

# ESP32-S2

## 技术参考手册



预发布 V0.2  
乐鑫信息科技  
版权 © 2020

## 关于本手册

《ESP32-S2 技术参考手册》的目标读者群体是使用 ESP32-S2 芯片的应用开发工程师。本手册提供了关于 ESP32-S2 的具体信息，包括各个功能模块的内部架构、功能描述和寄存器配置等。

芯片的管脚描述、电气特性和封装信息等可以从 [《ESP32-S2 技术规格书》](#) 获取。

## 文档版本

请至乐鑫官网 <https://www.espressif.com/zh-hans/support/download/documents> 下载最新本本文档。

## 修订历史

请至文档最后页查看 [修订历史](#)。

## 文档变更通知

用户可以通过乐鑫官网订阅页面 [www.espressif.com/zh-hans/subscribe](http://www.espressif.com/zh-hans/subscribe) 订阅技术文档变更的电子邮件通知。

## 证书下载

用户可以通过乐鑫官网证书下载页面 [www.espressif.com/zh-hans/certificates](http://www.espressif.com/zh-hans/certificates) 下载产品证书。

## 免责声明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，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 © 2020 乐鑫所有。保留所有权利。

# 目录

<b>1</b>	<b>系统和存储器</b>	<b>16</b>
1.1	概述	16
1.2	主要特性	16
1.3	功能描述	17
1.3.1	地址映射	17
1.3.2	内部存储器	18
1.3.2.1	Internal ROM 0	19
1.3.2.2	Internal ROM 1	19
1.3.2.3	Internal SRAM 0	19
1.3.2.4	Internal SRAM 1	19
1.3.2.5	RTC FAST Memory	20
1.3.2.6	RTC SLOW Memory	20
1.3.3	外部存储器	20
1.3.3.1	外部存储器地址映射	20
1.3.3.2	高速缓存	21
1.3.3.3	Cache 操作	21
1.3.4	DMA 地址空间	21
1.3.5	模块 / 外设地址空间	22
1.3.5.1	外设总线名称约定	22
1.3.5.2	外设总线区别	22
1.3.5.3	模块 / 外设地址空间列表	23
1.3.5.4	PeriBus1 访问受限地址列表	24
<b>2</b>	<b>复位和时钟</b>	<b>26</b>
2.1	复位	26
2.1.1	概述	26
2.1.2	复位源	26
2.2	系统时钟	27
2.2.1	概述	27
2.2.2	时钟源	28
2.2.3	CPU 时钟	29
2.2.4	外设时钟	30
2.2.4.1	APB_CLK 源	30
2.2.4.2	REF_TICK 源	30
2.2.4.3	LEDC_PWM_CLK 源	31
2.2.4.4	APLL_SCLK 源	31
2.2.4.5	PLL_160M_CLK 源	31
2.2.4.6	时钟源注意事项	31
2.2.5	Wi-Fi 时钟	32
2.2.6	RTC 时钟	32
2.2.7	音频 PLL 时钟	32
<b>3</b>	<b>芯片 Boot 控制</b>	<b>33</b>

3.1	概述	33
3.2	Boot 控制	33
3.3	ROM Code 打印	34
3.4	VDD_SPI 电压	34
<b>4</b>	<b>中断矩阵</b>	<b>35</b>
4.1	概述	35
4.2	主要特性	35
4.3	功能描述	35
4.3.1	外部中断源	35
4.3.2	CPU 中断	39
4.3.3	分配外部中断源至 CPU 外部中断	40
4.3.3.1	分配一个外部中断源 Source_X 至 CPU 外部中断	40
4.3.3.2	分配多个外部中断源 Source_X <sub>n</sub> 至 CPU 外部中断	40
4.3.3.3	关闭 CPU 外部中断源 Source_X	40
4.3.4	关闭 CPU 的 NMI 类型中断源	41
4.3.5	查询外部中断源当前的中断状态	41
4.4	基地址	41
4.5	寄存器列表	41
4.6	寄存器	46
<b>5</b>	<b>IO MUX 和 GPIO 交换矩阵</b>	<b>83</b>
5.1	概述	83
5.2	通过 GPIO 交换矩阵的外设输入	84
5.2.1	概述	84
5.2.2	信号同步	84
5.2.3	功能描述	85
5.2.4	简单 GPIO 输入	86
5.3	通过 GPIO 交换矩阵的外设输出	86
5.3.1	概述	86
5.3.2	功能描述	87
5.3.3	简单 GPIO 输出	88
5.3.4	Sigma Delta 调制输出 (SDM)	88
5.3.4.1	功能描述	88
5.3.4.2	配置方法	89
5.4	IO MUX 的直接 I/O 功能	89
5.4.1	概述	89
5.4.2	功能描述	89
5.5	RTC IO MUX 的低功耗和模拟 I/O 功能	89
5.5.1	概述	89
5.5.2	功能描述	90
5.6	Light-sleep 模式管脚功能	90
5.7	Pad Hold 特性	90
5.8	I/O Pad 供电	91
5.8.1	电源管理	91
5.9	外设信号列表	91

5.10 IO MUX Pad 列表	95
5.11 RTC IO MUX 管脚清单	97
5.12 基地址	98
5.13 寄存器列表	98
5.13.1 GPIO 交换矩阵寄存器列表	98
5.13.2 IO MUX 寄存器列表	99
5.13.3 SDM 寄存器列表	101
5.13.4 RTC IO MUX 寄存器列表	101
5.14 寄存器	102
5.14.1 GPIO 交换矩阵寄存器	102
5.14.2 IO MUX 寄存器	114
5.14.3 SDM 寄存器	115
5.14.4 RTC IO MUX 寄存器	117
<b>6 系统寄存器</b>	<b>130</b>
6.1 概述	130
6.2 主要特性	130
6.3 功能描述	130
6.3.1 系统和存储器寄存器	130
6.3.2 复位和时钟寄存器	132
6.3.3 中断矩阵寄存器	132
6.3.4 JTAG 软件使能寄存器	132
6.3.5 低功耗管理寄存器	133
6.3.6 外设时钟门控和复位寄存器	133
6.4 基地址	135
6.5 寄存器列表	135
6.6 寄存器	136
<b>7 DMA 控制器</b>	<b>151</b>
7.1 概述	151
7.2 特性	151
7.3 功能描述	152
7.3.1 DMA 引擎的架构	152
7.3.2 链表	152
7.3.3 启动 DMA	153
7.3.4 读链表	154
7.3.5 数据传输结束标志	154
7.3.6 访问片外 RAM	154
7.4 Copy DMA 控制器	155
7.5 UART DMA (UDMA) 控制器	155
7.6 SPI DMA 控制器	156
7.7 I <sup>2</sup> S DMA 控制器	157
<b>8 通用异步收发传输器 (UART)</b>	<b>159</b>
8.1 概述	159
8.2 主要特性	159

8.3	功能描述	159
8.3.1	UART 简介	159
8.3.2	UART 架构	160
8.3.3	UART RAM	161
8.3.4	波特率产生与检测	161
8.3.4.1	波特率产生	161
8.3.4.2	波特率检测	162
8.3.5	UART 数据帧	163
8.3.6	RS485	164
8.3.6.1	驱动控制	164
8.3.6.2	转换延时	164
8.3.6.3	总线侦听	165
8.3.7	IrDA	165
8.3.8	唤醒	166
8.3.9	流控	166
8.3.9.1	硬件流控	167
8.3.9.2	软件流控	168
8.3.10	UDMA	168
8.3.11	UART 中断	168
8.3.12	UCHI 中断	169
8.4	基地址	170
8.5	寄存器列表	170
8.6	寄存器	173
<b>9</b>	<b>LED PWM 控制器</b>	215
9.1	概述	215
9.2	特性	215
9.3	功能描述	215
9.3.1	架构	215
9.3.2	定时器	216
9.3.3	PWM 生成器	217
9.3.4	渐变占空比	217
9.3.5	中断	218
9.4	基地址	218
9.5	寄存器列表	219
9.6	寄存器	221
<b>10</b>	<b>红外遥控</b>	226
10.1	概述	226
10.2	功能描述	226
10.2.1	RMT 架构	226
10.2.2	RMT RAM	227
10.2.3	时钟	227
10.2.4	发射器	227
10.2.5	接收器	228
10.2.6	中断	228

10.3 基地址	229
10.4 寄存器列表	229
10.5 寄存器	231
<b>11 脉冲计数器</b>	<b>240</b>
11.1 特性	240
11.2 功能描述	241
11.3 应用实例	243
11.3.1 通道 0 独自递增计数	243
11.3.2 通道 0 独自递减计数	243
11.3.3 通道 0 和通道 1 同时递增计数	244
11.4 基地址	245
11.5 寄存器列表	245
11.6 寄存器	247
<b>12 64 位定时器</b>	<b>252</b>
12.1 概述	252
12.2 功能描述	253
12.2.1 16 位预分频器与时钟选择	253
12.2.2 64 位时基计数器	253
12.2.3 报警产生	253
12.2.4 定时器重新加载	253
12.2.5 中断	254
12.3 配置与使用	254
12.3.1 定时器用作简单时钟	254
12.3.2 定时器用于一次性报警	255
12.3.3 定时器用于周期性报警	255
12.4 基地址	256
12.5 寄存器列表	256
12.6 寄存器	258
<b>13 看门狗定时器</b>	<b>267</b>
13.1 概述	267
13.2 特性	267
13.3 功能描述	267
13.3.1 时钟源与 32 位计数器	267
13.3.2 阶段与超时动作	268
13.3.3 写保护	268
13.3.4 Flash 引导保护	268
13.4 寄存器	269
<b>14 eFuse 控制器</b>	<b>270</b>
14.1 概述	270
14.2 主要特性	270
14.3 功能描述	270
14.3.1 结构	270

14.3.1.1	EFUSE_WR_DIS	273
14.3.1.2	EFUSE_RD_DIS	274
14.3.1.3	数据存储方式	274
14.3.2	软件烧写参数	274
14.3.3	软件读取参数	276
14.3.4	时序	277
14.3.4.1	eFuse 烧写时序	277
14.3.4.2	eFuse VDDQ 时序	278
14.3.4.3	eFuse 读取时序	278
14.3.5	硬件模块使用参数	279
14.3.6	中断	279
14.4	基地址	279
14.5	寄存器列表	279
14.6	寄存器	283
<b>15</b>	<b>I<sup>2</sup>C 控制器</b>	<b>305</b>
15.1	概述	305
15.2	主要特性	305
15.3	I <sup>2</sup> C 功能描述	305
15.3.1	I <sup>2</sup> C 简介	305
15.3.2	I <sup>2</sup> C 架构	306
15.3.2.1	TX/RX RAM	307
15.3.2.2	CMD_Controller	307
15.3.2.3	SCL_FSM	308
15.3.2.4	SCL_MAIN_FSM	309
15.3.2.5	DATA_Shifter	309
15.3.2.6	SCL_Filter 和 SDA_Filter	309
15.3.3	I <sup>2</sup> C 总线时序	309
15.4	典型应用	310
15.4.1	I <sup>2</sup> C 主机写入从机, 7-bit 寻址, 单次命令序列	311
15.4.2	I <sup>2</sup> C 主机写入从机, 10-bit 寻址, 单次命令序列	312
15.4.3	I <sup>2</sup> C 主机写入从机, 7-bit 双地址寻址, 单次命令序列	313
15.4.4	I <sup>2</sup> C 主机写入从机, 7-bit 寻址, 多次命令序列	313
15.4.5	I <sup>2</sup> C 主机读取从机, 7-bit 寻址, 单次命令序列	314
15.4.6	I <sup>2</sup> C 主机读取从机, 10-bit 寻址, 单次命令序列	315
15.4.7	I <sup>2</sup> C 主机读取从机, 7-bit 双寻址, 单次命令序列	315
15.4.8	I <sup>2</sup> C 主机读取从机, 7-bit 寻址, 多次命令序列	316
15.5	SCL 延展传输	317
15.6	中断	317
15.7	基地址	318
15.8	寄存器列表	318
15.9	寄存器	320
<b>16</b>	<b>AES 加速器</b>	<b>340</b>
16.1	概述	340
16.2	主要特性	340



16.3 工作模式简介	340
16.4 Typical AES 工作模式	341
16.4.1 密钥、明文、密文	341
16.4.2 字节序	342
16.4.3 Typical AES 工作模式的流程	346
16.5 DMA-AES 工作模式	347
16.5.1 密钥、明文、密文	347
16.5.2 字节序	348
16.5.3 标准增量函数	348
16.5.4 块个数	349
16.5.5 初始向量	349
16.5.6 块运算模式的流程	349
16.5.7 GCM 运算模式的流程	350
16.6 GCM 算法	351
16.6.1 哈希子密钥 (Hash subkey)	352
16.6.2 $J_0$	352
16.6.3 认证标签 (Authenticated Tag)	352
16.6.4 附加认证消息块个数 (AAD Block Number)	352
16.6.5 不完整块的有效比特数 (Remainder Bit Number)	352
16.7 基地址	353
16.8 存储器列表	353
16.9 寄存器列表	354
16.10 寄存器	355
<b>17 SHA 加速器</b>	<b>359</b>
17.1 概述	359
17.2 主要特性	359
17.3 工作模式简介	359
17.4 功能描述	360
17.4.1 信息预处理	360
17.4.1.1 附加填充比特	360
17.4.1.2 信息解析	361
17.4.1.3 哈希初始值 (Initial Hash Value)	362
17.4.2 哈希运算流程	363
17.4.2.1 Typical SHA 模式下的运算流程	363
17.4.2.2 DMA-SHA 模式下的运算流程	365
17.4.3 信息摘要存储	367
17.4.4 中断	367
17.5 基地址	367
17.6 寄存器列表	368
17.7 寄存器	369
<b>18 RSA 加速器</b>	<b>374</b>
18.1 概述	374
18.2 主要特性	374
18.3 功能描述	374

18.3.1 大数模幂运算	375
18.3.2 大数模乘运算	376
18.3.3 大数乘法运算	377
18.3.4 加速选项	377
18.4 基地址	378
18.5 存储器列表	378
18.6 寄存器列表	379
18.7 寄存器	379
<b>19 随机数发生器</b>	<b>383</b>
19.1 概述	383
19.2 主要特性	383
19.3 功能描述	383
19.4 基地址	384
19.5 寄存器列表	384
19.6 寄存器	384
<b>20 片外存储器加密与解密</b>	<b>385</b>
20.1 概述	385
20.2 主要特性	385
20.3 功能描述	385
20.3.1 XTS 算法	386
20.3.2 密钥 Key	386
20.3.3 目标空间	387
20.3.4 数据填充	387
20.3.5 手动加密模块	388
20.3.6 自动加密模块	389
20.3.7 自动解密模块	389
20.4 基地址	390
20.5 寄存器列表	390
20.6 寄存器	391
<b>21 数字签名</b>	<b>395</b>
21.1 概述	395
21.2 主要特性	395
21.3 功能描述	395
21.3.1 概述	395
21.3.2 私钥运算符	395
21.3.3 约定	396
21.3.4 软件存储私钥	396
21.3.5 硬件工作流程	397
21.3.6 软件层面的 DS 操作	398
21.4 基地址	399
21.5 存储器列表	399
21.6 寄存器列表	399
21.7 寄存器	400

修订历史

402

## 表格

1	地址映射	18
2	内部存储器地址映射	18
3	外部存储器地址映射	20
4	具有 DMA 功能的模块 / 外设	22
5	模块 / 外设地址空间映射表	23
6	访问受限的地址	24
7	复位源	27
8	CPU_CLK 源	29
9	CPU_CLK 选择	29
10	外设时钟用法	30
11	APB_CLK 源	30
12	REF_TICK 源	31
13	LEDC_PWM_CLK 源	31
14	Strapping 管脚默认上拉/下拉	33
15	系统启动模式	33
16	ROM Code 打印控制	34
17	CPU 外部中断配置寄存器、外部中断状态寄存器、外部中断源	36
18	CPU 中断	39
19	中断矩阵基地址	41
21	IO MUX Light-sleep 管脚功能寄存器	90
22	GPIO 交换矩阵	91
23	IO MUX Pad 列表	95
24	RTC IO MUX 管脚清单	97
25	GPIO 与 RTCIO 基地址	98
30	ROM 控制位	131
31	SRAM 控制位	131
32	外设时钟门控与复位控制位	133
33	系统寄存器基地址	135
35	配置寄存器与写地址的关系表	154
36	UART0、UART1 与 UHCI 基地址	170
39	LED_PWM 基地址	218
41	RMT 基地址	229
43	控制信号为低电平时输入脉冲信号上升沿的计数模式	241
44	控制信号为高电平时输入脉冲信号上升沿的计数模式	242
45	控制信号为低电平时输入脉冲信号下降沿的计数模式	242
46	控制信号为高电平时输入脉冲信号下降沿的计数模式	242
47	PCNT 基地址	245
49	64 位定时器基地址	256
51	BLOCK0 参数	270
52	密钥用途数值对应的含义	272
53	BLOCK1-10 参数	273
55	eFuse 烧写时序参数配置	277
56	VDDQ 时序参数配置	278
57	eFuse 读取时序参数配置	279

58	eFuse 控制器基地址	279
60	I <sup>2</sup> C 控制器基地址	318
62	工作模式	341
63	运算模式选择	341
64	状态返回值	341
65	Typical AES 文本字节序	342
66	AES-128 密钥字节序	344
67	AES-192 密钥字节序	344
68	AES-256 密钥字节序	345
69	运算模式选择	347
70	状态返回值	347
71	TEXT-PADDING	348
72	DMA AES 存储字节序	348
73	AES 基地址	353
76	工作模式选择	360
77	运算标准选择	360
81	不同运算标准信息摘要的寄存器占用情况	367
82	SHA 基地址	367
84	加速效果	378
85	RSA 基地址	378
88	随机数发生器基地址	384
90	Key 值映射表	386
91	目标空间与寄存器堆的映射关系	388
92	手动加密模块基地址	390
94	基地址	399

## 插图

1-1	系统结构与地址映射结构	17
1-2	Cache 系统框图	21
2-1	四种复位等级	26
2-2	系统时钟	28
4-1	中断矩阵结构图	35
5-1	IO MUX、RTC IO MUX 和 GPIO 交换矩阵结构框图	83
5-2	GPIO 输入经 clock 上升沿或下降沿同步	85
5-3	GPIO 输入信号滤波时序图	85
7-1	具有 DMA 功能的模块和支持的数据传输类型	151
7-2	DMA 引擎的架构	152
7-3	链表结构图	152
7-4	链表关系图	154
7-5	Copy DMA 引擎架构	155
7-6	UDMA 模式数据传输	156
7-7	SPI DMA	156
8-1	UART 基本架构图	160
8-2	UART 共享 RAM 图	161
8-3	UART 控制器分频	162
8-4	UART 信号下降沿较差时序图	162
8-5	UART 数据帧结构	163
8-6	AT_CMD 字符格式	163
8-7	RS485 模式驱动控制结构图	164
8-8	SIR 模式编解码时序图	165
8-9	IrDA 编解码结构图	166
8-10	硬件流控图	167
8-11	硬件流控信号连接图	167
9-1	LED_PWM 架构	215
9-2	LED_PWM 生成器图	216
9-3	LED_PWM 分频器	216
9-4	LED_PWM 输出信号图	217
9-5	渐变占空比输出信号图	218
10-1	RMT 结构框图	226
10-2	RAM 中脉冲编码结构	227
11-1	PCNT 框图	240
11-2	PCNT 单元基本架构图	241
11-3	通道 0 递增计数图	243
11-4	通道 0 递减计数图	244
11-5	双通道递增计数图	244
12-1	定时器组	252
14-1	移位寄存器电路图	274
14-2	eFuse 烧写时序图	277
14-3	eFuse 读取时序图	278
15-1	I <sup>2</sup> C Master 基本架构	306
15-2	I <sup>2</sup> C Slave 基本架构	306

15-3 I <sup>2</sup> C 命令寄存器结构	307
15-4 I <sup>2</sup> C 时序图	309
15-5 I <sup>2</sup> C Master 写 7-bit 寻址的 Slave	311
15-6 I <sup>2</sup> C Master 写 10-bit 寻址的 Slave	312
15-7 I <sup>2</sup> C Master 写 7-bit 寻址 Slave 的 M 地址 RAM	313
15-8 I <sup>2</sup> C Master 分段写 7-bit 寻址的 Slave	313
15-9 I <sup>2</sup> C Master 读 7-bit 寻址的 Slave	314
15-10 I <sup>2</sup> C Master 读 10-bit 寻址的 Slave	315
15-11 I <sup>2</sup> C Master 从 7-bit 寻址 Slave 的 M 地址读取 N 个数据	315
15-12 I <sup>2</sup> C Master 分段读 7-bit 寻址的 Slave	316
16-1 GCM 加密操作流程图	351
19-1 噪声源	383
20-1 片外存储器加解密模块架构	385
21-1 软件准备工作与硬件工作流程	396

# 1. 系统和存储器

## 1.1 概述

ESP32-S2 采用哈佛结构 Xtensa® LX7 CPU 构成单核系统。所有的内部存储器、外部存储器以及外设都分布在 CPU 的总线上。

## 1.2 主要特性

- **地址空间**
  - 数据总线与指令总线总共有 4 GB（32 位）地址空间
  - 464 KB 内部存储器指令地址空间
  - 400 KB 内部存储器数据地址空间
  - 1.77 MB 外设地址空间
  - 7.5 MB 外部存储器指令虚地址空间
  - 14.5 MB 外部存储器数据虚地址空间
  - 320 KB 内部 DMA 地址空间
  - 10.5 MB 外部 DMA 地址空间
- **内部存储器**
  - 128 KB Internal ROM
  - 320 KB Internal SRAM
  - 8 KB RTC FAST Memory
  - 8 KB RTC SLOW Memory
- **外部存储器**
  - 最大支持 1 GB 外部 SPI flash
  - 最大支持 1 GB 外部 SPI RAM
- **DMA**
  - 9 个具有 DMA 功能的模块 / 外设

图 1-1 描述了系统结构与地址映射结构。



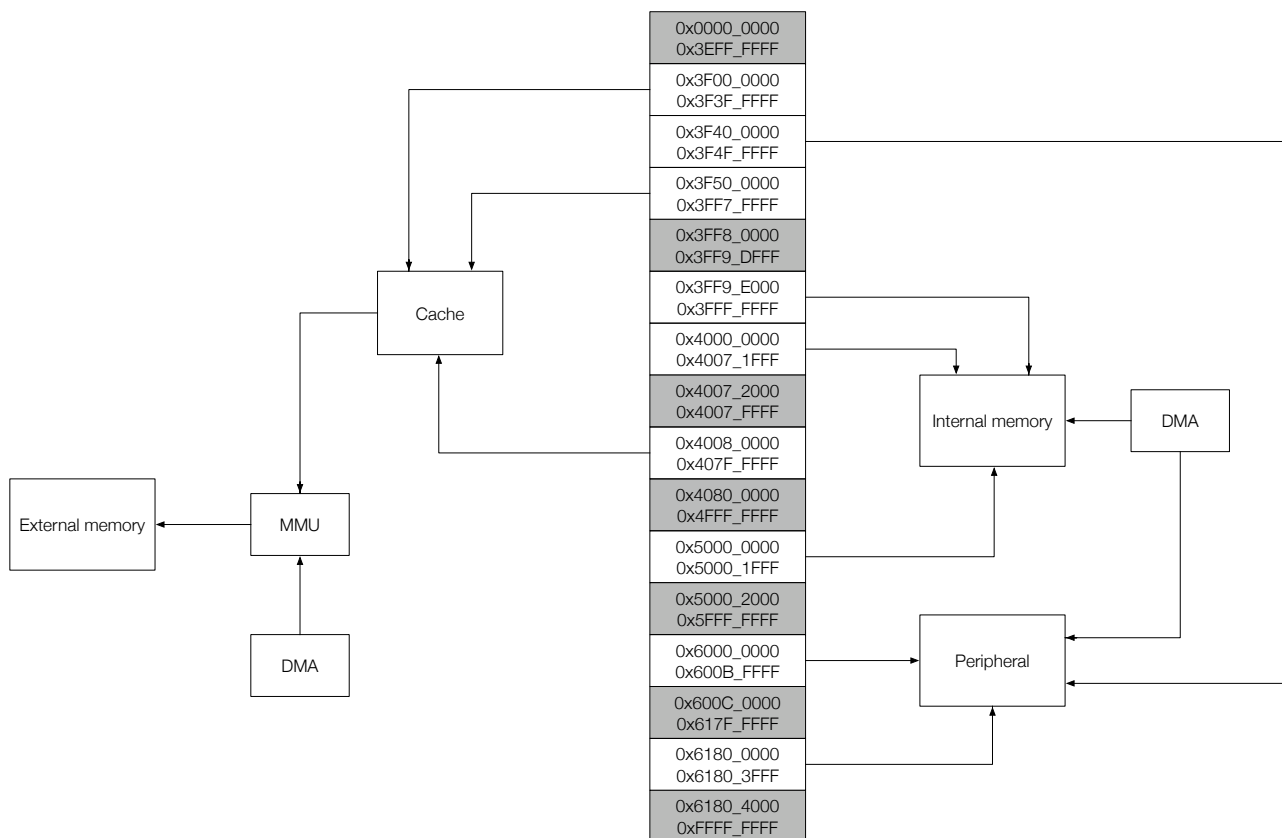


图 1-1. 系统结构与地址映射结构

**说明：**

- 图中灰色背景标注的地址空间不可用。
- 地址空间中可用的地址范围可能大于或小于实际可用的内存。

## 1.3 功能描述

### 1.3.1 地址映射

系统由一个哈佛结构 Xtensa® LX7 CPU 构成，CPU 具有 4 GB（32 位）的地址空间寻址能力。

地址 0x4000\_0000 以下的部分属于数据总线的地址范围，地址 0x4000\_0000 ~ 0x4FFF\_FFFF 部分为指令总线的地址范围，地址 0x5000\_0000 及以上的部分是数据总线与指令总线共用的地址范围。

CPU 的数据总线与指令总线都为小端序。CPU 可以通过数据总线进行字节、半字、字对齐的数据访问。CPU 也可以通过指令总线进行数据访问，但必须是字对齐方式；非对齐数据访问会导致 CPU 工作异常。

CPU 能够：

- 通过数据总线与指令总线直接访问内部存储器；
- 通过 cache 直接访问映射到地址空间的外部存储器；
- 通过数据总线直接访问模块 / 外设。

表 1 描述了 CPU 的数据总线与指令总线中的各段地址所能访问的目标。

系统中部分内部存储器与部分外部存储器既可以被数据总线访问也可以被指令总线访问，这种情况下，CPU 可以通过多个地址访问到同一目标。

表 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	0x3EFF_FFFF		保留
数据	0x3F00_0000	0x3F3F_FFFF	4 MB	外部存储器
	0x3F40_0000	0x3F4F_FFFF	1 MB	外设
	0x3F50_0000	0x3FF7_FFFF	10.5 MB	外部存储器
	0x3FF8_0000	0x3FF9_DFFF		保留
数据	0x3FF9_E000	0x3FFF_FFFF	392 KB	内部存储器
指令	0x4000_0000	0x4007_1FFF	456 KB	内部存储器
	0x4007_2000	0x4007_FFFF		保留
指令	0x4008_0000	0x407F_FFFF	7.5 MB	外部存储器
	0x4080_0000	0x4FFF_FFFF		保留
数据 / 指令	0x5000_0000	0x5000_1FFF	8 KB	内部存储器
	0x5000_2000	0x5FFF_FFFF		保留
数据 / 指令	0x6000_0000	0x600B_FFFF	768 KB	外设
	0x600C_0000	0x617F_FFFF		保留
数据 / 指令	0x6180_0000	0x6180_3FFF	16 KB	外设
	0x6180_4000	0xFFFF_FFFF		保留

### 1.3.2 内部存储器

内部存储器分为四个部分：Internal ROM (128 KB)、Internal SRAM (320 KB)、RTC FAST Memory (8 KB)、RTC SLOW Memory (8 KB)。

128 KB Internal ROM 分为 64 KB Internal ROM 0、64 KB Internal ROM 1 两部分。

320 KB Internal SRAM 分为 32 KB Internal SRAM 0、288 KB Internal SRAM 1 两部分。

RTC FAST Memory 与 RTC SLOW Memory 都为 SRAM。

表 2 列出了所有内部存储器以及访问内部存储器的数据总线与指令总线地址段。

表 2: 内部存储器地址映射

总线类型	边界地址		容量	目标	权限管理
	低位地址	高位地址			
数据	0x3FF9_E000	0x3FF9_FFFF	8 KB	RTC FAST Memory	YES
	0x3FFA_0000	0x3FFA_FFFF	64 KB	Internal ROM 1	NO
	0x3FFB_0000	0x3FFB_7FFF	32 KB	Internal SRAM 0	YES
	0x3FFB_8000	0x3FFF_FFFF	288 KB	Internal SRAM 1	YES

总线类型	边界地址		容量	目标	权限管理
	低位地址	高位地址			
指令	0x4000_0000	0x4000_FFFF	64 KB	Internal ROM 0	NO
	0x4001_0000	0x4001_FFFF	64 KB	Internal ROM 1	NO
	0x4002_0000	0x4002_7FFF	32 KB	Internal SRAM 0	YES
	0x4002_8000	0x4006_FFFF	288 KB	Internal SRAM 1	YES
	0x4007_0000	0x4007_1FFF	8 KB	RTC FAST Memory	YES
总线类型	边界地址		容量	目标	权限管理
	低位地址	高位地址			
数据 / 指令	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	YES

**说明：**

表 2 中的“权限管理”一栏中标注为 YES，指总线在访问目标存储器时需要获取权限。用户可以配置权限控制寄存器对指令 / 数据总线访问目标存储器进行限制。

### 1.3.2.1 Internal ROM 0

Internal ROM 0 的容量为 64 KB，只读。如表 2 所示，CPU 只可以通过指令总线访问这部分存储器。

### 1.3.2.2 Internal ROM 1

Internal ROM 1 的容量为 64 KB，只读。如表 2 所示，CPU 可以通过数据或指令总线进行访问。

这两段地址同序访问 Internal ROM 1，即地址 0x3FFA\_0000 与 0x4001\_0000 访问到相同的字，0x3FFA\_0004 与 0x4001\_0004 访问到相同的字，0x3FFA\_0008 与 0x4001\_0008 访问到相同的字，以此类推。

### 1.3.2.3 Internal SRAM 0

Internal SRAM 0 的容量为 32 KB，可读可写。如表 2 所示，CPU 可以通过数据或指令总线同序访问。

通过配置，这部分存储器中的 8 KB、16 KB、24 KB 或全部 32 KB 可以被 cache 占用，用来缓存外部存储器的数据。被 cache 占用的部分不可以被 CPU 访问，未被 cache 占用的部分仍然可以被 CPU 访问。

### 1.3.2.4 Internal SRAM 1

Internal SRAM 1 容量为 288 KB，可读可写。如表 2 所示，CPU 可以通过数据或指令总线同序访问。

Internal SRAM 1 存储器的 288 KB 地址空间由 18 个容量为 16 KB 的小存储器（子存储器）组成。其中至多有 1 个可以作为 Trace Memory 用于 CPU 的 Trace 功能，但 Trace Memory 无法被 CPU 访问。

### 1.3.2.5 RTC FAST Memory

RTC FAST Memory 为 8 KB SRAM，可读可写。如表 2 所示，CPU 可以通过数据或指令总线同序访问。

### 1.3.2.6 RTC SLOW Memory

RTC SLOW Memory 为 8 KB SRAM，可读可写。如表 2 所示，CPU 可以通过数据 / 指令总线的共用地址段访问这部分存储器。

RTC SLOW Memory 也可以被当作一个外设来使用，此时 CPU 可以通过数据总线的地址段 0x3F42\_1000 ~ 0x3F42\_2FFF 或地址段 0x6002\_1000 ~ 0x6002\_2FFF 来访问它。

## 1.3.3 外部存储器

ESP32-S2 支持多个外部 QSPI / OSPI flash 和随机存储器 (RAM)。ESP32-S2 还支持基于 XTS-AES 算法的硬件加解密功能，从而保护开发者 flash 和片外 RAM 中的程序和数据。

### 1.3.3.1 外部存储器地址映射

CPU 通过高速缓存 (cache) 来访问外部存储器。Cache 将根据 MMU 中的信息把 CPU 指令总线或数据总线的地址变换为访问外部 flash 与片外 RAM 的实地址。经过变换的实地址所组成的实地址空间最大支持 1 GB 的外部 flash 与 1 GB 的片外 RAM。

通过高速缓存，ESP32-S2 一次最多可以同时有：

- 7.5 MB 的指令总线地址空间，通过指令缓存 (ICache) 以 64 KB 为单位映射到 flash 或片外 RAM，支持字节、半字、字对齐的读访问。
- 4 MB 的只读数据总线地址空间，通过指令缓存 (ICache) 以 64 KB 为单位映射到 flash 或片外 RAM，支持字节、半字、字对齐的读访问。
- 10.5 MB 的数据总线地址空间，通过数据缓存 (DCache) 以 64 KB 为单位映射到片外 RAM，支持字节、半字、字对齐的读写访问。这部分地址空间也可以用作只读数据空间，映射到 flash 与片外 RAM。

表 3 列出了在访问外部存储器时 CPU 的数据总线与指令总线与 cache 的对应关系。

表 3: 外部存储器地址映射

总线类型	边界地址		容量	目标	权限管理
	低位地址	高位地址			
数据	0x3F00_0000	0x3F3F_FFFF	4 MB	ICache	YES
数据	0x3F50_0000	0x3FF7_FFFF	10.5 MB	DCache	YES
指令	0x4008_0000	0x407F_FFFF	7.5 MB	ICache	YES

**说明：**

表 3 中的“权限管理”一栏中标注为 YES，指总线在访问目标存储器时需要获取权限。用户可以配置权限控制寄存器对指令 / 数据总线访问目标存储器进行限制。

### 1.3.3.2 高速缓存

如图 1-2 所示，ESP32-S2 cache 采用分立结构，以便当 CPU 的指令总线和数据总线同时发起请求时，也可以迅速响应。Cache 的存储空间与内部存储空间复用（详见章节 1.3.2.3）。当 cache 缺失时，cache 控制器会向外部存储器发起请求，当 ICache 和 DCache 同时发起外部存储器请求时，由仲裁器决定谁先获得外部存储器的使用权。ICache 和 DCache 的缓存大小可配置为 8 KB 和 16 KB，块大小可配置为 16 B 和 32 B。

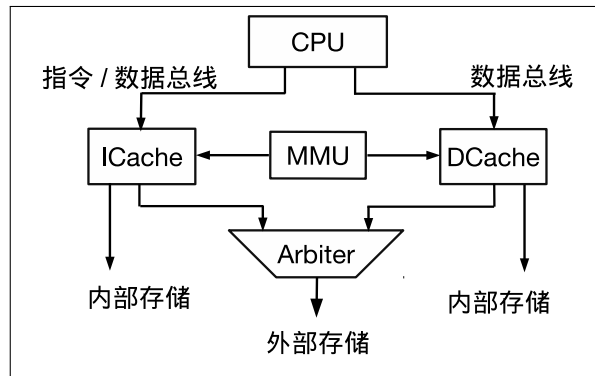


图 1-2. Cache 系统框图

### 1.3.3.3 Cache 操作

ESP32-S2 cache 共有如下几种操作：

1. **失效 (Invalidate)**：清除 Tag 的有效标志位 (Valid bit)。如果 CPU 接着去访问该数据，那么需要访问外部存储器。失效分为自动失效和手动失效。手动失效仅对 cache 中落入指定区域的地址对应的数据做失效处理，而自动失效会对 cache 中的所有数据做失效处理。ICache 和 DCache 均具有此功能。
2. **清除 (Clean)**：清除 Tag 的脏标志位 (Dirty bit)，保留有效标志位。如果 CPU 接着去访问该数据，那么可以直接从 cache 中访问到。只有 DCache 具有此功能。
3. **写回 (Write-back)**：清除 Tag 的脏块标志位，保留有效标志位，并强制将对应地址的数据写回到外部存储器。如果 CPU 接着去访问该数据，那么可以直接从 cache 中访问到。只有 DCache 具有此功能。
4. **预取 (Preload)**：Preload 功能用于将指令和数据提前加载到 cache 中。预取操作的最小单位为 1 个块。预取分为手动预取和自动预取，手动预取是指硬件按软件指定的虚地址预取一段连续的数据；自动预取是指硬件根据当前命中 / 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)**：锁定分为预锁定和手动锁定，预锁定开启时，cache 在填充缺失数据到 cache memory 时将落在指定区域的数据锁定，手动锁定开启时，cache 检查已填充到 cache memory 的数据，并将落在指定区域的数据锁定。锁定区域的数据会一直保存在 cache 中，而不会被替换掉。但当所有路都被锁定时，cache 将进行正常替换，就像 cache 没有被锁定一样。解锁是锁定的逆操作。Cache 的手动失效操作、清除和写回操作均需要在解锁后才可使用。

### 1.3.4 DMA 地址空间

ESP32-S2 可以借助直接存储访问 (direct memory access, DMA) 完成：

- 模块 / 外设与内部存储器之间的数据搬运；

- 内部存储器与内部存储器之间的数据搬运；
- 模块 / 外设与外部存储器之间的数据搬运；
- 内部存储器与外部存储器之间的数据搬运。

DMA 可以通过与 CPU 数据总线完全相同的地址访问 Internal SRAM 0 与 Internal SRAM 1，即使用地址 0x3FFB\_0000 ~ 0x3FFB\_7FFF 访问 Internal SRAM 0，使用地址 0x3FFB\_8000 ~ 0x3FFF\_FFFF 访问 Internal SRAM 1。但 DMA 无法访问被 cache 占用的内部存储器。

另外，DMA 可以使用与 CPU 访问 DCache 相同的地址 (0x3F50\_0000 ~ 0x3FF7\_FFFF) 来访问外部存储器，但只可以访问片外 RAM。当 DCache 与 DMA 同时访问外部存储器时，务必要注意数据一致性问题。

系统中具有 DMA 功能的模块 / 外设总共有 9 个，如表 4 所示。其中，有些模块 / 外设通过 DMA 只可以访问内部存储器，有些既可以访问内部存储器又可以访问外部存储器。

表 4: 具有 DMA 功能的模块 / 外设

UART0	UART1
SPI2	SPI3
I2S0	
ADC Controller	
Copy DMA	
AES Accelerator	SHA Accelerator

### 1.3.5 模块 / 外设地址空间

CPU 可以通过数据总线 0x3F40\_0000 ~ 0x3F4F\_FFFF、数据 / 指令总线的共用地址段 0x6000\_0000 ~ 0x600B\_FFFF 和 0x6180\_0000 ~ 0x6180\_3FFF 来访问模块 / 外设。

#### 1.3.5.1 外设总线名称约定

为了后续描述的方便，作如下约定：

- **PeriBus1**：总线地址段 0x3F40\_0000 ~ 0x3F4F\_FFFF 被记作 “PeriBus1 总线”，简记为 “**PeriBus1**”，它的基地址为 0x3F40\_0000。
- **PeriBus2**：总线地址段 0x6000\_0000 ~ 0x600B\_FFFF 和 0x6180\_0000 ~ 0x6180\_3FFF 整体被记作 “PeriBus2 总线”，简记为 “**PeriBus2**”，它的基地址为 0x6000\_0000。

在后续章节中出现的所有 “**PeriBus1**” 和 “**PeriBus2**” 都指代对应的总线地址段。

#### 1.3.5.2 外设总线区别

相比于 CPU 通过 **PeriBus2** 访问模块 / 外设，CPU 通过 **PeriBus1** 访问模块 / 外设效率更高。但是 **PeriBus1** 有预测性读 (speculative read) 的特点，不能保证每一次的读访问都是真实有效的，因此，在访问模块 / 外设中的某些特殊寄存器如 FIFO 寄存器时，必须使用 **PeriBus2** 进行访问。

另外，PeriBus1 会打乱总线上的读写操作的先后顺序以提升性能，这可能会导致对读写操作的先后顺序有严格要求的程序发生崩溃，对于这种情况，请在程序语句之前增加 volatile，也可以改用 PeriBus2 进行访问。

### 1.3.5.3 模块 / 外设地址空间列表

表 5 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。请注意，“边界地址”一栏中的数值是相对于总线基地址的地址偏移量，而非绝对地址。访问某一模块 / 外设的绝对地址等于总线基地址加上相应的地址偏移量。

表 5: 模块 / 外设地址空间映射表

目标	边界地址		Size	说明
	低位地址	高位地址		
UART0	0x0000_0000	0x0000_0FFF	4 KB	1, 2, 3
保留	0x0000_1000	0x0000_1FFF		
SPI1	0x0000_2000	0x0000_2FFF	4 KB	1, 2
SPI0	0x0000_3000	0x0000_3FFF	4 KB	1, 2
GPIO	0x0000_4000	0x0000_4FFF	4 KB	1, 2
保留	0x0000_5000	0x0000_6FFF		
TIMER	0x0000_7000	0x0000_7FFF	4 KB	1, 2
RTC	0x0000_8000	0x0000_8FFF	4 KB	1, 2
IO MUX	0x0000_9000	0x0000_9FFF	4 KB	1, 2
保留	0x0000_A000	0x0000_EFFF		
I2S0	0x0000_F000	0x0000_FFFF	4 KB	1, 2, 3
UART1	0x0001_0000	0x0001_0FFF	4 KB	1, 2, 3
保留	0x0001_1000	0x0001_2FFF		
I2C0	0x0001_3000	0x0001_3FFF	4 KB	1, 2, 3
UHCIO	0x0001_4000	0x0001_4FFF	4 KB	1, 2
保留	0x0001_5000	0x0001_5FFF		
RMT	0x0001_6000	0x0001_6FFF	4 KB	1, 2, 3
PCNT	0x0001_7000	0x0001_7FFF	4 KB	1, 2
保留	0x0001_8000	0x0001_8FFF		
LED PWM Controller	0x0001_9000	0x0001_9FFF	4 KB	1, 2
eFuse Controller	0x0001_A000	0x0001_AFFF	4 KB	1, 2
保留	0x0001_B000	0x0001_EFFF		
Timer Group 0	0x0001_F000	0x0001_FFFF	4 KB	1, 2
Timer Group 1	0x0002_0000	0x0002_0FFF	4 KB	1, 2
RTC SLOW Memory	0x0002_1000	0x0002_2FFF	8 KB	1, 2, 3
System Timer	0x0002_3000	0x0002_3FFF	4 KB	1, 2
SPI2	0x0002_4000	0x0002_4FFF	4 KB	1, 2
SPI3	0x0002_5000	0x0002_5FFF	4 KB	1, 2
APB Controller	0x0002_6000	0x0002_6FFF	4 KB	1, 2
I2C1	0x0002_7000	0x0002_7FFF	4 KB	1, 2, 3
保留	0x0002_8000	0x0002_AFFF		
TWAI Controller	0x0002_B000	0x0002_BFFF	4 KB	1, 2



