_TX_OVF_INT_ST SLCHOST_SLC1_TX_OVF_INT 的屏蔽中断状态。(RO)

SLCHOST_SLC1_RX_NEW_PACKET_INT_ST SLCHOST_SLC1_RX_NEW_PACKET_INT 的屏蔽中断状态。(RO)

Register 29.45. SLCHOST CONF W7 REG (0x008C)

SLCHOST_SLCHOST_CONF31								(reserved)								SLCHOST_SLCHOST_CONF29								(reserved)																																							
31								24								23								16								15								8								7								0							
0x0								0x0								0x0								0x0								Reset																															

SLCHOST_SLCHOST_CONF29 SLCINT_SLC_FRHOST_BIT n _INT (n : 0-7) 的中断设置位，这些位不会自动清除。(R/W)

SLCHOST_SLCHOST_CONF31 SLCINT_SLC_FRHOST_BIT n _INT (n : 8-15) 的中断设置位，这些位不会自动清除。(R/W)

Register 29.46. SLCHOST_SLC0HOST_INT_CLR_REG (0x00D4)

31	24	23	22	18	17	16	15	8	7	6	5	4	3	2	1	0	
0x0		0	0x0		0	0	0x0		0	0	0	0	0	0	0	0	Reset
(reserved)		SLCHOST - SLCO_RX_NEW_PACKET_INT - CLR		(reserved)		SLCHOST - SLCO_TX_OVF_INT - CLR SLCHOST - SLCO_RX_UDF_INT - CLR		(reserved)		SLCHOST - SLCO_TOHOST_BIT7_INT - CLR SLCHOST - SLCO_TOHOST_BIT6_INT - CLR SLCHOST - SLCO_TOHOST_BIT5_INT - CLR SLCHOST - SLCO_TOHOST_BIT4_INT - CLR SLCHOST - SLCO_TOHOST_BIT3_INT - CLR SLCHOST - SLCO_TOHOST_BIT2_INT - CLR SLCHOST - SLCO_TOHOST_BIT1_INT - CLR SLCHOST - SLCO_TOHOST_BIT0_INT - CLR							

SLCHOST_SLCO_TOHOST_BIT n _INT_CLR (n : 0-7) 写 1 清除 SLCHOST_SLCO_TOHOST_BIT n _INT (n : 0-7)。(WT)

SLCHOST_SLCO_RX_UDF_INT_CLR 写1清除 SLCHOST_SLCO_RX_UDF_INT。(WT)

SLCHOST_SLCO_TX_OVF_INT_CLR 写1清除 SLCHOST_SLCO_TX_OVF_INT。(WT)

SLCHOST_SLCO_RX_NEW_PACKET_INT_CLR 写 1 清除 SLCHOST_SLCO_RX_NEW_PACKET_INT。
(WT)

Register 29.47. SLCHOST_SLC1HOST_INT_CLR_REG (0x00D8)

[illegible]

SLCHOST_SLC1_TOHOST_BIT n _INT_CLR (n : 0-7) 写 1 清除 SLCHOST_SLC1_TOHOST_BIT n _INT (n : 0-7)。(WT)

SLCHOST_SLC1_RX_UDF_INT_CLR 写1清除 SLCHOST_SLC1_RX_UDF_INT。(WT)

SLCHOST_SLC1_TX_OVF_INT_CLR 写1清除 SLCHOST_SLC1_TX_OVF_INT。(WT)

SLCHOST_SLC1_RX_NEW_PACKET_INT_CLR 写 1 清除 SLCHOST_SLC1_RX_NEW_PACKET_INT。
(WT)

Register 29.48. SLCHOST_SLC0HOST_FUNC1_INT_ENA_REG (0x00DC)

[illegible]

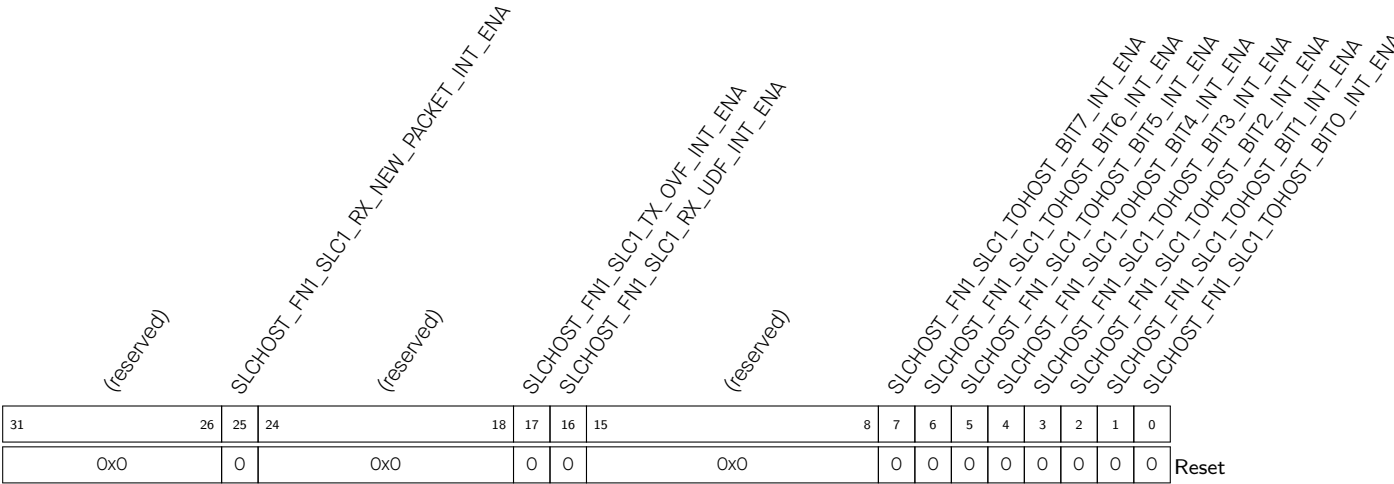
SLCHOST_FN1_SLC0_TOHOST_BIT	n	INT_ENA	(n : 0-7)	写	1	使 能	SL-
CHOST_SLC0_TOHOST_BIT							
n : INT (n : 0-7)。(R/W)							

SLCHOST_FN1_SLC0_RX_UDF_INT_ENA 写1使能SLCHOST_SLC0_RX_UDF_INT。(R/W)

SLCHOST_FN1_SLCO_TX_OVF_INT_ENA 写1使能 SLCHOST_SLCO_TX_OVF_INT。(R/W)

SLCHOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA 写 1 使能 SLCHOST_SLC0_RX_NEW_PACKET_INT。(R/W)

Register 29.49. SLCHOST_SLC1HOST_FUNC1_INT_ENA_REG (0x00E0)



SLCHOST_FN1_SLC1_TOHOST_BIT n _INT_ENA (n : 0-7) 写 1 使能 SLCHOST_SLC1_TOHOST_BIT n _INT (n : 0-7)。(R/W)

SLCHOST_FN1_SLC1_RX_UDF_INT_ENA 写 1 使能 SLCHOST_SLC1_RX_UDF_INT。(R/W)

SLCHOST_FN1_SLC1_TX_OVF_INT_ENA 写 1 使能 SLCHOST_SLC1_TX_OVF_INT。(R/W)

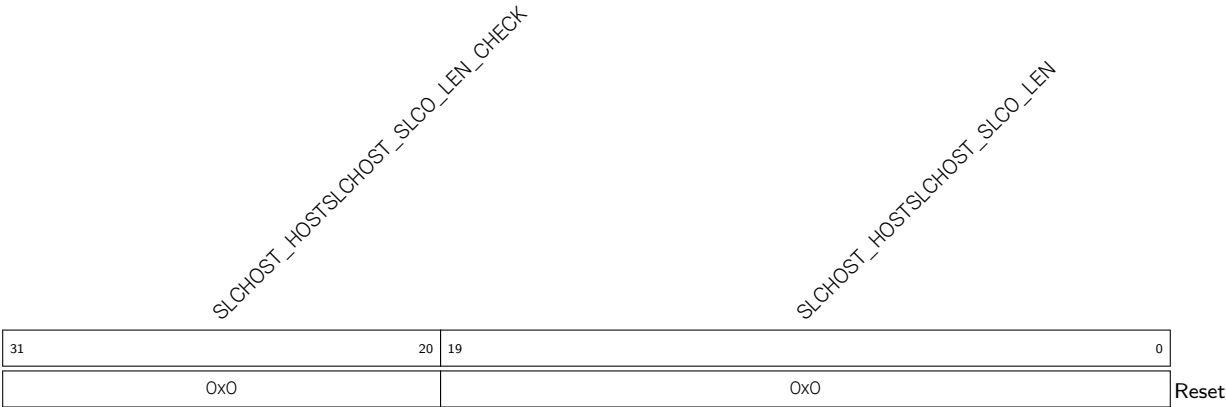
SLCHOST_FN1_SLC1_RX_NEW_PACKET_INT_ENA 写 1 使能 SLCHOST_SLC1_RX_NEW_PACKET_INT。(R/W)

Register 29.50. SLCHOST_SLC0HOST_TOKEN_RDATA_REG (0x0044)



SLCHOST_HOSTSLCHOST_SLC0_TOKEN1 表示可用于通过 SLC0 通道接收数据的 buffer 的累计值。
(RO)

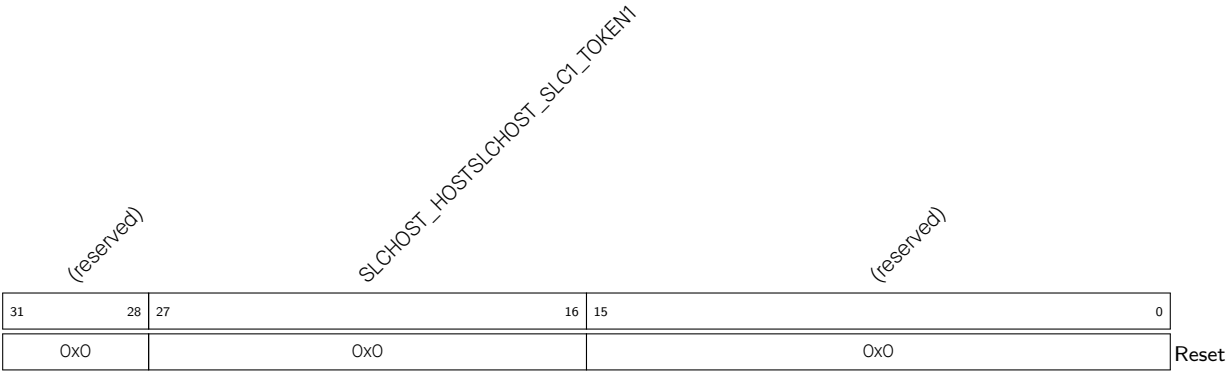
Register 29.51. SLCHOST_PKT_LEN_REG (0x0060)



SLCHOST_HOSTSLCHOST_SLC0_LEN 表示从机总共希望发送的数据量，该字段的值只在被主机读取时更新。(RO)

SLCHOST_HOSTSLCHOST_SLC0_LEN_CHECK 校验 SLCHOST_HOSTSLCHOST_SLC0_LEN 字段的值，该字段的值等于 SLCHOST_HOSTSLCHOST_SLC0_LEN bit[9:0] 加上 bit[19:10]。(RO)

Register 29.52. SLCHOST_SLC1HOST_TOKEN_RDATA_REG (0x00C4)



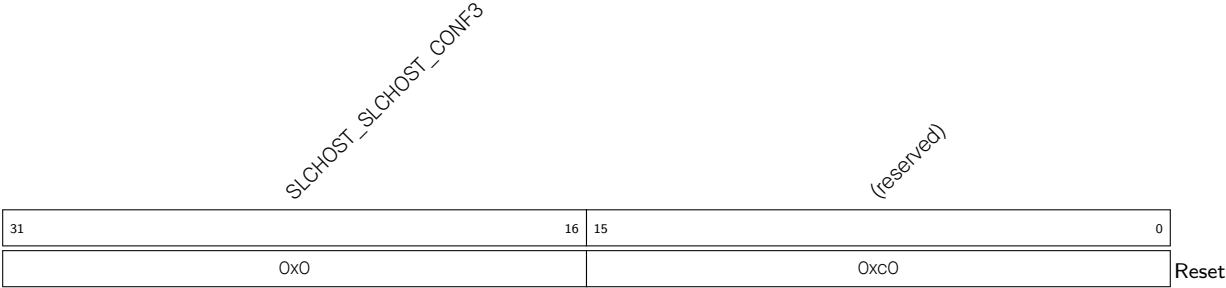
SLCHOST_HOSTSLCHOST_SLC1_TOKEN1 表示可用于通过 SLC1 通道接收数据的 buffer 的累计值。(RO)

Register 29.53. SLCHOST_CONF_Wn_REG(*n*: 0-2) (0x006C+0x4**n*)



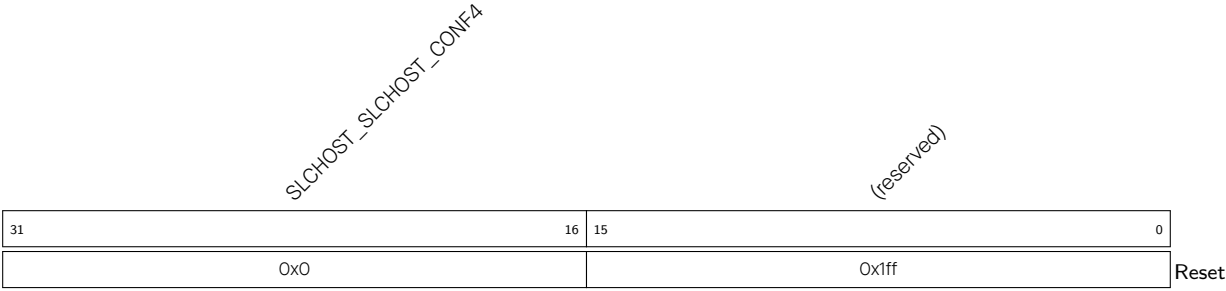
SLCHOST_SLCHOST_CONF*n* 主机与从机之间的信息交互寄存器。(R/W)

Register 29.54. SLCHOST_CONF_W3_REG (0x0078)



SLCHOST_SLCHOST_CONF3 主机与从机之间的信息交互寄存器。(R/W)

Register 29.55. SLCHOST_CONF_W4_REG (0x007C)



SLCHOST_SLCHOST_CONF4 主机与从机之间的信息交互寄存器。(R/W)

Register 29.56. SLCHOST_CONF_W6_REG (0x0088)



SLCHOST_SLCHOST_CONF6 主机与从机之间的信息交互寄存器。(R/W)

Register 29.57. SLCHOST_CONF_Wn_REG(*n*: 8-15) (0x009C+0x4*(*n*-8))



SLCHOST_SLCHOST_CONF*n* 主机与从机之间的信息交互寄存器。(R/W)

第 30 章

LED PWM 控制器 (LEDC)

30.1 概述

LED PWM 控制器用于生成控制 LED 的脉冲宽度调制信号 (PWM)，具有占空比自动渐变等功能。该外设也可生成 PWM 信号用作其他用途。

30.2 主要特性

LED PWM 控制器具有如下特性：

- 6 个独立的 PWM 生成器（即 6 个通道）
- PWM 占空比最大精度为 20 位
- 4 个独立定时器，可实现小数分频
- PWM 输出信号相位可调节
- PWM 占空比微调
- 占空比自动渐变，即 PWM 信号占空比可逐渐增加或减小，无须处理器干预，渐变完成时产生中断
- 每个 PWM 生成器包含 16 个占空比渐变区间，用于生成占空比伽马曲线渐变的信号。每个区间都可以独立配置占空比变化方向（增加或减少）、变化步长（占空比每次增加或减少的量）、变化次数（占空比在一个范围内的渐变次数）以及变化频率
- 低功耗模式 (Light-sleep mode) 下可输出 PWM 信号
- 可以生成 ETM（事件任务矩阵）外设相关的事件，可以接收 ETM 外设相关的任务

由于 4 个定时器具有相同的功能和运行方式，下文将 4 个定时器统称为定时器 x （ x 的范围是 0 到 3）。同样的，6 个 PWM 生成器的功能和运行方式也相同，因此下文将统称为 PWM n （ n 的范围是 0 到 5）。

30.3 结构概览

图 30.3-1 为 LED PWM 控制器的架构。

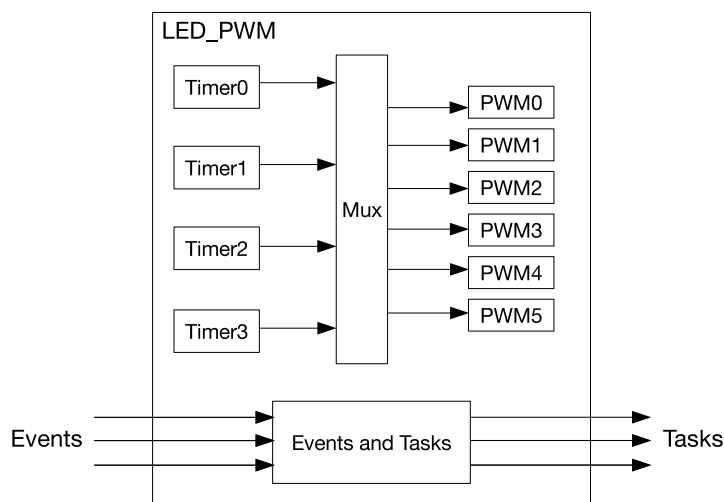


图 30.3-1. LED PWM 控制器架构

4 个定时器每个内部都有一个时基计数器（即基于基准时钟周期计数的计数器），因此每个定时器都可以独立配置（可配置时钟分频器和计数器最大值）。每个 PWM 生成器通过配置 `LEDC_TIMER_SEL_CHn` 可以在 4 个定时器中择一，并以该定时器的计数值 `timerx_cnt` 为基准生成 PWM 信号。

图 30.3-2 为定时器和 PWM 生成器的主要功能块。

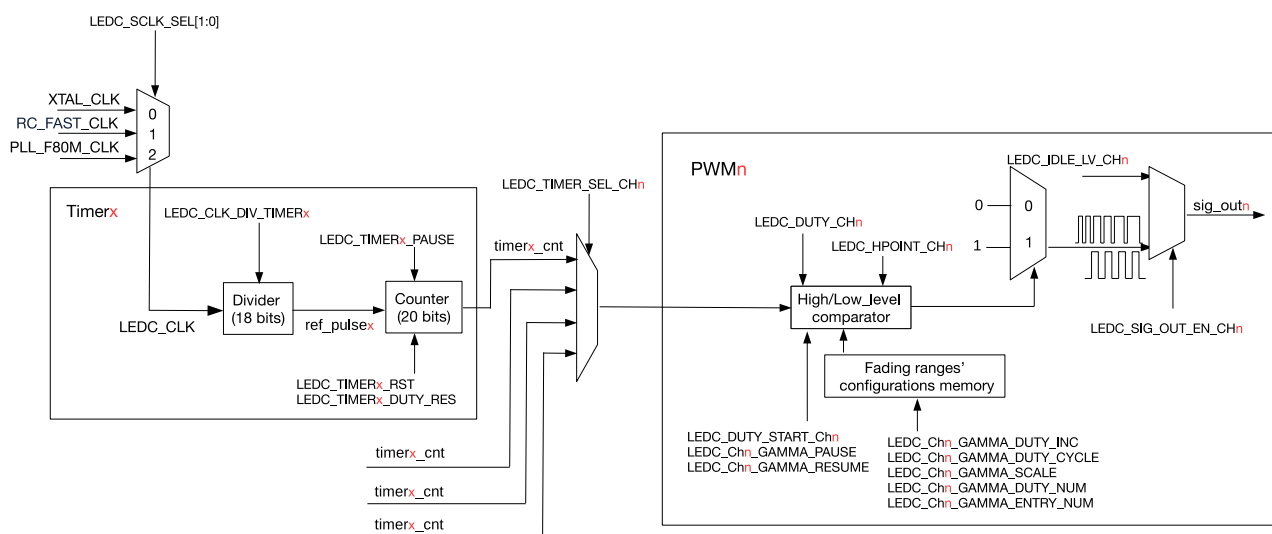


图 30.3-2. 定时器和 PWM 生成器功能框图

30.4 功能描述

30.4.1 定时器

LED PWM 控制器的每个定时器内部都有一个时基计数器。图 30.3-2 中时基计数器使用的时钟信号称为 `ref_pulsex`。所有定时器使用同一个时钟源信号 `LEDC_CLK`，该时钟源信号经分频器分频后产生 `ref_pulsex` 供计数器使用。

30.4.1.1 时钟源

软件可配置的 LED PWM 寄存器由 APB_CLK 时钟驱动。要使用 LED PWM 控制器，需使能 LED PWM 的 APB_CLK 时钟信号。APB_CLK 时钟信号可通过置位 PCR_LEDC_CONF_REG 寄存器的 PCR_LEDC_CLK_EN 字段使能。LED PWM 的 LEDC_CLK 时钟信号可通过置位 PCR_LEDC_SCLK_CONF_REG 寄存器的 PCR_LEDC_SCLK_EN 字段使能。LED PWM 可通过软件置位 PCR_LEDC_CONF_REG 寄存器的 PCR_LEDC_RST_EN 字段复位。

LED PWM 控制器的定时器从下述时钟信号中选择一个作为其时钟源信号：XTAL_CLK、RC_FAST_CLK 和 PLL_F80M_CLK。为 LEDC_CLK 选择时钟源信号的配置如下：

- XTAL_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 0
- RC_FAST_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 1
- PLL_F80M_CLK：将 PCR_LEDC_SCLK_SEL[1:0] 置 2

之后，LEDC_CLK 信号会进入时钟分频器。

更多信息，请参阅章节 6 复位和时钟。

30.4.1.2 时钟分频器配置

LEDC_CLK 信号传输到时钟分频器，产生 ref_pulse_x 信号供计数器使用。ref_pulse_x 的频率等于 LEDC_CLK 的频率经分频系数 LEDC_CLK_DIV 分频后的结果（见图 30.3-2）。

由于时钟分频器为小数分频，因此分频系数 LEDC_CLK_DIV 可为非整数，其可根据下列等式配置：

$$\text{LEDC\_CLK\_DIV} = A + \frac{B}{256} \quad (30.1)$$

- 整数部分 A 为 LEDC_CLK_DIV_TIMER_x 字段的高 10 位（即 LEDC_TIMER_x_CONF_REG[22:13]）
- 小数部分 B 为 LEDC_CLK_DIV_TIMER_x 字段的低 8 位（即 LEDC_TIMER_x_CONF_REG[12:5]）

小数部分 B 为 0 时，LEDC_CLK_DIV 的值为整数（整数分频）。也就是说，每 A 个 LEDC_CLK 时钟周期产生一个 ref_pulse_x 时钟脉冲。

小数部分 B 不为 0 时，LEDC_CLK_DIV 的值为非整数。时钟分频器按照 A 个 LEDC_CLK 时钟周期和 $(A+1)$ 个 LEDC_CLK 时钟周期轮流产生 ref_pulse_x 时钟脉冲，实现非整数分频。这样一来，ref_pulse_x 时钟脉冲的平均频率便会是理想值（非整数分频的频率）。每 256 个 ref_pulse_x 时钟脉冲中：

- 有 B 个 ref_pulse_x 时钟脉冲以 $(A+1)$ 个 LEDC_CLK 时钟周期分频
- 有 $(256-B)$ 个 ref_pulse_x 时钟脉冲以 A 个 LEDC_CLK 时钟周期分频
- 以 $(A+1)$ 个 LEDC_CLK 时钟周期分频的 ref_pulse_x 时钟脉冲均匀分布在以 A 个 LEDC_CLK 时钟周期分频的 ref_pulse_x 时钟脉冲中

图 30.4-1 展示了分频系数 LEDC_CLK_DIV 非整数时，LEDC_CLK 时钟脉冲和 ref_pulse_x 时钟脉冲的关系。

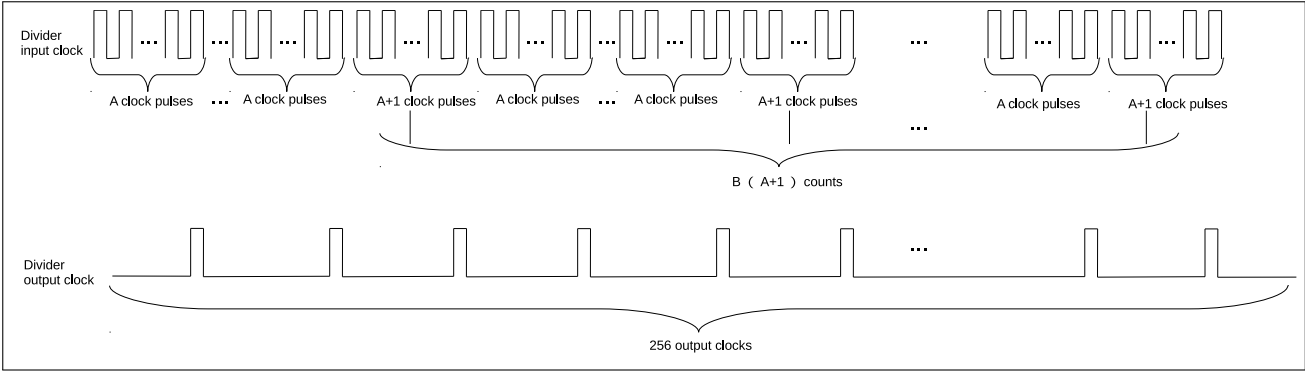


图 30.4-1. LEDC_CLK_DIV 为非整数时的分频

在运行时改变定时器时钟的分频系数，需先配置 `LEDC_CLK_DIV_TIMERx` 字段，然后置位 `LEDC_TIMERx_PARA_UP` 字段应用新配置。新配置会在计数器下次溢出时生效。`LEDC_TIMERx_PARA_UP` 字段由硬件自动清除。

30.4.1.3 20 位计数器

每个定时器有一个以 `ref_pulsex` 为基准时钟的 20 位时基计数器（见图 30.3-2）。`LEDC_TIMERx_DUTY_RES` 字段用于配置 20 位计数器的实际使用位宽。因此，PWM 信号的最大精度为 20 位。计数器最大可计数至 $2^{\text{LEDC\_TIMERx\_DUTY\_RES}} - 1$ ，然后溢出并重新从 0 开始计数。软件可以读取、复位、暂停计数器。计数器和 PWM 信号精度的关系如图 30.4-2 所示。

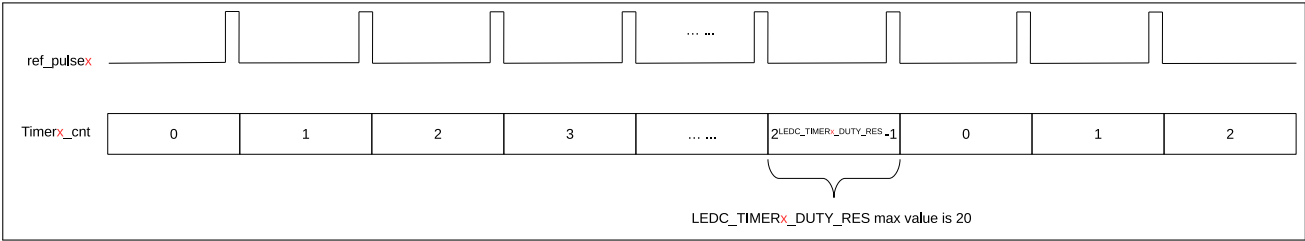


图 30.4-2. 计数器和 PWM 信号精度关系

计数器可在每次溢出时触发 `LEDC_TIMERx_OVF_INT` 中断（这个中断为硬件自动产生，不需要配置）。计数器也可配置为在溢出 `LEDC_OVF_NUM_CHn + 1` 次时触发 `LEDC_OVF_CNT_CHn_INT` 中断，该中断配置步骤如下：

1. 配置 `LEDC_TIMER_SEL_CHn` 为 PWM 生成器选择该定时器
2. 置位 `LEDC_OVF_CNT_EN_CHn` 使能溢出次数计数器
3. 把 `LEDC_OVF_NUM_CHn` 的值设为触发中断的计数器溢出次数减 1
4. 置位 `LEDC_OVF_CNT_CHn_INT_ENA` 使能溢出中断
5. 配置 `LEDC_TIMERx_DUTY_RES` 为所选的定时器 `Timerx` 指定计数器的位宽，等待 `LEDC_OVF_CNT_CHn_INT` 中断产生

在运行时改变计数器的最大值，需先配置 `LEDC_TIMERx_DUTY_RES` 字段，然后置位 `LEDC_TIMERx_PARA_UP` 字段。新的配置在计数器下一次溢出时生效。如果重新配置 `LEDC_OVF_CNT_EN_CHn` 字段，需置位 `LEDC_PARA_UP_CHn` 应用新配置。总之，更改配置时需置位 `LEDC_TIMERx_PARA_UP` 或 `LEDC_PARA_UP_CHn` 应用新配置。

LEDC_TIMER x _PARA_UP 和 LEDC_PARA_UP_CH n 字段由硬件自动清除。

如图 30.3-2 所示，PWM 生成器输出信号 sig_out n 的频率取决于定时器时钟源 LEDC_CLK 的频率、时钟分频系数 LEDC_CLK_DIV 以及占空比精度（计数器位宽）LEDC_TIMER x _DUTY_RES：

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMER}_x\text{\_DUTY\_RES}}} \quad (30.2)$$

上述公式变形后，可得到以下公式计算预期的占空比精度：

$$\text{LEDC\_TIMER}_x\text{\_DUTY\_RES} = \log_2 \left(\frac{f_{\text{LEDC\_CLK}}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}} \right) \quad (30.3)$$

表 30.4-1 列出了常用配置频率及其对应精度。

表 30.4-1. 常用配置频率及精度

LEDC_CLK	PWM 频率	最高精度（位） ¹	最低精度（位） ²
REF_80M_CLK (80 MHz)	1 kHz	16	6
REF_80M_CLK (80 MHz)	5 kHz	13	3
REF_80M_CLK (80 MHz)	10 kHz	12	2
XTAL_40M_CLK (40 MHz)	1 kHz	15	5
XTAL_40M_CLK (40 MHz)	4 kHz	13	3
FOSC_20M_CLK (20 MHz)	1 kHz	14	4
FOSC_20M_CLK (20 MHz)	2 kHz	13	3

¹ 最高精度指时钟分频系数 LEDC_CLK_DIV 为 1 时的精度。如果经公式计算出的最高精度超过了计数器位宽 20 位，则最高精度为 20。

² 最低精度指时钟分频系数 LEDC_CLK_DIV 为 $1023 + \frac{255}{256}$ 时的精度。如果经公式计算出的最低精度小于 0，则最低精度为 1。

30.4.2 PWM 生成器

要生成 PWM 信号，PWM 生成器 (PWM n) 需选择一个定时器 (Timer x)。每个 PWM 生成器均可通过置位 LEDC_TIMER_SEL_CH n 单独配置，在 4 个定时器中选择一个用于生成 PWM 信号。

如图 30.3-2 所示，每个 PWM 生成器主要包括一个高低电平比较器和两个选择器。PWM 生成器将定时器的 20 位计数值 (Timer x _cnt) 与高低电平比较器的值 Hpoint n 和 Lpoint n 比较。如果定时器的计数值等于 Hpoint n 或 Lpoint n ，PWM 信号可以输出高低电平：

- 如果 Timer x _cnt == Hpoint n ，则 sig_out n 为 1。
- 如果 Timer x _cnt == Lpoint n ，则 sig_out n 为 0。

图 30.4-3 展示了如何使用 Hpoint n 和 Lpoint n 生成占空比固定的 PWM 信号。

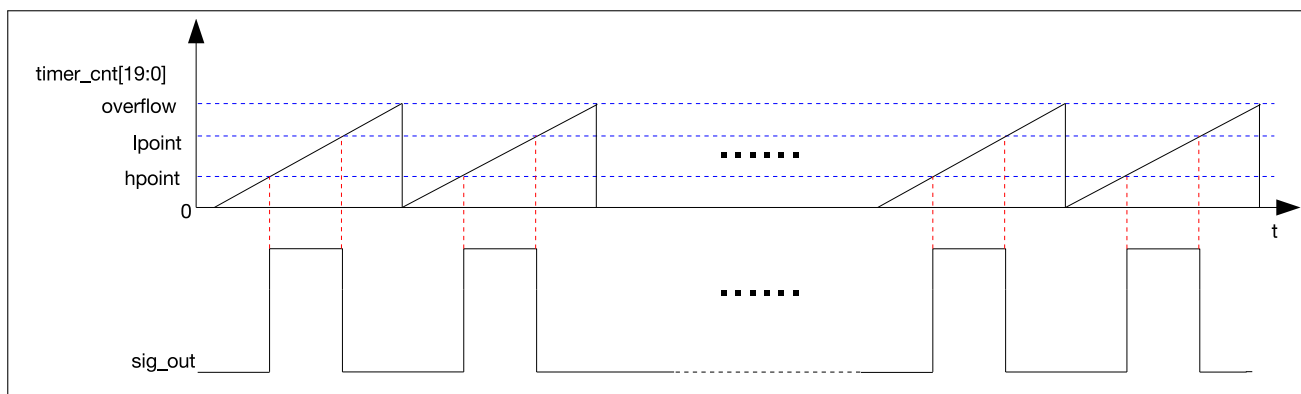


图 30.4-3. LED PWM 输出信号图

每当所选定时器的计数器溢出时，PWM 生成器 (PWM n) 的 Hpoint n 值更新为 LEDC_HPOINT_CH n 。Lpoint n 的值同样在计数器每次溢出时更新，为 LEDC_DUTY_CH n [24:4] 和 LEDC_HPOINT_CH n 的和。通过配置 LEDC_DUTY_CH n [24:4] 和 LEDC_HPOINT_CH n 两个字段可对 Hpoint n 和 Lpoint n 进行设置，进而可设置 PWM 输出信号的相对相位和占空比。

置位 LEDC_SIG_OUT_EN_CH n ，开启 PWM 信号 (sig_out n) 输出；清除 LEDC_SIG_OUT_EN_CH n ，关闭 PWM 信号输出，输出信号 (sig_out n) 输出恒定电平，电平值为 LEDC_IDLE_LV_CH n 。

LEDC_DUTY_CH n [3:0] 通过周期性改变 PWM 输出信号 sig_out n 的占空比实现微调。如 LEDC_DUTY_CH n [3:0] 不为 0，那么 sig_out n 每 16 个周期中，有 LEDC_DUTY_CH n [3:0] 个周期的 PWM 脉冲占空比要比 (16 - LEDC_DUTY_CH n [3:0]) 个周期的脉冲占空比多一个定时器的计数周期。比如，如果 LEDC_DUTY_CH n [24:4] 设为 10，LEDC_DUTY_CH n [3:0] 设为 5，则 16 个周期中，有 5 个周期的 PWM 脉冲占空比为 11，剩余 11 个周期的 PWM 脉冲占空比为 10。16 个周期的平均占空比为 10.3125。

如果重新配置 LEDC_TIMER_SEL_CH n 、LEDC_HPOINT_CH n 、LEDC_DUTY_CH n [24:4] 和 LEDC_SIG_OUT_EN_CH n 字段，需置位 LEDC_PARA_UP_CH n 应用新配置。新配置在计数器下次溢出时生效。LEDC_PARA_UP_CH n 字段由硬件自动清除。

30.4.3 占空比渐变

PWM 生成器可以渐变 PWM 输出信号的占空比（即由一种占空比逐渐变为为另一种占空比）。每个 PWM 生成器最多可以包含 16 个占空比渐变区间，每个区间可以独立配置占空比渐变方向（增加或减少）、渐变步长、渐变次数和渐变频率。如果开启占空比渐变功能，每个区间的 Lpoint n 的值会根据该区间的渐变配置进行变化。

请注意在占空比渐变过程中，如果 PWM 输出信号 (PWM n) 的占空比达到 100%，则应将 LEDC_HPOINT_CH n 设置为 0，以确保 Lpoint n 正确。

30.4.3.1 线性占空比渐变

通过配置第一个占空比渐变区间的渐变方向、渐变步长、渐变次数和渐变频率，可以产生线性占空比渐变 PWM 信号。

配置步骤请见 30.7。

当上述配置步骤完成后，每当计数器溢出 LEDC_CH n _GAMMA_DUTY_CYCLE 次后，PWM 生成器将 PWM 信号的占空比渐变一次。PWM 信号每次渐变时，其 Lpoint n 会增加/减少（由 LEDC_CH n _GAMMA_DUTY_INC 配置）LEDC_CH n _GAMMA_SCALE，相应地占空比会增加/减少（由 LEDC_CH n _GAMMA_DUTY_INC 配置）。

$$\frac{\text{LEDC\_CH}_n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}_x\text{\_DUTY\_RES}} \quad (30.4)$$

PWM 信号的占空比会渐变 LEDC_CH n _GAMMA_DUTY_NUM 次。完成该次数的渐变后，PWM 信号的占空比停止变化并保持以该占空比输出。由于每次渐变时占空比增加/减少的值都相同，所以产生的 PWM 信号为线性占空比渐变信号。

图 30.4-4 展示了一个 PWM 信号占空比线性渐变的例子。

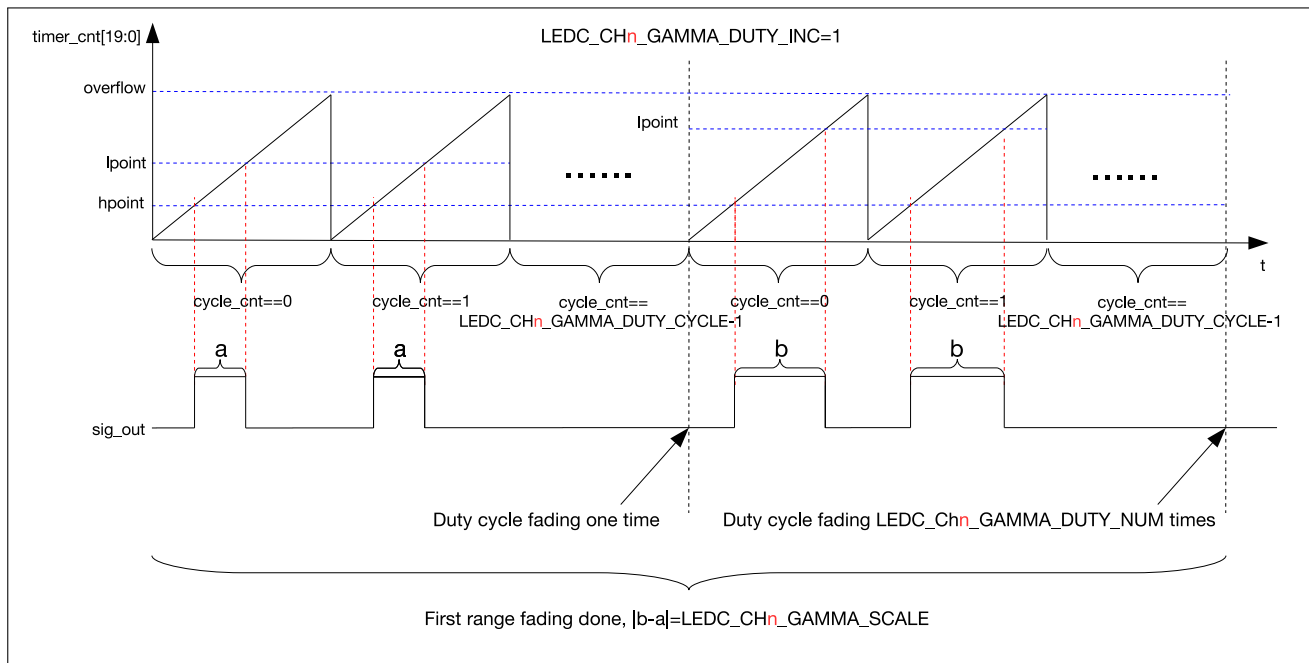


图 30.4-4. 输出信号占空比线性渐变图

30.4.3.2 伽马曲线渐变

通过配置多个占空比渐变区间的占空比渐变方向、渐变步长、渐变次数和渐变频率，PWM 生成器可以产生伽马曲线渐变的 PWM 信号。

配置步骤请见 30.7。

经过上述所有的配置步骤，PWM 生成器会产生包含 LEDC_CH n _GAMMA_ENTRY_NUM 个占空比渐变区间的 PWM 信号。首先，PWM 信号会根据序号为 0 的占空比渐变区间的配置进行占空比渐变，渐变完成后，按照序号为 1 的占空比渐变区间的配置进行占空比渐变，直到序号为 (LEDC_CH n _GAMMA_ENTRY_NUM - 1) 的占空比渐变区间（即最后一个区间）完成渐变。在每个区间中，PWM 信号都是独立变化的。对于每个区间，PWM 信号会在计数器每溢出 LEDC_CH n _GAMMA_DUTY_CYCLE 次后，进行一次占空比渐变，对于每次渐变，其 Lpoint n 会增加/减少（由 LEDC_CH n _GAMMA_DUTY_INC 配置）LEDC_CH n _GAMMA_SCALE，相对的，占空比会增加/减少（由 LEDC_CH n _GAMMA_DUTY_INC 配置）

$$\frac{\text{LEDC\_CH}_n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}_x\text{\_DUTY\_RES}} \quad (30.5)$$

在该区间内，占空比会渐变 LEDC_CH n _GAMMA_DUTY_NUM 次，当指定次数的占空比渐变完成后，视为当前区间的占空比渐变完成。

当 `LEDC_CH $n$ _GAMMA_ENTRY_NUM` 个区间的占空比渐变都完成后，PWM 信号会停止占空比渐变，其占空比会保持最后一次渐变的占空比。由于每个占空比渐变区间的渐变过程都是独立不同的线性渐变，因此多个区间的线性占空比渐变就可以拟合成一个特定的伽马曲线占空比渐变。

图 30.4-5 展示了一个伽马曲线渐变 PWM 信号。

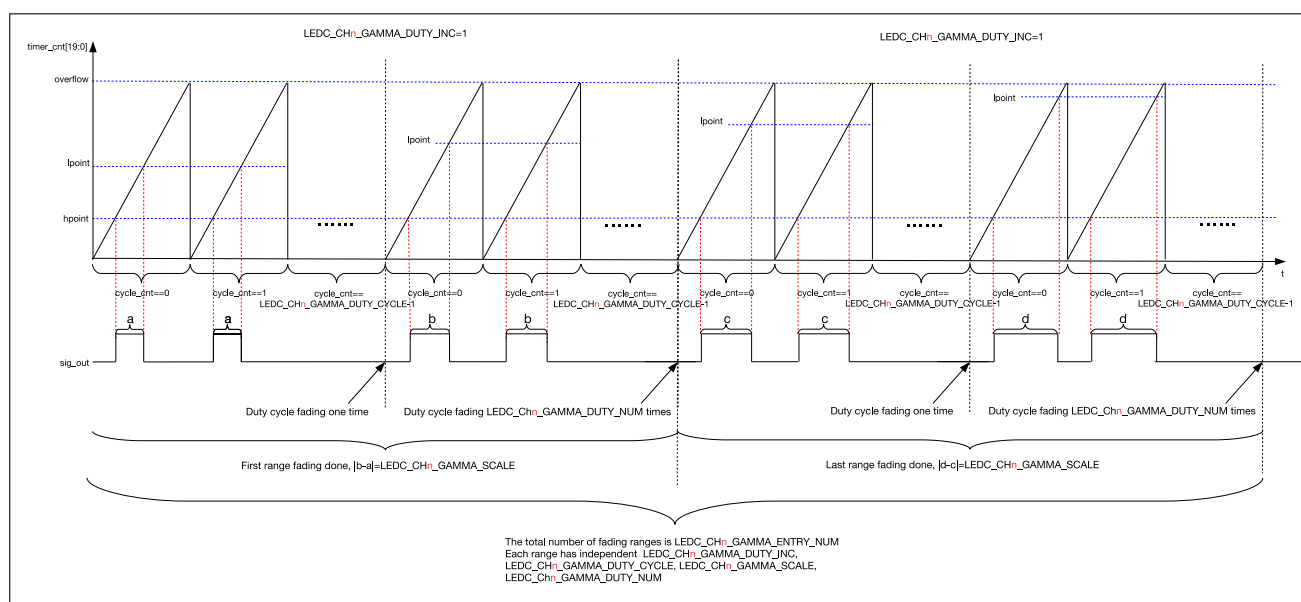


图 30.4-5. 输出信号占空比伽马曲线渐变图

30.4.3.3 占空比渐变暂停和恢复

向 `LEDC_CH $n$ _GAMMA_CONF_REG` 寄存器的 `LEDC_CH $n$ _GAMMA_PAUSE` 字段写 1 可以暂停已经开始的占空比渐变过程。一旦将 `LEDC_CH $n$ _GAMMA_PAUSE` 字段置 1，PWM 信号将会保持最近一次渐变后的占空比。

向 `LEDC_CH $n$ _GAMMA_CONF_REG` 寄存器的 `LEDC_CH $n$ _GAMMA_RESUME` 字段写 1 可以恢复被暂停的占空比渐变过程。一旦将 `LEDC_CH $n$ _GAMMA_RESUME` 字段置 1，PWM 信号将会按照暂停时所在的占空比渐变区间的配置，从暂停时的占空比继续进行占空比渐变，直到所有有效的区间都完成指定的占空比渐变 (`LEDC_CH $n$ _GAMMA_RESUME` 字段置 1 后，硬件会自动清除 `LEDC_CH $n$ _GAMMA_PAUSE` 字段)。

30.5 事件任务矩阵功能

在 ESP32-C5 中，LEDC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 LEDC 的 ETM 任务，或者通过 LEDC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与 LEDC 相关的 ETM 任务和 ETM 事件。

ETM 外设相关的事件和任务的使能由寄存器 `LEDC_EVT_TASK_ENO_REG`、`LEDC_EVT_TASK_EN1_REG` 和 `LEDC_EVT_TASK_EN2_REG` 的相关字段所配置。事件和任务与寄存器 `LEDC_EVT_TASK_ENO_REG`、`LEDC_EVT_TASK_EN1_REG`、`LEDC_EVT_TASK_EN2_REG` 的每个字段的对应关系参照章节 30.9。

LEDC 可接收的 ETM 任务有：

- `LEDC_TASK_DUTY_SCALE_UPDATE_CH $n$` ：如果 `LEDC_TASK_DUTY_SCALE_UPDATE_CH $n$ _EN` 字段使能，当接收到 `LEDC_TASK_DUTY_SCALE_UPDATE_CH $n$` 任务后，相应的 LEDC 通道 (PWM n) 将会根据最新配置的 `LEDC_CH $n$ _GAMMA_SCALE` 的值，重新开始生成占空比渐变 PWM 信号。

- LEDC_TASK_TIMER x _RES_UPDATE: 如果 LEDC_TASK_TIMER x _RES_UPDATE_EN 字段使能, 当接收到 LEDC_TASK_TIMER x _RES_UPDATE 任务后, 相应的定时器 (Timer x) 将会根据最新配置的 LEDC_TIMER x _DUTY_RES 字段的值, 在下次计数器溢出时, 更新计数器的最大计数值为该值。
- LEDC_TASK_TIMER x _CAP: 如果 LEDC_TASK_TIMER x _CAP_EN 字段使能, 当接收到 LEDC_TASK_TIMER x _CAP 任务时, 相应的定时器 (Timer x) 将会抓取当前计数器的计数值。该计数值将会存储在寄存器 LEDC_TIMER x _CNT_CAP_REG 的 LEDC_TIMER x _CNT_CAP 字段。
- LEDC_TASK_SIG_OUT_DIS_CH n : 如果 LEDC_TASK_SIG_OUT_DIS_CH n _EN 字段使能, 当接收到 LEDC_TASK_SIG_OUT_DIS_CH n 任务时, 相应的 LEDC 通道 (PWM n) 的 PWM 信号将停止输出, 该通道的输出信号 (sig_out n) 将会变为 LEDC_IDLE_LV_CH n 字段所配置电平的固定电平信号, 如图 30.3-2 所示。
- LEDC_TASK_OVF_CNT_RST_CH n : 如果 LEDC_TASK_OVF_CNT_RST_CH n _EN 字段使能, 当接收到 LEDC_TASK_OVF_CNT_RST_CH n 任务时, 相应的 LEDC 通道 (PWM n) 定时器的计数器溢出次数的计数器值将会被复位为 0。
- LEDC_TASK_TIMER x _RST: 如果 LEDC_TASK_TIMER x _RST_EN 字段使能, 当接收到 LEDC_TASK_TIMER x _RST 任务时, 相应的定时器 (Timer x) 的计数器值将会被复位为 0。
- LEDC_TASK_TIMER x _RESUME 和 LEDC_TASK_TIMER x _PAUSE: 如果 LEDC_TASK_TIMER x _PAUSE_RESUME_EN 字段使能, 当接收到 LEDC_TASK_TIMER x _RESUME 和 LEDC_TASK_TIMER x _PAUSE 任务时, 相应的定时器 (Timer x) 将会被交替暂停或恢复。即, 当该任务被接收一次时, 定时器 (Timer x) 将会暂停, 当该任务再被接收一次时, 定时器 (Timer x) 将会恢复。
- LEDC_TASK_GAMMA_RESTART_CH n : 如果 LEDC_TASK_GAMMA_RESTART_CH n _EN 字段使能, 当接收到 LEDC_TASK_GAMMA_RESTART_CH n 任务时, 相应的 LEDC 通道 (PWM n) 将会重新开始产生占空比渐变 PWM 信号。
- LEDC_TASK_GAMMA_PAUSE_CH n : 如果 LEDC_TASK_GAMMA_PAUSE_CH n _EN 字段使能, 当接收到 LEDC_TASK_GAMMA_PAUSE_CH n 任务时, 相应的 LEDC 通道 (PWM n) 将会在计数器下次溢出时停止占空比的渐变。即, 当该任务被接收后, 相应的 LEDC 通道 (PWM n) 的 PWM 信号的占空比将不会再进行渐变。
- LEDC_TASK_GAMMA_RESUME_CH n : 如果 LEDC_TASK_GAMMA_RESUME_CH n _EN 字段使能, 当接收到 LEDC_TASK_GAMMA_RESUME_CH n 任务时, 相应的 LEDC 通道 (PWM n) 将会在计数器下次溢出时恢复占空比的渐变。即, 当该任务被接收后, 相应的 LEDC 通道 (PWM n) 的 PWM 信号将会按照暂停时的占空比渐变配置继续进行占空比的渐变。

LEDC 可产生的 ETM 事件有:

- LEDC_EVT_DUTY_CHNG_END_CH n : 如果 LEDC_EVT_DUTY_CHNG_END_CH n _EN 字段使能, 当相应的 LEDC 通道 (PWM n) 占空比渐变完成后, 将会产生 LEDC_EVT_DUTY_CHNG_END_CH n 事件。
- LEDC_EVT_OVF_CNT_PLS_CH n : 如果 LEDC_EVT_OVF_CNT_PLS_CH n _EN 字段使能, 当相应的 LEDC 通道 (PWM n) 的定时器的计数器溢出 LEDC_OVF_NUM_CH n + 1 次后, 将会产生 LEDC_EVT_OVF_CNT_PLS_CH n 事件。
- LEDC_EVT_TIME_OVF_TIMER x : 如果 LEDC_EVT_TIME_OVF_TIMER x _EN 字段使能, 当相应的定时器 (Timer x) 的计数器溢出时, 将会产生 LEDC_EVT_TIME_OVF_TIMER x 事件。
- LEDC_EVT_TIMER x _CMP: 如果 LEDC_EVT_TIMER x _CMP_EN 字段使能, 当相应的定时器 (Timer x) 的计数器值等于寄存器 LEDC_TIMER x _CMP_REG 的 LEDC_TIMER x _CMP 字段的值时, 将会产生 LEDC_EVT_TIMER x _CMP 事件。

在具体应用中，LEDC 的 ETM 事件可以用来触发 LEDC 的 ETM 任务，例如，LEDC_EVT_DUTY_CHNG_END_CH n 事件可以触发 LEDC_TASK_GAMMA_RESTART_CH n 任务，从而在一次渐变完成后直接开始下一次渐变。

30.6 中断

ESP32-C5 的 LEDC 模块会产生 LEDC_INT 中断信号，该信号会被发送到 [中断矩阵](#)。

下列内部中断源可以产生 LEDC_INT 中断信号：

- LEDC_OVF_CNT_CH n _INT：定时器计数器溢出 LEDC_OVF_NUM_CH n + 1 次且寄存器 LEDC_OVF_CNT_EN_CH n 置 1 时触发中断。
- LEDC_DUTY_CHNG_END_CH n _INT：PWM n 渐变完成后触发中断。
- LEDC_TIMER x _OVF_INT：定时器达到最大计数值时触发中断。

只有当中断使能寄存器 XXX_ENA 置位为 1 时，才能触发上述中断源。置位 XXX_CLR 可清除中断（XXX 为中断源）。

30.7 配置流程

LED PWM 控制器生成 PWM 信号的配置流程为：

1. 参照 [30.4.1](#) 对定时器 Timer x 进行配置。配置完成后，时钟源、分频器与计数器就配置完成了。
2. 参照 [30.4.2](#) 对 PWM n 生成器进行配置。配置完成后，PWM n 从四个定时器中选择其中一个，并设置 PWM n 输出的相位和占空比。
3. 根据不同的占空比渐变类型选择相应的配置过程：

(a) 线性占空比渐变：

- i. 配置寄存器字段 LEDC_DUTY_CH n 。该字段用来设置 Lpoint n 的初始值。
- ii. 置位寄存器字段 LEDC_DUTY_START_CH n 。当该字段置位/清除，占空比渐变会使能/关闭。
- iii. 将线性占空比渐变配置写入寄存器（LEDC_CH n _MEM 起始地址 + 占空比区间值 * 4），LEDC_CH n _MEM 起始地址包含在表 [30.8-1](#) 中。

请注意以下的字段名称只是为了方便叙述，寄存器中并不存在这些字段。

- A. 第 0 位（为方便叙述，以下称为 LEDC_CH n _GAMMA_DUTY_INC）用于配置方向。当该字段置位或清零时，当前所配置的占空比渐变区间的 Lpoint n 会增加/减少。
- B. 第 1 ~ 10 位（为方便叙述，以下称为 LEDC_CH n _GAMMA_DUTY_CYCLE）用来设置当前所配置的占空比渐变区间的 Lpoint n 每增加/减少一次，计数器溢出的次数。即当计数器溢出次数为配置次数，Lpoint n 将会增加/减少一次。
- C. 第 11 ~ 20 位（为方便叙述，以下称为 LEDC_CH n _GAMMA_SCALE）用来设置当前所配置的占空比渐变区间的 Lpoint n 每次增加/减少的量。
- D. 第 21 ~ 30 位（为方便叙述，以下称为 LEDC_CH n _GAMMA_DUTY_NUM）用来设置当前所配置的占空比渐变区间的渐变次数。

- E. 占空比范围值 (从 0 到 15) 用于指定上述配置的属性属于哪个占空比渐变区间。由于线性占空比渐变只需要配置第一个占空比渐变区间即可, 所以占空比范围值应配置为 0。
- iv. 将寄存器 `LEDC_CH $n$ _GAMMA_CONF_REG` 的 `LEDC_CH $n$ _GAMMA_ENTRY_NUM` 字段配置为 1。该字段用来指定一次渐变过程中有效的占空比渐变区间的个数 (当指定数目的区间都渐变完成后, PWM 信号会停止占空比渐变, PWM 生成器会触发 `LEDC_DUTY_CHNG_END_CH $n$ _INT` 中断)。由于线性占空比渐变只包含一个渐变区间 (第一个占空比渐变区间), 所以 `LEDC_CH $n$ _GAMMA_ENTRY_NUM` 字段配置为 1。
- v. 置位寄存器字段 `LEDC_PARA_UP_CH $n$` 应用上述步骤的配置。当该字段被置位后, 上述的占空比渐变配置会在下一次计数器溢出时生效, PWM 生成器会输出指定配置的线性占空比渐变 PWM 信号 (`LEDC_PARA_UP_CH $n$` 字段会被硬件自动清除)。

(b) 伽马曲线渐变:

- i. 与章节 30.4.3.1 的步骤 1 相同。
- ii. 与章节 30.4.3.1 的步骤 2 相同。
- iii. 配置多个占空比渐变区间: 将伽马曲线渐变配置写入寄存器 (`LEDC_CH $n$ _MEM` 起始地址 + 占空比区间值 * 4), `LEDC_CH $n$ _MEM` 起始地址包含在表 30.8-1 中。
 - A. 第 0 位 (为方便叙述, 以下称为 `LEDC_CH $n$ _GAMMA_DUTY_INC`) 用于配置方向。当该字段置位或清零时, 当前所配置的占空比渐变区间的 `Lpoint $n$` 会增加/减少。
 - B. 第 1 ~ 10 位 (为方便叙述, 以下称为 `LEDC_CH $n$ _GAMMA_DUTY_CYCLE`) 用来设置当前所配置的占空比渐变区间的 `Lpoint $n$` 每增加/减少一次, 计数器溢出的次数。即当计数器溢出次数为配置次数, `Lpoint $n$` 将会增加/减少一次。
 - C. 第 11 ~ 20 位 (为方便叙述, 以下称为 `LEDC_CH $n$ _GAMMA_SCALE`) 用来设置当前所配置的占空比渐变区间的 `Lpoint $n$` 每次增加/减少的量。
 - D. 第 21 ~ 30 位 (为方便叙述, 以下称为 `LEDC_CH $n$ _GAMMA_DUTY_NUM`) 用来设置当前所配置的占空比渐变区间的渐变次数。
 - E. 占空比范围值 (0 ~ 15) 用于指定上述配置的属性属于哪个占空比渐变区间。对于伽马曲线渐变, 所配置的占空比渐变区间的序号必须从 0 开始, 每配置完一个占空比渐变区间, 下一个要配置的占空比渐变区间的序号需要加 1。
 - F. 上述步骤配置完毕后, 一个占空比渐变区间的配置视为完成, 然后可以重复上述的步骤配置其他的占空比渐变区间的参数。每个占空比渐变区间的配置都是独立的, 拥有有效配置的区间个数 (0 ~ 15) 可以是任意的。
- iv. 当所需的占空比渐变区间都配置完毕后, 将 `LEDC_CH $n$ _GAMMA_CONF_REG` 寄存器的 `LEDC_CH $n$ _GAMMA_ENTRY_NUM` 字段配置为步骤 3 中配置过的占空比渐变区间的总个数。
- v. 置位寄存器的 `LEDC_PARA_UP_CH $n$` 字段, 应用以上步骤的配置。当该字段被置位后, 上述步骤的占空比渐变配置会在下一次计数器溢出时生效, PWM 生成器会按照配置输出伽马曲线渐变的 PWM 信号 (`LEDC_PARA_UP_CH $n$` 字段会自动被硬件清除)。

按照上述步骤配置完成后, LED PWM 控制器会生成所需的 PWM 信号。

占空比渐变随时都可以暂停或恢复, 详情请参考 30.4.3.3。如果需要在 PWM 信号停止渐变之前 (即 `LEDC_DUTY_CHNG_END_CH $n$ _INT` 尚未触发) 更改占空比渐变配置, 则需要在更改配置之前暂停占空比渐变。但是, 如果需要在 PWM 信号停止渐变后 (即 `LEDC_DUTY_CHNG_END_CH $n$ _INT` 已被触发) 更改占空比渐变配置, 则可以直接进行更改。

软件随时都可以启用或禁用 PWM 信号输出，也可对输出信号的电平进行更改。详情请参考 30.4.2。

30.8 存储空间

每个 LEDC 通道都有一块存储空间用于存放占空比渐变的配置数据。每个存储空间最多可储存 16 个区间的配置数据。本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

表 30.8-1. LEDC 存储空间

名称	描述	大小	起始地址	终止地址	访问权限
LEDC_CHn_MEM	PWMn 内存	64 B	0x400 + PWMn * 64	0x400 + (PWMn + 1) * 64	R/W

30.9 寄存器列表

本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LEDC_CHO_CONFO_REG	通道 0 的配置寄存器 0	0x0000	varies
LEDC_CHO_HPOINT_REG	通道 0 的 Hpoint 寄存器	0x0004	R/W
LEDC_CHO_DUTY_REG	通道 0 的初始占空比	0x0008	R/W
LEDC_CHO_CONF1_REG	通道 0 的配置寄存器 1	0x000C	R/W/SC
LEDC_CH1_CONFO_REG	通道 1 的配置寄存器 0	0x0014	varies
LEDC_CH1_HPOINT_REG	通道 1 的 Hpoint 寄存器	0x0018	R/W
LEDC_CH1_DUTY_REG	通道 1 的初始占空比	0x001C	R/W
LEDC_CH1_CONF1_REG	通道 1 的配置寄存器 1	0x0020	R/W/SC
LEDC_CH2_CONFO_REG	通道 2 的配置寄存器 0	0x0028	varies
LEDC_CH2_HPOINT_REG	通道 2 的 Hpoint 寄存器	0x002C	R/W
LEDC_CH2_DUTY_REG	通道 2 的初始占空比	0x0030	R/W
LEDC_CH2_CONF1_REG	通道 2 的配置寄存器 1	0x0034	R/W/SC
LEDC_CH3_CONFO_REG	通道 3 的配置寄存器 0	0x003C	varies
LEDC_CH3_HPOINT_REG	通道 3 的 Hpoint 寄存器	0x0040	R/W
LEDC_CH3_DUTY_REG	通道 3 的初始占空比	0x0044	R/W
LEDC_CH3_CONF1_REG	通道 3 的配置寄存器 1	0x0048	R/W/SC
LEDC_CH4_CONFO_REG	通道 4 的配置寄存器 0	0x0050	varies
LEDC_CH4_HPOINT_REG	通道 4 的 Hpoint 寄存器	0x0054	R/W
LEDC_CH4_DUTY_REG	通道 4 的初始占空比	0x0058	R/W
LEDC_CH4_CONF1_REG	通道 4 的配置寄存器 1	0x005C	R/W/SC
LEDC_CH5_CONFO_REG	通道 5 的配置寄存器 0	0x0064	varies
LEDC_CH5_HPOINT_REG	通道 5 的 Hpoint 寄存器	0x0068	R/W
LEDC_CH5_DUTY_REG	通道 5 的初始占空比	0x006C	R/W
LEDC_CH5_CONF1_REG	通道 5 的配置寄存器 1	0x0070	R/W/SC
LEDC_TIMER0_CONF_REG	配置定时器 0	0x00A0	varies
LEDC_TIMER1_CONF_REG	配置定时器 1	0x00A8	varies
LEDC_TIMER2_CONF_REG	配置定时器 2	0x00B0	varies
LEDC_TIMER3_CONF_REG	配置定时器 3	0x00B8	varies
LEDC_CHO_GAMMA_CONF_REG	通道 0 伽马配置寄存器	0x0100	varies
LEDC_CH1_GAMMA_CONF_REG	通道 1 伽马配置寄存器	0x0104	varies
LEDC_CH2_GAMMA_CONF_REG	通道 2 伽马配置寄存器	0x0108	varies
LEDC_CH3_GAMMA_CONF_REG	通道 3 伽马配置寄存器	0x010C	varies
LEDC_CH4_GAMMA_CONF_REG	通道 4 伽马配置寄存器	0x0110	varies
LEDC_CH5_GAMMA_CONF_REG	通道 5 伽马配置寄存器	0x0114	varies
LEDC_EVT_TASK_EN0_REG	LEDC 事件任务使能寄存器 0	0x0120	R/W
LEDC_EVT_TASK_EN1_REG	LEDC 事件任务使能寄存器 1	0x0124	R/W

名称	描述	地址	访问
LEDC_EVT_TASK_EN2_REG	LEDC 事件任务使能寄存器 2	0x0128	R/W
LEDC_TIMER0_CMP_REG	定时器 0 的比较值	0x0140	R/W
LEDC_TIMER1_CMP_REG	定时器 1 的比较值	0x0144	R/W
LEDC_TIMER2_CMP_REG	定时器 2 的比较值	0x0148	R/W
LEDC_TIMER3_CMP_REG	定时器 3 的比较值	0x014C	R/W
LEDC_CONF_REG	LEDC 全局配置寄存器	0x0170	R/W
状态寄存器			
LEDC_CHO_DUTY_R_REG	通道 0 的当前占空比	0x0010	RO
LEDC_CH1_DUTY_R_REG	通道 1 的当前占空比	0x0024	RO
LEDC_CH2_DUTY_R_REG	通道 2 的当前占空比	0x0038	RO
LEDC_CH3_DUTY_R_REG	通道 3 的当前占空比	0x004C	RO
LEDC_CH4_DUTY_R_REG	通道 4 的当前占空比	0x0060	RO
LEDC_CH5_DUTY_R_REG	通道 5 的当前占空比	0x0074	RO
LEDC_TIMER0_VALUE_REG	配置定时器 0 的当前计数值	0x00A4	RO
LEDC_TIMER1_VALUE_REG	配置定时器 1 的当前计数值	0x00AC	RO
LEDC_TIMER2_VALUE_REG	配置定时器 2 的当前计数值	0x00B4	RO
LEDC_TIMER3_VALUE_REG	配置定时器 3 的当前计数值	0x00BC	RO
LEDC_TIMER0_CNT_CAP_REG	抓取定时器 0 的计数值	0x0150	RO
LEDC_TIMER1_CNT_CAP_REG	抓取定时器 1 的计数值	0x0154	RO
LEDC_TIMER2_CNT_CAP_REG	抓取定时器 2 的计数值	0x0158	RO
LEDC_TIMER3_CNT_CAP_REG	抓取定时器 3 的计数值	0x015C	RO
中断寄存器			
LEDC_INT_RAW_REG	原始中断寄存器	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	屏蔽中断寄存器	0x00C4	RO
LEDC_INT_ENA_REG	中断使能寄存器	0x00C8	R/W
LEDC_INT_CLR_REG	中断清除寄存器	0x00CC	WT
版本寄存器			
LEDC_DATE_REG	版本控制寄存器	0x0174	R/W

30.10 寄存器

本小节的所有地址均为相对于 LED PWM 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 [4 系统和存储器](#) 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 30.1. LEDC_CHn_CONFO_REG (n: 0-5) (0x0000+0x14*n)

(reserved)																LEDC_OVF_CNT_RESET_CHn LEDC_OVF_CNT_EN_CHn					LEDC_OVF_NUM_CHn					LEDC_PARA_UP_CHn LEDC_IDLE_LV_CHn LEDC_SIG_OUT_EN_CHn LEDC_TIMER_SEL_CHn				
31															17	16	15	14						5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	0	0						0	0	0		0	Reset	

LEDC_TIMER_SEL_CHn 配置通道 n 选取的定时器。

- 0: 定时器 0
- 1: 定时器 1
- 2: 定时器 2
- 3: 定时器 3
- (R/W)

LEDC_SIG_OUT_EN_CHn 配置是否开启通道 n 的信号输出。

- 0: 关闭
- 1: 开启
- (R/W)

LEDC_IDLE_LV_CHn 配置通道 n 关闭（即 LEDC_SIG_OUT_EN_CHn 为 0）时的输出电平值。

- 0: 低电平
- 1: 高电平
- (R/W)

LEDC_PARA_UP_CHn 配置是否更新以下字段：LEDC_HPOINT_CHn、LEDC_DUTY_START_CHn、LEDC_SIG_OUT_EN_CHn、LEDC_TIMER_SEL_CHn、LEDC_OVF_CNT_EN_CHn 和 占空比区间配置

- 0: 无效值，没有作用
- 1: 更新
- 该字段由硬件自清
- (WT)

LEDC_OVF_NUM_CHn 配置定时器的最大溢出次数减 1。

通道 n 的定时器溢出次数达到 (LEDC_OVF_NUM_CHn + 1 次时，触发 LEDC_OVF_CNT_CHn_INT 中断。(R/W)

LEDC_OVF_CNT_EN_CHn 配置是否开启通道 n 的溢出次数计数器。

- 0: 关闭
- 1: 开启
- (R/W)

见下页...