目标	边界地址		Size	说明
	低位地址	高位地址		
保留	0x0002_C000	0x0003_8FFF		
USB OTG	0x0003_9000	0x0003_9FFF	4 KB	1, 2, 3, 4
AES Accelerator	0x0003_A000	0x0003_AFFF	4 KB	1, 2
SHA Accelerator	0x0003_B000	0x0003_BFFF	4 KB	1, 2
RSA Accelerator	0x0003_C000	0x0003_CFFF	4 KB	1, 2
Digital Signature	0x0003_D000	0x0003_DFFF	4 KB	1, 2
HMAC	0x0003_E000	0x0003_EFFF	4 KB	1, 2
Crypto DMA	0x0003_F000	0x0003_FFFF	4 KB	1, 2
保留	0x0004_4000	0x000C_DFFF		
ADC Controller	0x0004_0000	0x0004_0FFF	4 KB	1, 2
保留	0x0004_1000	0x0007_FFFF		
USB	0x0008_0000	0x000B_FFFF	256 KB	1, 2, 3, 4
System Registers	0x000C_0000	0x000C_0FFF	4 KB	1
Sensitive Register	0x000C_1000	0x000C_1FFF	4 KB	1
Interrupt Matrix	0x000C_2000	0x000C_2FFF	4 KB	1
Copy DMA	0x000C_3000	0x000C_3FFF	4 KB	1
保留	0x000C_4000	0x000C_EFFF		
Dedicated GPIO	0x000C_F000	0x000C_FFFF	4 KB	1
保留	0x000D_1000	0x000F_FFFF		
Configure Cache	0x0180_0000	0x0180_3FFF	16 KB	2

**说明:**

1. 该模块 / 外设可以被 [PeriBus1](#) 总线访问。
2. 该模块 / 外设可以被 [PeriBus2](#) 总线访问。
3. 当 [PeriBus1](#) 总线访问该模块 / 外设时，不能读访问一些特殊的地址（详见 [1.3.5.4](#) 章节）。
4. 该模块 / 外设的地址空间不连续。

### 1.3.5.4 PeriBus1 访问受限地址列表

如前文 [1.3.5.2](#) 所述，[PeriBus1](#) 有预测性读 (speculative read) 的特点，因此被禁止读访问模块 / 外设的 FIFO 寄存器。表 6 列出了禁止被 [PeriBus1](#) 读访问的地址。另外，4 个用户自定义寄存器可以根据用户需求进行配置，用于将更多地址添加到访问受限地址列表中。

表 6: 访问受限的地址

目标外设	受限地址值 / 区间
UART0	0x3F40_0000
UART1	0x3F41_0000
I2S0	0x3F40_F004
RMT	0x3F41_6000 ~ 0x3F41_600F



目标外设	受限地址值 / 区间
I2C0	0x3F41_301C
I2C1	0x3F42_701C
USB OTG	0x3F48_0020, 0x3F48_1000 ~ 0x3F49_0FFF

## 2. 复位和时钟

### 2.1 复位

#### 2.1.1 概述

系统提供四种级别的复位方式，分别是 CPU 复位、内核复位、系统复位和芯片复位。

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 2-1 展示了整个芯片系统的结构以及四种复位等级。

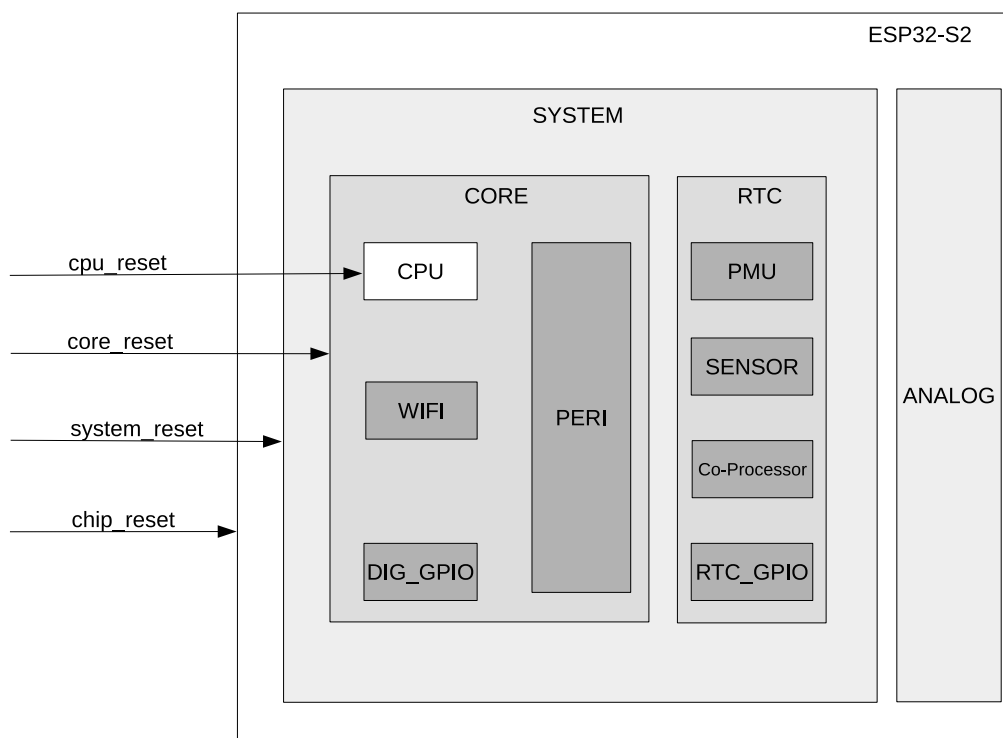


图 2-1. 四种复位等级

- CPU 复位：只复位 CPU core，复位释放后，程序将从 CPU reset vector 开始执行；
- 内核复位：复位除 RTC 以外的其他数字系统，包括 CPU、外设、Wi-Fi 及数字 GPIO；
- 系统复位：复位包括 RTC 在内的整个数字系统；
- 芯片复位：复位整个芯片。

#### 2.1.2 复位源

上述任一复位源产生时，CPU 均将立刻复位。复位释放后，CPU 可以通过读取寄存器 RTC\_CNTL\_RESET\_CAUSE\_PROCPU 来获取复位源。

表 7 列出了从这一寄存器中可能读出的复位源。

表 7: 复位源

编码	复位源	复位方式	注释
0x01	芯片复位	芯片复位	见表下方芯片复位的触发源
0x0F	欠压系统复位	系统复位	欠压检测器触发的系统复位
0x10	RWDT 系统复位	系统复位	详见章节 13 看门狗定时器
0x13	GLITCH 复位	系统复位	-
0x03	软件系统复位	内核复位	配置 RTC_CNTL_SW_SYS_RST 寄存器触发
0x05	Deep-sleep 复位	内核复位	-
0x07	MWDT0 全局复位	内核复位	详见章节 13 看门狗定时器
0x08	MWDT1 全局复位	内核复位	详见章节 13 看门狗定时器
0x09	RWDT 内核复位	内核复位	详见章节 13 看门狗定时器
0x0B	MWDT0 CPU 复位	CPU 复位	详见章节 13 看门狗定时器
0x0C	软件 CPU 复位	CPU 复位	配置 RTC_CNTL_SW_PROCPU_RST 寄存器触发
0x0D	RWDT CPU 复位	CPU 复位	详见章节 13 看门狗定时器
0x11	MWDT1 CPU 复位	CPU 复位	详见章节 13 看门狗定时器

注意：

- 芯片复位的触发源包括以下三项：
  - 芯片上电
  - 欠压检测器触发的芯片复位
  - SWD 触发的芯片复位
- 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。

## 2.2 系统时钟

### 2.2.1 概述

ESP32-S2 提供了多种不同频率的时钟选择，可以灵活配置 CPU、外设、以及 RTC 的工作频率，以满足不同功耗和性能需求。下图 2-2 为系统时钟结构。

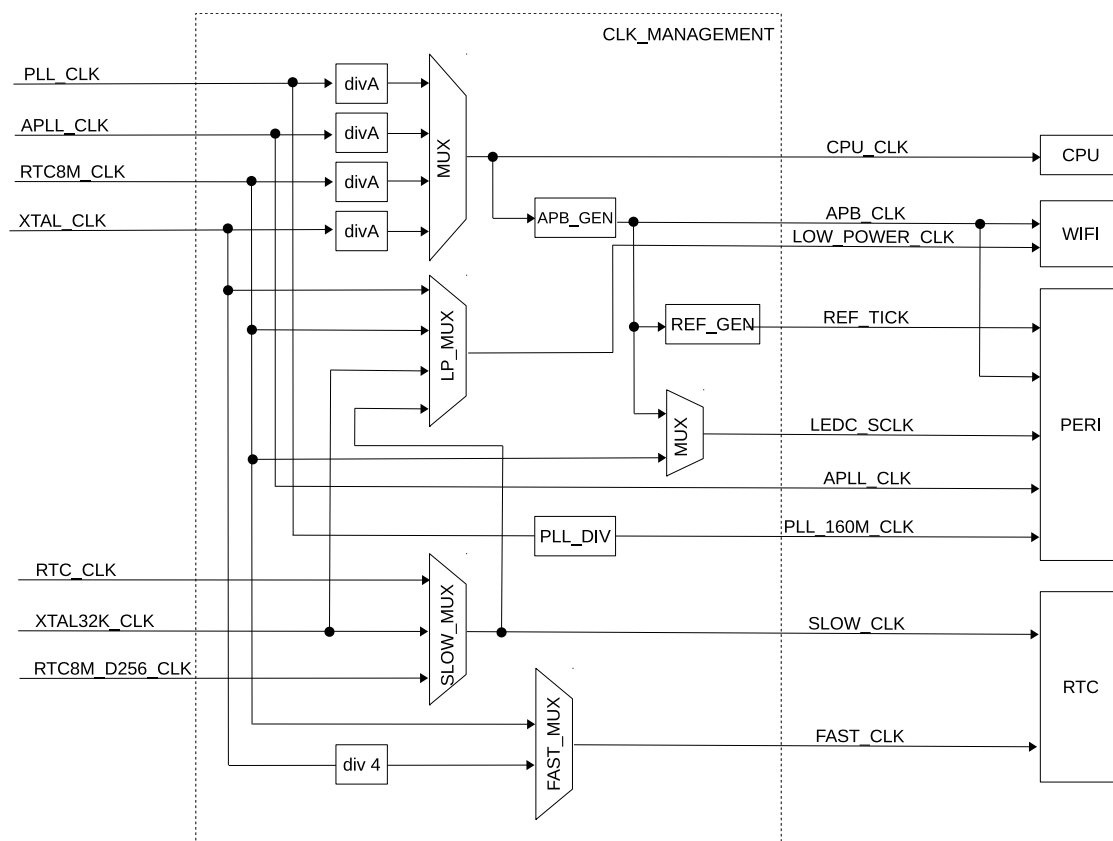


图 2-2. 系统时钟

### 2.2.2 时钟源

ESP32-S2 有三种时钟源：外部晶振、内部 PLL 和震荡电路，用于生成多种时钟。这些时钟根据频率不同可分为以下三种类型。

- 快速时钟：供 CPU 及数字外设等高速设备使用
  - PLL\_CLK：320 MHz 或 480 MHz 内部 PLL 时钟；
  - XTAL\_CLK：40 MHz 外部晶振时钟。
- 低功耗慢速时钟：供低功耗设备使用，包括功耗管理单元以及低功耗外设等
  - XTAL32K\_CLK：32 kHz 外部晶振时钟；
  - RTC8M\_CLK：8 MHz 内部时钟，频率可调；
  - RTC8M\_D256\_CLK：由 RTC8M\_CLK 经 256 分频所得，频率为  $\text{RTC8M\_CLK} / 256$ 。当 RTC8M\_CLK 的初始频率为 8 MHz 时，该时钟以 31.250 kHz 的频率运行；
  - RTC\_CLK：150 kHz 内部低功耗时钟，频率可调。
- 音频时钟：供音频相关设备使用
  - APLL\_CLK：16 MHz ~ 128 MHz 内部 Audio PLL 时钟。

### 2.2.3 CPU 时钟

如图 2-2 所示，CPU\_CLK 为 CPU 主时钟，CPU 在最高效工作模式下，主频可以达到 240 MHz。同时，CPU 能够在超低频下工作（通常为 2 MHz），以减少功耗。

CPU\_CLK 由 `SYSTEM_SOC_CLK_SEL` 来选择时钟源，允许选择 PLL\_CLK、APLL\_CLK、RTC8M\_CLK 或 XTAL\_CLK 作为 CPU\_CLK 的时钟源。具体请参考表 8 和表 9。

表 8: CPU\_CLK 源

SYSTEM_SOC_CLK_SEL 值	时钟源
0	XTAL_CLK
1	PLL_CLK
2	RTC8M_CLK
3	APLL_CLK

表 9: CPU\_CLK 选择

时钟源	SEL_0*	SEL_1*	SEL_2*	CPU 时钟频率
XTAL_CLK	0	-	-	$CPU\_CLK = XTAL\_CLK / (SYSTEM\_PRE\_DIV\_CNT + 1)$ SYSTEM_PRE_DIV_CNT 默认值为 1，范围 0 ~ 1023。
PLL_CLK (480 MHz)	1	1	0	$CPU\_CLK = PLL\_CLK / 6$ CPU_CLK 频率为 80 MHz。
PLL_CLK (480 MHz)	1	1	1	$CPU\_CLK = PLL\_CLK / 3$ CPU_CLK 频率为 160 MHz。
PLL_CLK (480 MHz)	1	1	2	$CPU\_CLK = PLL\_CLK / 2$ CPU_CLK 频率为 240 MHz。
PLL_CLK (320 MHz)	1	0	0	$CPU\_CLK = PLL\_CLK / 4$ CPU_CLK 频率为 80 MHz。
PLL_CLK (320 MHz)	1	0	1	$CPU\_CLK = PLL\_CLK / 2$ CPU_CLK 频率为 160 MHz。
RTC8M_CLK	2	-	-	$CPU\_CLK = RTC8M\_CLK / (SYSTEM\_PRE\_DIV\_CNT + 1)$ SYSTEM_PRE_DIV_CNT 默认值为 1，范围 0 ~ 1023。
APLL_CLK	3	0	0	$CPU\_CLK = APLL\_CLK / 4$
APLL_CLK	3	0	1	$CPU\_CLK = APLL\_CLK / 2$

\*SEL\_0: 寄存器 `SYSTEM_SOC_CLK_SEL` 的值；

\*SEL\_1: 寄存器 `SYSTEM_PLL_FREQ_SEL` 的值；

\*SEL\_2: 寄存器 `SYSTEM_CPUPERIOD_SEL` 的值。

注意：

- 当 CPU 选择 XTAL\_CLK 用作时钟源，通过配置寄存器 `SYSTEM_PRE_DIV_CNT` 调节 XTAL\_CLK 的时钟分频系数时，需要遵循以下规则：
  - 目标分频系数为 x 分频（x 不等于 1 分频）且当前分频系数为 2 分频时（`SYSTEM_PRE_DIV_CNT = 1`），需要先将分频系数配置为 1 分频（`SYSTEM_PRE_DIV_CNT = 0`），再调节分频系数为 x 分频（`SYSTEM_PRE_DIV_CNT = x - 1`）；
  - 目标分频系数为 2 分频且当前分频系数为 x 分频（`SYSTEM_PRE_DIV_CNT = x - 1`），需要先将分频

系数调节为 1 分频 ( $\text{SYSTEM\_PRE\_DIV\_CNT} = 0$ )，再调节分频系数为 2 分频 ( $\text{SYSTEM\_PRE\_DIV\_CNT} = 1$ )；

- 如需调至其它目标分频系数  $x$ ，直接调整分频系数即可 ( $\text{SYSTEM\_PRE\_DIV\_CNT} = x - 1$ )。

## 2.2.4 外设时钟

外设所需要的时钟包括 APB\_CLK、REF\_TICK、LEDC\_PWM\_CLK、APLL\_CLK 和 PLL\_160M\_CLK。下表 10 为接入各个外设的时钟。

表 10: 外设时钟用法

外设	APB_CLK	REF_TICK	LEDC_PWM_CLK	APLL_CLK	PLL_160M_CLK
TIMG	Y	Y			
I <sup>2</sup> S	Y			Y	Y
UHCI	Y				
UART	Y	Y			
RMT	Y	Y			
LED_PWM	Y	Y	Y		
I <sup>2</sup> C	Y	Y			
SPI	Y				
PCNT	Y				
eFuse Controller	Y				
SARADC/DAC	Y			Y	
USB	Y				
CRYPTO					Y
TWAI Controller	Y				
System Timer	Y				

### 2.2.4.1 APB\_CLK 源

如表 11 所示，APB\_CLK 的频率由 CPU\_CLK 的时钟源决定。

表 11: APB\_CLK 源

CPU_CLK 源	APB_CLK 频率
PLL_CLK	80 MHz
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

### 2.2.4.2 REF\_TICK 源

REF\_TICK 由 XTAL\_CLK 或 RTC8M\_CLK 分频产生。当 CPU 时钟源为 PLL\_CLK、APLL\_CLK、XTAL\_CLK 时，REF\_TICK 由 XTAL\_CLK 分频获得；当 CPU 时钟源为内部 RTC8M\_CLK 时，REF\_TICK 由内部 RTC8M\_CLK 时

钟分频获得。由此可以保证 REF\_TICK 在 APB\_CLK 切换时维持频率不变。寄存器配置如表 12 所示：

表 12: REF\_TICK 源

CPU_CLK 源	时钟分频寄存器
PLL_CLK   XTAL_CLK   APLL_CLK	APB_CTRL_XTAL_TICK_NUM
RTC8M_CLK	APB_CTRL_CK8M_TICK_NUM

通常 REF\_TICK 的周期为  $1\ \mu s$ ，所以 APB\_CTRL\_XTAL\_TICK\_NUM 需配置为 39；  
APB\_CTRL\_CK8M\_TICK\_NUM 需配置为 7。

### 2.2.4.3 LEDC\_PWM\_CLK 源

LEDC\_PWM\_CLK 时钟源由寄存器 LEDC\_APB\_CLK\_SEL 决定，如表 13 所示。

表 13: LEDC\_PWM\_CLK 源

LEDC_APB_CLK_SEL 值	LEDC_PWM_CLK 源
0 (默认)	-
1	APB_CLK
2	RTC8M_CLK
3	XTAL_CLK

### 2.2.4.4 APLL\_SCLK 源

APLL\_CLK 来自内部 PLL\_CLK，其输出频率通过使用 APLL 配置寄存器来配置。配置方式见第 2.2.7 节。

### 2.2.4.5 PLL\_160M\_CLK 源

PLL\_160M\_CLK 是 PLL\_CLK 根据当前 PLL 的频率分频所得。

### 2.2.4.6 时钟源注意事项

需要与其他时钟配合工作的外设（如 RMT、I<sup>2</sup>C 等）一般在选择 PLL\_CLK 时钟源的情况下工作。若频率发生变化，外设需要通过修改配置才能以同样的频率工作。接入 REF\_TICK 的外设允许在切换时钟源的情况下，不修改外设配置即可工作。详情请参考表 10。

LED 模块能将 RTC8M\_CLK 作为时钟源使用，即在 APB\_CLK 关闭的时候，LED 也可工作。换言之，当系统处于低功耗模式时，其他外设都将停止工作（APB\_CLK 关闭），但是 LED 仍然可以通过 RTC8M\_CLK 来正常工作。

### 2.2.5 Wi-Fi 时钟

Wi-Fi 必须在 APB\_CLK 时钟源选择 PLL\_CLK 下才能工作。只有当 Wi-Fi 进入低功耗模式时，才能暂时关闭 PLL\_CLK。

LOW\_POWER\_CLK 允许选择 XTAL32K\_CLK、XTAL\_CLK、RTC8M\_CLK、SLOW\_CLK（RTC 当前所选的慢速时钟）用于 Wi-Fi 的低功耗模式。

### 2.2.6 RTC 时钟

SLOW\_CLK 和 FAST\_CLK 的时钟源为低频时钟。RTC 模块能够在大多数时钟源关闭的状态下工作。

SLOW\_CLK 允许选择 RTC\_CLK、XTAL32K\_CLK 或 RTC8M\_D256\_CLK，用于驱动功耗模块。

FAST\_CLK 允许选择 XTAL\_CLK 的分频时钟或 RTC8M\_CLK，用于驱动片上传感器模块。

### 2.2.7 音频 PLL 时钟

音频应用和其他对于数据传输时效性要求很高的应用都需要高度可配置、低抖动并且精确的时钟源。来自系统时钟的时钟源可能会携带抖动，并且不支持高精度的时钟频率配置。

在满足应用需求的前提下，为了最大限度地降低系统成本，ESP32-S2 集成了专门用于 I<sup>2</sup>S 外设的音频 PLL。用户可使用 APLL 时钟对 I<sup>2</sup>S 模块进行计时。

Audio PLL 公式如下：

$$f_{\text{out}} = \frac{f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4)}{2(\text{odiv} + 2)}$$

其中，

- $f_{\text{xtal}}$ ：晶振频率，通常为 40 MHz
- sdm0：可配参数 0 ~ 255
- sdm1：可配参数 0 ~ 255
- sdm2：可配参数 0 ~ 63
- odiv：可配参数 0 ~ 31
- 公式的分子频率工作范围在 350 MHz ~ 500 MHz

$$350\text{MHz} < f_{\text{xtal}}(\text{sdm2} + \frac{\text{sdm1}}{2^8} + \frac{\text{sdm0}}{2^{16}} + 4) < 500\text{MHz}$$

Audio PLL 可通过使能寄存器 RTC\_CNTL\_PLLA\_FORCE\_PU 强行打开，或者使能寄存器 RTC\_CNTL\_PLLA\_FORCE\_PD 强行关闭，关闭优先级大于打开优先级。当 RTC\_CNTL\_PLLA\_FORCE\_PU 和 RTC\_CNTL\_PLLA\_FORCE\_PD 同时为 0 时，PLL 会跟随系统状态，当系统进入睡眠模式时自动关闭，系统被唤醒时自动打开。



## 3. 芯片 Boot 控制

### 3.1 概述

ESP32-S2 共有三个 Strapping 管脚：

- GPIO0
- GPIO45
- GPIO46

软件可以从 GPIO\_STRAPPING 寄存器中读取这三个 Strapping 管脚的值。在芯片复位过程中，包括上电复位、欠压复位和模拟超级看门狗复位（请参考章节 2 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关闭。

GPIO0、GPIO45 和 GPIO46 默认连接内部上拉/下拉。如果这些管脚没有外部连接或者连接的外部线路处于高阻抗状态，内部弱上拉/下拉将决定这几个管脚输入电平的默认值，如表 14 所示。

表 14: Strapping 管脚默认上拉/下拉

管脚	GPIO0	GPIO45	GPIO46
默认值	上拉	下拉	下拉

如需改变 Strapping 管脚的值，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-S2 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。

### 3.2 Boot 控制

复位释放后，GPIO0 和 GPIO46 共同控制 Boot 模式。

表 15: 系统启动模式

管脚	SPI Boot 模式	Download Boot 模式
GPIO0	1	0
GPIO46	x	0

表 15 列出了 GPIO0 和 GPIO46 的 Strapping 值及其对应的系统启动模式。此处“x”表示该项为无关项。ESP32-S2 芯片当前仅支持 SPI Boot 模式和 Download Boot 模式。GPIO0、GPIO46 组合为 (0, 1) 不可使用。

在 SPI Boot 模式下，CPU 通过从 SPI Flash 中读取程序来启动系统；在 Download Boot 模式下，用户可以通过 UART0、UART1、QPI 或 USB 接口将代码下载到 SRAM 或 Flash 中，或者将程序加载到 SRAM 中并在 Download Boot 模式下执行程序。

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），软件可通过配置 RTC\_FORCE\_DOWNLOAD\_BOOT 寄存器，触发 CPU 复位，将芯片启动模式强制从 SPI Boot 模式切换

至 Download Boot 模式；如果此 eFuse 设置为 1，则禁用 RTC\_FORCE\_DOWNLOAD\_BOOT 寄存器。

- **EFUSE\_DIS\_DOWNLOAD\_MODE**：如果此 eFuse 设置为 1，则关闭 Download Boot 模式。
- **EFUSE\_ENABLE\_SECURITY\_DOWNLOAD**：如果此 eFuse 设置为 1，则在 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 Download Boot 模式，请忽略此 eFuse。

### 3.3 ROM Code 打印

在系统启动过程中，GPIO46 与 UART\_PRINT\_CONTROL 一起控制 ROM Code 打印。

表 16: ROM Code 打印控制

UART_PRINT_CONTROL	GPIO46	ROM Code 打印
0	-	ROM Code 打印至 UART，此时未使用 GPIO46
1	0	使能打印
	1	关闭打印
2	0	关闭打印
	1	使能打印
3	-	系统启动过程中始终关闭打印，此时未使用 GPIO46

ROM Code 上电打印默认通过 U0TXD 管脚，可以由 UART\_PRINT\_CHANNEL (0: UART0; 1: DAC\_1) 控制切换到 DAC\_1 管脚。

### 3.4 VDD\_SPI 电压

芯片复位时，GPIO45 可用于选择 VDD\_SPI 电压：

- GPIO45 = 0 时，VDD\_SPI 由 VDD3P3\_RTC\_IO 通过电阻  $R_{SPI}$  后供电（电压典型值为 3.3 V）；
- GPIO45 = 1 时，VDD\_SPI 可选择由内置 LDO 供电（电压为 1.8 V）。

eFuse 中 VDD\_SPI\_FORC 设置为 1 时，可关闭上述功能。此时 VDD\_SPI 电压由 VDD\_SPI\_TIEH 的值决定：

- VDD\_SPI\_FORCE = 1 且 VDD\_SPI\_TIEH = 0 时，VDD\_SPI 连接 1.8 V LDO；
- VDD\_SPI\_FORCE = 1 且 VDD\_SPI\_TIEH = 1 时，VDD\_SPI 连接 VDD3P3\_RTC\_IO。

## 4. 中断矩阵

### 4.1 概述

ESP32-S2 中断矩阵 (Interrupt Matrix) 可以将任一外部中断源单独分配至 CPU 任一外部中断，以便在外设中断信号产生后，及时通知 CPU 进行处理。这一功能灵活强，可以满足不同的应用需求。

### 4.2 主要特性

- 接收 95 个外部中断源作为输入
- 生成 26 个外部中断作为输出
- 关闭 CPU 的 NMI 类型中断源
- 查询外部中断源当前的中断状态

中断矩阵的结构如图 4-1 所示。

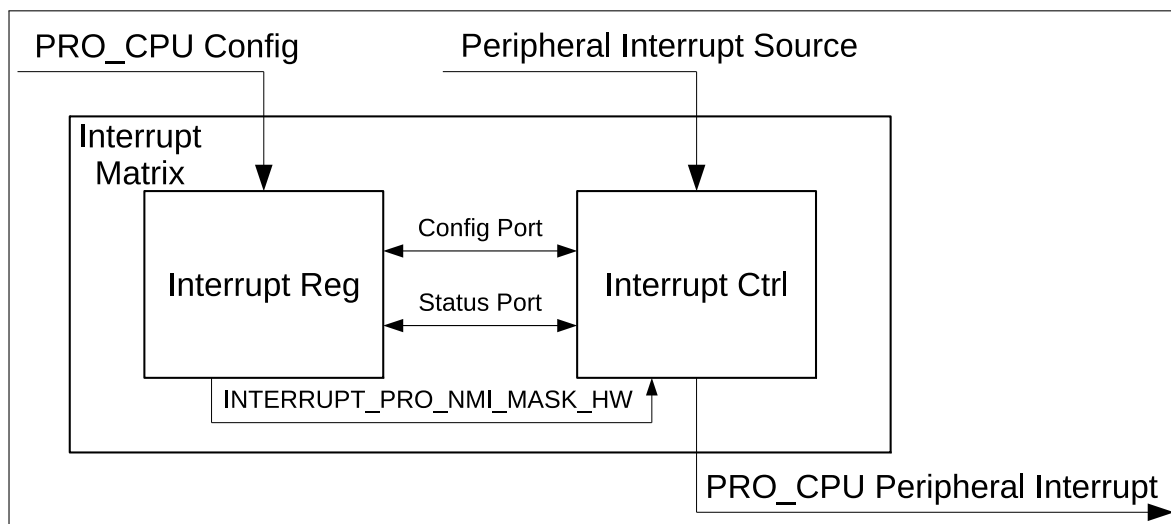


图 4-1. 中断矩阵结构图

### 4.3 功能描述

#### 4.3.1 外部中断源

ESP32-S2 共有 95 个外部中断源。表 17 列出了所有外部中断源，以及对应的中断配置寄存器与中断状态寄存器。上述 95 个外部中断源均可分配至 CPU 外部中断。

表 17: CPU 外部中断配置寄存器、外部中断状态寄存器、外部中断源

No.	中断源	配置寄存器	位	状态寄存器名称
0	MAC_INTR	INTERRUPT_PRO_MAC_INTR_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_0_REG
1	MAC_NMI	INTERRUPT_PRO_MAC_NMI_MAP_REG	1	
2	PWR_INTR	INTERRUPT_PRO_PWR_INTR_MAP_REG	2	
3	BB_INT	INTERRUPT_PRO_BB_INT_MAP_REG	3	
4	BT_MAC_INT	INTERRUPT_PRO_BT_MAC_INT_MAP_REG	4	
5	BT_BB_INT	INTERRUPT_PRO_BT_BB_INT_MAP_REG	5	
6	BT_BB_NMI	INTERRUPT_PRO_BT_BB_NMI_MAP_REG	6	
7	RWBT_IRQ	INTERRUPT_PRO_RWBT_IRQ_MAP_REG	7	
8	RWBLE_IRQ	INTERRUPT_PRO_RWBLE_IRQ_MAP_REG	8	
9	RWBT_NMI	INTERRUPT_PRO_RWBT_NMI_MAP_REG	9	
10	RWBLE_NMI	INTERRUPT_PRO_RWBLE_NMI_MAP_REG	10	
11	SLC0_INTR	INTERRUPT_PRO_SLC0_INTR_MAP_REG	11	
12	SLC1_INTR	INTERRUPT_PRO_SLC1_INTR_MAP_REG	12	
13	UHCIO_INTR	INTERRUPT_PRO_UHCIO_INTR_MAP_REG	13	
14	UHC1_INTR	INTERRUPT_PRO_UHC1_INTR_MAP_REG	14	
15	TG_T0_LEVEL_INT	INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG	15	
16	TG_T1_LEVEL_INT	INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG	16	
17	TG_WDT_LEVEL_INT	INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG	17	
18	TG_LACT_LEVEL_INT	INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG	18	
19	TG1_T0_LEVEL_INT	INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG	19	
20	TG1_T1_LEVEL_INT	INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG	20	
21	TG1_WDT_LEVEL_INT	INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	21	
22	TG1_LACT_LEVEL_INT	INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	22	
23	GPIO_INTERRUPT_PRO	INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG	23	
24	GPIO_INTERRUPT_PRO_NMI	INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	24	
25	GPIO_INTERRUPT_APP	INTERRUPT_PRO_GPIO_INTERRUPT_APP_MAP_REG	25	
26	GPIO_INTERRUPT_APP_NMI	INTERRUPT_PRO_GPIO_INTERRUPT_APP_NMI_MAP_REG	26	
27	DEDICATED_GPIO_IN_INTR	INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG	27	
28	CPU_INTR_FROM_CPU_0	INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	28	
29	CPU_INTR_FROM_CPU_1	INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	29	
30	CPU_INTR_FROM_CPU_2	INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	30	
31	CPU_INTR_FROM_CPU_3	INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	31	
32	SPI_INTR_1	INTERRUPT_PRO_SPI_INTR_1_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_1_REG
33	SPI_INTR_2	INTERRUPT_PRO_SPI_INTR_2_MAP_REG	1	
34	SPI_INTR_3	INTERRUPT_PRO_SPI_INTR_3_MAP_REG	2	
35	I2S0_INT	INTERRUPT_PRO_I2S0_INT_MAP_REG	3	
36	I2S1_INT	INTERRUPT_PRO_I2S1_INT_MAP_REG	4	
37	UART_INTR	INTERRUPT_PRO_UART_INTR_MAP_REG	5	
38	UART1_INTR	INTERRUPT_PRO_UART1_INTR_MAP_REG	6	

No.	中断源	配置寄存器	位	状态寄存器名称
39	UART2_INTR	INTERRUPT_PRO_UART2_INTR_MAP_REG	7	INTERRUPT_PRO_INTR_STATUS_REG_1_REG
40	SDIO_HOST_INTERRUPT	INTERRUPT_PRO_SDIO_HOST_INTERRUPT_MAP_REG	8	
41	PWM0_INTR	INTERRUPT_PRO_PWM0_INTR_MAP_REG	9	
42	PWM1_INTR	INTERRUPT_PRO_PWM1_INTR_MAP_REG	10	
43	PWM2_INTR	INTERRUPT_PRO_PWM2_INTR_MAP_REG	11	
44	PWM3_INTR	INTERRUPT_PRO_PWM3_INTR_MAP_REG	12	
45	LEDC_INT	INTERRUPT_PRO_LEDC_INT_MAP_REG	13	
46	EFUSE_INT	INTERRUPT_PRO_EFUSE_INT_MAP_REG	14	
47	CAN_INT	INTERRUPT_PRO_CAN_INT_MAP_REG	15	
48	USB_INTR	INTERRUPT_PRO_USB_INTR_MAP_REG	16	
49	RTC_CORE_INTR	INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG	17	
50	RMT_INTR	INTERRUPT_PRO_RMT_INTR_MAP_REG	18	
51	PCNT_INTR	INTERRUPT_PRO_PCNT_INTR_MAP_REG	19	
52	I2C_EXT0_INTR	INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG	20	
53	I2C_EXT1_INTR	INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG	21	
54	RSA_INTR	INTERRUPT_PRO_RSA_INTR_MAP_REG	22	
55	SHA_INTR	INTERRUPT_PRO_SHA_INTR_MAP_REG	23	
56	AES_INTR	INTERRUPT_PRO_AES_INTR_MAP_REG	24	
57	SPI2_DMA_INT	INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG	25	
58	SPI3_DMA_INT	INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG	26	
59	WDG_INT	INTERRUPT_PRO_WDG_INT_MAP_REG	27	
60	TIMER_INT	INTERRUPT_PRO_TIMER_INT1_MAP_REG	28	
61	TIMER_INT2	INTERRUPT_PRO_TIMER_INT2_MAP_REG	29	
62	TG_T0_EDGE_INT	INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG	30	
63	TG_T1_EDGE_INT	INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG	31	
64	TG_WDT_EDGE_INT	INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG	0	INTERRUPT_PRO_INTR_STATUS_REG_2_REG
65	TG_LACT_EDGE_INT	INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG	1	
66	TG1_T0_EDGE_INT	INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG	2	
67	TG1_T1_EDGE_INT	INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG	3	
68	TG1_WDT_EDGE_INT	INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG	4	
69	TG1_LACT_EDGE_INT	INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG	5	
70	CACHE_IA_INT	INTERRUPT_PRO_CACHE_IA_INT_MAP_REG	6	
71	SYSTIMER_TARGET0_INT	INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG	7	
72	SYSTIMER_TARGET1_INT	INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG	8	
73	SYSTIMER_TARGET2_INT	INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG	9	
74	ASSIST_DEBUG_INTR	INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG	10	
75	PMS_PRO_IRAM0_ILG_INTR	INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG	11	
76	PMS_PRO_DRAM0_ILG_INTR	INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG	12	
77	PMS_PRO_DPORT_ILG_INTR	INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG	13	
78	PMS_PRO_AHB_ILG_INTR	INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG	14	
79	PMS_PRO_CACHE_ILG_INTR	INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG	15	

No.	中断源	配置寄存器	状态寄存器	
			位	名称
80	PMS_DMA_APB_I_ILG_INTR	<a href="#">INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG</a>	16	<a href="#">INTERRUPT_PRO_INTR_STATUS_REG_2_REG</a>
81	PMS_DMA_RX_I_ILG_INTR	<a href="#">INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG</a>	17	
82	PMS_DMA_TX_I_ILG_INTR	<a href="#">INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG</a>	18	
83	SPI_MEM_REJECT_INTR	<a href="#">INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG</a>	19	
84	DMA_COPY_INTR	<a href="#">INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG</a>	20	
85	SPI4_DMA_INT	<a href="#">INTERRUPT_PRO_SPI4_DMA_INT_MAP_REG</a>	21	
86	DCACHE_PRELOAD_INT	<a href="#">INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG</a>	22	
87	DCACHE_PRELOAD_INT	<a href="#">INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG</a>	23	
88	ICACHE_PRELOAD_INT	<a href="#">INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG</a>	24	
89	APB_ADC_INT	<a href="#">INTERRUPT_PRO_APB_ADC_INT_MAP_REG</a>	25	
90	CRYPTO_DMA_INT	<a href="#">INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG</a>	26	
91	CPU_PERI_ERROR_INT	<a href="#">INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG</a>	27	
92	APB_PERI_ERROR_INT	<a href="#">INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG</a>	28	
93	DCACHE_SYNC_INT	<a href="#">INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG</a>	29	
94	ICACHE_SYNC_INT	<a href="#">INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG</a>	30	

### 4.3.2 CPU 中断

CPU 共有 32 个中断，其中包括 26 个外部中断和六个内部中断。表 18 列出了所有中断：

- 外部中断
  - 电平触发型中断：由高电平触发，要求保持中断的电平状态直到 CPU 响应。
  - 边沿触发型中断：由上升沿触发，此中断一旦发生，CPU 即可响应。
  - NMI 类型中断：软件不可使用 CPU 寄存器屏蔽此类中断。
- 内部中断
  - 定时器类型中断：由内部定时器触发，可用于产生周期性中断。
  - 软件类型中断：软件写特殊寄存器时将触发此中断。
  - 解析类型中断：用于性能监测与分析。

ESP32-S2 支持 6 级中断，在下表优先级一栏中，数字越大代表中断优先级越高。其中，NMI 拥有最高优先级，一旦 NMI 中断发生，CPU 必定会响应。

表 18: CPU 中断

编号	类别	种类	优先级
0	外部中断	电平触发	1
1	外部中断	电平触发	1
2	外部中断	电平触发	1
3	外部中断	电平触发	1
4	外部中断	电平触发	1
5	外部中断	电平触发	1
6	内部中断	定时器 0	1
7	内部中断	软件	1
8	外部中断	电平触发	1
9	外部中断	电平触发	1
10	外部中断	边沿触发	1
11	内部中断	解析	3
12	外部中断	电平触发	1
13	外部中断	电平触发	1
14	外部中断	NMI	NMI
15	内部中断	定时器 1	3
16	内部中断	定时器 2	5
17	外部中断	电平触发	1
18	外部中断	电平触发	1
19	外部中断	电平触发	2
20	外部中断	电平触发	2
21	外部中断	电平触发	2
22	外部中断	边沿触发	3
23	外部中断	电平触发	3
24	外部中断	电平触发	4

编号	类别	种类	优先级
25	外部中断	电平触发	4
26	外部中断	电平触发	5
27	外部中断	电平触发	3
28	外部中断	边沿触发	4
29	内部中断	软件	3
30	外部中断	边沿触发	4
31	外部中断	电平触发	5

### 4.3.3 分配外部中断源至 CPU 外部中断

在本小节中，我们将使用以下术语描述中断矩阵相关操作：

- Source\_X：用于表示某个外部中断源，其中 X 为中断源编号，详见表 17。
- INTERRUPT\_PRO\_X\_MAP\_REG：用于表示 CPU 的某个外部中断配置寄存器。此外部中断配置寄存器与外部中断源 Source\_X 相对应。即表 17 中“配置寄存器”一栏与“中断源”一栏应一一对应。例如：MAC\_INTR 的中断配置寄存器为处于同一行的 INTERRUPT\_PRO\_MAC\_INTR\_MAP\_REG。
- Interrupt\_P：表示 CPU 中断序号为 Num\_P 的外部中断，Num\_P 的取值范围为 0 ~ 5、8 ~ 10、12 ~ 14、17 ~ 28、30 ~ 31，详见表 18。
- Interrupt\_I：表示 CPU 中断序号为 Num\_I 的内部中断，Num\_I 的取值范围为 6、7、11、15、16、29，详见表 18。

#### 4.3.3.1 分配一个外部中断源 Source\_X 至 CPU 外部中断

将外部中断源 Source\_X 对应的寄存器 INTERRUPT\_PRO\_X\_MAP\_REG 配置成 Num\_P，即可将该中断源分配至序号为 Num\_P 的外部中断 (Interrupt\_P)。Num\_P 可以取任一 CPU 外部中断值，包括 0 ~ 5、8 ~ 10、12 ~ 14、17 ~ 28、30 ~ 31。一个 CPU 外部中断可以被多个外设共享。

#### 4.3.3.2 分配多个外部中断源 Source\_X<sub>n</sub> 至 CPU 外部中断

将各个中断源对应的寄存器 INTERRUPT\_PRO\_X<sub>n</sub>\_MAP\_REG 均配置成相同的 Num\_P，即可将多个中断源 Source\_X<sub>n</sub> 分配至同一 CPU 外部中断 Interrupt\_P。上述任一外设中断源均会触发 CPU 外部中断 Interrupt\_P。待中断触发后，软件需要查询中断状态寄存器，用于判断哪个外设产生中断。

#### 4.3.3.3 关闭 CPU 外部中断源 Source\_X

将中断源对应的寄存器 INTERRUPT\_PRO\_X\_MAP\_REG 配置成任意 Num\_I，即可关闭外部中断源。这是因为任何被配置成 Num\_I 的外部中断均无法连接至 CPU，而且选择任一内部中断值 (6、7、11、15、16、29) 不会造成其他影响，可用于关闭外部中断。



### 4.3.4 关闭 CPU 的 NMI 类型中断源

中断矩阵内部的 Interrupt Reg 寄存器配置子模块可以产生 `INTERRUPT_PRO_NMI_MASK_HW` 信号，中断矩阵根据此内部信号可以通过硬件关闭所有被分配到 CPU 第 14 号 NMI 中断的外部中断源。信号 `INTERRUPT_PRO_NMI_MASK_HW` 可以配置为高电平，此时 CPU 不响应 NMI 中断，也可以配置为低电平，则 CPU 响应 NMI 中断。具体结构请参考图 4-1 所示。

### 4.3.5 查询外部中断源当前的中断状态

读取寄存器 `INTERRUPT_PRO_INTR_STATUS_REG_n`（只读）中的特定 Bit 值可以获取外部中断源当前的中断状态。寄存器 `INTERRUPT_PRO_INTR_STATUS_REG_n` 与外部中断源的对应关系如表 17 所示。

## 4.4 基地址

用户可以通过寄存器基地址访问中断矩阵，如表 19 所示。更多信息，请访问章节 1 [系统和存储器](#)。

表 19: 中断矩阵基地址

访问总线	基地址
PeriBUS1	0x3F4C2000

## 4.5 寄存器列表

请注意，下表中的地址都是相对于中断矩阵基地址的地址偏移量（相对地址）。更多有关中断矩阵基地址的信息，请前往第 4.4 节。

名称	描述	地址	访问
配置寄存器			
<a href="#">INTERRUPT_PRO_MAC_INTR_MAP_REG</a>	MAC_INTR 中断配置寄存器	0x0000	读/写
<a href="#">INTERRUPT_PRO_MAC_NMI_MAP_REG</a>	MAC_NMI 中断配置寄存器	0x0004	读/写
<a href="#">INTERRUPT_PRO_PWR_INTR_MAP_REG</a>	PWR_INTR 中断配置寄存器	0x0008	读/写
<a href="#">INTERRUPT_PRO_BB_INT_MAP_REG</a>	BB_INT 中断配置寄存器	0x000C	读/写
<a href="#">INTERRUPT_PRO_BT_MAC_INT_MAP_REG</a>	BT_MAC 中断配置寄存器	0x0010	读/写
<a href="#">INTERRUPT_PRO_BT_BB_INT_MAP_REG</a>	BT_BB_INT 中断配置寄存器	0x0014	读/写
<a href="#">INTERRUPT_PRO_BT_BB_NMI_MAP_REG</a>	BT_BB_NMI 中断配置寄存器	0x0018	读/写
<a href="#">INTERRUPT_PRO_RWB_T_IRQ_MAP_REG</a>	RWB_T_IRQ 中断配置寄存器	0x001C	读/写
<a href="#">INTERRUPT_PRO_RWBLE_IRQ_MAP_REG</a>	RWBLE_IRQ 中断配置寄存器	0x0020	读/写
<a href="#">INTERRUPT_PRO_RWB_T_NMI_MAP_REG</a>	RWB_T_NMI 中断配置寄存器	0x0024	读/写
<a href="#">INTERRUPT_PRO_RWBLE_NMI_MAP_REG</a>	RWBLE_NMI 中断配置寄存器	0x0028	读/写
<a href="#">INTERRUPT_PRO_SLC0_INTR_MAP_REG</a>	SLC0_INTR 中断配置寄存器	0x002C	读/写
<a href="#">INTERRUPT_PRO_SLC1_INTR_MAP_REG</a>	SLC1_INTR 中断配置寄存器	0x0030	读/写
<a href="#">INTERRUPT_PRO_UHCI0_INTR_MAP_REG</a>	UHCI0_INTR 中断配置寄存器	0x0034	读/写
<a href="#">INTERRUPT_PRO_UHCI1_INTR_MAP_REG</a>	UHCI1_INTR 中断配置寄存器	0x0038	读/写
<a href="#">INTERRUPT_PRO_TG_T0_LEVEL_INT_MAP_REG</a>	TG_T0_LEVEL_INT 中断配置寄存器	0x003C	读/写
<a href="#">INTERRUPT_PRO_TG_T1_LEVEL_INT_MAP_REG</a>	TG_T1_LEVEL_INT 中断配置寄存器	0x0040	读/写
<a href="#">INTERRUPT_PRO_TG_WDT_LEVEL_INT_MAP_REG</a>	TG_WDT_LEVEL_INT 中断配置寄存器	0x0044	读/写
<a href="#">INTERRUPT_PRO_TG_LACT_LEVEL_INT_MAP_REG</a>	TG_LACT_LEVEL_INT 中断配置寄存器	0x0048	读/写
<a href="#">INTERRUPT_PRO_TG1_T0_LEVEL_INT_MAP_REG</a>	TG1_T0_LEVEL_INT 中断配置寄存器	0x004C	读/写
<a href="#">INTERRUPT_PRO_TG1_T1_LEVEL_INT_MAP_REG</a>	TG1_T1_LEVEL_INT 中断配置寄存器	0x0050	读/写
<a href="#">INTERRUPT_PRO_TG1_WDT_LEVEL_INT_MAP_REG</a>	TG1_WDT_LEVEL_INT 中断配置寄存器	0x0054	读/写
<a href="#">INTERRUPT_PRO_TG1_LACT_LEVEL_INT_MAP_REG</a>	TG1_LACT_LEVEL_INT 中断配置寄存器	0x0058	读/写
<a href="#">INTERRUPT_PRO_GPIO_INTERRUPT_PRO_MAP_REG</a>	GPIO_INTERRUPT_PRO 中断配置寄存器	0x005C	读/写
<a href="#">INTERRUPT_PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG</a>	GPIO_INTERRUPT_PRO_NMI 中断配置寄存器	0x0060	读/写
<a href="#">INTERRUPT_PRO_GPIO_INTERRUPT_APP_MAP_REG</a>	GPIO_INTERRUPT_APP 中断配置寄存器	0x0064	读/写
<a href="#">INTERRUPT_PRO_GPIO_INTERRUPT_APP_NMI_MAP_REG</a>	GPIO_INTERRUPT_APP_NMI 中断配置寄存器	0x0068	读/写
<a href="#">INTERRUPT_PRO_DEDICATED_GPIO_IN_INTR_MAP_REG</a>	DEDICATED_GPIO_IN_INTR 中断配置寄存器	0x006C	读/写
<a href="#">INTERRUPT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG</a>	CPU_INTR_FROM_CPU_0 中断配置寄存器	0x0070	读/写
<a href="#">INTERRUPT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG</a>	CPU_INTR_FROM_CPU_1 中断配置寄存器	0x0074	读/写

名称	描述	地址	访问
<a href="#">INTERRUPT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG</a>	CPU_INTR_FROM_CPU_2 中断配置寄存器	0x0078	读/写
<a href="#">INTERRUPT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG</a>	CPU_INTR_FROM_CPU_3 中断配置寄存器	0x007C	读/写
<a href="#">INTERRUPT_PRO_SPI_INTR_1_MAP_REG</a>	SPI_INTR_1 中断配置寄存器	0x0080	读/写
<a href="#">INTERRUPT_PRO_SPI_INTR_2_MAP_REG</a>	SPI_INTR_2 中断配置寄存器	0x0084	读/写
<a href="#">INTERRUPT_PRO_SPI_INTR_3_MAP_REG</a>	SPI_INTR_3 中断配置寄存器	0x0088	读/写
<a href="#">INTERRUPT_PRO_I2S0_INT_MAP_REG</a>	I2S0_INT 中断配置寄存器	0x008C	读/写
<a href="#">INTERRUPT_PRO_I2S1_INT_MAP_REG</a>	I2S1_INT 中断配置寄存器	0x0090	读/写
<a href="#">INTERRUPT_PRO_UART_INTR_MAP_REG</a>	UART_INTR 中断配置寄存器	0x0094	读/写
<a href="#">INTERRUPT_PRO_UART1_INTR_MAP_REG</a>	UART1_INTR 中断配置寄存器	0x0098	读/写
<a href="#">INTERRUPT_PRO_UART2_INTR_MAP_REG</a>	UART2_INTR 中断配置寄存器	0x009C	读/写
<a href="#">INTERRUPT_PRO_SDIO_HOST_INTERRUPT_MAP_REG</a>	SDIO_HOST_INTERRUPT 中断配置寄存器	0x00A0	读/写
<a href="#">INTERRUPT_PRO_PWM0_INTR_MAP_REG</a>	PWM0_INTR 中断配置寄存器	0x00A4	读/写
<a href="#">INTERRUPT_PRO_PWM1_INTR_MAP_REG</a>	PWM1_INTR 中断配置寄存器	0x00A8	读/写
<a href="#">INTERRUPT_PRO_PWM2_INTR_MAP_REG</a>	PWM2_INTR 中断配置寄存器	0x00AC	读/写
<a href="#">INTERRUPT_PRO_PWM3_INTR_MAP_REG</a>	PWM3_INTR 中断配置寄存器	0x00B0	读/写
<a href="#">INTERRUPT_PRO_LEDC_INT_MAP_REG</a>	LEDC_INT 中断配置寄存器	0x00B4	读/写
<a href="#">INTERRUPT_PRO_EFUSE_INT_MAP_REG</a>	EFUSE_INT 中断配置寄存器	0x00B8	读/写
<a href="#">INTERRUPT_PRO_CAN_INT_MAP_REG</a>	CAN_INT 中断配置寄存器	0x00BC	读/写
<a href="#">INTERRUPT_PRO_USB_INTR_MAP_REG</a>	USB_INTR 中断配置寄存器	0x00C0	读/写
<a href="#">INTERRUPT_PRO_RTC_CORE_INTR_MAP_REG</a>	RTC_CORE_INTR 中断配置寄存器	0x00C4	读/写
<a href="#">INTERRUPT_PRO_RMT_INTR_MAP_REG</a>	RMT_INTR 中断配置寄存器	0x00C8	读/写
<a href="#">INTERRUPT_PRO_PCNT_INTR_MAP_REG</a>	PCNT_INTR 中断配置寄存器	0x00CC	读/写
<a href="#">INTERRUPT_PRO_I2C_EXT0_INTR_MAP_REG</a>	I2C_EXT0_INTR 中断配置寄存器	0x00D0	读/写
<a href="#">INTERRUPT_PRO_I2C_EXT1_INTR_MAP_REG</a>	I2C_EXT1_INTR 中断配置寄存器	0x00D4	读/写
<a href="#">INTERRUPT_PRO_RSA_INTR_MAP_REG</a>	RSA_INTR 中断配置寄存器	0x00D8	读/写
<a href="#">INTERRUPT_PRO_SHA_INTR_MAP_REG</a>	SHA_INTR 中断配置寄存器	0x00DC	读/写
<a href="#">INTERRUPT_PRO_AES_INTR_MAP_REG</a>	AES_INTR 中断配置寄存器	0x00E0	读/写
<a href="#">INTERRUPT_PRO_SPI2_DMA_INT_MAP_REG</a>	SPI2_DMA_INT 中断配置寄存器	0x00E4	读/写
<a href="#">INTERRUPT_PRO_SPI3_DMA_INT_MAP_REG</a>	SPI3_DMA_INT 中断配置寄存器	0x00E8	读/写
<a href="#">INTERRUPT_PRO_WDG_INT_MAP_REG</a>	WDG_INT 中断配置寄存器	0x00EC	读/写
<a href="#">INTERRUPT_PRO_TIMER_INT1_MAP_REG</a>	TIMER_INT1 中断配置寄存器	0x00F0	读/写

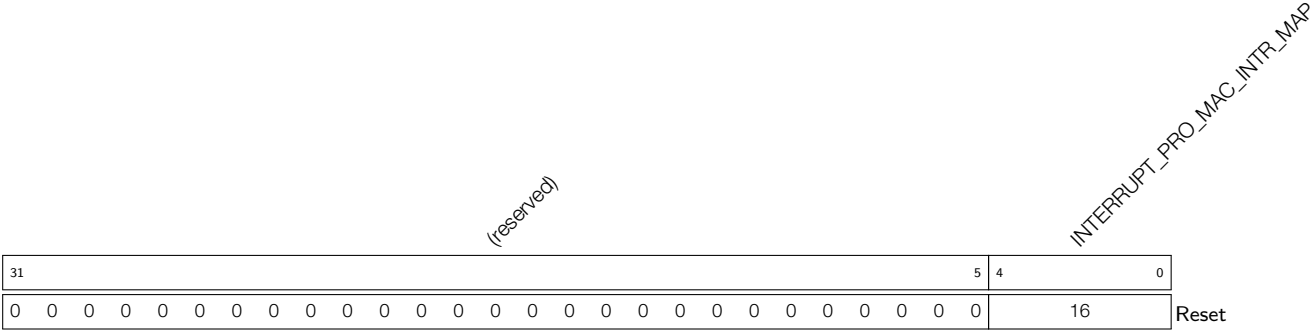
名称	描述	地址	访问
INTERRUPT_PRO_TIMER_INT2_MAP_REG	TIMER_INT2 中断配置寄存器	0x00F4	读/写
INTERRUPT_PRO_TG_T0_EDGE_INT_MAP_REG	TG_T0_EDGE_INT 中断配置寄存器	0x00F8	读/写
INTERRUPT_PRO_TG_T1_EDGE_INT_MAP_REG	TG_T1_EDGE_INT 中断配置寄存器	0x00FC	读/写
INTERRUPT_PRO_TG_WDT_EDGE_INT_MAP_REG	TG_WDT_EDGE_INT 中断配置寄存器	0x0100	读/写
INTERRUPT_PRO_TG_LACT_EDGE_INT_MAP_REG	TG_LACT_EDGE_INT 中断配置寄存器	0x0104	读/写
INTERRUPT_PRO_TG1_T0_EDGE_INT_MAP_REG	TG1_T0_EDGE_INT 中断配置寄存器	0x0108	读/写
INTERRUPT_PRO_TG1_T1_EDGE_INT_MAP_REG	TG1_T1_EDGE_INT 中断配置寄存器	0x010C	读/写
INTERRUPT_PRO_TG1_WDT_EDGE_INT_MAP_REG	TG1_WDT_EDGE_INT 中断配置寄存器	0x0110	读/写
INTERRUPT_PRO_TG1_LACT_EDGE_INT_MAP_REG	TG1_LACT_EDGE_INT 中断配置寄存器	0x0114	读/写
INTERRUPT_PRO_CACHE_IA_INT_MAP_REG	CACHE_IA_INT 中断配置寄存器	0x0118	读/写
INTERRUPT_PRO_SYSTIMER_TARGET0_INT_MAP_REG	SYSTIMER_TARGET0_INT 中断配置寄存器	0x011C	读/写
INTERRUPT_PRO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1 中断配置寄存器	0x0120	读/写
INTERRUPT_PRO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2 中断配置寄存器	0x0124	读/写
INTERRUPT_PRO_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR 中断配置寄存器	0x0128	读/写
INTERRUPT_PRO_PMS_PRO_IRAM0_ILG_INTR_MAP_REG	PMS_PRO_IRAM0_ILG_INTR 中断配置寄存器	0x012C	读/写
INTERRUPT_PRO_PMS_PRO_DRAM0_ILG_INTR_MAP_REG	PMS_PRO_DRAM0_ILG_INTR 中断配置寄存器	0x0130	读/写
INTERRUPT_PRO_PMS_PRO_DPORT_ILG_INTR_MAP_REG	PMS_PRO_DPORT_ILG_INTR 中断配置寄存器	0x0134	读/写
INTERRUPT_PRO_PMS_PRO_AHB_ILG_INTR_MAP_REG	PMS_PRO_AHB_ILG_INTR 中断配置寄存器	0x0138	读/写
INTERRUPT_PRO_PMS_PRO_CACHE_ILG_INTR_MAP_REG	PMS_PRO_CACHE_ILG_INTR 中断配置寄存器	0x013C	读/写
INTERRUPT_PRO_PMS_DMA_APB_I_ILG_INTR_MAP_REG	PMS_DMA_APB_I_ILG_INTR 中断配置寄存器	0x0140	读/写
INTERRUPT_PRO_PMS_DMA_RX_I_ILG_INTR_MAP_REG	PMS_DMA_RX_I_ILG_INTR 中断配置寄存器	0x0144	读/写
INTERRUPT_PRO_PMS_DMA_TX_I_ILG_INTR_MAP_REG	PMS_DMA_TX_I_ILG_INTR 中断配置寄存器	0x0148	读/写
INTERRUPT_PRO_SPI_MEM_REJECT_INTR_MAP_REG	SPI_MEM_REJECT_INTR 中断配置寄存器	0x014C	读/写
INTERRUPT_PRO_DMA_COPY_INTR_MAP_REG	DMA_COPY_INTR 中断配置寄存器	0x0150	读/写
INTERRUPT_PRO_SPI4_DMA_INT_MAP_REG	SPI4_DMA_INT 中断配置寄存器	0x0154	读/写
INTERRUPT_PRO_SPI_INTR_4_MAP_REG	SPI_INTR_4 中断配置寄存器	0x0158	读/写
INTERRUPT_PRO_DCACHE_PRELOAD_INT_MAP_REG	DCACHE_PRELOAD_INT 中断配置寄存器	0x015C	读/写
INTERRUPT_PRO_ICACHE_PRELOAD_INT_MAP_REG	ICACHE_PRELOAD_INT 中断配置寄存器	0x0160	读/写
INTERRUPT_PRO_APB_ADC_INT_MAP_REG	APB_ADC_INT 中断配置寄存器	0x0164	读/写
INTERRUPT_PRO_CRYPTO_DMA_INT_MAP_REG	CRYPTO_DMA_INT 中断配置寄存器	0x0168	读/写
INTERRUPT_PRO_CPU_PERI_ERROR_INT_MAP_REG	cpu peri error 中断配置寄存器	0x016C	读/写

名称	描述	地址	访问
<a href="#">INTERRUPT_PRO_APB_PERI_ERROR_INT_MAP_REG</a>	CPU_PERI_ERROR_INT 中断配置寄存器	0x0170	读/写
<a href="#">INTERRUPT_PRO_DCACHE_SYNC_INT_MAP_REG</a>	DCACHE_SYNC_INT 中断配置寄存器	0x0174	读/写
<a href="#">INTERRUPT_PRO_ICACHE_SYNC_INT_MAP_REG</a>	ICACHE_SYNC_INT 中断配置寄存器	0x0178	读/写
<a href="#">INTERRUPT_CLOCK_GATE_REG</a>	NMI 中断信号屏蔽寄存器	0x0188	读/写
中断状态寄存器			
<a href="#">INTERRUPT_PRO_INTR_STATUS_REG_0_REG</a>	中断状态寄存器 0	0x017C	只读
<a href="#">INTERRUPT_PRO_INTR_STATUS_REG_1_REG</a>	中断状态寄存器 1	0x0180	只读
<a href="#">INTERRUPT_PRO_INTR_STATUS_REG_2_REG</a>	中断状态寄存器 2	0x0184	只读
版本寄存器			
<a href="#">INTERRUPT_REG_DATE_REG</a>	版本控制寄存器	0x0FFC	读/写

### 4.6 寄存器

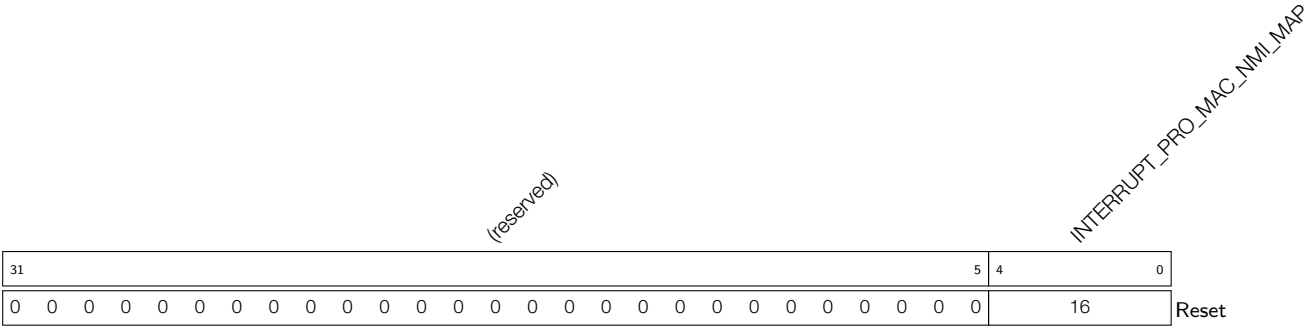
请注意，下表中的地址都是相对于中断矩阵基地址的地址偏移量（相对地址）。更多有关中断矩阵基地址的信息，请前往第 4.4 节。

Register 4.1: INTERRUPT\_PRO\_MAC\_INTR\_MAP\_REG (0x0000)



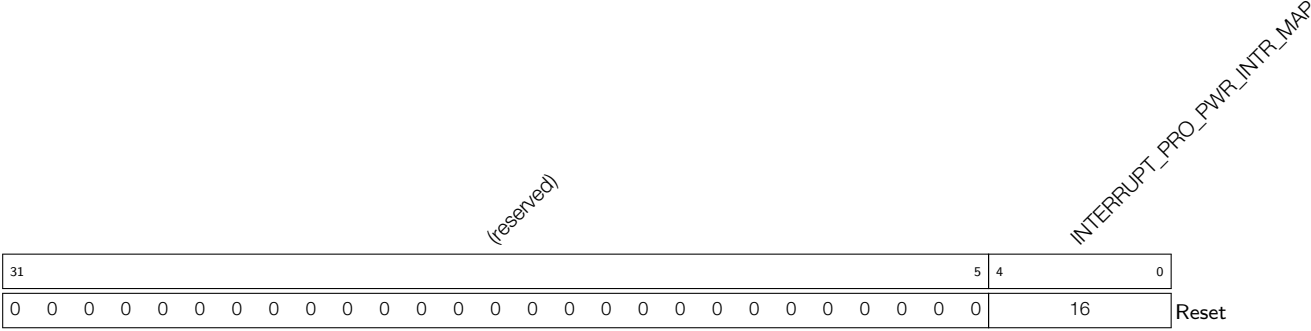
**INTERRUPT\_PRO\_MAC\_INTR\_MAP** 用于将 MAC\_INTR 中断信号映射至 CPU 中断。（读/写）

Register 4.2: INTERRUPT\_PRO\_MAC\_NMI\_MAP\_REG (0x0004)



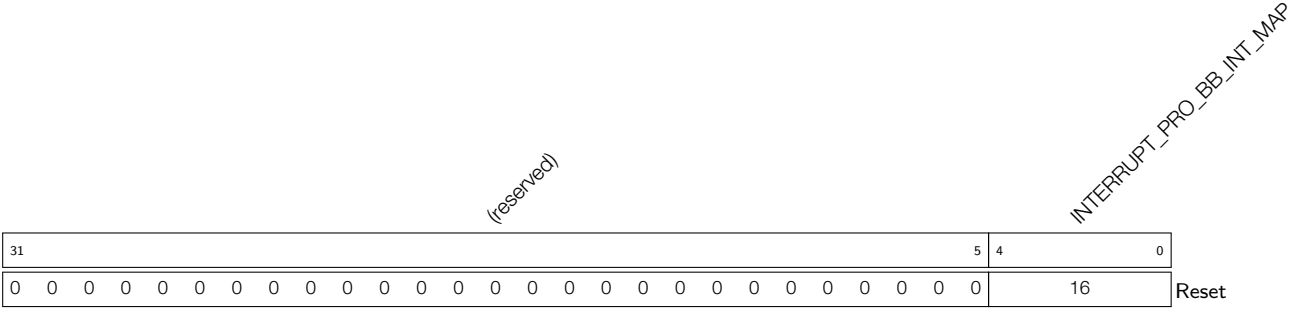
**INTERRUPT\_PRO\_MAC\_NMI\_MAP** 用于将 MAC\_NMI 中断信号映射至 CPU 中断。（读/写）

Register 4.3: INTERRUPT\_PRO\_PWR\_INTR\_MAP\_REG (0x0008)



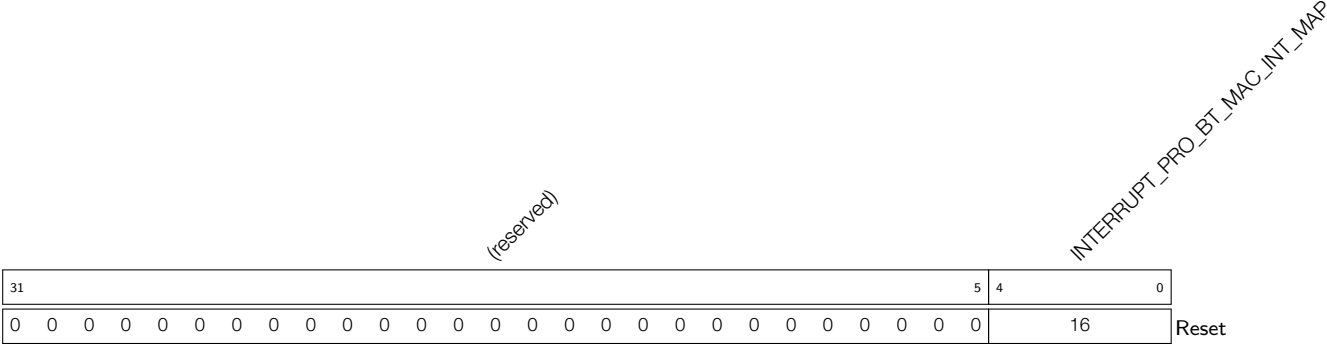
**INTERRUPT\_PRO\_PWR\_INTR\_MAP** 用于将 PWR\_INTR 中断信号映射至 CPU 中断。（读/写）

Register 4.4: INTERRUPT\_PRO\_BB\_INT\_MAP\_REG (0x000C)



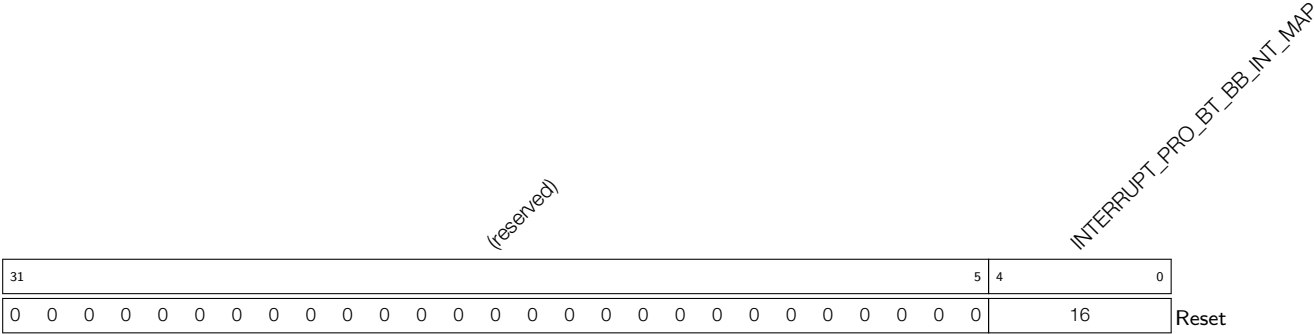
INTERRUPT\_PRO\_BB\_INT\_MAP 用于将 BB\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.5: INTERRUPT\_PRO\_BT\_MAC\_INT\_MAP\_REG (0x0010)



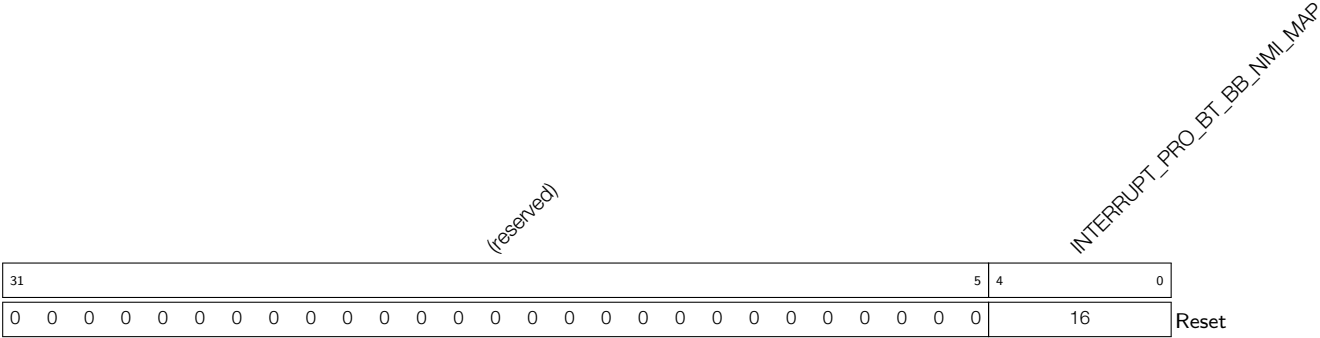
INTERRUPT\_PRO\_BT\_MAC\_INT\_MAP 用于将 BT\_MAC\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.6: INTERRUPT\_PRO\_BT\_BB\_INT\_MAP\_REG (0x0014)



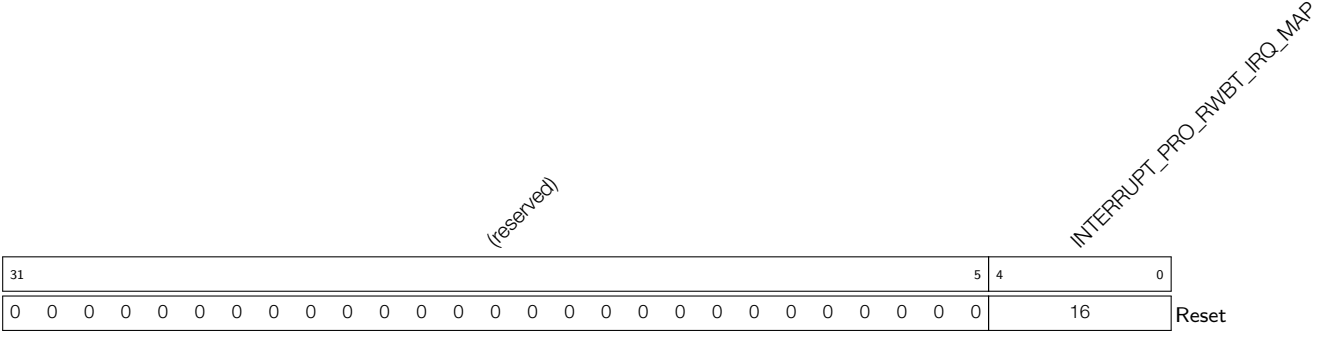
INTERRUPT\_PRO\_BT\_BB\_INT\_MAP 用于将 BT\_BB\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.7: INTERRUPT\_PRO\_BT\_BB\_NMI\_MAP\_REG (0x0018)



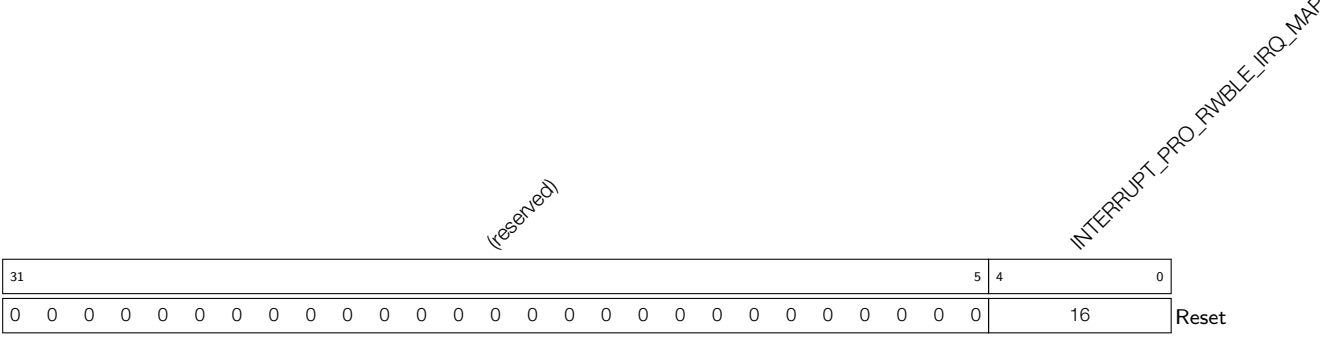
**INTERRUPT\_PRO\_BT\_BB\_NMI\_MAP** 用于将 BT\_BB\_NMI 中断信号映射至 CPU 中断。(读/写)

Register 4.8: INTERRUPT\_PRO\_RWBT\_IRQ\_MAP\_REG (0x001C)



**INTERRUPT\_PRO\_RWBT\_IRQ\_MAP** 用于将 RWBT\_IRQ 中断信号映射至 CPU 中断。(读/写)

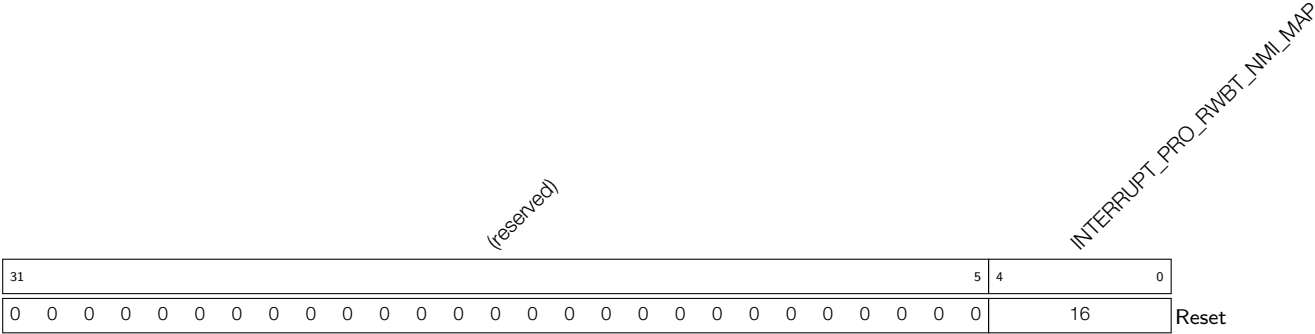
Register 4.9: INTERRUPT\_PRO\_RWBLE\_IRQ\_MAP\_REG (0x0020)



**INTERRUPT\_PRO\_RWBLE\_IRQ\_MAP** 用于将 RWBLE\_IRQ 中断信号映射至 CPU 中断。(读/写)

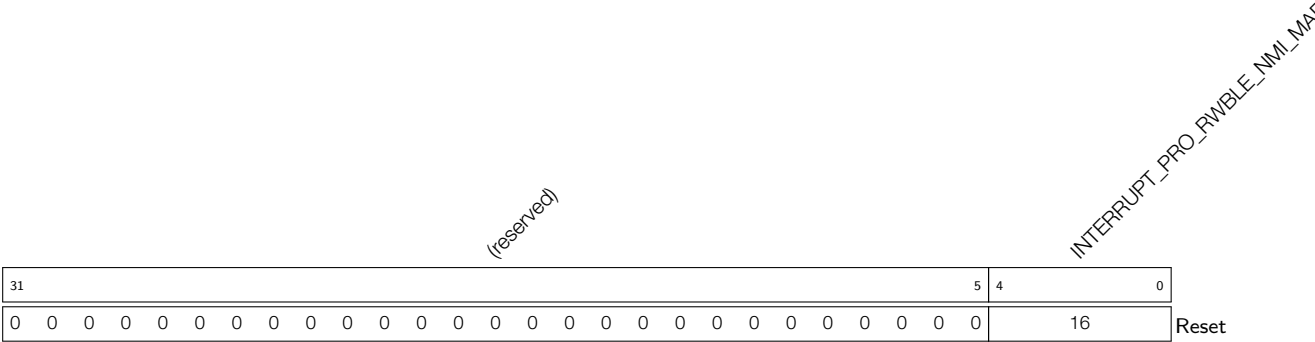


Register 4.10: INTERRUPT\_PRO\_RWBT\_NMI\_MAP\_REG (0x0024)



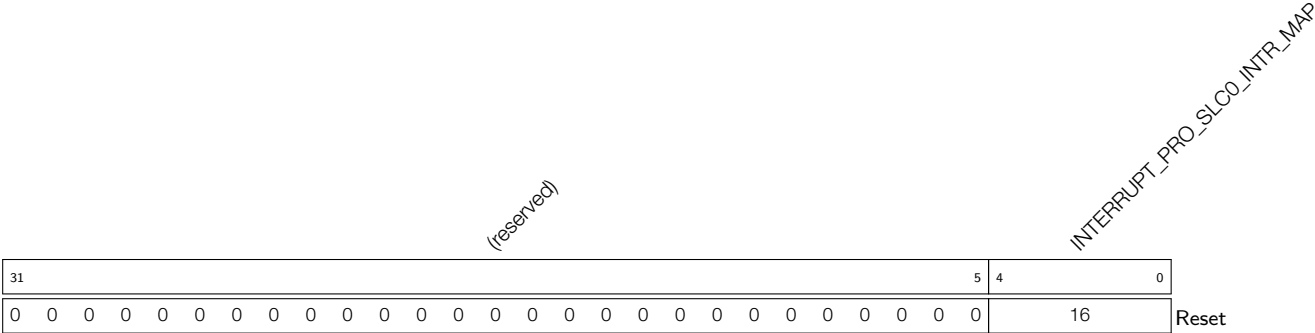
**INTERRUPT\_PRO\_RWBT\_NMI\_MAP** 用于将 RWBT\_NMI 中断信号映射至 CPU 中断。(读/写)

Register 4.11: INTERRUPT\_PRO\_RWBLE\_NMI\_MAP\_REG (0x0028)



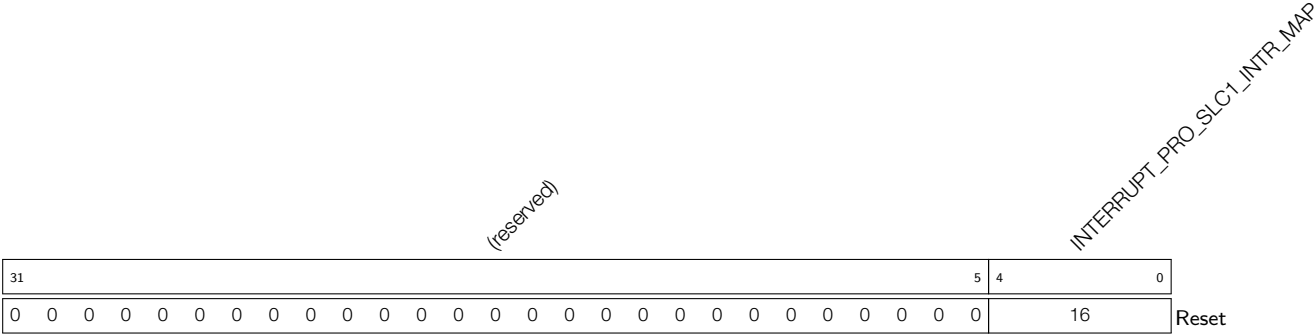
**INTERRUPT\_PRO\_RWBLE\_NMI\_MAP** 用于将 RWBLE\_NMI 中断信号映射至 CPU 中断。(读/写)

Register 4.12: INTERRUPT\_PRO\_SLC0\_INTR\_MAP\_REG (0x002C)



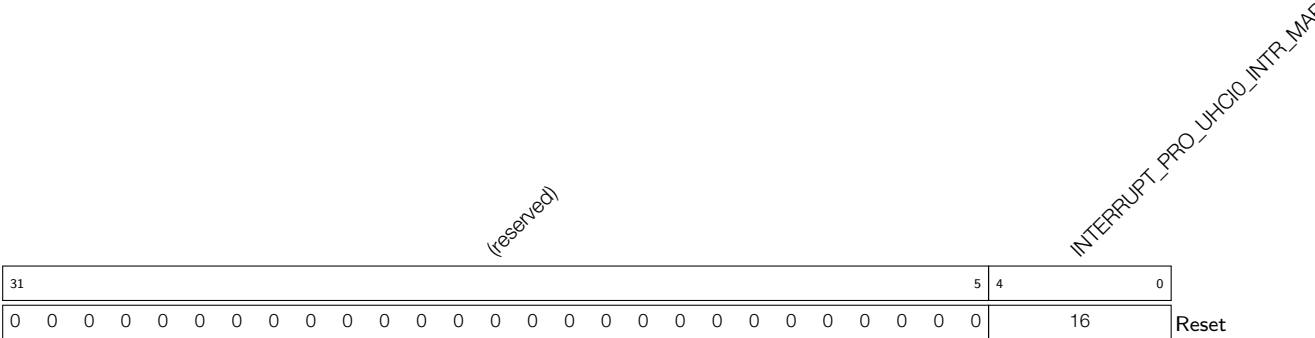
**INTERRUPT\_PRO\_SLC0\_INTR\_MAP** 用于将 SLC0\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.13: INTERRUPT\_PRO\_SLC1\_INTR\_MAP\_REG (0x0030)



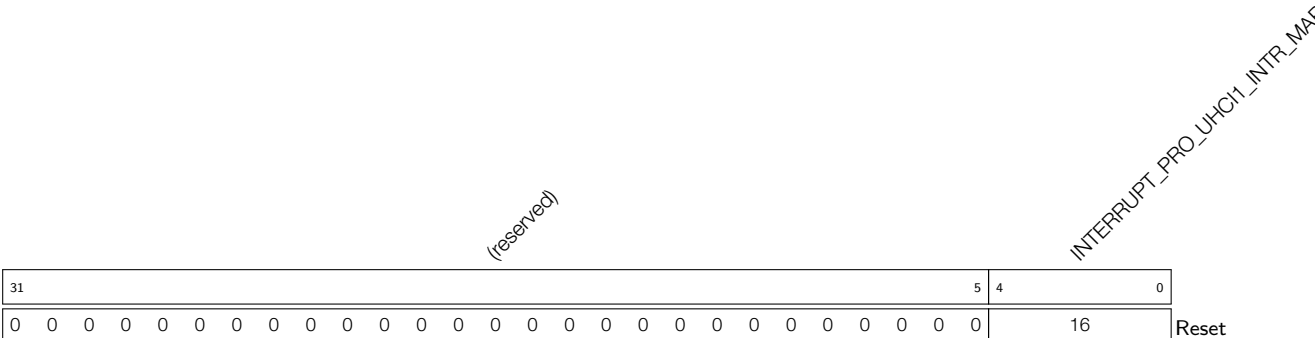
**INTERRUPT\_PRO\_SLC1\_INTR\_MAP** 用于将 SLC1\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.14: INTERRUPT\_PRO\_UHCI0\_INTR\_MAP\_REG (0x0034)



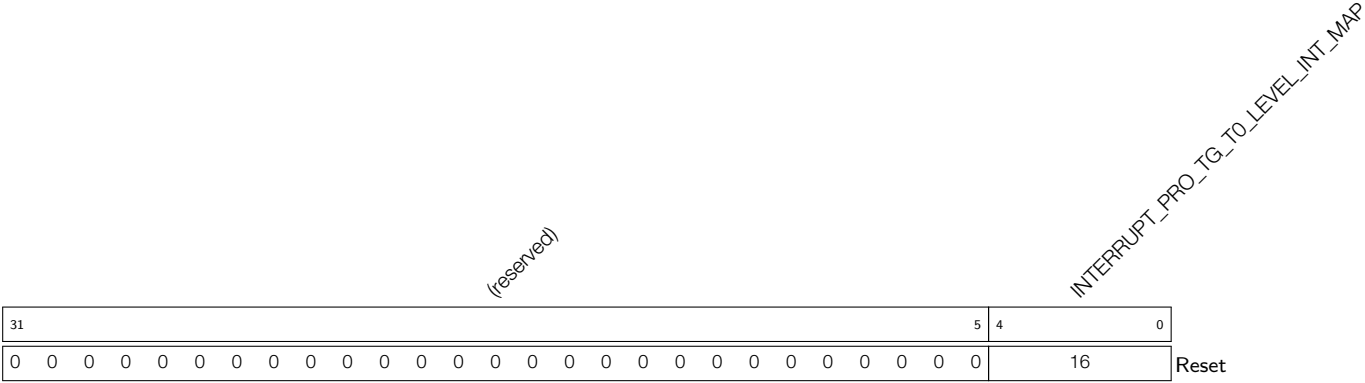
**INTERRUPT\_PRO\_UHCI0\_INTR\_MAP** 用于将 UHCI0\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.15: INTERRUPT\_PRO\_UHCI1\_INTR\_MAP\_REG (0x0038)