Register 30.1. LEDC_CHn_CONFO_REG (n: 0-5) (0x0000+0x14*n)

接上页...

LEDC_OVF_CNT_RESET_CHn 配置是否复位通道 n 的溢出次数计数器。

0: 无效值，没有作用

1: 复位

(WT)

Register 30.2. LEDC_CHn_HPOINT_REG (n: 0-5) (0x0004+0x14*n)

(reserved)												LEDC_HPOINT_CH ⁿ																																											
31												20												19																0															
0 0 0 0 0 0 0 0 0 0 0 0												0x000																Reset																											

LEDC_HPOINT_CHn 配置通道 n 输出的 Hpoint 值。当定时器达到该寄存器的规定值时，输出值变为高电平。(R/W)

Register 30.3. LEDC_CHn_DUTY_REG (n: 0-5) (0x0008+0x14*n)

(reserved)																LEDC_DUTY_CH ⁿ																															
31								25								24								0																							
0								0								0								0								0x00000								Reset							

LEDC_DUTY_CHn 配置通道 n 输出信号的占空比。(R/W)

Register 30.4. LEDC_CH n _CONF1_REG (n : 0-5) (0x000C+0x14* n)

Diagram illustrating the LDCR register structure. The register is 32 bits wide, with bits 31 down to 30 labeled `LED_C_DUTY_START_CHn` and bits 29 down to 0 labeled `(reserved)`. A 'Reset' value of 0 is indicated for the entire register.

LEDC_DUTY_START_CH_n 配置占空比渐变配置是否生效。

0: 不生效

1: 生效

(R/W/SC)

Register 30.5. LEDC_TIMER_x_CONF_REG (x: 0-3) (0x00A0+0x8*x)

(reserved)					LED_TIMER_X_PARA_UP					(reserved)					LED_TIMER_X_RST					LED_TIMER_X_PAUSE					LED_TIMER_X_DUTY_CYCLE															LED_TIMER_X_TIMER_X					LED_TIMER_X_DIV_TIMER_X					LED_TIMER_X_DUTY_CYCLE				
31					27					26	25	24	23	22																5	4	0																						
0					0					0	0	0	1	0	0x000															0x0															Reset									

LEDC_TIMERx_DUTY_RES 配置定时器 x 计数器的位宽，有效值为 1 到 20。(R/W)

LEDC_CLK_DIV_TIMER_x 配置定时器 _x 中分频器的分频系数。

低 8 位为小数部分，高 10 位为整数部分。(R/W)

LEDC TIMER_x PAUSE 配置是否暂停定时器 **x** 的计数器。

0: 不暂停

1: 暂停

(R/W)

LEDC_TIMER x _RST 配置是否复位定时器 x 的计数器（复位后，计数器的值为 0。）。

0: 不复位

1: 复位

(R/W)

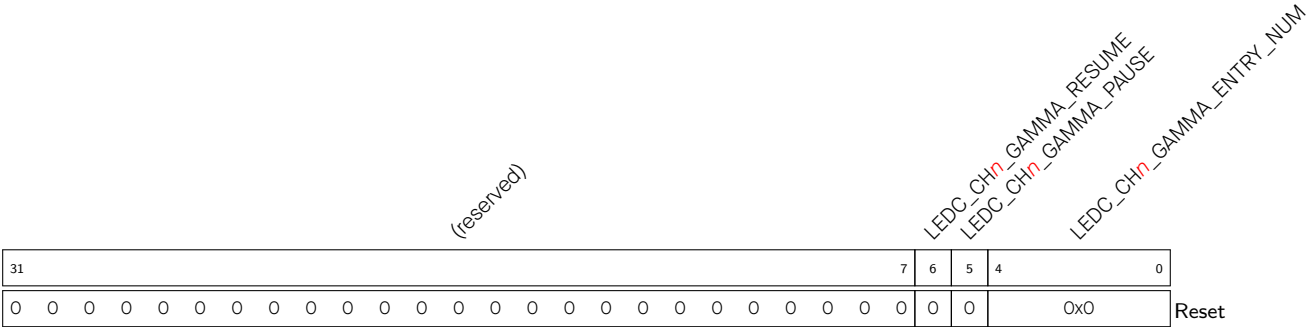
LEDC_TIMER_x_PARA_UP 配置是否更新 **LEDC_CLK_DIV_TIMER_x** 和 **LEDC_TIMER_x_DUTY_RES** 字段。

0: 无效值, 没有作用

1: 更新

(WT)

Register 30.6. LEDC_CHn_GAMMA_CONF_REG (n: 0-5) (0x0100+0x4*n)



LEDC_CHn_GAMMA_ENTRY_NUM 配置通道 n 的占空比渐变区间数量。(R/W)

LEDC_CHn_GAMMA_PAUSE 配置是否暂停占空比渐变。

- 0：无效值，没有作用
- 1：暂停
- (WT)

LEDC_CHn_GAMMA_RESUME 配置是否恢复已暂停的占空比渐变。

- 0：无效值，没有作用
- 1：恢复
- (WT)

Register 30.7. LEDC_EVT_TASK_ENO_REG (0x0120)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset																															

LEDC_EVT_DUTY_CHNG_END_CH n _EN (n : 0-5) 配置是否使能 LEDC_EVT_DUTY_CHNG_END_CH n 事件。

0: 关闭

1: 使能

(R/W)

LEDC_EVT_OVF_CNT_PLS_CH n _EN (n : 0-5) 配置是否使能 LEDC_EVT_OVF_CNT_PLS_CH n 事件。

0: 关闭

1: 使能

(R/W)

LEDC_EVT_TIME_OVF_TIMER_x_EN (x: 0-3) 配置是否使能 LEDC_EVT_TIME_OVF_TIMER_x 事件。

0: 关闭

1: 使能

(R/W)

LEDC_EVT_TIMER_x_CMP_EN (x: 0-3) 配置是否使能 LEDC_EVT_TIMER_x_CMP 事件。

0: 关闭

1: 使能

(R/W)

LEDC_TASK_DUTY_SCALE_UPDATE_CH n _EN (n : 0-5) 配 置 是 否 使 能
LEDC_TASK_DUTY_SCALE_UPDATE_CH n 事件。

0: 关闭

1: 使能

(R/W)

Register 30.8. LEDC_EVT_TASK_EN1_REG (0x0124)

LEDC_TASK_TIMER3_PAUSE_RESUME_EN																																LEDC_TASK_TIMER2_PAUSE_RESUME_EN																																LEDC_TASK_TIMER1_PAUSE_RESUME_EN																																LEDC_TASK_TIMER0_PAUSE_RESUME_EN																																LEDC_TASK_TIMER3_RST_EN																																LEDC_TASK_TIMER2_RST_EN																																LEDC_TASK_TIMER1_RST_EN																																(reserved)																																LEDC_TASK_OVF_CNT_RST_CH5_EN																																LEDC_TASK_OVF_CNT_RST_CH4_EN																																LEDC_TASK_OVF_CNT_RST_CH3_EN																																LEDC_TASK_OVF_CNT_RST_CH2_EN																																LEDC_TASK_OVF_CNT_RST_CH1_EN																																(reserved)																																LEDC_TASK_SIG_OUT_DIS_CH5_EN																																LEDC_TASK_SIG_OUT_DIS_CH4_EN																																LEDC_TASK_SIG_OUT_DIS_CH3_EN																																LEDC_TASK_SIG_OUT_DIS_CH2_EN																																LEDC_TASK_TIMER3_CAP_EN																																LEDC_TASK_TIMER2_CAP_EN																																LEDC_TASK_TIMER1_CAP_EN																																LEDC_TASK_TIMER0_CAP_EN																																LEDC_TASK_TIMER3_RES_UPDATE_EN																																LEDC_TASK_TIMER2_RES_UPDATE_EN																																LEDC_TASK_TIMER1_RES_UPDATE_EN																																LEDC_TASK_TIMER0_RES_UPDATE_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
31																																30																																29																																28																																27																																26																																25																																24																																23																																22																																21																																20																																19																																18																																17																																16																																15																																14																																13																																12																																11																																10																																9																																8																																7																																6																																5																																4																																3																																2																																1																																0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															
0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																																0																															

LEDC_TASK_TIMER x _RES_UPDATE_EN (x : 0-3) 配置是否使能 LEDC_TASK_TIMER x _RES_UPDATE 任务。

0: 关闭

1: 使能

(R/W)

LEDC_TASK_TIMER x _CAP_EN (x : 0-3) 配置是否使能 LEDC_TASK_TIMER x _CAP 任务。

0: 关闭

1: 使能

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH n _EN (n : 0-5) 配置是否使能 LEDC_TASK_SIG_OUT_DIS_CH n 任务。

0: 关闭

1: 使能

(R/W)

LEDC_TASK_OVF_CNT_RST_CH n _EN (n : 0-5) 配置是否使能 LEDC_TASK_OVF_CNT_RST_CH n 任务。

0: 关闭

1: 使能

(R/W)

LEDC_TASK_TIMER x _RST_EN (x : 0-3) 配置是否使能 LEDC_TASK_TIMER x _RST 任务。

0: 关闭

1: 使能

(R/W)

LEDC_TASK_TIMER x _PAUSE_RESUME_EN (x : 0-3) 配置是否使能 LEDC_TASK_TIMER x _PAUSE 和 LEDC_TASK_TIMER x _RESUME 任务。

0: 关闭

1: 使能

(R/W)

Register 30.9. LEDC_EVT_TASK_EN2_REG (0x0128)

[illegible]

LEDC_TASK_GAMMA_RESTART_CH n _EN (n : 0-5) 配置是否使能 LEDC_TASK_GAMMA_RESTART_CH n 任务。
0: 关闭
1: 使能
(R/W)

LEDC_TASK_GAMMA_PAUSE_CH n _EN (n : 0-5) 配置是否使能 LEDC_TASK_GAMMA_PAUSE_CH n 任务。
0: 关闭
1: 使能
(R/W)

LEDC_TASK_GAMMA_RESUME_CH n _EN (n : 0-5) 配置是否使能 LEDC_TASK_GAMMA_RESUME_CH n 任务。
0: 关闭
1: 使能
(R/W)

Register 30.10. LEDC_TIMER_x_CMP_REG (x: 0-3) (0x0140+0x4*x)

(reserved)																LEDC_TIMER_CMP															
31																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x000															

Reset

LEDC_TIMER_x_CMP 配置 LEDC 定时器 **x** 的比较值。(R/W)

Register 30.11. LEDC_CONF_REG (0x0170)

Register 0x00000000: LEDC_CLK_EN (bits 31:0). The register is 32 bits wide. Bits 31:8 are reserved. Bits 7:0 are LEDC_CLK_EN. The diagram shows the register structure with bit positions 31, 30, 8, 7, 6, 5, 4, 3, 2, 1, 0. The LEDC_CLK_EN signal is connected to bit 0. The Reset signal is connected to the reset pin of the register block.

LEDC_GAMMA_RAM_CLK_EN_CH n (n : 0-5) 配置是否开启通道 n 伽马 RAM 时钟门控。

0: 仅在应用读或写通道 *n* 的伽马 RAM 时开启时钟门控

1: 强制为通道 **n** 伽马 RAM 开启时钟门控

(R/W)

LEDC CLK EN 配置是否开启寄存器门控。

0: 仅在应用写寄存器时开启时钟门控

1: 强制为寄存器开启时钟门控

(R/W)

Register 30.12. LEDC_CH n _DUTY_R_REG (n : 0-5) (0x0010+0x14* n)

Register structure for LEDC_DUTY_CHn_R:

31	25	24	0
(reserved)		0x000000	

LEDC_DUTY_CH n _R 通道 n 输出信号的当前占空比。(RO)

Register 30.13. LEDC_TIMER_x_VALUE_REG (x: 0-3) (0x00A4+0x8*x)

Register structure for LEDC_TIMER_CNT:

- Bits 31-20: (reserved)
- Bits 19-0: LEDC_TIMER_CNT (Timer Counter)
- Reset

LEDC_TIMER_x_CNT 定时器 *x* 计数器的当前数值。(RO)

Register 30.14. LEDC_TIMER x _CNT_CAP_REG (x : 0-3) (0x0150+0x4* x)

(reserved)												LEDC_TIMER x _CNT_CAP											
31											20	19											0
0	0	0	0	0	0	0	0	0	0	0	0	0	0x000										Reset

LEDC_TIMER x _CNT_CAP 抓取的定时器 x 的计数器数值。(RO)

Register 30.15. LEDC_INT_RAW_REG (0x00C0)

(reserved)																		LEDC_OVF_CNT_CH5_INT_RAW																		LEDC_OVF_CNT_CH4_INT_RAW																		LEDC_OVF_CNT_CH3_INT_RAW																		LEDC_OVF_CNT_CH2_INT_RAW																		LEDC_OVF_CNT_CH1_INT_RAW																		(reserved)																		LEDC_DUTY_CHNG_END_CH5_INT_RAW																		LEDC_DUTY_CHNG_END_CH4_INT_RAW																		LEDC_DUTY_CHNG_END_CH3_INT_RAW																		LEDC_DUTY_CHNG_END_CH2_INT_RAW																		LEDC_DUTY_CHNG_END_CH1_INT_RAW																		LEDC_TIMER3_OVF_INT_RAW																		LEDC_TIMER2_OVF_INT_RAW																		LEDC_TIMER1_OVF_INT_RAW																		LEDC_TIMER0_OVF_INT_RAW																	
31																		18																		17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset																																																																																																																																																																																																																																									
0																		0																		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																																																																																																																																																																																																																								

- LEDC_TIMER x _OVF_INT_RAW (x : 0-3) 原始状态位: LEDC_TIMER x _OVF_INT 的原始中断状态。定时器 x 的计数器到达最大值时, 触发该中断。(R/WTC/SS)
- LEDC_DUTY_CHNG_END_CH n _INT_RAW (n : 0-5) 原始状态位: LEDC_DUTY_CHNG_END_CH n _INT 的原始中断状态。占空比渐变结束时触发该中断。(R/WTC/SS)
- LEDC_OVF_CNT_CH n _INT_RAW (n : 0-5) 原始状态位: LEDC_OVF_CNT_CH n _INT 的原始中断状态。当 ovf_cnt 达到 LEDC_OVF_NUM_CH n 的设定值时, 触发该中断。(R/WTC/SS)

Register 30.16. LEDC_INT_ST_REG (0x00C4)

(reserved)																																LEDC_OVF_CNT_CH5_INT_ST LEDC_OVF_CNT_CH4_INT_ST LEDC_OVF_CNT_CH3_INT_ST LEDC_OVF_CNT_CH2_INT_ST LEDC_OVF_CNT_CH1_INT_ST LEDC_OVF_CNT_CHO_INT_ST (reserved) LEDC_DUTY_CHNG_END_CH5_INT_ST LEDC_DUTY_CHNG_END_CH4_INT_ST LEDC_DUTY_CHNG_END_CH3_INT_ST LEDC_DUTY_CHNG_END_CH2_INT_ST LEDC_DUTY_CHNG_END_CH1_INT_ST LEDC_TIMER3_OVF_INT_ST LEDC_TIMER2_OVF_INT_ST LEDC_TIMER1_OVF_INT_ST LEDC_TIMER0_OVF_INT_ST																				
31																18																17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0																0																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

LEDC_TIMER x _OVF_INT_ST (x : 0-3) 屏蔽状态位: LEDC_TIMER x _OVF_INT 的屏蔽中断状态。

仅在 LEDC_TIMER x _OVF_INT_ENA 为 1 时有效。(RO)

LEDC_DUTY_CHNG_END_CH n _INT_ST (n : 0-5) 屏蔽状态位: LEDC_DUTY_CHNG_END_CH n _INT 的屏蔽中断状态。

仅在 LEDC_DUTY_CHNG_END_CH n _INT_ENA 为 1 时有效。(RO)

LEDC_OVF_CNT_CH n _INT_ST (n : 0-5) 屏蔽状态位: LEDC_OVF_CNT_CH n _INT 的屏蔽中断状态。

仅在 LEDC_OVF_CNT_CH n _INT_ENA 为 1 时有效。(RO)

Register 30.17. LEDC_INT_ENA_REG (0x00C8)

(reserved)																																LEDC_OVF_CNT_CH5_INT_ENA LEDC_OVF_CNT_CH4_INT_ENA LEDC_OVF_CNT_CH3_INT_ENA LEDC_OVF_CNT_CH2_INT_ENA LEDC_OVF_CNT_CH1_INT_ENA LEDC_OVF_CNT_CHO_INT_ENA (reserved) LEDC_DUTY_CHNG_END_CH5_INT_ENA LEDC_DUTY_CHNG_END_CH4_INT_ENA LEDC_DUTY_CHNG_END_CH3_INT_ENA LEDC_DUTY_CHNG_END_CH2_INT_ENA LEDC_DUTY_CHNG_END_CH1_INT_ENA LEDC_TIMER3_OVF_INT_ENA LEDC_TIMER2_OVF_INT_ENA LEDC_TIMER1_OVF_INT_ENA LEDC_TIMER0_OVF_INT_ENA																			
31																18																17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0																0																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

LEDC_TIMER x _OVF_INT_ENA (x : 0-3) 使能位: 写 1 使能 LEDC_TIMER x _OVF_INT。(R/W)

LEDC_DUTY_CHNG_END_CH n _INT_ENA (n : 0-5) 使能位: 写 1 使能 LEDC_DUTY_CHNG_END_CH n _INT。(R/W)

LEDC_OVF_CNT_CH n _INT_ENA (n : 0-5) 使能位: 写 1 使能 LEDC_OVF_CNT_CH n _INT。(R/W)

Register 30.18. LEDC_INT_CLR_REG (0x00CC)

[illegible]

LEDC_TIMER_x_OVF_INT_CLR (x: 0-3) 清零位: 写1清零 LEDC_TIMER_x_OVF_INT。(WT)

LEDC_DUTY_CHNG_END_CH n _INT_CLR (n : 0-5) 清 零 位: 写 1 清 零
LEDC_DUTY_CHNG_END_CH n _INT。(WT)

LEDC_OVF_CNT_CH n _INT_CLR (n : 0-5) 清零位: 写 1 清零 LEDC_OVF_CNT_CH n _INT。(WT)

Register 30.19. LEDC_DATE_REG (0x0174)

(reserved)				LEDC_LEDC_DATE																				
31	28	27	0																					
0	0	0	0	0x2311220																				Reset

LEDC_LEDC_DATE 版本控制寄存器。(R/W)

第 31 章

电机控制脉宽调制器 (MCPWM)

31.1 概述

电机控制脉宽调制器 (MCPWM) 外设用于电机和电源控制。该外设提供了 6 个 PWM 输出，可在几种拓扑结构中运行。常见的拓扑结构之一是用一对 PWM 输出来驱动 H 桥以控制电机旋转速度和旋转方向。

MCPWM 外设主要分为五个子模块：PWM 定时器、PWM 操作器、捕获模块、事件矩阵 (Event Task Matrix, ETM) 模块、故障检测模块。PWM 定时器提供参考时序，可以自由运行，或同步到其他定时器或外部源。每个 PWM 操作器具有生成一个 PWM 通道波形对所需的所有控制资源。捕获模块用于需要精确定时外部事件的系统。ETM 模块对输入到 MCPWM 的任务做出相应的响应动作，根据不同的运动状态产生相应的事件。故障检测模块用于捕获外部故障，使得系统可以选择做出响应。

ESP32-C5 有一个 MCPWM 外设，即 MCPWM0。

31.2 主要特性

MCPWM 外设有一个时钟分频器（预分频器）、三个 PWM 定时器、三个 PWM 操作器、一个捕获模块、一个 ETM 模块和一个故障检测模块。MCPWM 的主时钟可以从三个时钟源中任选其一，分别为：PLL_F160M_CLK, XTAL_CLK 和 RC_FAST_CLK（通过 [PCR_PWM_CLK_CONF_REG](#) 寄存器的 PCR_PWM_CLKM_SEL 字段配置）。图 31.2-1 描述了 MCPWM 内的模块和接口上的信号。PWM 定时器用于生成定时参考，PWM 操作器将根据生成的定时参考生成所需的波形。通过配置，任一 PWM 操作器可以使用任一 PWM 定时器的定时参考。不同的 PWM 操作器可以使用相同的 PWM 定时器的定时参考产生 PWM 信号，也可以使用不同的 PWM 定时器的值来生成单独的 PWM 信号。不同的 PWM 定时器可以进行同步。

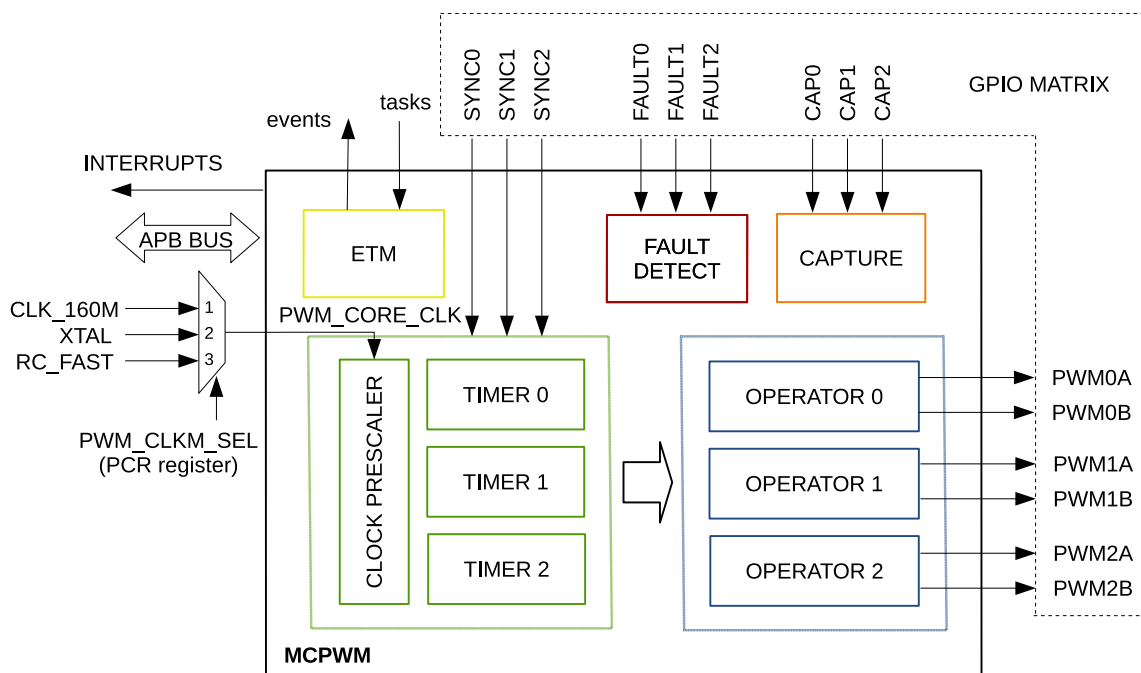


图 31.2-1. MCPWM 外设概览

以下是图 31.2-1 中模块的功能概述：

- PWM 定时器 0、1 和 2：
 - 每个 PWM 定时器都有一个专用的 8 位时钟预分频器。
 - PWM 定时器中的 16 位计数器的工作模式包括：递增计数模式，递减计数模式，递增递减循环计数模式。
 - 硬件/软件同步可以触发 PWM 定时器重载，重载值位于相位寄存器中；同时触发预分频的重启，从而同步定时器的时钟。硬件同步源可以来自任何 GPIO 或任何其他 PWM 定时器的 sync_out 信号。软件同步源来自于向 `MCPWM_TIMERn_SYNC_SW` 位写入取反值。
- PWM 操作器 0、1 和 2：
 - 每个 PWM 操作器有两个 PWM 输出（PWMxA 和 PWMxB），可以在对称和非对称配置中独立工作。
 - 可以通过异步方式更新对 PWM 信号的控制。
 - 死区时间在上升沿和下降沿可配置，并可分别设置。
 - 所有事件都可触发 CPU 中断。
 - 通过高频载波信号调制 PWM 输出，在使用变压器隔离栅极驱动器时可发挥巨大作用。
 - 周期、时间戳寄存器和其他主要的控制寄存器有影子寄存器，具有灵活的更新方式。
- 故障检测模块：
 - 出现故障时，可选择在逐周期模式或一次性模式下处理。
 - 出现故障时可强制 PWM 输出高或低电平。
- 捕获模块：
 - 捕获模块的时钟为 MCPWM 的主时钟。

- 旋转电机的速度测量。
 - 位置传感器脉冲之间的间隔时间测量。
 - 脉冲序列信号的周期和占空比测量。
 - 从电流/电压传感器的占空比编码信号导出的解码电流或电压振幅。
 - 3 个独立的捕获通道，各具备一个 32 位的时间戳寄存器。
 - 输入捕获信号可以预分频，边沿极性可选。
 - 捕获定时器可以与 PWM 定时器或外部信号同步。
 - 3 个捕获通道上都可以产生中断。
- ETM 模块：
 - 根据每个定时器和操作器不同的运行状态，产生不同的事件。
 - 每个定时器和操作器响应其对应的任务，自动进行相应的操作。
 - 每个事件和任务都可以独立使能。某个事件不使能时，不产生对应的事件；某个任务不使能时，不响应对应的任务。

31.3 模块

31.3.1 概述

每个 MCPWM 包含的关键模块为定时器模块、操作器模块、故障检测模块、捕获模块和 ETM 模块，关键模块的功能详见下文。

31.3.1.1 定时器模块

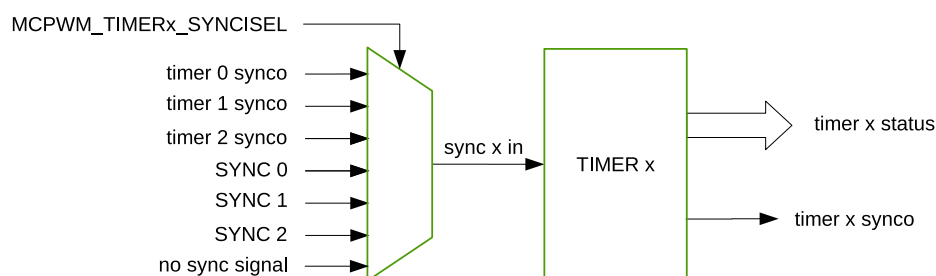


图 31.3-1. 定时器模块

该模块包含一个定时器，它可以按照指定的周期进行计数，它可以工作在递增计数模式，递减计数模式或递增递减循环计数模式下，它可以选择不同的同步输入源（共 7 个可选输入源）来进行计数值和方向的重载，从而使多个定时器彼此同步，它还可以输出一个同步输出源（共 4 个可选输出源）供其他定时器使用。

图中的 timer x status 指定定时器模块输出的定时器状态，如计数模式，计数值等于 0 或周期等。

31.3.1.2 操作器模块

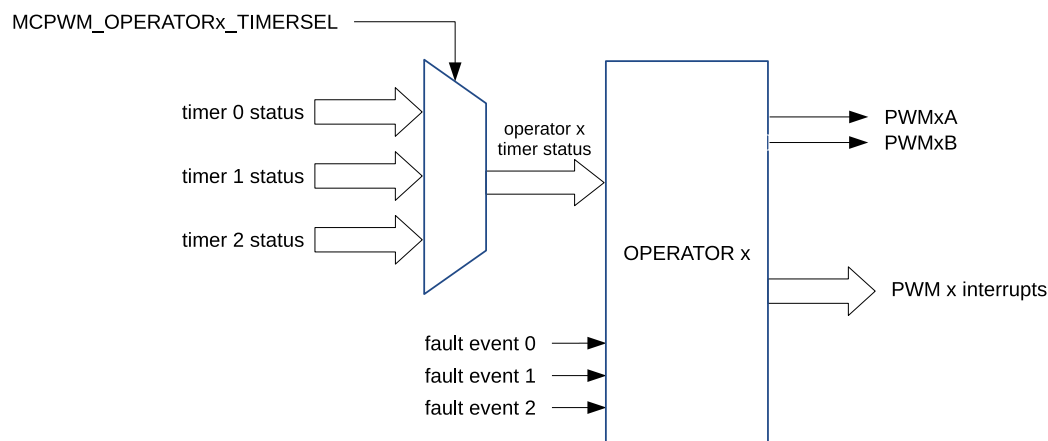


图 31.3-2. 操作器模块

一个 PWM 操作器包含一个 PWM 生成器、一个死区生成器、一个故障检测模块以及一个 PWM 载波模块，它们的功能如表 31.3-1 所示：

表 31.3-1. 操作器模块包含的各个子模块功能

子模块	功能
PWM 生成器	按照指定的占空比和波形，生成 PWM 信号
死区生成器	可以选择对 PWM 生成器产生的 PWM 信号的上升沿和下降沿添加死区（即对上升沿和下降沿添加延迟）以及进行反相等操作
PWM 载波	可以选择对死区生成器产生的 PWM 信号添加载波，载波的频率可以配置，并且添加载波后的 PWM 波形的第一个脉冲持续时间可以配置
故障检测	检测故障并生成对应的故障事件，按照指定的间隔模式（一次性模式或逐周期模式）响应故障事件以及发生故障事件时，按照指定的方式对 PWM 载波模块产生的 PWM 信号采取操作

31.3.1.3 捕获模块

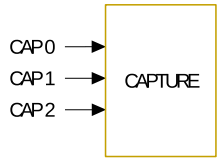


图 31.3-3. 捕获模块

该模块包含一个可以进行同步的定时器，当输入的捕获信号有效时，将捕获定时器的计数值。

31.3.1.4 ETM 模块

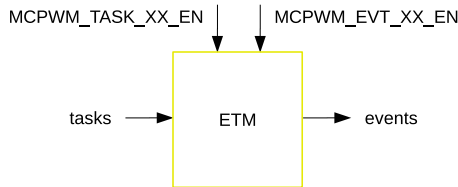


图 31.3-4. ETM 模块

该模块对接收到的任务进行响应，或产生事件并输出，可以独立配置每个事件和任务的使能。

31.3.2 PWM 定时器模块

MCPWM 外设有三个 PWM 定时器模块。它们中的任何一个都可以决定三个 PWM 操作器模块中任意一个的必要事件时序。通过使用 GPIO 交换矩阵的同步信号，内置同步逻辑允许一个或多个 MCPWM 外设中的多个 PWM 定时器模块作为一个系统协同工作。

31.3.2.1 PWM 定时器模块的配置

用户可配置 PWM 定时器模块的以下功能：

- 通过指定 PWM 定时器频率或周期来控制事件发生的频率

- 配置某个 PWM 定时器与其他 PWM 定时器或模块进行相位同步
- 配置定时器计数模式：递增，递减，或递增递减循环计数模式
- 使用预分频器更改 PWM 定时器时钟 (PT_CLK) 的速率。每个定时器都有自己的预分频器，通过寄存器 `MCPWM_TIMER0_CFG0_REG` 的 `MCPWM_TIMER $n$ _PRESCALE` 配置。PWM 定时器根据该寄存器的设置以较慢的速度递增或递减。当定时器停止后并再次开始计数时，新的 `MCPWM_TIMER $n$ _PRESCALE` 配置值才会生效。

31.3.2.2 PWM 定时器工作模式和定时事件生成

PWM 定时器有三种工作模式，由 `PWM $x$` 定时器模式字段配置：

- 递增计数模式：
定时器从零增加到周期字段中配置的值。达到周期值后，PWM 定时器清零，并再次开始递增。PWM 周期 = 周期寄存器中的周期值 + 1。
说明：周期寄存器为 `MCPWM_TIMER $n$ _PERIOD` ($x = 0, 1, 2$)，对应 `MCPWM_TIMER0_PERIOD`、`MCPWM_TIMER1_PERIOD`、`MCPWM_TIMER2_PERIOD`。
- 递减计数模式：
PWM 定时器从周期寄存器中的值开始递减到零。达到零后，将恢复为周期值，并再次开始递减。在这种情况下，PWM 周期 = 周期寄存器中的周期值 + 1。
- 递增-递减循环模式：
此模式结合了上述两种模式。PWM 定时器从零开始递增，直到达到周期值，再次递减为零。PWM 定时器按照此模式循环递增递减。PWM 周期 = 周期寄存器的周期值 $\times$ 2。

图 31.3-5 至 31.3-8 显示不同模式下的 PWM 定时器波形，包括同步事件期间的定时器行为。递增计数模式中，同步后永远保持递增计数；递减计数模式中，同步后永远保持递减计数；递增-递减循环模式中，同步后由 `MCPWM_TIMER $n$ _PHASE_DIRECTION` 配置计数方向。

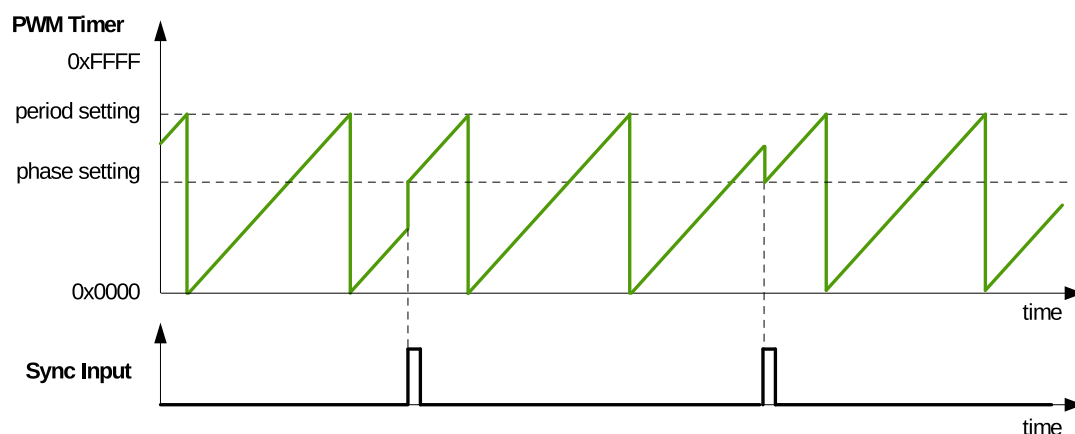


图 31.3-5. 递增计数模式波形

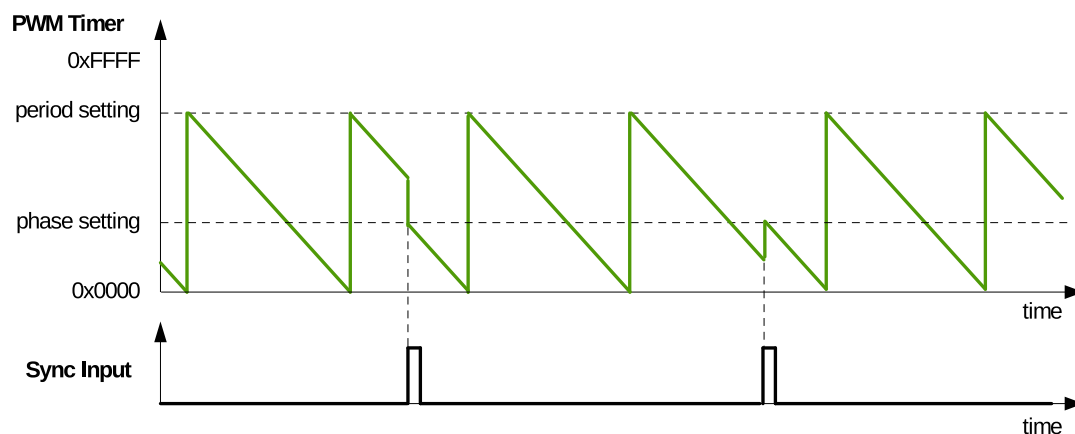


图 31.3-6. 递减计数模式波形

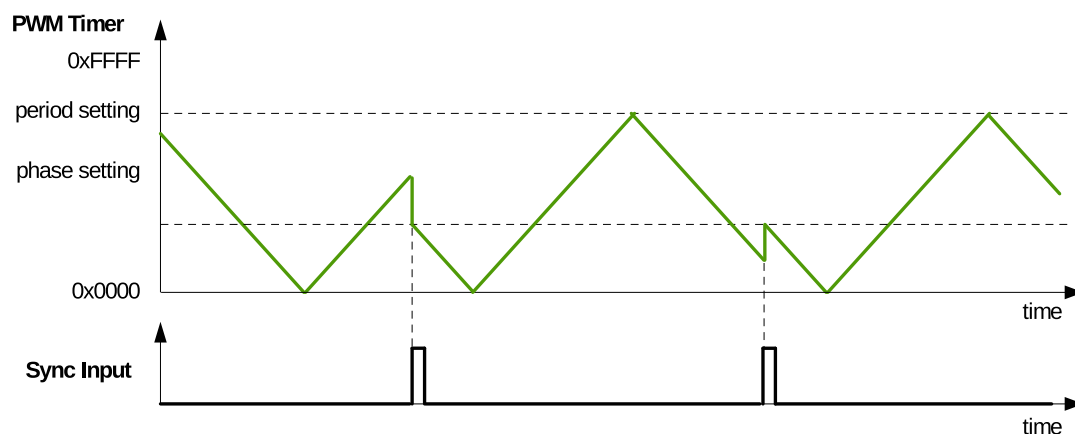


图 31.3-7. 递增递减循环模式波形，同步事件后递减

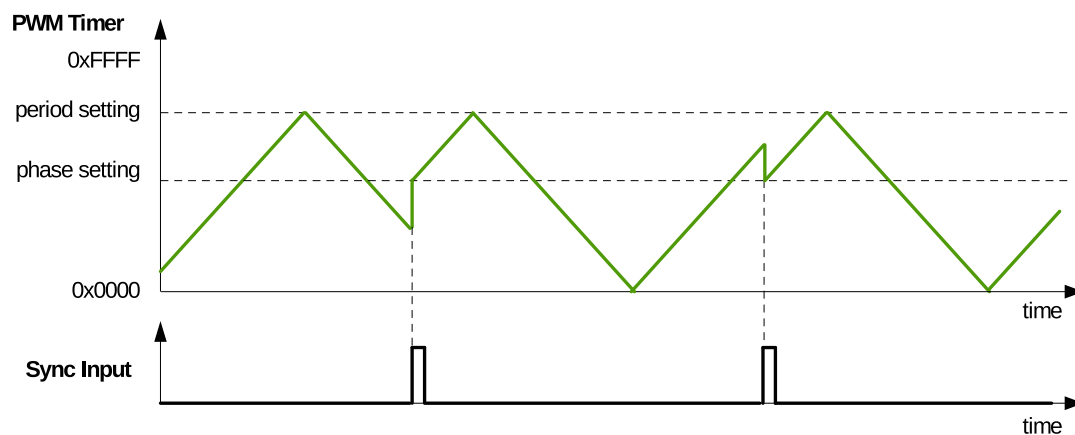


图 31.3-8. 递增递减循环模式波形，同步事件后递增

PWM 定时器运行时，定期自动生成以下定时事件：

- UTEP: 当 PWM 定时器等于周期字段的值 (`MCPWM_TIMER $n$ _PERIOD`) 且 PWM 定时器递增计数时, 生成的定时事件。
- UTEZ: 当 PWM 定时器等于零且 PWM 定时器递增计数时, 生成的定时事件。
- DTEP: 当 PWM 定时器等于周期字段的值 (`MCPWM_TIMER $n$ _PERIOD`) 且 PWM 定时器递减时, 生成的定时事件。
- DTEZ: 当 PWM 定时器等于零且 PWM 定时器递减时, 生成的定时事件。

图 31.3-9 至 31.3-11 为 U/DTEP 和 U/DTEZ 定时事件的时序波形。

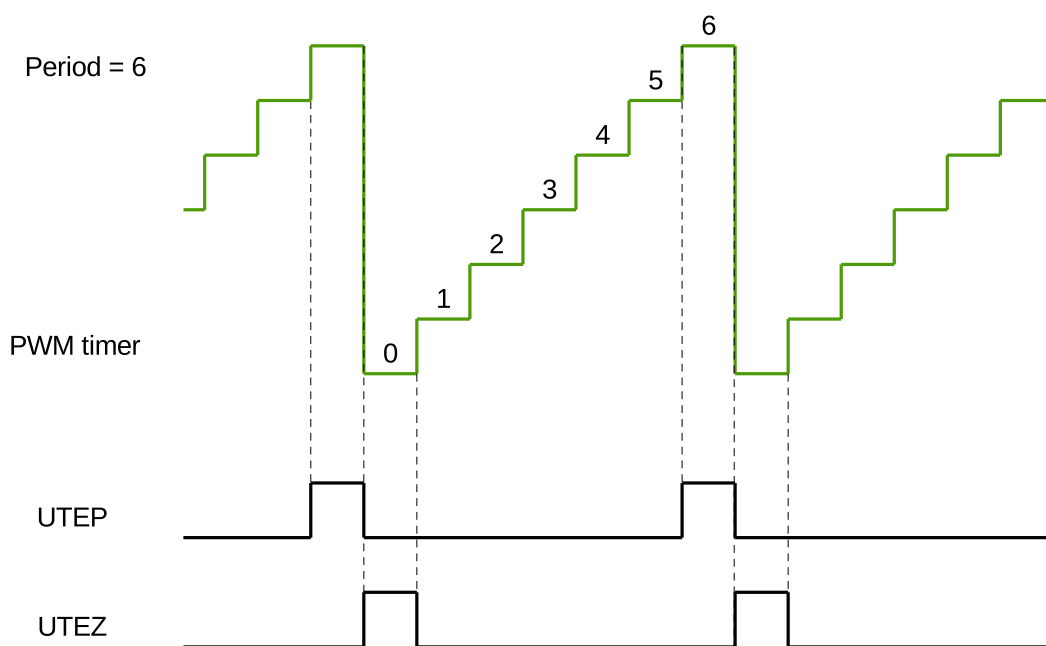


图 31.3-9. 递增模式中生成的 UTEP 和 UTEZ

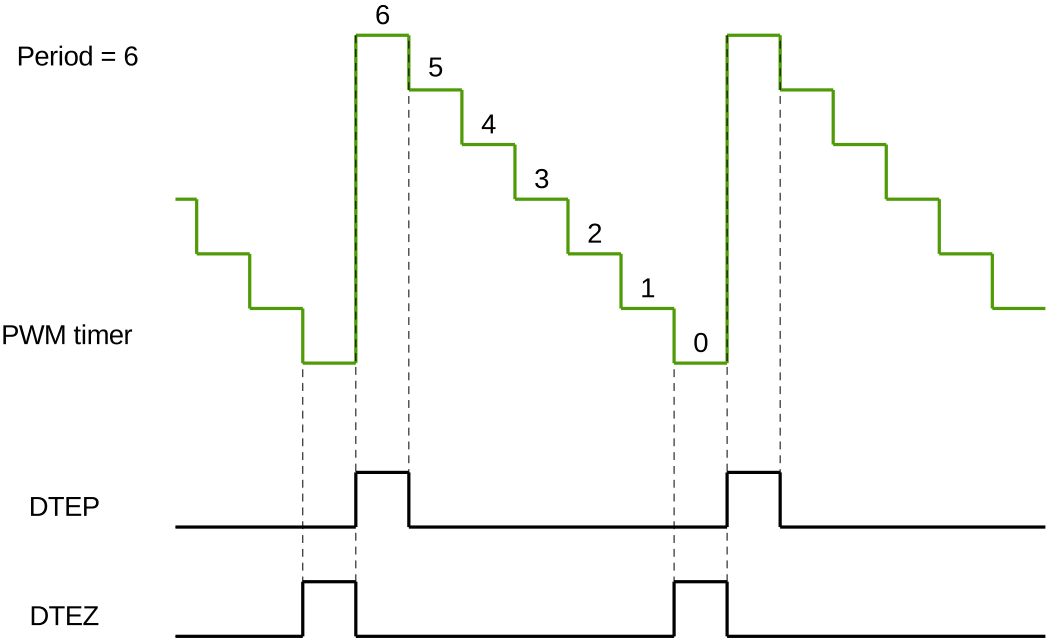


图 31.3-10. 递减模式中生成的 DTEP 和 DTEZ

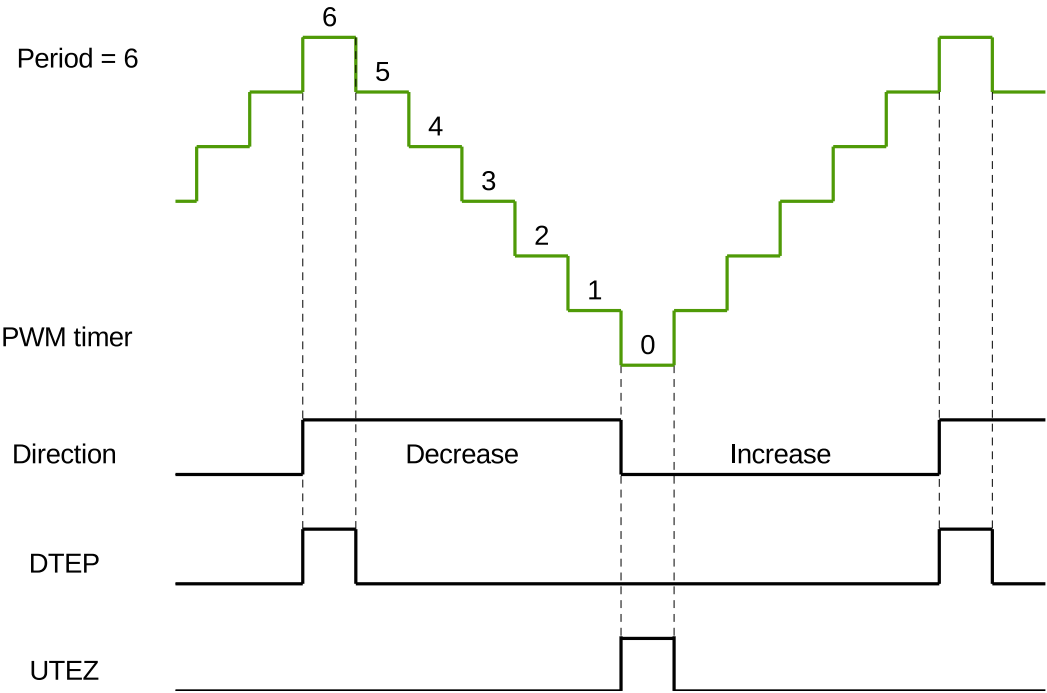


图 31.3-11. 递增递减模式中生成的 DTEP 和 UTEZ

需注意，在递增递减模式下，当定时器递增计数时，定时器范围为 [0, 周期值 - 1]；递减计数时，定时器范围为 [周期值, 1]。也就是说，在该模式下，将定时器同步为 0 时，则不支持递减计数，即

`MCPWM_TIMER $n$ _PHASE_DIRECTION` 不能为 1；定时器同步为周期值时，也不支持递增计数，即 `MCPWM_TIMER $n$ _PHASE_DIRECTION` 不能为 0。所以，当定时器同步为 0 时，计数方向只能为递增，即将 `MCPWM_TIMER $n$ _PHASE_DIRECTION` 配置为 0；定时器同步为周期值时，计数方向只能为递减，即将 `MCPWM_TIMER $n$ _PHASE_DIRECTION` 配置为 1。

31.3.2.3 PWM 定时器影子寄存器

PWM 定时器周期寄存器和 PWM 定时器时钟预分频器寄存器配有影子寄存器。影子寄存器能够备份即将写入有效寄存器的值，并在硬件同步的特定时刻写入有效寄存器。有效寄存器和影子寄存器的功能如下：

- 有效寄存器：直接控制硬件执行的所有操作。
- 影子寄存器：作为临时缓冲区，存储即将写入有效寄存器的值。在用户配置的某个时间点，影子寄存器中的值被写入有效寄存器。在此之前，影子寄存器的内容对受控硬件没有任何直接影响，防止寄存器由软件异步修改时可能发生的错误硬件操作。影子寄存器和有效寄存器具有相同的存储器地址。软件始终在影子寄存器中进行读写操作。

当定时器开始工作时，时钟预分频器的有效寄存器更新。当 `MCPWM_GLOBAL_UP_EN` 置 1 时，可通过以下方式选择更新有效寄存器的时间点：

- 将 `MCPWM_TIMER $n$ _PERIOD_UPMETHOD` 置 0，立即更新。
- 将 `MCPWM_TIMER $n$ _PERIOD_UPMETHOD` 置 1，当 PWM 定时器值为 0 时，开始更新。
- 将 `MCPWM_TIMER $n$ _PERIOD_UPMETHOD` 置 2，当 PWM 定时器同步时，开始更新。
- 将 `MCPWM_TIMER $n$ _PERIOD_UPMETHOD` 置 3，当 PWM 定时器值为 0 时或同步时，开始更新。
- 软件也可以触发全局强制更新位 `MCPWM_GLOBAL_FORCE_UP`，该位将触发模块中的所有寄存器根据影子寄存器进行更新。

31.3.2.4 PWM 定时器同步和锁相

PWM 模块采用灵活的同步方法。每个 PWM 定时器都有一个同步输入和一个同步输出。同步输入可以从 GPIO 矩阵的三个同步输出和三个 PWM 定时器的三个同步输出信号中选择。同步输出可以在 PWM 定时器等于周期、PWM 定时器等于零、同步输入有效或软件同步时生成。因此，每个 PWM 定时器都可以灵活的选择同步源，从而实现多种相位同步方式。在同步时，PWM 定时器的预分频器会复位其计数器，并重新开始对时钟的分频，以保证定时器的正确性。

31.3.3 PWM 操作器模块

PWM 操作器模块具备以下功能：

- 根据相应 PWM 定时器的定时参考生成 PWM 信号对。
- PWM 信号对的每个信号都可以独立设置特定的死区（即对 PWM 信号的上升沿和下降沿添加延迟）。
- 可通过配置将载波叠加到 PWM 信号上。
- 故障条件下处理响应。

图 31.3-12 为 PWM 操作器的框图。

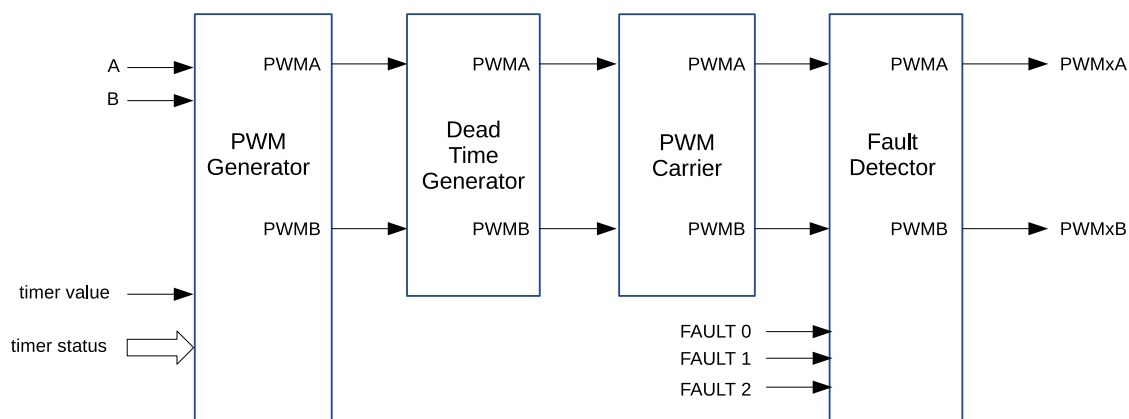


图 31.3-12. PWM 操作器的子模块

31.3.3.1 PWM 生成器模块

PWM 生成器模块的作用

此模块可以生成或导入重要的定时事件，转化为特定操作后，在 PWMxA 和 PWMxB 输出处生成所需的波形。

PWM 生成器模块执行以下操作：

- 基于使用寄存器 A 和 B 配置的时间戳生成定时事件。满足以下条件时发生定时事件（寄存器 A 和 B 的配置方法在章节 31.3.3.1 描述）：
 - UTEA: PWM 定时器递增计数并且其值等于寄存器 A。
 - UTEB: PWM 定时器递增计数并且其值等于寄存器 B。
 - DTEA: PWM 定时器递减计数并且其值等于寄存器 A。
 - DTEB: PWM 定时器递减计数并且其值等于寄存器 B。
- 基于故障或同步事件生成 U/DT0、U/DT1 定时事件。
 - UT0: 定时器递增计数并且检测到 FAULT0 事件（字段 MCPWM_GENn_TO_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENn_TO_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENn_TO_SEL 置 2）或同步（字段 MCPWM_GENn_TO_SEL 置 3）。
 - UT1: 定时器递增计数并且检测到 FAULT0 事件（字段 MCPWM_GENn_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENn_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENn_T1_SEL 置 2）或同步（字段 MCPWM_GENn_T1_SEL 置 3）。
 - DT0: 定时器递减计数并且检测到 FAULT0 事件（字段 MCPWM_GENn_TO_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENn_TO_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENn_TO_SEL 置 2）或同步（字段 MCPWM_GENn_TO_SEL 置 3）。
 - DT1: 定时器递减计数并且检测到 FAULT0 事件（字段 MCPWM_GENn_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENn_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENn_T1_SEL 置 2）或同步（字段 MCPWM_GENn_T1_SEL 置 3）。
- 当这些定时事件同时发生时管理优先级。
- 基于定时事件产生置 1、置 0 和取反操作。

- 根据 PWM 生成器模块的配置来控制 PWM 占空比。
- 使用影子寄存器处理新的时间戳值，以防止 PWM 波形中的毛刺干扰。

PWM 生成器影子寄存器

硬件使用的时间戳寄存器 A 和 B 都配有影子寄存器，为 `MCPWM_GENn_A_REG` 和 `MCPWM_GENn_B_REG`。影子寄存器提供了一种与硬件同步更新寄存器的方法。

当 `MCPWM_GLOBAL_UP_EN` 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。`MCPWM_GENn_A_REG` 和 `MCPWM_GENn_B_REG` 的更新方式字段为 `MCPWM_GENn_CFG_UPMETHOD`。软件也可以触发全局强制更新位 `MCPWM_GLOBAL_FORCE_UP`，该位将触发模块中的所有寄存器根据影子寄存器进行更新。更多关于影子寄存器的描述，请参考章节 31.3.2.3。

定时事件

表 31.3-2 概括了所有定时信号和事件。

表 31.3-2. PWM 生成器中的所有定时事件

信号	事件描述	PWM 定时器操作
DTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递减计数
DTEZ	PWM 定时器的值等于 0	
DTEA	PWM 定时器的值等于寄存器 A	
DTEB	PWM 定时器的值等于寄存器 B	
DT0 事件	基于故障或同步事件	
DT1 事件	基于故障或同步事件	
UTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递增计数
UTEZ	定时器的值等于 0	
UTEA	PWM 定时器的值等于寄存器 A	
UTEB	PWM 定时器的值等于寄存器 B	
UT0 事件	基于故障或同步事件	
UT1 事件	基于故障或同步事件	
软件强制事件	软件触发的异步事件	-

软件强制事件用于在 PWMxA 和 PWMxB 的输出上施加非连续或连续的强制电平。此更改是异步完成的。软件强制事件由寄存器 `MCPWM_GENn_FORCE_REG` 控制。

PWM 生成器模块中 TO/T1 的选择和配置独立于故障处理模块中的故障事件的配置，某个特定的跳闸事件可以不被配置为在故障处理器模块中引起跳闸动作，而是用于由 PWM 生成器用于触发 TO/T1 以控制 PWM 波形。

需要注意的是，当 PWM 定时器处于递增递减循环计数模式时，它将在 TEP 事件后递减，在 TEZ 事件后递增。因此，在此模式下，将出现 DTEP 和 UTEZ，但不会出现 UTEP 和 DTEZ。

PWM 生成器可以同时处理多个事件。事件优先级由硬件决定，详见表 31.3-3 和表 31.3-4。优先级从 1（最高）到 7（最低）排列。需要注意的是，TEP 和 TEZ 事件的优先级取决于 PWM 定时器的计数模式。

如果 A 或 B 的值设置为大于周期，则 U/DTEA 和 U/DTEB 将永远不会发生。

表 31.3-3. PWM 定时器递增计数时，定时事件的优先级

优先级	事件
1 (最高)	软件强制事件
2	UTEP
3	UTO
4	UT1
5	UTEB
6	UTEA
7 (最低)	UTEZ

表 31.3-4. PWM 定时器递减计数时，定时事件的优先级

优先级	事件
1 (最高)	软件强制事件
2	DTEZ
3	DTO
4	DT1
5	DTEB
6	DTEA
7 (最低)	DTEP

说明：

1. UTEP 和 UTEZ 不同时发生。当 PWM 定时器处于递增计数模式，UTEP 将始终比 UTEZ 提前一个周期发生，如图 31.3-9 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，UTEP 不会发生。
2. DTEP 和 DTEZ 不同时发生。当 PWM 定时器处于递减计数模式时，DTEZ 始终比 DTEP 早一个周期发生，如图 31.3-10 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，DTEZ 不会发生。

PWM 信号生成

当某个定时事件发生时，PWM 生成器控制输出 PWMxA 和 PWMxB 的电平。定时事件通过 PWM 定时器计数模式（递增或递减）进一步限定。根据定时器计数方向，模块可以对 PWM 定时器递增或递减计数的阶段执行不同的操作。

可以在 PWMxA 和 PWMxB 输出上配置以下操作：

- 置为高电平：将 PWMxA 或 PWMxB 的输出配置为高电平。
- 置为低电平：通过将 PWMxA 或 PWMxB 的输出配置为低电平，来清除 PWMxA 或 PWMxB 的输出。
- 取反：将 PWMxA 或 PWMxB 的当前输出电平配置为相反的值。如果它当前被拉高，则拉低，或反之。
- 不进行操作：保持 PWMxA 和 PWMxB 输出电平不变。在这种状态下，仍然可以触发中断。

输出上的操作通过寄存器 MCPWM_GENn_A_REG 和 MCPWM_GENn_B_REG 配置。每一次输出的操作都独立配置。此外，可以根据事件，灵活地在输出上执行不同操作。表 31.3-2 中列举的任何事件都可以作用于 PWMxA 或 PWMxB 输出上。关于生成器 0、1 或 2 的寄存器信息，请参考章节 31.5。

常见配置的波形

在下面各种波形的配置中，period 指 PWM 操作器所选 PWM 定时器的周期寄存器的配置值（即 `MCPWM_TIMERn_PERIOD` ($x = 0, 1, 2$))，配置方法参考章节 31.3.2。A 和 B 指硬件使用的时间戳寄存器 A 和 B，配置方法参考章节 31.3.3.1。UTEA/B、UTEZ/P、DTEA/B、DTEZ/P、DTO/1 和 UTO/1 指 PWM 生成器的各种定时事件，其定义参考章节 31.3.3.1。向上的箭头、向下的箭头、T 标识分别代表定时事件发生时将 PWM 波形置为高电平、置为低电平、取反，配置方法参考章节 31.3.3.1。PWMxA 和 PWMxB 是 PWM 生成器 x 的输出信号。

图 31.3-13 为 PWM 定时器在递增递减循环计数时生成的对称 PWM 波形。该模式下的直流 0%-100% 调制可由以下公式获得：

$$Duty = (Period - A) \div Period$$

如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递增，则 PWM 输出被拉高。如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递减，则 PWM 输出被拉低。