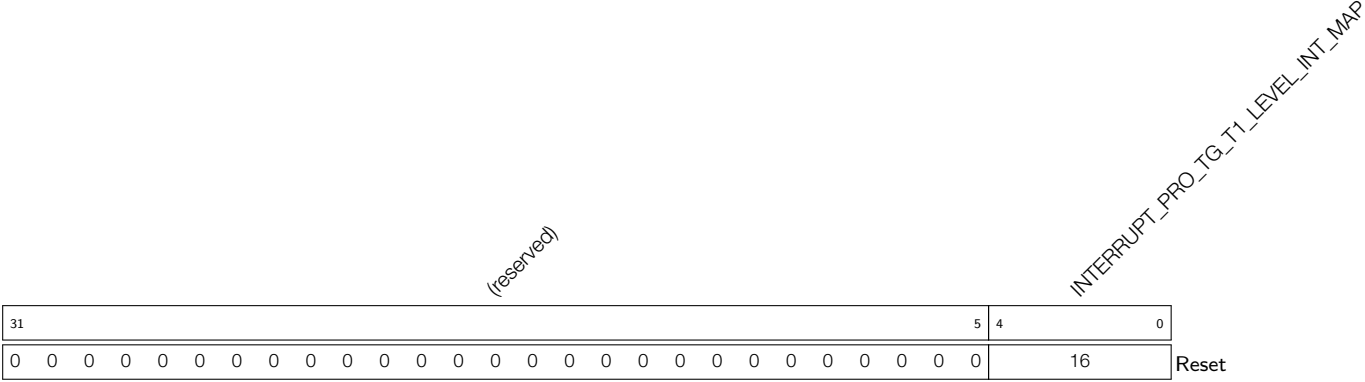
**INTERRUPT\_PRO\_UHCI1\_INTR\_MAP** 用于将 UHCI1\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.16: INTERRUPT\_PRO\_TG\_T0\_LEVEL\_INT\_MAP\_REG (0x003C)



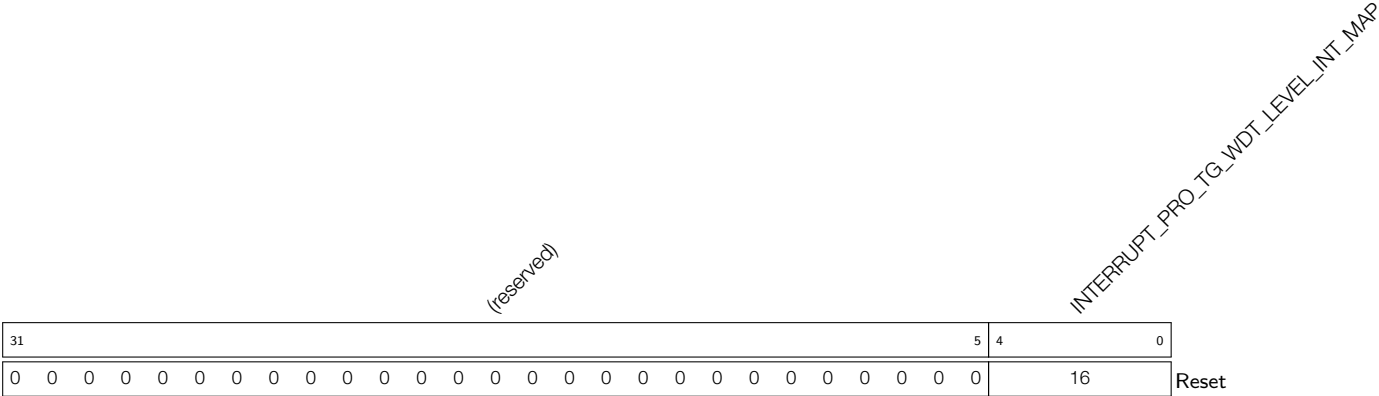
**INTERRUPT\_PRO\_TG\_T0\_LEVEL\_INT\_MAP** 用于将 TG\_T0\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.17: INTERRUPT\_PRO\_TG\_T1\_LEVEL\_INT\_MAP\_REG (0x0040)



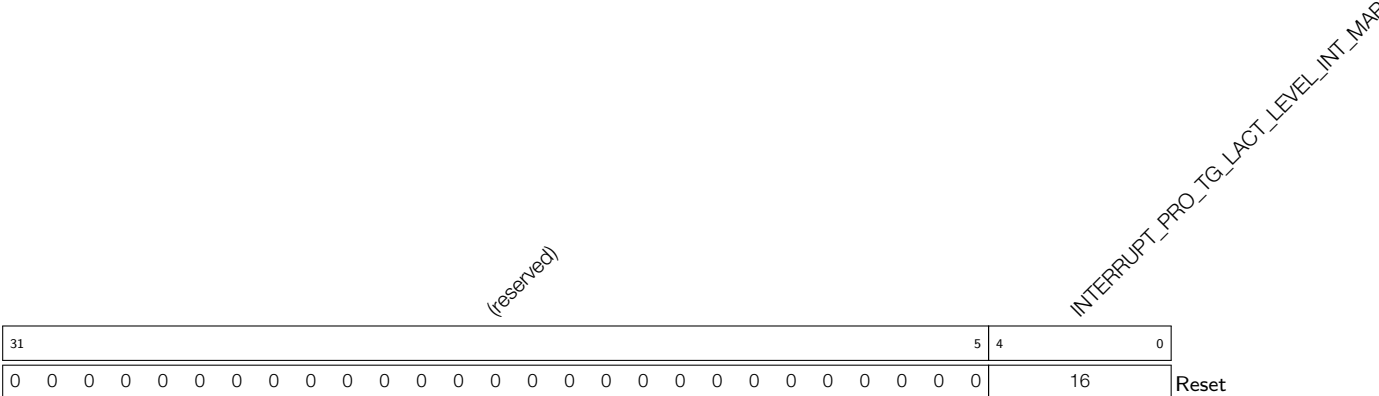
**INTERRUPT\_PRO\_TG\_T1\_LEVEL\_INT\_MAP** 用于将 TG\_T1\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.18: INTERRUPT\_PRO\_TG\_WDT\_LEVEL\_INT\_MAP\_REG (0x0044)



INTERRUPT\_PRO\_TG\_WDT\_LEVEL\_INT\_MAP 用于将 TG\_WDT\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.19: INTERRUPT\_PRO\_TG\_LACT\_LEVEL\_INT\_MAP\_REG (0x0048)



INTERRUPT\_PRO\_TG\_LACT\_LEVEL\_INT\_MAP 用于将 TG\_LACT\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

## 53



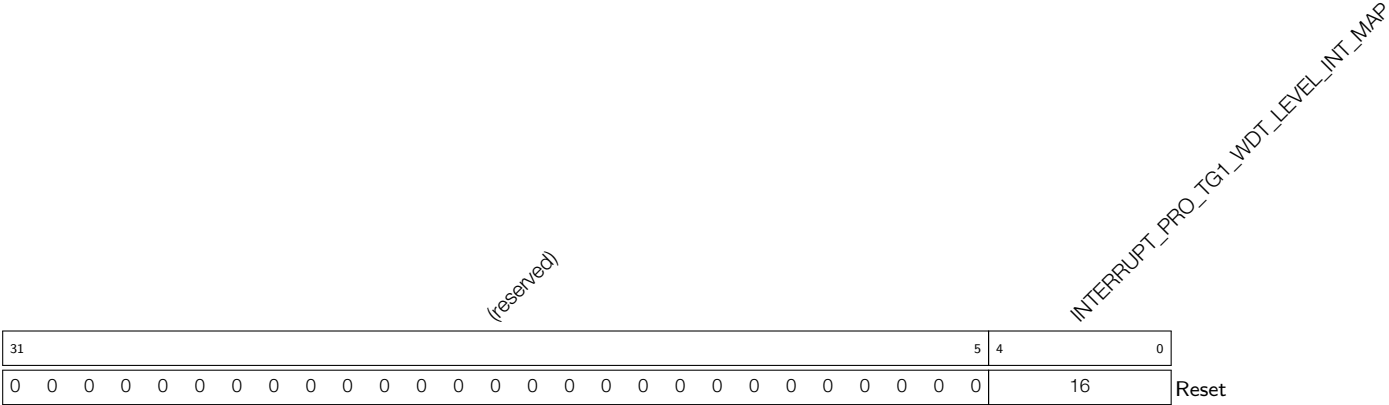
### 反馈文档意见

## 53



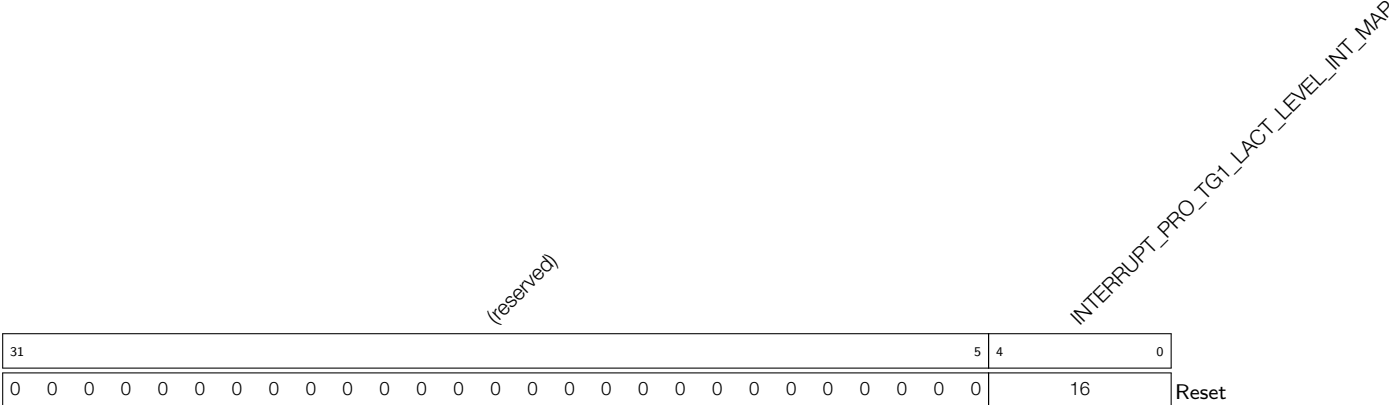
**INTERRUPT\_PRO\_TG1\_T1\_LEVEL\_INT\_MAP** 用于将 TG1\_T1\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.22: INTERRUPT\_PRO\_TG1\_WDT\_LEVEL\_INT\_MAP\_REG (0x0054)



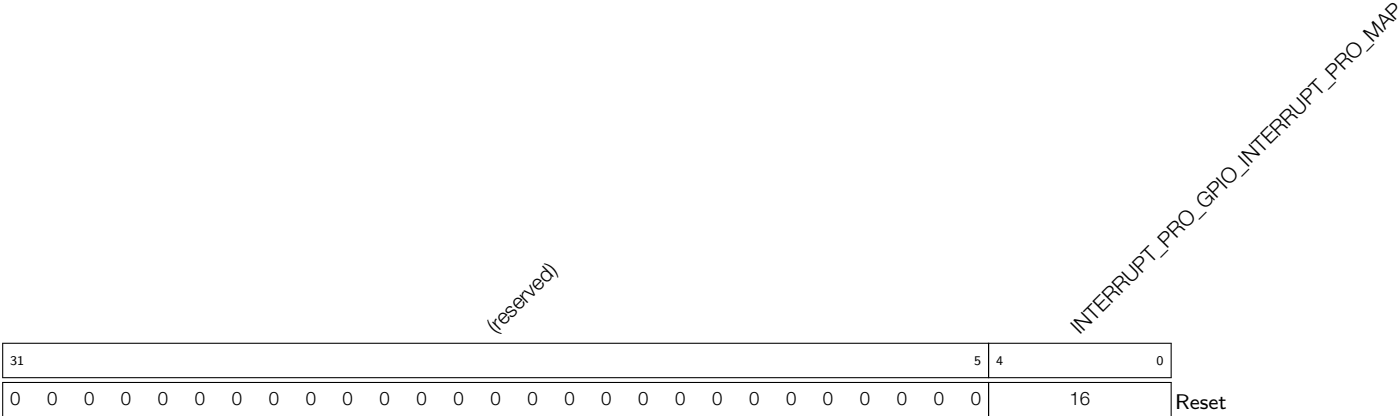
**INTERRUPT\_PRO\_TG1\_WDT\_LEVEL\_INT\_MAP** 用于将 TG1\_WDT\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.23: INTERRUPT\_PRO\_TG1\_LACT\_LEVEL\_INT\_MAP\_REG (0x0058)



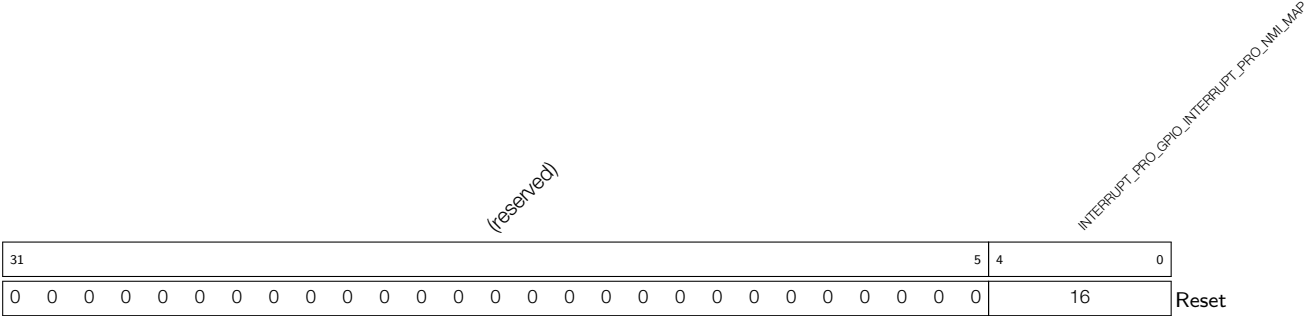
**INTERRUPT\_PRO\_TG1\_LACT\_LEVEL\_INT\_MAP** 用于将 TG1\_LACT\_LEVEL\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.24: INTERRUPT\_PRO\_GPIO\_INTERRUPT\_PRO\_MAP\_REG (0x005C)



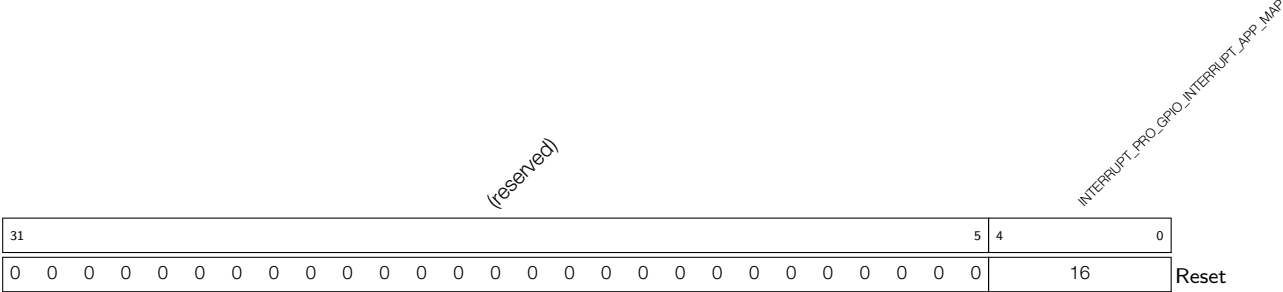
**INTERRUPT\_PRO\_GPIO\_INTERRUPT\_PRO\_MAP** 用于将 GPIO\_INTERRUPT\_PRO 中断信号映射至 CPU 中断。(读/写)

Register 4.25: INTERRUPT\_PRO\_GPIO\_INTERRUPT\_PRO\_NMI\_MAP\_REG (0x0060)



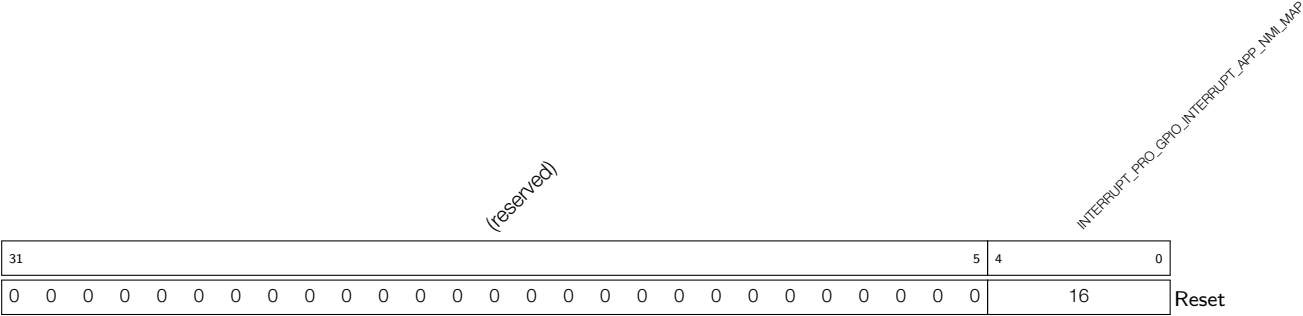
**INTERRUPT\_PRO\_GPIO\_INTERRUPT\_PRO\_NMI\_MAP** 用于将 GPIO\_INTERRUPT\_PRO\_NMI 中断信号映射至 CPU 中断。(读/写)

Register 4.26: INTERRUPT\_PRO\_GPIO\_INTERRUPT\_APP\_MAP\_REG (0x0064)



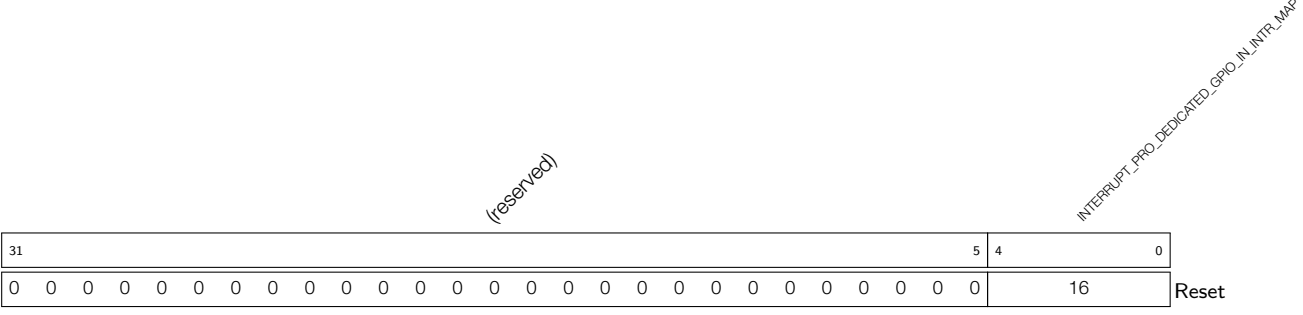
**INTERRUPT\_PRO\_GPIO\_INTERRUPT\_APP\_MAP** 用于将 GPIO\_INTERRUPT\_APP 中断信号映射至 CPU 中断。(读/写)

Register 4.27: INTERRUPT\_PRO\_GPIO\_INTERRUPT\_APP\_NMI\_MAP\_REG (0x0068)



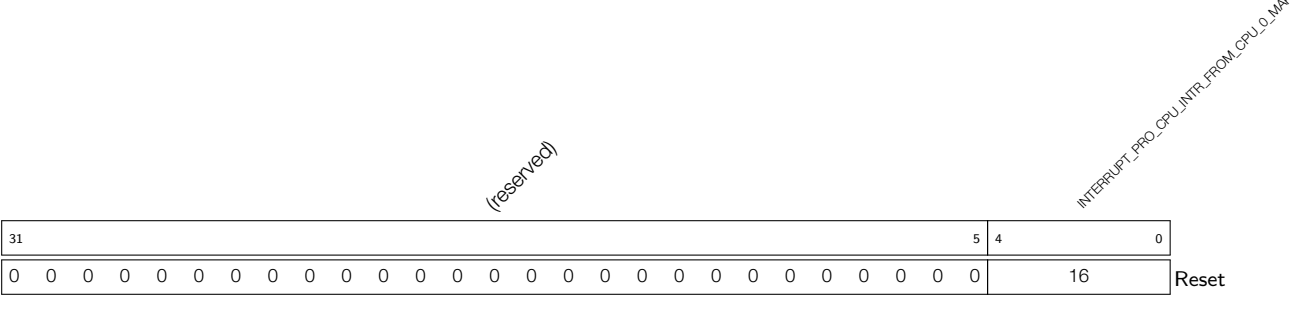
**INTERRUPT\_PRO\_GPIO\_INTERRUPT\_APP\_NMI\_MAP** 用于将 GPIO\_INTERRUPT\_APP 中断信号映射至 CPU 中断。(读/写)

Register 4.28: INTERRUPT\_PRO\_DEDICATED\_GPIO\_IN\_INTR\_MAP\_REG (0x006C)



**INTERRUPT\_PRO\_DEDICATED\_GPIO\_IN\_INTR\_MAP** 用于将 DEDICATED\_GPIO\_IN\_INTR 中断信号映射至 CPU 中断。(读/写)

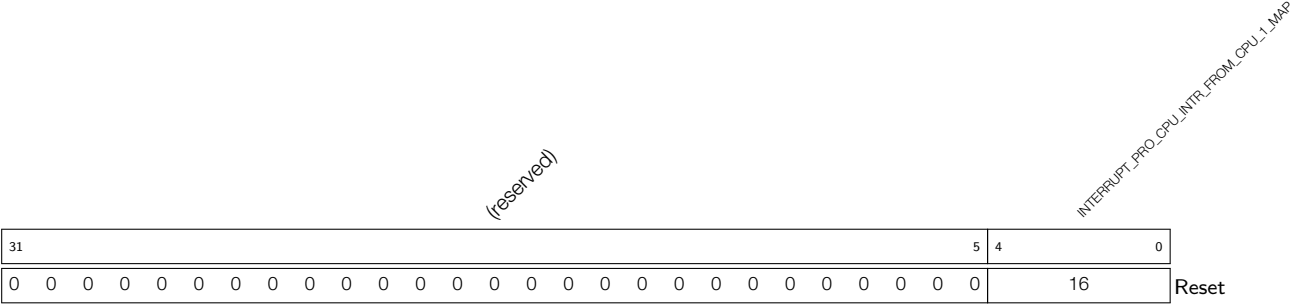
Register 4.29: INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_0\_MAP\_REG (0x0070)



**INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_0\_MAP** 用于将 CPU\_INTR\_FROM\_CPU\_0 中断信号映射至 CPU 中断。(读/写)

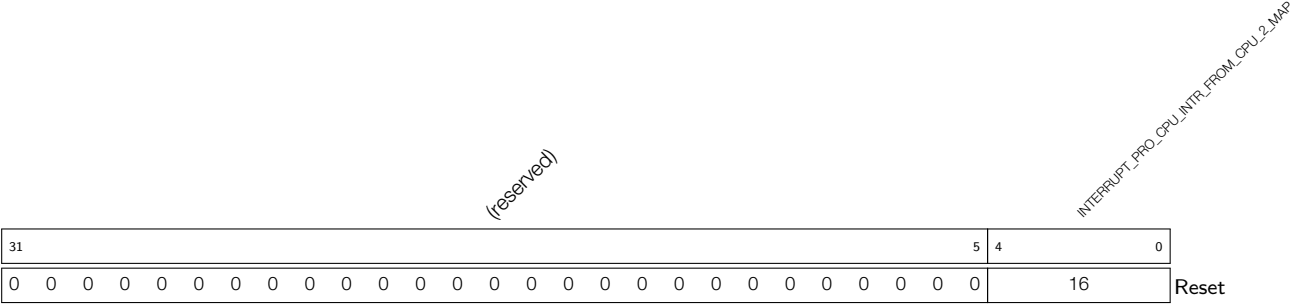


Register 4.30: INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_1\_MAP\_REG (0x0074)



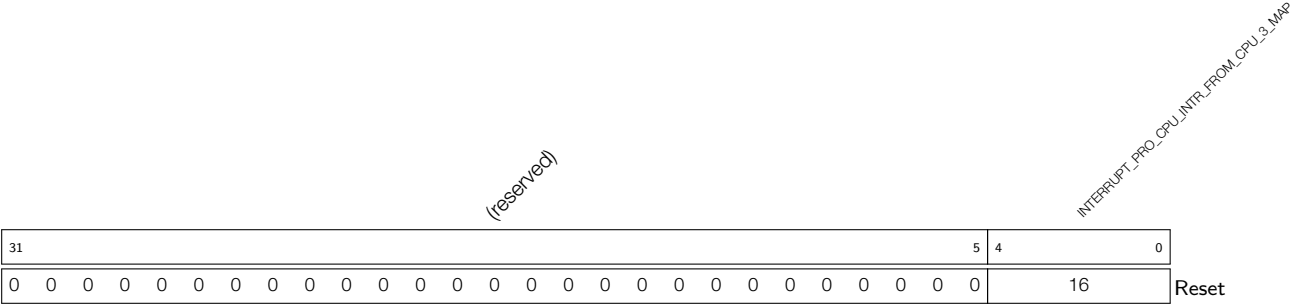
**INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_1\_MAP** 用于将 CPU\_INTR\_FROM\_CPU\_1 中断信号映射至 CPU 中断。(读/写)

Register 4.31: INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_2\_MAP\_REG (0x0078)



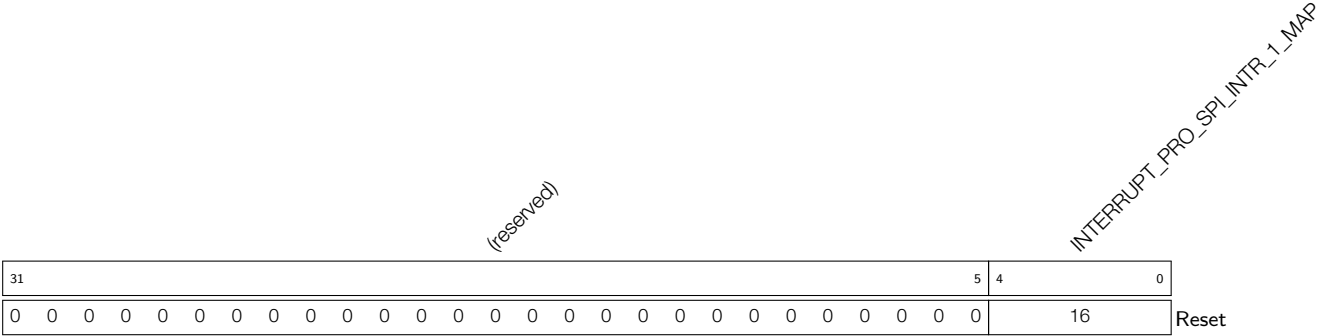
**INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_2\_MAP** 用于将 CPU\_INTR\_FROM\_CPU\_2 中断信号映射至 CPU 中断。(读/写)

Register 4.32: INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_3\_MAP\_REG (0x007C)



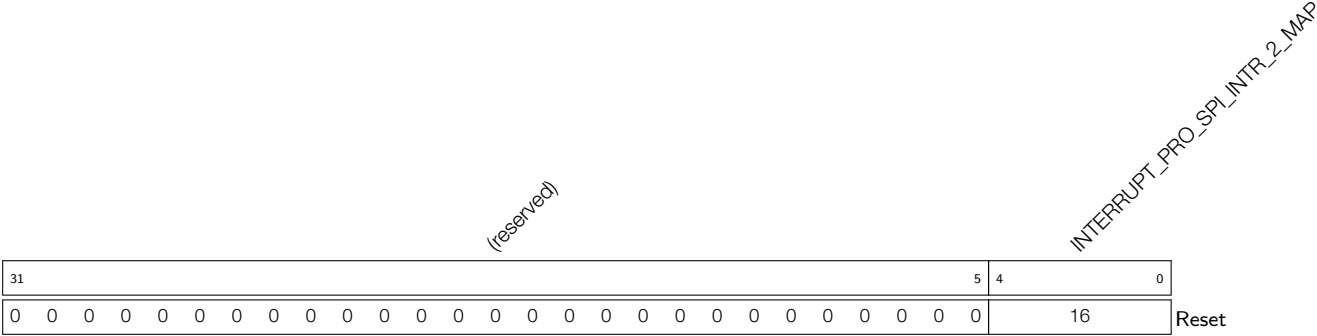
**INTERRUPT\_PRO\_CPU\_INTR\_FROM\_CPU\_3\_MAP** 用于将 CPU\_INTR\_FROM\_CPU\_3 中断信号映射至 CPU 中断。(读/写)

Register 4.33: INTERRUPT\_PRO\_SPI\_INTR\_1\_MAP\_REG (0x0080)



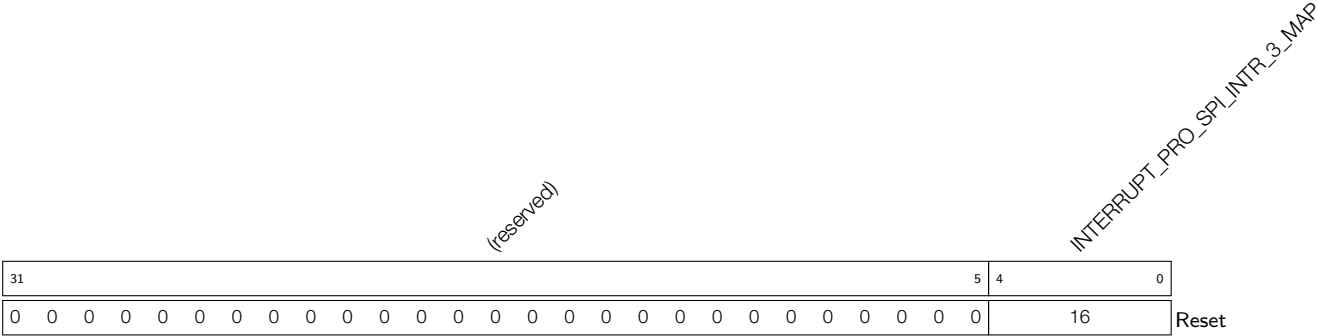
INTERRUPT\_PRO\_SPI\_INTR\_1\_MAP 用于将 SPI\_INTR\_1 中断信号映射至 CPU 中断。(读/写)

Register 4.34: INTERRUPT\_PRO\_SPI\_INTR\_2\_MAP\_REG (0x0084)



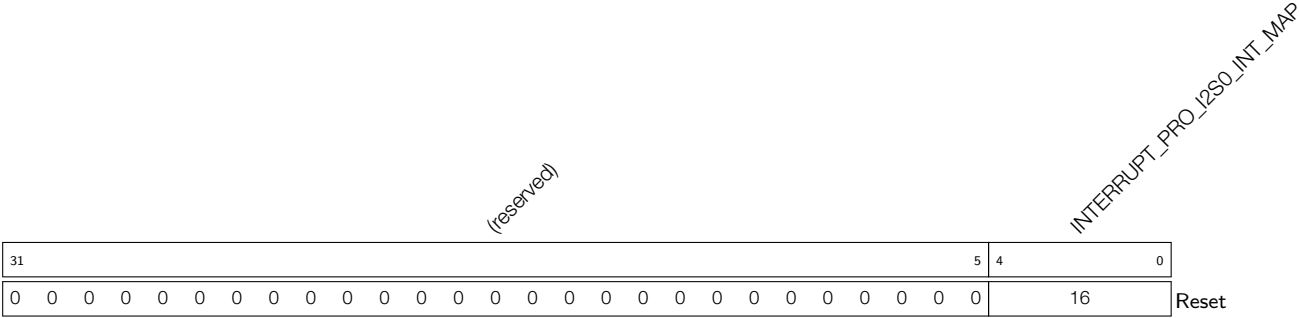
INTERRUPT\_PRO\_SPI\_INTR\_2\_MAP 用于将 SPI\_INTR\_2 中断信号映射至 CPU 中断。(读/写)

Register 4.35: INTERRUPT\_PRO\_SPI\_INTR\_3\_MAP\_REG (0x0088)



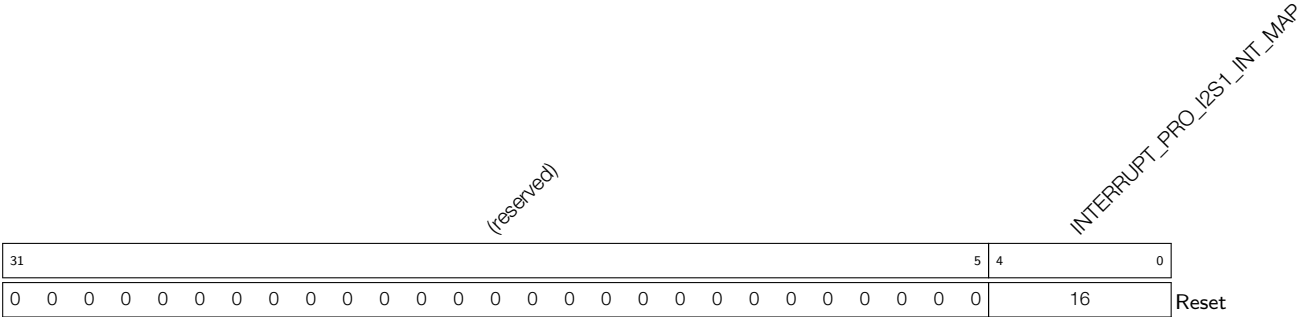
INTERRUPT\_PRO\_SPI\_INTR\_3\_MAP 用于将 SPI\_INTR\_3 中断信号映射至 CPU 中断。(读/写)

Register 4.36: INTERRUPT\_PRO\_I2S0\_INT\_MAP\_REG (0x008C)



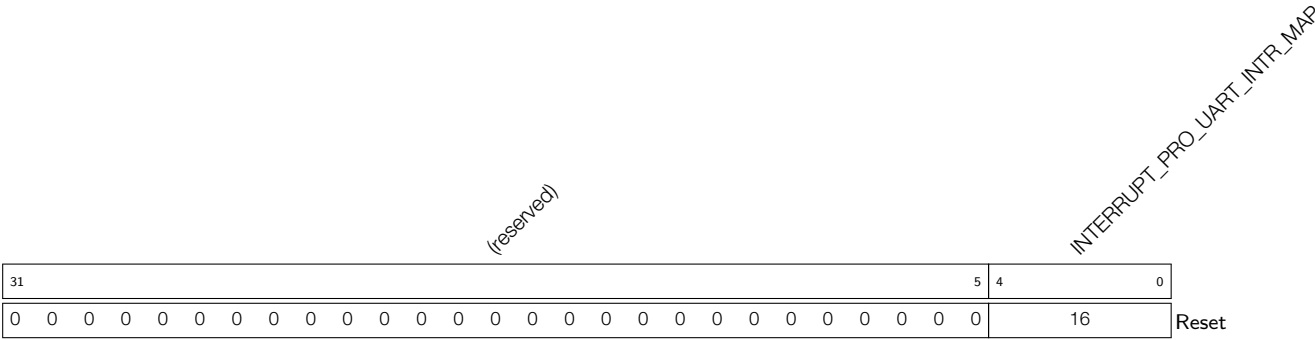
**INTERRUPT\_PRO\_I2S0\_INT\_MAP** 用于将 I2S0\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.37: INTERRUPT\_PRO\_I2S1\_INT\_MAP\_REG (0x0090)



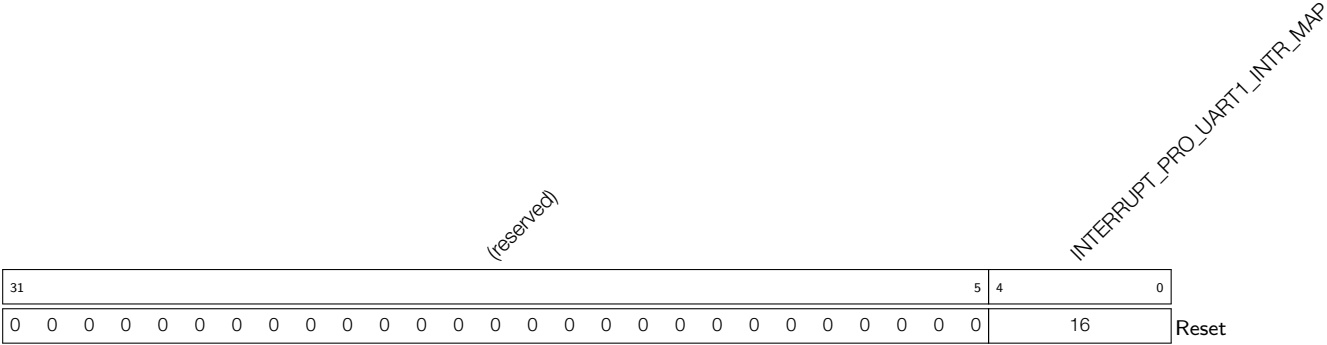
**INTERRUPT\_PRO\_I2S1\_INT\_MAP** 用于将 I2S1\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.38: INTERRUPT\_PRO\_UART\_INTR\_MAP\_REG (0x0094)



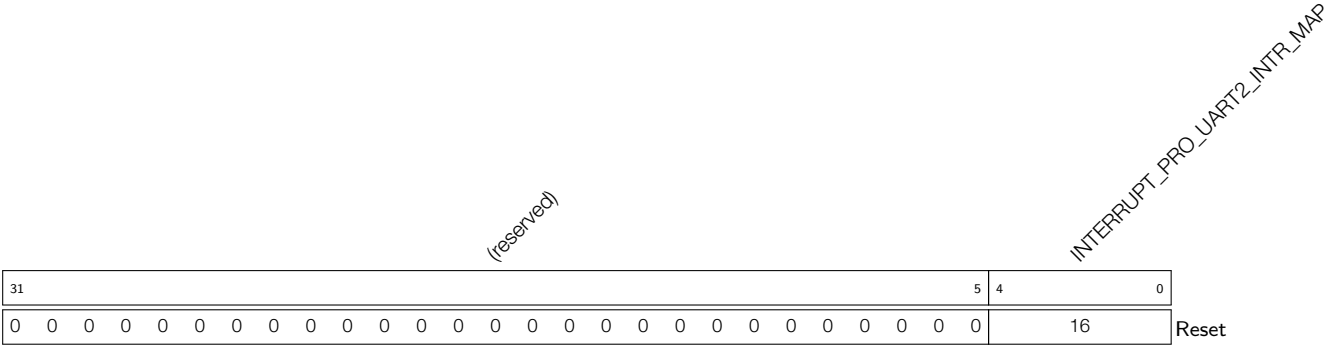
**INTERRUPT\_PRO\_UART\_INTR\_MAP** 用于将 UART\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.39: INTERRUPT\_PRO\_UART1\_INTR\_MAP\_REG (0x0098)



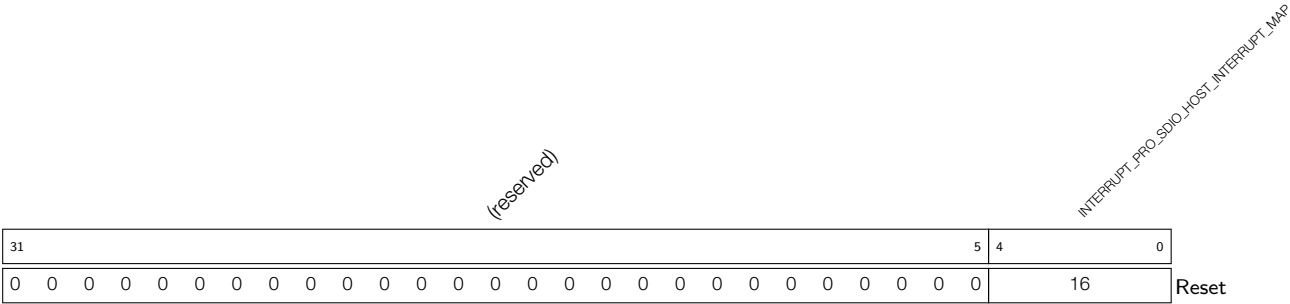
**INTERRUPT\_PRO\_UART1\_INTR\_MAP** 用于将 UART1\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.40: INTERRUPT\_PRO\_UART2\_INTR\_MAP\_REG (0x009C)



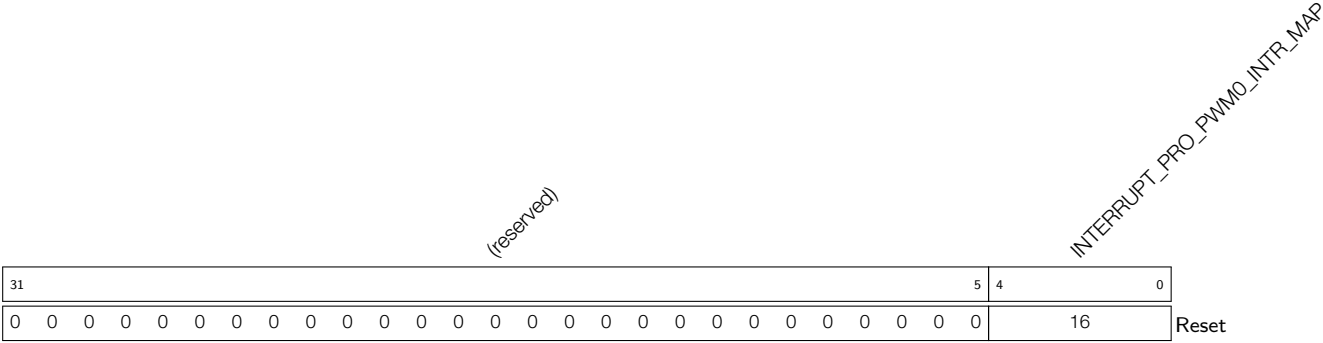
**INTERRUPT\_PRO\_UART2\_INTR\_MAP** 用于将 UART2\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.41: INTERRUPT\_PRO\_SDIO\_HOST\_INTERRUPT\_MAP\_REG (0x00A0)