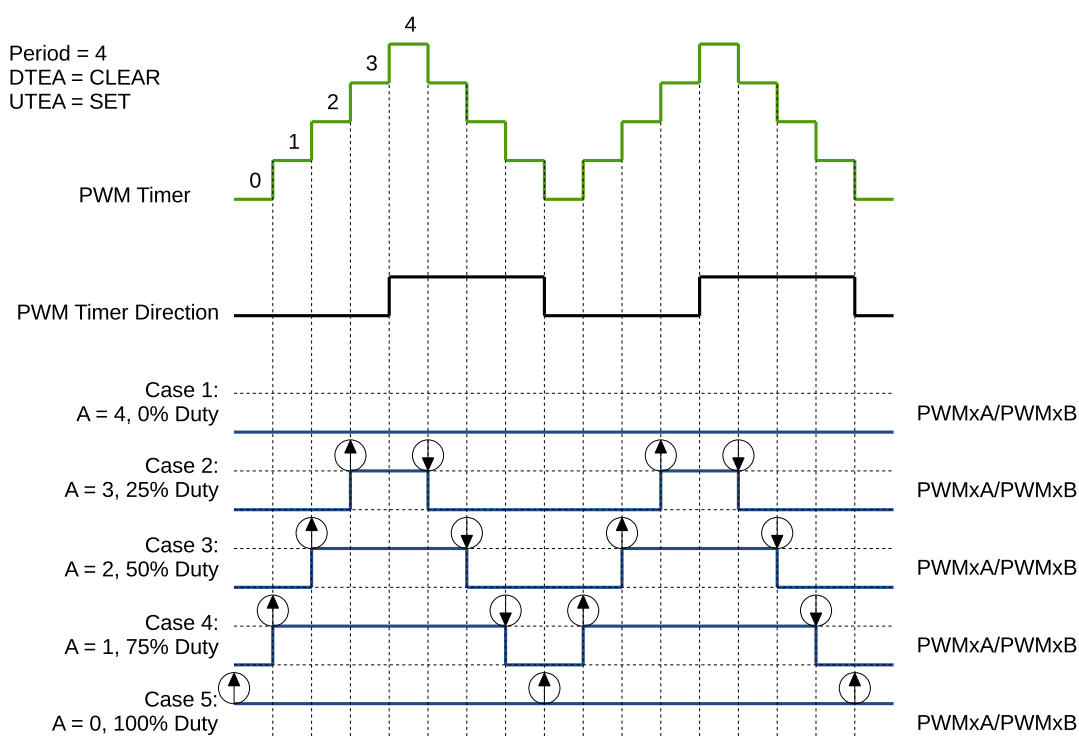


图 31.3-13. 递增递减模式下的对称波形

图 31.3-14 至图 31.3-17 的 PWM 波形描述了常见的 PWM 生成器配置。

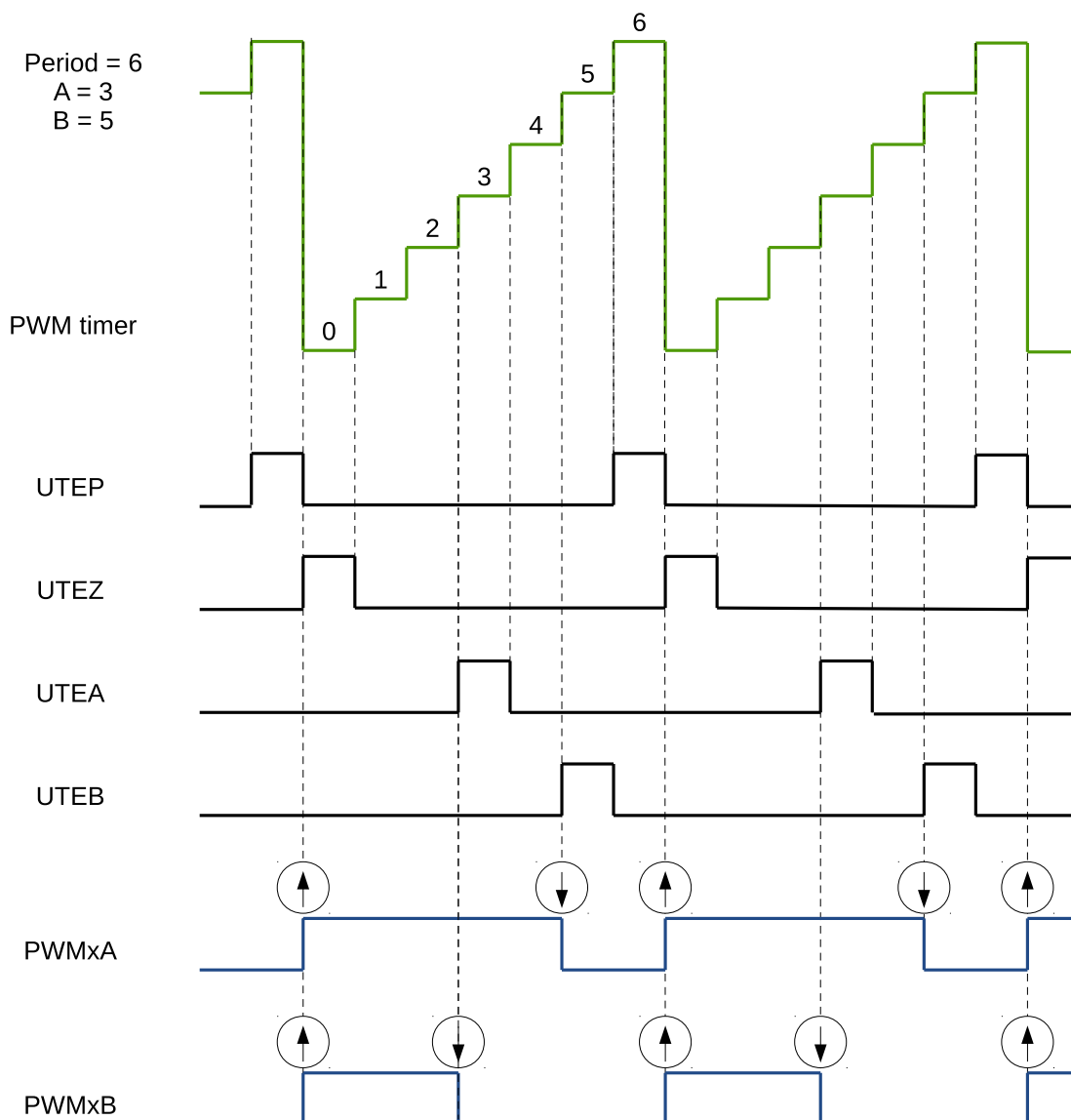


图 31.3-14. 递增计数模式，单边不对称波形，PWMxA 和 PWMxB 独立调制—高电平

PWMxA 的占空比调制由 B 设置，高电平有效，与 B 成正比。

PWMxB 的占空比调制由 A 设置，高电平有效，与 A 成正比。

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

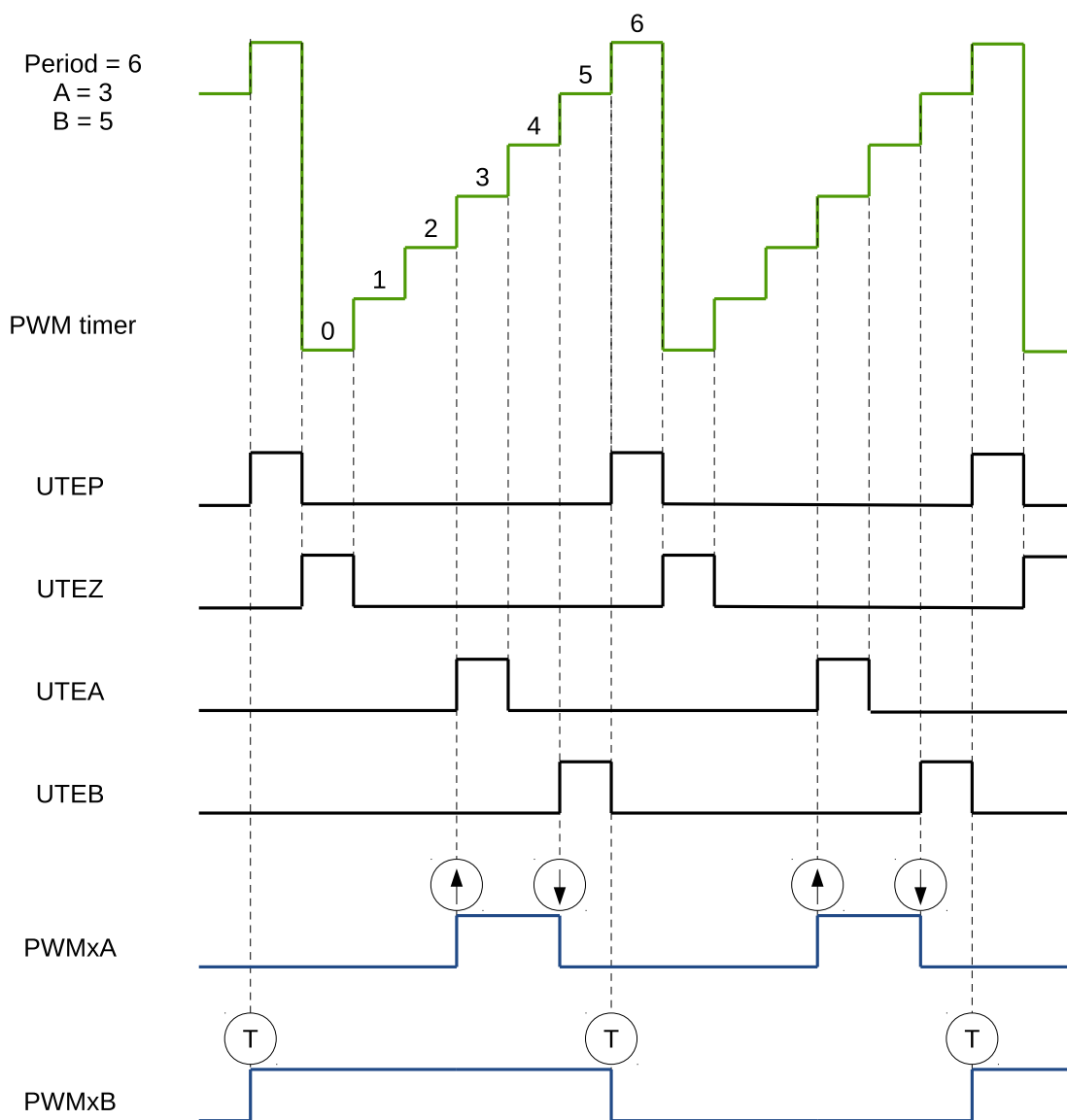


图 31.3-15. 递增计数模式，脉冲位置不对称波形，PWMxA 独立调制

脉冲可以在 PWM 波形内（零至周期值之间）的任何地方生成。

PWMxA 占空比与 $(B - A)$ 成正比。

$$Period = (MCPWM\_TIMERn\_PERIOD + 1) \times T_{PT\_clk}$$

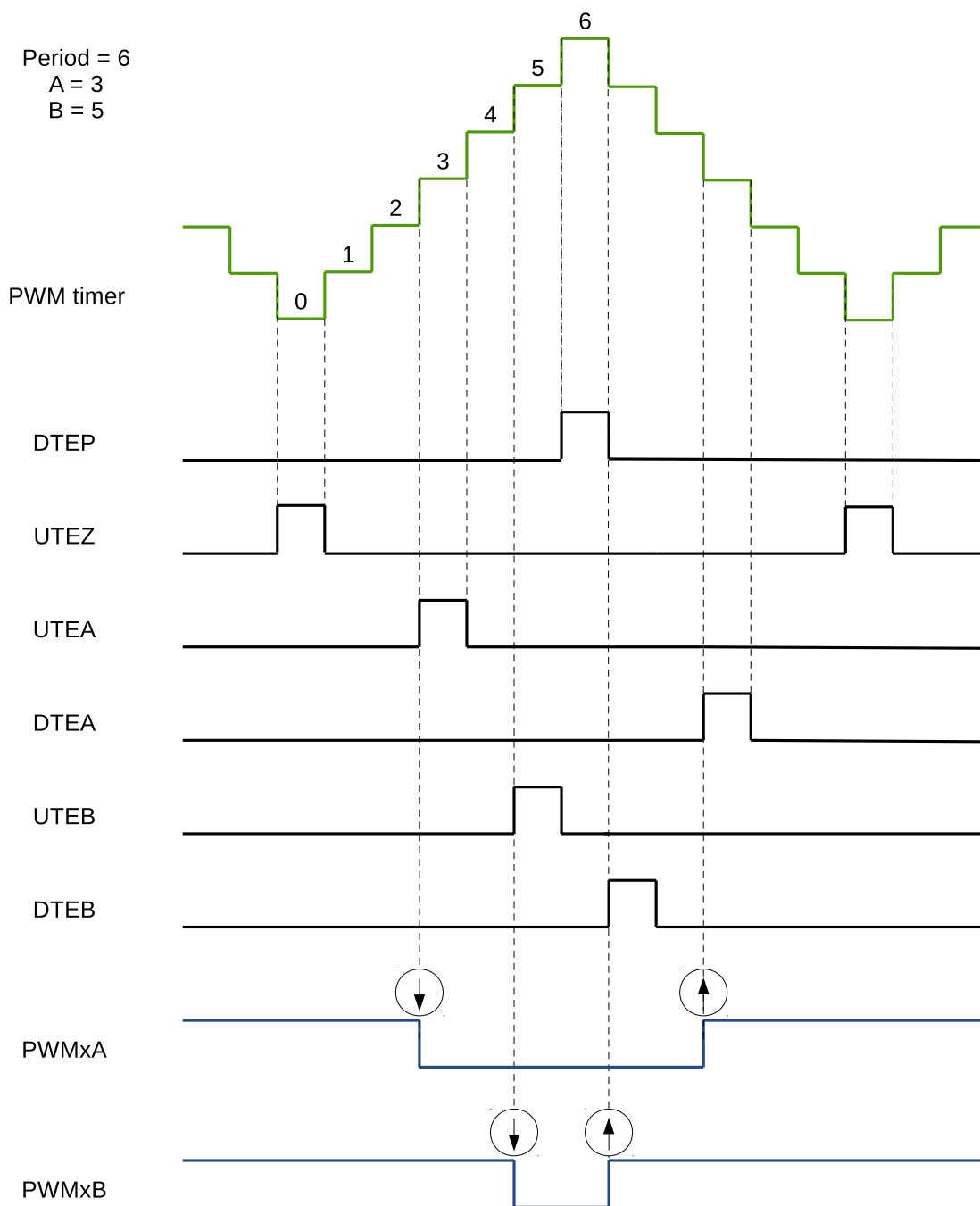


图 31.3-16. 递增递减循环计数模式，双沿对称波形，在 PWMxA 和 PWMxB 上独立调制—高电平有效

PWMxA 的占空比调制由 A 设置，高电平有效，与 A 成正比。

PWMxB 的占空比调制由 B 设置，高电平有效，与 B 成正比。

输出 PWMxA 和 PWMxB 可驱动不同开关。

$$Period = (2 \times MCPWM\_TIMERn\_PERIOD) \times T_{PT\_clk}$$

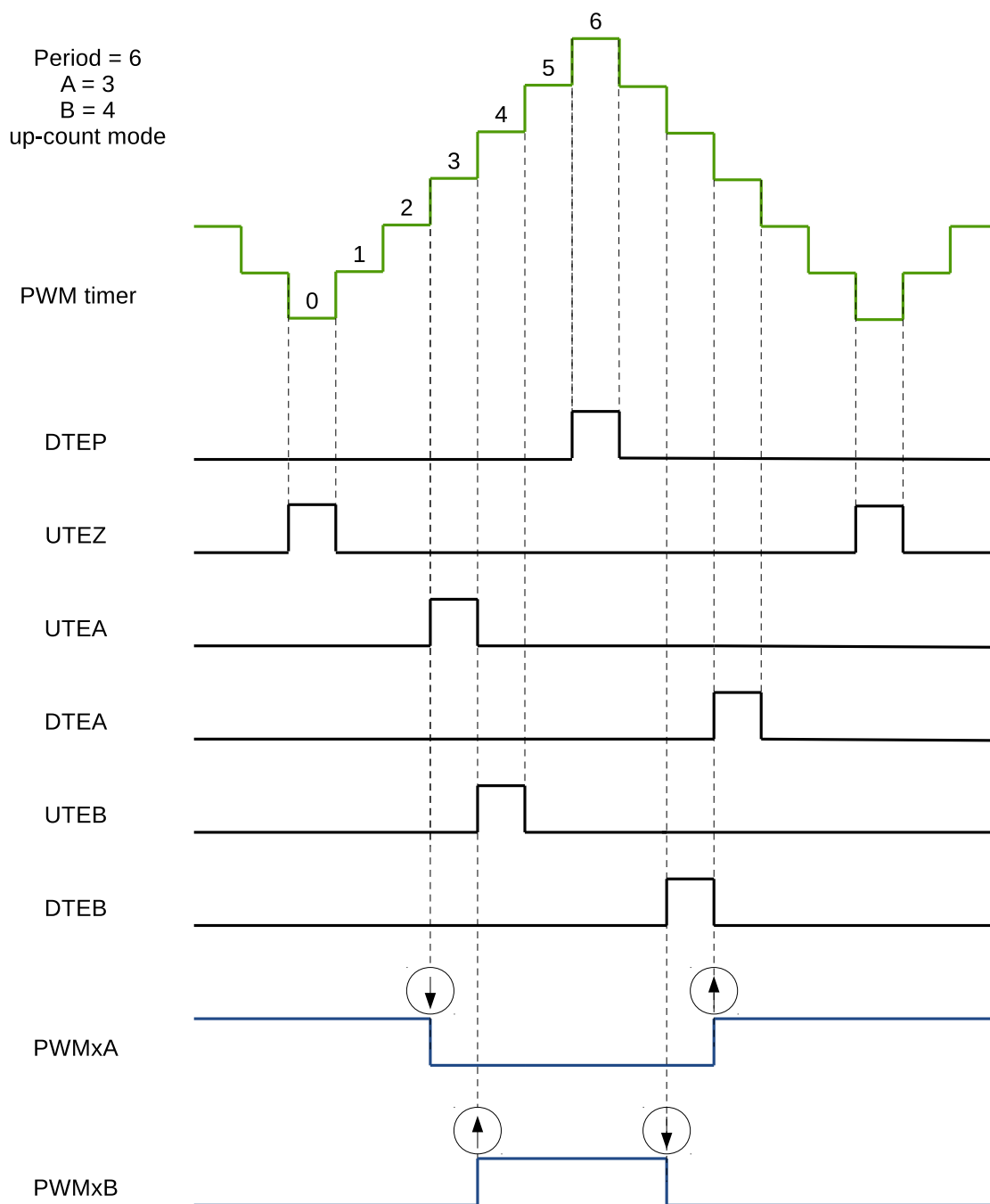


图 31.3-17. 递增递减循环计数模式，双沿对称波形，在 PWMxA 和 PWMxB 上独立调制—互补

PWMxA 的占空比调制由 A 设置，高电平有效，与 A 成正比。

PWMxB 的占空比调制由 B 设置，高电平有效，与 B 成正比。

PWMxA/B 输出可驱动上/下（互补）开关。

死区 = B - A，边沿位置完全可由软件配置。必要时，可使用死区生成器模块设置其他边沿延迟方式。

$$Period = (2 \times MCPWM\_TIMER\_PERIOD) \times T_{PT\_clk}$$

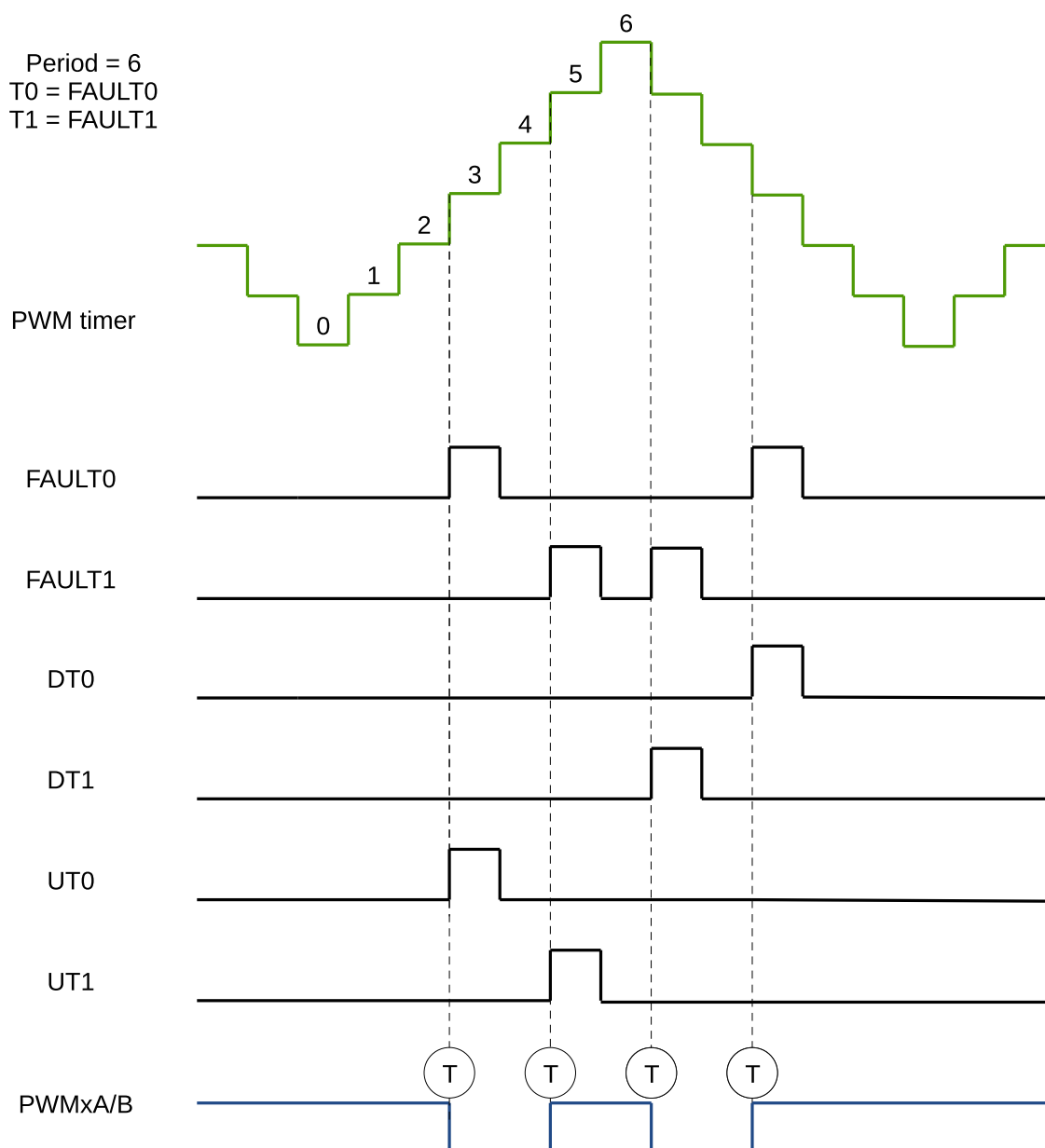


图 31.3-18. 递增递减循环计数模式，错误或同步事件，在 PWMxA 和 PWMxB 上相同调制

图 31.3-18 为当 UT0/1 和 DT0/1 事件产生时的波形。在本例中，T0 选择 FAULT0，T1 选择 FAULT1。T0 和 T1 所选择的事件可以独立配置，这些事件可以是 FAULT0、FAULT1、FAULT2 或同步事件。具体配置方法见章节 31.3.3.1。

软件强制事件

在 PWM 生成器内有 2 种软件强制事件：

- 非连续即时 (NCI) 软件强制事件：当由软件触发时，这类事件在 PWM 输出上立即生效。强制是不连续的，意味着下一个激活的定时事件能够改变 PWM 输出。
- 连续 (CNTU) 软件强制事件：这类事件是连续的。强制 PWM 将持续输出，直到通过软件释放。事件触发器可配置。这类事件可配置为定时或即时发生。

对于 NCI 软件强制事件，向寄存器 MCPWM_GENn_A_NCIFORCE 或 MCPWM_GENn_B_NCIFORCE 写 1 再写

0，触发 PWMxA 或 PWMxB 的 NCI 软件强制事件。配置寄存器 `MCPWM_GENn_A_NCIFORCE_MODE` 或 `MCPWM_GENn_B_NCIFORCE_MODE` 指定当 NCI 软件强制事件发生时，对 PWMxA 或 PWMxB 波形的操作，配置为 0 或 3 表示无操作，配置为 1 表示置为低电平，配置为 2 表示置为高电平。

对于 CNTU 软件强制事件，配置寄存器 `MCPWM_GENn_CNTUFORCE_UPMETHOD`，指定 CNTU 软件强制事件的触发方式，所有 bit 配置为 0，表示立即触发 CNTU 软件强制事件，bit 0 配置为 1，表示 TEZ 事件触发 CNTU 软件强制事件，bit 1 配置为 1，表示 TEP 事件触发 CNTU 软件强制事件，bit 2 配置为 1，表示 TEA 事件触发 CNTU 软件强制事件，bit 3 配置为 1，表示 TEB 事件触发 CNTU 软件强制事件，bit 4 配置为 1，表示 PWM 生成器所选的 PWM 定时器同步时触发 CNTU 软件强制事件，bit 5 配置为 1，表示不触发 CNTU 软件强制事件。配置寄存器 `MCPWM_GENn_A_CNTUFORCE_MODE` 或 `MCPWM_GENn_B_CNTUFORCE_MODE` 指定当 CNTU 软件强制事件发生时，对 PWMxA 或 PWMxB 波形的操作，配置为 0 或 3 表示无操作，配置为 1 表示置为低电平，配置为 2 表示置为高电平。

图 31.3-19 为 NCI 软件强制事件的一种波形。NCI 用于单独强制 PWMxA 输出为低电平，PWMxB 不受强制事件控制。

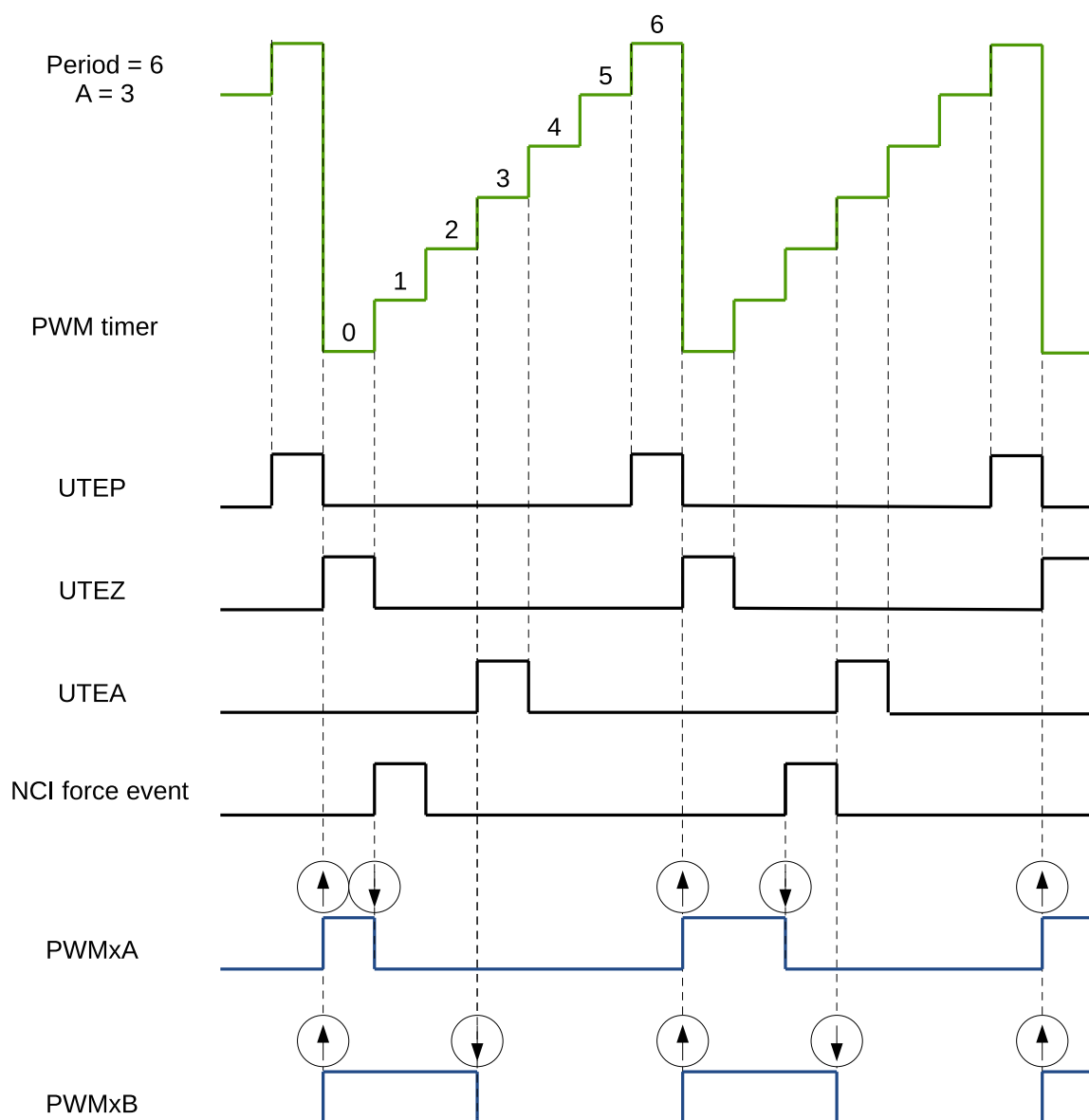


图 31.3-19. NCI 在 PWMxA 输出上软件强制事件示例

图 31.3-20 为 CNTU 软件强制事件的波形。TEZ 事件被选为 CNTU 软件强制事件的触发器。CNTU 用于单独强制 PWMxB 输出为低电平, 但 PWMxA 不受强制。

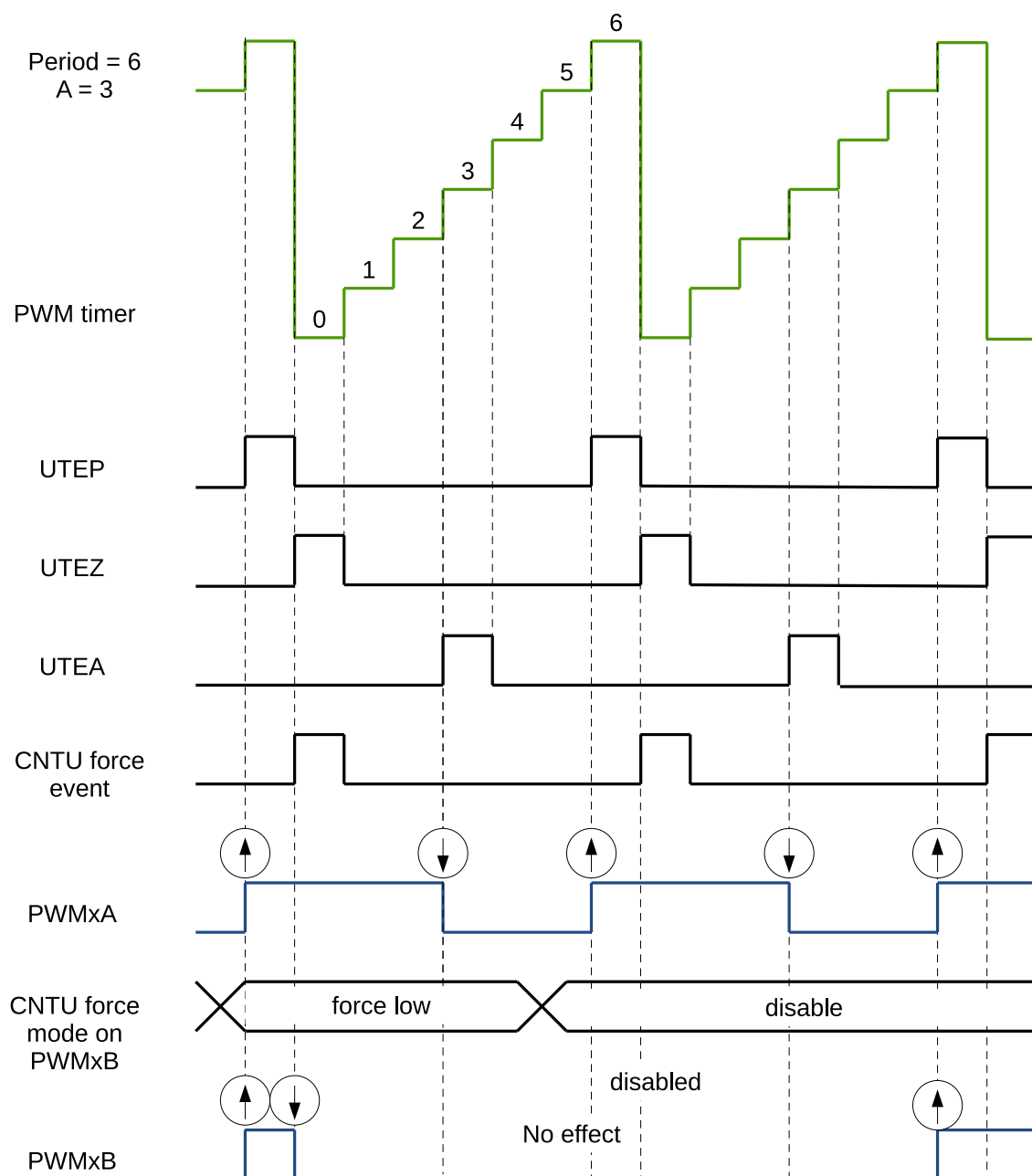


图 31.3-20. CNTU 在 PWMxB 输出上软件强制事件示例

31.3.3.2 死区生成器模块

死区生成器模块作用

章节 31.3.3.1 介绍了在 PWMxA 和 PWMxB 输出上生成信号的几种方式/事件，包括信号边沿的特定位置。通过改变信号之间的边沿位置以及设置信号的占空比，可获得所需的死区。此外，也可以使用专门的死区生成器模块控制死区。

死区生成器模块的主要功能如下：

- 可以将输入的 PWMxA 或 PWMxB 信号作为公共源来生成具有死区的 PWMxA 和 PWMxB 信号对输出
- 通过在信号边沿增加延迟来生成死区：
 - 上升沿延迟 (RED)
 - 下降沿延迟 (FED)
- 可以生成多种 PWM 波形，经典的死区配置为：
 - 高电平有效互补 (AHC)
 - 低电平有效互补 (ALC)
 - 高电平有效 (AH)
 - 低电平有效 (AL)
- 如果死区已经在生成器中配置，死区发生器也可以不进行配置。

死区模块生成器影子寄存器

延迟寄存器 RED 和 FED 的影子寄存器为 MCPWM_DT n _RED_CFG_REG 和 MCPWM_DT n _FED_CFG_REG。

MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。

MCPWM_DT n _RED_CFG_REG 的更新方式寄存器为 MCPWM_DB n _RED_UPMETHOD；

MCPWM_DT n _FED_CFG_REG 的更新方式寄存器为 MCPWM_DB n _FED_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。寄存器描述详见章节 31.3.2.3。

死区生成器模块的操作要点

图 31.3-21 描述了创建死区模块的开关拓扑。

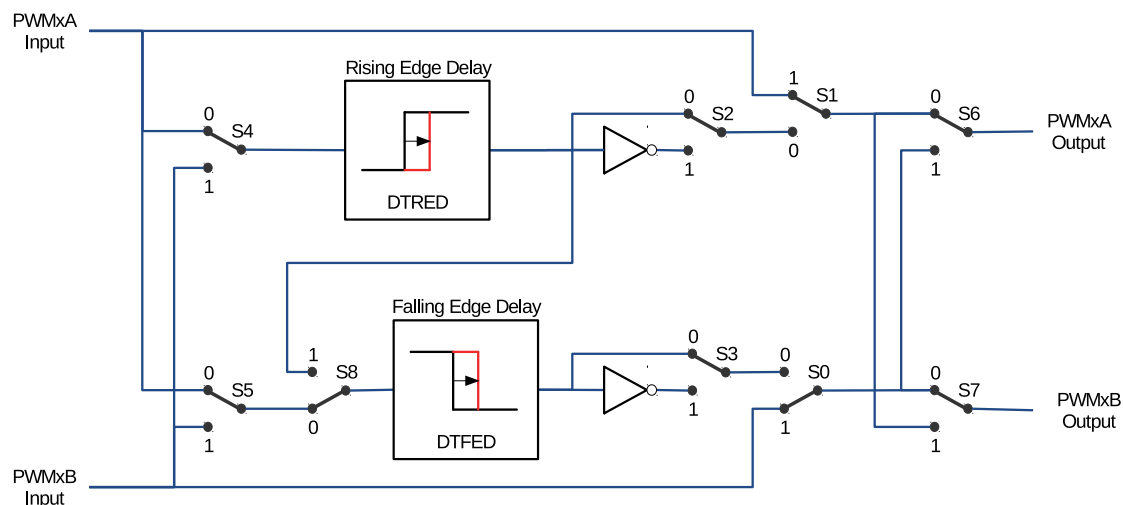


图 31.3-21. 死区模块的开关拓扑

上图中的 S0 - S8 开关由 MCPWM_DT n _CFG_REG 寄存器中的字段控制，详细信息见表 31.3-5。

表 31.3-5. 控制死区时间生成器开关的字段

开关	字段
S0	MCPWM_DB n _B_OUTBYPASS
S1	MCPWM_DB n _A_OUTBYPASS
S2	MCPWM_DB n _RED_OUTINVERT
S3	MCPWM_DB n _FED_OUTINVERT
S4	MCPWM_DB n _RED_INSEL
S5	MCPWM_DB n _FED_INSEL
S6	MCPWM_DB n _A_OUTSWAP
S7	MCPWM_DB n _B_OUTSWAP
S8	MCPWM_DB n _DEB_MODE

所有开关组合皆受支持，但不是所有的开关模式都是典型的使用模式。表 31.3-6 列举了一些典型的死区配置。在这些配置中，S4 和 S5 的开关位置将 PWMxA 设置为下降沿和上升沿延迟的公共源。表 31.3-6 中的模式可分为以下几类：

- **模式 1：绕过下降沿 (FED) 和上升沿 (RED) 的延迟**

该模式下，死区模块被关闭。PWMxA 和 PWMxB 信号的波形无变化。

- **模式 2-5：经典死区极性设置**

这些模式为典型极性配置，涵盖工业电源栅极驱动器中的高 / 低电平有效模式。图 31.3-22 至 31.3-25 为典型波形。

- **模式 6 和 7：绕过下降沿 (FED) 或上升沿 (RED) 的延迟**

这两种模式下，绕过上升沿延迟 (RED) 或下降沿延迟 (FED)。因此，无需使用对应延迟。

表 31.3-6. 死区生成器的典型操作模式

模式	描述	S0	S1	S2	S3
1	PWMxA 和 PWMxB 波形无变化	1	1	X	X
2	高电平有效互补 (AHC), 参见图 31.3-22	0	0	0	1
3	低电平有效互补 (ALC), 参见图 31.3-23	0	0	1	0
4	高电平有效 (AH), 参见图 31.3-24	0	0	0	0
5	低电平有效 (AL), 参见图 31.3-25	0	0	1	1
6	PWMxA 输出 = PWMxA 输入 (无延迟) PWMxB 输出 = PWMxA 输入, 下降沿延迟	0	1	0 或 1	0 或 1
7	PWMxA 输出 = PWMxA 输入, 上升沿延迟 PWMxB 输出 = PWMxB 输入 (无延迟)	1	0	0 或 1	0 或 1

说明:
以上所有模式中, S4 - S8 的开关位置都置 0。

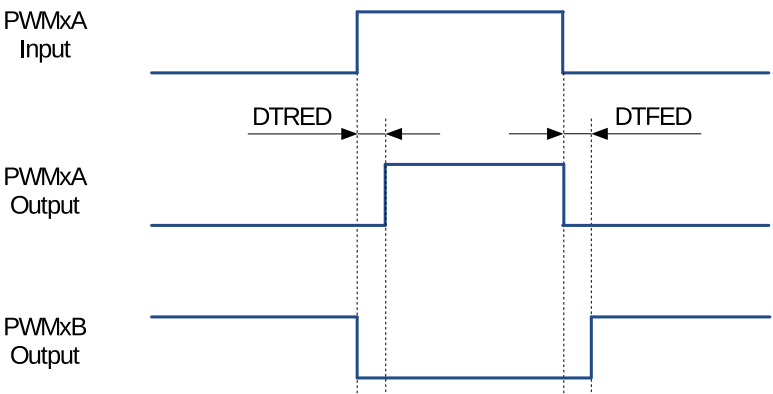


图 31.3-22. 高电平有效互补 (AHC) 死区波形

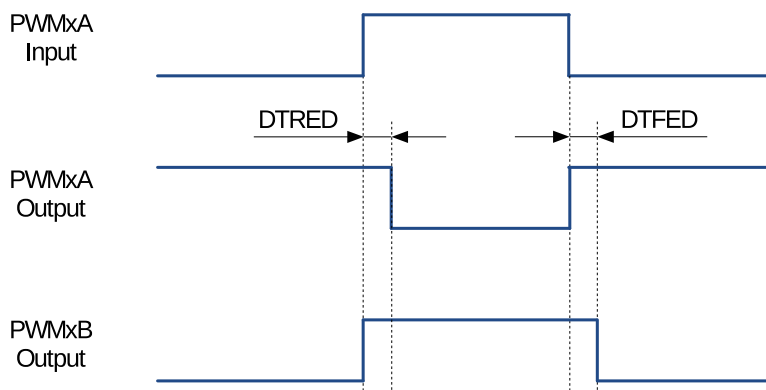


图 31.3-23. 低电平有效互补 (ALC) 死区波形

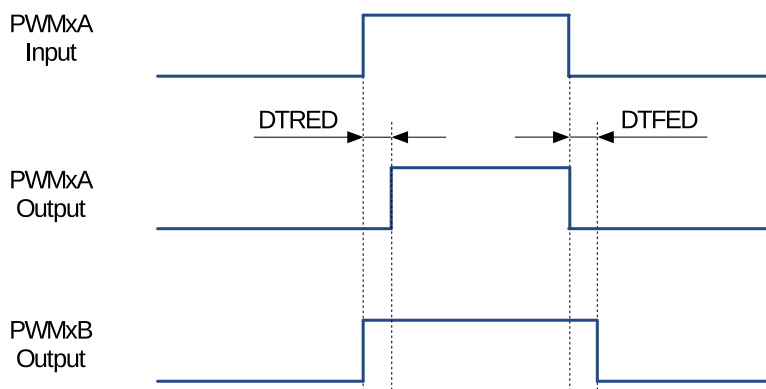


图 31.3-24. 高电平有效 (AH) 死区波形

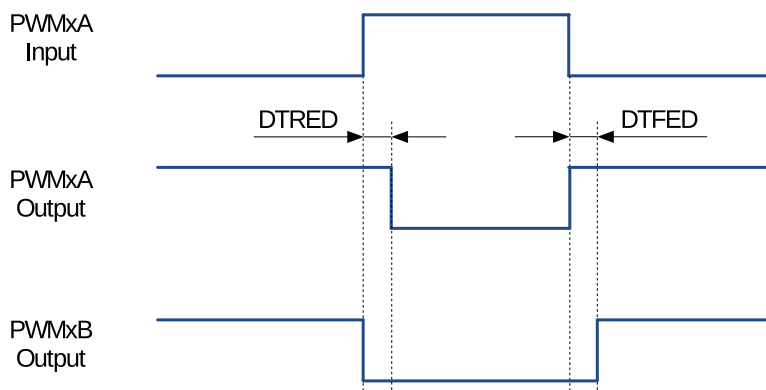


图 31.3-25. 低电平有效 (AL) 死区波形

上升沿延迟 (RED) 和下降沿延迟 (FED) 可分别设置。延迟的值通过 16 位寄存器 `MCPWM_DT $x$ _RED` 和 `MCPWM_DT $x$ _FED` 配置。寄存器值表示一个信号边沿可以延迟的 `DT_clk` 时钟周期值。`DT_clk` 可通过寄存器 `MCPWM_DB $n$ _CLK_SEL` 从 `PWM_clk` 或 `PT_clk` 中选择。

通过以下公式计算下降沿延迟 (FED) 和上升沿延迟 (RED) 的值:

$$FED = MCPWM\_DTn\_FED \times T_{DT\_clk}$$

$$RED = MCPWM\_DTn\_RED \times T_{DT\_clk}$$

31.3.3.3 PWM 载波模块

将 PWM 输出耦合到电机驱动器可能需要使用变压器隔离。变压器只提供交流信号，而 PWM 信号的占空比可能在 0% 到 100% 之间变化。PWM 载波模块可以通过使用高频载波对其进行调制，将该信号传递给变压器。

功能概述

此模块的以下关键功能可配置:

- 载波频率
- 第一个脉冲的脉宽
- 第二个以及之后的脉冲的占空比
- 开启/关闭载波

操作要点

PWM 载波时钟 (`PC_clk`) 来自于 `PWM_CLK`。通过寄存器 `MCPWM_CARRIER $n$ _CFG_REG` 的 `MCPWM_CHOPPER $n$ _PRESCALE` 和 `MCPWM_CHOPPER $n$ _DUTY` 位配置频率和占空比。一次性脉冲的功能在于提供高能量脉冲以接通电源开关。随后的脉冲用于保持上电的状态。一次性脉冲宽度可通过 `MCPWM_CHOPPER $n$ _OSHTWTH` 位进行配置。通过 `MCPWM_CHOPPER $n$ _EN` 位来使能/禁用载波模块。

载波示例

图 31.3-26 描述了载波叠加在原始 PWM 脉冲上的示例波形。图中未显示第一个脉冲和占空比控制，相关详细信息将在后两节中介绍。

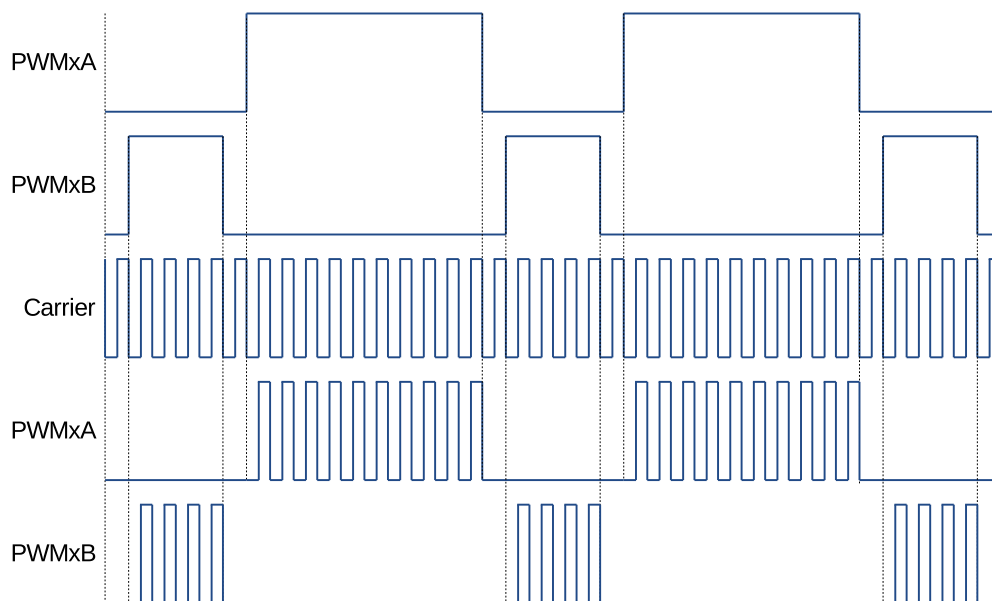


图 31.3-26. PWM 载波操作的波形示例

第一个脉冲

第一个脉冲的宽度可配置，其值有 16 种可能，可通过下公式计算：

$$T_{1stpulse} = T_{PWM\_clk} \times 8 \times (MCPWM\_CARRIERn\_PRESCALE + 1) \times (MCPWM\_CARRIERn\_OSHTWTH + 1)$$

其中：

- T_{PWM_clk} 为 PWM 时钟周期 (PWM_clk)
- $(MCPWM\_CARRIERn\_OSHTWTH + 1)$ 为一次性脉冲宽度值 (取值范围：1-16)
- $(MCPWM\_CARRIERn\_PRESCALE + 1)$ PWM 载波时钟 (PC_clk) 预分频值

图 31.3-27 展示了第一个脉冲和之后持续的脉冲。

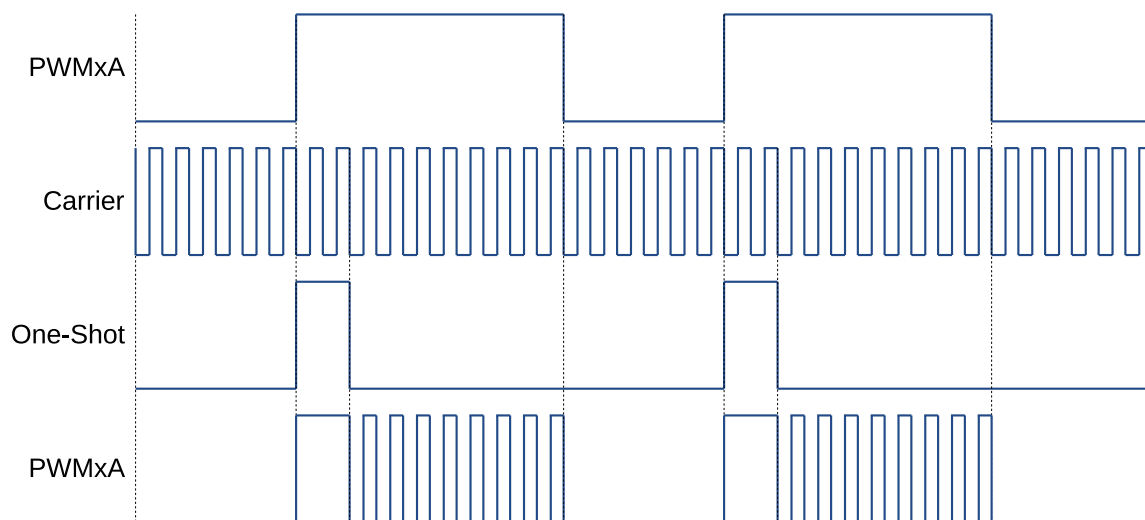


图 31.3-27. 载波模块的第一个脉冲和之后持续的脉冲示例

占空比控制

在发出第一个一次性脉冲之后，根据载波频率调制剩余的 PWM 信号。用户可配置该信号的占空比。在一定情况下，调整占空比可使信号通过隔离变压器后仍然可以开启或关闭电动机驱动器，改变电机旋转速度和方向。

占空比通过 `MCPWM_CHOPPERn_DUTY` 或寄存器 `MCPWM_CARRIERn_CFG_REG` 的 [7:5] 位设置，其值有 7 种可能性。

占空比的值可通过以下方式计算：

$$Duty = MCPWM\_CARRIER_n\_DUTY \div 8$$

图 31.3-28 为所有 7 种占空比设置。

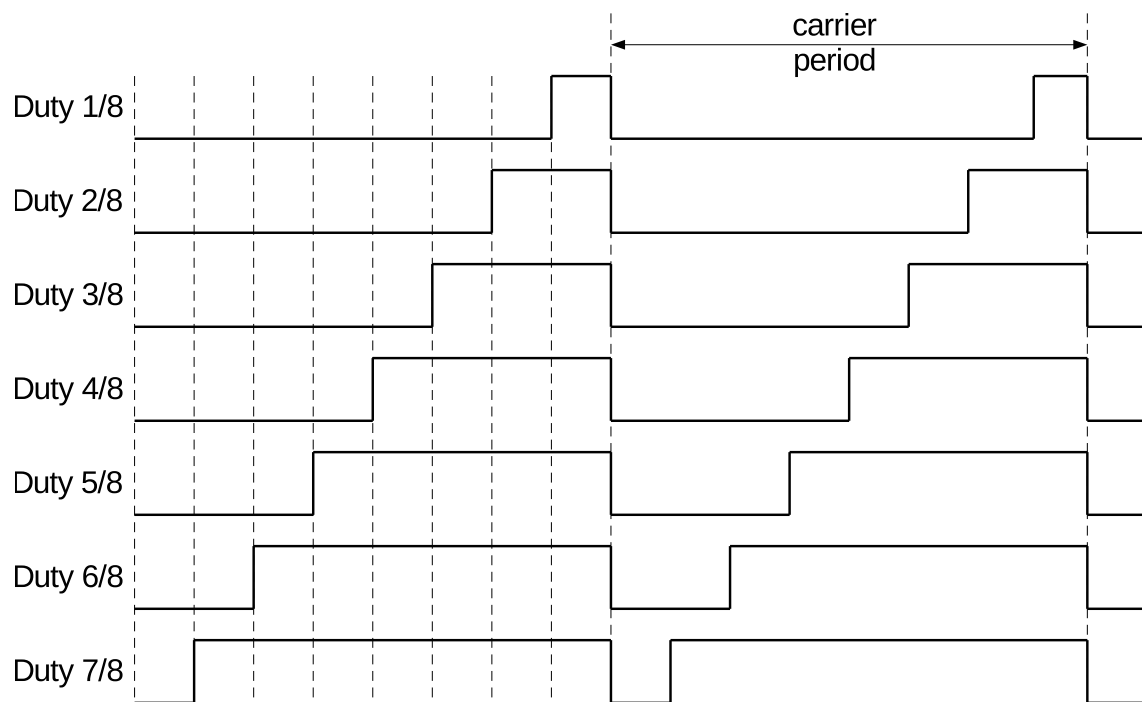


图 31.3-28. PWM 载波模块中持续脉冲的 7 种占空比设置

31.3.3.4 故障检测模块

每个 MCPWM 外设都连接来自 GPIO 矩阵的 3 个故障信号 (FAULT0、FAULT1 和 FAULT2)。这些信号用于指示外部故障状况，并且可由故障检测模块预处理后生成故障事件。故障事件通过执行用户代码，针对特定故障调整 MCPWM 输出。

故障检测模块功能

故障检测模块的主要功能为：

- 在检测到故障时强制 PWMxA 和 PWMxB 输出信号进入以下状态之一：
 - 高
 - 低
 - 取反
 - 无
- 在检测到过电流/短路时执行一次性跳闸 (OST)。
- 逐周期跳闸 (CBC) 以提供限流操作。
- 每个故障信号单独分配一次性或逐周期操作。
- 每个故障输入都生成中断。
- 支持软件强制跳闸。
- 根据需要开启或关闭模块功能。

操作与配置要点

本节介绍故障处理模块的操作要点和配置选项。

来自管脚的故障信号在 GPIO 矩阵中采样和同步。为了保证故障脉冲成功采样，每个脉冲持续时间必须至少为 2 个 APB 时钟周期。故障检测模块使用 PWM_clk 对故障信号进行采样，因此，来自 GPIO 矩阵的故障脉冲持续时间必须至少为 1 个 PWM_clk 周期。换句话说，无论 APB 时钟周期和 PWM_clk 周期的大小关系如何，管脚上的故障信号脉冲的宽度必须至少等于两个 APB 时钟周期与一个 PWM_clk 周期的和。

故障处理器模块可以使用故障信号 FAULT0 至 FAULT2 中的高电平或低电平来生成故障事件 fault_event0 至 fault_event2。每个故障事件可以单独配置为进行 CBC 操作、OST 操作或无操作。

- **逐周期 (CBC) 操作：**

触发 CBC 操作时，PWMxA 和 PWMxB 的状态立即根据字段 MCPWM_TZn_A_CBC_U/D 和 MCPWM_TZn_B_CBC_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 CBC 操作中断。通过状态字段 MCPWM_TZn_CBC_ON 开启或关闭 CBC 操作。在没有故障事件时，将在指定时间点，即发生 D/UTEP 或 D/UTEZ 事件时，清除 PWMxA/B 上的 CBC 操作。由字段 MCPWM_TZn_CBCPULSE 决定 PWMxA 和 PWMxB 恢复正常的事件。因此，在此模式下，每个 PWM 循环后，CBC 操作都将清除或刷新。

- **一次性 (OST) 操作：**

触发 OST 操作时，PWMxA 和 PWMxB 的状态立即根据字段 MCPWM_TZn_A_OST_U/D 和 MCPWM_TZn_B_OST_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 OST 操作中断。通过状态字段 MCPWM_TZn_OST_ON 开启或关闭 OST 操作。在没有故障事件时，PWMxA/B 上的 OST 操作无法自动清除。须通过将 MCPWM_TZn_CLR_OST 位置 1 来清除一次性操作。

31.3.4 捕获模块

31.3.4.1 介绍

捕获模块包含 3 个完整的捕获通道。通道输入信号 CAPO、CAP1 和 CAP2 来自于 GPIO 矩阵。由于 GPIO 矩阵的灵活性，CAPO、CAP1 和 CAP2 可以通过任一管脚输入配置。多个捕获通道可以来自同一管脚输入，每个通道的预分频可以分别设置。同时，每个捕获通道也可以来自不同的管脚输入。因此，可以通过后台硬件用多种方式处理捕获信号，而不直接由 CPU 处理。

捕捉模块有以下独立资源：

- 一个 32 位定时器（计数器），可与 PWM 定时器、另一个模块或软件同步。
- 三个捕获通道，每个通道配有一个 32 位时间戳和一个捕获预分频器。
- 任何捕获通道的边沿极性（上升/下降沿）可独立选择。
- 输入捕获信号预分频（分频取值范围：1 ~ 256）。
- 三个捕获事件都有中断功能。

31.3.4.2 捕获定时器

捕获定时器是一个 32 位计数器，使能时不断递增计数。将 MCPWM_CAP_TIMER_EN 置 1 使能捕获寄存器。其操作时钟源为 MCPWM 模块的主时钟。配置 MCPWM_CAP_SYNCI_EN 后，发生同步事件时，存储在

MCPWM_CAP_TIMER_PHASE_REG 寄存器中的相位将被加载至计数器中。同步事件可来自 PWM 定时器同步输出或 PWM 模块同步输入，通过 MCPWM_CAP_SYNCI_SEL 进行配置。也可将 MCPWM_CAP_SYNC_SW 置 1 生成同步事件。捕获定时器为所有三个捕获通道提供定时参考。

31.3.4.3 捕获通道

必要时，到达捕获通道的捕获信号可先被反相，然后预分频。每个捕获通道都有一个预分频寄存器 MCPWM_CAP n _PRESCALE。最后，预处理后的捕获信号的指定边沿将触发捕获事件。将 MCPWM_CAP n _EN 置 1 使能捕获通道。捕获事件将在 MCPWM_CAP n _MODE 配置的时间发生。在捕获事件发生时，捕获定时器的值存储在时间戳寄存器 MCPWM_CAP_CH n _REG 中。捕获事件中的不同捕获通道可生成不同的中断。触发捕获事件的边沿储存在寄存器 MCPWM_CAP n _EDGE 中。捕获事件可通过将 MCPWM_CAP n _SW 置 1 由软件强制发生。

31.3.5 ETM 模块

31.3.5.1 介绍

在 ESP32-C5 中，MCPWM 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 MCPWM 的 ETM 任务，或者通过 MCPWM 的 ETM 事件触发任意外设的 ETM 任务。MCPWM 的捕获模块、故障检测模块、三个定时器以及三个操作器都可以独立产生事件或响应任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与 MCPWM 相关的 ETM 任务和 ETM 事件。

31.3.5.2 MCPWM 可产生的 ETM 事件

将使能字段置 1 后，满足对应生成条件时，将产生对应 ETM 事件，详见表 31.3-7：

表 31.3-7. MCPWM 可产生的 ETM 事件

使能字段	生成条件	产生事件
MCPWM_EVT_CAP n _EN	发生 CAP n 捕获事件	MCPWM0_EVT_CAP n
MCPWM_EVT_TZ n _OST_EN	PWM 操作器 n 执行一次性跳闸 (OST) 操作	MCPWM0_EVT_TZ n _OST
MCPWM_EVT_TZ n _CBC_EN	PWM 操作器 n 执行逐周期跳闸 (CBC) 操作	MCPWM0_EVT_TZ n _CBC
MCPWM_EVT_F n _CLR_EN	清除故障事件 fault_event n	MCPWM0_EVT_F n _CLR
MCPWM_EVT_F n _EN	产生故障事件 fault_event n	MCPWM0_EVT_F n
MCPWM_EVT_OP n _TEB_EN	PWM 操作器 n 选择的定时器计数值等于时间戳 B 的值	MCPWM0_EVT_OP n _TEB
MCPWM_EVT_OP n _TEA_EN	PWM 操作器 n 选择的定时器计数值等于时间戳 A 的值 ¹	MCPWM0_EVT_OP n _TEA
MCPWM_EVT_OP n _TEE1_EN	PWM 操作器 n 选择的定时器计数值等于寄存器 MCPWM_OP n _TSTMP_E1_REG 的值	MCPWM0_EVT_OP n _TEE1
MCPWM_EVT_OP n _TEE2_EN	PWM 操作器 n 选择的定时器计数值等于寄存器 MCPWM_OP n _TSTMP_E2_REG 的值	MCPWM0_EVT_OP n _TEE2
MCPWM_EVT_TIMER n _TEP_EN	定时器 n 的计数值等于周期值 MCPWM_TIMER n _PERIOD	MCPWM0_EVT_TIMER n _TEP
MCPWM_EVT_TIMER n _TEZ_EN	定时器 n 的计数值等于 0	MCPWM0_EVT_TIMER n _TEZ
MCPWM_EVT_TIMER n _STOP_EN	定时器 n 停止计数	MCPWM0_EVT_TIMER n _STOP

¹ 有关时间戳 A 和时间戳 B 的描述，详见章节 31.3.3.1。

31.3.5.3 MCPWM 可接收的 ETM 任务

将使能字段置 1 后，接收对应有效任务后，将产生对应的响应操作，详见表 31.3-8：

表 31.3-8. MCPWM 可接收的 ETM 任务

使能字段	接收的有效任务	响应操作
MCPWM_TASK_CAP n _EN	MCPWMO_TASK_CAP n	CAP n 通道进行捕获操作
MCPWM_TASK_CLR n _OST_EN	MCPWMO_TASK_CLR n _OST	清除 PWM 操作器 n 的一次性跳闸 (OST) 操作
MCPWM_TASK_TZ n _OST_EN	MCPWMO_TASK_TZ n _OST	PWM 操作器 n 执行一次性跳闸 (OST) 操作
MCPWM_TASK_TIMER n _PERIOD_UP_EN	MCPWMO_TASK_TIMER n _PERIOD_UP	定时器 n 的周期更新为周期寄存器 MCPWM_TIMER n _PERIOD 配置的值
MCPWM_TASK_TIMER n _SYNC_EN	MCPWMO_TASK_TIMER n _SYN	定时器 n 执行一次同步操作
MCPWM_TASK_GEN_STOP_EN	MCPWMO_TASK_GEN_STOP	所有定时器停止计数，所有的 PWM 操作器输出的 PWM 信号保持不变
MCPWM_TASK_CMPR n _B_UP_EN	MCPWMO_TASK_CMPR n _B_UP	将 PWM 操作器 n 的时间戳 B 更新为时间戳 B 影子寄存器 MCPWM_GEN n _B 的值
MCPWM_TASK_CMPR n _A_UP_EN	MCPWMO_TASK_CMPR n _A_UP	将 PWM 操作器 n 的时间戳 A 更新为时间戳 A 影子寄存器 MCPWM_GEN n _A 的值

31.4 中断

ESP32-C5 的 MCPWMO 会分别生成 PWM0_INTR 中断信号，并发送给中断矩阵。

PWM0_INTR 中断信号由以下内部中断源生成：

- MCPWM_TIMER x _STOP_INT：定时器 x 停止后触发。
- MCPWM_TIMER x _TEZ_INT：由 PWM 定时器 x TEZ 事件触发。
- MCPWM_TIMER x _TEP_INT：由 PWM 定时器 x TEP 事件触发。
- MCPWM_FAULT x _INT：fault_event x 开始后触发。
- MCPWM_FAULT x _CLR_INT：fault_event x 结束后触发。
- MCPWM_CMPR x _TEA_INT：由 PWM 操作器 x TEA 事件触发。
- MCPWM_CMPR x _TEB_INT：由 PWM 操作器 x TEB 事件触发。
- MCPWM_TZ x _CBC_INT：由 PWM x 逐周期操作触发。
- MCPWM_TZ x _OST_INT：由 PWM x 一次性操作触发。
- MCPWM_CAP x _INT：由信道 x 上捕获事件触发。

上述中断源仅在对应的中断寄存器使能字段 `interrupt_source_name_ENA` 置 1 时才能触发，将 `interrupt_source_name_CLR` 字段置 1 即清除对应中断。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 *[to be added later]* > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [31.5 寄存器列表](#)。

31.5 寄存器列表

本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
预分频器配置			
MCPWM_CLK_CFG_REG	配置预分频器	0x0000	R/W
MCPWM 定时器 0 配置与状态			
MCPWM_TIMER0_CFG0_REG	定时器周期与更新方法	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	工作模式与开始/停止控制	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM 定时器 0 同步功能配置寄存器	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM 定时器 0 状态	0x0010	RO
MCPWM 定时器 1 配置与状态			
MCPWM_TIMER1_CFG0_REG	定时器更新方式与周期	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	工作模式与开始/停止控制	0x0018	varies
MCPWM_TIMER1_SYNC_REG	同步设置	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM 定时器 1 状态	0x0020	RO
MCPWM 定时器 2 配置与状态			
MCPWM_TIMER2_CFG0_REG	定时器方式与周期	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	工作模式与开始/停止控制	0x0028	varies
MCPWM_TIMER2_SYNC_REG	同步设置	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM 定时器 2 状态	0x0030	RO
MCPWM 定时器常见配置			
MCPWM_TIMER_SYNCI_CFG_REG	定时器同步输入选择	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	为 PWM 操作器选择特定的计时器	0x0038	R/W
MCPWM 操作器 0 配置与状态			
MCPWM_GEN0_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	寄存器 B 的影子寄存器	0x0044	R/W
MCPWM_GEN0_CFG0_REG	故障事件 T0 和 T1 处理	0x0048	R/W
MCPWM_GEN0_FORCE_REG	软件强制 PWM0A 和 PWM0B 输出	0x004C	R/W
MCPWM_GEN0_A_REG	PWM0A 输出上事件触发的操作	0x0050	R/W
MCPWM_GEN0_B_REG	PWM0B 输出上事件触发的操作	0x0054	R/W
MCPWM.DTO_CFG_REG	死区配置	0x0058	R/W
MCPWM.DTO_FED_CFG_REG	FED 的影子寄存器	0x005C	R/W
MCPWM.DTO_RED_CFG_REG	RED 的影子寄存器	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	载波使能与配置	0x0064	R/W
MCPWM_FH0_CFG0_REG	故障事件中 PWM0A 和 PWM0B 上的操作	0x0068	R/W
MCPWM_FH0_CFG1_REG	故障处理的软件触发	0x006C	R/W
MCPWM_FH0_STATUS_REG	故障事件状态	0x0070	RO
MCPWM 操作器 1 配置与状态			
MCPWM_GEN1_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x0074	varies