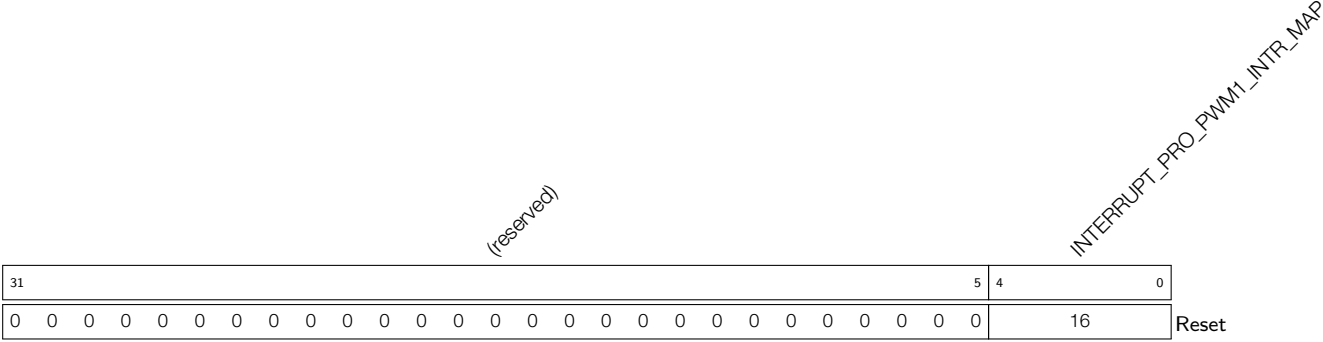
**INTERRUPT\_PRO\_SDIO\_HOST\_INTERRUPT\_MAP** 用于将 SDIO\_HOST 中断信号映射至 CPU 中断。(读/写)

Register 4.42: INTERRUPT\_PRO\_PWM0\_INTR\_MAP\_REG (0x00A4)



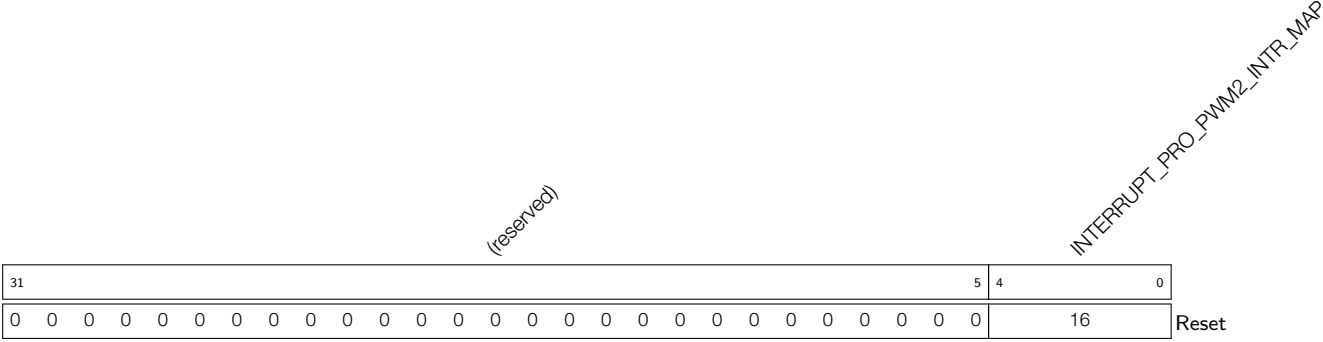
**INTERRUPT\_PRO\_PWM0\_INTR\_MAP** 用于将 PWM0\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.43: INTERRUPT\_PRO\_PWM1\_INTR\_MAP\_REG (0x00A8)



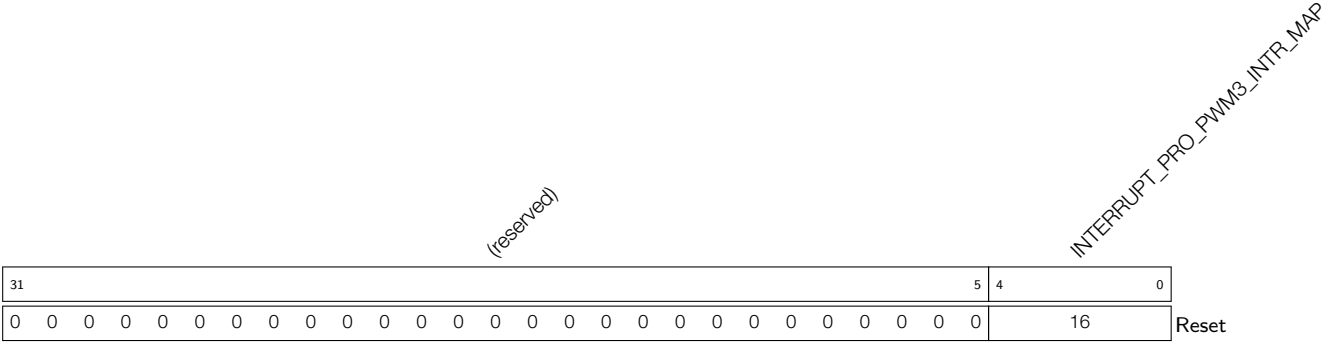
**INTERRUPT\_PRO\_PWM1\_INTR\_MAP** 用于将 PWM1\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.44: INTERRUPT\_PRO\_PWM2\_INTR\_MAP\_REG (0x00AC)



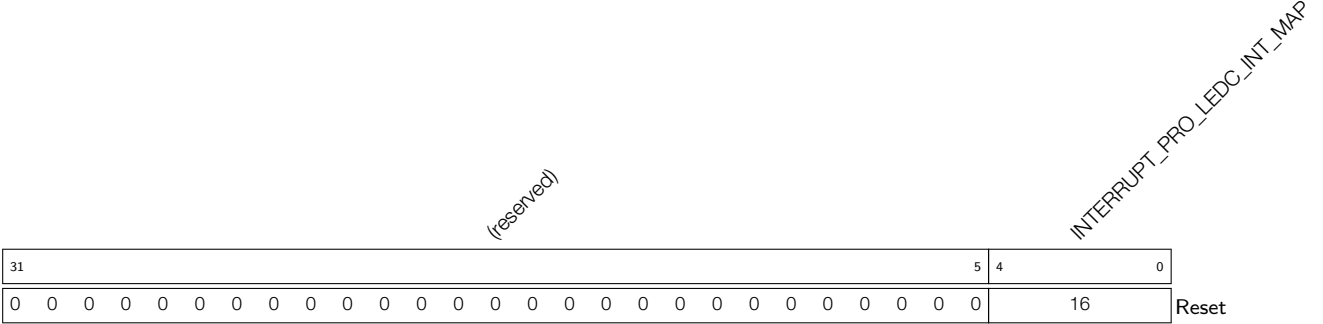
**INTERRUPT\_PRO\_PWM2\_INTR\_MAP** 用于将 PWM2\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.45: INTERRUPT\_PRO\_PWM3\_INTR\_MAP\_REG (0x00B0)



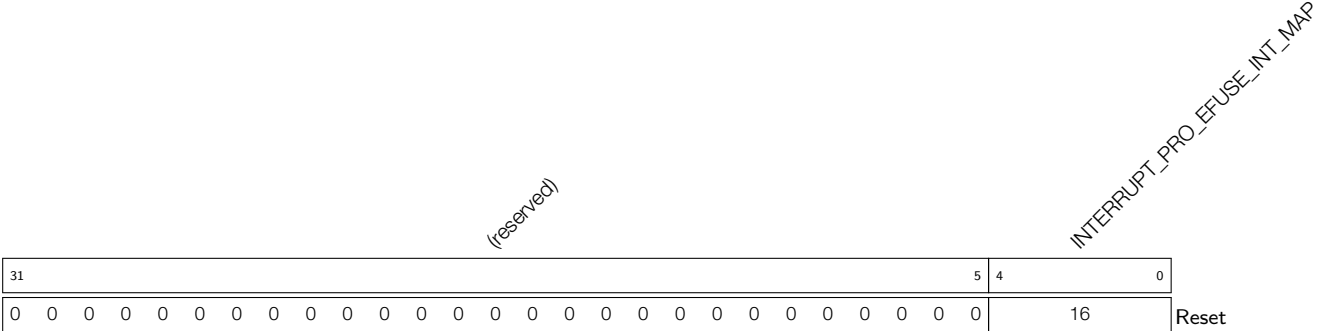
**INTERRUPT\_PRO\_PWM3\_INTR\_MAP** 用于将 PWM3\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.46: INTERRUPT\_PRO\_LEDC\_INT\_MAP\_REG (0x00B4)



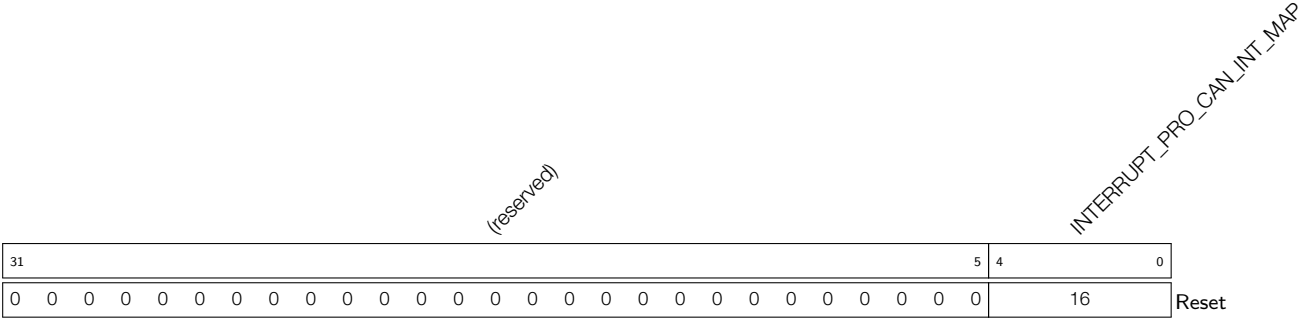
**INTERRUPT\_PRO\_LEDC\_INT\_MAP** 用于将 LEDC\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.47: INTERRUPT\_PRO\_EFUSE\_INT\_MAP\_REG (0x00B8)



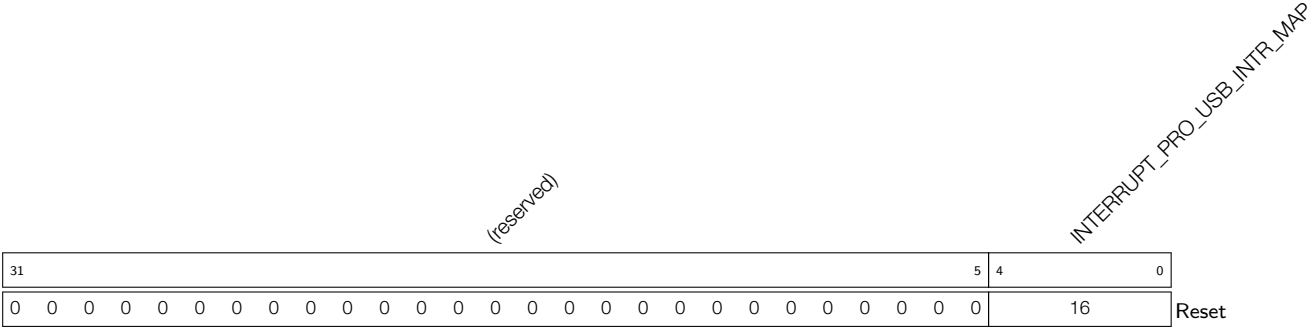
**INTERRUPT\_PRO\_EFUSE\_INT\_MAP** 用于将 EFUSE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.48: INTERRUPT\_PRO\_CAN\_INT\_MAP\_REG (0x00BC)



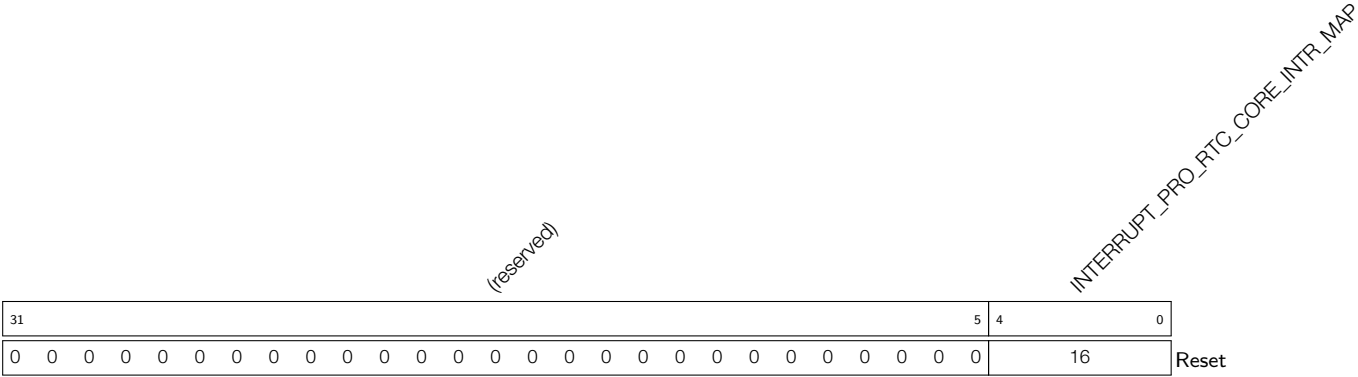
**INTERRUPT\_PRO\_CAN\_INT\_MAP** 用于将 CAN\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.49: INTERRUPT\_PRO\_USB\_INTR\_MAP\_REG (0x00C0)



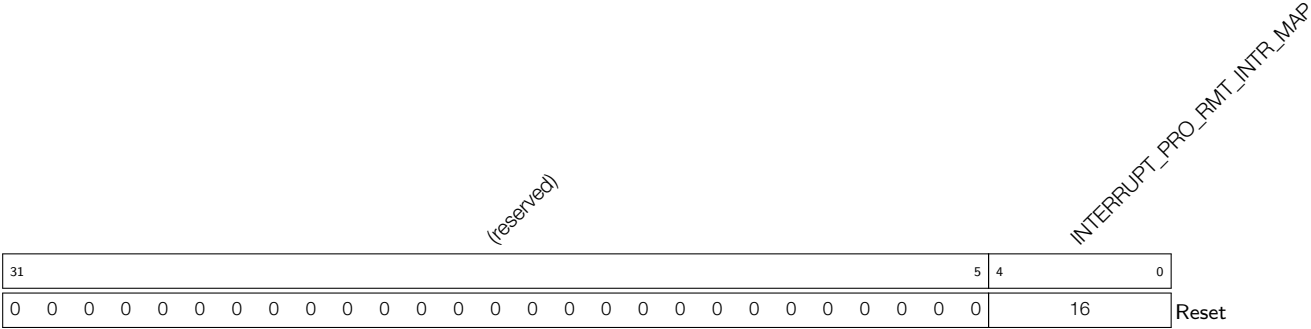
**INTERRUPT\_PRO\_USB\_INTR\_MAP** 用于将 USB\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.50: INTERRUPT\_PRO\_RTC\_CORE\_INTR\_MAP\_REG (0x00C4)



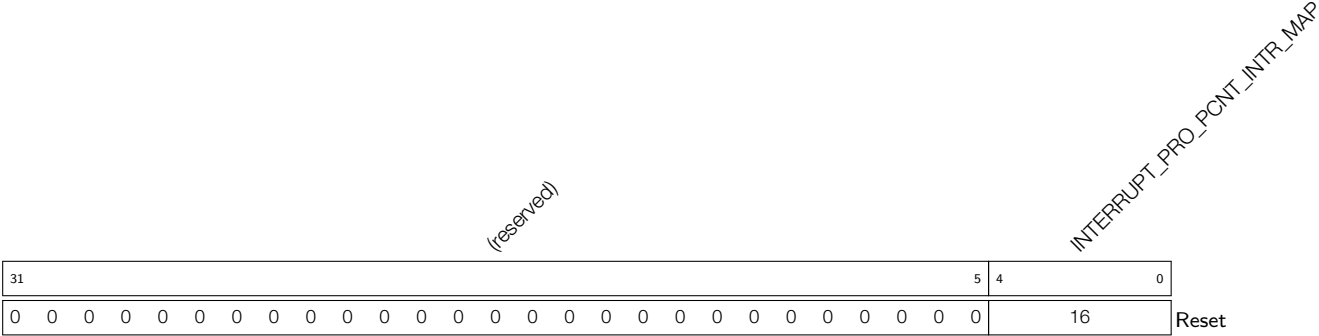
**INTERRUPT\_PRO\_RTC\_CORE\_INTR\_MAP** 用于将 RTC\_CORE\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.51: INTERRUPT\_PRO\_RMT\_INTR\_MAP\_REG (0x00C8)



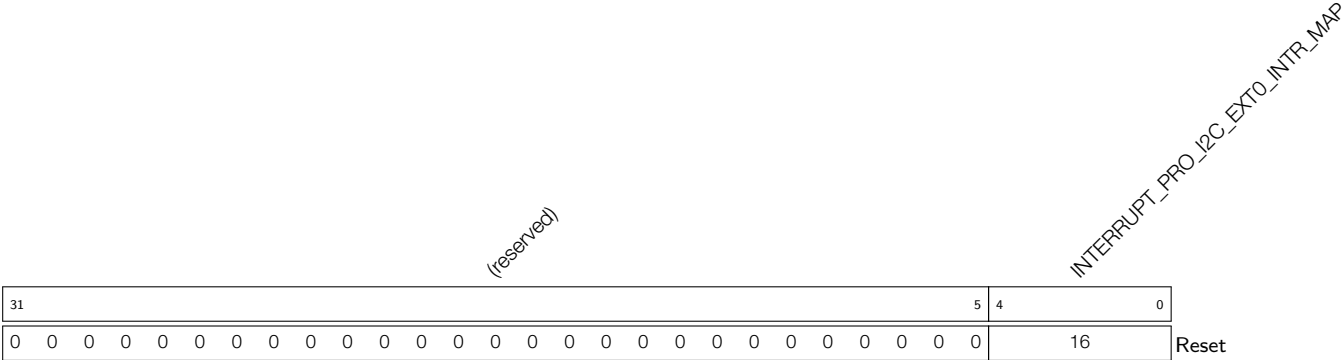
INTERRUPT\_PRO\_RMT\_INTR\_MAP 用于将 RMT\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.52: INTERRUPT\_PRO\_PCNT\_INTR\_MAP\_REG (0x00CC)



INTERRUPT\_PRO\_PCNT\_INTR\_MAP 用于将 PCNT\_INTR 中断信号映射至 CPU 中断。(读/写)

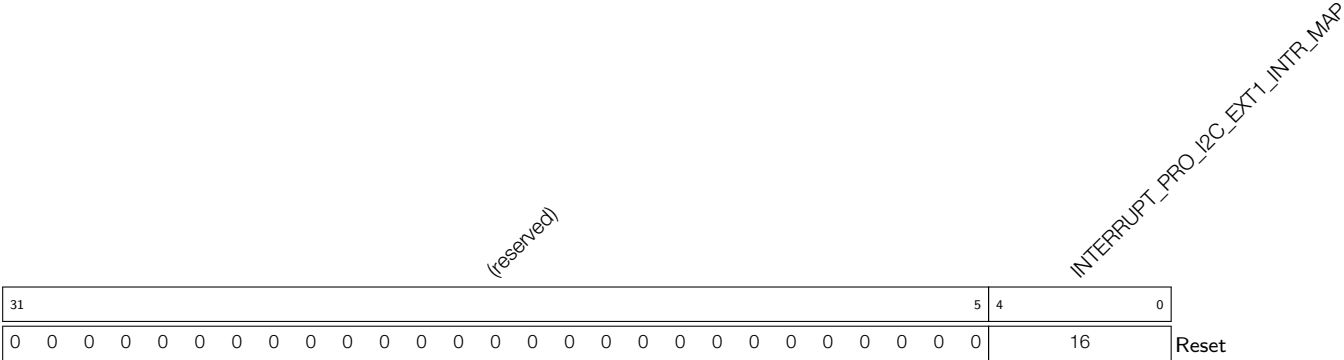
Register 4.53: INTERRUPT\_PRO\_I2C\_EXT0\_INTR\_MAP\_REG (0x00D0)



INTERRUPT\_PRO\_I2C\_EXT0\_INTR\_MAP 用于将 I2C\_EXT0\_INTR 中断信号映射至 CPU 中断。(读/写)

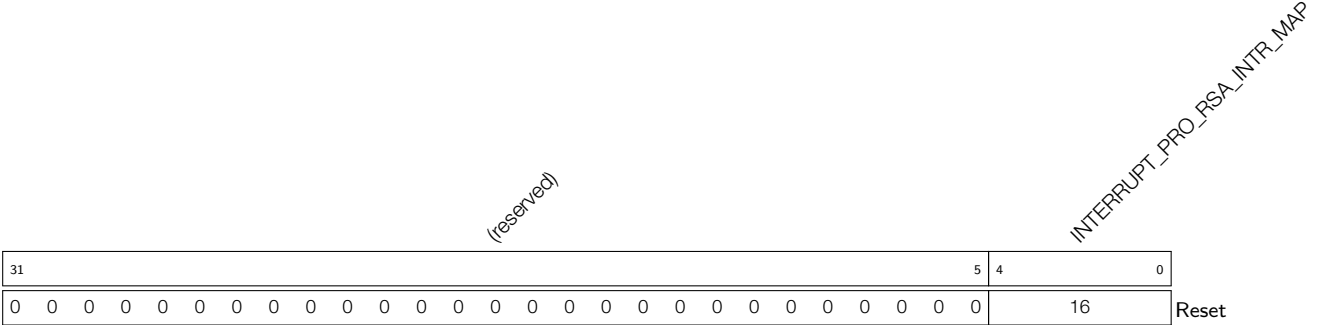


Register 4.54: INTERRUPT\_PRO\_I2C\_EXT1\_INTR\_MAP\_REG (0x00D4)



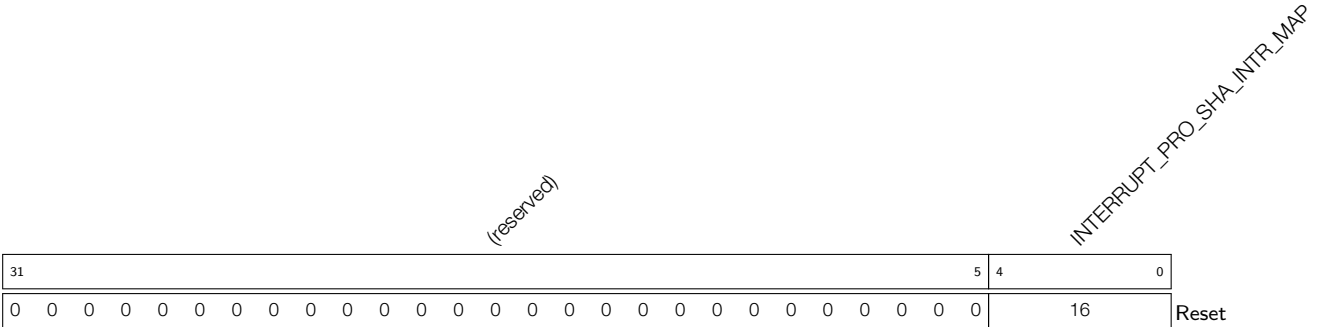
INTERRUPT\_PRO\_I2C\_EXT1\_INTR\_MAP 用于将 I2C\_EXT1\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.55: INTERRUPT\_PRO\_RSA\_INTR\_MAP\_REG (0x00D8)



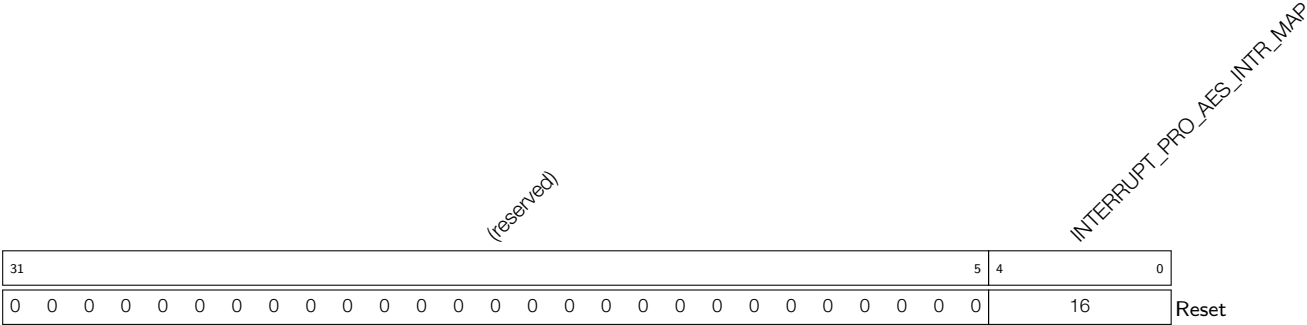
INTERRUPT\_PRO\_RSA\_INTR\_MAP 用于将 RSA\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.56: INTERRUPT\_PRO\_SHA\_INTR\_MAP\_REG (0x00DC)



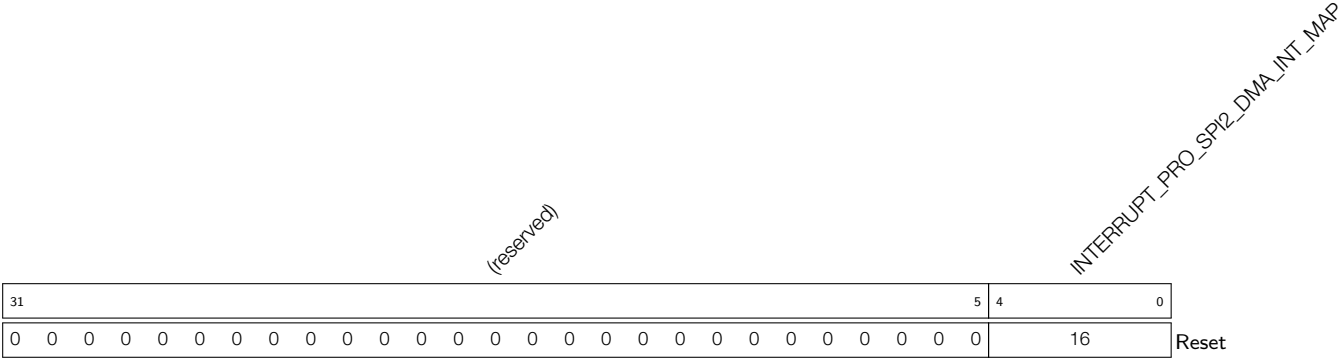
INTERRUPT\_PRO\_SHA\_INTR\_MAP 用于将 SHA\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.57: INTERRUPT\_PRO\_AES\_INTR\_MAP\_REG (0x00E0)



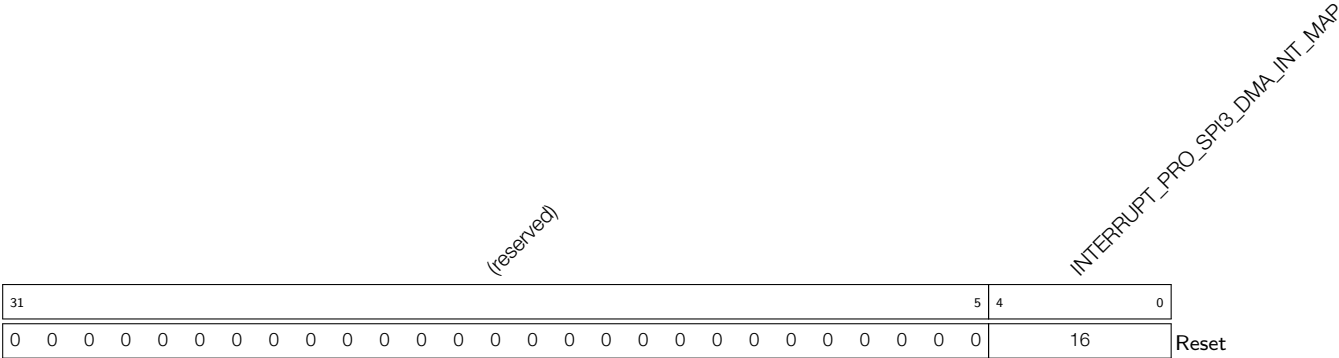
INTERRUPT\_PRO\_AES\_INTR\_MAP 用于将 AES\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.58: INTERRUPT\_PRO\_SPI2\_DMA\_INT\_MAP\_REG (0x00E4)



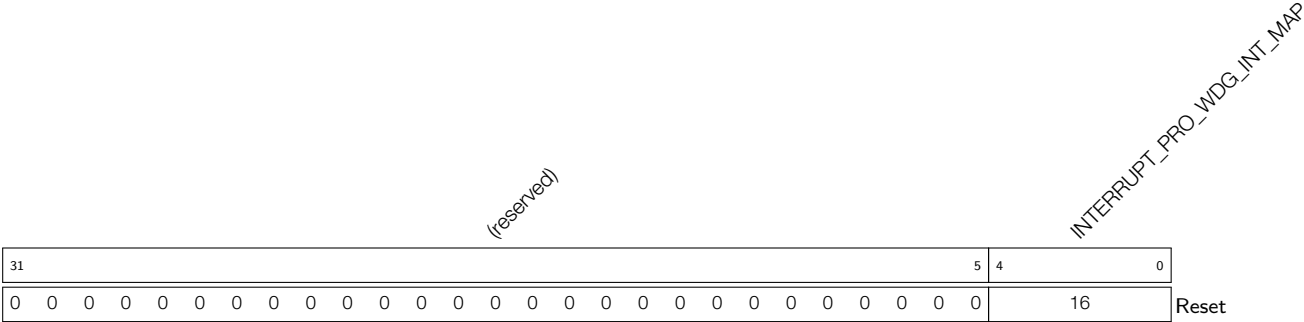
INTERRUPT\_PRO\_SPI2\_DMA\_INT\_MAP 用于将 SPI2\_DMA\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.59: INTERRUPT\_PRO\_SPI3\_DMA\_INT\_MAP\_REG (0x00E8)



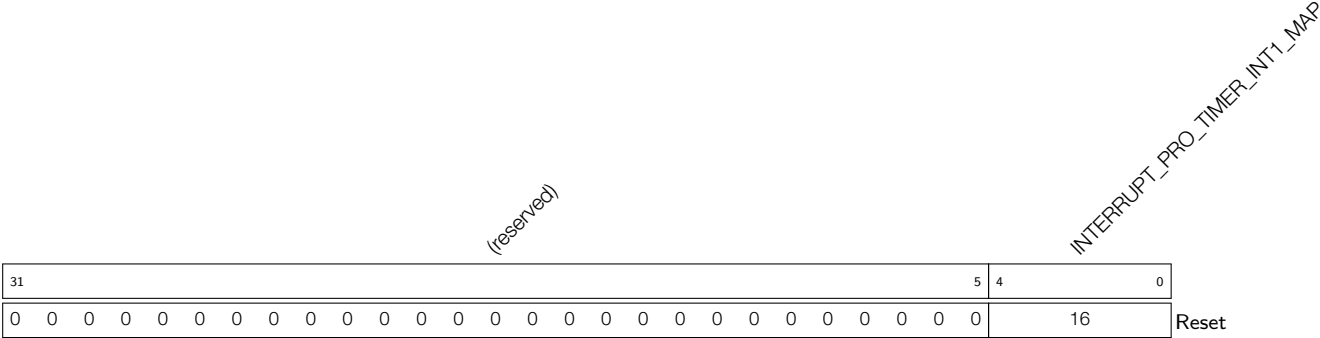
INTERRUPT\_PRO\_SPI3\_DMA\_INT\_MAP 用于将 SPI3\_DMA\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.60: INTERRUPT\_PRO\_WDG\_INT\_MAP\_REG (0x00EC)



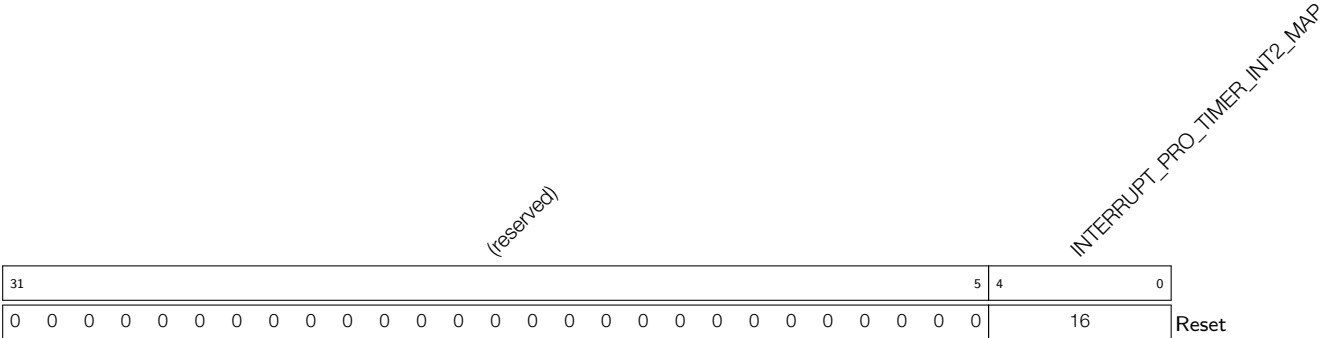
INTERRUPT\_PRO\_WDG\_INT\_MAP 用于将 WDG\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.61: INTERRUPT\_PRO\_TIMER\_INT1\_MAP\_REG (0x00F0)



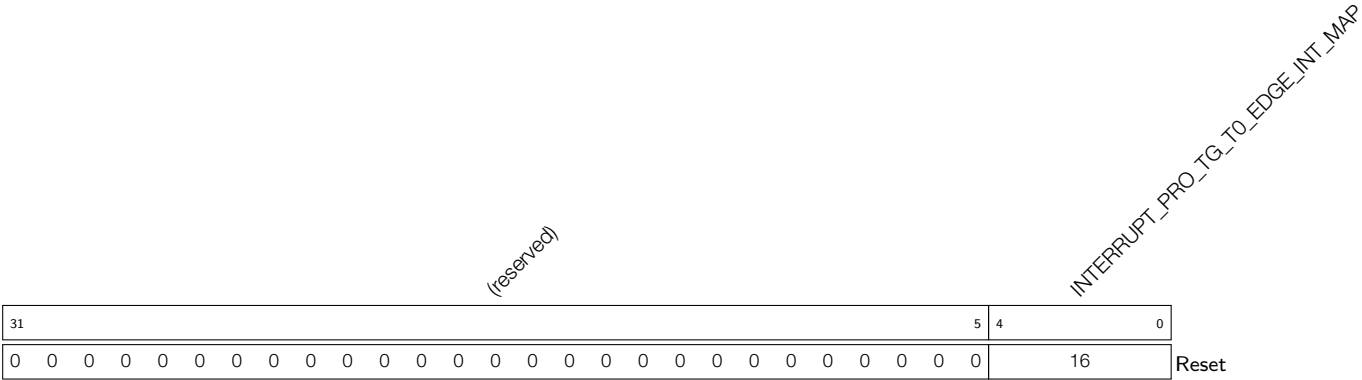
INTERRUPT\_PRO\_TIMER\_INT1\_MAP 用于将 TIMER\_INT1 中断信号映射至 CPU 中断。(读/写)

Register 4.62: INTERRUPT\_PRO\_TIMER\_INT2\_MAP\_REG (0x00F4)



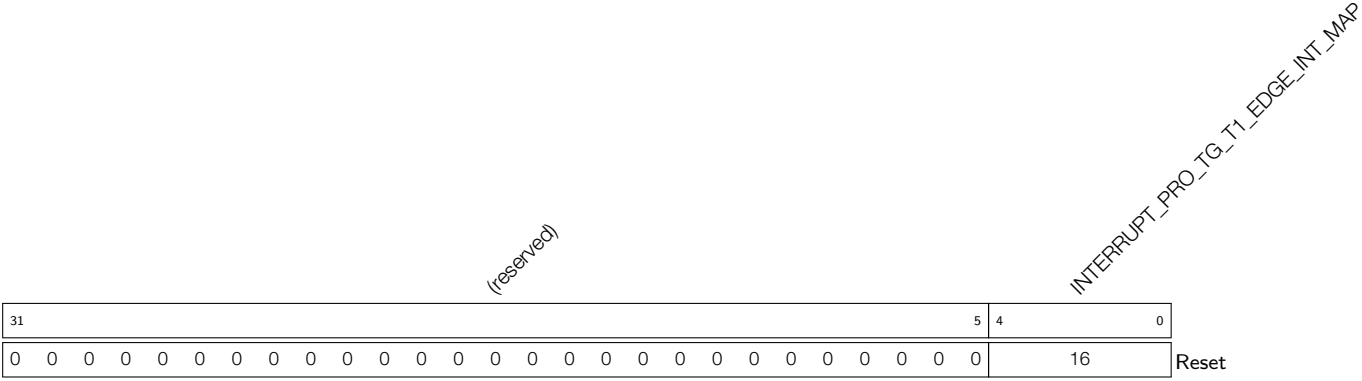
INTERRUPT\_PRO\_TIMER\_INT2\_MAP 用于将 TIMER\_INT2 中断信号映射至 CPU 中断。(读/写)

Register 4.63: INTERRUPT\_PRO\_TG\_T0\_EDGE\_INT\_MAP\_REG (0x00F8)



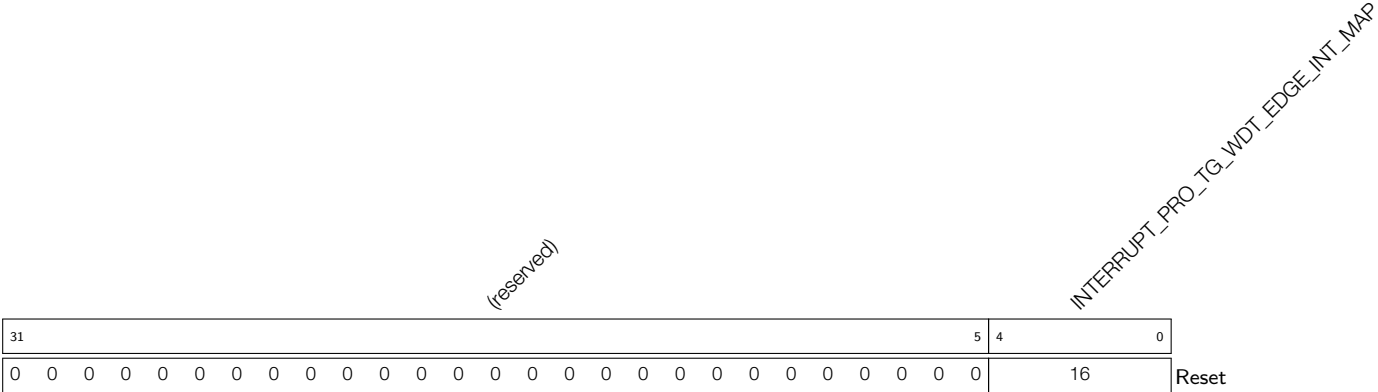
**INTERRUPT\_PRO\_TG\_T0\_EDGE\_INT\_MAP** 用于将 TG\_T0\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.64: INTERRUPT\_PRO\_TG\_T1\_EDGE\_INT\_MAP\_REG (0x00FC)



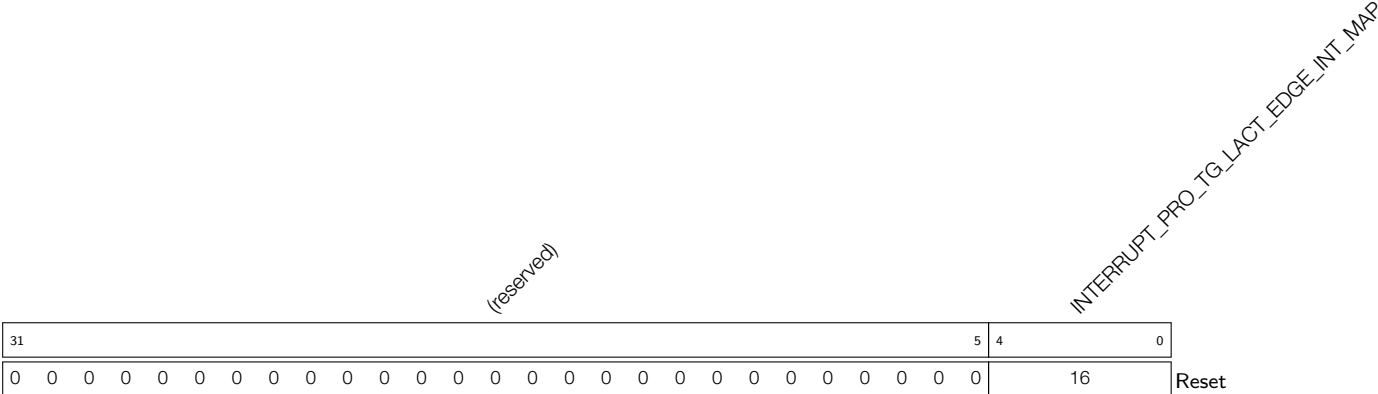
**INTERRUPT\_PRO\_TG\_T1\_EDGE\_INT\_MAP** 用于将 TG\_T1\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.65: INTERRUPT\_PRO\_TG\_WDT\_EDGE\_INT\_MAP\_REG (0x0100)



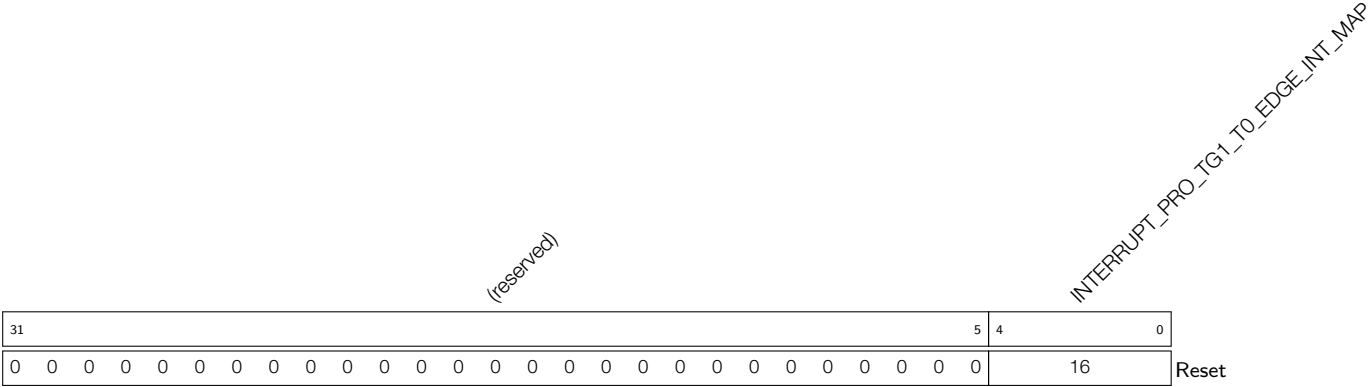
**INTERRUPT\_PRO\_TG\_WDT\_EDGE\_INT\_MAP** 用于将 TG\_WDT\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.66: INTERRUPT\_PRO\_TG\_LACT\_EDGE\_INT\_MAP\_REG (0x0104)