名称	描述	地址	访问
MCPWM_GEN1_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0078	R/W
MCPWM_GEN1_TSTMP_B_REG	寄存器 B 的影子寄存器	0x007C	R/W
MCPWM_GEN1_CFGO_REG	故障事件 T0 和 T1 处理	0x0080	R/W
MCPWM_GEN1_FORCE_REG	软件强制 PWM1A 和 PWM1B 输出	0x0084	R/W
MCPWM_GEN1_A_REG	PWM1A 输出上的事件触发的操作	0x0088	R/W
MCPWM_GEN1_B_REG	PWM1B 输出上的事件触发的操作	0x008C	R/W
MCPWM_DT1_CFG_REG	死区配置	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	FED 的影子寄存器	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	RED 的影子寄存器	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	使能与配置载波	0x009C	R/W
MCPWM_FH1_CFGO_REG	故障事件中 PWM1A 和 PWM1B 输出上的操作	0x00A0	R/W
MCPWM_FH1_CFG1_REG	故障处理的软件触发	0x00A4	R/W
MCPWM_FH1_STATUS_REG	故障事件状态	0x00A8	RO
MCPWM 操作器 2 的配置与状态			
MCPWM_GEN2_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	寄存器 A 的影子寄存器	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	寄存器 B 的影子寄存器	0x00B4	R/W
MCPWM_GEN2_CFGO_REG	故障事件 T0 和 T1 处理	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	软件强制 PWM2A 和 PWM2B 输出	0x00BC	R/W
MCPWM_GEN2_A_REG	PWM2A 输出上的事件触发的操作	0x00C0	R/W
MCPWM_GEN2_B_REG	PWM2B 输出上的事件触发的操作	0x00C4	R/W
MCPWM_DT2_CFG_REG	死区类型的选择与配置	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	FED 影子寄存器	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	RED 影子寄存器	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	使能与配置载波	0x00D4	R/W
MCPWM_FH2_CFGO_REG	故障事件中 PWM2A 和 PWM2B 输出上的操作	0x00D8	R/W
MCPWM_FH2_CFG1_REG	故障处理的软件触发	0x00DC	R/W
MCPWM_FH2_STATUS_REG	故障事件状态	0x00E0	RO
故障检测与配置			
MCPWM_FAULT_DETECT_REG	故障检测与配置	0x00E4	varies
捕获配置与状态			
MCPWM_CAP_TIMER_CFG_REG	配置捕获定时器	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	捕获定时器同步相位	0x00EC	R/W
MCPWM_CAP_CHO_CFG_REG	捕获通道 0 的配置与使能	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	捕获通道 1 的配置与使能	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	捕获通道 2 的配置与使能	0x00F8	varies
MCPWM_CAP_CHO_REG	捕获通道 0 值的状态	0x00FC	RO
MCPWM_CAP_CH1_REG	捕获通道 1 值的状态	0x0100	RO
MCPWM_CAP_CH2_REG	捕获通道 2 值的状态	0x0104	RO
MCPWM_CAP_STATUS_REG	上一次捕获触发器的边沿	0x0108	RO
使能有效寄存器的更新			
MCPWM_UPDATE_CFG_REG	使能更新	0x010C	R/W
管理中断			

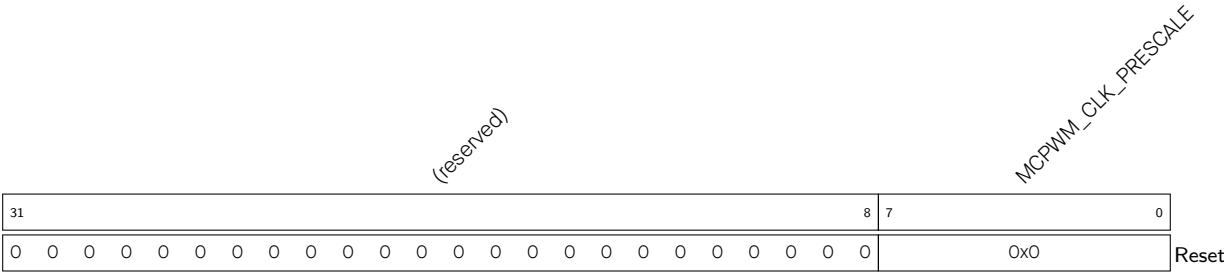
名称	描述	地址	访问
MCPWM_INT_ENA_REG	中断使能位	0x0110	R/W
MCPWM_INT_RAW_REG	原始中断状态	0x0114	R/ WTC / SS
MCPWM_INT_ST_REG	屏蔽中断状态	0x0118	RO
MCPWM_INT_CLR_REG	中断清除位	0x011C	WT
使能 MCPWM 事件			
MCPWM_EVT_EN_REG	使能 MCPWM 事件	0x0120	R/W
MCPWM_EVT_EN2_REG	使能 MCPWM 事件	0x0128	R/W
使能 MCPWM 任务			
MCPWM_TASK_EN_REG	使能 MCPWM 任务	0x0124	R/W
MCPWM 生成器配置			
MCPWM_OPO_TSTMP_E1_REG	配置生成器 0 时间戳 E1 值	0x012C	R/W
MCPWM_OPO_TSTMP_E2_REG	配置生成器 0 时间戳 E2 值	0x0130	R/W
MCPWM_OP1_TSTMP_E1_REG	配置生成器 1 时间戳 E1 值	0x0134	R/W
MCPWM_OP1_TSTMP_E2_REG	配置生成器 1 时间戳 E2 值	0x0138	R/W
MCPWM_OP2_TSTMP_E1_REG	配置生成器 2 时间戳 E1 值	0x013C	R/W
MCPWM_OP2_TSTMP_E2_REG	配置生成器 2 时间戳 E2 值	0x0140	R/W
MCPWM APB 配置			
MCPWM_CLK_REG	MCPWM APB 配置	0x0144	R/W
版本寄存器			
MCPWM_VERSION_REG	版本控制寄存器	0x0148	R/W

31.6 寄存器

本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

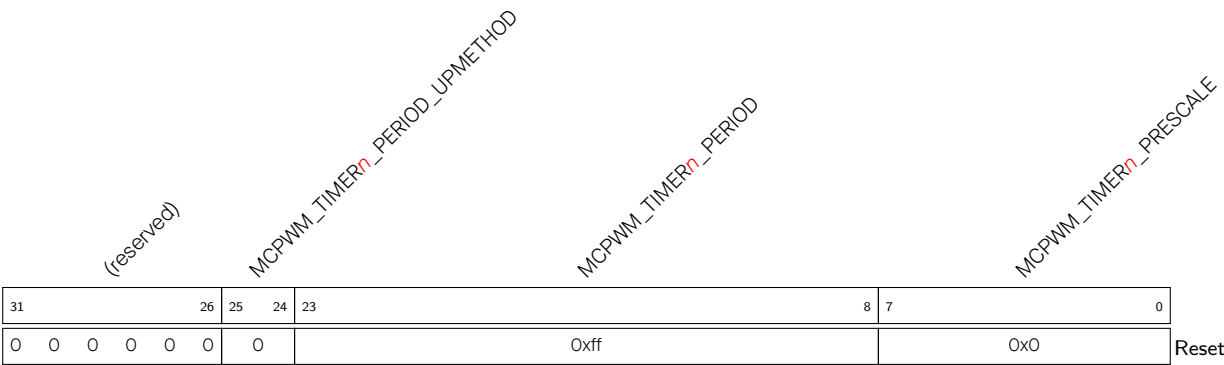
关于保留 (reserved) 域的处理，请查看章节 [如何配置寄存器的保留域](#)。

Register 31.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE 配置时钟预分频系数，使得 PWM_CLK 的周期为 6.25 ns * (PWM_CLK_PRESCALE + 1)。(R/W)

Register 31.2. MCPWM_TIMERn_CFG0_REG (n: 0-2) (0x0004+0x10*n)



MCPWM_TIMERn_PRESCALE 配置定时器n的预分频系数，使得 PTO_CLK 的周期 = PWM_CLK 的周期 * (PWM_TIMERn_PRESCALE + 1)。(R/W)

MCPWM_TIMERn_PERIOD 配置定时器n的影子周期。(R/W)

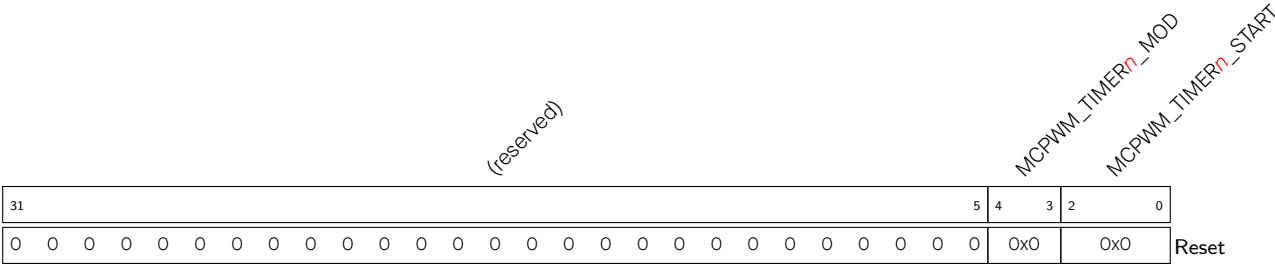
MCPWM_TIMERn_PERIOD_UPMETHOD 配置 PWM 定时器n 周期有效寄存器的更新方式。

- 0: 立即更新
- 1: 发生 TEZ 事件时更新
- 2: 同步时更新
- 3: 发生 TEZ 事件或同步时更新

本文档中，TEZ 指定定时器为 0 时的事件。

(R/W)

Register 31.3. MCPWM_TIMERn_CFG1_REG(n: 0-2) (0x0008+0x10*n)



MCPWM_TIMERn_START 控制 PWM 定时器的开启与关闭。

- 0：如果开启，在 TEZ 事件发生时停止
- 1：如果开启，在 TEP 事件发生时停止
- 2：开启
- 3：开启，并在下一个 TEZ 事件发生时停止
- 4：开启，并在下一个 TEP 事件发生时停止
- 其他值：无效值，不产生影响
- 本文档中，TEP 指定时器为周期值时发生的事件。
- (R/W/SC)

MCPWM_TIMERn_MOD 配置 PWM 定时器 0 的工作模式。

- 0：暂停
- 1：递增模式
- 2：递减模式
- 3：递增递减循环模式
- (R/W)

Register 31.4. MCPWM_TIMERn_SYNC_REG (n: 0-2) (0x000C+0x10*n)

(reserved)												MCPWM_TIMER _n _PHASE_DIRECTION												MCPWM_TIMER _n _PHASE												MCPWM_TIMER _n _SYNCO_SEL												MCPWM_TIMER _n _SYNC_SW												MCPWM_TIMER _n _SYNCl_EN											
31											21	20	19											4	3	2	1	0																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										0	0	0	Reset																																										

MCPWM_TIMERn_SYNCI_EN 配置是否开启同步输入事件发生时的定时器n 相位重载。

0: 关闭

1: 开启

(R/W)

MCPWM_TIMERn_SYNC_SW 配置是否触发软件同步。

0: 无效

1: 触发软件同步

(R/W)

MCPWM_TIMERn_SYNCO_SEL 选择 PWM 定时器n 的同步输出来源。

0: 同步。取反 MCPWM_TIMERn_SYNC_SW 位时始终生成同步输出。

1: TEZ

2: TEP

3: 无效

(R/W)

MCPWM_TIMERn_PHASE 同步事件中定时器n 重载的相位。

(R/W)

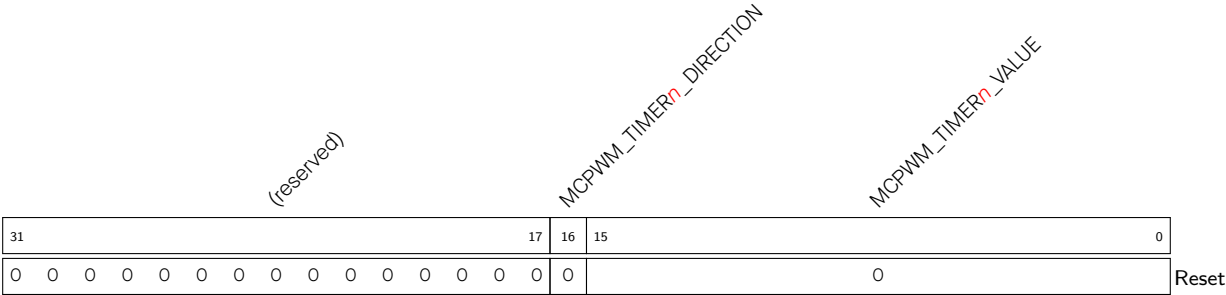
MCPWM_TIMERn_PHASE_DIRECTION 当定时器n 为递增-递减循环模式时，配置该定时器方向。

0: 递增

1: 递减

(R/W)

Register 31.5. MCPWM_TIMER n _STATUS_REG(n : 0-2) (0x0010+0x10* n)

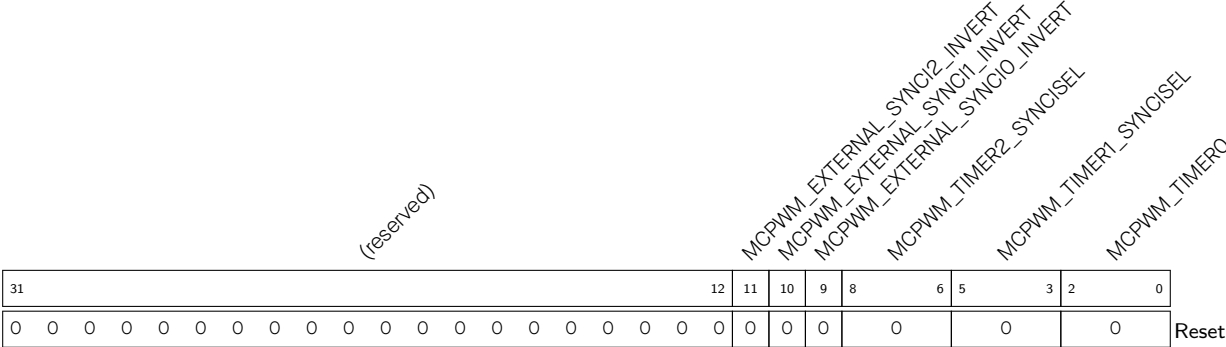


MCPWM_TIMER n _VALUE 表示当前 PWM 定时器 n 计数器的值。(RO)

MCPWM_TIMER n _DIRECTION 配置当前 PWM 定时器 n 的计数器模式。

- 0：递增模式
- 1：递减模式
- (RO)

Register 31.6. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



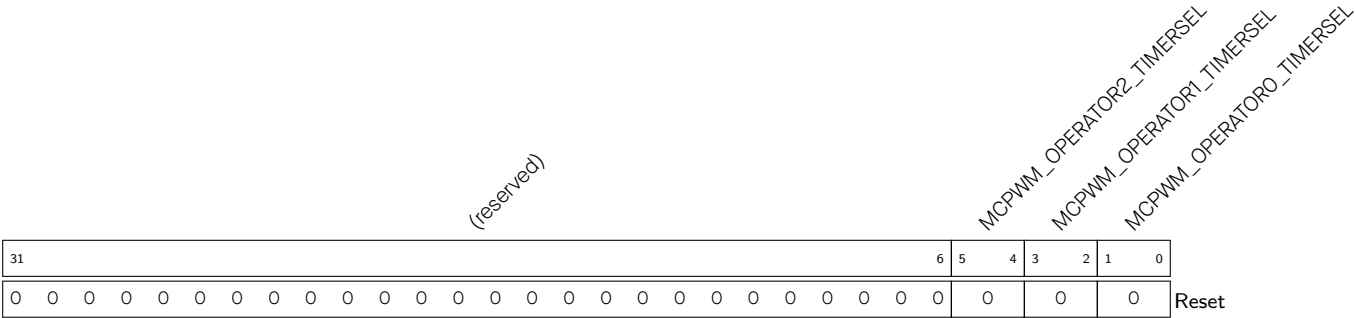
MCPWM_TIMER n _SYNCISEL 选择 PWM 定时器 n 的同步输入来源。

- 1: PWM 定时器 0 同步输出
- 2: PWM 定时器 1 同步输出
- 3: PWM 定时器 2 同步输出
- 4: 来自 GPIO 矩阵的 SYNC0
- 5: 来自 GPIO 矩阵的 SYNC1
- 6: 来自 GPIO 矩阵的 SYNC2
- 其他值：未选择任何同步输入
- (R/W)

MCPWM_EXTERNAL_SYNCI n _INVERT 配置是否将来自 GPIO 矩阵的 SYNC n 反相。

- 0：不反相
- 1：反相
- (R/W)

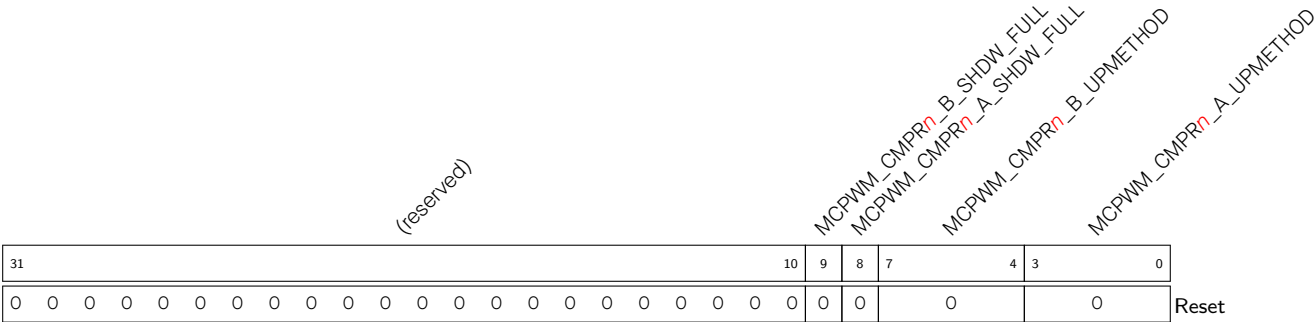
Register 31.7. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR*n*_TIMERSEL 选择 PWM 操作器*n* 的定时参考来源。

- 0: 定时器 0
 - 1: 定时器 1
 - 2: 定时器 2
 - 3: 无效，将选择定时器 2
- (R/W)

Register 31.8. MCPWM_GEN n _STMP_CFG_REG (n : 0-2) (0x003C+0x38* n)



MCPWM_CMPR n _A_UPMETHOD 配置 PWM 生成器 n 时间戳寄存器 A 的有效寄存器的更新方式。

- 所有位值为 0：立即更新
 - bit0 为 1：发生 TEZ 事件时更新
 - bit1 为 1：发生 TEP 事件时更新
 - bit2 为 1：发生同步时间时更新
 - bit3 为 1：关闭更新
- (R/W)

MCPWM_CMPR n _B_UPMETHOD 配置 PWM 生成器 n 时间戳寄存器 B 的有效寄存器的更新方式。

- 所有位值为 0：立即更新
 - bit0 为 1：发生 TEZ 事件时更新
 - bit1 为 1：发生 TEP 事件时更新
 - bit2 为 1：发生同步时间时更新
 - bit3 为 1：关闭更新
- (R/W)

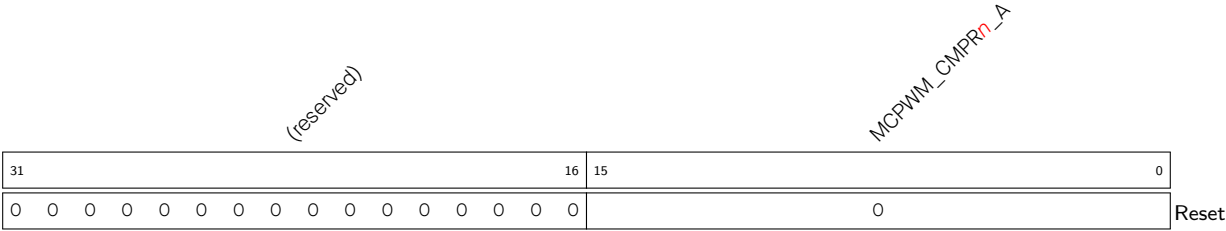
MCPWM_CMPR n _A_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

- 0：将影子寄存器中最新的值写入有效寄存器 A 中
 - 1：将值写入 PWM 生成器 n 时间戳寄存器 A 的影子寄存器中，等待传输给有效寄存器 A。
- (R/SC/WTC)

MCPWM_CMPR n _B_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

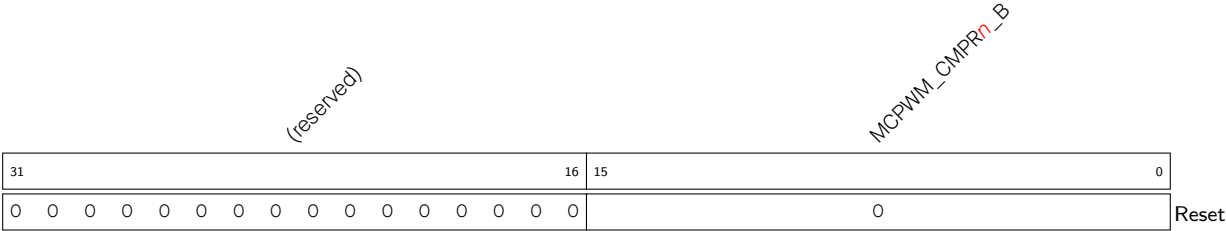
- 0：将影子寄存器中最新的值写入有效寄存器 B 中
 - 1：将值写入 PWM 生成器 n 时间戳寄存器 B 的影子寄存器中，等待传输给有效寄存器 B。
- (R/SC/WTC)

Register 31.9. MCPWM_GEN n _TSTMP_A_REG(n : 0-2) (0x0040+0x38* n)



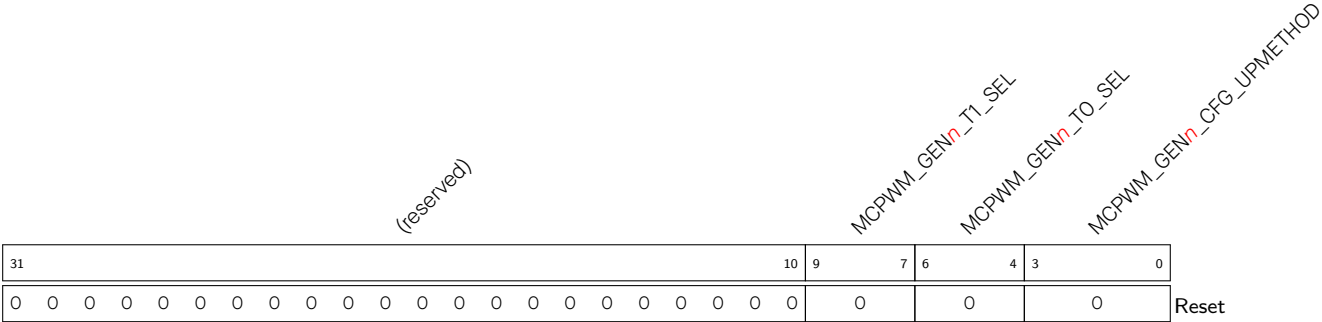
MCPWM_CMPR n _A 配置 PWM 生成器 n 时间戳寄存器 A 的影子寄存器。
(R/W)

Register 31.10. MCPWM_GEN n _TSTMP_B_REG(n : 0-2) (0x0044+0x38* n)



MCPWM_CMPR n _B 配置 PWM 生成器 n 时间戳寄存器 B 的影子寄存器。
(R/W)

Register 31.11. MCPWM_GENn_CFGO_REG(*n*: 0-2) (0x0048+0x38**n*)



MCPWM_GENn_CFG_UPMETHOD 配置 PWM 生成器*n* 有效配置寄存器的更新方式。

- 所有 bit 值为 0: 立即更新
- bit0 为 1: 发生 TEZ 事件时更新
- bit1 为 1: 发生 TEP 事件时更新
- bit2 为 1: 发生同步时间时更新
- bit3 为 1: 关闭更新
- (R/W)

MCPWM_GENn_TO_SEL 选择 PWM 生成器*n* event_t0 的信号源，立即生效。

- 0: fault_event0
- 1: fault_event1
- 2: fault_event2
- 3: sync_taken
- 4: 无
- (R/W)

MCPWM_GENn_T1_SEL 选择 PWM 生成器*n* event_t1 的信号源，立即生效。

- 0: fault_event0
- 1: fault_event1
- 2: fault_event2
- 3: sync_taken
- 4: 无
- (R/W)

Register 31.12. MCPWM_GEN n _FORCE_REG(n : 0-2) (0x004C+0x38* n)

[illegible]

MCPWM_GEN n _CNTUFORCE_UPMETHOD 配置生成器 n 的连续软件强制事件更新方式。

所有 bit 为 0 时：立即更新

bit0 为 1: 发生 TEZ 事件时更新

bit1 为 1: 发生 TEP 事件时更新

bit2 为 1: 发生 TEA 事件时更新

bit3 为 1: 发生 TEB 事件时更新

bit4: 发生同步事件时更新

bit5 为 1: 关闭更新

本文档中，TEA/B 指定定时器值为寄存器 A/B 的值时生成的事件。

(R/W)

MCPWM_GEN n _A_CNTUFORCE_MODE 配置 PWM n A 的连续软件强制模式。

0: 关闭

1: 低电平

2: 高电平

3: 关闭

(R/W)

MCPWM_GEN_n_B_CNTUFORCE_MODE 配置 PWM_n B 的连续软件强制模式。

0: 关闭

1: 低电平

2: 高电平

3: 关闭

(R/W)

MCPWM_GEN n _A_NCIFORCE 配置是否触发 PWM n A 上的非连续即时软件强制事件。

0: 无效

1: 触发 PWM_n A 上的非连续即时软件强制事件

(R/W)

见下页...

Register 31.12. MCPWM_GEN n _FORCE_REG(n : 0-2) (0x004C+0x38* n)

... 接上页

MCPWM_GEN n _A_NCIFORCE_MODE 配置用于 PWM n A 的非连续即时软件强制模式。

- 0: 关闭
 - 1: 低电平
 - 2: 高电平
 - 3: 关闭
- (R/W)

MCPWM_GEN n _B_NCIFORCE 配置是否触发 PWM n B 上的非连续即时软件强制事件。

- 0: 无效
 - 1: 触发 PWM n B 上的非连续即时软件强制事件
- (R/W)

MCPWM_GEN n _B_NCIFORCE_MODE 配置用于 PWM n B 的非连续即时软件强制模式。

- 0: 关闭
 - 1: 低电平
 - 2: 高电平
 - 3: 关闭
- (R/W)

Register 31.13. MCPWM_GENn_A_REG (n: 0-2) (0x0050+0x38*n)

(reserved)								MCPWM_GENn_A_DT1		MCPWM_GENn_A DTO		MCPWM_GENn_A DTEB		MCPWM_GENn_A DTEA		MCPWM_GENn_A DTEP		MCPWM_GENn_A DTEZ		MCPWM_GENn_A_UT1		MCPWM_GENn_A_UTO		MCPWM_GENn_A_UTEB		MCPWM_GENn_A_UTEA		MCPWM_GENn_A_UTEP		MCPWM_GENn_A_UTEZ		
31								24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_GENn_A_UTEZ 配置定时器递增时，TEZ 事件在 PWMnA 上触发的操作。

- 0：波形无改变
 - 1：拉低
 - 2：拉高
 - 3：取反
- (R/W)

MCPWM_GENn_A_UTEP 配置定时器递增时，TEP 事件在 PWMnA 上触发的操作。

- 0：波形无改变
 - 1：拉低
 - 2：拉高
 - 3：取反
- (R/W)

MCPWM_GENn_A_UTEA 配置定时器递增时，TEA 事件在 PWMnA 上触发的操作。

- 0：波形无改变
 - 1：拉低
 - 2：拉高
 - 3：取反
- (R/W)

MCPWM_GENn_A_UTEB 配置定时器递增时，TEB 事件在 PWMnA 上触发的操作。

- 0：波形无改变
 - 1：拉低
 - 2：拉高
 - 3：取反
- (R/W)

见下页...

Register 31.13. MCPWM_GEN n _A_REG(n : 0-2) (0x0050+0x38* n)

... 接上页

MCPWM_GEN n _A_UTO 配置定时器递增时, event_t0 在 PWM n A 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _A_UT1 配置定时器递增时, event_t1 在 PWM n A 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _A_DTEZ 配置定时器递减时, TEZ 事件在 PWM n A 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _A_DTEP 配置定时器递减时, TEP 事件在 PWM n A 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _A_DTEA 配置定时器递减时, TEA 事件在 PWM n A 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

见下页...

Register 31.13. MCPWM_GEN n _A_REG(n : 0-2) (0x0050+0x38* n)

... [接上页](#)

MCPWM_GEN n _A_DTEB 配置定时器递减时, TEB 事件在 PWM n A 上触发的操作。

0: 波形无改变

1: 拉低

2: 拉高

3: 取反

(R/W)

MCPWM_GEN n _A_DTO 配置定时器递减时, event_t0 在 PWM n A 上触发的操作。

0: 波形无改变

1: 拉低

2: 拉高

3: 取反

(R/W)

MCPWM_GEN n _A_DT1 配置定时器递减时, event_t1 在 PWM n A 上触发的操作。

0: 波形无改变

1: 拉低

2: 拉高

3: 取反

(R/W)

Register 31.14. MCPWM_GENn_B_REG (n: 0-2) (0x0054+0x38*n)

(reserved)																MCPWM_GENn_B_DT1		MCPWM_GENn_B DTO		MCPWM_GENn_B DTEB		MCPWM_GENn_B DTEA		MCPWM_GENn_B DTEP		MCPWM_GENn_B DTEZ		MCPWM_GENn_B_UT1		MCPWM_GENn_B_UTO		MCPWM_GENn_B_UTEB		MCPWM_GENn_B_UTEA		MCPWM_GENn_B_UTEZ	
31								24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset					

MCPWM_GENn_B_UTEZ 定时器递增时，TEZ 在 PWMnB 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GENn_B_UTEP 定时器递增时，TEP 在 PWMnB 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GENn_B_UTEA 定时器递增时，TEA 在 PWMnB 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GENn_B_UTEB 定时器递增时，TEB 在 PWMnB 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

见下页...

Register 31.14. MCPWM_GEN n _B_REG(n : 0-2) (0x0054+0x38* n)

... 接上页

MCPWM_GEN n _B_UTO 定时器递增时, event_t0 在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _B_UT1 定时器递增时, event_t1 在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _B_DTEZ 定时器递减时, TEZ 事件在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _B_DTEP 定时器递减时, TEP 事件在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _B_DTEA 定时器递减时, TEA 事件在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

见下页...

Register 31.14. MCPWM_GEN n _B_REG(n : 0-2) (0x0054+0x38* n)

... [接上页](#)

MCPWM_GEN n _B_DTEB 定时器递减时, TEB 事件在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _B_DTO 定时器递减时, event_t0 事件在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

MCPWM_GEN n _B_DT1 定时器递减时, event_t1 在 PWM n B 上触发的操作。

- 0: 波形无改变
 - 1: 拉低
 - 2: 拉高
 - 3: 取反
- (R/W)

Register 31.15. MCPWM_DTn_CFG_REG(n: 0-2) (0x0058+0x38*n)

(reserved)																		MCPWM_DB _n _CLK_SEL										MCPWM_DB _n _B_OUTBYPASS										MCPWM_DB _n _A_OUTBYPASS										MCPWM_DB _n _FED_OUTINVERT										MCPWM_DB _n _RED_OUTINVERT										MCPWM_DB _n _B_OUTSWAP										MCPWM_DB _n _A_OUTSWAP										MCPWM_DB _n _DEB_MODE										MCPWM_DB _n _RED_UPMETHOD										MCPWM_DB _n _FED_UPMETHOD									
31																		18	17	16	15	14	13	12	11	10	9	8	7			4	3			0																																																																																	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0		Reset																																																																																											

MCPWM_DBn_FED_UPMETHOD 配置下降沿延迟有效寄存器的更新方式。

- 所有位值为 0: 立即更新
 - bit0 为 1: 发生 TEZ 事件时更新
 - bit1 为 1: 发生 TEP 事件时更新
 - bit2 为 1: 发生同步时间时更新
 - bit3 为 1: 关闭更新
- (R/W)

MCPWM_DBn_RED_UPMETHOD 配置上升沿延迟有效寄存器的更新方式。

- 所有位值为 0: 立即更新
 - bit0 为 1: 发生 TEZ 事件时更新
 - bit1 为 1: 发生 TEP 事件时更新
 - bit2 为 1: 发生同步时间时更新
 - bit3 为 1: 关闭更新
- (R/W)

MCPWM_DBn_DEB_MODE 配置表 31.3-5 中的 S8 开关。详细配置信息请参考表 31.3-6。

- 0: FED/RED 分别在不同路径上生效
 - 1: FED/RED 在 B 路径上生效, A 输出为旁路或 dulpB 模式
- (R/W)

MCPWM_DBn_A_OUTSWAP 配置表 31.3-5 中的 S6 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_B_OUTSWAP 配置表 31.3-5 中的 S7 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_RED_INSEL 配置表 31.3-5 中的 S4 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_FED_INSEL 配置表 31.3-5 中的 S5 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_RED_OUTINVERT 配置表 31.3-5 中的 S2 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_FED_OUTINVERT 配置表 31.3-5 中的 S3 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_A_OUTBYPASS 配置表 31.3-5 中的 S1 开关。详细配置信息请参考表 31.3-6。 (R/W)

MCPWM_DBn_B_OUTBYPASS 配置表 31.3-5 中的 S0 开关。详细配置信息请参考表 31.3-6。 (R/W)

见下页...

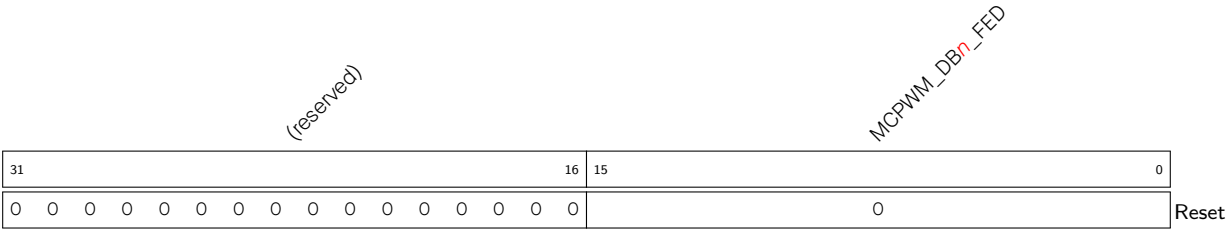
Register 31.15. MCPWM_DTn_CFG_REG(n: 0-2) (0x0058+0x38*n)

... 接上页

MCPWM_DBn_CLK_SEL 选择死区时间生成器 0 的时钟。

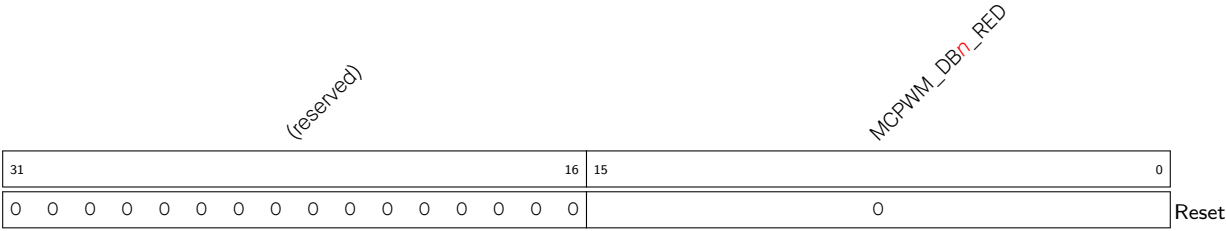
0: PWM_CLK
1: PT_CLK
(R/W)

Register 31.16. MCPWM_DTn_FED_CFG_REG(n: 0-2) (0x005C+0x38*n)



MCPWM_DBn_FED 配置 FED 的影子寄存器。(R/W)

Register 31.17. MCPWM_DTn_RED_CFG_REG(n: 0-2) (0x0060+0x38*n)



MCPWM_DBn_RED 配置 RED 的影子寄存器。(R/W)

Register 31.18. MCPWM_CARRIER n _CFG_REG(n : 0-2) (0x0064+0x38* n)

(reserved)														MCPWM_CHOPPER _n _IN_INVERT		MCPWM_CHOPPER _n _OUT_INVERT		MCPWM_CHOPPER _n _OSHTWTH		MCPWM_CHOPPER _n _DUTY		MCPWM_CHOPPER _n _PRESCALE		MCPWM_CHOPPER _n _EN	
31														14	13	12	11	8	7	5	4	1	0	Reset	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								

- MCPWM_CHOPPER n _EN 配置是否使能载波 n 。
0: 绕过
1: 使能
(R/W)
- MCPWM_CHOPPER n _PRESCALE 配置 PWM 载波 n 时钟 (PC_CLK) 的预分频值。PC_CLK 周期 = PWM_CLK 周期 * (PWM_CARRIER n _PRESCALE + 1)。(R/W)
- MCPWM_CHOPPER n _DUTY 配置载波占空比。占空比 = PWM_CARRIER n _DUTY/8。(R/W)
- MCPWM_CHOPPER n _OSHTWTH 配置载波第一个脉冲的宽度，单位为载波周期。(R/W)
- MCPWM_CHOPPER n _OUT_INVERT 配置是否将此模块的 PWM n A 和 PWM n B 输出反相。
0: 正常输出
1: 反相
(R/W)
- MCPWM_CHOPPER n _IN_INVERT 配置是否将此模块的 PWM n A 和 PWM n B 输入反相。
0: 正常输入
1: 反相
(R/W)

Register 31.19. MCPWM_FH_{*n*}_CFG0_REG(*n*: 0-2) (0x0068+0x38n*)**

[illegible]

MCPWM_TZ_n_SW_CBC 配置是否使能软件强制逐周期模式操作。

0: 关闭

1: 使能

(R/W)

MCPWM_TZ_n_F2_CBC 配置 fault_event2 是否触发逐周期模式操作。

0: 关闭

1: 使能

(R/W)

MCPWM_TZ_n_F1_CBC 配置 fault_event1 是否触发逐周期模式操作。

0: 关闭

1: 使能

(R/W)

MCPWM_TZ_n_F0_CBC 配置 fault_event0 是否触发逐周期模式操作。

0: 关闭

1: 使能

(R/W)

MCPWM_TZ_n_SW_OST 配置是否使能软件强制一次性模式操作。

0: 关闭

1: 使能

(R/W)

MCPWM_TZ_n_F2_OST 配置 fault_event2 是否触发一次性模式操作。

0: 关闭

1: 使能

(R/W)

见下页...

Register 31.19. MCPWM_FH n _CFG0_REG(n : 0-2) (0x0068+0x38* n)

... 接上页

MCPWM_TZ n _F1_OST 配置 fault_event1 是否触发一次性模式操作。

- 0: 关闭
 - 1: 使能
- (R/W)

MCPWM_TZ n _Fn_OST 配置 fault_event0 是否触发一次性模式操作。

- 0: 关闭
 - 1: 使能
- (R/W)

MCPWM_TZ n _A_CBC_D 配置定时器递减计数并且发生故障事件时, PWM n A 上的逐周期模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

MCPWM_TZ n _A_CBC_U 配置定时器递增计数并且发生故障事件时, PWM n A 上的逐周期模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

MCPWM_TZ n _A_OST_D 配置定时器递减计数并且发生故障事件时, PWM n A 上的一次性模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

见下页...

Register 31.19. MCPWM_FH n _CFG0_REG(n : 0-2) (0x0068+0x38* n)

... 接上页

MCPWM_TZ n _A_OST_U 配置定时器递增计数并且发生故障事件时, PWM n A 上的一次性模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

MCPWM_TZ n _B_CBC_D 配置定时器递减计数并且发生故障事件时, PWM n B 上的逐周期模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

MCPWM_TZ n _B_CBC_U 配置定时器递增计数并且发生故障事件时, PWM n B 上的逐周期模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

MCPWM_TZ n _B_OST_D 配置定时器递减计数并且发生故障事件时, PWM n B 上的一次性模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

MCPWM_TZ n _B_OST_U 配置定时器递增计数并且发生故障事件时, PWM n B 上的一次性模式操作。

- 0: 无
 - 1: 强制拉低
 - 2: 强制拉高
 - 3: 取反
- (R/W)

Register 31.20. MCPWM_FH_n_CFG1_REG(*n*: 0-2) (0x006C+0x38**n*)

(reserved)																								MCPWM_TZ _n _FORCE_OST MCPWM_TZ _n _FORCE_CBC MCPWM_TZ _n _CBCPULSE MCPWM_TZ _n _CLR_OST						
31																								5	4	3	2	1	0	Reset
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																								0	0	0	0			

MCPWM_TZ_n_CLR_OST 配置是否通过软件清除持续一次性模式操作。

0: 无效

1: 通过软件清除持续一次性模式操作

(R/W)

MCPWM_TZ_n_CBCPULSE 配置逐周期模式操作更新的时间点。

bit0 为 1: 发生 TEZ 事件时

bit1 为 1: 发生 TEP 事件时

(R/W)

MCPWM_TZ_n_FORCE_CBC 配置是否触发逐周期模式的操作。

0: 无效

1: 触发

(R/W)

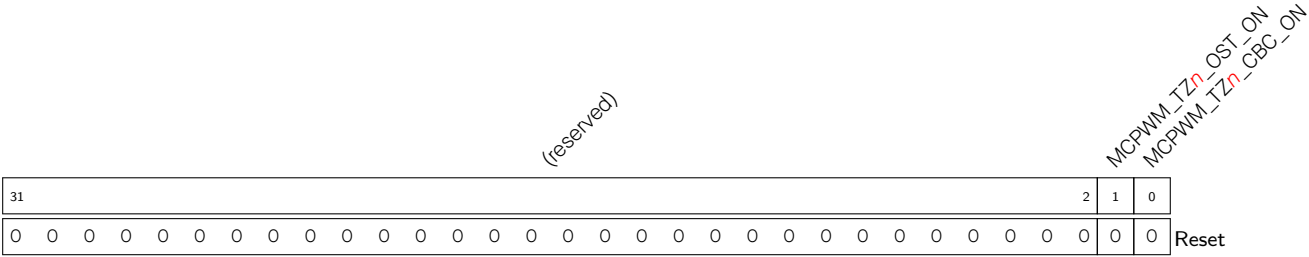
MCPWM_TZ_n_FORCE_OST 配置是否触发一次性模式的操作。

0: 无效

1: 触发

(R/W)

Register 31.21. MCPWM_FH n _STATUS_REG(n : 0-2) (0x0070+0x38* n)



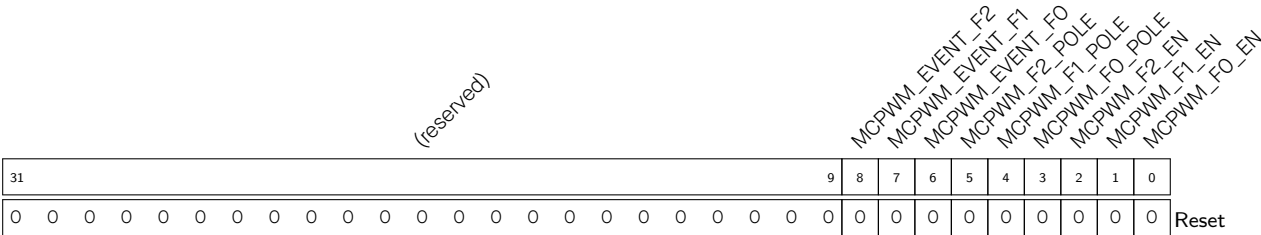
MCPWM_TZ n _CBC_ON 表示是否正在进行逐周期模式的操作。

- 0: 不存在正在进行的逐周期模式操作
- 1: 逐周期模式的操作正在进行
- (RO)

MCPWM_TZ n _OST_ON 表示是否正在进行一次性模式的操作。

- 0: 不存在正在进行的一次性模式操作
- 1: 一次性模式的操作正在进行
- (RO)

Register 31.22. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F n _EN 配置是否使能生成 fault_event n 。

- 0: 无效
- 1: 使能
- (R/W)

MCPWM_F n _POLE 配置来自 GPIO 矩阵的 FAULT n 信号源触发 fault_event n 时极性。

- 0: 低电平触发
- 1: 高电平触发
- (R/W)

MCPWM_EVENT_F n 表示 fault_event n 事件的状态。此字段由硬件置 1 和清零。

- 0: fault_event n 事件停止
- 1: fault_event n 事件持续
- (RO)

Register 31.23. MCPWM_CAP_TIMER_CFG_REG (0x00E8)

(reserved)																								MCPWM_CAP_SYNC_SW		MCPWM_CAP_SYNCI_SEL		MCPWM_CAP_SYNCI_EN		MCPWM_CAP_TIMER_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
31																								6	5	4		2		1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

MCPWM_CAP_TIMER_EN 配置是否使能捕获定时器在 APB_CLK 下的递增。

- 0: 无效
- 1: 使能捕获定时器在 APB_CLK 下的递增
- (R/W)

MCPWM_CAP_SYNCI_EN 配置是否使能捕获定时器同步。

- 0: 无效
- 1: 使能捕获定时器同步
- (R/W)

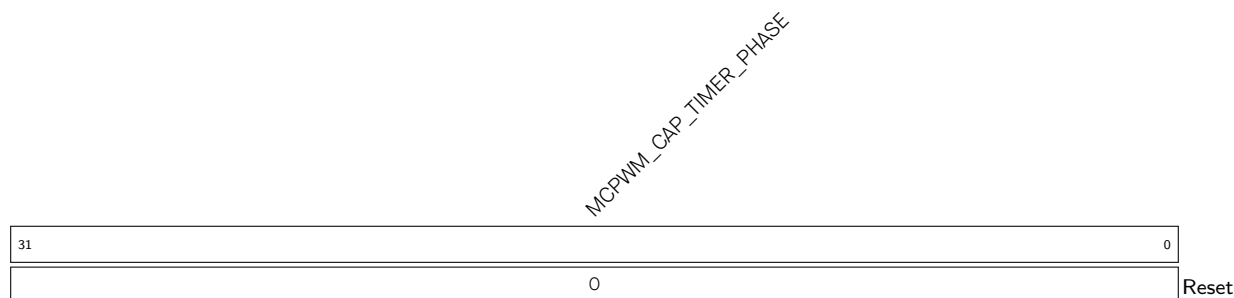
MCPWM_CAP_SYNCI_SEL 配置捕获模块的同步输入。

- 0: 无
- 1: 定时器 0 的同步输出
- 2: 定时器 1 的同步输出
- 3: 定时器 2 的同步输出
- 4: 来自 GPIO 矩阵的 SYNC0
- 5: 来自 GPIO 矩阵的 SYNC1
- 6: 来自 GPIO 矩阵的 SYNC2
- 7: 无
- (R/W)

MCPWM_CAP_SYNC_SW 当 **MCPWM_CAP_SYNCI_EN** 置位时，配置是否同步捕获定时器，在捕获定时器中写入相位寄存器的值。

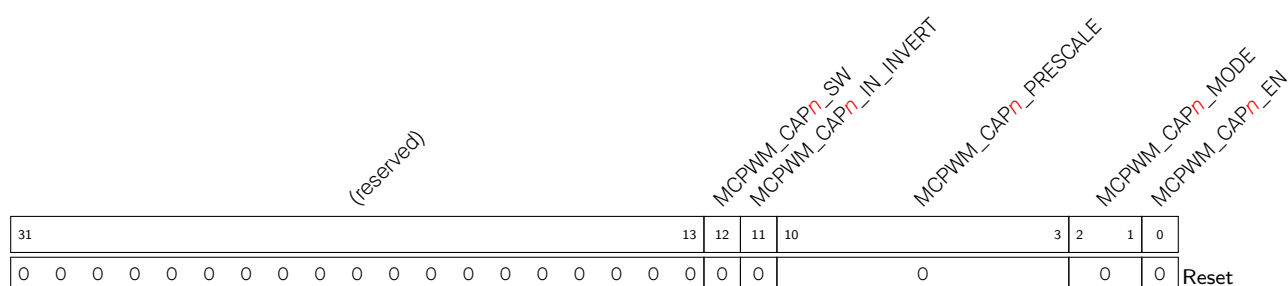
- 0: 无效
- 1: 同步捕获定时器
- (WT)

Register 31.24. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM CAP TIMER PHASE 配置捕获定时器同步操作的相位值。(R/W)

Register 31.25. MCPWM_CAP_CH n _CFG_REG (n : 0-2) (0x00F0+0x4* n)



MCPWM_CAP n _EN 配置是否使能信道 n 上的捕获事件。

0: 不使能

1: 使能

(R/W)

MCPWM_CAP n _MODE 配置预分频后信道 n 上的捕获沿。

0: 无

bit0 为 1: 使能下降沿捕获。

bit1 为 1: 使能上升沿捕获。

(R/W)

MCPWM_CAP n _PRESCALE 配置 CAP n 上升沿的预分频值。预分频值 = PWM_CAP n _PRESCALE + 1。(R/W)

MCPWM CAP_n IN INVERT 配置是否在预分频前反相来自 GPIO 矩阵的 CAP_n。

0: 无效

1: 在预分频前反相来自 GPIO 矩阵的 CAP_n

(R/W)

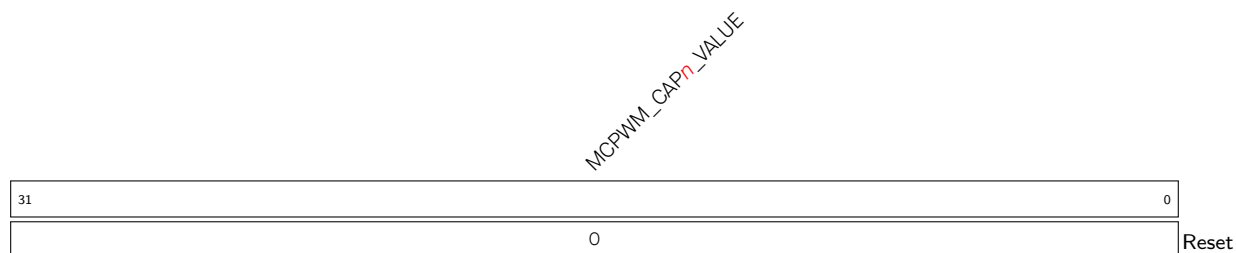
MCPWM CAP_n SW 配置是否触发信道 0 上的软件强制捕获事件。

0: 不触发

1: 触发

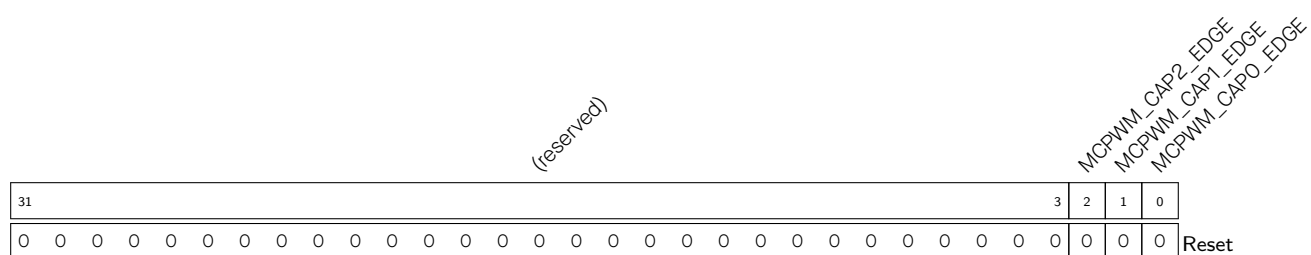
(WT)

Register 31.26. MCPWM_CAP_CH n _REG (n : 0-2) (0x00FC+0x4* n)



MCPWM_CAP_{*n*}_VALUE 表示信道 *n* 上一次捕获的值。(RO)

Register 31.27. MCPWM_CAP_STATUS_REG (0x0108)



MCPWM_CAP n _EDGE 信道 n 上一次捕获触发事件的边沿。

0: 上升沿

1: 下降沿

(RO)

Register 31.28. MCPWM_UPDATE_CFG_REG (0x010C)

(reserved)																								MCPWM_OP2_FORCE_UP MCPWM_OP2_UP_EN MCPWM_OP1_FORCE_UP MCPWM_OP1_UP_EN MCPWM_OPO_FORCE_UP MCPWM_OPO_UP_EN MCPWM_GLOBAL_FORCE_UP MCPWM_GLOBAL_UP_EN															
31																							8	7	6	5	4	3	2	1	0								
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1					
																																Reset							

MCPWM_GLOBAL_UP_EN 配置是否更新 MCPWM 模块的所有有效寄存器。

0: 无效

1: 更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_GLOBAL_FORCE_UP 配置是否强制更新 MCPWM 模块的所有有效寄存器。

0: 无效

1: 强制更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_OPO_UP_EN 当 **MCPWM_GLOBAL_UP_EN** 置1时, 配置是否更新 PWM 操作器 0 中的有效寄存器。

0: 无效

1: 更新 PWM 操作器 0 中的有效寄存器

(R/W)

MCPWM_OPO_FORCE_UP 配置是否强制更新 PWM 操作器 0 中的有效寄存器。

0: 无效

1: 强制更新

(R/W)

MCPWM_OP1_UP_EN 当 **MCPWM_GLOBAL_UP_EN** 置 1 时，配置是否更新 PWM 操作器 1 中的有效寄存器。

0: 无效

1: 更新 PWM 操作器 1 中的有效寄存器

(R/W)

MCPWM_OP1_FORCE_UP 配置是否强制更新 PWM 操作器 1 中的有效寄存器。

0: 无效

1: 强制更新

(R/W)

MCPWM_OP2_UP_EN 当 **MCPWM_GLOBAL_UP_EN** 置1时，配置是否更新 PWM 操作器 2 中的有效寄存器。

0: 无效

1: 更新 PWM 操作器 2 中的有效寄存器

(R/W)

见下页...

Register 31.28. MCPWM_UPDATE_CFG_REG (0x010C)

... [接上页](#)

MCPWM_OP2_FORCE_UP 配置是否强制更新 PWM 操作器 2 中的有效寄存器。

0: 无效

1: 强制更新

(R/W)

Register 31.29. MCPWM_INT_ENA_REG (0x0110)

(reserved) MCPWM_CAP2_INT_ENA MCPWM_CAP1_INT_ENA MCPWM_CAP0_INT_ENA MCPWM_TZ2_INT_ENA MCPWM_TZ1_INT_ENA MCPWM_TZ0_INT_ENA MCPWM_TZ2_OST_INT_ENA MCPWM_TZ1_OST_INT_ENA MCPWM_TZ0_OST_INT_ENA MCPWM_TZ1_CBC_INT_ENA MCPWM_TZ0_CBC_INT_ENA MCPWM_CMPR2_INT_ENA MCPWM_CMPR1_INT_ENA MCPWM_CMPR0_INT_ENA MCPWM_FAULT2_INT_ENA MCPWM_FAULT1_INT_ENA MCPWM_FAULT0_INT_ENA MCPWM_TIMER2_INT_ENA MCPWM_TIMER1_INT_ENA MCPWM_TIMER0_INT_ENA MCPWM_TIMER2_TEP_INT_ENA MCPWM_TIMER1_TEP_INT_ENA MCPWM_TIMER0_TEP_INT_ENA MCPWM_TIMER2_TZ_INT_ENA MCPWM_TIMER1_TZ_INT_ENA MCPWM_TIMER0_TZ_INT_ENA MCPWM_TIMER2_STOP_INT_ENA MCPWM_TIMER1_STOP_INT_ENA MCPWM_TIMER0_STOP_INT_ENA																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_ENA 写 1 使能 MCPWM_TIMER0_STOP_INT。 (R/W)

MCPWM_TIMER1_STOP_INT_ENA 写 1 使能 MCPWM_TIMER1_STOP_INT。 (R/W)

MCPWM_TIMER2_STOP_INT_ENA 写 1 使能 MCPWM_TIMER2_STOP_INT。 (R/W)

MCPWM_TIMER0_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER0_TEZ_INT。 (R/W)

MCPWM_TIMER1_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER1_TEZ_INT。 (R/W)

MCPWM_TIMER2_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER2_TEZ_INT。 (R/W)

MCPWM_TIMER0_TEP_INT_ENA 写 1 使能 MCPWM_TIMER0_TEP_INT。 (R/W)

MCPWM_TIMER1_TEP_INT_ENA 写 1 使能 MCPWM_TIMER1_TEP_INT。 (R/W)

MCPWM_TIMER2_TEP_INT_ENA 写 1 使能 MCPWM_TIMER2_TEP_INT。 (R/W)

MCPWM_FAULT0_INT_ENA 写 1 使能 MCPWM_FAULT0_INT。 (R/W)

MCPWM_FAULT1_INT_ENA 写 1 使能 MCPWM_FAULT1_INT。 (R/W)

MCPWM_FAULT2_INT_ENA 写 1 使能 MCPWM_FAULT2_INT。 (R/W)

MCPWM_FAULT0_CLR_INT_ENA 写 1 使能 MCPWM_FAULT0_CLR_INT。 (R/W)

MCPWM_FAULT1_CLR_INT_ENA 写 1 使能 MCPWM_FAULT1_CLR_INT。 (R/W)

MCPWM_FAULT2_CLR_INT_ENA 写 1 使能 MCPWM_FAULT2_CLR_INT。 (R/W)

MCPWM_CMPRO_TEA_INT_ENA 写 1 使能 MCPWM_CMPRO_TEA_INT。 (R/W)

MCPWM_CMPR1_TEA_INT_ENA 写 1 使能 MCPWM_CMPR1_TEA_INT。 (R/W)

MCPWM_CMPR2_TEA_INT_ENA 写 1 使能 MCPWM_CMPR2_TEA_INT。 (R/W)

见下页...

Register 31.29. MCPWM_INT_ENA_REG (0x0110)

... 接上页

MCPWM_CMPRO_TEB_INT_ENA 写 1 使能 [MCPWM_CMPRO_TEB_INT](#)。(R/W)

MCPWM_CMPR1_TEB_INT_ENA 写 1 使能 [MCPWM_CMPR1_TEB_INT](#)。(R/W)

MCPWM_CMPR2_TEB_INT_ENA 写 1 使能 [MCPWM_CMPR2_TEB_INT](#)。(R/W)

MCPWM_TZO_CBC_INT_ENA 写 1 使能 [MCPWM_TZO_CBC_INT](#)。(R/W)

MCPWM_TZ1_CBC_INT_ENA 写 1 使能 [MCPWM_TZ1_CBC_INT](#)。(R/W)

MCPWM_TZ2_CBC_INT_ENA 写 1 使能 [MCPWM_TZ2_CBC_INT](#)。(R/W)

MCPWM_TZO_OST_INT_ENA 写 1 使能 [MCPWM_TZO_OST_INT](#)。(R/W)

MCPWM_TZ1_OST_INT_ENA 写 1 使能 [MCPWM_TZ1_OST_INT](#)。(R/W)

MCPWM_TZ2_OST_INT_ENA 写 1 使能 [MCPWM_TZ2_OST_INT](#)。(R/W)

MCPWM_CAPO_INT_ENA 写 1 使能 [MCPWM_CAPO_INT](#)。(R/W)

MCPWM_CAP1_INT_ENA 写 1 使能 [MCPWM_CAP1_INT](#)。(R/W)

MCPWM_CAP2_INT_ENA 写 1 使能 [MCPWM_CAP2_INT](#)。(R/W)

Register 31.30. MCPWM_INT_RAW_REG (0x0114)

(reserved) MCPWM_CAP2_INT_RAW MCPWM_CAP1_INT_RAW MCPWM_CAPO_INT_RAW MCPWM_TZ2_OST_INT_RAW MCPWM_TZ1_OST_INT_RAW MCPWM_TZO_OST_INT_RAW MCPWM_TZ2_CBC_INT_RAW MCPWM_TZ1_CBC_INT_RAW MCPWM_TZO_CBC_INT_RAW MCPWM_CMPR2_INT_RAW MCPWM_CMPR1_TEB_INT_RAW MCPWM_CMPRO_TEB_INT_RAW MCPWM_CMPR2_TEA_INT_RAW MCPWM_CMPRO_TEA_INT_RAW MCPWM_FAULT2_CLR_INT_RAW MCPWM_FAULT1_CLR_INT_RAW MCPWM_FAULTO_CLR_INT_RAW MCPWM_FAULT2_INT_RAW MCPWM_FAULT1_INT_RAW MCPWM_TIMER2_TEP_INT_RAW MCPWM_TIMER1_TEP_INT_RAW MCPWM_TIMER2_TEZ_INT_RAW MCPWM_TIMER1_TEZ_INT_RAW MCPWM_TIMER2_STOP_INT_RAW MCPWM_TIMER1_STOP_INT_RAW MCPWM_TIMERO_STOP_INT_RAW MCPWM_TIMERO_STOP_INT_RAW																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_RAW MCPWM_TIMER0_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW MCPWM_TIMER1_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW MCPWM_TIMER2_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW MCPWM_TIMER0_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW MCPWM_TIMER1_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW MCPWM_TIMER2_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW MCPWM_TIMER0_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW MCPWM_TIMER1_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW MCPWM_TIMER2_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_INT_RAW MCPWM_FAULT0_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_INT_RAW MCPWM_FAULT1_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_INT_RAW MCPWM_FAULT2_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW MCPWM_FAULT0_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW MCPWM_FAULT1_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_CLR_INT_RAW MCPWM_FAULT2_CLR_INT 的原始状态位。(R/WTC/SS)

见下页...

Register 31.30. MCPWM_INT_RAW_REG (0x0114)

... 接上页

MCPWM_CMPRO_TEA_INT_RAW [MCPWM_CMPRO_TEA_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW [MCPWM_CMPR1_TEA_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW [MCPWM_CMPR2_TEA_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CMPRO_TEB_INT_RAW [MCPWM_CMPRO_TEB_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW [MCPWM_CMPR1_TEB_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW [MCPWM_CMPR2_TEB_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_TZO_CBC_INT_RAW [MCPWM_TZO_CBC_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW [MCPWM_TZ1_CBC_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW [MCPWM_TZ2_CBC_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_TZO_OST_INT_RAW [MCPWM_TZO_OST_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW [MCPWM_TZ1_OST_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_TZ2_OST_INT_RAW [MCPWM_TZ2_OST_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CAPO_INT_RAW [MCPWM_CAPO_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CAP1_INT_RAW [MCPWM_CAP1_INT](#) 的原始状态位。(R/WTC/SS)

MCPWM_CAP2_INT_RAW [MCPWM_CAP2_INT](#) 的原始状态位。(R/WTC/SS)

Register 31.31. MCPWM_INT_ST_REG (0x0118)

(reserved) MCPWM_CAP2_INT_ST MCPWM_CAP1_INT_ST MCPWM_CAPO_INT_ST MCPWM_TZ2_OST_ST MCPWM_TZ1_OST_INT_ST MCPWM_TZ0_OST_INT_ST MCPWM_TZ2_CBC_INT_ST MCPWM_TZ1_CBC_INT_ST MCPWM_TZ0_CBC_INT_ST MCPWM_CMPR2_TEB_INT_ST MCPWM_CMPR1_TEB_INT_ST MCPWM_CMPR2_TEA_INT_ST MCPWM_CMPR1_TEA_INT_ST MCPWM_FAULT2_CLR_INT_ST MCPWM_FAULT1_CLR_INT_ST MCPWM_FAULT0_CLR_INT_ST MCPWM_FAULT2_INT_ST MCPWM_FAULT1_INT_ST MCPWM_FAULT0_INT_ST MCPWM_TIMER2_TEP_INT_ST MCPWM_TIMER1_TEP_INT_ST MCPWM_TIMER2_TEZ_INT_ST MCPWM_TIMER1_TEZ_INT_ST MCPWM_TIMER2_STOP_INT_ST MCPWM_TIMER1_STOP_INT_ST MCPWM_TIMER0_STOP_INT_ST																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_ST MCPWM_TIMER0_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_STOP_INT_ST MCPWM_TIMER1_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_STOP_INT_ST MCPWM_TIMER2_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEZ_INT_ST MCPWM_TIMER0_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEZ_INT_ST MCPWM_TIMER1_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEZ_INT_ST MCPWM_TIMER2_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEP_INT_ST MCPWM_TIMER0_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEP_INT_ST MCPWM_TIMER1_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEP_INT_ST MCPWM_TIMER2_TEP_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_INT_ST MCPWM_FAULT0_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_INT_ST MCPWM_FAULT1_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_INT_ST MCPWM_FAULT2_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_CLR_INT_ST MCPWM_FAULT0_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_CLR_INT_ST MCPWM_FAULT1_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_CLR_INT_ST MCPWM_FAULT2_CLR_INT 的屏蔽状态位。(RO)

MCPWM_CMPRO_TEA_INT_ST MCPWM_CMPRO_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEA_INT_ST MCPWM_CMPR1_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEA_INT_ST MCPWM_CMPR2_TEA_INT 的屏蔽状态位。(RO)

见下页...