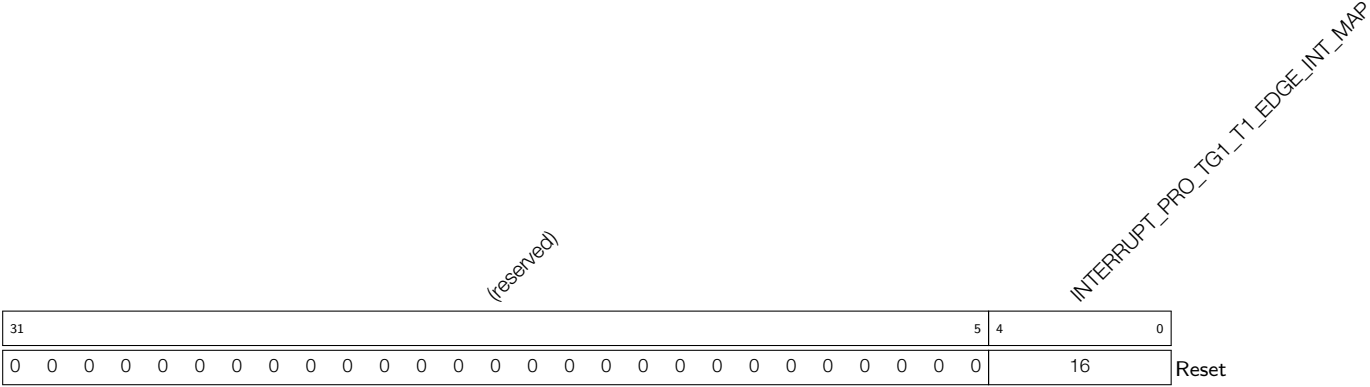
**INTERRUPT\_PRO\_TG\_LACT\_EDGE\_INT\_MAP** 用于将 TG\_LACT\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.67: INTERRUPT\_PRO\_TG1\_T0\_EDGE\_INT\_MAP\_REG (0x0108)



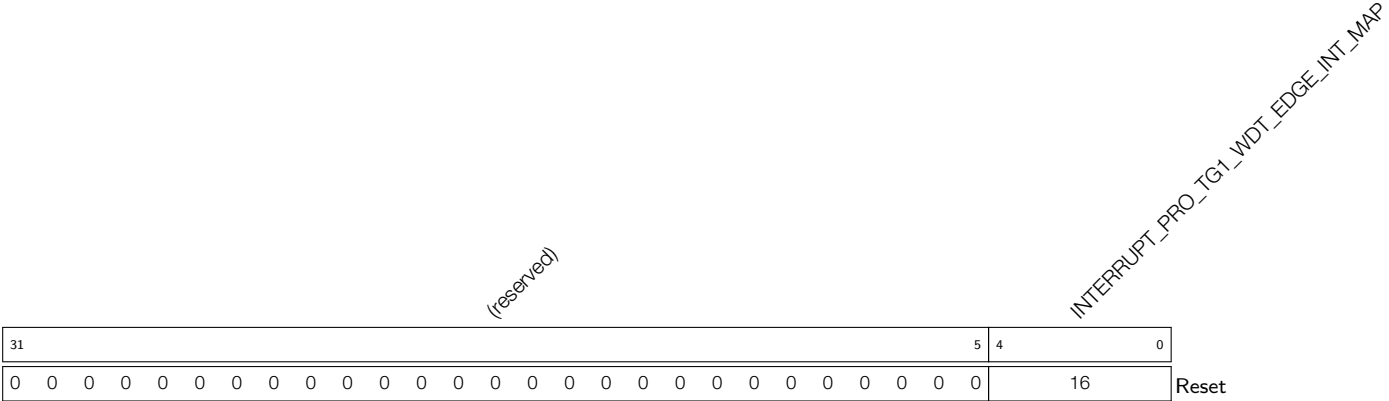
**INTERRUPT\_PRO\_TG1\_T0\_EDGE\_INT\_MAP** 用于将 TG1\_T0\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.68: INTERRUPT\_PRO\_TG1\_T1\_EDGE\_INT\_MAP\_REG (0x010C)



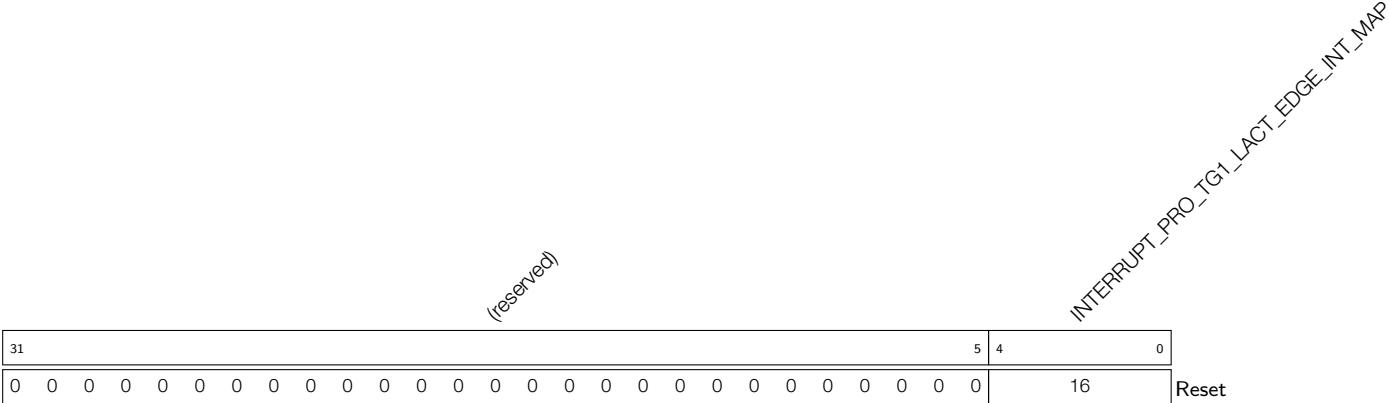
**INTERRUPT\_PRO\_TG1\_T1\_EDGE\_INT\_MAP** 用于将 TG1\_T1\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.69: INTERRUPT\_PRO\_TG1\_WDT\_EDGE\_INT\_MAP\_REG (0x0110)



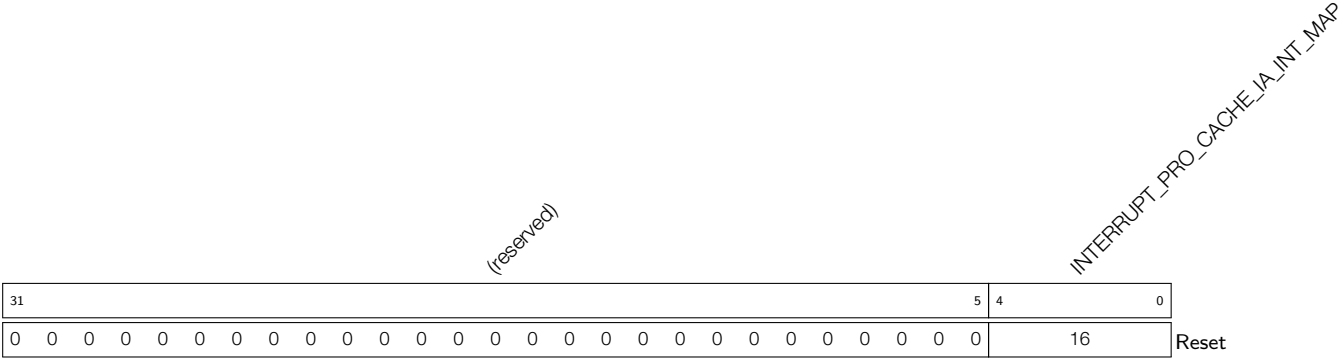
**INTERRUPT\_PRO\_TG1\_WDT\_EDGE\_INT\_MAP** 用于将 TG1\_WDT\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.70: INTERRUPT\_PRO\_TG1\_LACT\_EDGE\_INT\_MAP\_REG (0x0114)



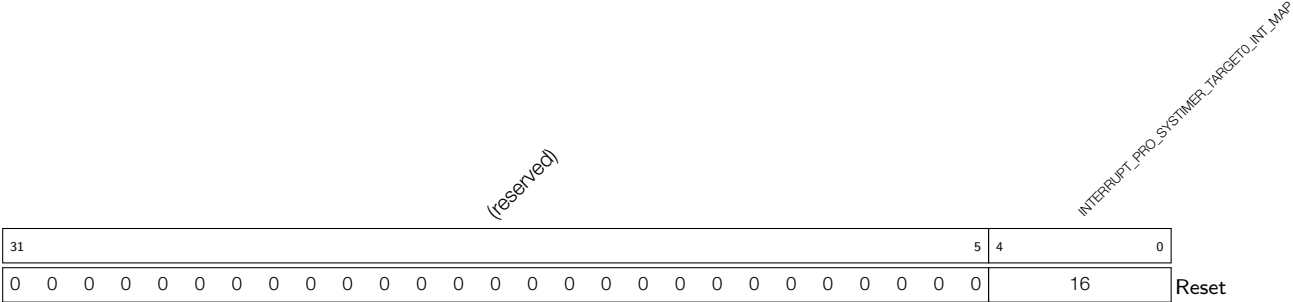
**INTERRUPT\_PRO\_TG1\_LACT\_EDGE\_INT\_MAP** 用于将 TG1\_LACT\_EDGE\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.71: INTERRUPT\_PRO\_CACHE\_IA\_INT\_MAP\_REG (0x0118)



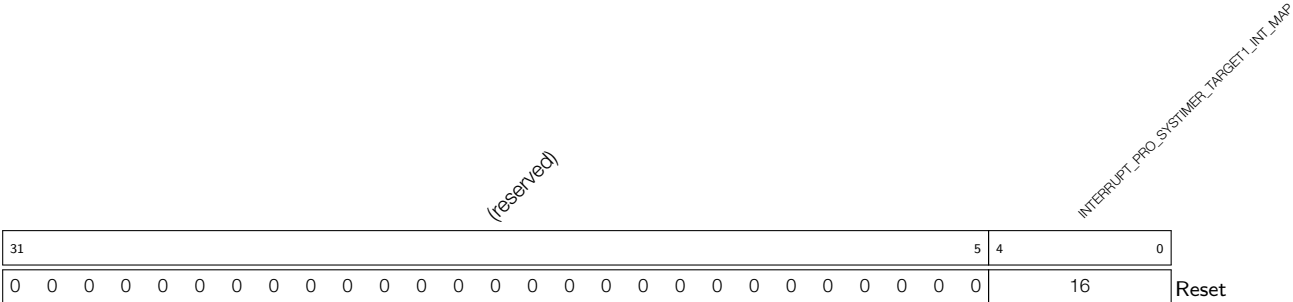
INTERRUPT\_PRO\_CACHE\_IA\_INT\_MAP 用于将 CACHE\_IA\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.72: INTERRUPT\_PRO\_SYSTIMER\_TARGET0\_INT\_MAP\_REG (0x011C)



INTERRUPT\_PRO\_SYSTIMER\_TARGET0\_INT\_MAP 用于将 SYSTIMER\_TARGET0 中断信号映射至 CPU 中断。(读/写)

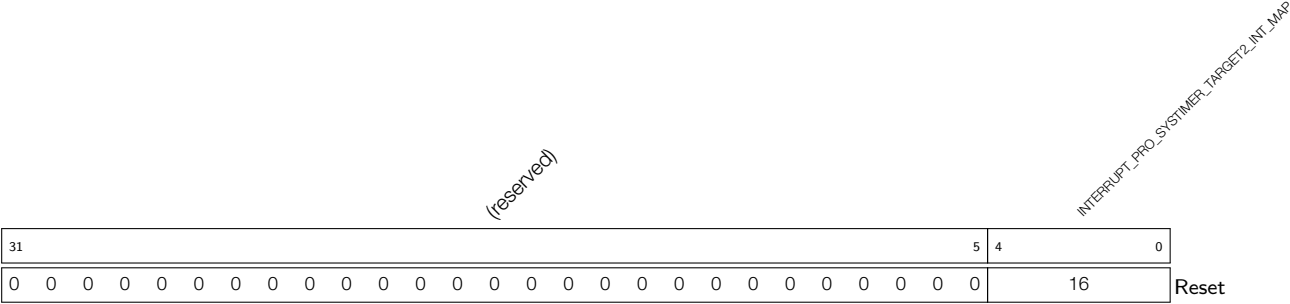
Register 4.73: INTERRUPT\_PRO\_SYSTIMER\_TARGET1\_INT\_MAP\_REG (0x0120)



INTERRUPT\_PRO\_SYSTIMER\_TARGET1\_INT\_MAP 用于将 SYSTIMER\_TARGET1 中断信号映射至 CPU 中断。(读/写)

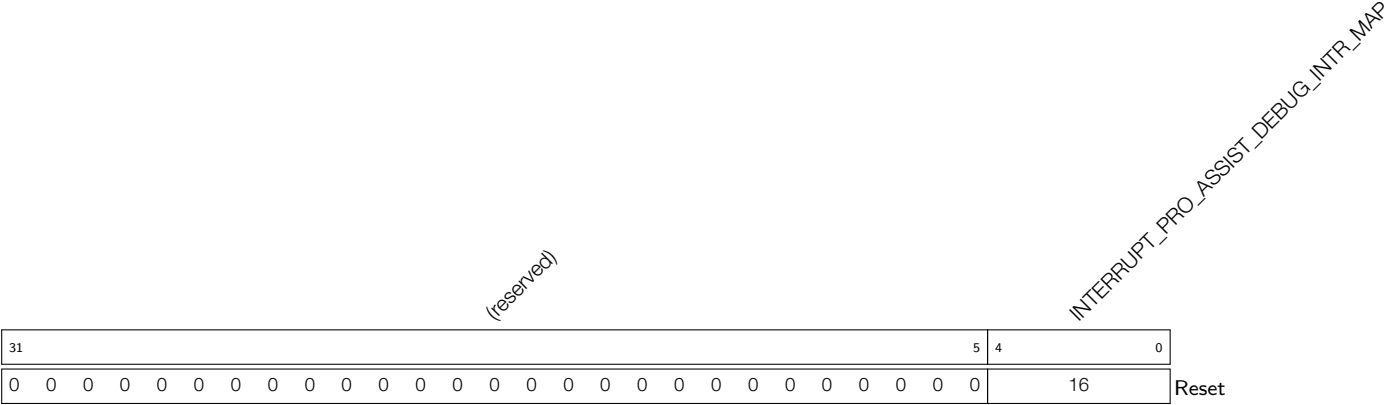


Register 4.74: INTERRUPT\_PRO\_SYSTIMER\_TARGET2\_INT\_MAP\_REG (0x0124)



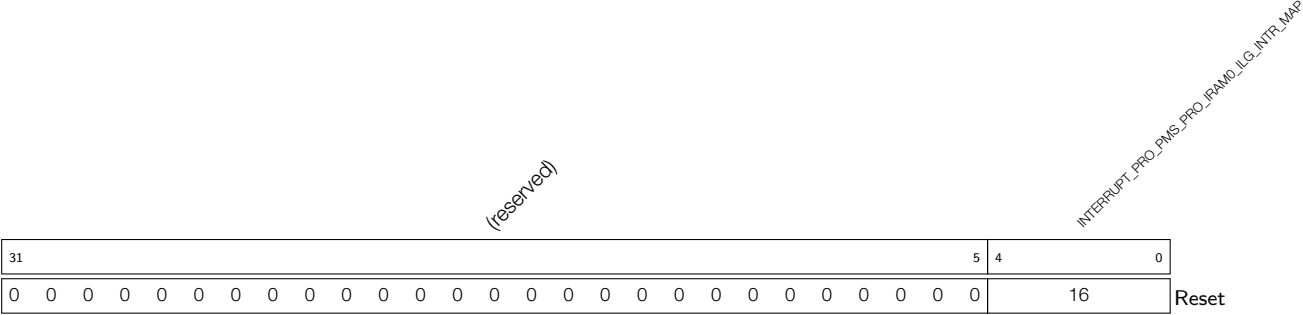
**INTERRUPT\_PRO\_SYSTIMER\_TARGET2\_INT\_MAP** 用于将 SYSTIMER\_TARGET2 中断信号映射至 CPU 中断。(读/写)

Register 4.75: INTERRUPT\_PRO\_ASSIST\_DEBUG\_INTR\_MAP\_REG (0x0128)



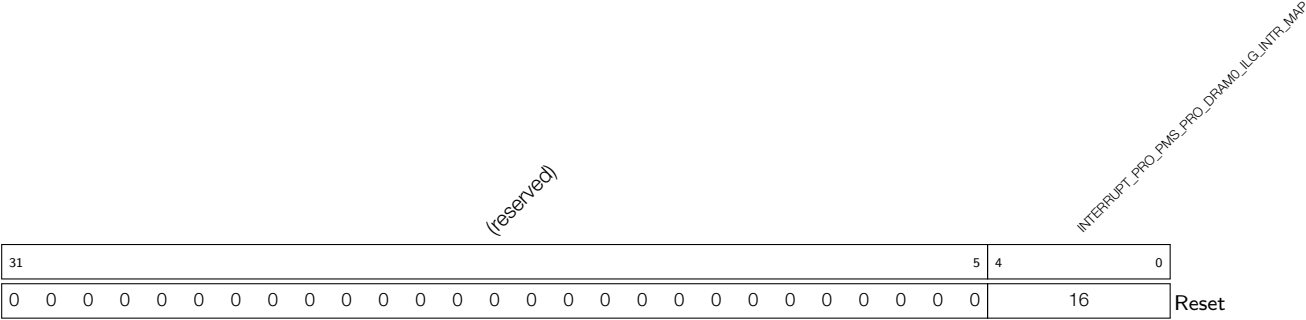
**INTERRUPT\_PRO\_ASSIST\_DEBUG\_INTR\_MAP** 用于将 ASSIST\_DEBUG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.76: INTERRUPT\_PRO\_PMS\_PRO\_IRAM0\_ILG\_INTR\_MAP\_REG (0x012C)



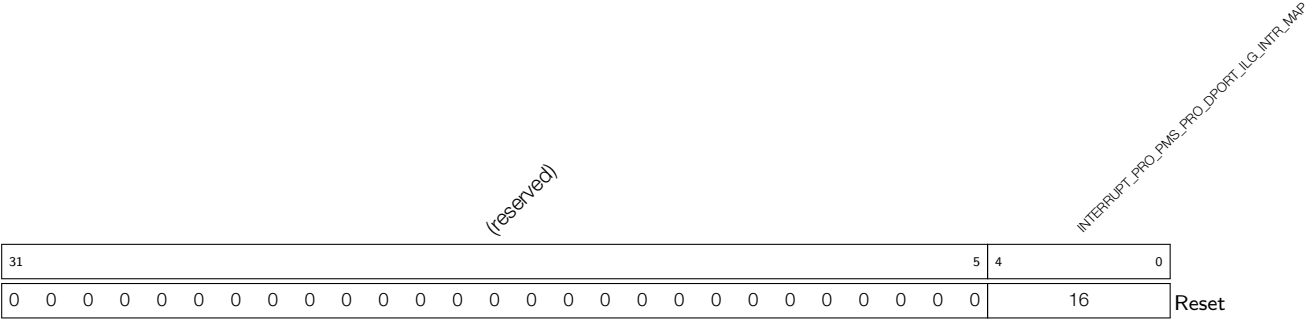
**INTERRUPT\_PRO\_PMS\_PRO\_IRAM0\_ILG\_INTR\_MAP** 用于将 PMS\_PRO\_IRAM0\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.77: INTERRUPT\_PRO\_PMS\_PRO\_DRAM0\_ILG\_INTR\_MAP\_REG (0x0130)



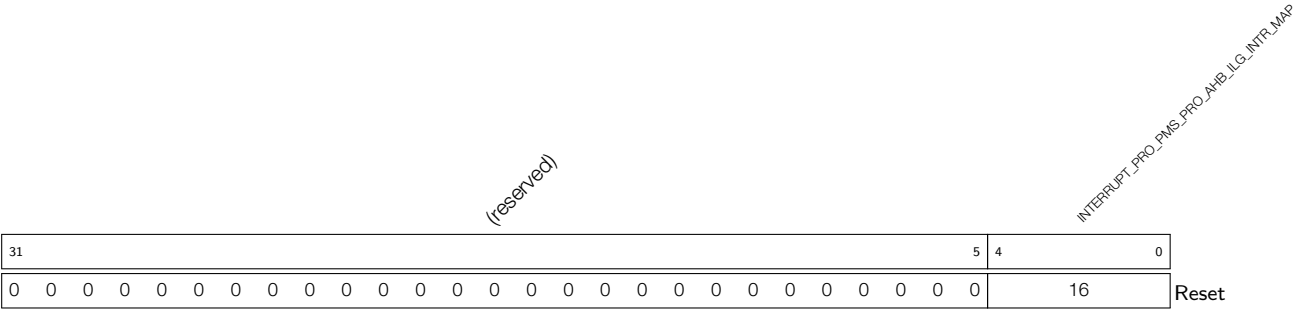
**INTERRUPT\_PRO\_PMS\_PRO\_DRAM0\_ILG\_INTR\_MAP** 用于将 PMS\_PRO\_DRAM0\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.78: INTERRUPT\_PRO\_PMS\_PRO\_DPORT\_ILG\_INTR\_MAP\_REG (0x0134)



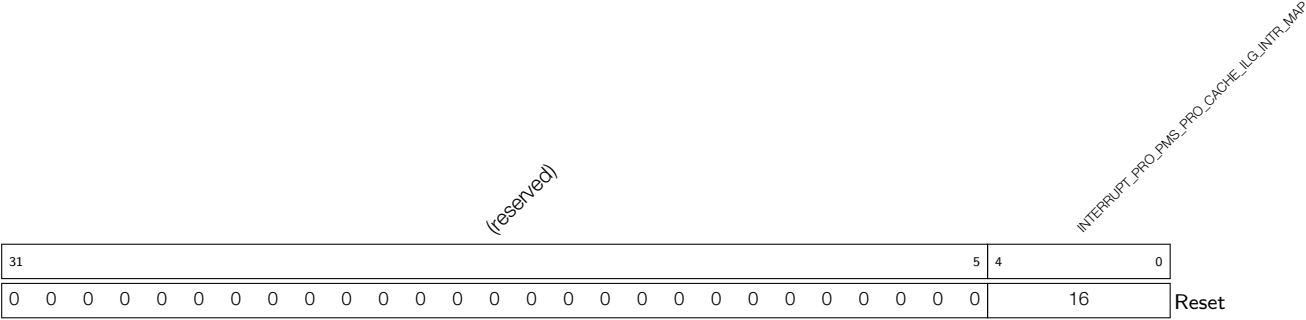
**INTERRUPT\_PRO\_PMS\_PRO\_DPORT\_ILG\_INTR\_MAP** 用于将 PMS\_PRO\_DPORT\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.79: INTERRUPT\_PRO\_PMS\_PRO\_AHB\_ILG\_INTR\_MAP\_REG (0x0138)



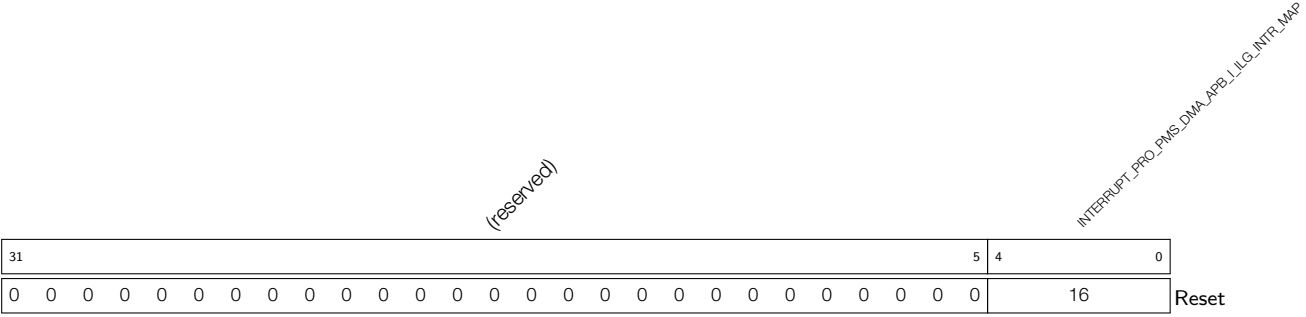
**INTERRUPT\_PRO\_PMS\_PRO\_AHB\_ILG\_INTR\_MAP** 用于将 PMS\_PRO\_AHB\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.80: INTERRUPT\_PRO\_PMS\_PRO\_CACHE\_ILG\_INTR\_MAP\_REG (0x013C)



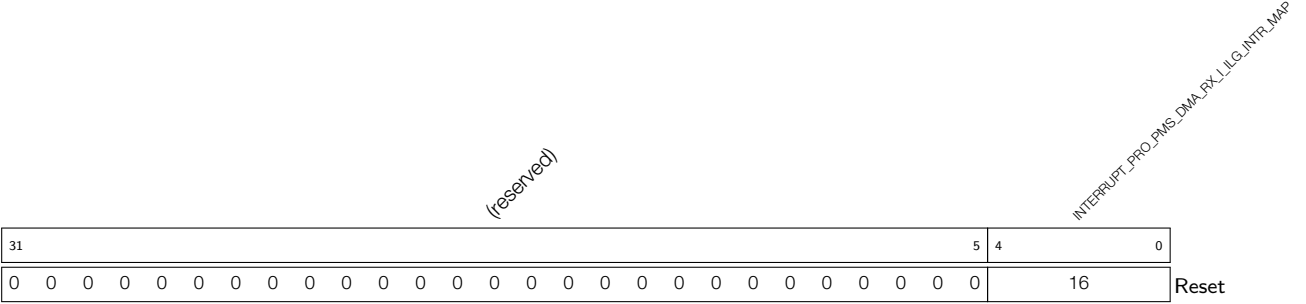
**INTERRUPT\_PRO\_PMS\_PRO\_CACHE\_ILG\_INTR\_MAP** 用于将 PMS\_PRO\_CACHE\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.81: INTERRUPT\_PRO\_PMS\_DMA\_APB\_I\_ILG\_INTR\_MAP\_REG (0x0140)



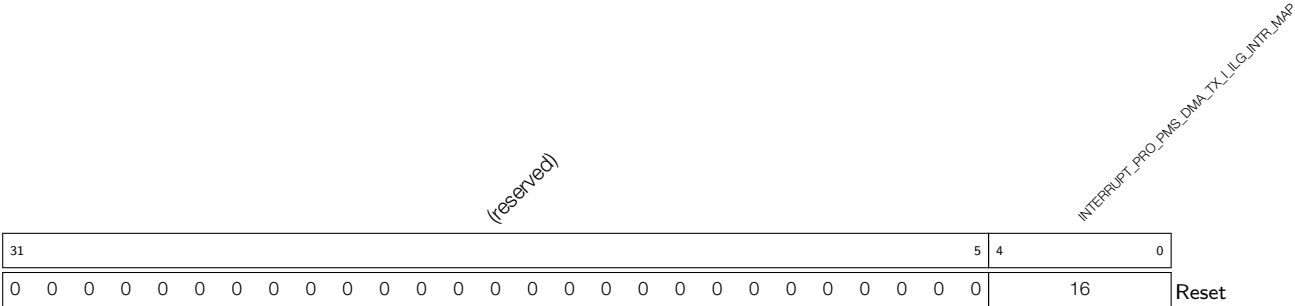
**INTERRUPT\_PRO\_PMS\_DMA\_APB\_I\_ILG\_INTR\_MAP** 用于将 PMS\_DMA\_APB\_I\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.82: INTERRUPT\_PRO\_PMS\_DMA\_RX\_I\_ILG\_INTR\_MAP\_REG (0x0144)



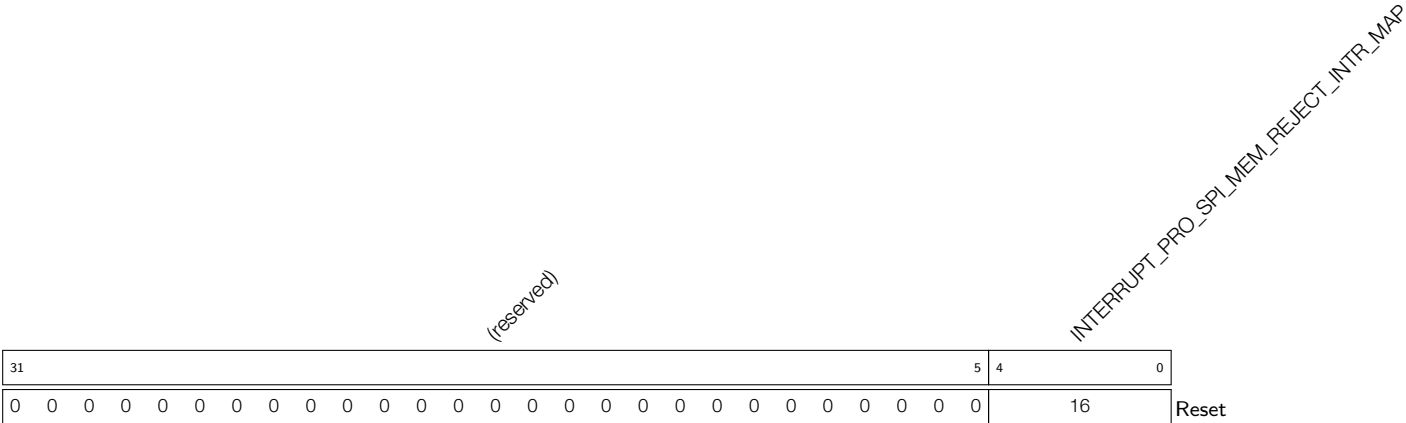
**INTERRUPT\_PRO\_PMS\_DMA\_RX\_I\_ILG\_INTR\_MAP** 用于将 PMS\_DMA\_RX\_I\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.83: INTERRUPT\_PRO\_PMS\_DMA\_TX\_I\_ILG\_INTR\_MAP\_REG (0x0148)



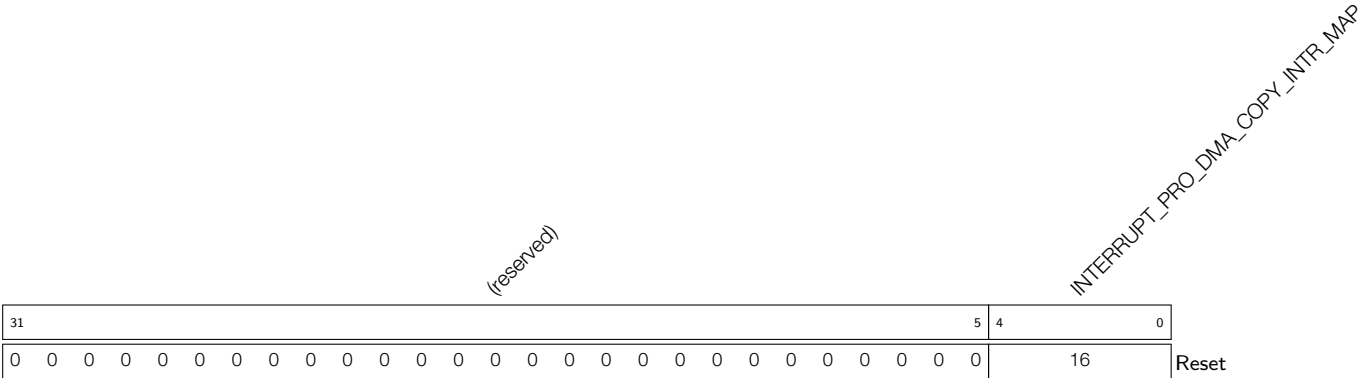
**INTERRUPT\_PRO\_PMS\_DMA\_TX\_I\_ILG\_INTR\_MAP** 用于将 PMS\_DMA\_TX\_I\_ILG\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.84: INTERRUPT\_PRO\_SPI\_MEM\_REJECT\_INTR\_MAP\_REG (0x014C)



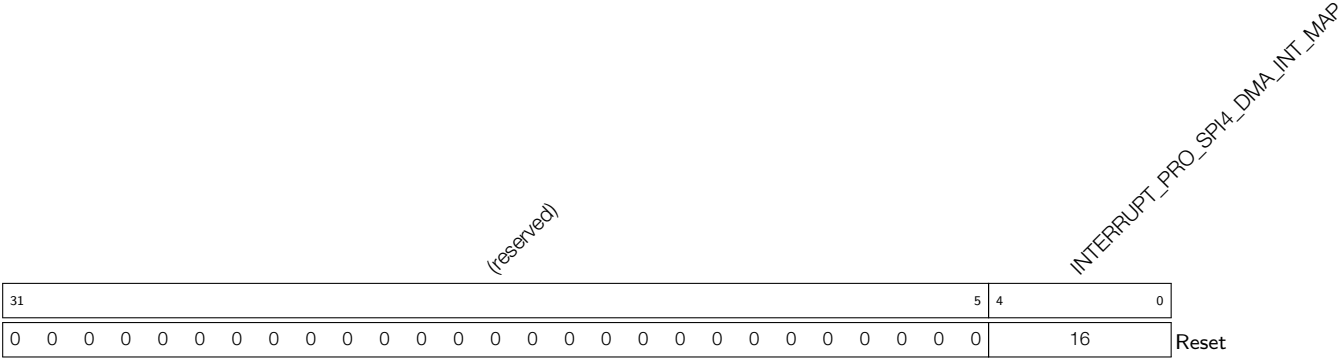
**INTERRUPT\_PRO\_SPI\_MEM\_REJECT\_INTR\_MAP** 用于将 SPI\_MEM\_REJECT\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.85: INTERRUPT\_PRO\_DMA\_COPY\_INTR\_MAP\_REG (0x0150)



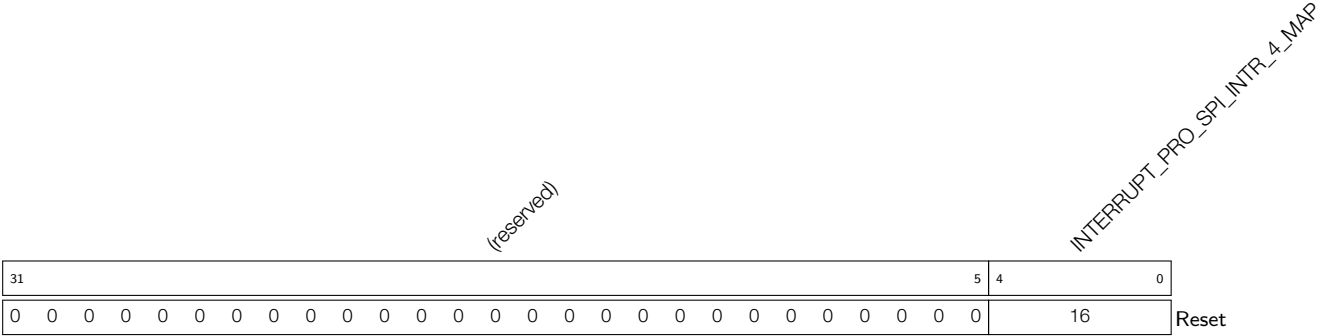
**INTERRUPT\_PRO\_DMA\_COPY\_INTR\_MAP** 用于将 DMA\_COPY\_INTR 中断信号映射至 CPU 中断。(读/写)

Register 4.86: INTERRUPT\_PRO\_SPI4\_DMA\_INT\_MAP\_REG (0x0154)



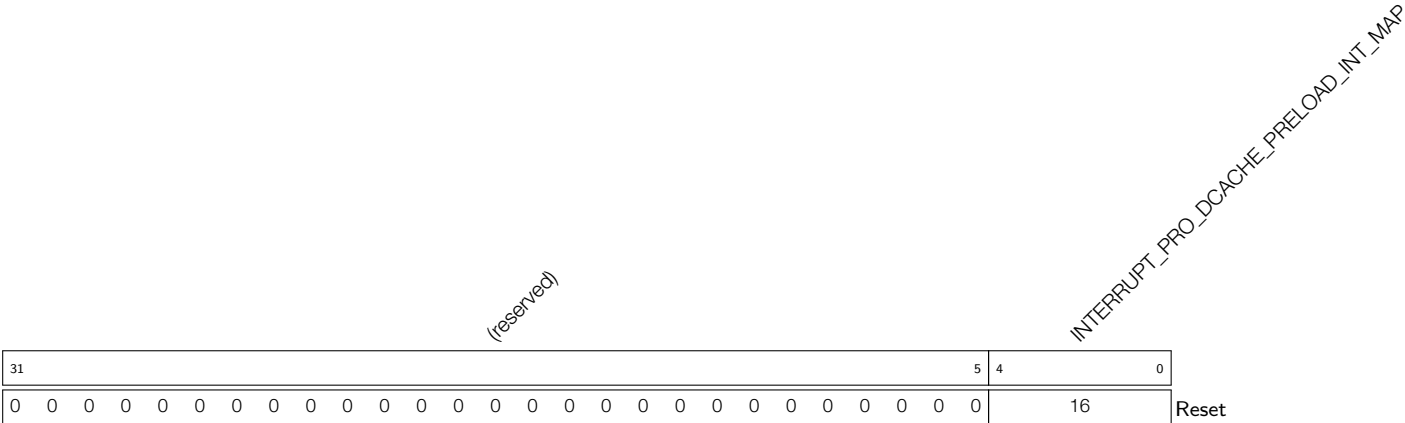
INTERRUPT\_PRO\_SPI4\_DMA\_INT\_MAP 用于将 SPI4\_DMA\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.87: INTERRUPT\_PRO\_SPI\_INTR\_4\_MAP\_REG (0x0158)



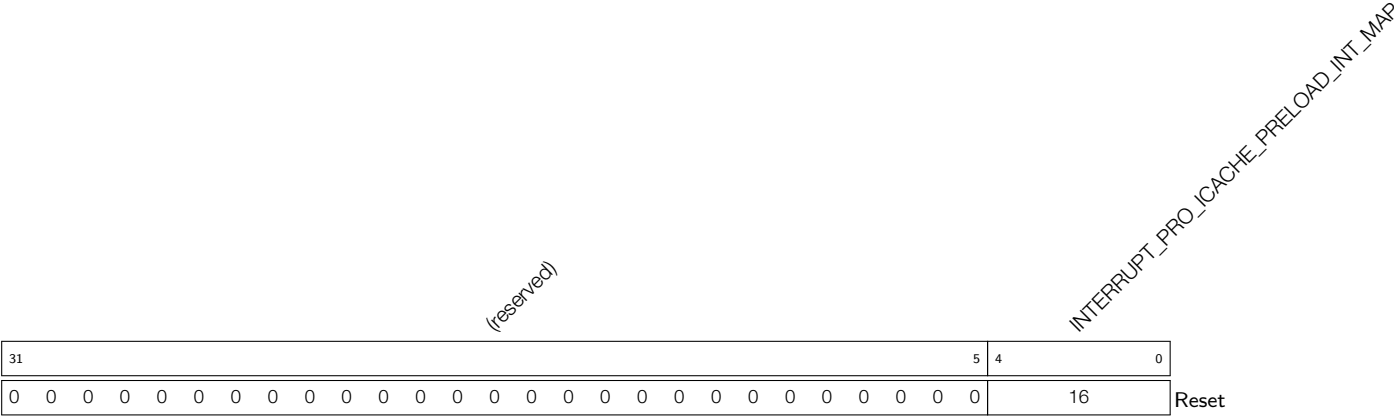
INTERRUPT\_PRO\_SPI\_INTR\_4\_MAP 用于将 SPI\_INTR\_4 中断信号映射至 CPU 中断。(读/写)

Register 4.88: INTERRUPT\_PRO\_DCACHE\_PRELOAD\_INT\_MAP\_REG (0x015C)



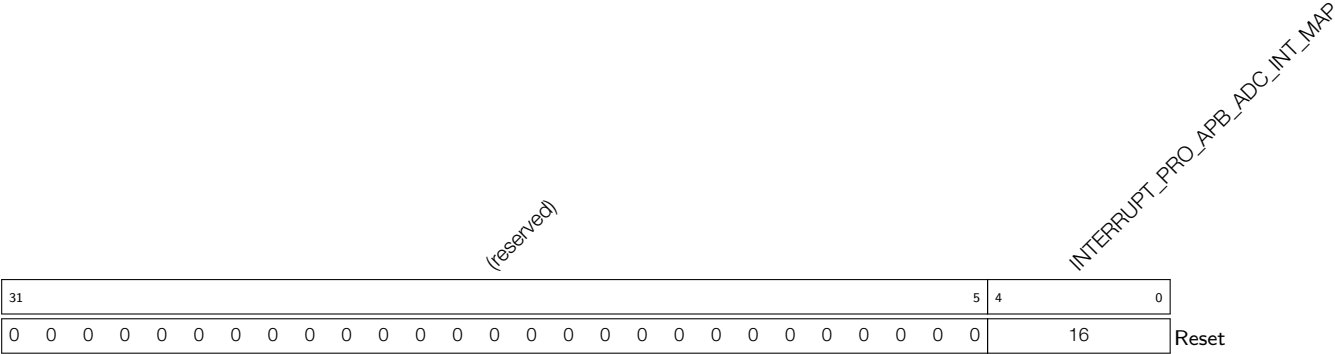
INTERRUPT\_PRO\_DCACHE\_PRELOAD\_INT\_MAP 用于将 DCACHE\_PRELOAD\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.89: INTERRUPT\_PRO\_ICACHE\_PRELOAD\_INT\_MAP\_REG (0x0160)



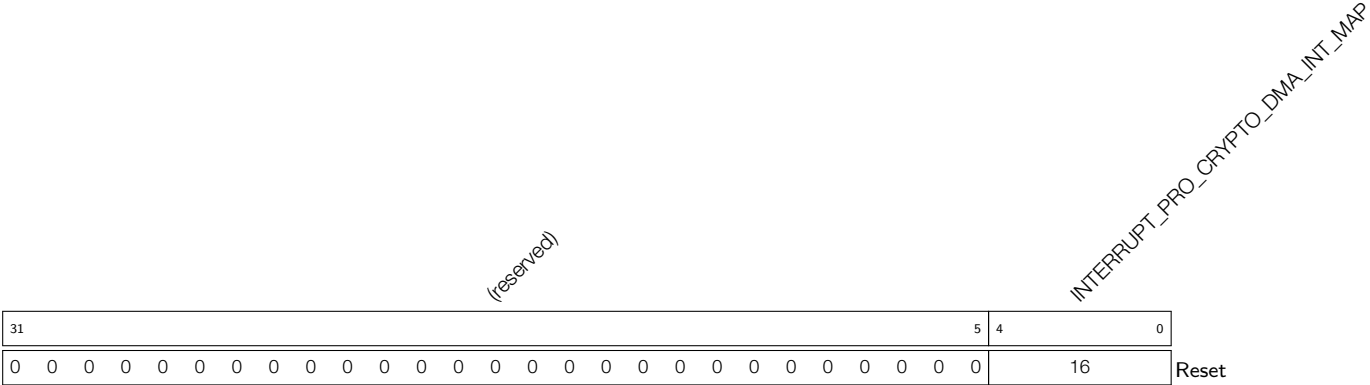
**INTERRUPT\_PRO\_ICACHE\_PRELOAD\_INT\_MAP** 用于将 ICACHE\_PRELOAD\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.90: INTERRUPT\_PRO\_APB\_ADC\_INT\_MAP\_REG (0x0164)



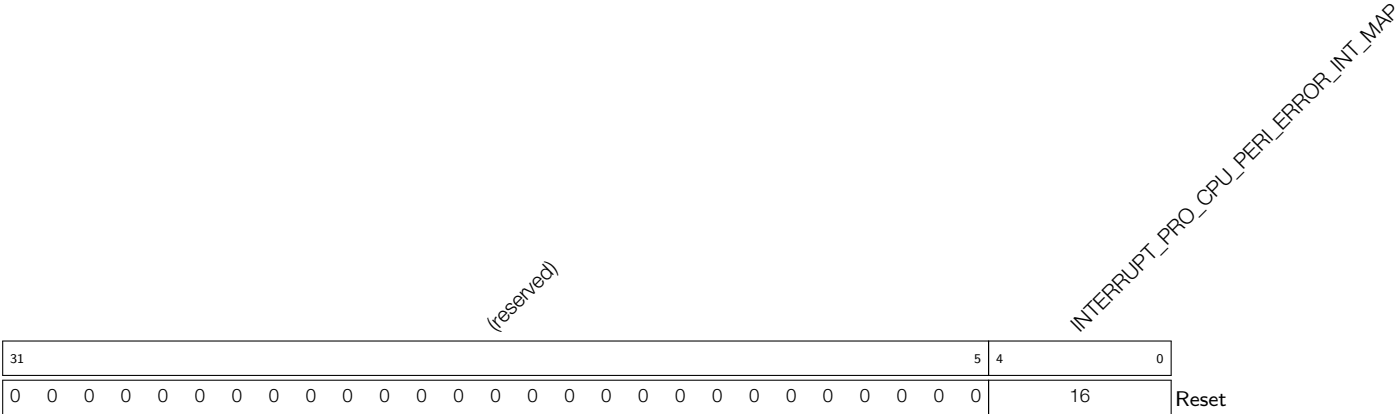
**INTERRUPT\_PRO\_APB\_ADC\_INT\_MAP** 用于将 APB\_ADC\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.91: INTERRUPT\_PRO\_CRYPTO\_DMA\_INT\_MAP\_REG (0x0168)



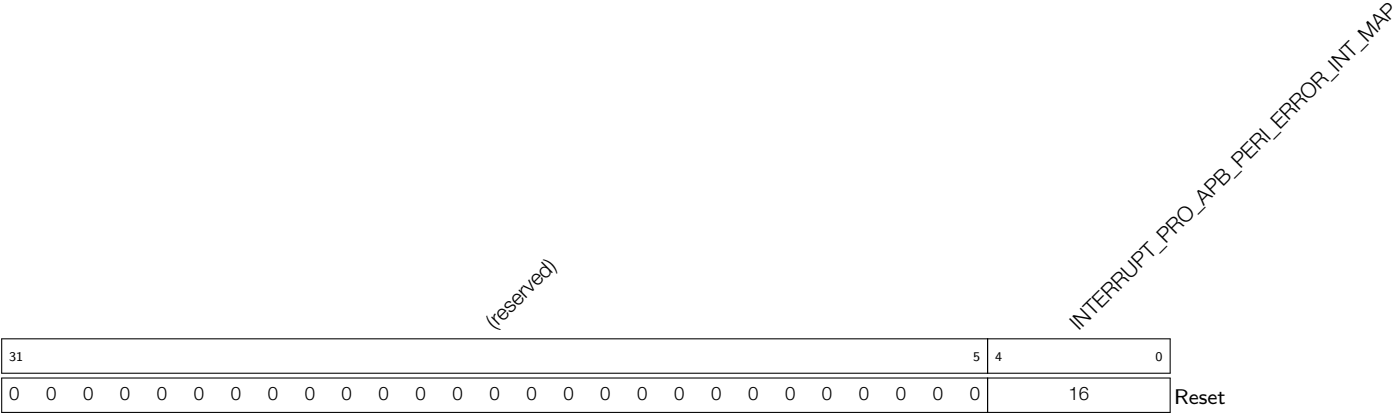
**INTERRUPT\_PRO\_CRYPTO\_DMA\_INT\_MAP** 用于将 CRYPTO\_DMA\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.92: INTERRUPT\_PRO\_CPU\_PERI\_ERROR\_INT\_MAP\_REG (0x016C)



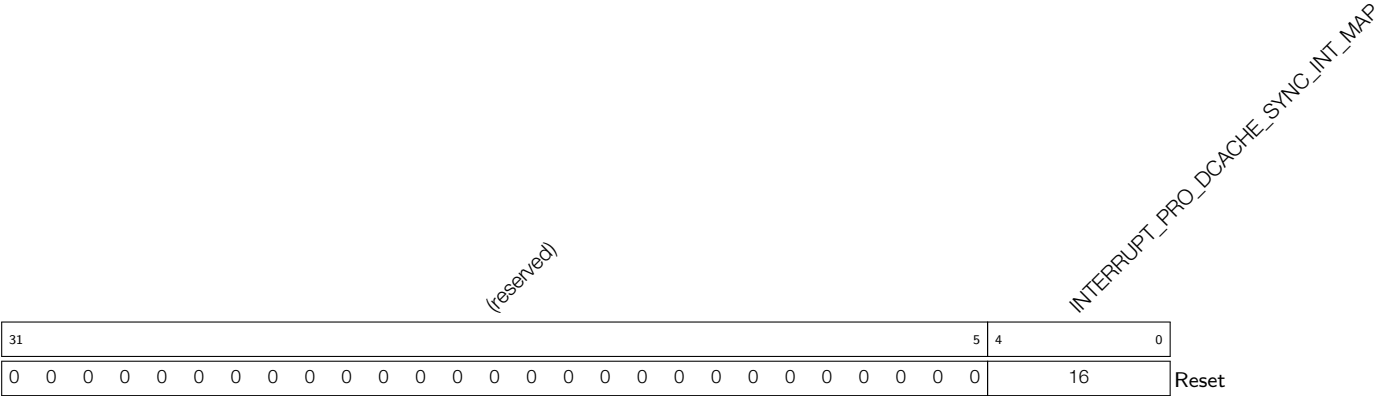
**INTERRUPT\_PRO\_CPU\_PERI\_ERROR\_INT\_MAP** 用于将 CPU\_PERI\_ERROR\_INT 中断信号映射至 CPU 中断。(读/写)

Register 4.93: INTERRUPT\_PRO\_APB\_PERI\_ERROR\_INT\_MAP\_REG (0x0170)



**INTERRUPT\_PRO\_APB\_PERI\_ERROR\_INT\_MAP** 用于将 APB\_PERI\_ERROR\_INT 中断信号映射至 CPU 中断。(读/写)

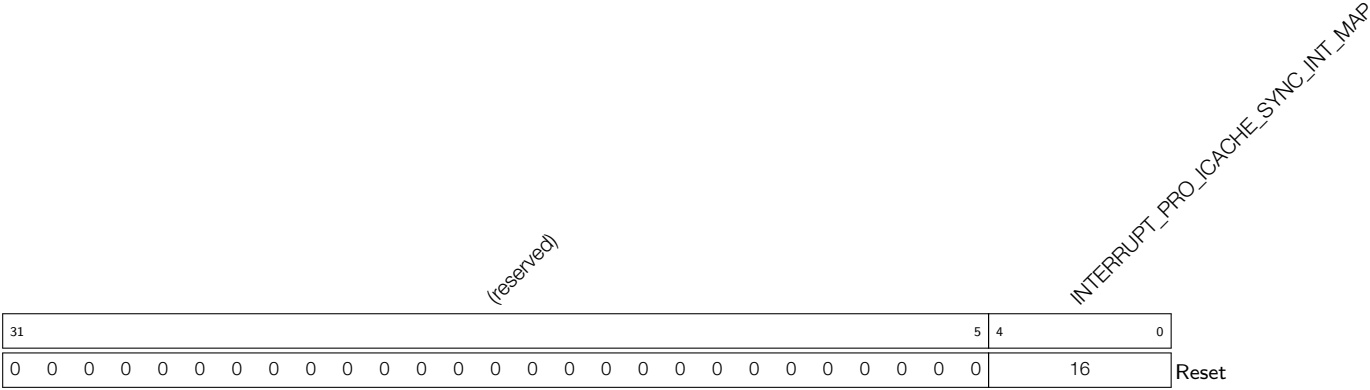
Register 4.94: INTERRUPT\_PRO\_DCACHE\_SYNC\_INT\_MAP\_REG (0x0174)



**INTERRUPT\_PRO\_DCACHE\_SYNC\_INT\_MAP** 用于将 DCACHE\_SYNC\_INT 中断信号映射至 CPU 中断。(读/写)

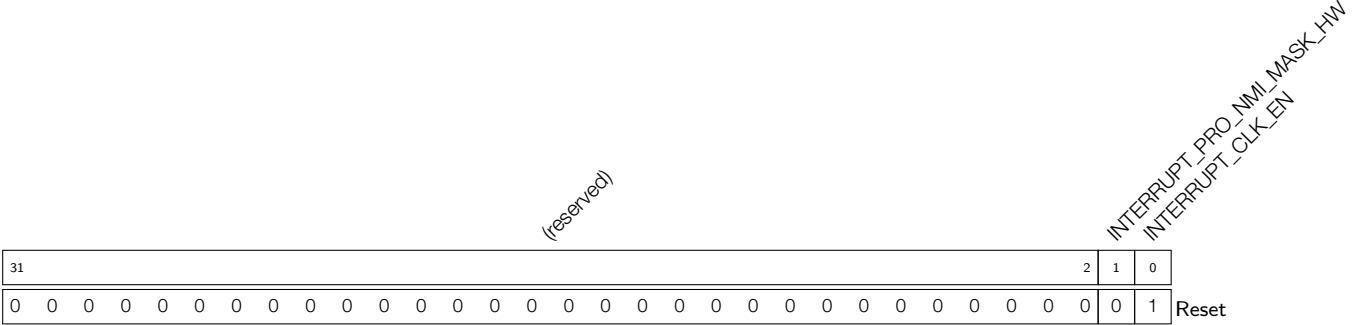


Register 4.95: INTERRUPT\_PRO\_ICACHE\_SYNC\_INT\_MAP\_REG (0x0178)



**INTERRUPT\_PRO\_ICACHE\_SYNC\_INT\_MAP** 用于将 ICACHE\_SYNC\_INT 中断信号映射至 CPU 中断。(读/写)

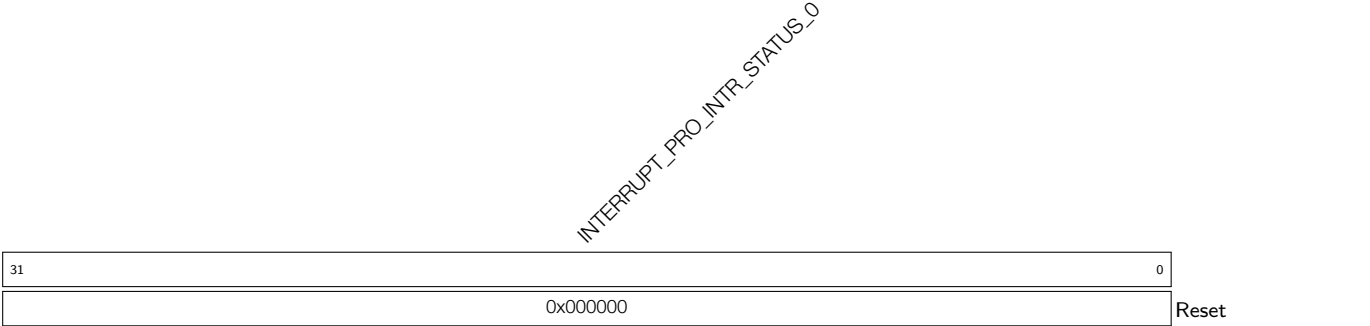
Register 4.96: INTERRUPT\_CLOCK\_GATE\_REG (0x0188)



**INTERRUPT\_CLK\_EN** 使能或关闭中断矩阵的时钟。1：使能时钟；0：关闭时钟。(读/写)

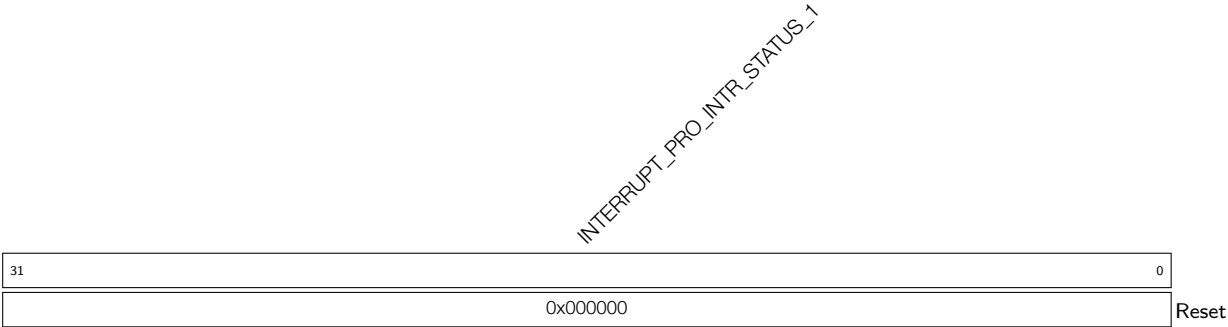
**INTERRUPT\_PRO\_NMI\_MASK\_HW** 屏蔽所有 NMI 中断信号映射至 CPU。(读/写)

Register 4.97: INTERRUPT\_PRO\_INTR\_STATUS\_REG\_0\_REG (0x017C)



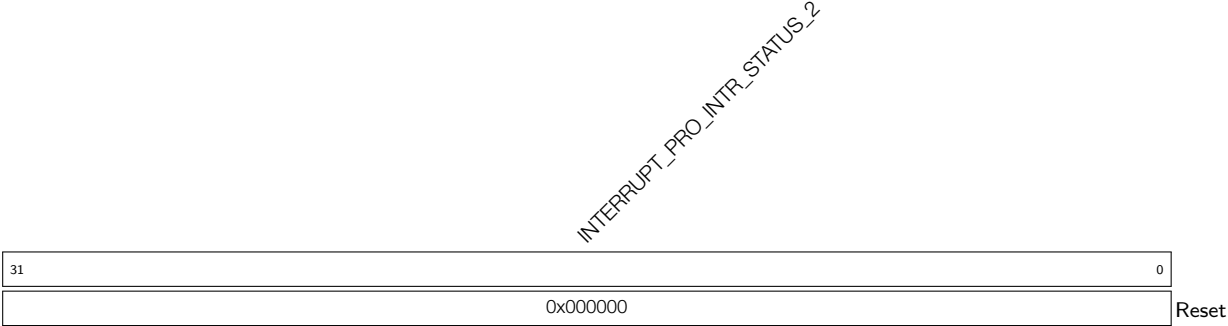
**INTERRUPT\_PRO\_INTR\_STATUS\_0** 用于存储前 32 个中断源的状态。(只读)

Register 4.98: INTERRUPT\_PRO\_INTR\_STATUS\_REG\_1\_REG (0x0180)



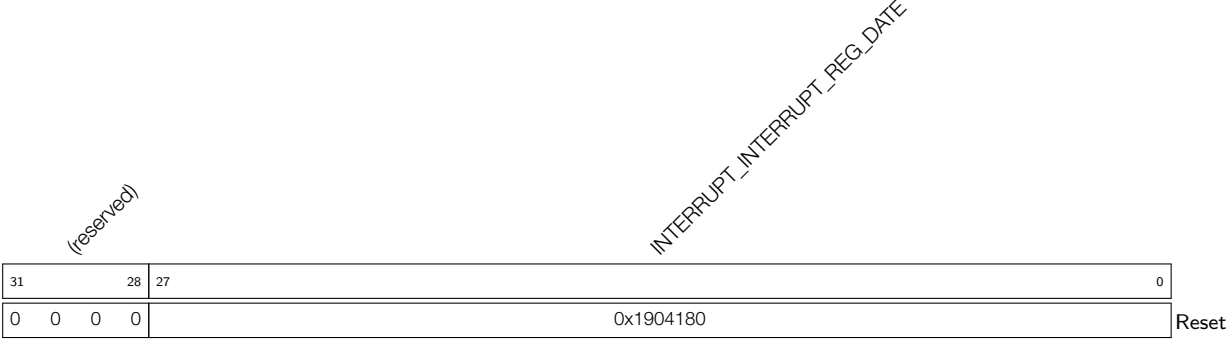
INTERRUPT\_PRO\_INTR\_STATUS\_1 用于存储中间 32 个中断源的状态。(只读)

Register 4.99: INTERRUPT\_PRO\_INTR\_STATUS\_REG\_2\_REG (0x0184)



INTERRUPT\_PRO\_INTR\_STATUS\_2 用于存储最后 31 个中断源的状态。(只读)

Register 4.100: INTERRUPT\_REG\_DATE\_REG (0x0FFC)



INTERRUPT\_DATE 版本控制寄存器。(读/写)

## 5. IO MUX 和 GPIO 交换矩阵

### 5.1 概述

ESP32-S2 芯片有 43 个物理 GPIO pad。每个 pad 都可用作一个通用 IO，或连接一个内部的外设信号。IO MUX、RTC IO MUX 和 GPIO 交换矩阵用于将信号从外设传输至 GPIO pad。这些模块共同组成了芯片的 IO 控制。

**注意：**这 43 个物理 GPIO pad 的序列号为：0 ~ 21, 26 ~ 46。其中 GPIO46 仅用作输入管脚，其他的既可以作为输入又可以作为输出管脚。

本章节主要描述了如何通过 GPIO 交换矩阵、IO MUX 或 RTC IO MUX 实现 43 个数字 pad（控制信号：DRV、IE、OE、WPU、WPD 等）与内部信号的选择和连接，这些内部信号包括：

- 116 个数字外设输入信号（控制信号：SIG\_IN\_SEL、SIG\_OUT\_SEL、IE、OE 等）
- 182 个数字外设输出信号（控制信号：SIG\_IN\_SEL、SIG\_OUT\_SEL、IE、OE 等）
- 快速外设输入和输出信号（控制信号：IE、OE 等）
- 22 个 RTC GPIO 信号

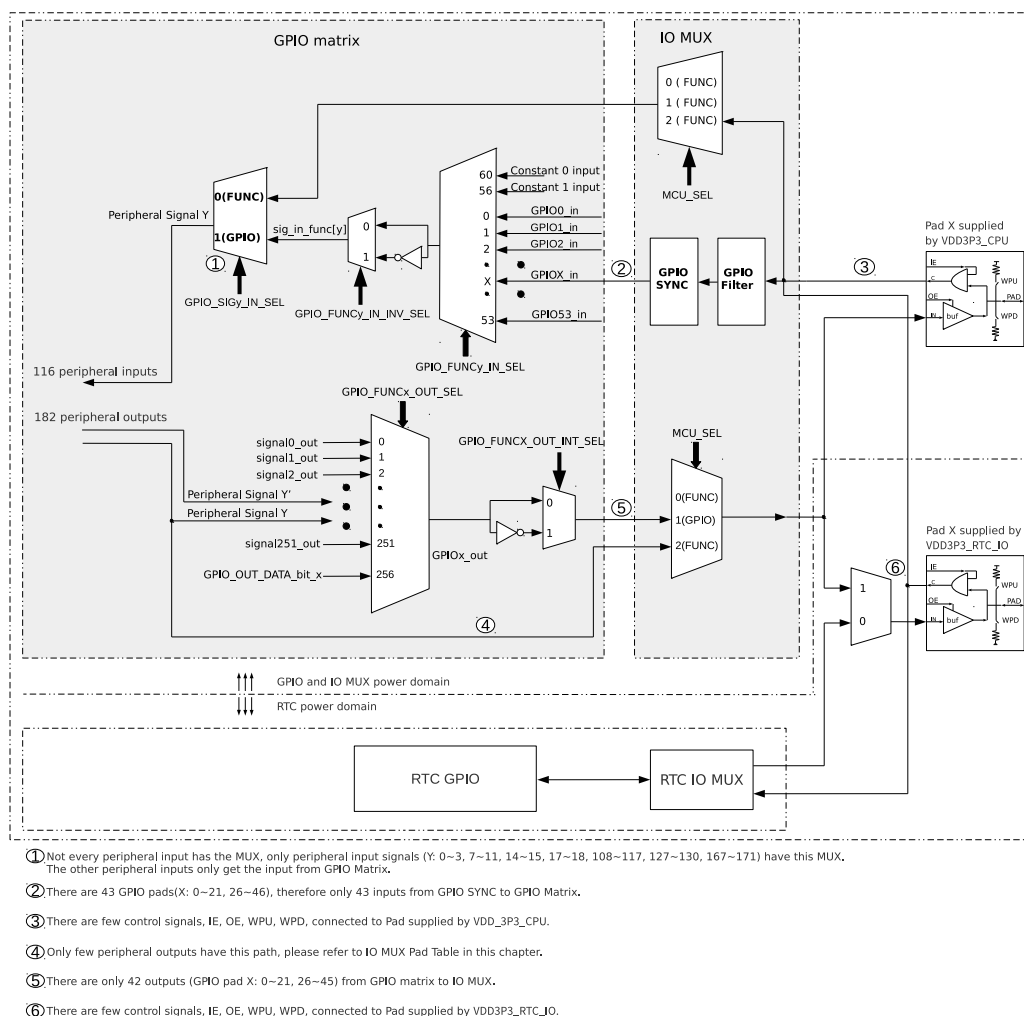


图 5-1. IO MUX、RTC IO MUX 和 GPIO 交换矩阵结构框图

图 5-1 为 IO MUX、RTC IO MUX 和 GPIO 交换矩阵结构框图。

1. IO MUX 中每个 GPIO pad 有一个寄存器 `IO_MUX_n_REG`，每个 pad 可以配置成：

- GPIO 功能，连接 GPIO 交换矩阵；
- 直连功能，旁路 GPIO 交换矩阵。

快速信号如 SPI、JTAG、UART 等可以旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以高速信号会直接通过 IO MUX 输入和输出。

章节 5.10 列出了所有 GPIO pad 的 IO MUX 功能。

2. GPIO 交换矩阵是外设输入输出信号和管脚之间的全交换矩阵。

- 芯片输入方向：116 个外设输入信号都可以选择任意一个 GPIO pad 的输入信号。
- 芯片输出方向：每个 GPIO pad 的输出信号可来自 182 个外设输出信号中的任意一个。

章节 5.9 列出了 GPIO 交换矩阵的外设信号。

3. RTC IO MUX 用于控制 GPIO pad 的低功耗和模拟功能。只有部分 GPIO pad 具有这些功能。

章节 5.11 列出了 RTC IO MUX 功能。

## 5.2 通过 GPIO 交换矩阵的外设输入

### 5.2.1 概述

为实现通过 GPIO 交换矩阵接收外设输入信号，需要配置 GPIO 交换矩阵从 43 个 GPIO (0~21, 26~46) 中获取外设输入信号的索引号，见交换矩阵表格 22。并需要配置外设输入选择通过 GPIO 交换矩阵接收输入信号。

### 5.2.2 信号同步

如图 5-1 所示，对于信号输入，外部输入信号从 pad 输入，经 GPIO SYNC 模块同步至 APB 总线时钟后进入 GPIO 交换矩阵。外部输入信号也可以通过 IO MUX 直接进入外设，但信号无法经由 GPIO SYNC 模块同步。

GPIO SYNC 模块的功能如图 5-2 所示。其中，negative sync 为 GPIO input 经过 APB clock 的下降沿同步，positive sync 为 GPIO input 经过 APB clock 上升沿同步。

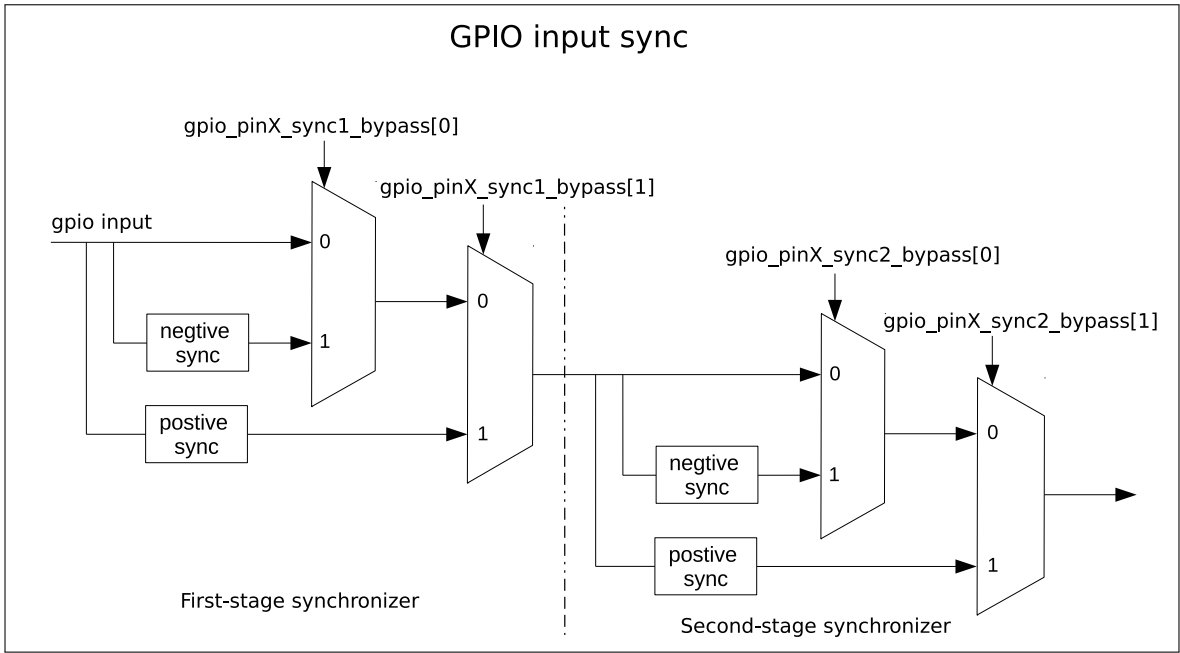


图 5-2. GPIO 输入经 clock 上升沿或下降沿同步

### 5.2.3 功能描述

把某个外设输入信号  $Y$  绑定到某个 GPIO pad  $X$  的配置过程为：

1. 在 GPIO 交换矩阵中配置外设信号  $Y$  的 `GPIO_FUNCy_IN_SEL_CFG_REG` 寄存器：

- 置位 `GPIO_SIGy_IN_SEL` 选择通过 GPIO 交换矩阵接收外部输入信号；
- 设置 `GPIO_FUNCy_IN_SEL` 为要读取的 GPIO pad  $X$  的值。

**注意：**并不是所有外设信号都有有效的 `GPIO_SIGy_IN_SEL`，即有些外设信号没有图 5-1 中所示的 MUX（见图下方的注释 ①）。这些外设信号只能通过 GPIO 交换矩阵接收外部输入信号。

2. 置位寄存器 `IO_MUX_FILTER_EN` 使能 pad 输入信号滤波，如图 5-3 所示。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。

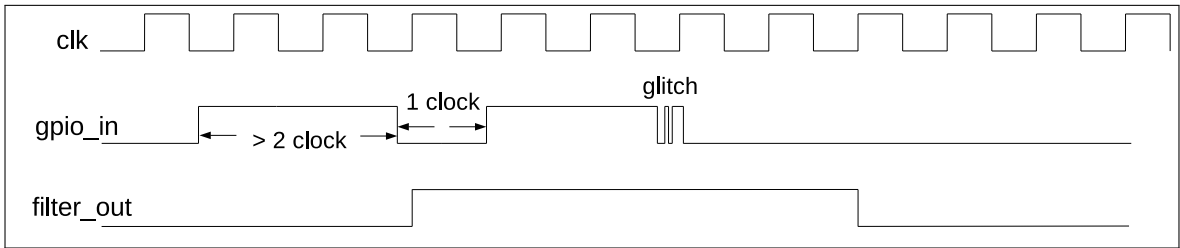


图 5-3. GPIO 输入信号滤波时序图

3. 配置 GPIO pad  $X$  的 `GPIO_PIN $x$ _REG` 来同步 GPIO input，过程如下：

- 如图 5-2 所示，配置 `GPIO_PIN $x$ _SYNC1_BYPASS` 使能输入信号第一拍为上升沿或者下降沿同步。
- 如图 5-2 所示，配置 `GPIO_PIN $x$ _SYNC2_BYPASS` 使能输入信号第二拍为上升沿或者下降沿同步。

4. 配置 GPIO pad  $X$  的 `IO_MUX_ $x$ _REG` 来使能 pad 输入功能，过程如下：

- 置位 `IO_MUX_FUN_IE` 使能输入；
- 置位或清零 `IO_MUX_FUN_WPU` 和 `IO_MUX_FUN_WPD`，使能或关闭内部上拉/下拉电阻器。

例如，要把 RMT 外设通道 0 的输入信号 `rmt_sig_in0`（信号索引号 83）绑定到 GPIO40，请按照以下步骤操作（请注意 GPIO40 也叫做 MTDO 管脚）：

1. 置位 `GPIO_FUNC83_IN_SEL_CFG_REG` 寄存器的 `GPIO_SIG83_IN_SEL`，使能通过 GPIO 交换矩阵接收外部输入信号；
2. 将 `GPIO_FUNC83_IN_SEL_CFG_REG` 寄存器的 `GPIO_FUNC83_IN_SEL` 字段设置为 40；
3. 置位 `IO_MUX_GPIO40_REG` 寄存器的 `IO_MUX_FUN_IE`（使能输入模式）。

说明：

- 同一个输入 pad 上可以同时绑定多个内部 `input_signals`。
- 置位 `GPIO_FUNC $y$ _IN_INV_SEL` 可以把输入的信号取反。
- 无需将输入信号绑定到一个 pad 也可以使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：
  - 当 `GPIO_FUNC $y$ _IN_SEL` 是 0x3C 时，`input_signal $X$`  始终为 0；
  - 当 `GPIO_FUNC $y$ _IN_SEL` 是 0x38 时，`input_signal $X$`  始终为 1。

## 5.2.4 简单 GPIO 输入

`GPIO_IN_REG`/`GPIO_IN1_REG` 寄存器存储着每一个 GPIO pad 的输入值。

任意 GPIO pad 的输入值都可以随时读取而无需为某一个外设信号配置 GPIO 交换矩阵。但是需要为 pad  $X$  的 `IO_MUX_ $n$ _REG` 寄存器配置 `IO_MUX_FUN_IE` 位以使能输入，如章节 5.2.2 所述。

## 5.3 通过 GPIO 交换矩阵的外设输出

### 5.3.1 概述

为实现通过 GPIO 交换矩阵输出外设信号，需要配置 GPIO 交换矩阵将输出索引号为 (0 ~ 11, 14 ~ 18, ...) 的外设信号输出到 42 个 GPIO (0 ~ 21, 26 ~ 45)。具体输出索引号见交换矩阵表格 22。

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 pad 为 GPIO 功能，这样输出 GPIO 信号就能连接到相应 pad。

**说明:**

输出索引号为 223 ~ 227 的外设信号，可配置为从一个 GPIO 管脚输入后，直接由另一个 GPIO 管脚输出。

**5.3.2 功能描述**

如图 5-1 所示，对于信号输出，182 个输出信号中的某一个信号通过 GPIO 交换矩阵到达 IO MUX，然后连接到某个 pad。

输出外设信号 *Y* 到某一 GPIO pad *X* 的步骤为：

- 在 GPIO 交换矩阵里配置 GPIO pad *X* 的 `GPIO_FUNCx_OUT_SEL_CFG_REG` 寄存器和 `GPIO_ENABLE_REG[i]` 字段。推荐使用相应 W1TS 或 W1TC 寄存器来更新 `GPIO_ENABLE_REG` 寄存器中的值：
  - 设置 `GPIO_FUNCx_OUT_SEL_CFG_REG` 寄存器的 `GPIO_FUNCx_OUT_SEL` 字段为外设输出信号 *Y* 的索引号 (*Y*)。
  - 要将信号强制使能为输出模式，需要将 GPIO pad *X* 对应的 `GPIO_FUNCx_OUT_SEL_CFG_REG` 寄存器的 `GPIO_FUNCx_OEN_SEL` 字段置位；同时需要将 `GPIO_ENABLE_W1TS_REG` 或 `GPIO_ENABLE1_W1TS_REG` 中的相应字段置位。或者，将 `GPIO_FUNCx_OEN_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。详细信息见表 22 中“输出信号的输出使能信号”一栏。
  - 清零 `GPIO_ENABLE_W1TC_REG` 或 `GPIO_ENABLE1_W1TC_REG` 中相应位可以关闭 GPIO pad 的输出。
- 要选择以开漏方式输出，可以设置 GPIO pad *X* 的 `GPIO_PINx_REG` 寄存器中的 `GPIO_PINx_PAD_DRIVER` 位。
- 配置 IO MUX 寄存器来选择 GPIO 交换矩阵。配置 GPIO pad *X* 的 `IO_MUX_x_REG` 的过程如下：
  - 设置功能字段 `IO_MUX_MCU_SEL` 为 GPIO pad *X* 的 IO MUX 功能（所有管脚的 Function 1，数值为 1）。
  - 设置 `IO_MUX_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：
    - 0: ~5 mA
    - 1: ~10 mA
    - 2: ~20 mA（默认值）
    - 3: ~40 mA
  - 在开漏模式下，通过置位/清零 `IO_MUX_FUN_WPU` 和 `IO_MUX_FUN_WPD` 使能或关闭上拉/下拉电阻器。

**说明:**

- 某一个外设的输出信号可以同时从多个 pad 输出。
- GPIO46 不可用于输出信号。
- 置位 `GPIO_FUNCn_OUT_INV_SEL` 可以把输出的信号取反。

### 5.3.3 简单 GPIO 输出

GPIO 交换矩阵也可以用于简单 GPIO 输出，具体配置如下：

- 设置 GPIO 交换矩阵 `GPIO_FUNCn_OUT_SEL` 寄存器为特定的外设索引值 256 (0x100)；
- 设置 `GPIO_OUT_REG[31:0]` 或者 `GPIO_OUT1_REG[21:0]` 寄存器中某一位的值为期望 GPIO 输出的值。

说明：

- `GPIO_OUT_REG[0] ~ GPIO_OUT_REG[31]` 对应 GPIO0 ~ GPIO31，`GPIO_OUT1_REG[25:22]` 无效；
- `GPIO_OUT1_REG[0] ~ GPIO_OUT1_REG[13]` 对应 GPIO32 ~ GPIO45，`GPIO_OUT1_REG[21:14]` 无效。
- 推荐使用相应的 W1TS (write 1 to set) 和 W1TC (write 1 to clear) 寄存器，例如：  
`GPIO_OUT_W1TS/GPIO_OUT_W1TC` 来改写 `GPIO_OUT_REG` 或 `GPIO_OUT1_REG` 中的值。

### 5.3.4 Sigma Delta 调制输出 (SDM)

#### 5.3.4.1 功能描述

182 个数字外设输出中有 8 个通道支持 1 比特二阶 Sigma Delta 调制输出，这 8 个通道对应的外设输出信号索引号为 100 ~ 107，默认使能输出。该调制器实现输出可配占空比的 PDM（脉冲密度调制）信号。二阶 Sigma Delta 调制器传输函数为：

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

$E(z)$  为量化误差， $X(z)$  为输入。

Sigma Delta 调制器内部支持对 APB\_CLK 的 1 ~ 256 倍分频：

- 置位 `GPIOSD_FUNCTION_CLK_EN` 使能调制器时钟；
- 配置寄存器 `GPIOSD_SDn_PRESCALE` 实现分频。 $n$  为 0 ~ 7，对应 8 个通道。

分频后的时钟周期为调制器输出单位脉冲的周期。

`GPIOSD_SDn_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<sup>1</sup>。

- `GPIOSD_SDn_IN` = -128，调制器输出信号占空比为 0%；
- `GPIOSD_SDn_IN` = 0，调制器输出信号占空比接近 50%；
- `GPIOSD_SDn_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：

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

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。



### 5.3.4.2 配置方法

SDM 的配置方法如下：

- 将 SDM 输出经 GPIO 交换矩阵连接至 pad，见 5.3.2 章节。
- 置位 `GPIOSD_FUNCTION_CLK_EN`，使能 SDM 时钟。
- 配置 `GPIOSD_SDn_PRESCALE` 寄存器设置时钟分频系数。
- 配置 `GPIOSD_SDn_IN` 设置 SDM 输出信号的占空比。

## 5.4 IO MUX 的直接 I/O 功能

### 5.4.1 概述

快速信号如 SPI、JTAG 等会旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以高速信号会直接通过 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 pad 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。

### 5.4.2 功能描述

对于外设输入信号，I/O 旁路 GPIO 交换矩阵必须配置两个寄存器：

1. GPIO pad 的 `IO_MUX_MCU_SEL` 必须设置为相应的 pad 功能，章节 5.10 列出了 pad 功能。
2. 对于输入信号，必须将 `GPIO_SIGn_IN_SEL` 寄存器设置为低电平，则输入信号直接连接到外设。

对于外设输出信号，I/O 旁路 GPIO 交换矩阵只需将 GPIO pad 的 `IO_MUX_MCU_SEL` 配置为相应的 pad 功能即可。

说明：

- 对于外设输入信号，并不是所有的外部输入信号都可以通过 IO MUX 连接到外设输入。某些外部输入信号只能通过 GPIO 交换矩阵连接到外设输入。
- 对于外设输出信号，并不是所有的外设输出都可以直接通过 IO MUX 连接到 pad。某些外设输出只能通过 GPIO 交换矩阵连接到 pad。

## 5.5 RTC IO MUX 的低功耗和模拟 I/O 功能

### 5.5.1 概述

ESP32-S2 中有 22 个 GPIO 管脚具有低功耗（低功耗 RTC）性能和模拟功能，由 RTC 子系统控制。这些功能不使用 IO MUX 和 GPIO 交换矩阵，而是使用 RTC IO MUX 将 I/O 指向 RTC 子系统。

当这些管脚被配置为 RTC GPIO 管脚，作为输出管脚时仍然能够在芯片处于 Deep-sleep 模式下保持输出电平值或者作为输入管脚使用时可以将芯片从 Deep-sleep 中唤醒。

章节 5.11 列出了 RTC\_MUX 管脚和功能。

### 5.5.2 功能描述

每个 pad 的模拟和 RTC 功能是由 `RTCIO_TOUCH_PAD $n$ _REG` 寄存器中的 `RTCIO_TOUCH_PAD $n$ _MUX_SEL` 位控制的。此位默认置为 0，通过 IO MUX 子系统输入输出信号，如前文所述。

如果置位 `RTCIO_TOUCH_PAD $n$ _MUX_SEL`，则输入输出信号会经过 RTC 子系统。在这种模式下，`RTCIO_TOUCH_PAD $n$ _REG` 寄存器用于数字 I/O，pad 的模拟功能也可以实现。章节 5.11 列出了 RTC 管脚的功能。

表 5.11 列出了 GPIO pad 与相应的 RTC 管脚和模拟功能的映射关系。请注意 `RTCIO_TOUCH_PAD $n$ _REG` 寄存器使用的是 RTC GPIO 管脚的序号，不是 GPIO pad 的序号。

## 5.6 Light-sleep 模式管脚功能

当 ESP32-S2 处于 Light-sleep 模式时管脚可以有不同的功能。如果某一 GPIO pad 的 `IO_MUX_ $n$ _REG` 寄存器中 `IO_MUX_SLP_SEL` 位置为 1，芯片处于 Light-sleep 模式下将由另一组不同的寄存器控制 pad。

表 21: IO MUX Light-sleep 管脚功能寄存器

IO MUX 功能	正常工作模式 或者 <code>IO_MUX_SLP_SEL = 0</code>	Light-sleep 模式 并且 <code>IO_MUX_SLP_SEL = 1</code>
输出驱动强度	<code>IO_MUX_FUN_DRV</code>	<code>IO_MUX_FUN_DRV</code>
上拉电阻	<code>IO_MUX_FUN_WPU</code>	<code>IO_MUX_MCU_WPU</code>
下拉电阻	<code>IO_MUX_FUN_WPD</code>	<code>IO_MUX_MCU_WPD</code>
输出使能	(From GPIO Matrix <code>_OEN</code> field) <sup>1</sup>	<code>IO_MUX_MCU_OE</code>

如果 `IO_MUX_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。

**说明：**

正常工作模式或者 `IO_MUX_SLP_SEL = 0` 时，输出使能配置请参考 5.3.2 章节。

## 5.7 Pad Hold 特性

每个 IO pad（包括 RTC pad）都有单独的 hold 功能，由 RTC 寄存器控制。pad 的 hold 功能被置上后，pad 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 pad 的状态。应用如果希望在看门狗超时触发内核复位和系统复位时或者 Deep-sleep 时 pad 的状态不被改变，就需要提前把 hold 置上。

**说明:**

- 对于数字 pad 而言,若要在深度睡眠掉电之后保持 pad 输入输出的状态值,需要在掉电之前把寄存器 RTC\_CNTL\_DG\_PAD\_FORCE\_UNHOLD 设置成 0。对于 RTC pad 而言, pad 的输入输出值,由寄存器 RTC\_PAD\_HOLD\_REG 中相应的位来控制 Hold 和 Unhold pad 的值。
- 在芯片被唤醒之后,若要关闭 Hold 功能,将寄存器 RTC\_CNTL\_DG\_PAD\_FORCE\_UNHOLD 设置成 1。若想继续保持 pad 的值,可把 RTC\_PAD\_HOLD\_REG 寄存器中相应的位设置成 1。