Register 31.31. MCPWM_INT_ST_REG (0x0118)

... 接上页

MCPWM_CMPRO_TEB_INT_ST [MCPWM_CMPRO_TEB_INT](#) 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEB_INT_ST [MCPWM_CMPR1_TEB_INT](#) 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEB_INT_ST [MCPWM_CMPR2_TEB_INT](#) 的屏蔽状态位。(RO)

MCPWM_TZO_CBC_INT_ST [MCPWM_TZO_CBC_INT](#) 的屏蔽状态位。(RO)

MCPWM_TZ1_CBC_INT_ST [MCPWM_TZ1_CBC_INT](#) 的屏蔽状态位。(RO)

MCPWM_TZ2_CBC_INT_ST [MCPWM_TZ2_CBC_INT](#) 的屏蔽状态位。(RO)

MCPWM_TZO_OST_INT_ST [MCPWM_TZO_OST_INT](#) 的屏蔽状态位。(RO)

MCPWM_TZ1_OST_INT_ST [MCPWM_TZ1_OST_INT](#) 的屏蔽状态位。(RO)

MCPWM_TZ2_OST_INT_ST [MCPWM_TZ2_OST_INT](#) 的屏蔽状态位。(RO)

MCPWM_CAPO_INT_ST [MCPWM_CAPO_INT](#) 的屏蔽状态位。(RO)

MCPWM_CAP1_INT_ST [MCPWM_CAP1_INT](#) 的屏蔽状态位。(RO)

MCPWM_CAP2_INT_ST [MCPWM_CAP2_INT](#) 的屏蔽状态位。(RO)

Register 31.32. MCPWM_INT_CLR_REG (0x011C)

(reserved) MCPWM_CAP2_INT_CLR MCPWM_CAP1_INT_CLR MCPWM_CAP0_INT_CLR MCPWM_TZ2_OST_CLR MCPWM_TZ1_OST_CLR MCPWM_TZ0_OST_INT_CLR MCPWM_TZ2_OST_INT_CLR MCPWM_TZ1_OST_INT_CLR MCPWM_TZ0_OST_INT_CLR MCPWM_CMPR2_TEB_INT_CLR MCPWM_CMPR1_TEB_INT_CLR MCPWM_CMPR0_TEB_INT_CLR MCPWM_FAULT2_TEA_INT_CLR MCPWM_FAULT1_TEA_INT_CLR MCPWM_FAULT0_TEA_INT_CLR MCPWM_TIMER2_TEP_INT_CLR MCPWM_TIMER1_TEP_INT_CLR MCPWM_TIMER0_TEP_INT_CLR MCPWM_TIMER2_TEZ_INT_CLR MCPWM_TIMER1_TEZ_INT_CLR MCPWM_TIMER0_TEZ_INT_CLR MCPWM_TIMER1_STOP_INT_CLR MCPWM_TIMER0_STOP_INT_CLR																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_TIMER0_STOP_INT_CLR 写 1 清除 MCPWM_TIMER0_STOP_INT。(WT)

MCPWM_TIMER1_STOP_INT_CLR 写 1 清除 MCPWM_TIMER1_STOP_INT。(WT)

MCPWM_TIMER2_STOP_INT_CLR 写 1 清除 MCPWM_TIMER2_STOP_INT。(WT)

MCPWM_TIMER0_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER0_TEZ_INT。(WT)

MCPWM_TIMER1_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER1_TEZ_INT。(WT)

MCPWM_TIMER2_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER2_TEZ_INT。(WT)

MCPWM_TIMER0_TEP_INT_CLR 写 1 清除 MCPWM_TIMER0_TEP_INT。(WT)

MCPWM_TIMER1_TEP_INT_CLR 写 1 清除 MCPWM_TIMER1_TEP_INT。(WT)

MCPWM_TIMER2_TEP_INT_CLR 写 1 清除 MCPWM_TIMER2_TEP_INT。(WT)

MCPWM_FAULT0_INT_CLR 写 1 清除 MCPWM_FAULT0_INT。(WT)

MCPWM_FAULT1_INT_CLR 写 1 清除 MCPWM_FAULT1_INT。(WT)

MCPWM_FAULT2_INT_CLR 写 1 清除 MCPWM_FAULT2_INT。(WT)

MCPWM_FAULT0_CLR_INT_CLR 写 1 清除 MCPWM_FAULT0_CLR_INT。(WT)

MCPWM_FAULT1_CLR_INT_CLR 写 1 清除 MCPWM_FAULT1_CLR_INT。(WT)

MCPWM_FAULT2_CLR_INT_CLR 写 1 清除 MCPWM_FAULT2_CLR_INT。(WT)

见下页...

Register 31.32. MCPWM_INT_CLR_REG (0x011C)

... 接上页

MCPWM_CMPRO_TEA_INT_CLR 写 1 清除 [MCPWM_CMPRO_TEA_INT](#)。(WT)

MCPWM_CMPR1_TEA_INT_CLR 写 1 清除 [MCPWM_CMPR1_TEA_INT](#)。(WT)

MCPWM_CMPR2_TEA_INT_CLR 写 1 清除 [MCPWM_CMPR2_TEA_INT](#)。(WT)

MCPWM_CMPRO_TEB_INT_CLR 写 1 清除 [MCPWM_CMPRO_TEB_INT](#)。(WT)

MCPWM_CMPR1_TEB_INT_CLR 写 1 清除 [MCPWM_CMPR1_TEB_INT](#)。(WT)

MCPWM_CMPR2_TEB_INT_CLR 写 1 清除 [MCPWM_CMPR2_TEB_INT](#)。(WT)

MCPWM_TZO_CBC_INT_CLR 写 1 清除 [MCPWM_TZO_CBC_INT](#)。(WT)

MCPWM_TZ1_CBC_INT_CLR 写 1 清除 [MCPWM_TZ1_CBC_INT](#)。(WT)

MCPWM_TZ2_CBC_INT_CLR 写 1 清除 [MCPWM_TZ2_CBC_INT](#)。(WT)

MCPWM_TZO_OST_INT_CLR 写 1 清除 [MCPWM_TZO_OST_INT](#)。(WT)

MCPWM_TZ1_OST_INT_CLR 写 1 清除 [MCPWM_TZ1_OST_INT](#)。(WT)

MCPWM_TZ2_OST_INT_CLR 写 1 清除 [MCPWM_TZ2_OST_INT](#)。(WT)

MCPWM_CAPO_INT_CLR 写 1 清除 [MCPWM_CAPO_INT](#)。(WT)

MCPWM_CAP1_INT_CLR 写 1 清除 [MCPWM_CAP1_INT](#)。(WT)

MCPWM_CAP2_INT_CLR 写 1 清除 [MCPWM_CAP2_INT](#)。(WT)

Register 31.33. MCPWM_EVT_EN_REG (0x0120)

(reserved) MCPWM_EVT_CAP2_EN MCPWM_EVT_CAP1_EN MCPWM_EVT_CAP0_EN MCPWM_EVT_TZ2_OST_EN MCPWM_EVT_TZ1_OST_EN MCPWM_EVT_TZ0_OST_EN MCPWM_EVT_TZ2_CBC_EN MCPWM_EVT_TZ1_CBC_EN MCPWM_EVT_TZ0_CBC_EN MCPWM_EVT_F2_CLR_EN MCPWM_EVT_F1_CLR_EN MCPWM_EVT_F0_CLR_EN MCPWM_EVT_F2_EN MCPWM_EVT_F1_EN MCPWM_EVT_F0_EN MCPWM_EVT_OP2_TEB_EN MCPWM_EVT_OP1_TEB_EN MCPWM_EVT_OP0_TEB_EN MCPWM_EVT_OP2_TEA_EN MCPWM_EVT_OP1_TEA_EN MCPWM_EVT_OP0_TEA_EN MCPWM_EVT_TIMER2_TEP_EN MCPWM_EVT_TIMER1_TEP_EN MCPWM_EVT_TIMER2_TEZ_EN MCPWM_EVT_TIMER1_TEZ_EN MCPWM_EVT_TIMER2_STOP_EN MCPWM_EVT_TIMER1_STOP_EN MCPWM_EVT_TIMER0_STOP_EN																																
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

MCPWM_EVT_TIMER0_STOP_EN 配置是否使能当定时器 0 停止时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER1_STOP_EN 配置是否使能当定时器 1 停止时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER2_STOP_EN 配置是否使能当定时器 2 停止时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN 配置是否使能当定时器 0 等于零时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN 配置是否使能当定时器 1 等于零时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN 配置是否使能当定时器 2 等于零时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER0_TEP_EN 配置是否使能当定时器 0 等于周期时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TIMER1_TEP_EN 配置是否使能当定时器 1 等于周期时的事件生成。

0: 关闭

1: 使能

(R/W)

见下页...

Register 31.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TIMER2_TEP_EN 配置是否使能当定时器 2 等于周期时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_OPO_TEA_EN 配置是否使能当生成器 0 等于时间戳 A 时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_OP1_TEA_EN 配置是否使能当生成器 1 等于时间戳 A 时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_OP2_TEA_EN 配置是否使能当生成器 2 等于时间戳 A 时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_OPO_TEB_EN 配置是否使能当生成器 0 等于时间戳 B 时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_OP1_TEB_EN 配置是否使能当生成器 1 等于时间戳 B 时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_OP2_TEB_EN 配置是否使能当生成器 2 等于时间戳 B 时的事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_F0_EN 配置是否使能 FAULT0 事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_F1_EN 配置是否使能 FAULT1 事件生成。

0: 关闭

1: 使能

(R/W)

见下页...

Register 31.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_F2_EN 配置是否使能 FAULT2 事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_F0_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_F1_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_F2_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TZO_CBC_EN 配置是否使能 CBC0 事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TZ1_CBC_EN 配置是否使能 CBC1 事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TZ2_CBC_EN 配置是否使能 CBC2 事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TZO_OST_EN 配置是否使能 OST0 事件生成。

0: 关闭

1: 使能

(R/W)

MCPWM_EVT_TZ1_OST_EN 配置是否使能 OST1 事件生成。

0: 关闭

1: 使能

(R/W)

见下页...

Register 31.33. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TZ2_OST_EN 配置是否使能 OST2 事件生成。

0: 关闭
1: 使能
(R/W)

MCPWM_EVT_CAO0_EN 配置是否使能 capture0 事件生成。

0: 关闭
1: 使能
(R/W)

MCPWM_EVT_CAP1_EN 配置是否使能 capture1 事件生成。

0: 关闭
1: 使能
(R/W)

MCPWM_EVT_CAP2_EN 配置是否使能 capture2 事件生成。

0: 关闭
1: 使能
(R/W)

Register 31.34. MCPWM_EVT_EN2_REG (0x0128)

(reserved)																								MCPWM_EVT_OP2_TEE2_EN MCPWM_EVT_OP1_TEE2_EN MCPWM_EVT_OP0_TEE2_EN MCPWM_EVT_OP2_TEE1_EN MCPWM_EVT_OP1_TEE1_EN MCPWM_EVT_OP0_TEE1_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
31																								6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

MCPWM_EVT_OPn_TEE1_EN 配置是否在 PWM 生成器计数器的值等于 MCPWM_OPn_TSTMP_E1_REG 寄存器的值时，生成 MCPWM_EVT_OPn_TEE1 事件。

0: 不生成
1: 生成
(R/W)

MCPWM_EVT_OPn_TEE2_EN 配置是否在 PWM 生成器计数器的值等于 MCPWM_OPn_TSTMP_E2_REG 寄存器的值时，生成 MCPWM_EVT_OPn_TEE2 事件。

0: 不生成
1: 生成
(R/W)

Register 31.35. MCPWM_TASK_EN_REG (0x0124)

(reserved)																																		
31											22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset		
										MCPWM_TASK_CAP2_EN MCPWM_TASK_CAP1_EN MCPWM_TASK_CAPO_EN MCPWM_TASK_CLR2_OST_EN MCPWM_TASK_CLR1_OST_EN MCPWM_TASK_TZ2_OST_EN MCPWM_TASK_TZ1_OST_EN MCPWM_TASK_TZO_OST_EN MCPWM_TASK_TIMER2_PERIOD_UP_EN MCPWM_TASK_TIMER1_PERIOD_UP_EN MCPWM_TASK_TIMER2_SYNC_EN MCPWM_TASK_TIMER1_SYNC_EN MCPWM_TASK_CMPR2_B_UP_EN MCPWM_TASK_CMPR1_B_UP_EN MCPWM_TASK_CMPR2_A_UP_EN MCPWM_TASK_CMPR1_A_UP_EN																								

MCPWM_TASK_CMPRO_A_UP_EN 配置是否接收 PWM 生成器 0 时间戳 A 的影子寄存器的更新任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN	配置是否接收 PWM 生成器 1 时间戳 A 的影子寄存器的更新任务。
0: 无效	
1: 接收	
(R/W)	

MCPWM_TASK_CMPR2_A_UP_EN	配置是否接收 PWM 生成器 2 时间戳 A 的影子寄存器的更新任务。
0: 无效	
1: 接收	
(R/W)	

MCPWM_TASK_CMPRO_B_UP_EN	配置是否接收 PWM 生成器 O 时间戳 B 的影子寄存器的更新任务。
0:	无效
1:	接收
	(R/W)

MCPWM_TASK_CMPR1_B_UP_EN	配置是否接收 PWM 生成器 1 时间戳 B 的影子寄存器的更新任务。
0: 无效	
1: 接收	
(R/W)	

MCPWM_TASK_CMPR2_B_UP_EN	配置是否接收 PWM 生成器 2 时间戳 B 的影子寄存器的更新任务。
0:	无效
1:	接收
	(R/W)

见下页...

Register 31.35. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_GEN_STOP_EN 配置是否接收所有 PWM 生成器的停止任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TIMER0_SYNC_EN 配置是否接收定时器 0 的同步任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TIMER1_SYNC_EN 配置是否接收定时器 1 的同步任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TIMER2_SYNC_EN 配置是否接收定时器 2 的同步任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN 配置是否接收定时器 0 的周期更新任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN 配置是否接收定时器 1 的周期更新任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN 配置是否接收定时器 2 的周期更新任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TZO_OST_EN 配置是否接收 OST0 任务。

0: 无效

1: 接收

(R/W)

MCPWM_TASK_TZ1_OST_EN 配置是否接收 OST1 任务。

0: 无效

1: 接收

(R/W)

见下页...

Register 31.35. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_TZ2_OST_EN 配置是否接收 OST2 任务。

0: 无效
1: 接收
(R/W)

MCPWM_TASK_CLRO_OST_EN 配置是否接收 OST0 清除任务。

0: 无效
1: 接收
(R/W)

MCPWM_TASK_CLR1_OST_EN 配置是否接收 OST1 清除任务。

0: 无效
1: 接收
(R/W)

MCPWM_TASK_CLR2_OST_EN 配置是否接收 OST2 清除任务。

0: 无效
1: 接收
(R/W)

MCPWM_TASK_CAPO_EN 配置是否接收 capture0 任务。

0: 无效
1: 接收
(R/W)

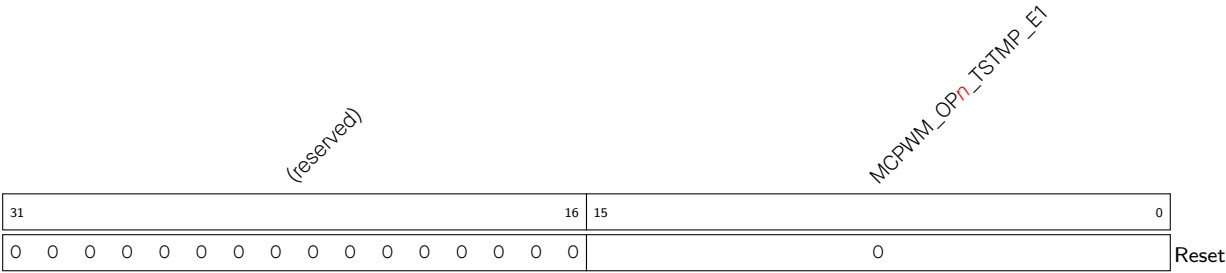
MCPWM_TASK_CAP1_EN 配置是否接收 capture1 任务。

0: 无效
1: 接收
(R/W)

MCPWM_TASK_CAP2_EN 配置是否接收 capture2 任务。

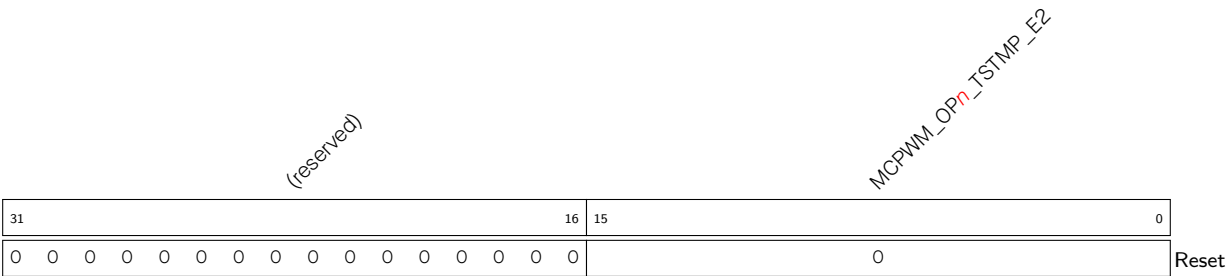
0: 无效
1: 接收
(R/W)

Register 31.36. MCPWM_OPn_TSTMP_E1_REG (n: 0-2) (0x012C+0x8*n)



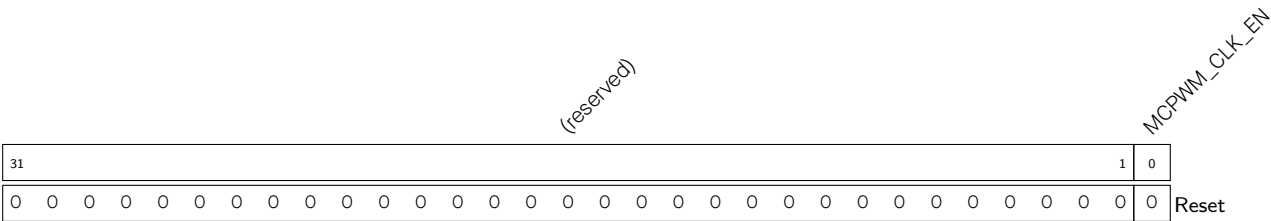
MCPWM_OPn_TSTMP_E1 配置生成器n 时间戳 E1 值。(R/W)

Register 31.37. MCPWM_OPn_TSTMP_E2_REG(n: 0-2) (0x0130+0x8*n)



MCPWM_OPn_TSTMP_E2 配置生成器n 时间戳 E2 值。(R/W)

Register 31.38. MCPWM_CLK_REG (0x0144)



MCPWM_CLK_EN 配置是否强制使能时钟。

0：无效

1：强制使能时钟

(R/W)

Register 31.39. MCPWM_VERSION_REG (0x0148)

(reserved)				MCPWM_DATE																									
31	28	27																											0
0	0	0	0	0x2107230																									

Reset

MCPWM_DATE 版本控制寄存器。(R/W)

第 32 章

红外遥控 (RMT)

32.1 概述

红外遥控 (RMT) 是一个红外发送和接收控制器，可通过软件编解码多种红外协议。RMT 模块可以将模块内置 RAM 中的脉冲编码转换为信号输出，或将模块的输入信号转换为脉冲编码存入 RAM 中。此外，RMT 模块也支持对输出信号进行载波调制或对输入信号进行解调和去噪处理。

RMT 共有四个通道，编码为 0 ~ 3，各通道可独立用于发送或接收信号：

- 0 ~ 1 通道专门用于信号发送；
- 2 ~ 3 通道专门用于信号接收。

每个发送通道和接收通道分别有一组功能相同的寄存器。为了方便叙述，以 n 表示各个发送通道，以 m 表示各个接收通道。

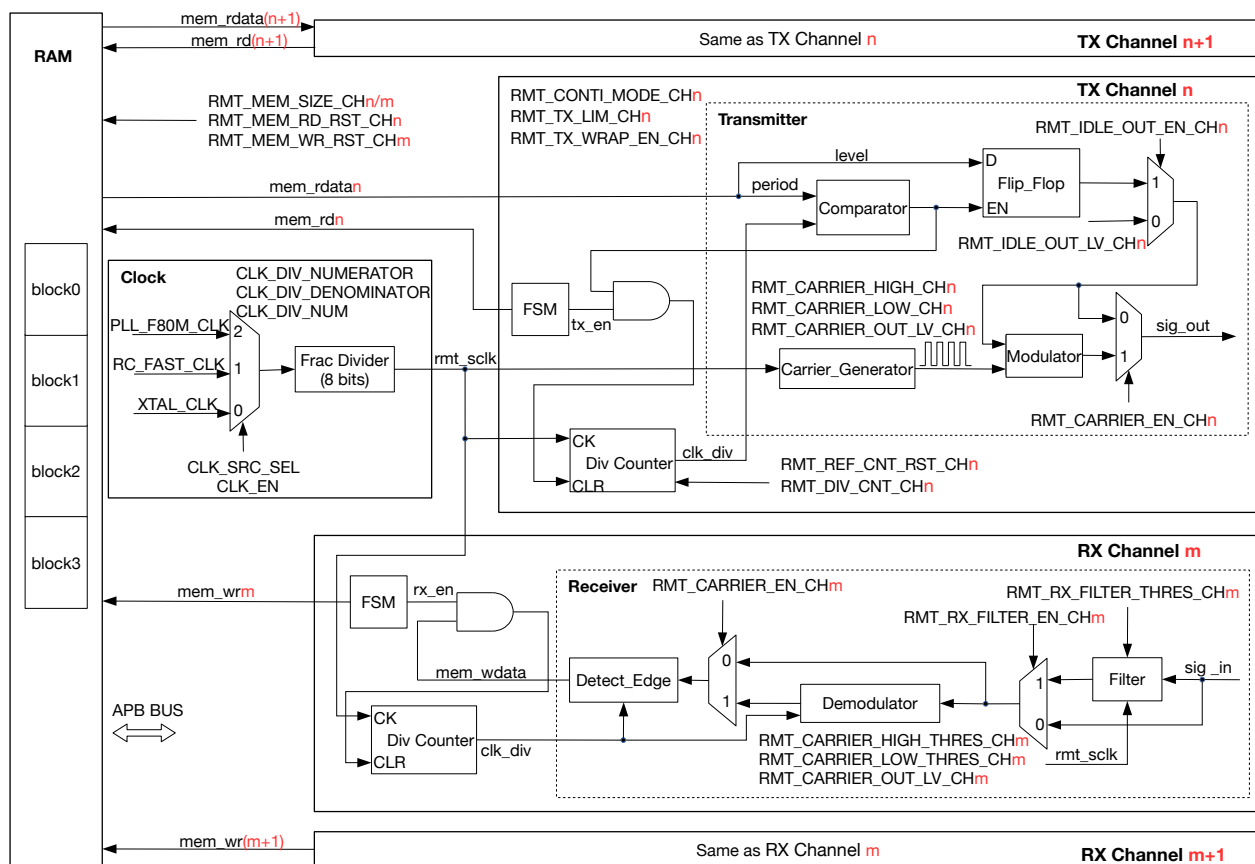
32.2 主要特性

RMT 控制器具有如下特性：

- 共配置四个通道：
 - 0 ~ 1 通道支持发送
 - 2 ~ 3 通道支持接收
 - 四个通道共享 192 x 32 位的 RAM
- 发射器支持以下模式：
 - 普通发送模式
 - 乒乓发送模式
 - 载波调制
 - 持续发送模式
 - 多通道同时发送
- 接收器支持以下模式：
 - 普通接收模式
 - 乒乓接收模式
 - 接收滤波
 - 载波解调

32.3 功能描述

32.3.1 RMT 架构



注：上图中， $n = 0$ ， $m = 2$ 。

图 32.3-1. RMT 结构框图

如图 32.3-1 所示，每个发送通道内部各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个发射器 (Transmitter)

每个接收通道内部也各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个接收器 (Receiver)

四个通道共享一块 192 x 32 位的 RAM。

图中“Clock”部分需要配置的 RMT 时钟相关寄存器具体如下：

- CLK_EN: [PCR_RMT_SCLK_EN](#)，用于使能 rmt_sclk 时钟

- CLK_SRC_SEL: `PCR_RMT_SCLK_SEL`, 用于选择 `rmt_sclk` 时钟
- CLK_DIV_NUM: `PCR_RMT_SCLK_DIV_NUM`, 用于配置 `rmt_sclk` 时钟的分频系数整数部分
- CLK_DIV_NUMERATOR: `PCR_RMT_SCLK_DIV_A`, 用于配置 `rmt_sclk` 时钟分频系数小数部分的分子
- CLK_DIV_DENOMINATOR: `PCR_RMT_SCLK_DIV_B`, 用于配置 `rmt_sclk` 时钟分频系数小数部分的分母

32.3.2 RAM

32.3.2.1 RAM 结构

RAM 中脉冲编码结构如图 32.3-2 所示。每个脉冲编码为 16 位, 由 level 与 period 两部分组成。其中 level 表示输入或输出信号的逻辑电平值 (0 或 1), period 表示该电平信号持续的时钟 (见图 32.3-1 中的 `clk_div`) 周期数。

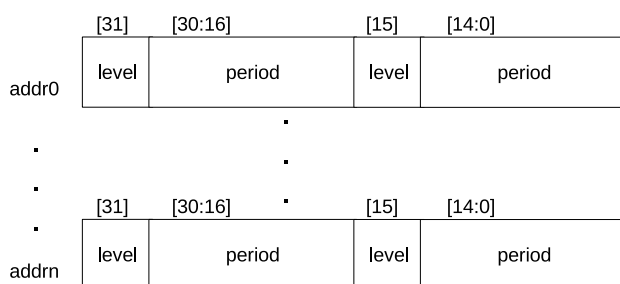


图 32.3-2. RAM 中脉冲编码结构

period 的最小值为 0, period 为 0 是一次传输的结束标志。对于非 0 的 period, 即非结束标志的 period, 其值需要满足与 APB 时钟和 RMT 时钟相关的限制条件, 具体请参考以下公式:

$$3 \times T_{apb\_clk} + 5 \times T_{rmt\_sclk} < period \times T_{clk\_div} \quad (32.1)$$

32.3.2.2 RAM 使用说明

RAM 按照 48 x 32 位分成四个 block。默认情况下, 每个通道只能使用一个 block (固定为通道 0 使用 block 0, 通道 1 使用 block 1, 以此类推)。

当发送通道 *n* 或接收通道 *m* 单次传输的脉冲编码数大于一个 block 时, 可以:

- 置位 `RMT_MEM_TX/RX_WRAP_EN_CHn/m` 使能乒乓操作;
- 或通过配置 `RMT_MEM_SIZE_CHn/m` 寄存器, 允许该通道占用多个 block。

当设置 `RMT_MEM_SIZE_CHn/m` > 1 时, 通道 *n/m* 将占用 block (*n/m*) ~ block (*n/m*+`RMT_MEM_SIZE_CHn/m`-1) 的存储空间, 此时通道 *n/m*+1 ~ *n/m*+`RMT_MEM_SIZE_CHn/m`-1 因为对应的 RAM block 被占用而无法使用。例如, 如果通道 0 配置使用了 block 0 和 block 1, 则通道 1 无法使用, 通道 2 和通道 3 不受影响。

注意, 每个通道使用 RAM 的空间是根据地址从低到高进行映射的, 因此通道 0 可以通过配置 `RMT_MEM_SIZE_CHO` 寄存器来使用通道 1、2、3 的 RAM 空间, 但是通道 3 不能使用通道 0、1 或 2 的 RAM 空间。因此 `RMT_MEM_SIZE_CHn` 的最大值不应超过 (4-*n*), `RMT_MEM_SIZE_CHm` 的最大值不应超过 (2-*m*)。

RAM 可被 APB 总线及通道的发射器或接收器访问。为了防止接收通道访问 RAM 和 APB 访问时发生冲突（例如 APB 读 RAM 时，通道向 RAM 发起写操作），可以通过配置 `RMT_MEM_OWNER_CHm` 来决定当前 RAM 的使用权。当接收通道发生越权访问时会产生 `RMT_MEM_OWNER_ERR_CHm` 标志信号。

32.3.2.3 RAM 访问方式

APB 总线访问 RAM 有 FIFO 和 NONFIFO（直接地址）两种模式：

- `RMT_APB_FIFO_MASK` 置 0 时，选择 FIFO 模式；
- `RMT_APB_FIFO_MASK` 置 1 时，选择 NONFIFO 模式。

FIFO 模式

在 FIFO 模式下，APB 通过固定地址 `RMT_CHn/mDATA_REG` 向 RAM 写数据或从 RAM 读数据。

NONFIFO 模式

在 NONFIFO 模式下，APB 向连续地址段写入数据，或从连续地址段读出数据：

- 发送通道 *n* 对应的写地址段的首地址是 (RMT 基地址 + 0x400 + (n-1) x 48)，第 2 个数据的访问地址是 (RMT 基地址 + 0x400 + (n-1) x 48 + 0x4)，以此类推，后面的地址依次加上 0x4。
- 接收通道 *m* 对应的读地址段的首地址是 (RMT 基地址 + 0x460 + (m-1) x 48)，第 2 个数据的访问地址是 (RMT 基地址 + 0x460 + (m-1) x 48 + 0x4)，以此类推，后面的地址依次加上 0x4。

32.3.3 时钟

用户可以通过配置 `PCR_RMT_SCLK_SEL` 选择 RMT 的时钟源：PLL_F80M_CLK、RC_FAST_CLK 或 XTAL_CLK，配置 `PCR_RMT_SCLK_EN` 为高电平来打开 RMT 的时钟。选择后的时钟经过小数分频得到 RMT 的工作时钟（见图 32.3-1 中的 `rmt_sclk`），分频系数为 `PCR_RMT_SCLK_DIV_NUM + 1 + PCR_RMT_SCLK_DIV_A / PCR_RMT_SCLK_DIV_B`。

更多信息，请参考章节 6 复位和时钟。`RMT_DIV_CNT_CHn/m` 用于配置 RMT 通道内部的时钟分频器的分频系数，除 0 表示 256 分频外，其他分频数等同于寄存器 `RMT_DIV_CNT_CHn/m` 的值。时钟分频器可以通过配置 `RMT_REF_CNT_RST_CHn/m` 进行复位。时钟分频器的分频时钟可供计数器使用，见图 32.3-1。

32.3.4 发射器

说明：

本小节以及后续小节所述的配置，均需要通过置位 `RMT_CONF_UPDATE_CHn/m` 的方式来进行更新，详情见第 32.4 小节。

32.3.4.1 普通发送模式

当 `RMT_TX_START_CHn` 置为 1 时，通道 *n* 的发射器开始从通道对应 RAM block 的起始地址，按照地址从低到高依次读取脉冲编码进行发送。当遇到结束标志（period 等于 0）时，发射器将结束发送返回空闲状态，并产生 `RMT_CHn_TX_END_INT` 中断。配置 `RMT_TX_STOP_CHn` 可以使发射器立刻停止发送并进入空闲状态。发射器空闲状态发送的电平由结束标志中的 level 段或者是 `RMT_IDLE_OUT_LV_CHn` 决定。用户可以配置 `RMT_IDLE_OUT_EN_CHn` 来选择这两种方式：

- `RMT_IDLE_OUT_EN_CHn` 置 0 时，发射器空闲状态发送的电平由结束标志中的 level 段决定；

- `RMT_IDLE_OUT_EN_CHn` 置 1 时, 发射器空闲状态发送的电平由 `RMT_IDLE_OUT_LV_CHn` 决定。

32.3.4.2 乒乓发送模式

当发送的脉冲编码较多时, 可通过置位 `RMT_MEM_TX_WRAP_EN_CHn` 使能通道 n 的乒乓模式。在乒乓操作模式下, 发射器会循环从通道对应的 RAM 区域取出脉冲编码进行发送, 直至遇到结束标识为止。例如, 当 `RMT_MEM_SIZE_CHn` = 1 时, 发射器将从 $48 * n$ 地址开始发送, 然后对应 RAM 的地址递增。发完 $(48 * (n+1)-1)$ 地址的数据后, 下次继续从 $48 * n$ 地址开始递增发送数据, 依此类推, 遇到结束标识时停止发送。`RMT_MEM_SIZE_CHn` > 1 的情形下, 乒乓操作同样适用。

每当发射器发送的脉冲编码数大于等于 `RMT_TX_LIM_CHn` 时, 会产生 `RMT_CHn_TX_THR_EVENT_INT` 中断。在乒乓模式下, 可以设置 `RMT_TX_LIM_CHn` 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 `RMT_CHn_TX_THR_EVENT_INT` 中断之后, 可以回收已使用过的 RAM 区域的脉冲编码, 从而实现乒乓操作。

32.3.4.3 发送加载波

此外, 发射器还可以对输出信号进行载波调制, 置位 `RMT_CARRIER_EN_CHn` 可以使能该功能。载波的波形可配置。一个载波周期中高电平持续时间为 $(RMT\_CARRIER\_HIGH\_CHn + 1)$ 个 `rmt_sclk` 时钟周期, 低电平持续的时间为 $(RMT\_CARRIER\_LOW\_CHn + 1)$ 个 `rmt_sclk` 时钟周期。置位 `RMT_CARRIER_OUT_LV_CHn` 时在输出信号高电平上加载波信号, 清零 `RMT_CARRIER_OUT_LV_CHn` 时在输出信号低电平上加载波信号。同时, 在进行载波调制时, 载波可以一直加载在输出信号上, 也可以仅加载在有效的脉冲编码 (RAM 中的数据) 上。通过配置 `RMT_CARRIER_EFF_EN_CHn` 寄存器, 可以选择这两种模式:

- `RMT_CARRIER_EFF_EN_CHn` 置 0 时, 在所有信号上加载波。
- `RMT_CARRIER_EFF_EN_CHn` 置 1 时, 在有效信号上加载波。

32.3.4.4 持续发送模式

置位 `RMT_TX_CONTI_MODE_CHn` 可以使能发射器的持续发送功能。置位后, 发射器会循环发送 RAM 中的脉冲编码。持续发送模式下:

- 如果遇到结束标志, 则重新从该通道 RAM 中的第一个数据开始发送;
- 如果发送到 RAM 中的最后一个数都没有结束标志, 存在两种情况。普通发送模式下 (`RMT_MEM_TX_WRAP_EN_CHn` = 0), 会因为 RAM 被读空没有可以继续发送的数而产生错误中断; 乒乓发送模式下 (`RMT_MEM_TX_WRAP_EN_CHn` = 1), 会在发送到最后一个数据处回卷, 重新开始发送第一个数据。

配置 `RMT_TX_LOOP_CNT_EN_CHn` 后, 发射器每遇到一次结束标志, 循环发送的次数会加 1。当该次数达到 `RMT_TX_LOOP_NUM_CHn` 设定的值时, 会产生 `RMT_CHn_TX_LOOP_INT` 中断。如果置位 `RMT_LOOP_STOP_EN_CHn`, 则发送会在产生 `RMT_CHn_TX_LOOP_INT` 中断后立即停止, 否则会继续发送。持续发送模式下, 如果遇到的结束标志类型是 `period[14:0]` 为 0, 那么这个结束标志前一个数据的 `period` 需要满足:

$$6 \times T_{apb\_clk} + 12 \times T_{rmt\_sclk} < period \times T_{clk\_div} \quad (32.2)$$

其他数据的 `period`, 满足 非 0 `period` 关系式 即可。

32.3.4.5 多通道同时发送

RMT 模块支持多通道同时发送，具体配置步骤如下所示：

1. 首先配置 `RMT_TX_SIM_CH $n$` ，选择同步发送的通道；
2. 然后置位 `RMT_TX_SIM_EN`，使能发射器多个通道同步发送的功能；
3. 最后将所选同步发送通道的 `RMT_TX_START_CH $n$` 置为 1。

最后一个通道完成配置时，多个通道会同时启动发送。由于硬件条件的限制，两条通道不能保证完全同时开始发送，两条通道发送的时间间隔在 $3 \times T_{clk\_div}$ 以内。

32.3.5 接收器

32.3.5.1 普通接收模式

`RMT_RX_EN_CH $m$` 置为 1 时接收器开始工作，置为 0 时会停止接收。接收器会从信号的第一个跳变沿开始计数，并检测信号电平及其持续的时钟周期数，将其按照脉冲编码的格式存入 RAM 中。当信号在一个电平下持续的时钟周期数超过 `RMT_IDLE_THRES_CH $m$` 时，接收器结束接收过程，返回空闲状态，并产生 `RMT_CH $m$ _RX_END_INT` 中断。`RMT_IDLE_THRES_CH $m$` 的值需要大于输入信号的高电平或低电平的最大时钟周期数，否则接收器会将该信号误判为空闲信号而进入空闲状态。当接收数据存满了接收通道设置的 RAM 空间时，会停止接收，在非乒乓模式下会产生 `RMT_CH $m$ _ERR_INT` 中断（由 RAM 满事件触发）。

32.3.5.2 乒乓接收模式

当接收的脉冲编码较多时，可通过置位 `RMT_MEM_RX_WRAP_EN_CH $m$` 使能通道 m 的乒乓模式。在乒乓操作模式下，接收器会将接收到的脉冲编码循环存入通道对应的 RAM 区域，当信号在一个电平下持续的时钟周期数超过 `RMT_IDLE_THRES_CH $m$` 时，接收器结束接收过程，返回空闲状态，并产生 `RMT_CH $m$ _RX_END_INT` 中断。例如，当 `RMT_MEM_SIZE_CH $m$` = 1 时，接收器将从 $48 * m$ 地址开始接收，然后对应 RAM 的地址递增。收完 $(48 * (m+1) - 1)$ 地址的数据后，下次继续从 $48 * m$ 地址开始递增接收数据，以此类推，遇到信号在一个电平下持续的时钟周期数超过 `RMT_IDLE_THRES_CH $m$` 时停止接收。`RMT_MEM_SIZE_CH $m$` > 1 的情形下，乒乓操作同样适用。

每当接收器接收的脉冲编码数大于或等于 `RMT_RX_LIM_CH $m$` 时，会产生 `RMT_CH $m$ _RX_THR_EVENT_INT` 中断。在乒乓模式下，可以设置 `RMT_RX_LIM_CH $m$` 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 `RMT_CH $m$ _RX_THR_EVENT_INT` 中断之后，可以回收已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

32.3.5.3 接收滤波

置位 `RMT_RX_FILTER_EN_CH $m$` 可使能通道 m 的接收器对输入信号进行滤波。滤波器的功能为连续采样输入信号，如果输入信号在连续 `RMT_RX_FILTER_THRES_CH $m$` 个 `rmt_sclk` 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，滤波器会滤除脉冲宽度小于 `RMT_RX_FILTER_THRES_CH $m$` 个 `rmt_sclk` 时钟周期的线路毛刺。

32.3.5.4 接收去载波

此外，接收器还可以对输入信号或滤波后的信号进行去载波调制，置位 `RMT_CARRIER_EN_CH $m$` 可以使能该功能。去载波可以对高电平调制或低电平调制的载波进行滤除：

- `RMT_CARRIER_OUT_LV_CH $m$` 置 0 时，配置滤除低电平载波；

- RMT_CARRIER_OUT_LV_CH m 置 1 时，配置滤除高电平载波。

配置 RMT_CARRIER_HIGH_THRES_CH m 和 RMT_CARRIER_LOW_THRES_CH m 可设置去载波的高电平阈值和低电平阈值。当信号的高电平持续时间小于 RMT_CARRIER_HIGH_THRES_CH m 个 clk_div 分频时钟周期，或者信号的低电平持续时间小于 RMT_CARRIER_LOW_THRES_CH m 个 clk_div 分频时钟周期，则信号会被认为是载波而被滤除。

32.4 配置参数更新

RMT 的配置寄存器均需要配置各自通道的 RMT_CONF_UPDATE_CH n/m 位来更新进入各自通道，更新方法是向 RMT_CONF_UPDATE_CH n/m 写入高电平。发送通道和接收通道需要通过这种方法更新的配置参数见表 32.4-1。

表 32.4-1. 更新配置参数

配置寄存器	配置参数
发送通道	
RMT_CH n CONFO_REG	RMT_CARRIER_OUT_LV_CH n
	RMT_CARRIER_EN_CH n
	RMT_CARRIER_EFF_EN_CH n
	RMT_DIV_CNT_CH n
	RMT_IDLE_OUT_EN_CH n
	RMT_IDLE_OUT_LV_CH n
	RMT_TX_CONTI_MODE_CH n
RMT_CH n CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH n
	RMT_CARRIER_LOW_CH n
RMT_CH n TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH n
	RMT_TX_LOOP_NUM_CH n
	RMT_TX_LIM_CH n
RMT_TX_SIM_REG	RMT_TX_SIM_EN
接收通道	
RMT_CH m CONFO_REG	RMT_CARRIER_OUT_LV_CH m
	RMT_CARRIER_EN_CH m
	RMT_IDLE_THRES_CH m
	RMT_DIV_CNT_CH m
RMT_CH m CONF1_REG	RMT_RX_FILTER_THRES_CH m
	RMT_RX_EN_CH m
RMT_CH m RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH m
	RMT_CARRIER_LOW_THRES_CH m
RMT_CH m RX_LIM_REG	RMT_RX_LIM_CH m
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH m

32.5 中断

ESP32-C5 的 RMT 控制器可以生成 RMT_INTR 中断信号并发送给 [中断矩阵](#)，该中断信号由 RMT 的内部中断源生成。

以下中断源生成 RMT_INTR 中断信号：

- RMT_CH n / m _ERR_INT：当通道 n / m 发生读写数据不正确，或内存空满错误时，即触发此中断。
- RMT_CH n _TX_THR_EVENT_INT：发射器每发送 RMT_CH n _TX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH m _RX_THR_EVENT_INT：接收器每接收 RMT_CH m _RX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH n _TX_END_INT：当发射器停止发送信号时，即触发此中断。
- RMT_CH m _RX_END_INT：当接收器停止接收信号时，即触发此中断。
- RMT_CH n _TX_LOOP_INT：发射器处于循环发送模式时，当循环次数达到 RMT_TX_LOOP_NUM_CH n 的值后，会产生此中断。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 *[to be added later]* > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [32.6 寄存器列表](#)。

32.6 寄存器列表

本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。

请查看章节 [寄存器的访问类型](#)，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 读/写寄存器			
RMT_CH0DATA_REG	通道 0 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0000	HRO
RMT_CH1DATA_REG	通道 1 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0004	HRO
RMT_CH2DATA_REG	通道 2 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0008	HRO
RMT_CH3DATA_REG	通道 3 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x000C	HRO
配置寄存器			
RMT_CHOCONF0_REG	通道 0 的配置寄存器 0	0x0010	varies
RMT_CH1CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
RMT_CH2CONF0_REG	通道 2 的配置寄存器 0	0x0018	R/W
RMT_CH2CONF1_REG	通道 2 的配置寄存器 1	0x001C	varies
RMT_CH3CONF0_REG	通道 3 的配置寄存器 0	0x0020	R/W
RMT_CH3CONF1_REG	通道 3 的配置寄存器 1	0x0024	varies
RMT_SYS_CONF_REG	RMT APB 配置寄存器	0x0068	R/W
RMT_REF_CNT_RST_REG	RMT 时钟分频器复位寄存器	0x0070	WT
状态寄存器			
RMT_CH0STATUS_REG	通道 0 的状态寄存器	0x0028	RO
RMT_CH1STATUS_REG	通道 1 的状态寄存器	0x002C	RO
RMT_CH2STATUS_REG	通道 2 的状态寄存器	0x0030	RO
RMT_CH3STATUS_REG	通道 3 的状态寄存器	0x0034	RO
中断寄存器			
RMT_INT_RAW_REG	原始中断状态寄存器	0x0038	R/ WTC/ SS
RMT_INT_ST_REG	屏蔽中断状态寄存器	0x003C	RO
RMT_INT_ENA_REG	中断使能寄存器	0x0040	R/W
RMT_INT_CLR_REG	中断清零寄存器	0x0044	WT
载波占空比寄存器			
RMT_CH0CARRIER_DUTY_REG	通道 0 的占空比配置寄存器	0x0048	R/W
RMT_CH1CARRIER_DUTY_REG	通道 1 的占空比配置寄存器	0x004C	R/W
RMT_CH2_RX_CARRIER_RM_REG	通道 2 的去载波寄存器	0x0050	R/W
RMT_CH3_RX_CARRIER_RM_REG	通道 3 的去载波寄存器	0x0054	R/W
TX 事件配置寄存器			
RMT_CHO_TX_LIM_REG	通道 0 的 TX 事件配置寄存器	0x0058	varies
RMT_CH1_TX_LIM_REG	通道 1 的 TX 事件配置寄存器	0x005C	varies

名称	描述	地址	访问
RMT_TX_SIM_REG	RMT TX 同步发送寄存器	0x006C	R/W
RX 事件配置寄存器			
RMT_CH2_RX_LIM_REG	通道 2 RX 事件配置寄存器	0x0060	R/W
RMT_CH3_RX_LIM_REG	通道 3 RX 事件配置寄存器	0x0064	R/W
版本寄存器			
RMT_DATE_REG	版本控制寄存器	0x00CC	R/W

32.7 寄存器

本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。

Register 32.1. RMT_CHnDATA_REG (n: 0-3) (0x0000+0x4*n)



RMT_CHnDATA 通道 n 通过 APB FIFO 进行读写操作时用到的数据寄存器。(HRO)

Register 32.2. RMT_CHnCONFO_REG (n: 0-1) (0x0010+0x4*n)



RMT_TX_START_CHn 配置是否使能通道 n 开始发送数据。
0: 无效
1: 使能
(WT)

RMT_MEM_RD_RST_CHn 配置是否复位通道 n 通过发射器访问的 RAM 读地址。
0: 无效
1: 复位
(WT)

RMT_APB_MEM_RST_CHn 配置是否复位通道 n 通过 APB FIFO 访问的读/写 RAM 地址。
0: 无效
1: 复位
(WT)

见下页...

Register 32.2. RMT_CH n CONFO_REG (n : 0-1) (0x0010+0x4* n)

接上页...

RMT_TX_CONTI_MODE_CH n 配置是否使能通道 n 的持续发送模式。

0: 无效

1: 使能

在这种模式下, 发射器从第一个数据开始发送, 如果遇到结束标志, 则再次发送第一个数据; 如果没有遇到结束标志, 发送完最后一个数据后, 回卷到第一个数据再次继续发送。

(R/W)

RMT_MEM_TX_WRAP_EN_CH n 配置是否使能通道 n 的乒乓发送模式。

0: 无效

1: 使能

在这种模式下, 如果待发送的数据长度大于该通道的 RAM Block 长度, 则发射器将继续从第一个数据开始循环发送。

(R/W)

RMT_IDLE_OUT_LV_CH n 配置通道 n 在空闲模式下的输出信号电平。(R/W)

RMT_IDLE_OUT_EN_CH n 配置是否使能通道 n 在空闲状态下的输出模式。

0: 无效

1: 使能

(R/W)

RMT_TX_STOP_CH n 配置通道 n 的发射器是否停止发送数据。

0: 无效

1: 停止

(R/W/SC)

RMT_DIV_CNT_CH n 配置通道 n 的时钟分频系数。

测量单位: rmt_sclk

(R/W)

RMT_MEM_SIZE_CH n 配置通道 n 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EFF_EN_CH n 配置是否仅在发送数据状态下对输出信号载波调制。

0: 配置通道 n 对发送数据状态和空闲状态均加载载波

1: 配置通道 n 仅在发送数据状态下对输出信号载波调制

仅在 RMT_CARRIER_EN_CH n 为 1 时有效。

(R/W)

RMT_CARRIER_EN_CH n 配置是否使能通道 n 对输出信号进行载波调制。

0: 不使能

1: 使能

(R/W)

见下页...

Register 32.2. RMT_CHnCONFO_REG (n: 0-1) (0x0010+0x4*n)

接上页...

RMT_CARRIER_OUT_LV_CHn 配置通道 n 的载波调制方式。

0: 载波加载在低电平上

1: 载波加载在高电平上

(R/W)

RMT_CONF_UPDATE_CHn 通道 n 的同步位。(WT)

Register 32.3. RMT_CHmCONFO_REG (m: 2-3) (0x0018+0x8*(m-2))

(reserved)				RMT_CARRIER_OUT_LV_CHm				RMT_CARRIER_EN_CHm				(reserved)				RMT_MEM_SIZE_CHm				RMT_IDLE_THRES_CHm								RMT_DIV_CNT_CHm							
31	30	29	28	27	26	25	23	22											8	7											0				
0	0	1	1	0	0	0x1		0x7fff										0x2										Reset							

RMT_DIV_CNT_CHm 配置通道 m 的时钟分频时钟。

测量单位: rmt_sclk

(R/W)

RMT_IDLE_THRES_CHm 配置接收阈值。

接收器长时间检测不到信号变化，且持续的时间大于 RMT_IDLE_THRES_CHm 的值，则接收器停止接收过程。

测量单位: clk_div

(R/W)

RMT_MEM_SIZE_CHm 配置通道 m 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EN_CHm 配置是否使能通道 m 对输出信号进行载波调制和解调。

0: 不使能

1: 使能

(R/W)

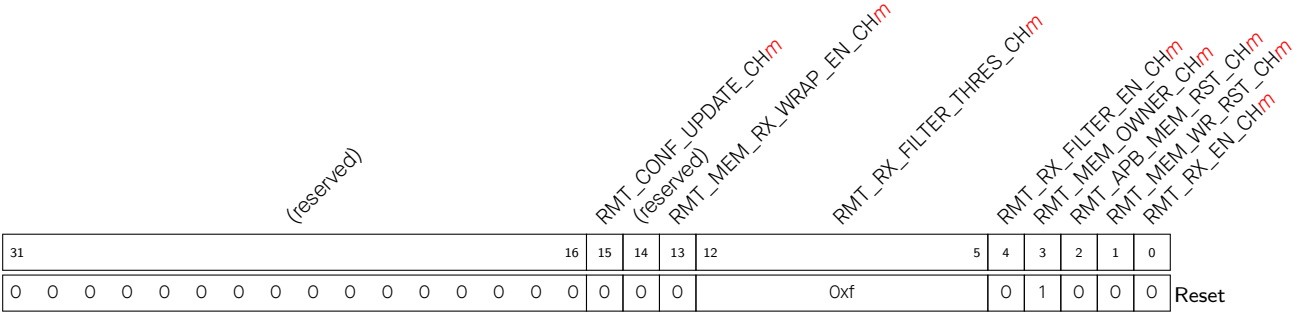
RMT_CARRIER_OUT_LV_CHm 配置通道 m 的载波调制方式。

0: 载波加载在低电平上

1: 载波加载在高电平上

(R/W)

Register 32.4. RMT_CH m CONF1_REG (m : 2-3) (0x001C+0x8*(m -2))



RMT_RX_EN_CH m 配置是否使能通道 m 的接收器，使接收器开始接收数据。

- 0: 不使能
 - 1: 使能
- (R/W)

RMT_MEM_WR_RST_CH m 配置是否复位通道 m 通过接收器访问的 RAM 写地址。

- 0: 无效
 - 1: 复位
- (WT)

RMT_APB_MEM_RST_CH m 配置是否复位通道 m 通过 APB FIFO 访问的读/写 RAM 地址。

- 0: 无效
 - 1: 复位
- (WT)

RMT_MEM_OWNER_CH m 配置通道 m 的 RAM 使用权。

- 0: APB 总线有权使用该通道的 RAM
 - 1: 接收器有权使用该通道的 RAM
- (R/W/SC)

RMT_RX_FILTER_EN_CH m 配置是否使能通道 m 的接收器滤波功能。

- 0: 不使能
 - 1: 使能
- (R/W)

RMT_RX_FILTER_THRES_CH m 配置是否在接收模式下，忽略宽度小于 RMT_RX_FILTER_THRES_CH m 个 rmt_sclk 周期的输入脉冲。

- 0: 无效
 - 1: 忽略
- (R/W)

RMT_MEM_RX_WRAP_EN_CH m 配置是否使能通道 m 的乒乓接收模式。

- 0: 不使能
 - 1: 使能
- 在这种模式下，如果待接收的数据长度大于该通道的 RAM Block 长度，则接收器将继续把接收数据存入 RAM 的第一个地址，依次循环。
- (R/W)

RMT_CONF_UPDATE_CH m 通道 m 的同步位。(WT)

Register 32.5. RMT_SYS_CONF_REG (0x0068)

Register diagram for RMT_CLK_EN. The register is 32 bits wide. Bit 31 is labeled RMT_CLK_EN. Bits 30-1 are labeled (reserved). Bit 0 is labeled RMT_APB_FIFO_MASK. The register is shown with a Reset signal.

RMT_APB_FIFO_MASK 配置 RAM 的访问模式。

0: 通过 FIFO 访问 RAM

1: 直接访问 RAM

(R/W)

RMT_CLK_EN 配置是否使能 RMT 寄存器的时钟门控。

0: 关闭寄存器的驱动时钟

1: 打开寄存器的驱动时钟

(R/W)

Register 32.6. RMT_REF_CNT_RST_REG (0x0070)

Diagram of the 32-bit RST register structure:

- Reserved (28 bits)
- RST_CNT (4 bits)
- RST_CH3 (1 bit)
- RST_CH2 (1 bit)
- RST_CH1 (1 bit)

RMT_REF_CNT_RST_CH n (n : 0-1) 配置是否复位通道 n 的时钟分频器。

0: 无效

1: 复位

(WT)

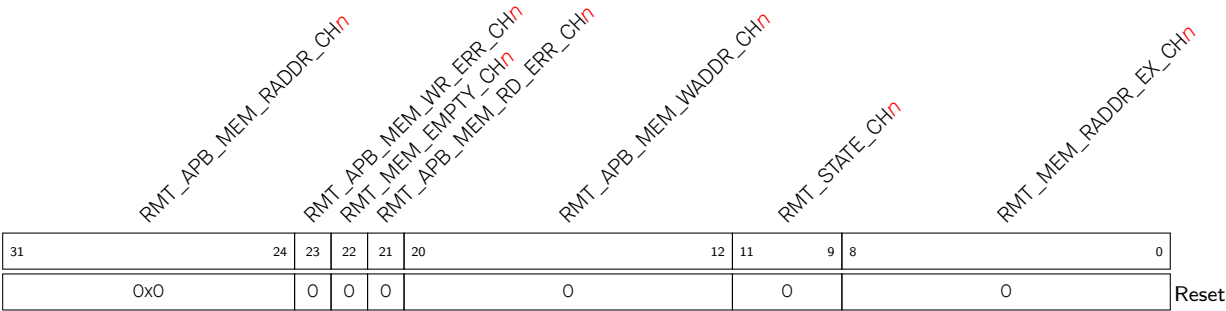
RMT_REF_CNT_RST_CH m (m : 2-3) 配置是否复位通道 m 的时钟分频器。

0: 无效

1: 复位

(WT)

Register 32.7. RMT_CH n STATUS_REG (n : 0-1) (0x0028+0x4* n)



RMT_MEM_RADDR_EX_CH n 表示通道 n 发射器使用 RAM 时的地址偏移量。(RO)

RMT_STATE_CH n 表示通道 n 的 FSM 状态。(RO)

RMT_APB_MEM_WADDR_CH n 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

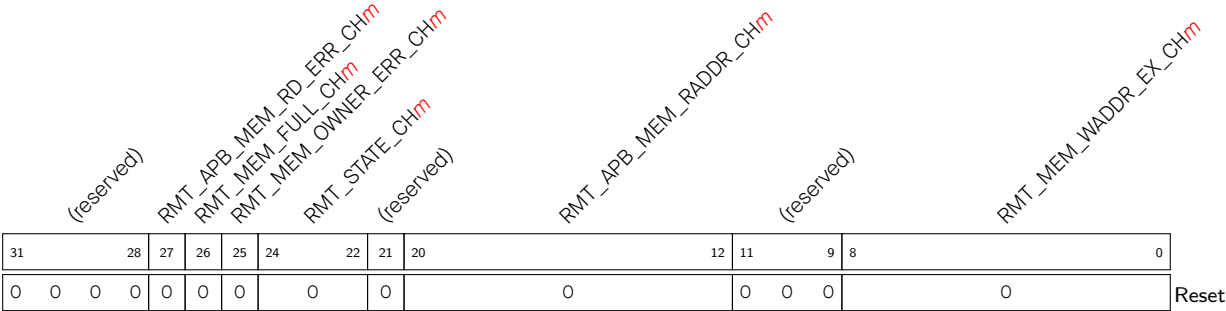
RMT_APB_MEM_RD_ERR_CH n 表示 RMT 使用 APB 总线进行读 RAM 操作时，偏移地址是否溢出 RAM Block。
0：未溢出
1：溢出
(RO)

RMT_MEM_EMPTY_CH n 表示是否发送的数据长度是否溢出 RAM Block，且乒乓模式未启用。
0：未溢出
1：溢出
(RO)

RMT_APB_MEM_WR_ERR_CH n 表示 RMT 使用 APB 总线进行写 RAM 操作时，偏移地址是否溢出 RAM Block。
0：未溢出
1：溢出
(RO)

RMT_APB_MEM_RADDR_CH n 表示 RMT 使用 APB 总线读 RAM 时的地址偏移量。(RO)

Register 32.8. RMT_CH m STATUS_REG (m : 2-3) (0x0030+0x4*(m -2))



- RMT_MEM_WADDR_EX_CH m 表示通道 m 接收器使用 RAM 时的地址偏移量。(RO)
- RMT_APB_MEM_RADDR_CH m 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)
- RMT_STATE_CH m 表示通道 m 的 FSM 状态。(RO)
- RMT_MEM_OWNER_ERR_CH m 表示 RAM Block 使用权是否发生错误。
- 0: 无错误
 - 1: 发生错误
- (RO)
- RMT_MEM_FULL_CH m 表示接收器接收的数据长度是否溢出 RAM Block。
- 0: 未溢出
 - 1: 溢出
- (RO)
- RMT_APB_MEM_RD_ERR_CH m 表示 RMT 使用 APB 总线执行 RAM 读操作时，偏移地址是否溢出 RAM Block。
- 0: 未溢出
 - 1: 溢出
- (RO)

Register 32.9. RMT_INT_RAW_REG (0x0038)

(reserved)														RMT_CH1_TX_LOOP_INT_RAW RMT_CH0_TX_LOOP_INT_RAW RMT_CH3_RX_THR_EVENT_INT_RAW RMT_CH2_RX_THR_EVENT_INT_RAW RMT_CH1_TX_THR_EVENT_INT_RAW RMT_CH0_TX_THR_EVENT_INT_RAW RMT_CH3_ERR_INT_RAW RMT_CH2_ERR_INT_RAW RMT_CH1_ERR_INT_RAW RMT_CH0_ERR_INT_RAW RMT_CH3_RX_END_INT_RAW RMT_CH2_RX_END_INT_RAW RMT_CH1_TX_END_INT_RAW RMT_CH0_TX_END_INT_RAW															
31														14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

RMT_CHn_TX_END_INT_RAW (n: 0-1) RMT_CHn_TX_END_INT 的原始中断状态。发送完成时触发。
(R/WTC/SS)

RMT_CHm_RX_END_INT_RAW (m: 2-3) RMT_CHm_RX_END_INT 的原始中断状态。接收完成时触发。
(R/WTC/SS)

RMT_CHn_ERR_INT_RAW (n: 0-3) RMT_CHn/m_ERR_INT 的原始中断状态。发生错误时触发。(R/WTC/SS)

RMT_CHn_TX_THR_EVENT_INT_RAW (n: 0-1) RMT_CHn_TX_THR_EVENT_INT 的原始中断状态。
发送器发送多于配置值时触发。(R/WTC/SS)

RMT_CHm_RX_THR_EVENT_INT_RAW (m: 2-3) RMT_CHm_RX_THR_EVENT_INT 的原始中断状态。
接收器接收多于配置值时触发。(R/WTC/SS)

RMT_CHn_TX_LOOP_INT_RAW (n: 0-1) RMT_CHn_TX_LOOP_INT 的原始中断状态。循环计数次数
到达配置阈值时触发。(R/WTC/SS)

Register 32.10. RMT_INT_ST_REG (0x003C)

(reserved)																																RMT_CH1_TX_LOOP_INT_ST RMT_CH0_TX_LOOP_INT_ST RMT_CH3_RX_THR_EVENT_INT_ST RMT_CH2_RX_THR_EVENT_INT_ST RMT_CH1_TX_THR_EVENT_INT_ST RMT_CH0_TX_THR_EVENT_INT_ST RMT_CH3_ERR_INT_ST RMT_CH2_ERR_INT_ST RMT_CH1_ERR_INT_ST RMT_CH0_ERR_INT_ST RMT_CH3_RX_END_INT_ST RMT_CH2_RX_END_INT_ST RMT_CH1_TX_END_INT_ST RMT_CH0_TX_END_INT_ST															
31																14																13	12	11	10	9	8	7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset																

RMT_CH n _TX_END_INT_ST (n : 0-1) RMT_CH n _TX_END_INT 的屏蔽中断状态。(RO)

RMT_CH m _RX_END_INT_ST (m : 2-3) RMT_CH m _RX_END_INT 的屏蔽中断状态。(RO)

RMT_CH n _ERR_INT_ST (n : 0-3) RMT_CH n/m _ERR_INT 的屏蔽中断状态。(RO)

RMT_CH n _TX_THR_EVENT_INT_ST (n : 0-1) RMT_CH n _TX_THR_EVENT_INT 的屏蔽中断状态。
(RO)

RMT_CH m _RX_THR_EVENT_INT_ST (m : 2-3) RMT_CH m _RX_THR_EVENT_INT 的屏蔽中断状态。
(RO)

RMT_CH n _TX_LOOP_INT_ST (n : 0-1) RMT_CH n _TX_LOOP_INT 的屏蔽中断状态。(RO)

Register 32.11. RMT_INT_ENA_REG (0x0040)

(reserved)																															RMT_CH1_TX_LOOP_INT_ENA	RMT_CH0_TX_LOOP_INT_ENA	RMT_CH3_RX_THR_EVENT_INT_ENA	RMT_CH2_RX_THR_EVENT_INT_ENA	RMT_CH1_TX_THR_EVENT_INT_ENA	RMT_CH0_TX_THR_EVENT_INT_ENA	RMT_CH3_ERR_INT_ENA	RMT_CH2_ERR_INT_ENA	RMT_CH1_ERR_INT_ENA	RMT_CH0_ERR_INT_ENA	RMT_CH3_RX_END_INT_ENA	RMT_CH2_RX_END_INT_ENA	RMT_CH1_TX_END_INT_ENA	RMT_CH0_TX_END_INT_ENA
31																14	13	12	11	10	9	8	7	6	5	4	3	2	1	0														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset															

RMT_CH n _TX_END_INT_ENA (n : 0-1) 写 1 使能 RMT_CH n _TX_END_INT。(R/W)

RMT_CH m _RX_END_INT_ENA (m : 2-3) 写 1 使能 RMT_CH m _RX_END_INT。(R/W)

RMT_CH n _ERR_INT_ENA (n : 0-3) 写 1 使能 RMT_CH n/m _ERR_INT。(R/W)

RMT_CH n _TX_THR_EVENT_INT_ENA (n : 0-1) 写 1 使能 RMT_CH n _TX_THR_EVENT_INT。(R/W)

RMT_CH m _RX_THR_EVENT_INT_ENA (m : 2-3) 写 1 使能 RMT_CH m _RX_THR_EVENT_INT。(R/W)

RMT_CH n _TX_LOOP_INT_ENA (n : 0-1) 写 1 使能 RMT_CH n _TX_LOOP_INT。(R/W)

Register 32.12. RMT_INT_CLR_REG (0x0044)

(reserved)														RMT_CH1_TX_LOOP_INT_CLR RMT_CH0_TX_LOOP_INT_CLR RMT_CH3_RX_THR_EVENT_INT_CLR RMT_CH2_RX_THR_EVENT_INT_CLR RMT_CH1_TX_THR_EVENT_INT_CLR RMT_CH0_TX_THR_EVENT_INT_CLR RMT_CH3_ERR_INT_CLR RMT_CH2_ERR_INT_CLR RMT_CH1_ERR_INT_CLR RMT_CH0_ERR_INT_CLR RMT_CH3_RX_END_INT_CLR RMT_CH2_RX_END_INT_CLR RMT_CH1_TX_END_INT_CLR RMT_CH0_TX_END_INT_CLR															
31														14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Reset
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0														0	0	0	0	0	0	0	0	0	0	0	0	0	0		

- RMT_CHn_TX_END_INT_CLR (n: 0-1) 写 1 清除 RMT_CHn_TX_END_INT。 (WT)
- RMT_CHm_RX_END_INT_CLR (m: 2-3) 写 1 清除 RMT_CHm_RX_END_INT。 (WT)
- RMT_CHn_ERR_INT_CLR (n: 0-3) 写 1 清除 RMT_CHn/m_ERR_INT。 (WT)
- RMT_CHn_TX_THR_EVENT_INT_CLR (n: 0-1) 写 1 清除 RMT_CHn_TX_THR_EVENT_INT。 (WT)
- RMT_CHm_RX_THR_EVENT_INT_CLR (m: 2-3) 写 1 清除 RMT_CHm_RX_THR_EVENT_INT。 (WT)
- RMT_CHn_TX_LOOP_INT_CLR (n: 0-1) 写 1 清除 RMT_CHn_TX_LOOP_INT。 (WT)

Register 32.13. RMT_CHnCARRIER_DUTY_REG (n: 0-1) (0x0048+0x4*n)

RMT_CARRIER_HIGH_CHn																RMT_CARRIER_LOW_CHn															
31															16	15														0	Reset
0x40																0x40															

- RMT_CARRIER_LOW_CHn 配置通道 n 载波的低电平时钟周期。
测量单位: rmt_sclk
(R/W)
- RMT_CARRIER_HIGH_CHn 配置通道 n 载波的高电平时钟周期。
测量单位: rmt_sclk
(R/W)

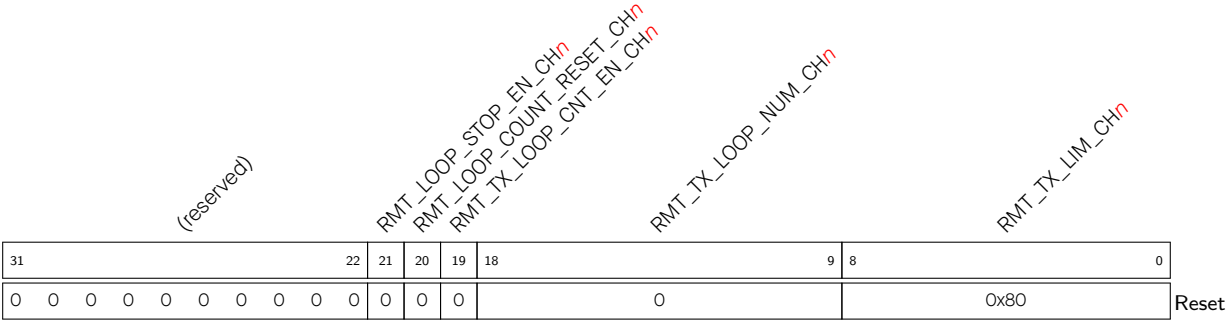
Register 32.14. RMT_CH m _RX_CARRIER_RM_REG (m : 2-3) (0x0050+0x4*(m -2))



RMT_CARRIER_LOW_THRES_CH m 配置载波调制模式下通道 m 的低电平周期。
载波调制模式下，通道 m 低电平周期为 RMT_CARRIER_LOW_THRES_CH m + 1。
测量单位：clk_div
(R/W)

RMT_CARRIER_HIGH_THRES_CH m 配置载波调制模式下通道 m 的高电平周期。
载波调制模式下，通道 m 高电平周期为 RMT_CARRIER_HIGH_THRES_CH m + 1。
测量单位：clk_div
(R/W)

Register 32.15. RMT_CHn_TX_LIM_REG (n: 0-1) (0x0058+0x4*n)



RMT_TX_LIM_CHn 配置通道 n 发送脉冲编码数量的上限值。(R/W)

RMT_TX_LOOP_NUM_CHn 配置持续发送模式下最大循环发送次数。(R/W)

RMT_TX_LOOP_CNT_EN_CHn 配置是否使能循环次数计数。

0: 无效

1: 使能

(R/W)

RMT_LOOP_COUNT_RESET_CHn 配置是否重置持续发送模式下的循环计数器。

0: 无效

1: 重置

(WT)

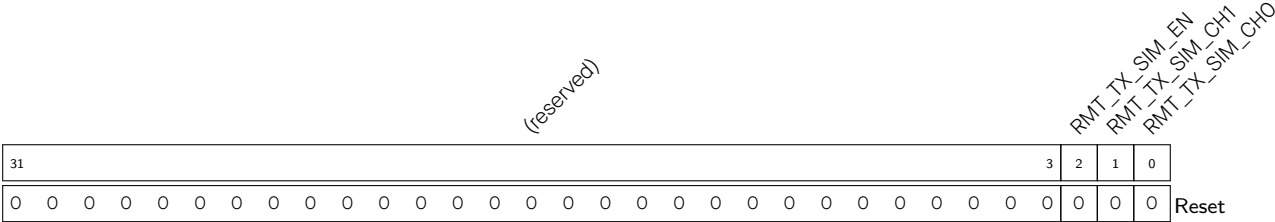
RMT_LOOP_STOP_EN_CHn 配置是否在通道 n 的循环计数器记到循环数后使能循环发送停止功能。

0: 无效

1: 使能

(R/W)

Register 32.16. RMT_TX_SIM_REG (0x006C)



RMT_TX_SIM_CHn (n: 0-1) 配置是否使能通道 n 与其它启用的通道同步开始发送数据。

0: 不使能

1: 使能

(R/W)

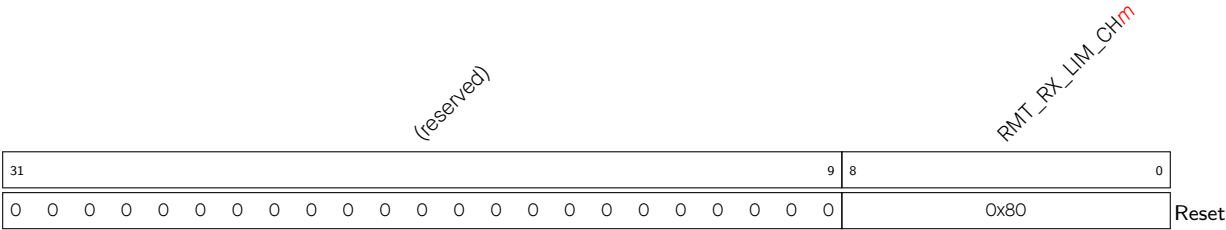
RMT_TX_SIM_EN 配置是否使能多个通道开始同步发送数据。

0: 不使能

1: 使能

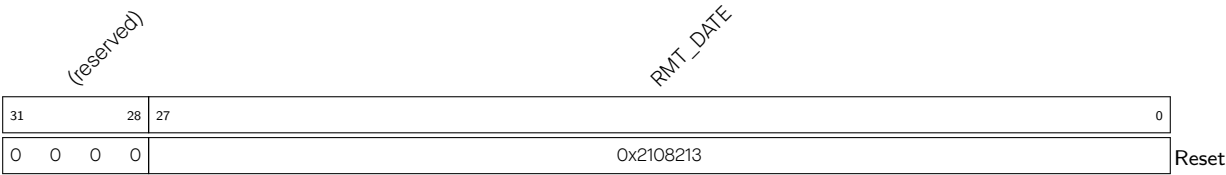
(R/W)

Register 32.17. RMT_CH m _RX_LIM_REG (m : 2-3) (0x0060+0x4*(m -2))



RMT_RX_LIM_CH m 配置通道 m 最大可接收的脉冲编码数量。(R/W)

Register 32.18. RMT_DATE_REG (0x00CC)



RMT_DATE 版本控制寄存器。(R/W)

第 33 章

并行 IO 控制器 (PARLIO)

33.1 概况

ESP32-C5 包含一个并行 IO 控制器 (PARLIO)，支持通过 GDMA 在并行总线上实现外部设备和内部存储器之间的数据通信。它由 TX 和 RX 两个子模块组成，分别作为发送器和接收器。PARLIO 支持全双工通信。

鉴于 IO 数据的灵活性，PARLIO 可用作通用接口，连接各种外围设备。例如可以把 SPI 作为主机，PARLIO 作为从机，实现点对点传输。详细的应用实例，请参考小节 33.8。

33.2 术语

为了更好地说明 PARLIO 的功能，本章使用了以下术语。

RX 模块	PARLIO 子模块，从外部并行总线接收数据并将其存储在内部存储器中。
TX 模块	PARLIO 子模块，将数据从内部存储器传输到外部并行总线上。
RXD	从 RX 模块的 IO 接口接收的并行数据。
TXD	从 TX 模块的 IO 接口发送的并行数据。
帧	传输数据的单位，从设置 START 信号时起，到收到 EOF 信号止。
自由运行时钟	持续翻转的时钟。若非自由运行时钟，则时钟只在收到有效数据的期间翻转，其余时间停止翻转。
GDMA SUC EOF	GDMA 帧传输成功结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧正确，且数据接收结束。
GDMA ERR EOF	GDMA 帧传输错误结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧错误，且数据接收结束。
CDC	跨时钟域处理 (Clock Domain Crossing)。

33.3 特性

PARLIO 模块具备以下特性：

- 支持多种时钟源可选：
 - 包括外部 IO 时钟 PAD_CLK_TX/RX、内部系统时钟 XTAL_CLK、PLL_F240M_CLK 和 RC_FAST_CLK
 - 支持最大时钟频率 40 MHz
 - 支持对时钟频率进行整数分频
- 支持将传输数据总线位宽配置为 1/2/4/8 位
- 支持 8 位全双工传输

- 总线位宽为 1/2/4 位时，支持在一个字节范围内对比特数据顺序进行翻转
- 包含用于接收 IO 并行数据的 RX 子模块：
 - 支持对输出时钟进行门控
 - 支持 RX 子模块输入时钟和输出时钟分别取反
 - 支持多种接收模式
 - 支持配置 GDMA SUC EOF 信号生成模式
 - 支持配置外部使能信号的 IO 管脚
- 包含用于发送 IO 并行数据的 TX 子模块：
 - 支持片选功能
 - 支持对输出时钟进行门控
 - 支持 TX 子模块输入时钟和输出时钟分别取反
 - 支持配置总线空闲值

33.4 系统架构

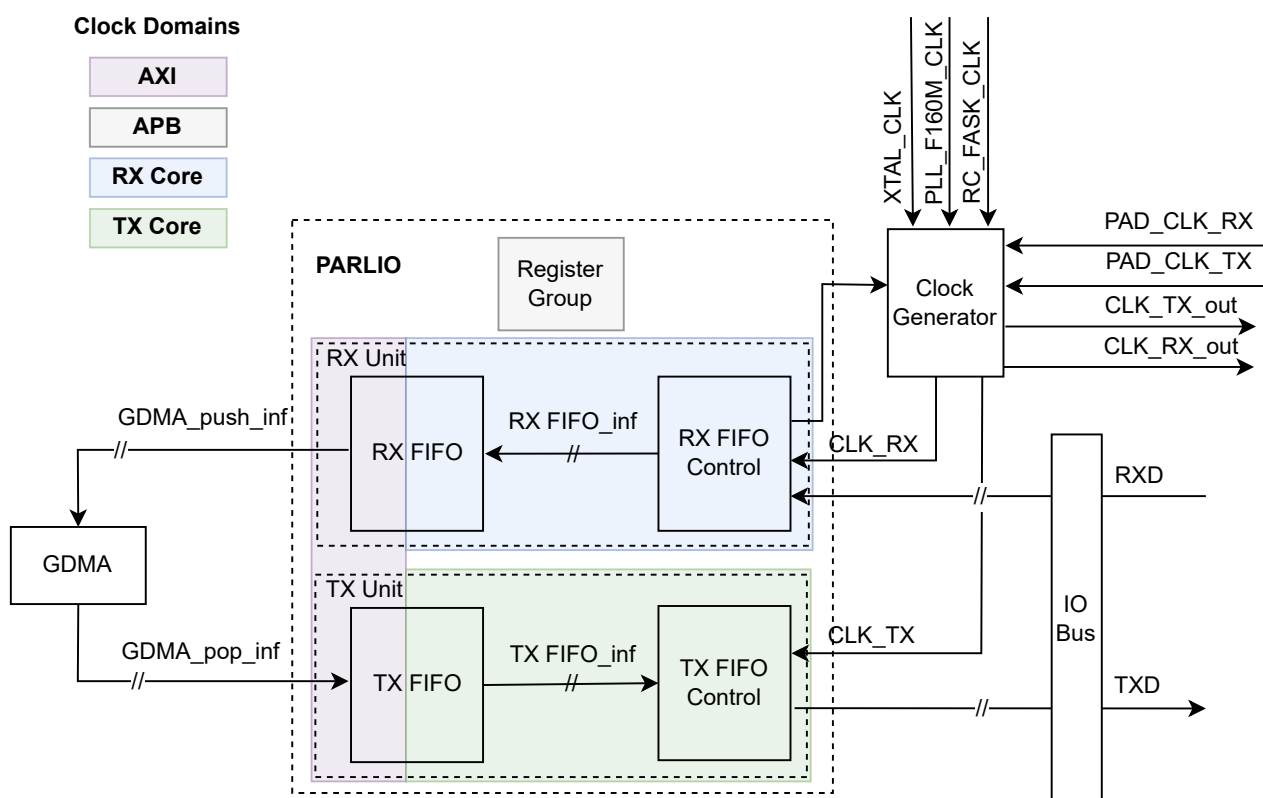


图 33.4-1. PARLIO 系统框图

图 33.4-1 展示了 PARLIO 的系统内部架构。除了 RX 模块和 TX 模块外，PARLIO 还包括一组状态配置寄存器。RX 模块接收 RXD 信号，并在完成串并转换后将其存入异步 FIFO，然后将这些数据通过 GDMA 存到内部存储器。

TX 模块通过 GDMA 从内部存储器获取数据并存入异步 FIFO，将这些数据进行并串转换后，通过 TXD 信号从 IO 总线输出。

33.5 功能描述

33.5.1 时钟生成器

PARLIO 模块有四个输入时钟域，即 RX Core、TX Core、AHB 和 APB，如图 33.4-1 所示。

状态配置寄存器组位于 APB 时钟域内。

GDMA 接口逻辑位于 AHB 时钟域内。

RX Core 和 TX Core 时钟域各有四个时钟源可供选择，分别为内部系统时钟源 XTAL_CLK、RC_FAST_CLK、PLL_F240M_CLK 和外部时钟源 (PAD_CLK_TX/RX)，如图 33.5-1 所示。配置 PCR_PARL_CLK_RX_SEL 和 PCR_PARL_CLK_TX_SEL 选择时钟源，配置 PCR_PARL_CLK_RX_DIV_NUM 和 PCR_PARL_CLK_TX_DIV_NUM 对时钟进行分频，分频系数最大可配置为 $(2^{16} - 1)$ 。

RX 模块和 TX 模块支持对输入时钟进行取反，同时其工作时钟可以输出到 IO，并在输出前支持对输出时钟进行取反。此外 RX 模块和 TX 模块还支持对输出时钟进行门控。

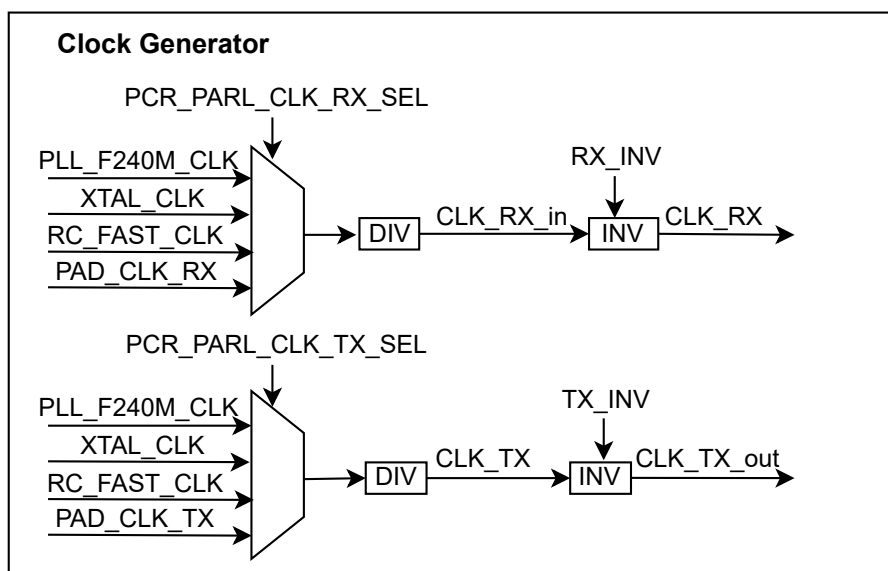


图 33.5-1. PARLIO 时钟生成

33.5.2 时钟复位使用限制

由于 PARLIO 模块的通用性，其 PAD 时钟 (PAD_CLK_TX/RX) 可能来自不同的主机（外部设备或内部时钟源），这些时钟可能是自由运行时钟，也可能不是。如果外部运行时钟不是自由运行时钟，PARLIO 内部的一些控制信号将无法进行 CDC，因此对操作过程会有一定限制。

1. 异步 FIFO 复位时，需要两个时钟周期才能初始化 FIFO。因此如果需要在非自由运行时钟域复位 AHB 时钟域，需保证提前两个时钟周期进行复位同步。具体的操作步骤见下表。

表 33.5-1. 时钟限制下复位 AHB 时钟域操作步骤

时钟限制		具体操作
当前帧传输时钟	下一帧传输时钟	
自由运行时钟	非自由运行时钟	在切换到非自由运行时钟之前复位下一帧传输，并在复位完成后配置时钟切换。
非自由运行时钟	自由运行时钟	只需确保在复位和正式开始传输期间有两个时钟周期的间隔。
非自由运行时钟	非自由运行时钟	如需复位下一帧传输，需要先切换到内部的自由运行时钟，复位完成后再切换回实际时钟。

2. 由于非自由运行时钟造成的限制，[PARL_IO_RX_START](#) 和 [PARL_IO_TX_START](#) 无法进行 CDC，所以在操作时必须等到 [PARL_IO_RX_START](#) 和 [PARL_IO_TX_START](#) 稳定后再开始数据传输，否则可能会发生亚稳态的问题。

RX 模块中的具体操作步骤为：

- (a) 清除 [PCR_PARL_CLK_RX_EN](#) 关闭 RX Core 时钟域；
- (b) 向 [PARL_IO_RX_START](#) 写 1；
- (c) 置位 [PCR_PARL_CLK_RX_EN](#) 打开 RX Core 时钟域；
- (d) 操作外部设备，使其开始发送数据；
- (e) 清除 [PCR_PARL_CLK_RX_EN](#) 关闭 RX Core 时钟域；
- (f) 向 [PARL_IO_RX_START](#) 写 0。

TX 模块中的具体操作步骤为：

- (a) 清除 [PCR_PARL_CLK_TX_EN](#) 关闭 TX Core 时钟域；
- (b) 向 [PARL_IO_TX_START](#) 写 1；
- (c) 置位 [PCR_PARL_CLK_TX_EN](#) 打开 TX Core 时钟域；
- (d) 操作外部设备，使其开始接收数据；
- (e) 清除 [PCR_PARL_CLK_TX_EN](#) 关闭 TX Core 时钟域；
- (f) 向 [PARL_IO_TX_START](#) 写 0。

3. 复位需遵循以下要求：

- 芯片启动时，时钟复位应遵循以下顺序：
 - (a) 复位 APB 时钟域；
 - (b) 复位 AHB 时钟域；
 - (c) 复位 Core 时钟域。
- 帧间传输需要复位 Core 时钟域和异步 FIFO。

33.5.3 主从机模式

RX 模块和 TX 模块均支持主从机模式。

当 TX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。TX 模块在时钟的上升沿驱动 TXD。

当 TX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 33.5-2. 时钟限制下 TX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求。
非自由运行时钟	正向波形（图 33.5-2）	主机需要在下降沿时采集。
非自由运行时钟	负向波形（图 33.5-3）	主机在输出前需要对原时钟进行取反，将其转换成正向波形。

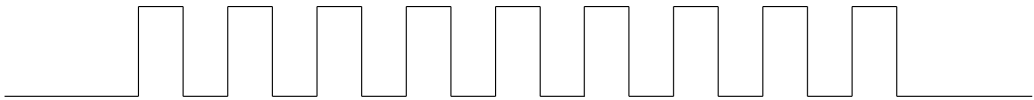


图 33.5-2. 主机时钟正向波形

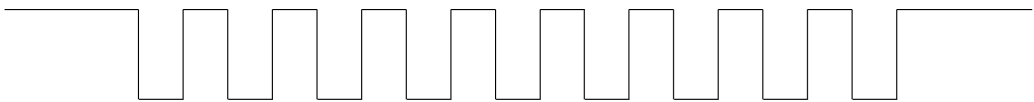


图 33.5-3. 主机时钟负向波形

当 RX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。RX 模块在时钟的上升沿驱动 RXD。

当 RX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 33.5-3. 时钟限制下 RX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求，采集的有效数据以外部使能信号为准。
非自由运行时钟	正向波形（图 33.5-2）	主机需在上升沿驱动数据，RX 模块需在下降沿采集数据（即对主时钟进行翻转）。
非自由运行时钟	负向波形（图 33.5-3）	主机需在下降沿驱动数据，RX 模块需在上升沿采集数据（即使用原始主时钟）

33.5.4 RX 模块接收模式

PARLIO 的 RX 模块有 8 种接收模式，可根据使能信号规格分为三大类：

- 电平使能模式：接收的数据由外部信号电平使能；
- 脉冲使能模式：接收的数据由外部信号脉冲使能；
- 软件使能模式：数据接收的使能信号可由用户直接配置。

RX 模块还支持对外部使能信号进行翻转。如果外部使能信号为低电平或负脉冲，可通过置位 `PARL_IO_RX_EXT_EN_INV` 使能该功能，转换为以下相应的接收模式。

33.5.4.1 电平使能模式

电平使能模式如图 33.5-4 所示。在这种模式下，外部使能信号的有效电平必须与有效数据对齐。由于外部电平使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 7 个。

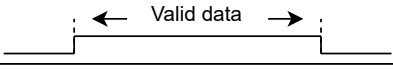
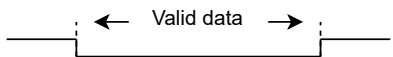
Mode	Sub-mode	Description
LEVEL_ENABLE	sub-mode 1	signal level high
		
	sub-mode 2	signal level low
		

图 33.5-4. RX 模块电平使能子模式

33.5.4.2 脉冲使能模式

根据脉冲有效电平及其与有效数据的对应性，脉冲使能模式可分为 6 种子模式，详细分类可参见图 33.5-5。

子模式 1 ~ 4 都包含起始脉冲和结束脉冲，不同之处在于起始脉冲和结束脉冲是否与有效数据对齐。

子模式 5 ~ 6 只包含起始脉冲，通过配置 `PARL_IO_RX_BITLEN` 可表示有效数据传输结束。

由于外部脉冲使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 7 个。但在子模式 4 和子模式 6 下，数据在脉冲的第一个边沿后和最后一个边沿前被认为有效，可以实现使能信号的 IO 管脚与数据 IO 管脚的复用，此时 RXD 可使用 8 个 IO 管脚。

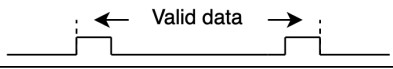
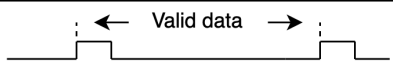
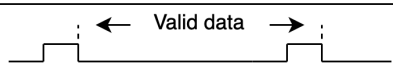
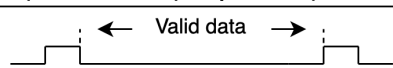
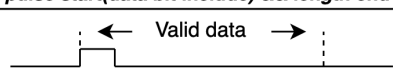
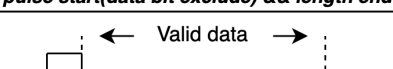
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	pulse start(data bit include) && pulse end(data bit include)
		
	sub-mode2	pulse start(data bit include) && pulse end(data bit exclude)
		
	sub-mode3	pulse start(data bit exclude) && pulse end(data bit include)
		
	sub-mode4	pulse start(data bit exclude) && pulse end(data bit exclude)
		
	sub-mode5	pulse start(data bit include) && length end
		
	sub-mode6	pulse start(data bit exclude) && length end
		

图 33.5-5. RX 模块脉冲使能子模式

33.5.4.3 软件使能模式

软件使能模式下的使能信号由内部配置寄存器决定。用户切换到此模式时，只有同时写 1 到 `PARL_IO_RX_SW_EN` 和 `PARL_IO_RX_START`，才能开始接收数据。

由于使能信号不占用接口 IO 管脚，RXD 在接口上的最大可用 IO 管脚数量为 8。由于时钟域不同，使能信号无法与有效数据对齐，因此只能通过有效时钟沿来判定有效数据。此时，RX Core 时钟需要与有效数据一一对应。

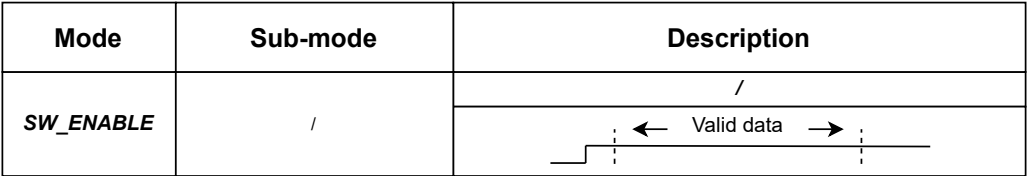


图 33.5-6. RX 模块软件使能子模式

33.5.5 RX 模块 GDMA SUC EOF 信号生成

RX 模块产生一个 GDMA SUC EOF 信号，表示当前帧传输结束，并将其发送到 GDMA 接口。GDMA SUC EOF 信号可由外部使能信号生成，也可基于内部配置的比特长度触发。

- 当 GDMA SUC EOF 信号由内部配置的比特长度生成时，接收模式的选择不受限制。但在这种情况下必须配置 `PARL_IO_RX_BITLEN`。如果 `PARL_IO_RX_BITLEN` 的配置值小于实际接收的数据，将提前触发 GDMA SUC EOF，此时 RX 模块停止从 FIFO 读取数据，但 FIFO 会继续接收外部数据，直到 `RX_FIFO_WOVF_INT` 中断被触发。
- 当 GDMA SUC EOF 信号由外部使能信号产生时，只可选脉冲使能模式的子模式 1、3。在这种模式下，传输不受 `PARL_IO_RX_BITLEN` 值的影响，并且传输帧的数据长度不受限制。

33.5.6 RX 模块超时

RX 模块支持接收超时功能。当 RX 模块 FIFO 长时间未接收到有效数据，超时将被触发，产生一个 GDMA ERR EOF 信号，并发送至 GDMA 接口以表示结束数据接收。配置 `PARL_IO_RX_TIMEOUT_THRES` 可设置超时阈值。

默认情况下启用超时功能，用户也可以选择禁用该功能。超时阈值的配置上限为 AHB 时钟域的 $(2^{16} - 1)$ 周期，下限取决于 AHB 时钟域和 RX Core 时钟域之间的相对频率关系。建议将 `PARL_IO_RX_TIMEOUT_THRES` 设置为较大的数值，以避免生成非期望的 GDMA ERR EOF 信号。

33.5.7 TX 模块无限制长度数据传输

默认情况下，TX 模块的传输结束标志由 `PARL_IO_TX_BITLEN` 的配置值决定。当发送的数据达到配置值时，系统会生成一个 TX EOF 中断信号，标志着该数据帧的传输结束。此时，单帧传输的最大值受到 `PARL_IO_TX_BITLEN` 的位宽限制，使其无法实现无限长度的数据传输。

为了实现更大长度的单帧传输，可以将 `PARL_IO_TX_EOF_GEN_SEL` 设置为 1。在这种配置下，硬件将使用 GDMA 发送链表配置的 EOF 信号作为单帧传输结束的标志。此时，数据传输不再受限于 `PARL_IO_TX_BITLEN` 的位宽，从而能够支持更大长度的数据传输。

33.5.8 TX 片选功能

为模拟 LCD 的 I8080 协议，TX 模块配备了一个可输出的片选信号，在拉低时有效。用户可通过配置 `PARL_IO_TX_CS_START_DELAY` 和 `PARL_IO_TX_CS_STOP_DELAY` 来控制 DMA 输出数据和片选信号的相位关系。需要注意的是，在开启时钟门控、且使用片选信号作为时钟门控控制源的情况下，对上述两个寄存器的操作在实现基础功能的同时，还可以控制输出时钟和片选信号的相位关系。

33.5.9 TX 模块输出时钟门控

TX 模块支持对输出时钟进行门控。时钟门控功能默认关闭，可通过软件置位 `PARL_IO_TX_GATING_EN` 使能该功能。

目前 TXD 最高位有两个控制源可配置，分别为 DMA 输出数据和片选信号 (CS)。将 `PARL_IO_TX_VALID_OUTPUT_EN` 配置为 0 即选择 DMA 输出数据为控制源，此时必须将 TX 模块配置为最大位宽，因为当控制源为 DMA 输出数据时，门控信号固定为 TXD 最高位，该位为高电平时，时钟信号可以翻转。需要注意的是，在选择 DMA 输出数据作为控制源时，如果外设处于空闲状态，则此时时钟门控将被用户配置的总线空闲值的最高位控制，应参照小节 33.5.10 TX 模块总线空闲值 中的注意事项进行配置；若选择 DMA 输出数据作为门控功能控制源时启用时钟门控功能，门控信号将占用一个 IO，此时 TXD 可以使用的 IO 管脚最多为 7 个。

将 `PARL_IO_TX_VALID_OUTPUT_EN` 配置为 1 即选择片选信号为控制源，此时对位宽无限制。

33.5.10 TX 模块总线空闲值

TX 模块不传输数据时被认为保持在空闲状态。TX 模块支持总线空闲值可配置。

注意，应避免配置的空闲值与其他已开启的功能产生冲突，例如当使用 TXD 的最高有效位作为有效信号时，应避免将空闲值的最高有效位配置为 1。

33.5.11 单帧数据传输

RX/TX 模块以比特为单位进行传输，即单帧至少传输 1 个比特的数据。

当 RX 模块通过比特长度生成 GDMA EOF 信号时，单帧传输的最大长度是 $(2^{19} - 1)$ 比特。当 RX 模块通过外部设备的使能信号产生 GDMA EOF 信号时，单帧传输的比特数没有限制。

TX 模块仅通过比特长度生成 EOF 信号，因此单帧传输的最大长度为 $(2^{19} - 1)$ 比特。

由于 PARLIO 在 GDMA 一侧以字节为单位进行传输，因此当 IO 一侧的数据不与字节对齐时，PARLIO 会对数据进行特殊处理。

- 在接收状态下，通过 GDMA 存入内存的数据将自动在高位补 0，从而与字节对齐。
- 在发送状态下，从内存中取出的数据将根据配置的 `PARL_IO_TX_BITLEN` 值被截断，超过该配置值的数据不会被发送。

当配置的数据总线宽度为 2/4/8 位时，比特长度必须配置为对应总线宽度的倍数。

33.5.12 字节范围内数据顺序翻转

总线位宽为 1/2/4 位时，可以对一个字节范围内的数据顺序进行翻转。以 RX 模块为例，当配置总线位宽为 2 比特时，数据在被写入 RX FIFO 之前需要被打包成一个字节。

假设原始的比特数据排列顺序为：

$$\{\{b_0, b_1\}, \{b_2, b_3\}, \{b_4, b_5\}, \{b_6, b_7\}\}$$

则启用该功能后，数据顺序将更改为：

$$\{\{b_6, b_7\}, \{b_4, b_5\}, \{b_2, b_3\}, \{b_0, b_1\}\}$$

33.6 中断

ESP32-C5 的 PARLIO 模块可以生成以下中断信号并发送给[中断矩阵](#)。

- PARL_IO_TX_INTR
- PARL_IO_RX_INTR

这些中断信号由 PARLIO 模块的内部中断源生成。表 33.6-1 列出了 PARLIO 的内部中断源、中断触发条件、以及生成的中断信号。

表 33.6-1. PARLIO 的内部中断源

内部中断源	触发条件	中断信号
TX_FIFO_EMPTY_INT	TX 模块 FIFO 空（此时 TX 发送的数据可能出现错误）	PARL_IO_TX_INTR
RX_FIFO_WOVF_INT	RX 模块 FIFO 满（此时 RX 接收的数据可能出现错误）	PARL_IO_RX_INTR
TX_EOF_INT	TX 模块发送完成完整的一帧数据	PARL_IO_TX_INTR

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 [中断矩阵 \[to be added later\]](#) > 3.1 [中断术语](#)。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [33.9 寄存器列表](#)。

33.7 配置流程

33.7.1 数据接收操作流程

本节介绍 RX 模块接收数据的配置流程。参照以下流程，从连接到外部设备的 IO 管脚接收并行数据，并存储在内部存储器中。关于 RX 模块时钟和复位的详细操作限制，请参考小节 [33.5.2](#)。

1. 复位 RX 模块。具体复位方法与顺序，请参考小节 [33.5.2](#)。
2. 置位 [PARL_IO_RX_FIFO_WOVF_INT_CLR](#) 和 [PARL_IO_RX_FIFO_WOVF_INT_ENA](#)。
3. 选择 RXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置 PCR 寄存器，选择时钟源并对时钟进行分频。
5. 关闭 RX 模块 Core 时钟域的时钟。
6. 按照小节 [33.3](#) 和 [33.5](#) 中的描述，选择所需接收模式并启用相关功能。
7. 配置 GDMA 接收链表。

8. 置位 [PARL_IO_RX_REG_UPDATE](#)，同步寄存器信号。
9. 置位 [PARL_IO_RX_START](#)。
10. 打开 RX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其发送数据。
12. 轮询 GDMA SUC EOF 中断信号。
13. 清除 GDMA SUC EOF 中断信号。
14. 关闭 RX 模块 Core 时钟域的时钟。
15. 清除 [PARL_IO_RX_START](#)。

33.7.2 数据发送操作流程

本节介绍 TX 模块发送数据的编程流程。参照以下流程，将内部存储器中的并行数据传输到与外部设备相连的 IO 管脚。关于 TX 模块时钟和复位的详细操作限制，请参考小节 [33.5.2](#)。

1. 复位 TX 模块。具体复位方法与顺序，请参考小节 [33.5.2](#)。
2. 依次置位 [PARL_IO_TX_FIFO_EMPTY_INT_CLR](#)、[PARL_IO_TX_EOF_INT_CLR](#)、[PARL_IO_TX_FIFO_EMPTY_INT_ENA](#) 和 [PARL_IO_TX_EOF_INT_ENA](#)。
3. 选择 TXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置 PCR 寄存器，选择时钟源并对时钟进行分频。
5. 关闭 TX 模块 Core 时钟域的时钟。
6. 按照小节 [33.5](#) 中的描述，选择开启所需功能。
7. 配置 GDMA 发送链表。
8. 轮询 [PARL_IO_TX_READY](#)。
9. 置位 [PARL_IO_TX_START](#)。
10. 打开 TX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其接收数据。
12. 轮询 [PARL_IO_TX_EOF_INT_ST](#) 中断信号。
13. 置位 [PARL_IO_TX_EOF_INT_CLR](#) 中断信号。
14. 关闭 TX 模块 Core 时钟域的时钟。
15. 清除 [PARL_IO_TX_START](#)。

33.8 应用示例

本节介绍 PARLIO 模块的应用示例及其具体操作过程。示例中使用的外围设备都来自 ESP 系列芯片，这些外设均可以和 PARLIO 模块组成完整的数据通路。

说明：

示例中所构建的数据通路可能不是最佳方案，例如，在实际应用中，用户可使用两颗相同 ESP 芯片上的 SPI 外设来完

成点对点传输，而无需使用 PARLIO。但本小节中的示例在一定程度上证明了 PARLIO 接口的灵活性。

33.8.1 与 SPI 进行数据传输

在本示例中，外部 SPI 作为主机发送数据，PARLIO RX 模块作为从机接收数据，与此同时，PARLIO TX 模块作为主机发送数据，外部 SPI 作为从机接收数据，从而实现点对点串行数据传输。

- 实现 SPI 发送、PARLIO 接收，请参考以下操作过程：
 1. 配置 SPI 时钟。
 2. 将 SPI 配置为主机。
 3. 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_RX，FSPICSO 管脚连接至 RXD[7]，FSPID 管脚连接至 RXD[0]。
 4. 将发送的数据写入 SPI 缓冲器中，并配置发送数据的位长。
 5. 置位 SPI_UPDATE，更新配置的寄存器值。
 6. 复位 PARLIO RX 模块。
 7. 配置 PARLIO RX 模块时钟。
 8. 关闭 PARLIO RX 模块 Core 时钟域。
 9. 将 PARLIO 接收模式配置为电平使能模式的子模式 1，配置 RX 模块数据总线宽度为 1 位，然后根据 SPI 的发送数据长度配置 [PARL_IO_RX_BITLEN](#)，最后置位 [PARL_IO_RX_REG_UPDATE](#)。
 10. 配置 PARLIO GDMA 接收链表。
 11. 置位 [PARL_IO_RX_START](#)。
 12. 打开 PARLIO RX 模块 Core 时钟域。
 13. 置位 SPI_USR，开始传输 SPI 的数据。
 14. 轮询 GDMA SUC EOF 中断信号。
 15. 清除 [PARL_IO_RX_START](#)。
- 实现 PARLIO 发送、SPI 接收，请参考以下操作过程：
 1. 配置 SPI 时钟。
 2. 将 SPI 配置为从机。
 3. 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_TX，FSPICSO 管脚连接至 TXD[7]，FSPID 管脚连接至 TXD[0]。
 4. 置位 SPI_RD_BIT_ORDER，翻转比特数据顺序。
 5. 置位 SPI_UPDATE，更新配置的寄存器值。
 6. 复位 PARLIO TX 模块。
 7. 置位 [PARL_IO_TX_EOF_INT_CLR](#) 和 [PARL_IO_TX_EOF_INT_ENA](#)。
 8. 配置 PARLIO TX 模块时钟。
 9. 关闭 PARLIO TX 模块 Core 时钟域。

10. 配置数据总线宽度为 1 位，写 1 至 `PARL_IO_TX_VALID_OUTPUT_EN`，然后配置 `PARL_IO_TX_BITLEN`。
11. 配置 PARLIO GDMA 发送链表。
12. 轮询 `PARL_IO_TX_READY`。
13. 写 1 至 `PARL_IO_TX_START`。
14. 打开 PARLIO TX 模块 Core 时钟域。
15. 开始数据传输。
16. 轮询 `PARL_IO_TX_EOF_INT_ST`。
17. 置位 `PARL_IO_TX_EOF_INT_CLR`。
18. 关闭 PARLIO TX 模块 Core 时钟域。
19. 清除 `PARL_IO_TX_START`。

33.8.2 与 I2S 进行数据传输

在本示例中，外部 I2S 作为主机发送数据，PARLIO RX 模块作为从机接收数据。PARLIO 支持 I2S TDM MSB 对齐标准模式和 TDM PCM 标准模式下的传输。当 I2S 传输协议为 TDM MSB 对齐标准模式时，需将 PARLIO 的接收模式配置为电平使能模式。当 I2S 传输协议为 TDM PCM 标准模式时，需将 PARLIO 的接收模式配置为脉冲使能模式的子模式 4，并置位 `PARL_IO_RX_EXT_EN_INV`。

本节以 TDM PCM 标准模式为例，具体操作过程如下：

1. 配置 I2S 时钟。
2. 配置信号管脚，将 `I2SO_BCK_out` 连接至 `PAD_CLK_RX`，`I2SO_WS_out` 连接至 `RXD[7]`，`I2SO_Data_out` 管脚连接至 `RXD[0]`。
3. 将 I2S 配置为主机。
4. 配置所需 I2S TX 数据和通道模式，并置位 `I2S_TX_UPDATE`。
5. 复位 I2S TX 模块和 TX FIFO。
6. 使能 `I2S_TX_DONE_INT`。
7. 配置 I2S GDMA 发送链表。
8. 置位 `I2S_TX_STOP_EN`。
9. 复位 PARLIO RX 模块。
10. 配置 PARLIO RX 模块时钟。
11. 关闭 PARLIO RX 模块 Core 时钟域。
12. 将 PARLIO 接收模式配置为脉冲使能模式的子模式 4，配置 RX 模块数据总线宽度为 1 位，然后根据 I2S 的发送数据长度配置 `PARL_IO_RX_BITLEN`，最后置位 `PARL_IO_RX_REG_UPDATE`。
13. 配置 PARLIO GDMA 接收链表。
14. 置位 `PARL_IO_RX_START`。
15. 打开 PARLIO RX 模块 Core 时钟域。

16. 置位 [I2S_TX_START](#)，开始发送数据。
17. 轮询 [I2S_TX_DONE_INT](#)。
18. 轮询 GDMA SUC EOF 中断信号。
19. 清除 [I2S_TX_START](#)。
20. 清除 [PARL_IO_RX_START](#)。

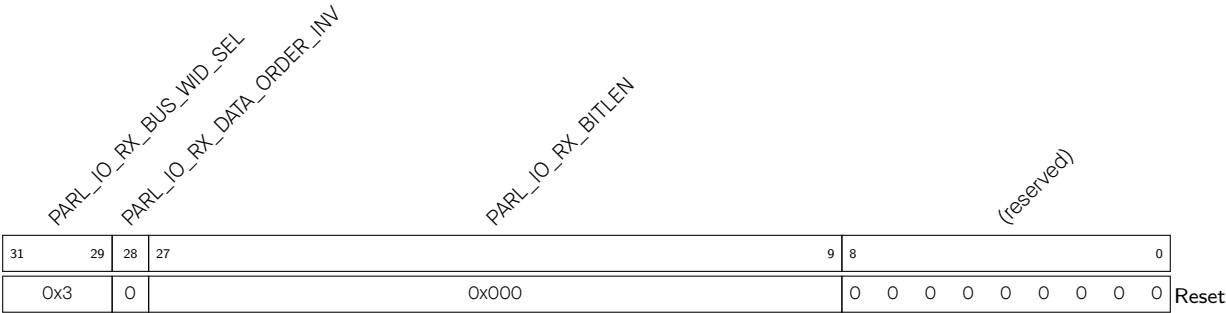
33.9 寄存器列表

本小节的所有地址均为相对于并行 IO 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
PARLIO RX 配置寄存器			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX 采样模式配置寄存器	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX 数据配置寄存器	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX 通用配置寄存器	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX 开始配置寄存器	0x000C	R/W
PARLIO TX 配置寄存器			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX 数据配置寄存器	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX 开始配置寄存器	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX 通用配置寄存器	0x0018	R/W
PARLIO 配置与状态寄存器			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO 配置寄存器	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO 寄存器更新配置寄存器	0x0020	WT
PARL_IO_ST_REG	PARLIO 状态寄存器	0x0024	RO
PARLIO 中断配置与状态寄存器			
PARL_IO_INT_ENA_REG	PARLIO 中断使能寄存器	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO 原始中断寄存器	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO 中断状态寄存器	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO 中断清除寄存器	0x0034	WT
PARLIO RX/TX 状态寄存器			
PARL_IO_RX_STO_REG	PARLIO RX 状态寄存器 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX 状态寄存器 1	0x003C	RO
PARL_IO_TX_STO_REG	PARLIO TX 状态寄存器 0	0x0040	RO
PARLIO 时钟配置寄存器			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX 时钟配置寄存器	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX 时钟配置寄存器	0x0048	R/W
PARL_IO_CLK_REG	PARLIO 时钟配置寄存器	0x0120	R/W
PARLIO TX 片选配置寄存器			
PARL_IO_TX_CS_CFG_REG	PARLIO TX 片选配置寄存器	0x004C	R/W
PARLIO 版本寄存器			
PARL_IO_VERSION_REG	版本控制寄存器	0x03FC	R/W

Register 33.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN 配置 RXD 的比特长度。(R/W)

PARL_IO_RX_DATA_ORDER_INV 配置是否翻转从 RX FIFO 发送到 GDMA 的数据的字节范围内比特顺序。

- 0: 无效
 - 1: 翻转数据顺序
- (R/W)

PARL_IO_RX_BUS_WID_SEL 配置 RXD 总线位宽。

- 0: 1 位
 - 1: 2 位
 - 2: 4 位
 - 3: 8 位
 - 4 ~ 15: 无效值，会引发错误
- (R/W)

Register 33.3. PARL_IO_RX_GENRL_CFG_REG (0x0008)

(reserved)				PARL_IO_RX_TIMEOUT_THRES												PARL_IO_RX_GATING_EN														(reserved)			
31	30	29	28													13	12	11													0		
0	0	1	0xfff												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset	

PARL_IO_RX_GATING_EN 配置是否使能 RX 输出时钟的时钟门控。

0: 不使能

1: 使能

(R/W)

PARL_IO_RX_TIMEOUT_THRES 配置 RX 超时计数器的阈值。(R/W)

PARL_IO_RX_TIMEOUT_EN 配置是否使能超时功能来生成 GDMA ERR EOF。

0: 不使能

1: 使能

(R/W)

PARL_IO_RX_EOF_GEN_SEL 配置 GDMA SUC EOF 的生成模式。

0: 通过配置的数据比特长度生成 GDMA SUC EOF

1: 通过外部使能信号生成 GDMA SUC EOF

(R/W)

Register 33.4. PARL_IO_RX_START_CFG_REG (0x000C)

PARL_IO_RX_START																														(reserved)			
31	30																																
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

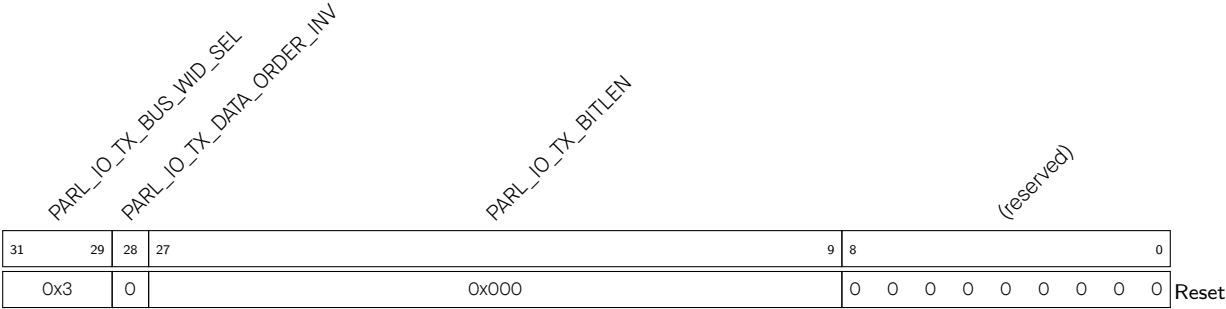
PARL_IO_RX_START 配置是否开始 RX 数据采集。

0: 无效

1: 开始

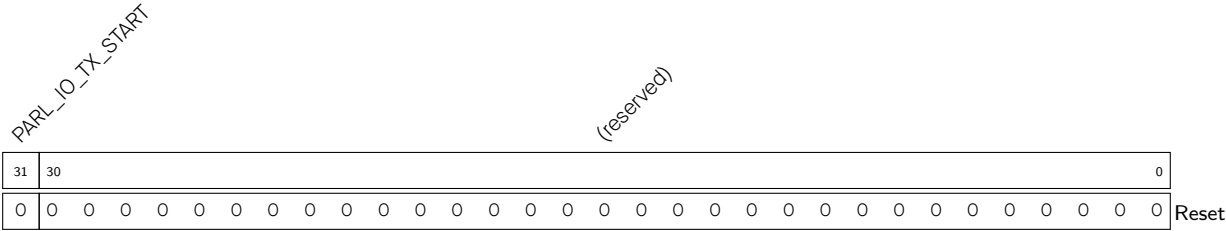
(R/W)

Register 33.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



- PARL_IO_TX_BITLEN** 配置 TXD 的比特长度。(R/W)
- PARL_IO_TX_DATA_ORDER_INV** 配置是否翻转从 TX_FIFO 发送到 IO 的数据的字节范围内比特顺序。
- 0: 无效
 - 1: 翻转数据顺序
- (R/W)
- PARL_IO_TX_BUS_WID_SEL** 配置 TXD 总线位宽。
- 0: 1 位
 - 1: 2 位
 - 2: 4 位
 - 3: 8 位
 - 4 ~ 15: 无效值，会引发错误
- (R/W)

Register 33.6. PARL_IO_TX_START_CFG_REG (0x0014)



- PARL_IO_TX_START** 配置是否开始 TX 数据输出。
- 0: 无效
 - 1: 开始
- (R/W)

Register 33.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)

31		30		29										14		13		12										0	
PARL_IO_TX_VALID_OUTPUT_EN		PARL_IO_TX_GATING_EN		PARL_IO_TX_IDLE_VALUE										PARL_IO_TX_EOF_GEN_SEL		(reserved)													
0	0	0x00										0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PARL_IO_TX_EOF_GEN_SEL 配置 TX EOF 的生成模式。

- 0: 通过配置的数据比特长度生成 TX EOF
- 1: 通过 GDMA EOF 信号生成 TX EOF
- (R/W)

PARL_IO_TX_IDLE_VALUE 配置 TX 总线上在未发送状态下的数据值。(R/W)

PARL_IO_TX_GATING_EN 配置是否使能 TX 输出时钟的时钟门控。

- 0: 不使能
- 1: 使能
- (R/W)

PARL_IO_TX_VALID_OUTPUT_EN 配置使能 DMA 输出数据或者片选信号作为 TXD 最高位控制源。

- 0: 选择 DMA 输出数据为控制源
- 1: 选择片选信号为控制源
- (R/W)

Register 33.8. PARL_IO_FIFO_CFG_REG (0x001C)

31		30		29																								0		
PARL_IO_RX_FIFO_SRST		PARL_IO_TX_FIFO_SRST		(reserved)																										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

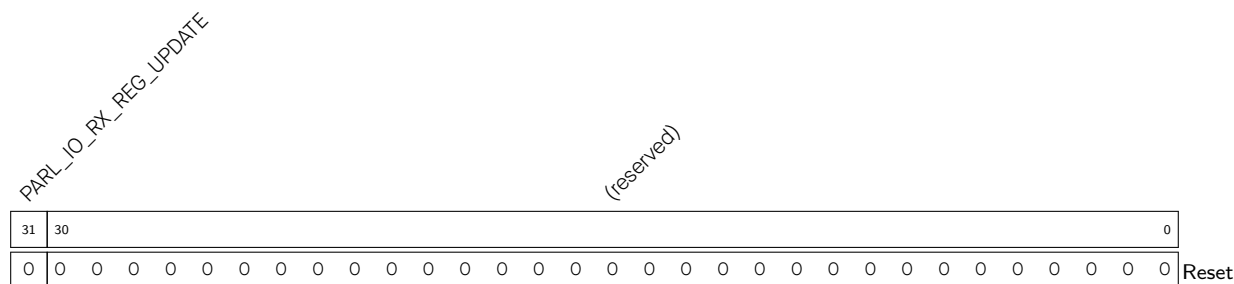
PARL_IO_TX_FIFO_SRST 配置是否复位 TX 异步 FIFO。

- 0: 无效
- 1: 复位
- (R/W)

PARL_IO_RX_FIFO_SRST 配置是否复位 RX 异步 FIFO。

- 0: 无效
- 1: 复位
- (R/W)

Register 33.9. PARL_IO_REG_UPDATE_REG (0x0020)



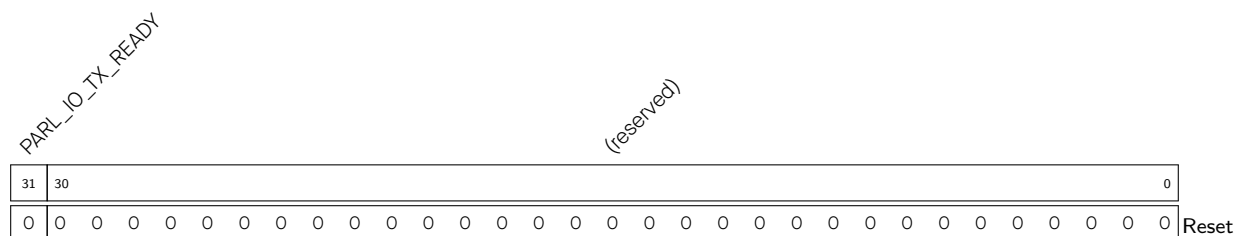
PARL_IO_RX_REG_UPDATE 配置是否更新 RX 寄存器配置。

0: 无效

1: 更新

(WT)

Register 33.10. PARL_IO_ST_REG (0x0024)



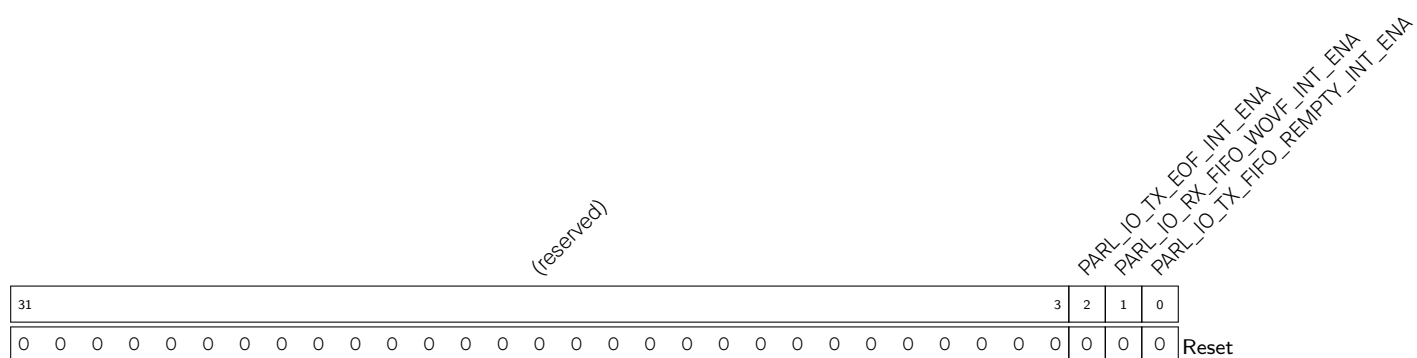
PARL_IO_TX_READY 表示 TX 模块是否已准备好传输数据。

0: 未准备好

1: 已准备好

(RO)

Register 33.11. PARL_IO_INT_ENA_REG (0x0028)

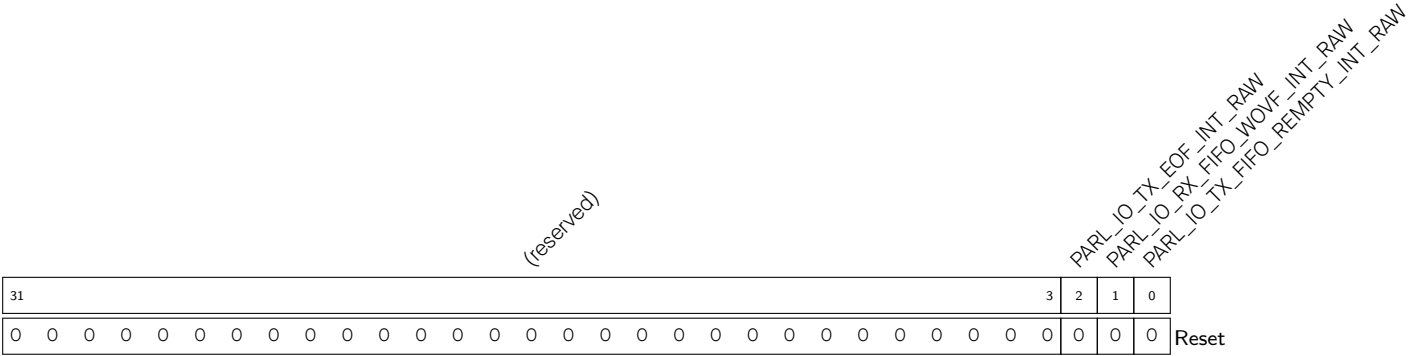


PARL_IO_TX_FIFO_EMPTY_INT_ENA 写1使能 **TX_FIFO_EMPTY_INT** 的屏蔽中断状态。(R/W)

PARL_IO_RX_FIFO_WOVF_INT_ENA 写1使能RX_FIFO_WOVF_INT的屏蔽中断状态。(R/W)

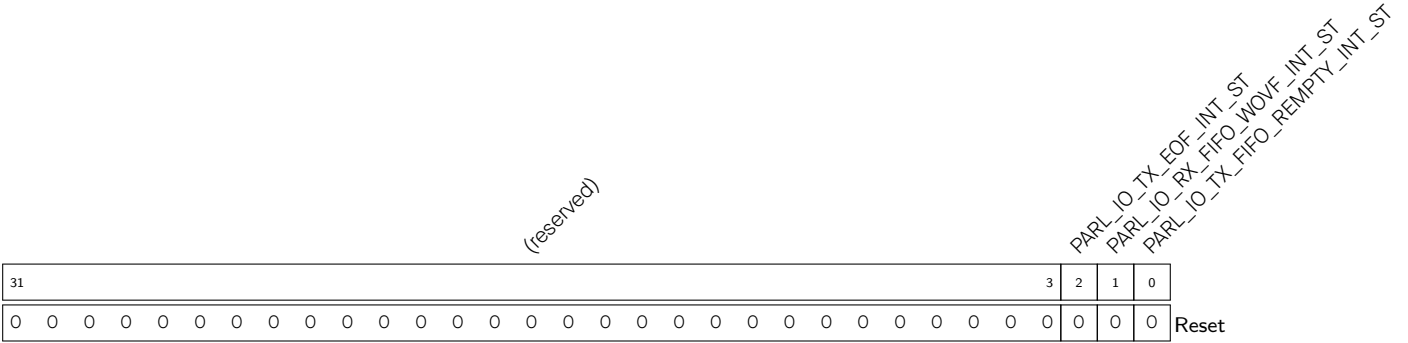
PARL IO TX EOF INT ENA 写1使能TX_EOF_INT的屏蔽中断状态。(R/W)

Register 33.12. PARL_IO_INT_RAW_REG (0x002C)



- PARL_IO_TX_FIFO_EMPTY_INT_RAW TX_FIFO_EMPTY_INT 的原始中断状态。(R/SS/WTC)
- PARL_IO_RX_FIFO_WOVF_INT_RAW RX_FIFO_WOVF_INT 的原始中断状态。(R/SS/WTC)
- PARL_IO_TX_EOF_INT_RAW TX_EOF_INT 的原始中断状态。(R/SS/WTC)

Register 33.13. PARL_IO_INT_ST_REG (0x0030)



- PARL_IO_TX_FIFO_EMPTY_INT_ST TX_FIFO_EMPTY_INT 的屏蔽中断状态。(RO)
- PARL_IO_RX_FIFO_WOVF_INT_ST RX_FIFO_WOVF_INT 的屏蔽中断状态。(RO)
- PARL_IO_TX_EOF_INT_ST TX_EOF_INT 的屏蔽中断状态。(RO)

Register 33.14. PARL_IO_INT_CLR_REG (0x0034)

(reserved)																												PARL_IO_TX_EOF_INT_CLR PARL_IO_RX_FIFO_WOVF_INT_CLR PARL_IO_TX_FIFO_EMPTY_INT_CLR			
31																												3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

- PARL_IO_TX_FIFO_EMPTY_INT_CLR 写 1 清除 TX_FIFO_EMPTY_INT。(WT)
- PARL_IO_RX_FIFO_WOVF_INT_CLR 写 1 清除 RX_FIFO_WOVF_INT。(WT)
- PARL_IO_TX_EOF_INT_CLR 写 1 清除 TX_EOF_INT。(WT)

Register 33.15. PARL_IO_RX_STO_REG (0x0038)

PARL_IO_RX_FIFO_WR_BIT_CNT												PARL_IO_RX_CNT								(reserved)							
31	13											12	8				7	0									
0x000												0x0				0	0	0	0	0	0	0	0	0	0	Reset	

- PARL_IO_RX_CNT 表示读取 RX FIFO 的时钟周期数。(RO)
- PARL_IO_RX_FIFO_WR_BIT_CNT 表示当前写入 RX FIFO 的位数。(RO)

Register 33.16. PARL_IO_RX_ST1_REG (0x003C)

PARL_IO_RX_FIFO_RD_BIT_CNT																(reserved)																															
31																12																0															
0x000																0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																Reset															

- PARL_IO_RX_FIFO_RD_BIT_CNT 表示当前从 RX FIFO 中读取的位数。(RO)

Register 33.17. PARL_IO_TX_STO_REG (0x0040)

PARL_IO_TX_FIFO_RD_BIT_CNT																PARL_IO_TX_CNT										(reserved)																																																																													
31																13																12																6																5								0																															
0x000																																0x0																0																0								0								0								0								Reset							

PARL_IO_TX_CNT 表示读取 TX FIFO 的周期数。(RO)

PARL_IO_TX_FIFO_RD_BIT_CNT 表示当前从 TX FIFO 中读取的位数。(RO)

Register 33.18. PARL_IO_RX_CLK_CFG_REG (0x0044)

PARL_IO_RX_CLK_O_INV																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
PARL_IO_RX_CLK_I_INV																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
31	30	29																													0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PARL_IO_RX_CLK_I_INV 配置是否翻转 RX 的输入 Core 时钟。

0: 无效

1: 翻转

(R/W)

PARL_IO_RX_CLK_O_INV 配置是否翻转 RX 的输出 Core 时钟。

0: 无效

1: 翻转

(R/W)

Register 33.19. PARL_IO_TX_CLK_CFG_REG (0x0048)

PARL_IO_TX_CLK_I_INV 配置是否翻转 TX 的输入 Core 时钟。

0: 无效

1: 翻转

(R/W)

PARL_IO_TX_CLK_O_INV 配置是否翻转 TX 的输出 Core 时钟。

0: 无效

1: 翻转

(R/W)

Register 33.20. PARL_IO_TX_CS_CFG_REG (0x004C)

Diagram of the PARL_IO_TX_CS_STOP_DELAY register. The register is 32 bits wide, divided into two 16-bit fields. The top 16 bits are labeled PARL_IO_TX_CS_START_DELAY and the bottom 16 bits are labeled PARL_IO_TX_CS_STOP_DELAY. Both fields are currently set to 0x00. A Reset label is at the bottom right.

PARL_IO_TX_CS_STOP_DELAY 配置 TX 数据发送结束与 TX_CS_0 上升沿之间的延迟。

单位: TX Core 时钟周期

(R/W)

PARL_IO_TX_CS_START_DELAY 配置 TX_CS_O 下降沿与 TX 数据开始传输之间的延迟。

单位: TX Core 时钟周期

(R/W)

Register 33.21. PARL_IO_CLK_REG (0x0120)

PARL_IO_CLK_EN																																(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
31	30																																																															0	Reset																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PARL_IO_CLK_EN 寄存器时钟门控。

0: 只在读写寄存器时开启寄存器时钟

1: 始终开启寄存器时钟

(R/W)

Register 33.22. PARL_IO_VERSION_REG (0x03FC)

(reserved)				PARL_IO_DATE																																Reset
31	28	27																																	0	
0	0	0	0	0x2409230																																

PARL_IO_DATE 版本控制寄存器。(R/W)

第 34 章

比特调节器

ESP32-C5 中有大量支持 DMA（直接存储器访问）的外设（详情请参考章节 3 通用 DMA 控制器 (GDMA)），它们可以在 CPU 不参与的情况下将数据从存储器传输至外设或从外设传输至存储器，但这需要外设传输的数据格式与软件支持的数据格式相同，如果格式不同，则需要 CPU 重写数据格式，如交换字节、反转字节和左右移位数据。

由于位操作通常相当耗费 CPU 资源，而设计 DMA 的初衷是在传输过程中避免使用 CPU，因此 ESP32-C5 集成了一个比特调节器 (BitScrambler)，专门用于修改存储器和外设之间数据传输的格式，支持的传输方向包括存储器到外设、存储器到存储器、外设到存储器。除此之外，比特调节器还是一个灵活的可编程状态机，能够执行更高级的操作。

34.1 概述

在 ESP32-C5 中，支持 DMA 的外设是指能够通过 DMA 通道直接从存储器读取数据和/或直接向存储器写入数据的外设。这些数据的读取/写入格式取决于外设，灵活性较低。对于 ESP32-C5 中大多数支持 DMA 的外设来说，可以在这些路径中挂载比特调节器来提高灵活性。

当某个外设挂载比特调节器后，比特调节器就可以修改与该外设连接的 DMA 通道上的任何数据。写入 DMA 通道的数据会出现在比特调节器的输入端，而比特调节器输出端的数据将沿着 DMA 通道继续传输。比特调节器内置指令存储器，存储的指令用于控制输入和输出数据格式的映射关系。因此，通过编写合适的比特调节器程序，可以让比特调节器改写源数据。

如图 34.1-1 所示，比特调节器可以放置在 DMA 的 TX 路径中，修改从存储器流向外设的数据；也可以放置在 DMA 的 RX 路径中，修改从外设流向存储器的数据；还可以配置为环回模式，用于修改从存储器流向存储器的数据。

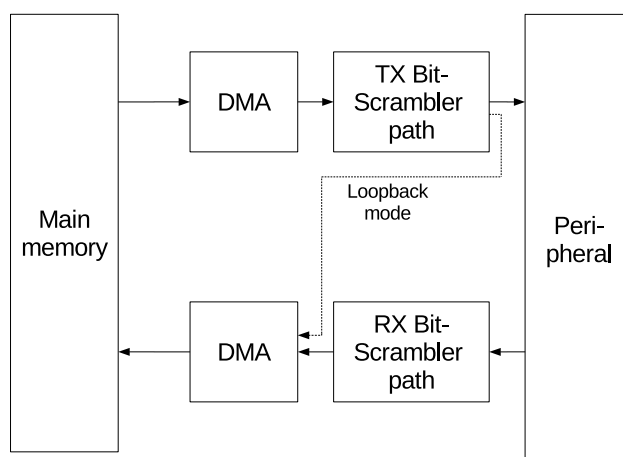


图 34.1-1. 比特调节器系统架构图

34.2 特性

比特调节器有以下特性：

- 一个比特调节器核心，可用于 RX 单元（外设到存储器）或 TX 单元（存储器到外设）
- 支持存储器到存储器的传输
- 每个 DMA 时钟周期最多可处理 32 位数据
- 数据路径由存储在指令存储器中的比特调节器程序控制
- 输入寄存器每个时钟周期可读取 0、8、16 或 32 位
- 输出寄存器：
 - 每个时钟周期可写入 0、8、16 或 32 位
 - 输出寄存器位的数据源：64 位输入数据、两个计数器、LUT RAM 数据、上个周期的数据输出、比较器
 - 32 位输出寄存器位中的每一位可以来自数据源的任意位
- 8 x 257 位指令存储器，用于存储八条指令，配置控制流和数据路径
- 2048 字节查找表（LUT）存储器，可配置为不同的字宽

34.3 应用示例

- 数据中字节、半字节或位顺序的交换
- 将基于调色板的位图转换为全 RGB 位图
- 数据的游程编码 (run-length encoding) 或解码
- 生成复杂波形，例如用于 WS2811 芯片

34.4 架构

比特调节器可以分为数据路径、控制路径以及寄存器，CPU 可通过这些寄存器配置数据路径和控制路径。数据路径在 DMA 输入流、内部寄存器和 DMA 输出流之间传输比特。控制路径解析指令存储器中的指令，以控制数据路径，并处理程序流程。

34.4.1 数据路径

比特调节器的数据路径从 DMA 输入流 (DMA incoming stream) 中获取数据，根据存储在指令存储器中的程序代码处理数据，然后将其写入 DMA 输出流 (DMA outgoing stream) 中。

1. 数据从 DMA 输入流保存到一个 64 位寄存器中。在每个指令周期结束时，根据指令的配置从 DMA 输入流中读取 N 位（N 可以是 0、8、16 或 32）。
2. 寄存器中的数据向寄存器的最低有效位（LSB）方向移动，最低有效位的 N 位消失。
3. 将最新读取的数据保存在最高的 N 位。

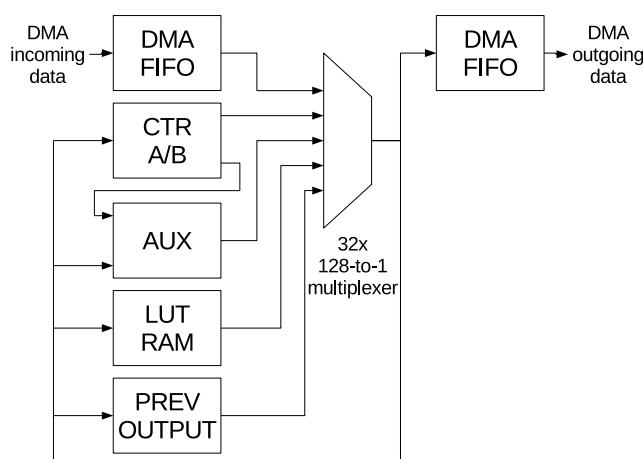


图 34.4-1. 比特调节器数据路径图

或者在比特调节器启动时，可以选择预先自动读取前 64 位到寄存器中。

在比特调节器的另一端，数据存入一个 32 位输出寄存器。在每个指令周期结束时，根据指令的配置将最低的 0、8、16 或 32 位写入 DMA 输出流。

比特调节器的名称来源于其可以将输入比特放置在输出中的任意位置，这是通过 32 个 128 选 1 多路复用器 (128-to-1 multiplexer) 来实现的。对于输出寄存器中的 32 位的每一位，都有一个多路复用器根据比特调节器控制路径的输出，从相应的数据源中选择输入信号。这些数据源包括：

- 从输入 FIFO 移入的 64 位数据（即图 34.4-1 中左侧的 DMA FIFO 模块）。如前所述，来自 DMA 输入流的数据被存储在此处。为了便于使用循环来遍历输入的 64 位数据，指令可以启用相对寻址。在这种寻址模式下，任何从输入 FIFO 寄存器获取比特的多路复用器实际上会获取偏移计数器 A 寄存器值的比特（即图 34.4-1 中的 CTR A）。
- 从上一个周期发送到输出的 32 位数据（即图 34.4-1 中的 PREV OUTPUT 模块）。发送到输出寄存器的数据将在一个时钟周期后出现在这个寄存器中。这使得比特调节器可以在多个时钟周期中“记住”数据：即使某个比特没有发送到输出 FIFO，只要是输出到了输出寄存器，下一个时钟周期仍然可以通过选择这个寄存器来访问该比特。通过再次从这个寄存器中选择比特进行输出，可以实现比特的长期记忆，使它们在随后的周期中出现在此寄存器中。
- 由 LUT RAM 输出的 8、16 或 32 位数据。当 LUT 宽度为 32 位、16 位、8 位时，则数据地址分别是上一周期输出数据的第 16 到 24、25、26 位。LUT RAM 是比特调节器的一部分，它首先接收到某个地址，然后在 LUT RAM 中查找该地址对应的数据，最后输出该数据。比特调节器初始化时可填充 LUT RAM，但比特调节器不能修改 LUT RAM。可根据需求将 LUT RAM 配置成以下模式进行初始化：
 - 512 x 32 位字
 - 1024 x 16 位字
 - 2048 x 8 位字

LUT RAM 可以用于数据的转换，例如用于某传感器的校准曲线，也可用于实现数学函数：将地址输入拆分成两个或两个以上的输入变量，数据输出作为函数的输出；存储器应加载每个输入值对应的函数输出。

- 2 x 16 位计数器提供的 32 位数据（即图 34.4-1 中的 CTR A/B 模块）。该寄存器包含两个 16 位计数器，分别命名为计数器 A 和 B，可通过控制路径指令对它们进行加载、递增和递减等操作。
- 计数器 B 和之前输出数据（对应图 34.4-1 中的 PREV OUT）之间的比较器中包含的 30 位数据，属于 AUX

模块的一部分比较结果可用于输出或程序流程，例如在解码游程编码流 (run-length encoded stream) 或生成 PWM 信号时非常有用。

- 2 位数据（属于图 34.4-1 中的 AUX 模块），一位固定为高电平，一位固定为低电平。使用场景：某协议需要输出一个始终为高电平或低电平的位，不受输入数据影响。

请注意，某些数据源无法同时访问。具体而言，指令无法同时从输入 FIFO 源寄存器的高 32 位以及计数器寄存器中获取数据。当从 LUT 读取数据时，对于宽度为 N 的 LUT，输入 FIFO 寄存器的最高 N 位和计数器寄存器的最高 N 位不可用。

34.4.2 控制路径

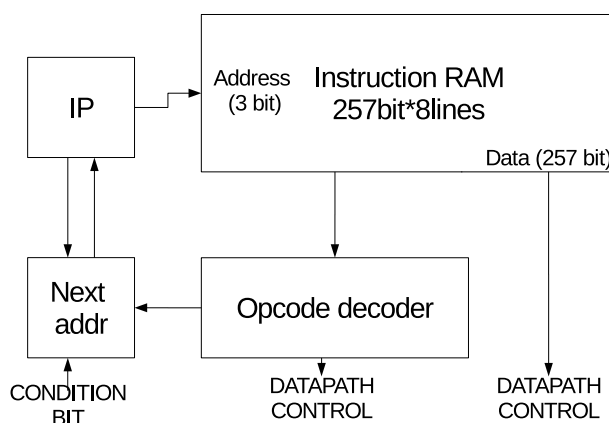


图 34.4-2. 比特调节器控制路径图

比特调节器的行为由存储在指令 RAM 中的程序控制。指令 RAM 最多可以存储 8 条指令，每条指令为 257 位。这些指令可配置该指令运行时钟周期内的数据路径，以及控制程序流程的操作码。

比特调节器的程序和微处理器的程序类似，每个周期都会执行指令存储器中的下一条指令，除非操作码明确指示需要跳转到另一个位置。具体来说，这种行为是使用指令指针（IP，参见图 34.4-2）来实现的，指向当前正在执行的指令。除此之外，操作码还可以影响计数器寄存器。

在配置数据路径的指令位中，一些位用于配置 32 个多路复用器，另一些位用于配置计数器位或 LUT 输出是否可用。指令还可以将多路复用器设置为相对模式，在这种模式下，从 DMA FIFO 获取的比特的位置将被偏移计数器 B 位。

34.5 功能描述

比特调节器是一种灵活的设备，其行为可通过加载到控制器内部的程序来控制。此外，一些配置寄存器也可以全局配置控制器的一些行为。因此，比特调节器程序包括 257 位 x 8 的指令 RAM 以及各种配置寄存器的设置，下文将对二者进行介绍。

34.5.1 指令

表 34.5-1 为 257 位指令的结构。

表 34.5-1. 指令结构

位	域	位	域
0 - 6	CTL_MUX_SEL_0	133 - 139	CTL_MUX_SEL_19
7 - 13	CTL_MUX_SEL_1	140 - 146	CTL_MUX_SEL_20
14 - 20	CTL_MUX_SEL_2	147 - 153	CTL_MUX_SEL_21
21 - 27	CTL_MUX_SEL_3	154 - 160	CTL_MUX_SEL_22
28 - 34	CTL_MUX_SEL_4	161 - 167	CTL_MUX_SEL_23
35 - 41	CTL_MUX_SEL_5	168 - 174	CTL_MUX_SEL_24
42 - 48	CTL_MUX_SEL_6	175 - 181	CTL_MUX_SEL_25
49 - 55	CTL_MUX_SEL_7	182 - 188	CTL_MUX_SEL_26
56 - 62	CTL_MUX_SEL_8	189 - 195	CTL_MUX_SEL_27
63 - 69	CTL_MUX_SEL_9	196 - 202	CTL_MUX_SEL_28
70 - 76	CTL_MUX_SEL_10	203 - 209	CTL_MUX_SEL_29
77 - 83	CTL_MUX_SEL_11	210 - 216	CTL_MUX_SEL_30
84 - 90	CTL_MUX_SEL_12	217 - 223	CTL_MUX_SEL_31
91 - 97	CTL_MUX_SEL_13	224 - 249	OPCODE
98 - 104	CTL_MUX_SEL_14	250 - 251	CTL_READ_IN
105 - 111	CTL_MUX_SEL_15	252 - 253	CTL_WR_OUT
112 - 118	CTL_MUX_SEL_16	254	CTL_MUX_REL
119 - 125	CTL_MUX_SEL_17	255	CTL_CTR_SRC_SEL
126 - 132	CTL_MUX_SEL_18	256	CTL_LUT_SRC_SEL

下文介绍指令各域的含义。

- CTL_MUX_SEL_n: 包含 7 位，选择输出寄存器中第 n 位的来源。详情参见表 34.5-3。
- OPCODE: 操作码，该域的位格式请参见表 34.5-4。
- CTL_READ_IN: 在指令结束时选择从输入 FIFO 读取的位数。
 - 0: 不读取任何数据
 - 1: 读取 8 位数据
 - 2: 读取 16 位数据
 - 3: 读取 32 位数据
- CTL_WR_OUT: 选择从输出寄存器写入到输出 FIFO 的位数:
 - 0: 不写入任何数据
 - 1: 写入 8 位数据
 - 2: 写入 16 位数据
 - 3: 写入 32 位数据
- CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL: 配置哪些位可以通过 CTL_MUX_SEL 域来选择。
 - CTRL_CTR_SRC_SEL (控制计数器源选择) 配置计数器 A 和 B 的位是否可用。

- CTL_LUT_SRC_SEL（控制 LUT 源选择）配置 LUT RAM 的输出值是否可用。详见表 34.5-3。
- 当 CTL_MUX_REL（控制多路复用器相对寻址）为 1 时，且 CTL_MUX_SEL_n 小于等于 63 时，比特调节器将使用计数器 A 重新计算有效的 CTL_MUX_SEL_n。具体行为取决于 CTR_SRC_SEL，详见表 34.5-2。

表 34.5-2. 当 CTL_MUX_REL 为 1 时 CTL_MUX_SEL_n 生效的值

条件	CTL_MUX_SEL 生效的值
CTL_MUX_SEL_n >= 64	CTL_MUX_SEL（不变）
CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 0	(CTL_MUX_SEL_n + CTRA) & 63
CTL_MUX_SEL_n < 31 and CTL_CTR_SRC_SEL = 1	(CTL_MUX_SEL_n + CTRA) & 31
32 <= CTL_MUX_SEL_n < 64 and CTL_CTR_SRC_SEL = 1	((CTL_MUX_SEL_n + CTRA) & 31) + 32

这些域中，大部分是数据路径配置项，其中 OPCODE 是影响控制路径的配置项，但 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 可通过调整 IF/IFN 操作码的来源来影响某些操作码的工作方式。

CTL_MUX_SEL_n: 每个 CTL_MUX_SEL_n 域都可用于选择输出寄存器中与之相对应的位的来源。具体选择哪个位在一定程度上取决于 CTL_MUX_REL、CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置。假设 CTL_MUX_SRC 的值为 N，选择该位的方式如表 34.5-3 所示。请注意，IF/IFN 操作码的数据源选择也采用相同的方式。

表 34.5-3. CTL_MUX_SEL 的值

位	描述
0-31	该位来自输入 FIFO 寄存器中的第 N 位。
32-63	该位的来源取决于 CTL_CTR_SRC_SEL 和 CTR_LUT_SRC_SEL 的设置 如果 CTL_CTR_SRC_SEL = 0，该位来源于输入 FIFO 寄存器中的第 N 位。 如果 CTL_CTR_SRC_SEL = 1，该位来源于计数器寄存器中的第 N-32 位。具体来说，如果 N 是 32 到 47，则该位来源于计数器 A 的第 N-32 位；如果 N 是 48 到 63，则该位来源于计数器 B 的第 N-48 位。 如果 CTL_LUT_SRC_SEL = 1，根据 LUT 的宽度（8、16 或 32 位），读取最高 8、16 或 32 位将返回来自 LUT 的数据，而不是 CTL_CTR_SRC_SEL 选择的数据。具体来说，假定 LUT 宽度为 N，63 将返回 LUT 输出的第 N-1 位，62 将返回 LUT 输出的第 N-2 位，依此类推。
64	CounterB[7:0] =< last[7:0]
65	CounterB[7:0] > last[7:0]
66	CounterB[7:0] = last[7:0]
67	CounterB[7:0] =< last[15:8]
68	CounterB[7:0] > last[15:8]
69	CounterB[7:0] = last[15:8]
70	CounterB[7:0] =< last[23:16]
71	CounterB[7:0] > last[23:16]
72	CounterB[7:0] = last[23:16]
73	CounterB[7:0] =< last[31:24]
74	CounterB[7:0] > last[31:24]
75	CounterB[7:0] = last[31:24]
76	CounterB[15:8] =< last[7:0]
77	CounterB[15:8] > last[7:0]
78	CounterB[15:8] = last[7:0]
79	CounterB[15:8] =< last[15:8]
80	CounterB[15:8] > last[15:8]
81	CounterB[15:8] = last[15:8]
82	CounterB[15:8] =< last[23:16]
83	CounterB[15:8] > last[23:16]
84	CounterB[15:8] = last[23:16]
85	CounterB[15:8] =< last[31:24]
86	CounterB[15:8] > last[31:24]
87	CounterB[15:8] = last[31:24]
88	CounterB[15:0] =< last[15:0]
89	CounterB[15:0] > last[15:0]
90	CounterB[15:0] = last[15:0]
91	CounterB[15:0] =< last[31:16]
92	CounterB[15:0] > last[31:16]
93	CounterB[15:0] = last[31:16]
94	始终为 0
95	始终为 1
96-127	这些位与上一周期结束时输出寄存器上的位相同

比特调节器可识别七种操作码，编码格式如表 34.5-4 所示。

表 34.5-4. 指令操作码

操作码	位	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LOOP		1	c	tgt			end_val																ctr_add				
ADD		0	c	h	l	0	0	0	0	0	ctr_add																
IF		0	0	tgt			0	0	0	0	1	0	0	0	0	0	0	0	0	0	src						
IFN		0	0	tgt			0	0	0	1	0	0	0	0	0	0	0	0	0	0	src						
LDCTD		0	c	h	l	0	0	0	0	1	1	ctr_set															
LDCTI		0	c	h	l	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ADDCTI		0	c	h	l	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

操作码域的定义如下：

- c：配置使用哪个计数器。
 - 0：计数器 A
 - 1：计数器 B
- end_val, ctr_add, ctr_set：16 位（或 5 位）值。这些域的含义请参阅下面的指令描述。
- tgt：指令的位置。范围：0 到 7。
- h：如果需要将高 8 位写回计数器，则为 1（如果操作码以 H 结尾）。
- l：如果需要将低 8 位写回计数器，则为 1（如果操作码以 L 结尾）。

请注意，如果操作码没有指定 H 或 L，则默认需要将 16 位全部写回，此时 h 和 l 位都会为 1。

下方表格主要介绍这七个操作码如何影响比特调节器的状态。

表 34.5-5. LOOP 操作码

LOOPc end_val, ctr_add, tgt	
使用 c 参数设置的计数器进行循环，将 ctr_add 加到计数器中，并循环回到 tgt 位置，直到计数器达到 end_val。当计数器达到 end_val 时，重置计数器并继续执行。	
参数	c：计数器 A 或 B，表示要使用的计数器 tgt：3 位值，表示目标指令地址 end_val：16 位值，用于与计数器进行比较 ctr_add：5 位有符号值，用于加到计数器上
伪代码	<pre>if ((unsigned)ctrC < end_val) begin IP <= tgt ctr_c <= ctr_c + sign_extend(ctr_add) end else begin IP <= IP + 1 ctr_c <= 0 end</pre>

表 34.5-6. ADD 操作码

ADDc[H L] ctr_add	
将立即数加到计数器上，可以选择只写回高 8 位或低 8 位。	
参数	c: 计数器 A 或 B，表示要使用的计数器 ctr_add: 16 位值，加到计数器上的值 如果操作码是 ADDcH，则只写回最高 8 位 如果操作码是 ADDcL，则只写回最低 8 位 如果操作码是 ADDc，则 16 位全部都将被写回 (h=1, l=1)
伪代码	<pre> tmp <= ctr_c + ctr_add if (h) begin ctr_c[15:8] <= tmp[15:8] end if (l) begin ctr_c[7:0] <= tmp[7:0] end IP <= IP + 1 </pre>
说明	NOP（无操作）伪操作码为“ADDA 0”。

表 34.5-7. IF 操作码

IF ctl_cond_src, tgt	
如果 src 域对应的多路复用器值为 1，则跳转到目标。	
参数	ctl_cond_src: 多路复用器来源，可参考表 CTR_MUX_SRC 来源 tgt: 3 位值，表示目标指令地址
伪代码	<pre> if (ctl_cond_src) begin IP <= tgt end else begin IP <= IP + 1 end </pre>
说明	JMP（始终跳转）伪操作码为“IF 95,tgt”，使用 AUX 位中始终为 1 的位。

表 34.5-8. IFN 操作码

IFN ctl_cond_src, tgt	
如果指定的多路复用器源位未设置，则跳转到目标（注：与 IF 操作相同，但条件取反）。	
参数	ctl_cond_src: 多路复用器来源，可参考表 CTR_MUX_SRC sources tgt: 3 位值，表示目标指令地址
伪代码	<pre> if (ctl_cond_src) begin IP <= IP + 1 end else begin IP <= tgt end </pre>

表 34.5-9. LDCTD 操作码

LDCTDc[H L] ctr_set	
将立即数直接加载到计数器，可以选择只加载计数器的高 8 位或低 8 位。	
参数	c: 计数器 A 或 B，表示要使用的计数器 ctr_set: 16 位值，要设置的计数器的值 h: 如果操作码是 LDCTDcH，则只设置最高 8 位 l: 如果操作码是 LDCTDcL，则只设置最低 8 位 如果操作码是 LDCTDc，则 16 位全部都将被设置 (h=1, l=1)
伪代码	<pre>if (h) begin ctr_c[15:8] <= ctr_set[15:8] end if (l) begin ctr_c[7:0] <= ctr_set[7:0] end IP <= IP + 1</pre>

表 34.5-10. LDCTI 操作码

LDCTIc[H L]	
将多路复用器输出的高 16 位间接加载到计数器，可以选择只加载到计数器的高 8 位或低 8 位。	
参数	c: 计数器 A 或 B，表示要使用的计数器 h: 如果操作码是 LDCTIcH，则只设置最高 8 位 l: 如果操作码是 LDCTIcL，则只设置最低 8 位 如果操作码是 LDCTIc，则 16 位全部都将被设置 (h=1, l=1)
伪代码	<pre>if (h) begin ctr_c[15:8] <= mux_out[31:24] end if (l) begin ctr_c[7:0] <= mux_out[23:16] end IP <= IP + 1</pre>

表 34.5-11. ADDCTI Opcode

ADDCTIc[H L]	
将多路复用器输出的高 16 位间接加到计数器，可以选择只加到计数器的高 8 位或低 8 位。	
参数	c: 计数器 A 或 B，表示要使用的计数器 h: 如果操作码是 ADDCTIcH，将多路复用器输出的 24–31 位加到计数器的高 8 位 l: 如果操作码是 ADDCTIcL，将多路复用器输出的 16–23 位加到计数器的低 8 位 如果操作码是 ADDCTIc，将多路复用器输出的 16–31 位加到计数器的全部 16 位 (h=1, l=1)
伪代码	<pre> if (h) begin ctr_c[15:8] <= ctr_c[15:8] + mux_out[31:24] end else if (l) begin ctr_c[7:0] <= ctr_c[7:0] + mux_out[23:16] end else begin ctr_c[15:0] <= ctr_c[15:0] + mux_out[31:16] end IP <= IP + 1 </pre>

34.5.2 配置寄存器

比特调节器的配置是通过一组寄存器实现的。这些寄存器可向比特调节器写入程序、写入 LUT RAM、配置控制器行为、以及启动和停止控制器的操作，还可将控制器挂载到外设的 DMA 路径。

ESP32-C5 只有一个比特调节器核心，可放置在 TX 路径（数据从存储器发送到外设）或 RX 路径（数据从外设发送到存储器）。为了兼容其他 ESP 系列芯片，当比特调节器核心处于 TX 路径时，则由 BITSCRAMBLER_TX_ 为前缀的寄存器控制；处于 RX 路径时，则由 BITSCRAMBLER_RX_ 为前缀的寄存器控制。哪个路径处于激活状态由 BITSCRAMBLER_RX_ENA 和 BITSCRAMBLER_TX_ENA 决定，且在任意时刻只允许其中一个处于激活状态。

说明：

为方便阐述，本节仅使用 TX 路径寄存器进行描述，若需 RX 路径寄存器的说明，只需替换描述中的前缀即可。

当比特调节器修改某外设的输入或输出时，需要挂载到该外设的 DMA 路径。为此，需将该外设的对应值写入 HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG 寄存器的 HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL 域或 HP_SYSTEM_BITSCRAMBLER_PERI_RX_SEL 域。

比特调节器也可以在存储器到存储器模式下运行，但需要将 BITSCRAMBLER_SYS_REG 寄存器中的 BITSCRAMBLER_LOOP_MODE 域置为 1。该模式下，TX 比特调节器进入环回模式，而 RX 比特调节器则未使用且无法使用。并且比特调节器仍然需要连接到外设，但该外设无需配置，而且对应的外设 DMA 路径将无法使用。

主处理器无法直接访问比特调节器的程序存储器和 LUT RAM，但可以间接访问，即在一个寄存器中设置地址，然后在另一个寄存器中写入数据。

具体而言，对于 LUT RAM，地址写入 BITSCRAMBLER_TX_LUT_CFG0_REG 的 BITSCRAMBLER_TX_LUT_IDX 域，数据写入 BITSCRAMBLER_TX_LUT_CFG1_REG 寄存器，或从该寄存器读取数据。LUT 的位宽对比特调节器可见的，可以使用 BITSCRAMBLER_TX_LUT_MODE 域进行配置。需要注意的是，当主处理器以上述方式写入 LUT 时，地址和数据大小需要符合配置的 LUT 位宽。

同样，对于指令存储器，地址写入 `BITSCRAMBLER_TX_INST_CFG0_REG`，数据写入或从 `BITSCRAMBLER_TX_INST_CFG1_REG` 中读取。由于比特调节器指令为 257 位，因此一个指令需要写入 9 个 32 位字，第 257 位位于第 9 个字的最低有效位。为此，`BITSCRAMBLER_TX_INST_CFG0_REG` 寄存器被分为两个域，指令的位置写入 `BITSCRAMBLER_TX_INST_IDX` 域，而指令内的字偏移写入 `BITSCRAMBLER_TX_INST_POS` 域。

由于比特调节器生成的数据可以比输入多也可以比输入少，所以它控制的 DMA 流可以以一种或两种方式结束。第一种情况是接收器（在 RX 比特调节器的情况下是存储器，而在 TX 比特调节器的情况下是外设）停止接收数据，因为它的配置是只接收一定数量的字节。这种情况比较简单，比特调节器因为无法再写入数据而停止。另一种方式是发射器（在 RX 比特调节器的情况下是外设，在 TX 比特调节器的情况下是存储器）停止发送数据。这种情况更加复杂，因为比特调节器可能正在处理需要写入输出的一些数据，也可能不在处理。

ESP32-C5 比特调节器设计了两种方法来处理输入数据流结束后的一定数量的数据，目的是为了完整地处理这些数据。这两种方法依赖于 `BITSCRAMBLER_RX_TAILING_BITS_REG` 寄存器设置的具体值 N：

- EOF-on-N-reads（读取 N 字节后结束）：在输入数据流结束后，比特调节器从输入读取 N 个虚拟字节。这些字节的值为零。在读取完第 N 个字节后，比特调节器停止，DMA 流结束。
- EOF-on-N-writes（写入 N 字节后结束）：在输入数据流结束后，比特调节器允许向输出写入 N 个字节。在此期间，任何输入流上的读取都会返回值为零的虚拟字节。在写入完第 N 个字节后，比特调节器停止，DMA 流结束。

请按照下表内容配置上述两种模式。

表 34.5-12. EOF 模式设置

寄存器	EOF-on-N-reads	EOF-on-N-writes
<code>BITSCRAMBLER_TX_RD_DUMMY</code>	1	1
<code>BITSCRAMBLER_TX_FETCH_MODE</code>	0	0
<code>BITSCRAMBLER_TX_EOF_MODE</code>	1	0
<code>BITSCRAMBLER_TX_HALT_MODE</code>	1	1

34.6 配置流程

请注意，下文描述的是 TX 路径使能时比特调节器的配置流程，路径方向为存储器到外设方向或者处于环回模式下存储器到存储器方向。对于外设到存储器方向的路径，则使用比特调节器的 RX 路径。除非另作说明，否则将下文描述中的 TX 替换为 RX 即为比特调节器 RX 路径的配置流程。

1. 配置 `HP_SYSTEM_BITSCRAMBLER_PERI_SEL_REG` 寄存器中的 `HP_SYSTEM_BITSCRAMBLER_PERI_TX_SEL` 域将比特调节器挂载到外设。请注意，即使在环回模式下，此步骤也是必须的。
2. 配置 `BITSCRAMBLER_TX_CTRL_REG` 寄存器中的 `BITSCRAMBLER_TX_ENA` 域使能比特调节器。请确保任意时刻 `BITSCRAMBLER_TX_ENA` 和 `BITSCRAMBLER_RX_ENA` 中只能有一个寄存器的值为 1。
3. 根据需要决定是否配置环回模式。如果需要，将 `BITSCRAMBLER_SYS_REG` 寄存器的 `BITSCRAMBLER_LOOP_MODE` 域置 1，否则清零。
4. 将程序加载到比特调节器中。对于每条指令和指令内的每个 32 位字，设置 `BITSCRAMBLER_TX_INST_CFG0_REG` 寄存器中的 `BITSCRAMBLER_TX_INST_IDX` 和

`BITSCRAMBLER_TX_INST_POS` 域，然后将字写入 `BITSCRAMBLER_TX_INST_CFG1_REG` 寄存器。注意，第 257 位写入到 `BITSCRAMBLER_TX_INST_CFG1_REG` 的第 0 位。

5. 如果需要，将 LUT 挂载到比特调节器中：

- (a) 配置 `BITSCRAMBLER_TX_LUT_CFG0_REG` 寄存器中 `BITSCRAMBLER_TX_LUT_MODE` 域来设置 LUT 宽度
- (b) 将地址写入 `BITSCRAMBLER_TX_LUT_CFG0_REG`
- (c) 将数据字或字节写入 `BITSCRAMBLER_TX_LUT_CFG1_REG`

请注意，在写入数据之后和启动比特调节器之前更改 LUT 宽度都是可以的。

- 6. 配置比特调节器。根据比特调节器程序的需求，设置或清除 `BITSCRAMBLER_TX_CTRL_REG` 寄存器中的相应位，也可将 `BITSCRAMBLER_TX_TAILING_BITS_REG` 设置为适当的值。
- 7. 清除 `BITSCRAMBLER_TX_CTRL_REG` 寄存器的 `BITSCRAMBLER_TX_HALT` 域，以启动比特调节器。
- 8. 启动 DMA 传输。有关 DMA 子系统的更多信息，请参阅章节 3 通用 DMA 控制器 (GDMA)，如果未使用换回模式，请参阅各外设章节。
- 9. 等待 DMA 传输完成。
- 10. 设置 `BITSCRAMBLER_TX_CTRL_REG` 寄存器中的 `BITSCRAMBLER_TX_HALT` 域停止比特调节器。
- 11. 等待比特调节器处于停止状态。当 `BITSCRAMBLER_TX_STATE_REG` 寄存器中的 `BITSCRAMBLER_TX_IN_IDLE` 位被置位时，比特调节器停止。
- 12. 在 `BITSCRAMBLER_TX_CTRL_REG` 寄存器中的 `BITSCRAMBLER_TX_FIFO_RST` 位先写入 1 然后再写入 0 来复位 FIFO。
- 13. 比特调节器可以使用当前的程序和 LUT 配置开始进行新的传输，从步骤 7 开始即可。

34.7 寄存器列表

本小节的所有地址均为相对于比特调节器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

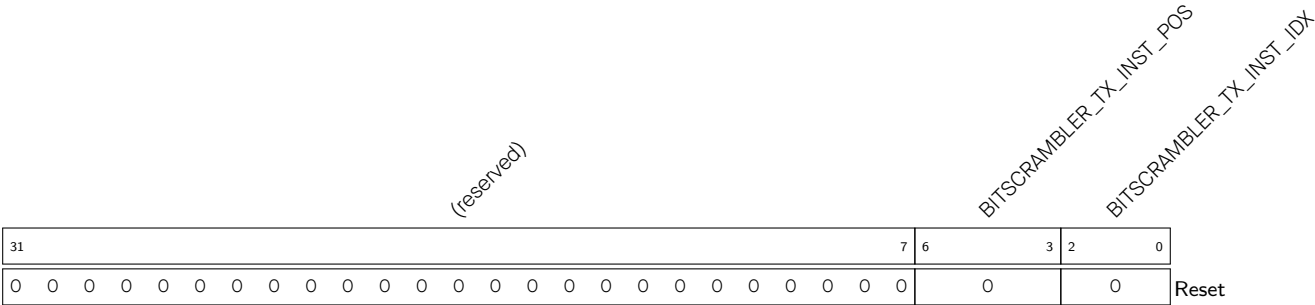
名称	描述	地址	访问
控制和配置寄存器			
BITSCRAMBLER_TX_INST_CFG0_REG	比特调节器 TX 路径指令存储器地址选择寄存器	0x0000	R/W
BITSCRAMBLER_TX_INST_CFG1_REG	比特调节器 TX 路径指令存储器访问寄存器	0x0004	R/W
BITSCRAMBLER_RX_INST_CFG0_REG	比特调节器 RX 路径指令存储器地址选择寄存器	0x0008	R/W
BITSCRAMBLER_RX_INST_CFG1_REG	比特调节器 RX 路径指令存储器访问寄存器	0x000C	R/W
BITSCRAMBLER_TX_LUT_CFG0_REG	比特调节器 TX 路径 LUT 存储器地址选择寄存器	0x0010	R/W
BITSCRAMBLER_TX_LUT_CFG1_REG	比特调节器 TX 路径 LUT 存储器访问寄存器	0x0014	R/W
BITSCRAMBLER_RX_LUT_CFG0_REG	比特调节器 RX 路径 LUT 存储器地址选择寄存器	0x0018	R/W
BITSCRAMBLER_RX_LUT_CFG1_REG	比特调节器 RX 路径 LUT 存储器访问寄存器	0x001C	R/W
配置寄存器			
BITSCRAMBLER_TX_TAILING_BITS_REG	比特调节器 TX 路径额外数据长度寄存器	0x0020	R/W
BITSCRAMBLER_RX_TAILING_BITS_REG	比特调节器 RX 路径额外数据长度寄存器	0x0024	R/W
BITSCRAMBLER_TX_CTRL_REG	比特调节器 TX 路径控制寄存器	0x0028	varies
BITSCRAMBLER_RX_CTRL_REG	比特调节器 RX 路径控制寄存器	0x002C	varies
BITSCRAMBLER_SYS_REG	环回控制寄存器	0x00F8	R/W
状态寄存器			
BITSCRAMBLER_TX_STATE_REG	比特调节器 TX 路径状态寄存器	0x0030	varies
BITSCRAMBLER_RX_STATE_REG	比特调节器 RX 路径状态寄存器	0x0034	varies
版本寄存器			
BITSCRAMBLER_VERSION_REG	版本控制寄存器	0x00FC	R/W

34.8 寄存器

本小节的所有地址均为相对于比特调节器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

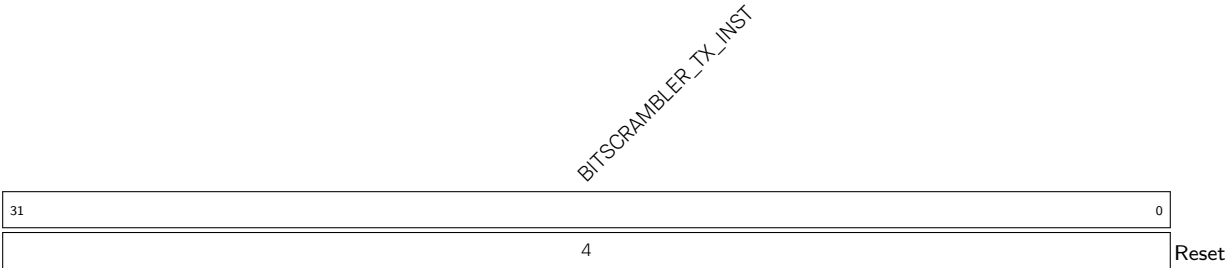
Register 34.1. BITSCRAMBLER_TX_INST_CFG0_REG (0x0000)



BITSCRAMBLER_TX_INST_IDX 配置比特调节器指令索引（通过 BITSCRAMBLER_TX_INST_CFG1_REG 访问指令）。(R/W)

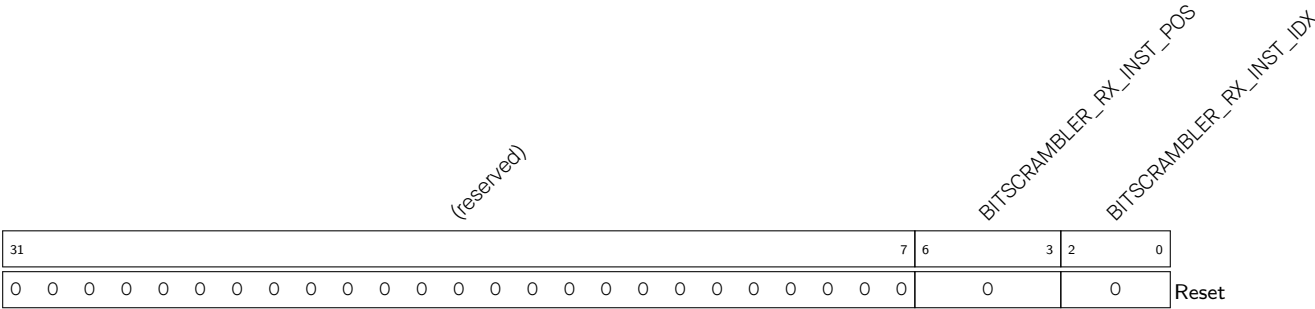
BITSCRAMBLER_TX_INST_POS 配置比特调节器 257 位指令的偏移量（以 32 位增量，通过 BITSCRAMBLER_TX_INST_CFG1_REG 访问指令）。(R/W)

Register 34.2. BITSCRAMBLER_TX_INST_CFG1_REG (0x0004)



BITSCRAMBLER_TX_INST 配置在 BITSCRAMBLER_TX_INST_CFG0_REG 指定的地址处访问的指令。(R/W)

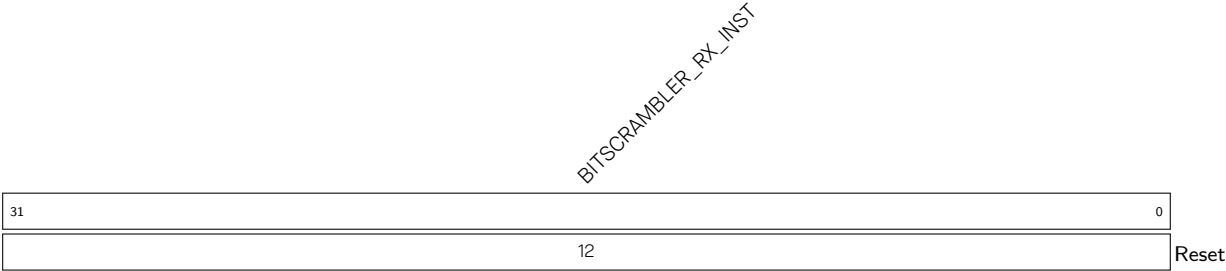
Register 34.3. BITSCRAMBLER_RX_INST_CFG0_REG (0x0008)



BITSCRAMBLER_RX_INST_IDX 配置比特调节器指令索引（通过 [BITSCRAMBLER_RX_INST_CFG1_REG](#) 访问指令）。(R/W)

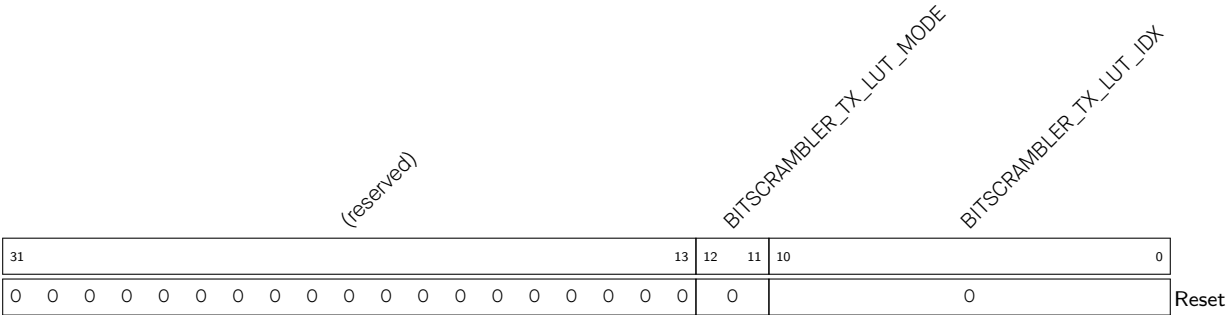
BITSCRAMBLER_RX_INST_POS 配置比特调节器 257 位指令的偏移量（以 32 位为增量，通过 [BITSCRAMBLER_RX_INST_CFG1_REG](#) 访问指令）。(R/W)

Register 34.4. BITSCRAMBLER_RX_INST_CFG1_REG (0x000C)



BITSCRAMBLER_RX_INST 配置在 [BITSCRAMBLER_RX_INST_CFG0_REG](#) 指定的地址处访问的指令。(R/W)

Register 34.5. BITSCRAMBLER_TX_LUT_CFG0_REG (0x0010)



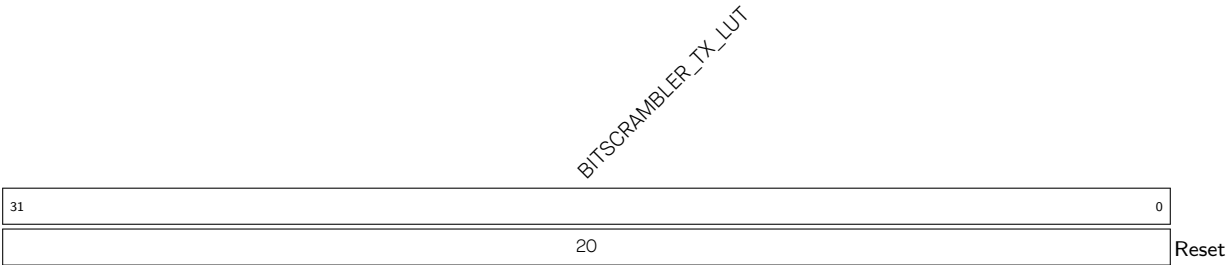
BITSCRAMBLER_TX_LUT_IDX 配置访问 LUT RAM 的位置。单位: [BITSCRAMBLER_TX_LUT_MODE](#) 配置的字大小。(R/W)

BITSCRAMBLER_TX_LUT_MODE 为比特调节器程序配置 LUT RAM 的字大小。

- 0: 1 字节
- 1: 2 字节
- 2: 4 字节
- 3: 保留

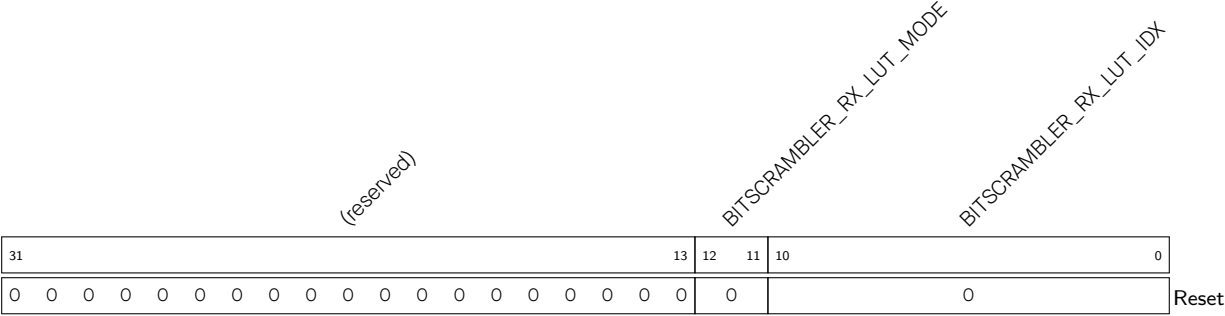
(R/W)

Register 34.6. BITSCRAMBLER_TX_LUT_CFG1_REG (0x0014)



BITSCRAMBLER_TX_LUT 配置在 [BITSCRAMBLER_TX_LUT_CFG0_REG](#) 指定的位置访问的 LUT 表项。(R/W)

Register 34.7. BITSCRAMBLER_RX_LUT_CFG0_REG (0x0018)

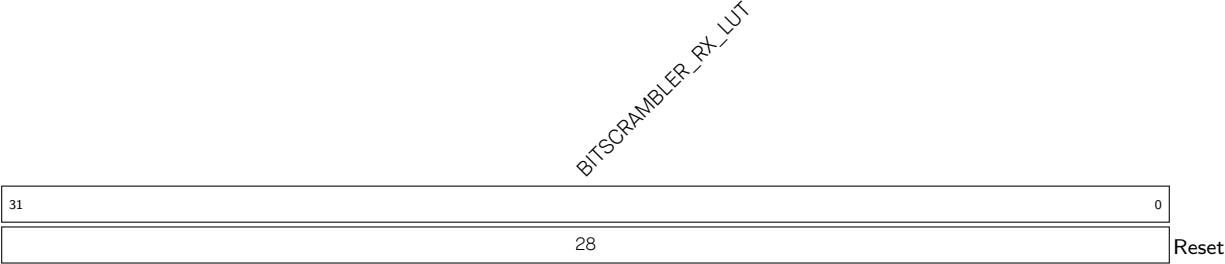


BITSCRAMBLER_RX_LUT_IDX 配置访问 LUT RAM 的位置。单位: [BITSCRAMBLER_RX_LUT_MODE](#) 配置的字大小。(R/W)

BITSCRAMBLER_RX_LUT_MODE 为比特调节器程序配置 LUT RAM 的字大小。

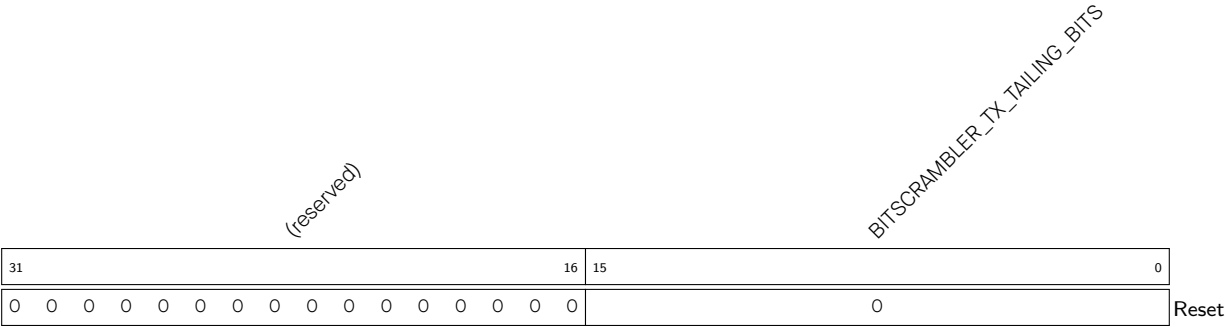
- 0: 1 字节
- 1: 2 字节
- 2: 4 字节
- 3: 保留

Register 34.8. BITSCRAMBLER_RX_LUT_CFG1_REG (0x001C)



BITSCRAMBLER_RX_LUT 配置在 [BITSCRAMBLER_RX_LUT_CFG0_REG](#) 指定的位置访问的 LUT 表项。(R/W)

Register 34.9. BITSCRAMBLER_TX_TAILING_BITS_REG (0x0020)



BITSCRAMBLER_TX_TAILING_BITS 配置比特调节器 TX 路径收到 EOF 后仍可处理额外数据的长度。单位: 比特。(R/W)

Register 34.10. BITSCRAMBLER_RX_TAILING_BITS_REG (0x0024)

(reserved)																BITSCRAMBLER_RX_TAILING_BITS																
31															16	15															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
																																Reset

BITSCRAMBLER_RX_TAILING_BITS 配置比特调节器 RX 路径收到 EOF 后仍可处理额外数据的长度。
单位： 比特。(R/W)

Register 34.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)

(reserved)																								BITSCRAMBLER_TX_FIFO_RST BITSCRAMBLER_TX_RD_DUMMY BITSCRAMBLER_TX_HALT_MODE BITSCRAMBLER_TX_FETCH_MODE BITSCRAMBLER_TX_COND_MODE BITSCRAMBLER_TX_EOF_MODE BITSCRAMBLER_TX_PAUSE BITSCRAMBLER_TX_ENA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
31																								9		8	7	6	5	4	3	2	1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- BITSCRAMBLER_TX_ENA** 配置是否在 TX 路径上启用比特调节器。
0: 禁用
1: 启用
注意，比特调节器核心只能在 RX 或 TX 路径中的一个路径中启用，不能同时启用。(R/W)
- BITSCRAMBLER_TX_PAUSE** 配置是否暂停比特调节器 TX 路径。暂停的路径可取消暂停并恢复执行。
0: 不暂停
1: 暂停
(R/W)
- BITSCRAMBLER_TX_HALT** 配置是否停止比特调节器 TX 路径。停止的 TX 路径将刷新 FIFO 并且无法恢复，需要重置 FIFO 并重新启动。
0: 不停止
1: 停止
(R/W)
- BITSCRAMBLER_TX_EOF_MODE** 配置比特调节器 TX 路径 EOF 信号生成模式，与 [BITSCRAMBLER_TX_TAILING_BITS](#) 结合使用。
0: 对写入输出 FIFO 的数据进行计数，生成延迟的 EOF
1: 对从输入 FIFO 读取的数据进行计数，生成延迟的 EOF
(R/W)
- BITSCRAMBLER_TX_COND_MODE** 配置比特调节器 TX LOOP 指令条件模式。
0: 使用小于运算符获取条件
1: 使用不等于运算符获取条件
(R/W)
- BITSCRAMBLER_TX_FETCH_MODE** 配置比特调节器 TX 路径获取指令模式。
0: 通过复位预取
1: 通过指令获取
(R/W)
- BITSCRAMBLER_TX_HALT_MODE** 当 [BITSCRAMBLER_TX_HALT](#) 置 1 时，配置比特调节器 TX 路径暂停模式。
0: 等待写回数据完成
1: 忽略回写数据
(R/W)

Register 34.11. BITSCRAMBLER_TX_CTRL_REG (0x0028)

[接上页……](#)

BITSCRAMBLER_TX_RD_DUMMY 当接收到 EOF 时，配置比特调节器 TX 路径读取数据模式。

0: 等待读取数据

1: 忽略读取数据

(R/W)

BITSCRAMBLER_TX_FIFO_RST 配置是否重置比特调节器 TX FIFO。

0: 不复位

1: 复位

(WT)

Register 34.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)

(reserved)																BITSCRAMBLER_RX_FIFO_RST			
																BITSCRAMBLER_RX_RD_DUMMY			
																BITSCRAMBLER_RX_HALT_MODE			
																BITSCRAMBLER_RX_FETCH_MODE			
																BITSCRAMBLER_RX_COND_MODE			
																BITSCRAMBLER_RX_EOF_MODE			
																BITSCRAMBLER_RX_HALT_PAUSE			
																BITSCRAMBLER_RX_ENA			
31																9	8	7	6
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
																5	4	3	2
																0	0	0	1
																0	0	0	0
																Reset			

- BITSCRAMBLER_RX_ENA** 配置是否在 RX 路径上启用比特调节器。
0: 禁用
1: 启用
注意，比特调节器核心只能在 RX 或 TX 路径中的一个路径中启用，不能同时启用。(R/W)
- BITSCRAMBLER_RX_PAUSE** 配置是否暂停比特调节器 RX 路径。暂停的路径可取消暂停并恢复执行。
0: 不暂停
1: 暂停
(R/W)
- BITSCRAMBLER_RX_HALT** 配置是否停止比特调节器 RX 路径。停止的 RX 路径将刷新 FIFO 并且无法恢复，需要重置 FIFO 并重新启动。
0: 不停止
1: 停止
(R/W)
- BITSCRAMBLER_RX_EOF_MODE** 配置比特调节器 RX 路径 EOF 信号生成模式，与 [BITSCRAMBLER_RX_TAILING_BITS](#) 结合使用。
0: 对写入输出 FIFO 的数据进行计数，生成延迟的 EOF
1: 对从输入 FIFO 读取的数据进行计数，生成延迟的 EOF
(R/W)
- BITSCRAMBLER_RX_COND_MODE** 配置比特调节器 RX LOOP 指令条件模式。
0: 使用小于运算符获取条件
1: 使用不等于运算符获取条件
(R/W)
- BITSCRAMBLER_RX_FETCH_MODE** 配置比特调节器 RX 路径获取指令模式。
0: 通过复位预取
1: 通过指令获取
(R/W)
- BITSCRAMBLER_RX_HALT_MODE** 当 [BITSCRAMBLER_RX_HALT](#) 置 1 时，配置比特调节器 RX 路径暂停模式。
0: 等待写回数据完成
1: 忽略回写数据
(R/W)

Register 34.12. BITSCRAMBLER_RX_CTRL_REG (0x002C)

接上页……

BITSCRAMBLER_RX_RD_DUMMY 当接收到 EOF 时，配置比特调节器 RX 路径读取数据模式。

- 0：等待读取数据
- 1：忽略读取数据

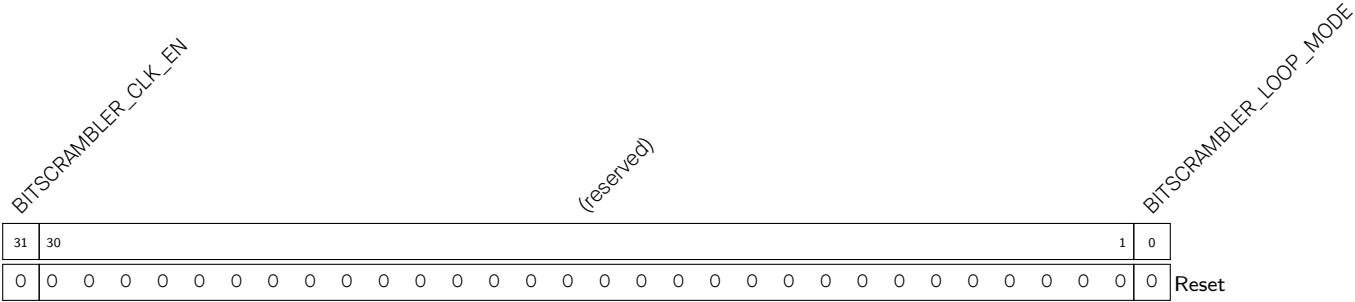
(R/W)

BITSCRAMBLER_RX_FIFO_RST 配置是否重置比特调节器 RX FIFO。

- 0：不复位
- 1：复位

(WT)

Register 34.13. BITSCRAMBLER_SYS_REG (0x00F8)



BITSCRAMBLER_LOOP_MODE 配置是否启用比特调节器 TX 到 DMA RX 环回模式

- 0：禁用
- 1：启用

(R/W)

BITSCRAMBLER_CLK_EN 配置是否常开配置寄存器时钟。

- 0：访问时打开
- 1：常开

(R/W)

Register 34.14. BITSCRAMBLER_TX_STATE_REG (0x0030)

BITSCRAMBLER_TX_EOF_TRACE_CLR			BITSCRAMBLER_TX_EOF_GET_CNT												(reserved)					BITSCRAMBLER_TX_FIFO_EMPTY				BITSCRAMBLER_TX_IN_PAUSE				BITSCRAMBLER_TX_IN_WAIT				BITSCRAMBLER_TX_IN_RUN				BITSCRAMBLER_TX_IN_IDLE						
31	30	29													16	15						5	4	3	2	1	0															
0	0	0												0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0												1	0	0	0	1	Reset											

BITSCRAMBLER_TX_IN_IDLE 表示比特调节器 TX 路径是否停止。

0: 未停止

1: 停止

(RO)

BITSCRAMBLER_TX_IN_RUN 表示比特调节器 TX 路径是否正在运行。

0: 未运行

1: 运行

(RO)

BITSCRAMBLER_TX_IN_WAIT 表示比特调节器 TX 路径是否正在等待写回完成。

0: 不等待

1: 等待

(RO)

BITSCRAMBLER_TX_IN_PAUSE 表示比特调节器 TX 路径是否暂停。

0: 未暂停

1: 暂停

(RO)

BITSCRAMBLER_TX_FIFO_EMPTY 表示比特调节器 TX FIFO 是否为空。

0: 不为空

1: 空

(RO)

BITSCRAMBLER_TX_EOF_GET_CNT 表示收到 EOF 后比特调节器 TX 路径的字节计数。(RO)

BITSCRAMBLER_TX_EOF_OVERLOAD 表示比特调节器 TX 路径是否尝试处理多个 EOF。

0: 不尝试处理多个 EOF

1: 尝试处理多个 EOF

(RO)

BITSCRAMBLER_TX_EOF_TRACE_CLR 配置是否清除 [BITSCRAMBLER_TX_EOF_OVERLOAD](#) 和 [BITSCRAMBLER_TX_EOF_GET_CNT](#)。

0: 不清楚

1: 清除

(WT)

Register 34.15. BITSCRAMBLER_RX_STATE_REG (0x0034)

31		30		29		16										15		5					4		3		2		1		0	
0		0				0										0		0 0 0 0 0 0 0 0 0 0					0		0		0		0		1	

BITSCRAMBLER_RX_EOF_TRACE_CLR

BITSCRAMBLER_RX_EOF_OVERLOAD

BITSCRAMBLER_RX_EOF_GET_CNT

(reserved)

BITSCRAMBLER_RX_FIFO_FULL

BITSCRAMBLER_RX_IN_PAUSE

BITSCRAMBLER_RX_IN_WAIT

BITSCRAMBLER_RX_IN_RUN

BITSCRAMBLER_RX_IN_IDLE

Reset

BITSCRAMBLER_RX_IN_IDLE 表示比特调节器 RX 路径是否停止。

- 0: 未停止
 - 1: 停止
- (RO)

BITSCRAMBLER_RX_IN_RUN 表示比特调节器 RX 路径是否正在运行。

- 0: 未运行
 - 1: 运行
- (RO)

BITSCRAMBLER_RX_IN_WAIT 表示比特调节器 RX 路径是否正在等待写回完成。

- 0: 不等待
 - 1: 等待
- (RO)

BITSCRAMBLER_RX_IN_PAUSE 表示比特调节器 RX 路径是否暂停。

- 0: 未暂停
 - 1: 暂停
- (RO)

BITSCRAMBLER_RX_FIFO_FULL 表示比特调节器 RX FIFO 是否已满。

- 0: 未满
 - 1: 已满
- (RO)

BITSCRAMBLER_RX_EOF_GET_CNT 表示收到 EOF 后比特调节器 RX 路径的字节计数。(RO)

BITSCRAMBLER_RX_EOF_OVERLOAD 表示比特调节器 RX 路径是否尝试处理多个 EOF。

- 0: 不尝试处理多个 EOF
 - 1: 尝试处理多个 EOF
- (RO)

BITSCRAMBLER_RX_EOF_TRACE_CLR 配置是否清除 [BITSCRAMBLER_RX_EOF_OVERLOAD](#) 和 [BITSCRAMBLER_RX_EOF_GET_CNT](#)。

- 0: 不清楚
 - 1: 清除
- (WT)

Register 34.16. BITSCRAMBLER_VERSION_REG (0x00FC)



BITSCRAMBLER_BITSCRAMBLER_VER 版本控制寄存器。(R/W)

第 VI 卷

模拟信号处理

该部分描述了与模数转换、电压比较、片上传感器以及温度感应等功能相关的模块，展示了系统在处理模拟信号方面的性能。

第 35 章

温度传感器

35.1 介绍

ESP32-C5 搭载了一个温度传感器，用于实时测量芯片内部温度。温度传感器可将输出的电压转换成数字值，并且带有补偿温度偏移的功能。

35.2 特性

温度传感器的主要特性包括：

- 支持软件触发测量温度，且一旦触发后，传感器可持续测量温度，软件可实时读取数据
- 支持硬件触发自动监测温度
- 支持两种自动监测模式且发送中断
- 支持根据使用环境配置温度偏移，提高测试精度
- 温度测量范围可配置
- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

35.3 结构概览

温度传感器的内部结构如图 35.3-1 所示。

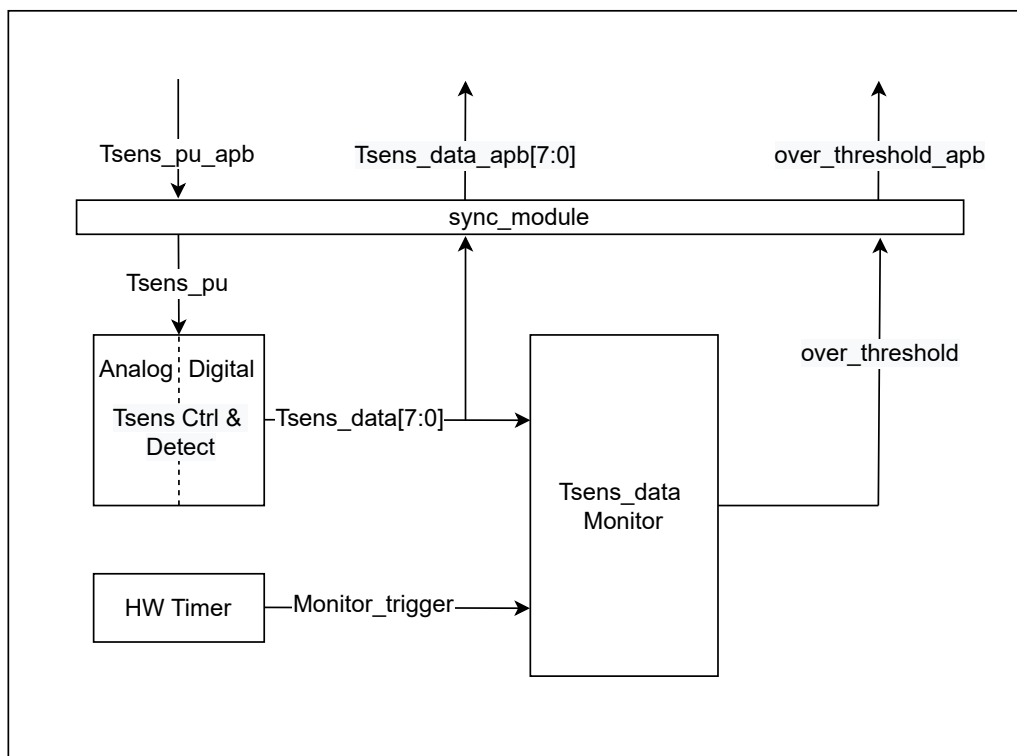


图 35.3-1. 温度传感器结构框图

根据图 35.3-1，温度传感器模块主要包括以下功能模块：

- Tsens Ctrl & Detect：温度传感器
- HW Timer（硬件定时器）：用于触发温度监测
- Tsens_data 监测器：用于监测温度是否超出阈值
- sync_module：APB 时钟域与温度传感器时钟域间的同步模块

35.4 功能描述

35.4.1 温度传感器上电

通过设置 `APB_SARADC_TSENS_PU` 给温度传感器上电。

35.4.2 时钟管理

温度传感器有两个时钟源：`RC_FAST_CLK` 和 `XTAL_CLK`。通过寄存器 `PCR_TSENS_CLK_SEL` 选择时钟源。时钟分频由寄存器 `APB_SARADC_TSENS_CLK_DIV` 配置。

35.4.3 自动监测模式

温度传感器自动监测模式有两种：绝对值模式和变化量模式，通过寄存器 `APB_SARADC_WAKEUP_MODE` 进行选择。

- 绝对值模式：

- 用于监测当前温度的绝对值。配置 `APB_SARADC_WAKEUP_TH_LOW` 和 `APB_SARADC_WAKEUP_TH_HIGH` 来决定温度阈值，温度超过高阈值或小于低阈值时将触发中断。
- 变化量模式：
 - 用于监测温度的变化量。若连续两次采样的温度值增量超过 `APB_SARADC_WAKEUP_TH_HIGH` 的配置值或减少量超过 `APB_SARADC_WAKEUP_TH_LOW` 的配置值，将触发中断。例如当 `APB_SARADC_WAKEUP_TH_LOW` 的值为 8 时，若连续两次温度采样分别为 28 和 19，即温度减少量为 9，将触发中断。

35.4.4 温度测量范围与温度偏移

为了提高测试精度，温度传感器的测量范围被设计为 5 个范围，如表 35.4-1 所示。每个测量范围都有一定的测温误差。用户无需关注测温误差，只需选择相应的温度偏移，即可获得校准后的数据。

表 35.4-1. 温度测量范围与温度偏移

测量范围 (°C)	温度偏移
50 ~ 125	-2
20 ~ 100	-1
-10 ~ 80	0
-30 ~ 50	1
-40 ~ 20	2

35.4.5 数据转换

温度传感器的输出值存储在寄存器 `APB_SARADC_TSENS_OUT` 中。用户可使用以下公式将输出值 `VALUE` 转换成实际的温度值 T (°C)。转换公式如下：

$$T = 0.4386 \times VALUE - 27.88 \times offset - 20.52$$

其中，`offset` 即表 35.4-1 所示温度偏移。

35.5 配置流程

温度传感器可由软件启动，具体配置如下：

1. 置位 `APB_SARADC_TSENS_PU` 使温度传感器上电；
2. 置位 `PCR_TSENS_CLK_EN` 使能温度传感器时钟；
3. 配置 `APB_SARADC_TSENS_CLK_SEL` 选择传感器时钟；
4. 等待 `APB_SARADC_TSENS_XPD_WAIT` 个时钟周期后，温度传感器的复位释放，开始测量环境温度；
5. 等待 100 μ s 后（输出值会随着测量时间的增加而逐渐逼近真实的温度值），直接从 `APB_SARADC_TSENS_OUT` 中读取温度值。

如要使用硬件自动温度监测功能，则可在给温度传感器上电之前增加如下步骤：

1. 配置 `APB_SARADC_TSENS_SAMPLE_RATE` 设置采样率；

2. 配置 `APB_SARADC_WAKEUP_MODE` 选择温度监测模式；
3. 配置 `APB_SARADC_WAKEUP_TH_HIGH/LOW` 设置温度高低阈值；
4. 置位 `APB_SARADC_WAKEUP_EN` 启用温度监测；
5. 置位 `APB_SARADC_TSENS_SAMPLE_EN` 自动触发温度监测。

自动温度监测工作模式下，温度传感器的输出值不会存储，但用户可以随时读取寄存器 `APB_SARADC_TSENS_OUT` 获取当前输出值。

35.6 中断

ESP32-C5 的温度传感器可以生成以下中断信号并发送给 [中断矩阵](#)。

- `LP_TSENS_INTR`

这个中断信号由以下内部中断源生成：

- `APB_SARADC_TSENS_INT`：温度采样值超过阈值，即触发此中断。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 [3 中断矩阵 \[to be added later\]](#) > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [35.8 寄存器列表](#)。

35.7 事件任务矩阵功能

在 ESP32-C5 中，温度传感器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发温度传感器的 ETM 任务，或者通过温度传感器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 [8 事件任务矩阵 \(ETM\)](#)。这里仅介绍与温度传感器相关的 ETM 任务和 ETM 事件。

温度传感器可接收的 ETM 任务有：

- `TMPSNSR_TASK_START_SAMPLE`：该任务被触发时，温度传感器开始采样。
- `TMPSNSR_TASK_STOP_SAMPLE`：该任务被触发时，温度传感器停止采样。

温度传感器可产生的 ETM 事件有：

- `TMPSNSR_EVT_OVER_LIMIT`：温度传感器发现温度超过阈值时生成。

在具体应用中，温度传感器的 ETM 事件可以用来触发温度传感器的 ETM 任务，例如，`TMPSNSR_EVT_OVER_LIMIT` 事件可以触发 `TMPSNSR_TASK_STOP_SAMPLE` 任务。

35.8 寄存器列表

本小节的所有地址均为相对于温度传感器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
APB_SARADC_INT_ENA_REG	SAR ADC / 温度传感器中断使能寄存器	0x0040	R/W
APB_SARADC_INT_RAW_REG	SAR ADC / 温度传感器原始中断寄存器	0x0044	R/ WTC/ SS
APB_SARADC_INT_ST_REG	SAR ADC / 温度传感器中断状态寄存器	0x0048	RO
APB_SARADC_INT_CLR_REG	SAR ADC / 温度传感器中断清除寄存器	0x004C	WT
APB_SARADC_APB_TSENS_CTRL_REG	温度传感器控制寄存器 1	0x0058	varies
APB_SARADC_APB_TSENS_CTRL2_REG	温度传感器控制寄存器 2	0x005C	R/W
APB_TSENS_WAKE_REG	温度传感器自动监测功能配置寄存器	0x0064	varies
APB_TSENS_SAMPLE_REG	温度传感器采样配置寄存器	0x0068	R/W
版本控制寄存器			
APB_SARADC_APB_CTRL_DATE_REG	版本控制寄存器	0x03FC	R/W

35.9 寄存器

本小节的所有地址均为相对于温度传感器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 35.1. APB_SARADC_APB_TSENS_CTRL_REG (0x0058)

(reserved)										APB_SARADC_TSENS_PU										APB_SARADC_TSENS_CLK_DIV										APB_SARADC_TSENS_IN_INV										(reserved)										APB_SARADC_TSENS_OUT																																																																					
31										23										22										21										14										13										12										8										7										0																													
0										0										0										0										0										0										0										0										0										0										0x80										Reset									

APB_SARADC_TSENS_OUT 表示温度传感器的输出值。(RO)

APB_SARADC_TSENS_IN_INV 配置是否反转温度传感器的输入值。

- 0: 无效
- 1: 反转
- (R/W)

APB_SARADC_TSENS_CLK_DIV 表示温度传感器的时钟分频系数。(R/W)

APB_SARADC_TSENS_PU 配置是否使能温度传感器上电。

- 0: 无效
- 1: 使能
- (R/W)

Register 35.2. APB_SARADC_APB_TSENS_CTRL2_REG (0x005C)

(reserved)																APB_SARADC_TSENS_CLK_SEL				APB_SARADC_TSENS_CLK_INV				APB_SARADC_TSENS_XPD_FORCE				APB_SARADC_TSENS_XPD_WAIT									
31																16	15	14	13	12	11	0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	0x0	0x0	0x2																Reset		

- APB_SARADC_TSENS_CLK_SEL 配置温度传感器时钟。
0: RC_FAST_CLK
1: XTAL_CLK
(R/W)
- APB_SARADC_TSENS_CLK_INV 配置传感器采样时钟的相位。(R/W)
- APB_SARADC_TSENS_XPD_FORCE 配置是否使能强制上电/断电温度传感器。
0: 禁用强制上电功能
1: 禁用强制断电功能
2: 强制上电温度传感器
3: 强制断电温度传感器
(R/W)
- APB_SARADC_TSENS_XPD_WAIT 配置温度传感器的复位释放前需要等待的时钟周期。(R/W)

Register 35.3. APB_TSENS_WAKE_REG (0x0064)

(reserved)																APB_SARADC_WAKEUP_EN				APB_SARADC_WAKEUP_MODE				APB_SARADC_WAKEUP_OVER_UPPER_TH				APB_SARADC_WAKEUP_TH_HIGH				APB_SARADC_WAKEUP_TH_LOW			
31																19				18	17	16	15				8				7	0			
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	0	0	0xff				0x0				Reset								

APB_SARADC_WAKEUP_TH_LOW 配置温度传感器自动监测功能的低阈值。(R/W)

APB_SARADC_WAKEUP_TH_HIGH 配置温度传感器自动监测功能的高阈值。(R/W)

APB_SARADC_WAKEUP_OVER_UPPER_TH 表示输出值是否超过阈值。

- 0：输出值低于低阈值
- 1：输出值高于高阈值
- (RO)

APB_SARADC_WAKEUP_MODE 配置温度监测模式。仅当 APB_SARADC_WAKEUP_EN=1 时有效。

- 0：绝对值模式
- 1：变化量模式
- (R/W)

APB_SARADC_WAKEUP_EN 配置是否启用温度监测功能。

- 0：禁用
- 1：启用
- (R/W)

Register 35.4. APB_TSENS_SAMPLE_REG (0x0068)

(reserved)																APB_SARADC_TSENS_SAMPLE_EN																APB_SARADC_TSENS_SAMPLE_RATE																															
31																17																15																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0																20																Reset															

APB_SARADC_TSENS_SAMPLE_RATE 配置硬件自动温度监测的采样率。采样周期 = 配置值 × 传感器工作时钟周期。(R/W)

APB_SARADC_TSENS_SAMPLE_EN 配置是否自动触发持续温度监测。

0: 不使能

1: 使能

(R/W)

Register 35.5. APB_SARADC_CTRL_DATA_REG (0x03FC)

Diagram of the APB_SARADC_DATE register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. The value 0x2206240 is shown in the center, and the word "Reset" is at the bottom right.

APB_SARADC_DATE SAR ADC（请参考[章节 36 ADC 控制器](#)获取详细信息）和温度传感器的版本控制寄存器。(R/W)

第 36 章

ADC 控制器

36.1 概述

ESP32-C5 搭载了一个 12 位逐次逼近型模拟数字转换器 (SAR ADC)，用于测量最多来自 6 个管脚上的模拟信号。

SAR ADC 由 DIG ADC 控制器控制，该控制器驱动电压采样，支持单次采样和多通道采样。

36.2 术语表

下列有关 SAR ADC 的术语是在《ESP32-C5 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

SAR ADC： 包含模拟 ADC 电路与数字 ADC 控制器。

单次采样模式： 在此模式下，ADC 一次对一个通道进行采样。

多通道采样模式： 在此模式下，ADC 依次对一组通道进行采样，或者连续对单个通道进行采样。

转换结果： 转换结果是模拟到数字转换后得到的二进制数字值。有时也被称为转换数据。

滤波数据： 滤波数据是转换数据经过滤波器处理得到的结果。

采样： 该术语通常指的是对模拟输入进行采样、转换采样数据，并将转换结果传输到内存的整个过程。在特定语境和流程中，“采样”也可能指 SAR ADC 从模拟信号中采集数据。

36.3 特性

SAR ADC 具有以下特性：

- 12 位分辨率
- 支持采集最多 6 个管脚上的模拟信号
- 支持单次采样模式和多通道采样模式
- 在多通道采样模式下，支持：
 - 自定义采样通道顺序
 - 两个滤波器，滤波系数可配
 - 阈值监控，滤波后数据大于设置的高阈值或小于设置的低阈值将产生中断
 - GDMA 连续数据搬运

- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

36.4 结构概览

SAR ADC 的主要子模块与连接情况见图 36.4-1。

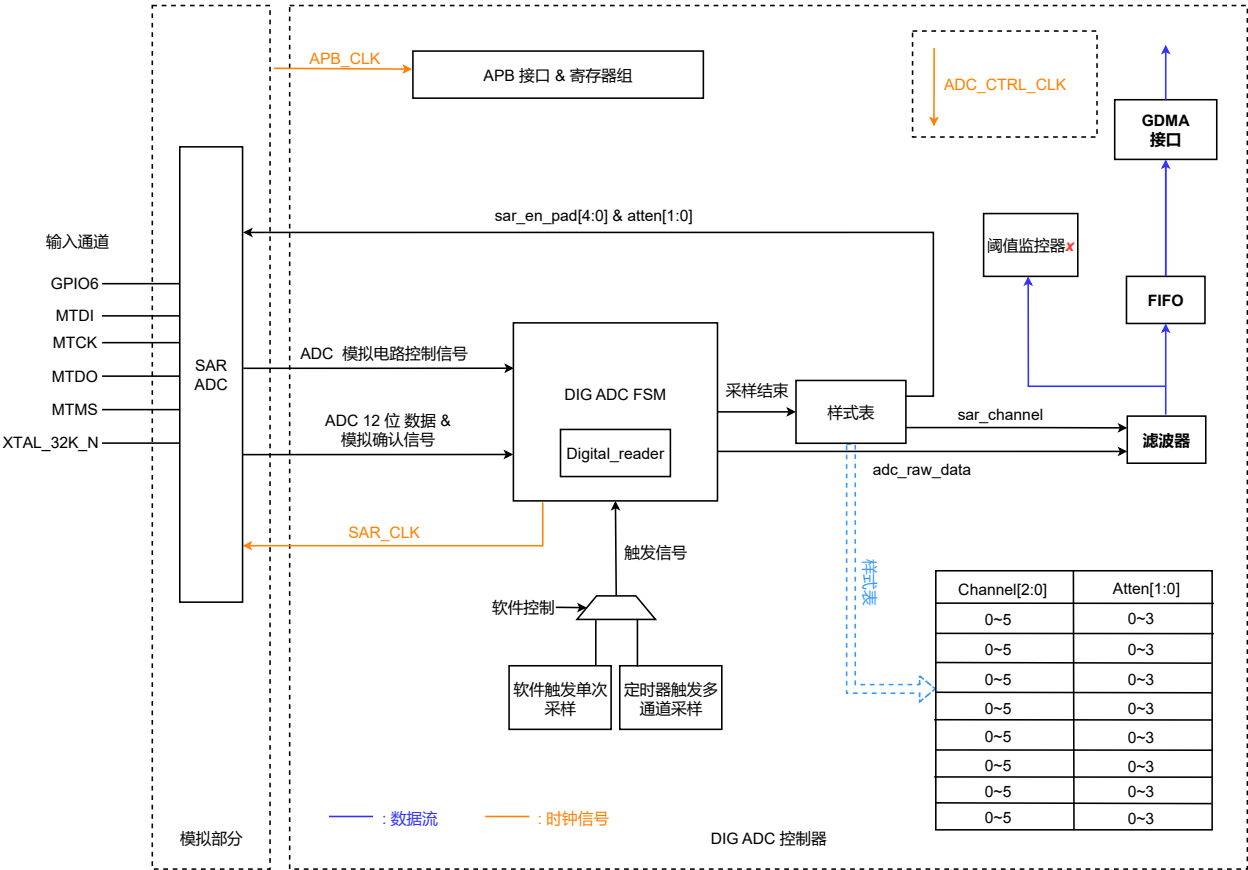


图 36.4-1. SAR ADC 结构图

根据图 36.4-1，按照从左至右、从上到下的顺序，SAR ADC 模块主要包括以下功能模块：

- 6 个通道，连接 6 个芯片管脚
- SAR ADC：SAR ADC 的模拟部分
- DIG ADC 控制器：SAR ADC 的数字部分，主要包括：
 - 时钟管理模块：对时钟源进行选择和分频
 - DIG ADC FSM（有限状态机）：生成整个 ADC 采样过程中所需的各种信号
 - Digital_reader：读取 SAR ADC 的数值，由 DIG ADC FSM 驱动
 - 滤波器：用于在多通道采样模式下对目标通道的转换数据进行滤波
 - Threshold Monitor^x：阈值监控器 1 和阈值监控器 2。可在采样值大于设定的高阈值，或小于设定的低阈值时触发中断。

36.5 功能描述

36.5.1 ADC 上电

置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 寄存器给 ADC 上电。上电后即可进行采样，硬件设计上已考虑等待时间，因此用户无需特意等待。

36.5.2 ADC 通道

SAR ADC 具有 6 个通道，连接芯片的 6 个管脚。ADC 需首先通过内部多路器选择待测量的模拟管脚，然后才能采样模拟信号。

表 36.5-1 列出了用作 ADC 通道的芯片管脚和相应的 GPIO 编号。

表 36.5-1. SAR ADC 通道

芯片管脚名称	GPIO 编号	ADC 通道编号
XTAL_32K_N	GPIO1	0
MTMS	GPIO2	1
MTDI	GPIO3	2
MTCK	GPIO4	3
MTDO	GPIO5	4
GPIO6	GPIO6	5

36.5.3 ADC 时钟

图 36.5-1 为 SAR ADC 的时钟结构图。

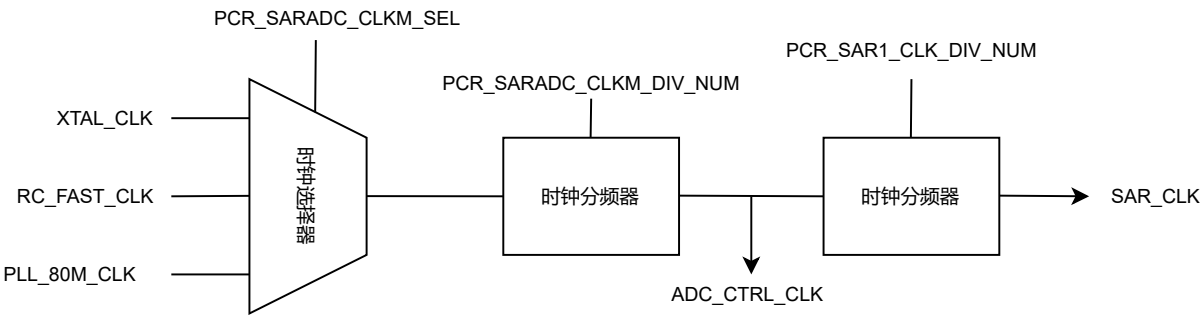


图 36.5-1. SAR ADC 时钟结构图

SAR ADC 的系统时钟为 ADC_CTRL_CLK，它是 DIG ADC FSM 和其他逻辑电路的工作时钟（APB 接口和 Digital_reader 除外）。

ADC_CTRL_CLK 可选择三个时钟源：XTAL_CLK，RC_FAST_CLK，或 PLL_80M_CLK。通过寄存器 [PCR_SARADC_CLKM_SEL](#) 选择时钟源。

SAR ADC 的分频时钟为 SAR_CLK，由 ADC_CTRL_CLK 分频而得。它是 SAR ADC 和 Digital_reader 的工作时钟，频率不得超过 5 MHz。

更多有关时钟的信息请参考章节 6 复位和时钟。

36.5.4 单次采样模式

单次采样模式下，ADC 只在一个通道进行一次采样。该模式由软件通过寄存器 APB_SARADC_ONETIME_SAMPLE 触发。一旦采样完成，转换数据将被存储在 APB_SARADC_DATA 中。如需转换采样通道，可继续配置 APB_SARADC_ONETIME_SAMPLE 寄存器。

36.5.5 多通道采样模式

多通道采样模式由专用定时器触发。在该模式下，ADC 会按照指定的顺序采样一组通道。通道顺序在样式表中进行定义。多通道采样模式也可以配置成连续采样一个通道。

用户通过配置寄存器 APB_SARADC_TIMER_EN 使能定时器，配置 APB_SARADC_TIMER_TARGET 设置定时器的触发周期。当定时器计数到配置周期数的 2 倍时，触发采样。定时器的工作时钟是 ADC_CTRL_CLK。

采样完成后，定时器复位为 0，重新开始计数。数据通过 GDMA 连续搬运至内存空间。

说明：

单次采样和多通道采样不能同时使用。

36.5.6 ADC 转换和衰减

SAR ADC 转换模拟信号时，转换电压范围为 0 mV ~ V_{ref} 。 V_{ref} 为 ADC 内部参考电压，出厂设定为 1100 mV。ADC 的转换结果是个 12 位的数字值，是原始值。用户可使用以下公式将原始值 $data$ 转换成模拟电压 V_{data} ：

$$V_{data} = \frac{V_{ref}}{k} \times \frac{data}{4095}$$

其中 k 为衰减对应的系数。

如需转换大于 V_{ref} 的电压，信号输入 SAR ADC 前可进行衰减。使用寄存器 APB_SARADC_ONETIME_ATTEN 可配置衰减为 0 dB、2.5 dB、6 dB 或 12 dB。

36.5.7 DIG ADC FSM

DIG ADC FSM（下文简称 FSM）在多通道采样模式下工作，用于生成整个 ADC 采样过程中所需的各种信号。图 36.5-2 展示了 FSM 的工作原理。

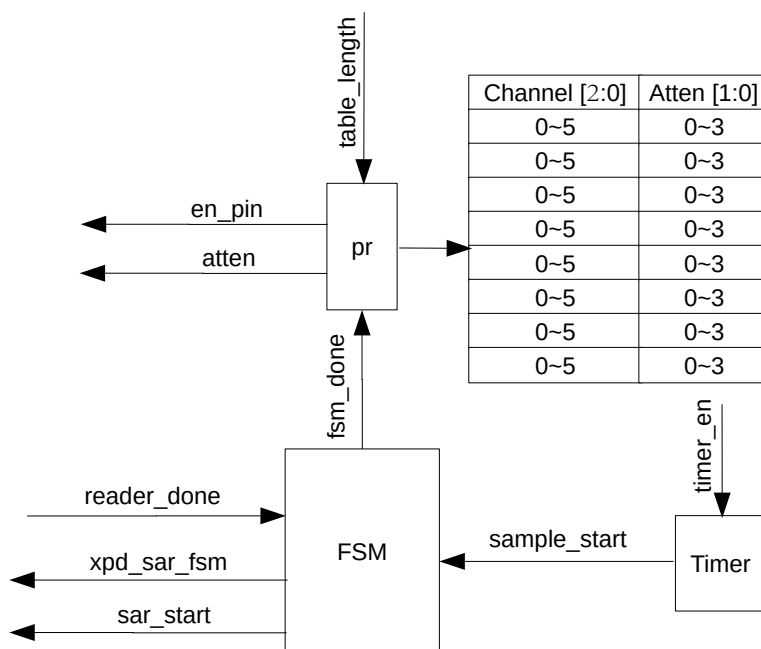


图 36.5-2. DIG ADC FSM 概况

其中，

- Timer（定时器）：表示 DIG ADC 控制器的专用定时器，用于触发多通道采样模式。定时器可产生 sample_start 信号，表示开始采样。
- pr：样式表指针，FSM 将根据该指针指向的样式配置，发送相应信号。

用户通过置位 [APB_SARADC_TIMER_EN](#) 使能定时器。定时器超时将触发 sample_start 信号驱动 FSM 模块开始采样。FSM 模块收到 sample_start 信号后，执行以下操作：

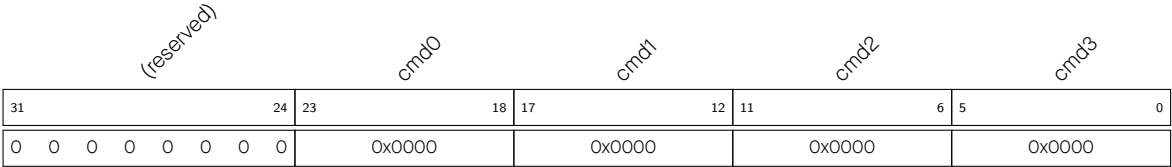
- 开启 SAR ADC 电源；
- 根据当前 pr 指向的样式，决定该次采样的 ADC 通道以及衰减；
- 根据配置信息，输出相应的 en_pin（使能管脚）以及 atten（衰减）信号到模拟端；
- 发起 sar_start 信号，开启采样。

FSM 收到 Digital_reader 返回的 reader_done 信号后，执行以下操作：

- 结束采样；
- 将数据传输给滤波器，然后阈值监控器通过 GDMA 将数据传输给内存（见图 36.4-1）；
- 更新样式表指针 pr，等待下一次采样。pr 在 0 - [APB_SARADC_SAR_PATT_LEN](#) (table_length) 之间循环计数。

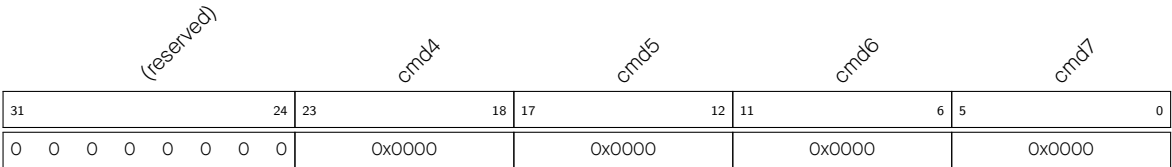
36.5.8 样式表

FSM 包含一个样式表，由 [APB_SARADC_SAR_PATT_TAB1_REG](#) 和 [APB_SARADC_SAR_PATT_TAB2_REG](#) 两个寄存器组成，每个寄存器包含四个样式，每个样式长度为六位，如图 36.5-3 和图 36.5-4 所示。



cmd x 表示样式表中的样式，即样式 0 ~ 样式 3。

图 36.5-3. APB_SARADC_SAR_PATT_TAB1_REG 包含样式 0 - 3



cmd x 表示样式表中的样式，即样式 4 ~ 样式 7。

图 36.5-4. APB_SARADC_SAR_PATT_TAB2_REG 包含样式 4 - 7

每个样式长度为六位，共包含三个字段，每个字段存储的是采样规则。图 36.5-5 所示为样式结构，字段描述参见后文。



图 36.5-5. 样式结构

atten 配置衰减：

- 0: 0 dB
- 1: 2.5 dB
- 2: 6 dB
- 3: 12 dB

ch_sel 配置通道：

- 0: 通道 0
- 1: 通道 1
- 2: 通道 2
- 3: 通道 3
- 4: 通道 4
- 5: 通道 5

(reserved) 保留字段

样式配置示例

注意：使用多通道连续采样时，每个通道的衰减必须设置成一致；例如，希望实现如下所示的多通道采样方式：

- 采样 SAR ADC 的通道 0，且衰减配置为 2.5 dB；

- 采样 SAR ADC 的通道 2，且衰减配置为 2.5 dB。

则具体的配置如下：

- 配置第一个样式 cmd0，如下图所示：

(reserved)		ch_sel		atten	
5	4	2	1	0	
0	0		1		

图 36.5-6. cmd0 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 0，即选择通道 0。

- 配置第二个样式 cmd1，如下图所示：

(reserved)		ch_sel		atten	
5	4	2	1	0	
0	2		1		

图 36.5-7. cmd1 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 2，即选择通道 2。

- 配置 `APB_SARADC_SAR_PATT_LEN` 为 1，即选择使用上述配置好的样式 0 和样式 1；
- 使能定时器，则 DIG ADC 控制器将根据上述样式配置周期性采样两个通道。

36.5.9 ADC 滤波器

DIG ADC 控制器提供两个滤波器，用于在多通道采样模式下对目标通道的转换数据进行滤波。两个滤波器均可配置给 SAR ADC 的任一通道，但不能配置给同一通道。如果配置给同一通道，则第一次配置的滤波器生效。

滤波公式如下所示：

$$data_{cur} = \frac{(k - 1) data_{prev}}{k} + \frac{data_{in}}{k} + 0.5$$

- $data_{cur}$ ：滤波后数据
- $data_{in}$ ：ADC 转换数据
- $data_{prev}$ ：上次滤波数据
- k ：滤波系数

配置滤波器如下：

- 配置 `APB_SARADC_FILTER_CHANNEL $x$` 设置滤波器 x ($x=0, 1$) 作用的 ADC 通道；
- 配置 `APB_SARADC_FILTER_FACTOR $x$` 设置滤波器 x 的滤波系数 k 。

36.5.10 阈值监控器

DIG ADC 控制器包含两个阈值监控器，用于在多通道采样模式下对滤波之后的数据进行监控。当数据大于设定的高阈值，则触发高阈值中断；当数据小于设定的低阈值，则触发低阈值中断。两个监控器可配置给 SAR ADC 的任一通道，但不能配置给同一通道。

阈值监控配置如下：

- 置位 `APB_SARADC_THRESx_EN` 使能阈值监控器 x ($x=0, 1$) ；
- 配置 `APB_SARADC_THRESx_LOW` 设置低阈值；
- 配置 `APB_SARADC_THRESx_HIGH` 设置高阈值；
- 配置 `APB_SARADC_THRESx_CHANNEL` 设置监控的通道；
- 配置 `APB_SARADC_THRES_ALL_EN` 使能阈值监控。

36.5.11 GDMA 支持

由于 ADC 仅有一个数据寄存器 `APB_SARADC_DATA` 用于存储单次采样的转换数据，因此当转换多个通道时需要使用 GDMA，通过 GDMA 接口将转换数据连续存储至内存。

由 DIG ADC 控制器的专用定时器产生 GDMA 功能触发信号。用户可通过软件配置 `APB_SARADC_APB_ADC_TRANS` 将 GDMA 的数据通路切换到 DIG ADC 控制器。关于 GDMA 的具体配置，请参考章节 3 通用 DMA 控制器 (GDMA)。

GDMA 数据格式

ADC 最终向 GDMA 传递 32 位数据，如下图所示：

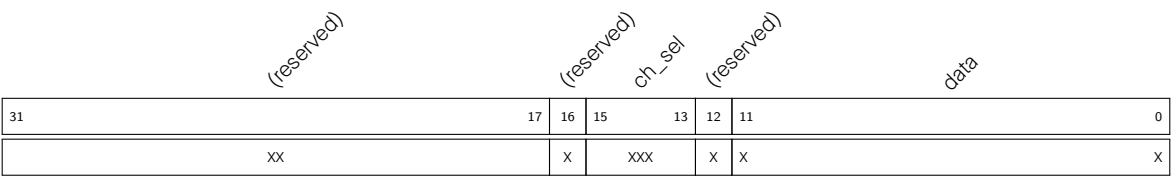


图 36.5-8. GDMA 数据格式

data ADC 转换结果，12 位

ch_sel 通道信息，3 位

36.6 事件任务矩阵功能

在 ESP32-C5 中，SAR ADC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 SAR ADC 的 ETM 任务，或者通过 SAR ADC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与 SAR ADC 相关的 ETM 任务和 ETM 事件。

SAR ADC 可接收的 ETM 任务有：

- `ADC_TASK_SAMPLE0`：该任务被触发时，ADC 开始进行单次采样。
- `ADC_TASK_START0`：该任务被触发时，ADC 开始进行多通道采样。
- `ADC_TASK_STOP0`：该任务被触发时，ADC 停止采样。

SAR ADC 可产生的 ETM 事件有：

- ADC_EVT_CONV_CMPLTO：在单次采样模式和多通道采样模式下，ADC 每完成一次采样时生成。
- ADC_EVT_EQ_ABOVE_THRESH x ：ADC 过滤后数据高于阈值时生成。 $x=0, 1$ ，代表阈值监控器 0, 1。
- ADC_EVT_EQ_BELOW_THRESH x ：ADC 过滤后数据低于阈值时生成。 $x=0, 1$ ，代表阈值监控器 0, 1。
- ADC_EVT_STARTEDO：ADC 开始采样时生成，单次采样不会触发该事件。
- ADC_EVT_STOPPEDO：ADC 停止采样时生成，单次采样不会触发该事件。

在具体应用中，SAR ADC 的 ETM 事件可以用来触发 SAR ADC 的 ETM 任务，例如，ADC_EVT_EQ_ABOVE_THRESH x 事件可以触发 ADC_TASK_STOPO 任务。

36.7 中断

ESP32-C5 的 SAR ADC 可以生成 APB_ADC_INTR 中断信号并发送给[中断矩阵](#)。这一中断信号由以下 ADC 的内部中断源生成。

- APB_SARADC_ADC_DONE_INT：SAR ADC 完成一次转换，即触发此中断。
- APB_SARADC_THRES x _HIGH_INT：过滤值超过阈值监控器 x 的高阈值，即触发此中断。
- APB_SARADC_THRES x _LOW_INT：过滤值低于阈值监控器 x 的低阈值，即触发此中断。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 [中断矩阵 \[to be added later\]](#) > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 [中断配置寄存器](#)。具体寄存器名称请查看 [36.9 寄存器列表](#)。

36.8 配置流程

36.8.1 单次采样配置

软件驱动的单次采样的具体配置如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 给 ADC 上电；
2. 配置 PCR_SARADC_CLKM_SEL 选择时钟源；
3. 配置时钟分频寄存器 PCR_SARADC_CLKM_DIV_NUM 和 PCR_SAR1_CLK_DIV_NUM；
4. 置位 PCR_SARADC_CLKM_EN 使能 SAR ADC 时钟；
5. 置位 APB_SARADC_ONETIME_SAMPLE 使能单次采样；
6. 配置 APB_SARADC_ONETIME_CHANNEL 选择采样通道；
7. 根据需要，配置 APB_SARADC_ONETIME_ATTEN 选择衰减；
8. 置位 APB_SARADC_ONETIME_START 启动单次采样。

采样结束即触发 APB_SARADC_ADC_DONE_INT_RAW 中断。软件检测到该中断后，可启用从 APB_SARADC_ADC_DATA 中读取样本值。如需切换采样通道无需重复 1 至 5 步骤。

36.8.2 多通道采样配置

专用定时器驱动的多通道采样的具体配置如下：

1. 置位 `PMU_XPD_PERIF_I2C` 和 `PMU_PERIF_I2C_RST` 给 ADC 上电；
2. 配置 `PCR_SARADC_CLKM_SEL` 选择时钟源；
3. 配置时钟分频寄存器 `PCR_SARADC_CLKM_DIV_NUM` 和 `PCR_SAR1_CLK_DIV_NUM`；
4. 置位 `PCR_SARADC_CLKM_EN` 使能 SAR ADC 时钟；
5. 按照章节 36.5.8 样式表 所述步骤配置样式表；
6. 按照章节 36.5.9 ADC 滤波器 所述步骤配置滤波器作用的通道和滤波系数；
7. 根据需要，按照章节 36.5.10 阈值监控器 所述步骤设置阈值监控；
8. 置位 `APB_SARADC_APB_ADC_TRANS` 选择使用 GDMA；
9. 配置 `APB_SARADC_TIMER_TARGET` 设置 DIG ADC 定时器的触发周期；
10. 置位 `APB_SARADC_TIMER_EN` 使能定时器。

定时器超时则将驱动 DIG ADC FSM 根据样式表进行采样。数据通过 GDMA 自动存储到内存空间中。当采集到 `APB_SARADC_APB_ADC_EOF_NUM` 寄存器设置的次数时，采样停止。

采样完成将产生 `APB_SARADC_ADC_DONE_INT_RAW` 中断。

36.9 寄存器列表

本小节的所有地址均为相对于 SAR ADC 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器中的表 4.3-2。请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
APB_SARADC_CTRL_REG	SAR ADC 控制寄存器 1	0x0000	R/W
APB_SARADC_CTRL2_REG	SAR ADC 控制寄存器 2	0x0004	R/W
APB_SARADC_FILTER_CTRL1_REG	滤波控制寄存器 1	0x0008	R/W
APB_SARADC_SAR_PATT_TAB1_REG	样式表寄存器 1	0x0018	R/W
APB_SARADC_SAR_PATT_TAB2_REG	样式表寄存器 2	0x001C	R/W
APB_SARADC_ONETIME_SAMPLE_REG	单次采样配置寄存器	0x0020	R/W
APB_SARADC_FILTER_CTRL0_REG	滤波控制寄存器 0	0x0028	R/W
APB_SARADC_SAR1DATA_STATUS_REG	SAR ADC 转换数据存储寄存器	0x002C	RO
APB_SARADC_THRES0_CTRL_REG	采样阈值控制寄存器 0	0x0034	R/W
APB_SARADC_THRES1_CTRL_REG	采样阈值控制寄存器 1	0x0038	R/W
APB_SARADC_THRES_CTRL_REG	采样阈值控制寄存器	0x003C	R/W
APB_SARADC_INT_ENA_REG	SAR ADC 中断使能寄存器	0x0040	R/W
APB_SARADC_INT_RAW_REG	SAR ADC 原始中断寄存器	0x0044	R/WTC/SS
APB_SARADC_INT_ST_REG	SAR ADC 中断状态寄存器	0x0048	RO
APB_SARADC_INT_CLR_REG	SAR ADC 中断清除寄存器	0x004C	WT
APB_SARADC_DMA_CONF_REG	SAR ADC DMA 配置寄存器	0x0050	R/W
APB_SARADC_APB_CTRL_DATE_REG	版本控制寄存器	0x03FC	R/W

36.10 寄存器

本小节的所有地址均为相对于 SAR ADC 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4.3-2。

Register 36.1. APB_SARADC_CTRL_REG (0x0000)

(reserved)				APB_SARADC_XPD_SAR_FORCE				(reserved)				APB_SARADC_SAR_PATT_P_CLEAR				(reserved)				APB_SARADC_SAR_PATT_LEN				(reserved)				APB_SARADC_SAR_CLK_GATED				(reserved)				APB_SARADC_START				APB_SARADC_START_FORCE			
31	29	28	27	26	24	23	22	18	17	15	14	7	6	5	2	1	0	Reset																									
0		0		0	0	0	0	0	0	0	0	7	4		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0										

APB_SARADC_START_FORCE 配置是否使用软件启动 SAR ADC 采样。

0: 选择使用 FSM 启动 SAR ADC 采样

1: 选择使用软件启动 SAR ADC 采样

(R/W)

APB_SARADC_START 配置是否使用软件启动 SAR ADC。

0: 无效

1: 使用软件启动 SAR ADC

仅当 **APB_SARADC_START_FORCE** = 1 时有效。

(R/W)

APB_SARADC_SAR_CLK_GATED 配置是否使能 SAR ADC 时钟门控。

0: 不使能

1: 使能

(R/W)

APB_SARADC_SAR_PATT_LEN 配置需要使用的样式数量。

0: 仅使用 cmd0

1: 使用 cmd0 和 cmd1

n: 依次使用 cmd0 至 cmdn, n 最大取 7

(R/W)

APB_SARADC_SAR_PATT_P_CLEAR 配置是否清除 DIG ADC 控制器样式表指针。

0: 无效

1: 清除

(R/W)

APB_SARADC_XPD_SAR_FORCE 配置是否强制上电 SAR ADC。

0: 无效

1: 强制上电

(R/W)

Register 36.2. APB_SARADC_CTRL2_REG (0x0004)

(reserved)								APB_SARADC_TIMER_EN																APB_SARADC_TIMER_TARGET																(reserved)								(reserved)								APB_SARADC_SAR1_INV																APB_SARADC_MAX_MEAS_NUM																APB_SARADC_MEAS_NUM_LIMIT																																
31								25								24	23								12								11	10	9	8								1								0																																																																				
0								0								0								0								0								0								0								0								0								10								0								0								0								255								0								Reset

APB_SARADC_MEAS_NUM_LIMIT 配置是否使能 SAR ADC 最大转换次数限制。

0: 不使能

1: 使能

(R/W)

APB_SARADC_MAX_MEAS_NUM 配置 SAR ADC 的最大转换次数。(R/W)

APB_SARADC_SAR1_INV 配置是否反转 SAR ADC 数据。

0: 无效

1: 反转 SAR ADC 数据

(R/W)

APB_SARADC_TIMER_TARGET 设置 SAR ADC 定时器目标，即定时器的触发周期。(R/W)

APB_SARADC_TIMER_EN 配置是否使能 SAR ADC 定时器触发。

0: 不使能

1: 使能

(R/W)

Register 36.3. APB_SARADC_FILTER_CTRL1_REG (0x0008)

APB_SARADC_FILTER_FACTOR0										APB_SARADC_FILTER_FACTOR1										(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
31	29			28	26			25																					0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					
0					0					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

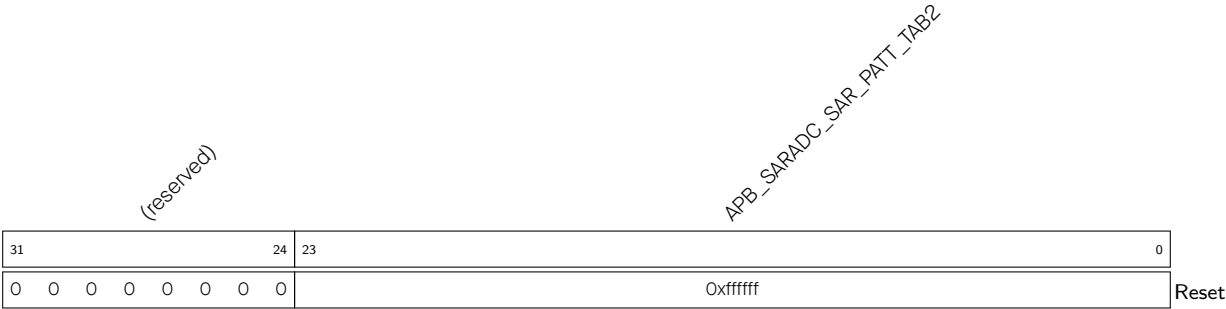
- APB_SARADC_FILTER_FACTOR1 设置 SAR ADC 滤波器 1 的滤波系数 k 。
0: $k=0$ （即滤波器关闭）
1: $k=2$
2: $k=4$
3: $k=8$
4: $k=16$
5: $k=32$
6: $k=64$
(R/W)
- APB_SARADC_FILTER_FACTOR0 设置 SAR ADC 滤波器 0 的滤波系数 k （配置同上）。(R/W)

Register 36.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)

(reserved)										APB_SARADC_SAR_PATT_TAB1																						
31									24	23																						0
0 0 0 0 0 0 0 0 0										0xffffffff																						Reset

- APB_SARADC_SAR_PATT_TAB1 配置样式表的第 0~3 个样式。每个样式占六位。具体见小节 [36.5.8 样式表](#)。(R/W)

Register 36.5. APB_SARADC_SAR_PATT_TAB2_REG (0x001C)



APB_SARADC_SAR_PATT_TAB2 配置样式表的第 4 ~ 7 个样式, 每个样式占六位。具体见小节 [36.5.8 样式表](#)。(R/W)

Register 36.7. APB_SARADC_FILTER_CTRL0_REG (0x0028)

APB_SARADC_FILTER_RESET						(reserved)						APB_SARADC_FILTER_CHANNEL0						APB_SARADC_FILTER_CHANNEL1						(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31	30					26	25	22			21	18			17																0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
0	0	0	0	0	0	0	13			13			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

APB_SARADC_FILTER_CHANNEL1 配置 SAR ADC 滤波通道 1。(R/W)

APB_SARADC_FILTER_CHANNEL0 配置 SAR ADC 滤波通道 0。(R/W)

APB_SARADC_FILTER_RESET 配置是否复位 SAR ADC 滤波器。
0: 无效
1: 复位
(R/W)

Register 36.8. APB_SARADC_1_DATA_STATUS_REG (0x002C)

(reserved)															APB_SARADC_DATA																
31																17	16														0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															Reset

APB_SARADC_DATA 存储 SAR ADC 单次采样的转换数据。(RO)

Register 36.9. APB_SARADC_THRESO_CTRL_REG (0x0034)

(reserved)		APB_SARADC_THRESO_LOW																APB_SARADC_THRESO_HIGH																(reserved)		APB_SARADC_THRESO_CHANNEL	
31	30	18																17	5																4	3	0
0	0																0x1fff																0	13		Reset	

APB_SARADC_THRESO_CHANNEL 配置 SAR ADC 阈值监控器 0 需要监控的通道。(R/W)

APB_SARADC_THRESO_HIGH 配置 SAR ADC 阈值监控器 0 的高阈值。(R/W)

APB_SARADC_THRESO_LOW 配置 SAR ADC 阈值监控器 0 的低阈值。(R/W)

Register 36.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

31	30	18	17	5	4	3	0	
0	0	0x1fff			0	13	Reset	

APB_SARADC_THRES1_CHANNEL 配置 SAR ADC 阈值监控器 1 需要监控的通道。(R/W)

APB_SARADC_THRES1_HIGH 配置 SAR ADC 閾值監控器 1 的高閾值。(R/W)

APB_SARADC_THRES1_LOW 配置 SAR ADC 閾值監控器 1 的低閾值。(R/W)

Register 36.11. APB_SARADC_THRES_CTRL_REG (0x003C)

Diagram illustrating the structure of the `APB_SARADC_THRES0_EN` register. The register is 32 bits wide, with bits 31, 30, 29, 28, 27, and 26 labeled. Bit 31 is labeled `APB_SARADC_THRES0_EN`, bit 30 is labeled `APB_SARADC_THRES1_EN`, bit 29 is labeled (reserved), bit 28 is labeled `APB_SARADC_THRES_ALL_EN`, bit 27 is labeled (reserved), and bit 26 is labeled (reserved). The register is shown as a horizontal bar with a 'Reset' label on the right.

APB_SARADC_THRES_ALL_EN 配置是否使能所有已配置通道的閾值監控。

0: 不使能

1: 使能

(R/W)

APB_SARADC_THRES1_EN 配置是否使能閾值監控器 1。

0: 不使能

1: 使能

(R/W)

APB_SARADC_THRES0_EN 配置是否使能閾值監控器 0。

0: 不使能

1: 使能

(R/W)

Register 36.12. APB_SARADC_INT_ENA_REG (0x0040)

Diagram illustrating the APB_SARADC register structure. The register is 32 bits wide, with bits 31 down to 24 labeled. Bit 31 is APB_SARADC_ADC_DONE_INT_ENA, bit 30 is (reserved), bit 29 is APB_SARADC_THRES0_HIGH_INT_ENA, bit 28 is APB_SARADC_THRES0_HIGH_INT_ENA, bit 27 is APB_SARADC_THRES0_LOW_INT_ENA, bit 26 is APB_SARADC_THRES1_LOW_INT_ENA, bit 25 is APB_SARADC_TSENS_INT_ENA, and bits 24 down to 0 are (reserved).

APB_SARADC_TSENS_INT_ENA 写1使能 APB_SARADC_TSENS_INT [TBA] 中断。(R/W)

APB_SARADC_THRES1_LOW_INT_ENA 写1使能 **APB_SARADC_THRES1_LOW_INT** 中断。(R/W)

APB_SARADC_THRESO_LOW_INT_ENA 写1使能 **APB_SARADC_THRESO_LOW_INT** 中断。(R/W)

APB_SARADC_THRES1_HIGH_INT_ENA 写1使能 APB_SARADC_THRES1_HIGH_INT 中断。(R/W)

APB_SARADC_THRESO_HIGH_INT_ENA 写1使能 APB_SARADC_THRESO_HIGH_INT 中断。(R/W)

APB_SARADC_ADC_DONE_INT_ENA 写1使能 APB_SARADC_ADC_DONE_INT 中断。(R/W)

Register 36.13. APB_SARADC_INT_RAW_REG (0x0044)

Diagram illustrating the structure of the **APB_SARADC_THRES1** register. The register is 32 bits wide, with bits 31 down to 24 labeled as follows:

- 31: **APB_SARADC_ADC_DONE_INT_RAW**
- 30: **(reserved)**
- 29: **APB_SARADC_THRES0_HIGH_INT_RAW**
- 28: **APB_SARADC_THRES1_HIGH_INT_RAW**
- 27: **APB_SARADC_THRES0_LOW_INT_RAW**
- 26: **APB_SARADC_THRES1_LOW_INT_RAW**
- 25: **APB_SARADC_TSENS_INT_RAW**
- 24: **(reserved)**

The remaining bits (23 down to 0) are labeled as **(reserved)**.

APB_SARADC_TSENS_INT_RAW **APB_SARADC_TSENS_INT** 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRES1_LOW_INT_RAW APB_SARADC_THRES1_LOW_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRESO_LOW_INT_RAW APB_SARADC_THRESO_LOW_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRES1_HIGH_INT_RAW APB_SARADC_THRES1_HIGH_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRESO_HIGH_INT_RAW APB_SARADC_THRESO_HIGH_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_ADC_DONE_INT_RAW APB_SARADC_ADC_DONE_INT 的原始中断状态。(R/WTC/SS)

Register 36.14. APB_SARADC_INT_ST_REG (0x0048)

APB_SARADC_ADC_DONE_INT_ST (reserved) APB_SARADC_THRES0_HIGH_INT_ST APB_SARADC_THRES1_HIGH_INT_ST APB_SARADC_THRES0_LOW_INT_ST APB_SARADC_THRES1_LOW_INT_ST APB_SARADC_TSENS_INT_ST (reserved)																																	
31	30	29	28	27	26	25	24																									0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

APB_SARADC_TSENS_INT_ST APB_SARADC_TSENS_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES1_LOW_INT_ST APB_SARADC_THRES1_LOW_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES0_LOW_INT_ST APB_SARADC_THRES0_LOW_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES1_HIGH_INT_ST APB_SARADC_THRES1_HIGH_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES0_HIGH_INT_ST APB_SARADC_THRES0_HIGH_INT 的屏蔽中断状态。(RO)

APB_SARADC_ADC_DONE_INT_ST APB_SARADC_ADC_DONE_INT 的屏蔽中断状态。(RO)

Register 36.15. APB_SARADC_INT_CLR_REG (0x004C)

APB_SARADC_ADC_DONE_INT_CLR (reserved) APB_SARADC_THRES0_HIGH_INT_CLR APB_SARADC_THRES1_HIGH_INT_CLR APB_SARADC_THRES0_LOW_INT_CLR APB_SARADC_THRES1_LOW_INT_CLR APB_SARADC_TSENS_INT_CLR (reserved)																																
31	30	29	28	27	26	25	24																									0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

APB_SARADC_TSENS_INT_CLR 写 1 清除 APB_SARADC_TSENS_INT 中断。(WT)

APB_SARADC_THRES1_LOW_INT_CLR 写 1 清除 APB_SARADC_THRES1_LOW_INT 中断。(WT)

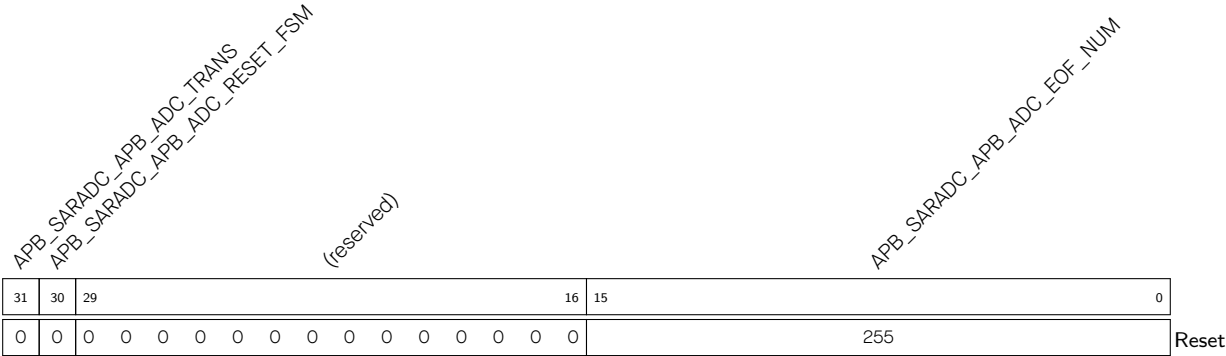
APB_SARADC_THRES0_LOW_INT_CLR 写 1 清除 APB_SARADC_THRES0_LOW_INT 中断。(WT)

APB_SARADC_THRES1_HIGH_INT_CLR 写 1 清除 APB_SARADC_THRES1_HIGH_INT 中断。(WT)

APB_SARADC_THRES0_HIGH_INT_CLR 写 1 清除 APB_SARADC_THRES0_HIGH_INT 中断。(WT)

APB_SARADC_ADC_DONE_INT_CLR 写 1 清除 APB_SARADC_ADC_DONE_INT 中断。(WT)

Register 36.16. APB_SARADC_DMA_CONF_REG (0x0050)



APB_SARADC_APB_ADC_EOF_NUM 当 ADC 采样得到的数据个数等于该寄存器配置值时，发送给 GDMA 的 EOF 标志位将被拉高。
(R/W)

APB_SARADC_APB_ADC_RESET_FSM 配置是否复位 DIG ADC 控制器状态。
0: 无效
1: 复位
(R/W)

APB_SARADC_APB_ADC_TRANS 配置 DIG ADC 控制器是否使用 GDMA。
0: 无效
1: 使用 GDMA
(R/W)

第 37 章

模拟电压比较器

37.1 概述

ESP32-C5 芯片集成了一个模拟电压比较器。模拟电压比较器依靠支持电压比较功能的特殊芯片焊盘 (PAD) 实现，用于监测 PAD 上的电压变化。模拟电压比较器有外部两个电压接入点，分为主电压和参考电压。模拟电压比较器输出的电压比较结果可以作为 ETM（事件任务矩阵）事件来驱动其他外设的 ETM 任务，也可以触发中断。

37.2 主要特性

模拟电压比较器具有以下主要特性：

- 支持电压比较功能
 - 电压比较模式可配置
 - 内部参考电压值可配置
- 支持电压比较中断
- 支持 ETM 事件

37.3 系统架构

图 37.3-1 展示了模拟电压比较器的系统架构。

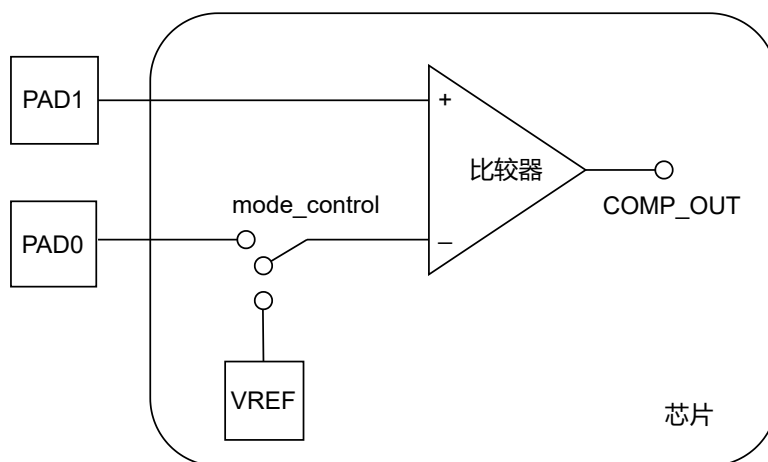


图 37.3-1. 模拟电压比较器系统框图

如上图所示，模拟电压比较器的结构如下：

- 两个 PAD 接口：
 - PAD1：主电压接口
 - PADO：外部参考电压接口
- VREF：内部参考电压接口
- mode_control：寄存器控制的选择开关，用于选择参考电压的输入源
- COMP_OUT：比较器输出的电压比较结果，为片上内部信号，外部无法直接获取
 - 若主电压高于参考电压，COMP_OUT 信号输出为高电平
 - 若主电压低于参考电压，COMP_OUT 信号输出为低电平

37.4 功能描述

模拟电压比较器具有以下功能：

- 电压比较

模拟电压比较器采用的是支持电压比较功能的特殊 PAD，这种特殊 PAD 由两个通用 PAD 组成。下表列出了模拟电压比较器的 PAD 与 GPIO 管脚之间的映射关系。

表 37.4-1. 模拟电压比较器 PAD 和 GPIO 之间的映射关系

模拟电压比较器	PADO	PAD1
模拟电压比较器	GPIO9	GPIO8

电压比较 PAD 的比较模式和参考电压可配置，具体配置流程见小节 [37.7 配置流程](#)。

电压比较结果输出为 COMP_OUT 信号。

- 电压比较模式控制

模拟电压比较器将主电压与参考电压进行比较，参考电压可选内部参考电压和外部参考电压。根据参考电压的选择不同，电压比较模式分为两种：

- 主电压与外部参考电压比较；
- 主电压与内部参考电压比较；

电压比较模式可通过 `GPIO_EXT_PAD_COMP_CONFIG_0_REG` 配置。

- 参考电压配置

内部参考电压范围为 $0 \sim (0.7 \times VDDPST1)$ V，步长为 $0.1 \times VDDPST1$ V，其中 VDDPST1 为模拟电压比较器的供电电压，等于芯片的电源电压（通常为 3.3 V）。内部参考电压值可通过寄存器 `LPSYSREG_DREF_COMPn` ($n = 0 \sim 1$) 配置。

外部参考电压直接从 PAD 接入，无需配置，输入范围与内部参考电压一致，为 $0 \sim (0.7 \times VDDPST1)$ V。

- 电压比较中断处理

电压比较的输出值 COMP_OUT 信号为高电平或低电平，只要输出值发生变化，就会产生对应的中断信号。

37.5 事件任务矩阵

在 ESP32-C5 中，模拟电压比较器支持 ETM 功能，即可以通过模拟电压比较器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 8 事件任务矩阵 (ETM)。这里仅介绍与模拟电压比较器相关的 ETM 事件。

模拟电压比较器可产生的 ETM 事件有：

- GPIO_EVT_ZERO_DET_POS：表示 COMP_OUT 信号由低电平变为高电平，即主电压从低于参考电压变为高于参考电压。
- GPIO_EVT_ZERO_DET_NEG：表示 COMP_OUT 信号由高电平变为低电平，即主电压从高于参考电压变为低于参考电压。

模拟电压比较器不支持任何 ETM 任务。

37.6 中断

ESP32-C5 的模拟电压比较器可以生成 GPIO_PAD_COMP_INT 中断信号并发送给中断矩阵。

这一中断信号由模拟电压比较器的内部中断源生成。表 37.6-1 列出了模拟电压比较器的内部中断源、中断触发条件、以及生成的中断信号。

表 37.6-1. 模拟电压比较器的内部中断源

内部中断源	触发条件	中断信号
GPIO_COMP_ALL_INT	COMP_OUT 信号发生跳变	GPIO_PAD_COMP_INT
GPIO_COMP_NEG_INT	COMP_OUT 信号由高电平变为低电平	GPIO_PAD_COMP_INT
GPIO_COMP_POS_INT	COMP_OUT 信号由低电平变为高电平	GPIO_PAD_COMP_INT

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 3 中断矩阵 [to be added later] > 3.1 中断术语。

每个中断源都有一组通用的配置寄存器，详见章节 中断配置寄存器。具体寄存器名称请查看 5.18.1 HP GPIO 交换矩阵寄存器列表。

37.7 配置流程

软件配置电压比较器功能的流程如下：

说明：

有关本节提及的寄存器字段的更多信息，请参阅章节 5 GPIO 交换矩阵和 IO MUX (GPIO, IO MUX)。

1. 将 PCR_IOMUX_FUNC_CLK_EN 设置为 1。
2. 将 GPIO_EXT_XPD_COMP_0 设置为 1，启用电压比较器功能。

3. 禁用 PAD1 的正常数字 PAD 功能，包括 IE、OE、WPU 和 WPD，如果使用外部参考电压，则同时禁用 PADO 的正常数字 PAD 功能。具体的配置方式请参考章节 [5 GPIO 交换矩阵和 IO MUX \(GPIO, IO MUX\)](#)。
4. 配置比较模式，将 [GPIO_EXT_PAD_COMP_CONFIG_0_REG](#) 设置为：
 - 0：选择主电压和内部参考电压进行比较；
 - 1：选择主电压和外部参考电压进行比较。
5. 通过 [GPIO_EXT_DREF_COMP](#) 配置内部参考电压，可调电压范围为 $0 \sim (0.7 \times VDDPST1) \text{ V}$ ，步长为 $0.1 \times VDDPST1 \text{ V}$ 。

第 VII 卷

附录

附录部分包括以下内容：

- [相关文档和资源](#)
- [词汇列表](#)
- [如何配置寄存器的保留域](#)
- [中断配置寄存器](#)
- [修订历史](#)

相关文档和资源

相关文档

- [《ESP32-C5 技术规格书》](#) – 提供 ESP32-C5 芯片的硬件技术规格。
- [《ESP32-C5 硬件设计指南》](#) – 提供基于 ESP32-C5 芯片的产品设计规范。
- 证书
<https://espressif.com/zh-hans/support/documents/certificates>
- ESP32-C5 产品/工艺变更通知 (PCN)
<https://espressif.com/zh-hans/support/documents/pcns?keys=ESP32-C5>
- ESP32-C5 公告 – 提供有关安全、bug、兼容性、器件可靠性的信息
<https://espressif.com/zh-hans/support/documents/advisories?keys=ESP32-C5>
- 文档更新和订阅通知
<https://espressif.com/zh-hans/support/download/documents>

开发者社区

- [《ESP32-C5 ESP-IDF 编程指南》](#) – ESP-IDF 开发框架的文档中心。
- ESP-IDF 及 GitHub 上的其它开发框架
<https://github.com/espressif>
- ESP32 论坛 – 工程师对工程师 (E2E) 的社区，您可以在这里提出问题、解决问题、分享知识、探索观点。
<https://esp32.com/>
- *The ESP Journal* – 分享乐鑫工程师的最佳实践、技术文章和工作随笔。
<https://blog.espressif.com/>
- SDK 和演示、App、工具、AT 等下载资源
<https://espressif.com/zh-hans/support/download/sdks-demos>

产品

- ESP32-C5 系列芯片 – ESP32-C5 全系列芯片。
<https://espressif.com/zh-hans/products/socs?id=ESP32-C5>
- ESP32-C5 系列模组 – ESP32-C5 全系列模组。
<https://espressif.com/zh-hans/products/modules?id=ESP32-C5>
- ESP32-C5 系列开发板 – ESP32-C5 全系列开发板。
<https://espressif.com/zh-hans/products/devkits?id=ESP32-C5>
- ESP Product Selector (乐鑫产品选型工具) – 通过筛选性能参数、进行产品对比快速定位您所需要的产品。
<https://products.espressif.com/#/product-selector?language=zh>

联系我们

- 商务问题、技术支持、电路原理图 & PCB 设计审阅、购买样品 (线上商店)、成为供应商、意见与建议
<https://espressif.com/zh-hans/contact-us/sales-questions>

词汇列表

外设相关词汇

ADC	ADC 控制器
AES	AES 加速器
CAN FD	控制器局域网灵活数据速率
DSA	数字签名算法
ECC	ECC 加速器
ECDSA	椭圆曲线数字签名算法
eFuse	eFuse 控制器
ETM	事件任务矩阵
GDMA	通用 DMA 控制器
HMAC	HMAC 加速器
I2C	I2C 控制器
I2S	I2S 控制器
LEDC	LED PWM 控制器
MCPWM	电机控制脉宽调制器
PARLIO	并行 IO 控制器
PCNT	脉冲计数器控制器
PMS	权限控制
RMT	红外遥控
RNG	随机数生成器
RSA	RSA 加速器
SDHOST	SD/MMC 主机控制器
SHA	SHA 加速器
SPI	SPI 控制器
TIMG	定时器组
TRACE	RISC-V Trace 编码器
TSENS	温度传感器
UART	UART 控制器
WDT	看门狗定时器
XTS_AES	片外存储器加密与解密

寄存器相关缩写

REG	寄存器。
SYSREG	系统寄存器 是一组控制系统复位、存储器、时钟、软件中断、电源管理、时钟门控等的寄存器。
ISO	隔离。 如果外设或其他芯片组件断电，其输出信号的管脚（若有）将会浮空。ISO 寄存器会隔离上述引脚并令其保持在某个确定值，以使连接到这些引脚的其他非断电外设/设备免受影响。
NMI	非屏蔽中断 是一种 CPU 指令无法禁用或忽略的硬件中断。出现此类中断说明发生严重错误。

W1TS 添加到寄存器/字段名称中的缩写，表示此类寄存器/字段用于置位名称相似寄存器中的相应字段。例如，寄存器 `GPIO_ENABLE_W1TS_REG` 用于置位寄存器 `GPIO_ENABLE_REG` 中的相应字段。

W1TC 与 *W1TS* 相同，但用于清除相应寄存器中的字段。

寄存器的访问类型

TRM 章节 寄存器列表 和 寄存器 详述了寄存器及其字段的访问类型。

常用访问类型及组合如下：

• RO	• R/W/SS	• R/WS/SS/SC
• WO	• R/W/SS/SC	• R/SS/WTC
• WT	• R/WC/SS	• R/SC/WTC
• R/W	• R/WC/SC	• R/SS/SC/WTC
• R/W1	• R/WC/SS/SC	• RF/WF
• WL	• R/WS/SC	• R/SS/RC
• R/W/SC	• R/WS/SS	• varies

下文提供了所有访问类型的具体描述。

- R 软件可读。**用户软件可以读取此寄存器/字段；通常与其他访问类型结合使用。
- RO 软件只读。**用户软件只可读取此寄存器/字段。
- HRO 硬件只读。**仅硬件可以读取此寄存器/字段；用于存储变量参数的默认设置。
- W 软件可写。**用户软件可以写入此寄存器/字段；通常与其他访问类型结合使用。
- WO 软件只写。**用户软件只可写入此寄存器/字段。
- W1 软件只写一次。**用户软件只可写入一次此寄存器/字段；只允许写入 1，写入 0 无效。
- SS 硬件置位。**在指定事件中，硬件自动将 1 写入此寄存器/字段；与一位字段一同使用。
- SC 硬件清零。**在指定事件中，硬件自动将 0 写入此寄存器/字段；与一位和多位字段一同使用。
- SM 硬件修改。**在指定事件中，硬件自动将指定值写入此寄存器/字段；与多位字段一同使用。
- SU 硬件更新。**在指定事件中，硬件自动更新此寄存器/字段；与多位字段一同使用。
- RS 软件读置位。**如果用户软件读取此寄存器/字段，硬件会自动写 1。
- RC 软件读清零。**如果用户软件读取此寄存器/字段，硬件会自动写 0。
- RF 软件读 FIFO。**如果用户软件将新数据写入 FIFO，寄存器/字段会自动读取。
- WF 软件写 FIFO。**如果用户软件将新数据写入此寄存器/字段，寄存器/字段会自动通过 APB 总线将数据传递到 FIFO。
- WS 软件写置位。**如果用户软件写入此寄存器/字段，硬件会自动置位此寄存器/字段。
- W1S 软件写 1 置位。**如果用户软件将 1 写入此寄存器/字段，硬件会自动置位此寄存器/字段。
- WOS 软件写 0 置位。**如果用户软件将 0 写入此寄存器/字段，硬件会自动置位此寄存器/字段。
- WC 软件写清零。**如果用户软件写入此寄存器/字段，硬件会自动清零此寄存器/字段。
- W1C 软件写 1 清零。**如果用户软件将 1 写入此寄存器/字段，硬件会自动清零此寄存器/字段。
- WOC 软件写 0 清零。**如果用户软件将 0 写入此寄存器/字段，硬件会自动清零此寄存器/字段。
- WT 软件写产生边沿触发信号。**如果用户软件将 1 写入此字段，将会产生边沿触发信号（APB 总线中的脉冲）或清除相应的 WTC 字段（详见 WTC）。

- WTC **软件写其他寄存器位清零本寄存器位。**如果用户软件将 1 写入相应的 WT 字段，硬件会自动清除此字段（详见 WT）。
- W1T **软件写 1 取反。**如果用户软件将 1 写入此字段，硬件会自动取反相应字段，否则不会取反。
- WOT **软件写 0 取反。**如果用户软件将 0 写入此字段，硬件会自动取反相应字段，否则不会取反。
- WL **软件仅在锁禁用时写。**如果锁被禁用，用户软件可以写入此寄存器/字段。
- varies **访问类型不定。**此寄存器中的不同字段访问类型可能不同。

如何配置寄存器的保留域

概述

寄存器的保留域指的是寄存器中不对用户开放、或者配置为非默认值时会导致不可预测的结果的域。

如何配置保留域

保留域的值不能修改。由于写寄存器时必须整体写，不能只写部分域，因此，在写带有保留域的寄存器时需要特别注意，只能采用以下两种方式：

1. 读取寄存器的值，仅修改需要配置的域，然后将修改后的值以及其他未修改的值一起写回寄存器，这样保留域的值就会保持不变。

或者

2. 仅修改需要配置的域，然后将保留域的默认值写回寄存器。默认值即寄存器图表中的“Reset”值。例如，寄存器 X 中 Field_A 的默认值为 1。

Register 371. 寄存器 X (地址)																																
(reserved)												Field_C				(reserved)												Field_B		Field_A		
3120												1916				152												1		0		
0 0 0 0 0 0 0 0 0 0 0 0												0000				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0												0		1		Reset

假设您要将 寄存器 X 的 Field_A、Field_B 和 Field_C 设置为 0x0、0x1 和 0x2，您可以：

- 使用第 1 种方式，修改这三个域的值，然后把读到的值写回寄存器。假设寄存器读取的值为 0x0000_0003，修改三个域的值后，将值 0x0002_0002 写入寄存器。
- 使用第 2 种方式，修改这三个域的值，然后把保留域的默认值写回寄存器，即把 0x0002_0002 写入寄存器。

中断配置寄存器

大部分外设的内部中断源都有以下配置寄存器：

- **RAW**（原始状态）寄存器：该寄存器指示原始中断状态，每个位对应一个内部中断源。当中断源触发时，其 RAW 位为 1。
- **ENA**（使能）寄存器：该寄存器用于启用或禁用内部中断源，每个位对应一个内部中断源。

通过操作 ENA 寄存器，可以根据需要屏蔽或取消屏蔽某个内部中断源。当中断源被屏蔽（禁用）时，它不会生成中断信号，但仍可以从 RAW 寄存器中读取其值。

- **ST**（状态）寄存器：该寄存器指示中断源的屏蔽状态，每个位对应一个内部中断源。ST 位为 1 代表 RAW 位和 ENA 位都为 1，即中断源已生成且未被屏蔽。RAW 位和 ENA 位的值为其他组合时，ST 位为 0。

ENA/RAW/ST 寄存器的配置见表 37.7-4。

- **CLR**（清除）寄存器：CLR 寄存器负责清除内部中断源。写 1 将清除该位对应的中断源。

表 37.7-4. ENA/RAW/ST 寄存器的配置

ENA 位的值	RAW 位的值	ST 位的值
0	忽略	0
1	0	0
	1	1

修订历史

日期	版本	发布说明
2025-05-20	v0.1	首次发布



免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，乐鑫不对信息的准确性、真实性做任何保证。

乐鑫不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他乐鑫提案、规格书或样品在他处提到的任何保证。

乐鑫不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

版权归 © 2025 乐鑫信息科技（上海）股份有限公司。保留所有权利。

www.espressif.com