## 5.8 I/O Pad 供电

IO pad 供电请参考 [《ESP32-S2 技术规格书》](#) 中管脚定义章节。

### 5.8.1 电源管理

ESP32-S2 的数字管脚可分为如下 4 种不同的电源域。

- VDD3P3\_RTC\_IO: RTC 和 CPU 的输入电源
- VDD3P3\_CPU: CPU 的输入电源
- VDD3P3\_RTC: RTC 模拟的输入电源
- VDD\_SPI: 可配置为输入电源或输出电源

VDD\_SPI 可配置使用一个内置 LDO, 该内置 LDO 的输入和输出均为 1.8 V。如未使能 LDO, VDD\_SPI 可以与 VDD3P3\_RTC\_IO 连接在相同的电源上。

VDD\_SPI 的具体配置由 GPIO45 的 Strapping 值决定, 用户可通过 eFuse 或寄存器修改 VDD\_SPI 的配置。请参考 [《ESP32-S2 技术规格书》](#) 中的电源管理章节和 Strapping 管脚章节查看更多信息。

## 5.9 外设信号列表

表 22 列出了 GPIO 交换矩阵的外设输入输出信号。

表 22: GPIO 交换矩阵

信号索引	输入信号	默认值 *	信号可经由 IO MUX 输出	输出信号	输出信号的输出使能信号
0	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe
1	SPID_in	0	yes	SPID_out	SPID_oe
2	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe
3	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe
4	-	-	-	SPICLK_out_mux	SPICLK_oe
5	-	-	-	SPICS0_out	SPICS0_oe
6	-	-	-	SPICS1_out	SPICS1_oe
7	SPID4_in	0	yes	SPID4_out	SPID4_oe
8	SPID5_in	0	yes	SPID5_out	SPID5_oe
9	SPID6_in	0	yes	SPID6_out	SPID6_oe
10	SPID7_in	0	yes	SPID7_out	SPID7_oe

信号索引	输入信号	默认值 *	信号可经由 IO MUX 输出	输出信号	输出信号的 输出使能信号
11	SPIDQS_in	0	yes	SPIDQS_out	SPIDQS_oe
14	U0RXD_in	0	yes	U0TXD_out	1'd1
15	U0CTS_in	0	yes	U0RTS_out	1'd1
16	U0DSR_in	0	no	U0DTR_out	1'd1
17	U1RXD_in	0	yes	U1TXD_out	1'd1
18	U1CTS_in	0	yes	U1RTS_out	1'd1
21	U1DSR_in	0	no	U1DTR_out	1'd1
23	I2S0O_BCK_in	0	no	I2S0O_BCK_out	1'd1
25	I2S0O_WS_in	0	no	I2S0O_WS_out	1'd1
27	I2S0I_BCK_in	0	no	I2S0I_BCK_out	1'd1
28	I2S0I_WS_in	0	no	I2S0I_WS_out	1'd1
29	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe
30	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe
39	pcnt_sig_ch0_in0	0	no	gpio_wlan_prio	1'd1
40	pcnt_sig_ch1_in0	0	no	gpio_wlan_active	1'd1
41	pcnt_ctrl_ch0_in0	0	no	-	1'd1
42	pcnt_ctrl_ch1_in0	0	no	-	1'd1
43	pcnt_sig_ch0_in1	0	no	-	1'd1
44	pcnt_sig_ch1_in1	0	no	-	1'd1
45	pcnt_ctrl_ch0_in1	0	no	-	1'd1
46	pcnt_ctrl_ch1_in1	0	no	-	1'd1
47	pcnt_sig_ch0_in2	0	no	-	1'd1
48	pcnt_sig_ch1_in2	0	no	-	1'd1
49	pcnt_ctrl_ch0_in2	0	no	-	1'd1
50	pcnt_ctrl_ch1_in2	0	no	-	1'd1
51	pcnt_sig_ch0_in3	0	no	-	1'd1
52	pcnt_sig_ch1_in3	0	no	-	1'd1
53	pcnt_ctrl_ch0_in3	0	no	-	1'd1
54	pcnt_ctrl_ch1_in3	0	no	-	1'd1
64	usb_otg_iddig_in	0	no	-	1'd1
65	usb_otg_avalid_in	0	no	-	1'd1
66	usb_srp_bvalid_in	0	no	usb_otg_idpullup	1'd1
67	usb_otg_vbusvalid_in	0	no	usb_otg_dppulldown	1'd1
68	usb_srp_sessend_in	0	no	usb_otg_dmpulldown	1'd1
69	-	-	-	usb_otg_drvvbus	1'd1
70	-	-	-	usb_srp_chrgvbus	1'd1
71	-	-	-	usb_srp_dischrgvbus	1'd1
72	SPI3_CLK_in	0	no	SPI3_CLK_out_mux	SPI3_CLK_oe
73	SPI3_Q_in	0	no	SPI3_Q_out	SPI3_Q_oe
74	SPI3_D_in	0	no	SPI3_D_out	SPI3_D_oe
75	SPI3_HD_in	0	no	SPI3_HD_out	SPI3_HD_oe
76	SPI3_CS0_in	0	no	SPI3_CS0_out	SPI3_CS0_oe

信号索引	输入信号	默认值 *	信号可经由 IO MUX 输出	输出信号	输出信号的 输出使能信号
77	-	-	-	SPI3_CS1_out	SPI3_CS1_oe
78	-	-	-	SPI3_CS2_out	SPI3_CS2_oe
79	-	-	-	ledc_ls_sig_out0	1'd1
80	-	-	-	ledc_ls_sig_out1	1'd1
81	-	-	-	ledc_ls_sig_out2	1'd1
82	-	-	-	ledc_ls_sig_out3	1'd1
83	rmt_sig_in0	0	no	ledc_ls_sig_out4	1'd1
84	rmt_sig_in1	0	no	ledc_ls_sig_out5	1'd1
85	rmt_sig_in2	0	no	ledc_ls_sig_out6	1'd1
86	rmt_sig_in3	0	no	ledc_ls_sig_out7	1'd1
87	-	-	-	rmt_sig_out0	1'd1
88	-	-	-	rmt_sig_out1	1'd1
89	-	-	-	rmt_sig_out2	1'd1
90	-	-	-	rmt_sig_out3	1'd1
95	I2CEXT1_SCL_in	1	no	I2CEXT1_SCL_out	I2CEXT1_SCL_oe
96	I2CEXT1_SDA_in	1	no	I2CEXT1_SDA_out	I2CEXT1_SDA_oe
100	-	-	-	gpio_sd0_out	1'd1
101	-	-	-	gpio_sd1_out	1'd1
102	-	-	-	gpio_sd2_out	1'd1
103	-	-	-	gpio_sd3_out	1'd1
104	-	-	-	gpio_sd4_out	1'd1
105	-	-	-	gpio_sd5_out	1'd1
106	-	-	-	gpio_sd6_out	1'd1
107	-	-	-	gpio_sd7_out	1'd1
108	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe
109	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe
110	FSPID_in	0	yes	FSPID_out	FSPID_oe
111	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe
112	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe
113	FSPIIO4_in	0	yes	FSPIIO4_out	FSPIIO4_oe
114	FSPIIO5_in	0	yes	FSPIIO5_out	FSPIIO5_oe
115	FSPIIO6_in	0	yes	FSPIIO6_out	FSPIIO6_oe
116	FSPIIO7_in	0	yes	FSPIIO7_out	FSPIIO7_oe
117	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe
118	-	-	-	FSPICS1_out	FSPICS1_oe
119	-	-	-	FSPICS2_out	FSPICS2_oe
120	-	-	-	FSPICS3_out	FSPICS3_oe
121	-	-	-	FSPICS4_out	FSPICS4_oe
122	-	-	-	FSPICS5_out	FSPICS5_oe
123	can_rx	1	no	can_tx	1'd1
124	-	-	-	can_bus_off_on	1'd1
125	-	-	-	can_clkout	1'd1

信号索引	输入信号	默认值 *	信号可经由 IO MUX 输出	输出信号	输出信号的 输出使能信号
126	-	-	-	SUBSPICLK_out_mux	SUBSPICLK_oe
127	SUBSPIQ_in	0	yes	SUBSPIQ_out	SUBSPIQ_oe
128	SUBSPID_in	0	yes	SUBSPID_out	SUBSPID_oe
129	SUBSPIHD_in	0	yes	SUBSPIHD_out	SUBSPIHD_oe
130	SUBSPIWP_in	0	yes	SUBSPIWP_out	SUBSPIWP_oe
131	-	-	-	SUBSPICS0_out	SUBSPICS0_oe
132	-	-	-	SUBSPICS1_out	SUBSPICS1_oe
133	-	-	-	FSPIDQS_out	FSPIDQS_oe
134	-	-	-	FSPI_HSYNC_out	FSPI_HSYNC_oe
135	-	-	-	FSPI_VSYNC_out	FSPI_VSYNC_oe
136	-	-	-	FSPI_DE_out	FSPI_DE_oe
137	-	-	-	FSPICD_out	FSPICD_oe
139	-	-	-	SPI3_CD_out	SPI3_CD_oe
140	-	-	-	SPI3_DQS_out	SPI3_DQS_oe
143	I2S0I_DATA_in0	0	no	I2S0O_DATA_out0	1'd1
144	I2S0I_DATA_in1	0	no	I2S0O_DATA_out1	1'd1
145	I2S0I_DATA_in2	0	no	I2S0O_DATA_out2	1'd1
146	I2S0I_DATA_in3	0	no	I2S0O_DATA_out3	1'd1
147	I2S0I_DATA_in4	0	no	I2S0O_DATA_out4	1'd1
148	I2S0I_DATA_in5	0	no	I2S0O_DATA_out5	1'd1
149	I2S0I_DATA_in6	0	no	I2S0O_DATA_out6	1'd1
150	I2S0I_DATA_in7	0	no	I2S0O_DATA_out7	1'd1
151	I2S0I_DATA_in8	0	no	I2S0O_DATA_out8	1'd1
152	I2S0I_DATA_in9	0	no	I2S0O_DATA_out9	1'd1
153	I2S0I_DATA_in10	0	no	I2S0O_DATA_out10	1'd1
154	I2S0I_DATA_in11	0	no	I2S0O_DATA_out11	1'd1
155	I2S0I_DATA_in12	0	no	I2S0O_DATA_out12	1'd1
156	I2S0I_DATA_in13	0	no	I2S0O_DATA_out13	1'd1
157	I2S0I_DATA_in14	0	no	I2S0O_DATA_out14	1'd1
158	I2S0I_DATA_in15	0	no	I2S0O_DATA_out15	1'd1
159	-	-	-	I2S0O_DATA_out16	1'd1
160	-	-	-	I2S0O_DATA_out17	1'd1
161	-	-	-	I2S0O_DATA_out18	1'd1
162	-	-	-	I2S0O_DATA_out19	1'd1
163	-	-	-	I2S0O_DATA_out20	1'd1
164	-	-	-	I2S0O_DATA_out21	1'd1
165	-	-	-	I2S0O_DATA_out22	1'd1
166	-	-	-	I2S0O_DATA_out23	1'd1
167	SUBSPID4_in	0	yes	SUBSPID4_out	SUBSPID4_oe
168	SUBSPID5_in	0	yes	SUBSPID5_out	SUBSPID5_oe
169	SUBSPID6_in	0	yes	SUBSPID6_out	SUBSPID6_oe
170	SUBSPID7_in	0	yes	SUBSPID7_out	SUBSPID7_oe

信号索引	输入信号	默认值 *	信号可经由 IO MUX 输出	输出信号	输出信号的 输出使能信号
171	SUBSPIDQS_in	0	yes	SUBSPIDQS_out	SUBSPIDQS_oe
193	I2S0I_H_SYNC	0	no	-	1'd1
194	I2S0I_V_SYNC	0	no	-	1'd1
195	I2S0I_H_ENABLE	0	no	-	1'd1
215	-	-	-	ant_sel0	1'd1
216	-	-	-	ant_sel1	1'd1
217	-	-	-	ant_sel2	1'd1
218	-	-	-	ant_sel3	1'd1
219	-	-	-	ant_sel4	1'd1
220	-	-	-	ant_sel5	1'd1
221	-	-	-	ant_sel6	1'd1
222	-	-	-	ant_sel7	1'd1
223	sig_in_func_223	0	no	sig_in_func223	1'd1
224	sig_in_func_224	0	no	sig_in_func224	1'd1
225	sig_in_func_225	0	no	sig_in_func225	1'd1
226	sig_in_func_226	0	no	sig_in_func226	1'd1
227	sig_in_func_227	0	no	sig_in_func227	1'd1
235	pro_alonegpio_in0	0	no	pro_alonegpio_out0	1'd1
236	pro_alonegpio_in1	0	no	pro_alonegpio_out1	1'd1
237	pro_alonegpio_in2	0	no	pro_alonegpio_out2	1'd1
238	pro_alonegpio_in3	0	no	pro_alonegpio_out3	1'd1
239	pro_alonegpio_in4	0	no	pro_alonegpio_out4	1'd1
240	pro_alonegpio_in5	0	no	pro_alonegpio_out5	1'd1
241	pro_alonegpio_in6	0	no	pro_alonegpio_out6	1'd1
242	pro_alonegpio_in7	0	no	pro_alonegpio_out7	1'd1
251	-	-	-	clk_i2s_mux	1'd1

## 5.10 IO MUX Pad 列表

表 23 列出了每个 I/O pad 的 IO MUX 功能。

表 23: IO MUX Pad 列表

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Reset	Notes
0	GPIO0	GPIO0	GPIO0	-	-	-	3	R
1	GPIO1	GPIO1	GPIO1	-	-	-	1	R
2	GPIO2	GPIO2	GPIO2	-	-	-	1	R
3	GPIO3	GPIO3	GPIO3	-	-	-	0	R
4	GPIO4	GPIO4	GPIO4	-	-	-	0	R
5	GPIO5	GPIO5	GPIO5	-	-	-	0	R
6	GPIO6	GPIO6	GPIO6	-	-	-	0	R
7	GPIO7	GPIO7	GPIO7	-	-	-	0	R

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Reset	Notes
8	GPIO8	GPIO8	GPIO8	-	SUBSPICS1	-	0	R
9	GPIO9	GPIO9	GPIO9	-	SUBSPIHD	FSPiHD	1	R
10	GPIO10	GPIO10	GPIO10	FSPIIO4	SUBSPICS0	FSPICS0	1	R
11	GPIO11	GPIO11	GPIO11	FSPIIO5	SUBSPID	FSPID	1	R
12	GPIO12	GPIO12	GPIO12	FSPIIO6	SUBSPICLK	FSPICLK	1	R
13	GPIO13	GPIO13	GPIO13	FSPIIO7	SUBSPIQ	FSPiQ	1	R
14	GPIO14	GPIO14	GPIO14	FSPIDQS	SUBSPIWP	FSPiWP	1	R
15	XTAL_32K_P	XTAL_32K_P	GPIO15	U0RTS	-	-	0	R
16	XTAL_32K_N	XTAL_32K_N	GPIO16	U0CTS	-	-	0	R
17	DAC_1	DAC_1	GPIO17	U1TXD	-	-	1	R
18	DAC_2	DAC_2	GPIO18	U1RXD	CLK_OUT3	-	1	R
19	GPIO19	GPIO19	GPIO19	U1RTS	CLK_OUT2	-	0	R
20	GPIO20	GPIO20	GPIO20	U1CTS	CLK_OUT1	-	0	R
21	GPIO21	GPIO21	GPIO21	-	-	-	0	R
26	SPICS1	SPICS1	GPIO26	-	-	-	3	-
27	SPIHD	SPIHD	GPIO27	-	-	-	3	-
28	SPIWP	SPIWP	GPIO28	-	-	-	3	-
29	SPICS0	SPICS0	GPIO29	-	-	-	3	-
30	SPICLK	SPICLK	GPIO30	-	-	-	3	-
31	SPIQ	SPIQ	GPIO31	-	-	-	3	-
32	SPID	SPID	GPIO32	-	-	-	3	-
33	GPIO33	GPIO33	GPIO33	FSPiHD	SUBSPIHD	SPIIO4	1	-
34	GPIO34	GPIO34	GPIO34	FSPICS0	SUBSPICS0	SPIIO5	1	-
35	GPIO35	GPIO35	GPIO35	FSPID	SUBSPID	SPIIO6	1	-
36	GPIO36	GPIO36	GPIO36	FSPICLK	SUBSPICLK	SPIIO7	1	-
37	GPIO37	GPIO37	GPIO37	FSPiQ	SUBSPIQ	SPIDQS	1	-
38	GPIO38	GPIO38	GPIO38	FSPiWP	SUBSPIWP	-	1	-
39	MTCK	MTCK	GPIO39	CLK_OUT3	SUBSPICS1	-	1	-
40	MTDO	MTDO	GPIO40	CLK_OUT2	-	-	1	-
41	MTDI	MTDI	GPIO41	CLK_OUT1	-	-	1	-
42	MTMS	MTMS	GPIO42	-	-	-	1	-
43	U0TXD	U0TXD	GPIO43	CLK_OUT1	-	-	3	-
44	U0RXD	U0RXD	GPIO44	CLK_OUT2	-	-	3	-
45	GPIO45	GPIO45	GPIO45	-	-	-	2	-
46	GPIO46	GPIO46	GPIO46	-	-	-	2	I

### 复位配置

“Reset” 一栏是每个 pad 复位后的默认配置。

- **0** - IE=0（输入关闭）
- **1** - IE=1（输入使能）
- **2** - IE=1, WPD=1（输入使能，下拉电阻使能）



- 3 - IE=1, WPU=1（输入使能，上拉电阻使能）

#### 说明

- R - Pad 通过 RTC IO MUX 具有 RTC/模拟功能。
- I - Pad 只能配置为输入 GPIO。

请参考 [《ESP32-S2 技术规格书》](#) 的附录查看管脚功能的完整表格。

## 5.11 RTC IO MUX 管脚清单

表 24 列出了 RTC 管脚和对应 GPIO pad。

表 24: RTC IO MUX 管脚清单

RTC GPIO Num	GPIO Num	Pad Name	Analog Function			
			1	2	3	4
0	0	TOUCH_PAD0 *	RTC_GPIO0	-	-	sar_i2c_scl_0
1	1	TOUCH_PAD1	RTC_GPIO1	-	-	sar_i2c_sda_0
2	2	TOUCH_PAD2	RTC_GPIO2	-	-	sar_i2c_scl_1
3	3	TOUCH_PAD3	RTC_GPIO3	-	-	sar_i2c_sda_1
4	4	TOUCH_PAD4	RTC_GPIO4	-	-	-
5	5	TOUCH_PAD5	RTC_GPIO5	-	-	-
6	6	TOUCH_PAD6	RTC_GPIO6	-	-	-
7	7	TOUCH_PAD7	RTC_GPIO7	-	-	-
8	8	TOUCH_PAD8	RTC_GPIO8	-	-	-
9	9	TOUCH_PAD9	RTC_GPIO9	-	-	-
10	10	TOUCH_PAD10	RTC_GPIO10	-	-	-
11	11	TOUCH_PAD11	RTC_GPIO11	-	-	-
12	12	TOUCH_PAD12	RTC_GPIO12	-	-	-
13	13	TOUCH_PAD13	RTC_GPIO13	-	-	-
14	14	TOUCH_PAD14	RTC_GPIO14	-	-	-
15	15	X32P	RTC_GPIO15	-	-	-
16	16	X32N	RTC_GPIO16	-	-	-
17	17	PDAC1	RTC_GPIO17	-	-	-
18	18	PDAC2	RTC_GPIO18	-	-	-
19	19	RTC_PAD19	RTC_GPIO19	-	-	-
20	20	RTC_PAD20	RTC_GPIO20	-	-	-
21	21	RTC_PAD21	RTC_GPIO21	-	-	-

**说明：** TOUCH\_PAD0 为内部通道，其模拟功能未引出至外部管脚。

## 5.12 基地址

对于 GPIO 与 RTCIO，都可以通过两个不同的寄存器基地址访问，其基地址如表 25 所示。更多信息，请访问章节 1 系统和存储器。

表 25: GPIO 与 RTCIO 基地址

模块名	访问方式	基地址值
GPIO	PeriBUS1	0x3F404000
	PeriBUS2	0x60004000
IO MUX	PeriBUS2	0x60004000
GPIOSD	PeriBUS2	0x60004F00
RTCIO	PeriBUS1	0x3F408400
	PeriBUS2	0x60008400

## 5.13 寄存器列表

### 5.13.1 GPIO 交换矩阵寄存器列表

请注意，下面的地址是相对于 GPIO 基地址的地址偏移量（相对地址），请参阅章节 5.12 获取有关 GPIO 基地址的信息。

名称	描述	地址	权限
<b>GPIO 配置寄存器</b>			
<a href="#">GPIO_BT_SELECT_REG</a>	GPIO 位选择寄存器	0x0000	读/写
<a href="#">GPIO_OUT_REG</a>	GPIO0 ~ 31 输出寄存器	0x0004	读/写
<a href="#">GPIO_OUT_W1TS_REG</a>	GPIO0 ~ 31 输出置位寄存器	0x0008	只写
<a href="#">GPIO_OUT_W1TC_REG</a>	GPIO0 ~ 31 输出清除寄存器	0x000C	只写
<a href="#">GPIO_OUT1_REG</a>	GPIO32 ~ 53 输出寄存器	0x0010	读/写
<a href="#">GPIO_OUT1_W1TS_REG</a>	GPIO32 ~ 53 输出置位寄存器	0x0014	只写
<a href="#">GPIO_OUT1_W1TC_REG</a>	GPIO32 ~ 53 输出清零寄存器	0x0018	只写
<a href="#">GPIO_SDIO_SELECT_REG</a>	GPIO SDIO 选择寄存器	0x001C	读/写
<a href="#">GPIO_ENABLE_REG</a>	GPIO0 ~ 31 输出使能寄存器	0x0020	读/写
<a href="#">GPIO_ENABLE_W1TS_REG</a>	GPIO0 ~ 31 输出使能置位寄存器	0x0024	只写
<a href="#">GPIO_ENABLE_W1TC_REG</a>	GPIO0 ~ 31 输出使能清零寄存器	0x0028	只写
<a href="#">GPIO_ENABLE1_REG</a>	GPIO32 ~ 53 输出使能寄存器	0x002C	读/写
<a href="#">GPIO_ENABLE1_W1TS_REG</a>	GPIO32 ~ 53 输出使能置位寄存器	0x0030	只写
<a href="#">GPIO_ENABLE1_W1TC_REG</a>	GPIO32 ~ 53 输出使能清零寄存器	0x0034	只写
<a href="#">GPIO_STRAP_REG</a>	Bootstrap 管脚寄存器	0x0038	只读
<a href="#">GPIO_IN_REG</a>	GPIO0 ~ 31 输入寄存器	0x003C	只读
<a href="#">GPIO_IN1_REG</a>	GPIO32 ~ 53 输入寄存器	0x0040	只读
<a href="#">GPIO_PIN0_REG</a>	配置 GPIO pin 0	0x0074	读/写
<a href="#">GPIO_PIN1_REG</a>	配置 GPIO pin 1	0x0078	读/写
<a href="#">GPIO_PIN2_REG</a>	配置 GPIO pin 2	0x007C	读/写
...	...	...	...

名称	描述	地址	权限
GPIO_PIN51_REG	配置 GPIO pin 51	0x0140	读/写
GPIO_PIN52_REG	配置 GPIO pin 52	0x0144	读/写
GPIO_PIN53_REG	配置 GPIO pin 53	0x0148	读/写
GPIO_FUNC0_IN_SEL_CFG_REG	外设 function 0 输入选择寄存器	0x0154	读/写
GPIO_FUNC1_IN_SEL_CFG_REG	外设 function 1 输入选择寄存器	0x0158	读/写
GPIO_FUNC2_IN_SEL_CFG_REG	外设 function 2 输入选择寄存器	0x015C	读/写
...	...	...	...
GPIO_FUNC253_IN_SEL_CFG_REG	外设 function 253 输入选择寄存器	0x0548	读/写
GPIO_FUNC254_IN_SEL_CFG_REG	外设 function 254 输入选择寄存器	0x054C	读/写
GPIO_FUNC255_IN_SEL_CFG_REG	外设 function 255 输入选择寄存器	0x0550	读/写
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 外设输出选择	0x0554	读/写
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 外设输出选择	0x0558	读/写
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 外设输出选择	0x055C	读/写
..	...	...	...
GPIO_FUNC51_OUT_SEL_CFG_REG	GPIO51 外设输出选择	0x0620	读/写
GPIO_FUNC52_OUT_SEL_CFG_REG	GPIO52 外设输出选择	0x0624	读/写
GPIO_FUNC53_OUT_SEL_CFG_REG	GPIO53 外设输出选择	0x0628	读/写
GPIO_CLOCK_GATE_REG	GPIO 时钟门控寄存器	0x062C	读/写
<b>中断配置寄存器</b>			
GPIO_STATUS_W1TS_REG	GPIO0 ~ 31 中断状态置位寄存器	0x0048	只写
GPIO_STATUS_W1TC_REG	GPIO0 ~ 31 中断状态清零寄存器	0x004C	只写
GPIO_STATUS1_W1TS_REG	GPIO32 ~ 53 中断状态置位寄存器	0x0054	只写
GPIO_STATUS1_W1TC_REG	GPIO32 ~ 53 中断状态清零寄存器	0x0058	只写
<b>GPIO 中断源寄存器</b>			
GPIO_STATUS_NEXT_REG	GPIO0 ~ 31 中断源寄存器	0x014C	只读
GPIO_STATUS_NEXT1_REG	GPIO32 ~ 53 中断源寄存器	0x0150	只读
<b>中断状态寄存器</b>			
GPIO_STATUS_REG	GPIO0 ~ 31 中断状态寄存器	0x0044	读/写
GPIO_STATUS1_REG	GPIO32 ~ 53 中断状态寄存器	0x0050	读/写
GPIO_PCPU_INT_REG	GPIO0 ~ 31 PRO_CPU 中断状态寄存器	0x005C	只读
GPIO_PCPU_NMI_INT_REG	GPIO0 ~ 31 PRO_CPU 非屏蔽中断状态寄存器	0x0060	只读
GPIO_PCPU_INT1_REG	GPIO32 ~ 53 PRO_CPU 中断状态寄存器	0x0068	只读
GPIO_PCPU_NMI_INT1_REG	GPIO 32 ~ 53 PRO_CPU 非屏蔽中断状态寄存器	0x006C	只读

### 5.13.2 IO MUX 寄存器列表

请注意，下面的地址是相对于 IO MUX 基地址的地址偏移量（相对地址），请参阅章节 5.12 获取有关 IO MUX 基地址的信息。

名称	描述	地址	访问
IO_MUX_PIN_CTRL	时钟输出配置寄存器	0x0000	读/写
IO_MUX_GPIO0_REG	GPIO0 配置寄存器	0x0004	读/写
IO_MUX_GPIO1_REG	GPIO1 配置寄存器	0x0008	读/写

名称	描述	地址	访问
IO_MUX_GPIO2_REG	GPIO2 配置寄存器	0x000C	读/写
IO_MUX_GPIO3_REG	GPIO3 配置寄存器	0x0010	读/写
IO_MUX_GPIO4_REG	GPIO4 配置寄存器	0x0014	读/写
IO_MUX_GPIO5_REG	GPIO5 配置寄存器	0x0018	读/写
IO_MUX_GPIO6_REG	GPIO6 配置寄存器	0x001C	读/写
IO_MUX_GPIO7_REG	GPIO7 配置寄存器	0x0020	读/写
IO_MUX_GPIO8_REG	GPIO8 配置寄存器	0x0024	读/写
IO_MUX_GPIO9_REG	GPIO9 配置寄存器	0x0028	读/写
IO_MUX_GPIO10_REG	GPIO10 配置寄存器	0x002C	读/写
IO_MUX_GPIO11_REG	GPIO11 配置寄存器	0x0030	读/写
IO_MUX_GPIO12_REG	GPIO12 配置寄存器	0x0034	读/写
IO_MUX_GPIO13_REG	GPIO13 配置寄存器	0x0038	读/写
IO_MUX_GPIO14_REG	GPIO14 配置寄存器	0x003C	读/写
IO_MUX_XTAL_32K_P_REG	XTAL_32K_P 配置寄存器	0x0040	读/写
IO_MUX_XTAL_32K_N_REG	XTAL_32K_N 配置寄存器	0x0044	读/写
IO_MUX_DAC_1_REG	DAC_1 配置寄存器	0x0048	读/写
IO_MUX_DAC_2_REG	DAC_2 配置寄存器	0x004C	读/写
IO_MUX_GPIO_19_REG	GPIO19 配置寄存器	0x0050	读/写
IO_MUX_GPIO_20_REG	GPIO20 配置寄存器	0x0054	读/写
IO_MUX_GPIO_21_REG	GPIO21 配置寄存器	0x0058	读/写
IO_MUX_SPICS1_REG	SPICS1 配置寄存器	0x006C	读/写
IO_MUX_SPIHD_REG	SPIHD 配置寄存器	0x0070	读/写
IO_MUX_SPIWP_REG	SPIWP 配置寄存器	0x0074	读/写
IO_MUX_SPICS0_REG	SPICS0 配置寄存器	0x0078	读/写
IO_MUX_SPICLK_REG	SPICLK 配置寄存器	0x007C	读/写
IO_MUX_SPIQ_REG	SPIQ 配置寄存器	0x0080	读/写
IO_MUX_SPID_REG	SPID 配置寄存器	0x0084	读/写
IO_MUX_GPIO_33_REG	GPIO33 配置寄存器	0x0088	读/写
IO_MUX_GPIO_34_REG	GPIO34 配置寄存器	0x008C	读/写
IO_MUX_GPIO_35_REG	GPIO35 配置寄存器	0x0090	读/写
IO_MUX_GPIO_36_REG	GPIO36 配置寄存器	0x0094	读/写
IO_MUX_GPIO_37_REG	GPIO37 配置寄存器	0x0098	读/写
IO_MUX_GPIO_38_REG	GPIO38 配置寄存器	0x009C	读/写
IO_MUX_MTCK_REG	MTCK 配置寄存器	0x00A0	读/写
IO_MUX_MTDO_REG	MTDO 配置寄存器	0x00A4	读/写
IO_MUX_MTDI_REG	MTDI 配置寄存器	0x00A8	读/写
IO_MUX_MTMS_REG	MTMS 配置寄存器	0x00AC	读/写
IO_MUX_U0TXD_REG	U0TXD 配置寄存器	0x00B0	读/写
IO_MUX_U0RXD_REG	U0RXD 配置寄存器	0x00B4	读/写
IO_MUX_GPIO_45_REG	GPIO45 配置寄存器	0x00B8	读/写
IO_MUX_GPIO_46_REG	GPIO46 配置寄存器	0x00BC	读/写

### 5.13.3 SDM 寄存器列表

请注意，下面的地址是相对于 GPIOSD 基地址的地址偏移量（相对地址），请参阅章节 5.12 获取有关 GPIOSD 基地址的信息。

名称	描述	地址	权限
<b>配置寄存器</b>			
GPIOSD_SIGMADELTA0_REG	SDM0 占空比配置寄存器	0x0000	读/写
GPIOSD_SIGMADELTA1_REG	SDM1 占空比配置寄存器	0x0004	读/写
GPIOSD_SIGMADELTA2_REG	SDM2 占空比配置寄存器	0x0008	读/写
GPIOSD_SIGMADELTA3_REG	SDM3 占空比配置寄存器	0x000C	读/写
GPIOSD_SIGMADELTA4_REG	SDM4 占空比配置寄存器	0x0010	读/写
GPIOSD_SIGMADELTA5_REG	SDM5 占空比配置寄存器	0x0014	读/写
GPIOSD_SIGMADELTA6_REG	SDM6 占空比配置寄存器	0x0018	读/写
GPIOSD_SIGMADELTA7_REG	SDM7 占空比配置寄存器	0x001C	读/写
GPIOSD_SIGMADELTA.CG_REG	时钟门控配置寄存器	0x0020	读/写
GPIOSD_SIGMADELTA_MISC_REG	MISC 寄存器	0x0024	读/写
GPIOSD_SIGMADELTA_VERSION_REG	版本控制寄存器	0x0028	读/写

### 5.13.4 RTC IO MUX 寄存器列表

请注意，下面的地址是相对于 RTCIO 基地址的地址偏移量（相对地址），请参阅章节 5.12 获取有关 RTCIO 基地址的信息。

名称	描述	地址	权限
<b>GPIO 配置 / 数据寄存器</b>			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO 输出寄存器	0x0000	读/写
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO 输出置位寄存器	0x0004	只写
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO 输出清零寄存器	0x0008	只写
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO 输出使能寄存器	0x000C	读/写
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO 输出使能置位寄存器	0x0010	只写
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO 输出使能清零寄存器	0x0014	只写
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO 中断状态寄存器	0x0018	读/写
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO 中断状态置位寄存器	0x001C	只写
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO 中断状态清零寄存器	0x0020	只写
RTCIO_RTC_GPIO_IN_REG	RTC GPIO 输入寄存器	0x0024	只读
RTCIO_RTC_GPIO_PIN0_REG	Pin0 RTC 配置	0x0028	读/写
RTCIO_RTC_GPIO_PIN1_REG	Pin1 RTC 配置	0x002C	读/写
RTCIO_RTC_GPIO_PIN2_REG	Pin2 RTC 配置	0x0030	读/写
RTCIO_RTC_GPIO_PIN3_REG	Pin3 RTC 配置	0x0034	读/写
...	...	...	...
RTCIO_RTC_GPIO_PIN19_REG	Pin19 RTC 配置	0x0074	读/写
RTCIO_RTC_GPIO_PIN20_REG	Pin20 RTC 配置	0x0078	读/写
RTCIO_RTC_GPIO_PIN21_REG	Pin21 RTC 配置	0x007C	读/写
<b>RTC GPIO 功能配置寄存器</b>			
RTCIO_TOUCH_PAD0_REG	Touch pad 0 配置寄存器	0x0084	读/写

名称	描述	地址	权限
RTCIO_TOUCH_PAD1_REG	Touch pad 1 配置寄存器	0x0088	读/写
RTCIO_TOUCH_PAD2_REG	Touch pad 2 配置寄存器	0x008C	读/写
...	...	...	...
RTCIO_TOUCH_PAD13_REG	Touch pad 13 配置寄存器	0x00B8	读/写
RTCIO_TOUCH_PAD14_REG	Touch pad 14 配置寄存器	0x00BC	读/写
RTCIO_XTAL_32P_PAD_REG	32KHz crystal P-pad 配置寄存器	0x00C0	读/写
RTCIO_XTAL_32N_PAD_REG	32KHz crystal N-pad 配置寄存器	0x00C4	读/写
RTCIO_PAD_DAC1_REG	DAC1 配置寄存器	0x00C8	读/写
RTCIO_PAD_DAC2_REG	DAC2 配置寄存器	0x00CC	读/写
RTCIO_RTC_PAD19_REG	Touch pad 19 配置寄存器	0x00D0	读/写
RTCIO_RTC_PAD20_REG	Touch pad 20 配置寄存器	0x00D4	读/写
RTCIO_RTC_PAD21_REG	Touch pad 21 配置寄存器	0x00D8	读/写
RTCIO_XTL_EXT_CTR_REG	晶振断电 GPIO 使能源	0x00E0	读/写
RTCIO_SAR_I2C_IO_REG	RTC I2C pad 选择寄存器	0x00E4	读/写

## 5.14 寄存器

### 5.14.1 GPIO 交换矩阵寄存器

Register 5.1: GPIO\_BT\_SELECT\_REG (0x0000)

31	0
0x000000	
Reset	

GPIO\_BT\_SEL 保留 (读/写)

Register 5.2: GPIO\_OUT\_REG (0x0004)

31	0
0x000000	
Reset	

**GPIO\_OUT\_DATA\_ORIG** 简单 GPIO 输出模式下 GPIO0 ~ GPIO31 的输出值。bit0 ~ bit31 的值分别对应 GPIO0 ~ 31 的输出值，bit22 ~ bit25 无效。(读/写)

Register 5.3: GPIO\_OUT\_W1TS\_REG (0x0008)

GPIO_OUT_W1TS	
31	0
0x000000	
Reset	

**GPIO\_OUT\_W1TS** GPIO0 ~ 31 输出置位寄存器。每一位置 1，[GPIO\\_OUT\\_REG](#) 中的相应位也置 1。  
注：推荐使用此寄存器来置位 [GPIO\\_OUT\\_REG](#)。（只写）

Register 5.4: GPIO\_OUT\_W1TC\_REG (0x000C)

GPIO_OUT_W1TC	
31	0
0x000000	
Reset	

**GPIO\_OUT\_W1TC** GPIO0 ~ 31 输出清零寄存器。每一位置 1，则 [GPIO\\_OUT\\_REG](#) 中的相应位会清零。注：推荐使用此寄存器来清零 [GPIO\\_OUT\\_REG](#)。（只写）

Register 5.5: GPIO\_OUT1\_REG (0x0010)

(reserved)										GPIO_OUT1_DATA_ORIG																						
31											22	21																				0
0	0	0	0	0	0	0	0	0	0	0	0x0000																				Reset	

**GPIO\_OUT1\_DATA\_ORIG** 简单 GPIO 输出模式下 GPIO32 ~ 53 的输出值。bit0 ~ bit13 的值分别对应 GPIO32 ~ GPIO45 的输出值，bit14 ~ bit21 无效。（读/写）



Register 5.6: GPIO\_OUT1\_W1TS\_REG (0x0014)

(reserved)										GPIO_OUT1_W1TS																															
31										22										21																					0
0 0 0 0 0 0 0 0 0 0										0x0000																					Reset										

**GPIO\_OUT1\_W1TS** GPIO32 ~ 53 输出置位寄存器。每一位置 1，则 **GPIO\_OUT1\_REG** 中的相应位也置 1。注：推荐使用此寄存器来置位 **GPIO\_OUT1\_REG**。（只写）

Register 5.7: GPIO\_OUT1\_W1TC\_REG (0x0018)

(reserved)										GPIO_OUT1_W1TC																															
31										22										21																					0
0 0 0 0 0 0 0 0 0 0										0x0000																					Reset										

**GPIO\_OUT1\_W1TC** GPIO32 ~ 53 输出清零寄存器。每一位置 1，则 **GPIO\_OUT1\_REG** 中的相应位会清零。注：推荐使用此寄存器来清零 **GPIO\_OUT1\_REG**。（只写）

Register 5.8: GPIO\_SDIO\_SELECT\_REG (0x001C)

(reserved)																								GPIO_SDIO_SEL															
31																								7								0							
0 0																								0x0								Reset							

**GPIO\_SDIO\_SEL** 保留（读/写）

Register 5.9: GPIO\_ENABLE\_REG (0x0020)

GPIO_ENABLE_DATA																																
31																															0	
0x000000																																Reset

**GPIO\_ENABLE\_DATA** GPIO0 ~ 31 输出使能寄存器。（读/写）





Register 5.13: GPIO\_ENABLE1\_W1TS\_REG (0x0030)

(reserved)											GPIO_ENABLE1_W1TS																					
3122											210																					
0000000000											0x0000																					Reset

**GPIO\_ENABLE1\_W1TS** GPIO32 ~ 53 输出使能置位寄存器。每一位置 1，则 [GPIO\\_ENABLE1\\_REG](#) 中的相应位也置 1。注：推荐使用此寄存器来置位 [GPIO\\_ENABLE1\\_REG](#)。（只写）

Register 5.14: GPIO\_ENABLE1\_W1TC\_REG (0x0034)

(reserved)											GPIO_ENABLE1_W1TC																						
31											22	21																					0
0	0	0	0	0	0	0	0	0	0	0	0x0000																					Reset	

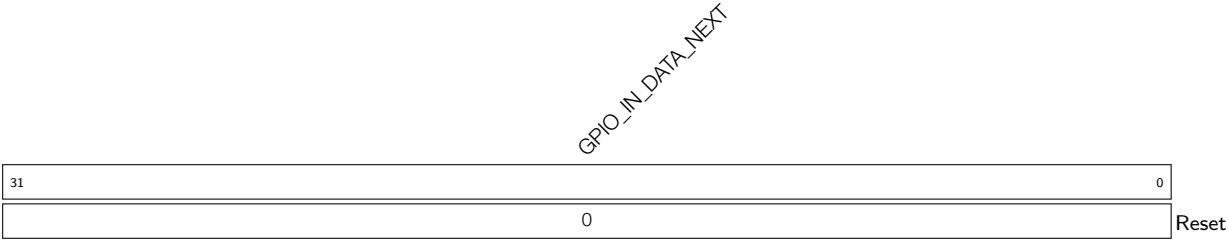
**GPIO\_ENABLE1\_W1TC** GPIO32 ~ 53 输出使能清零寄存器。每一位置 1，则 [GPIO\\_ENABLE1\\_REG](#) 中的相应位会清零。注：推荐使用此寄存器清零 [GPIO\\_ENABLE1\\_REG](#)。（只写）

Register 5.15: GPIO\_STRAP\_REG (0x0038)

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

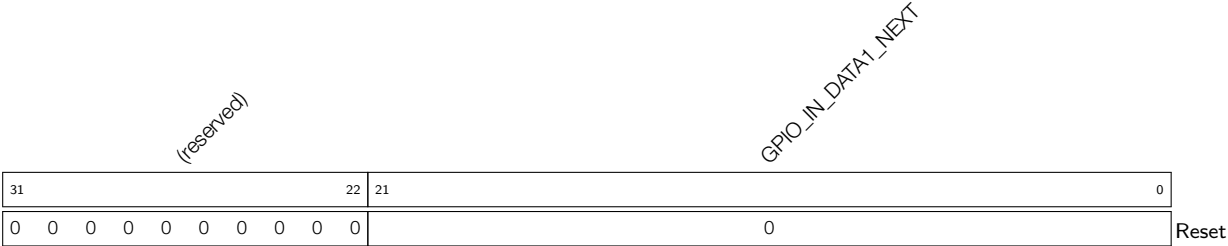
**GPIO\_STRAPPING** GPIO Strapping 值：bit4 ~ bit2 分别对应 GPIO45、GPIO0 和 GPIO46。（只读）

Register 5.16: GPIO\_IN\_REG (0x003C)



**GPIO\_IN\_DATA\_NEXT** GPIO00 ~ 31 输入值。每个 bit 代表 pad 的片外输入值，比如片外引脚为高电平，此 bit 值应为 1；片外引脚为低电平，此 bit 值应为 0。（只读）

Register 5.17: GPIO\_IN1\_REG (0x0040)



**GPIO\_IN\_DATA1\_NEXT** GPIO32 ~ 53 输入值。每个 bit 代表 pad 的片外输入值。（只读）

Register 5.18: GPIO\_PIN $n$ \_REG ( $n$ :0-53) (0x0074+4\* $n$ )

(reserved)																GPIO_PIN <sub>n</sub> _INT_ENA				GPIO_PIN <sub>n</sub> _CONFIG		GPIO_PIN <sub>n</sub> _WAKEUP_ENABLE		(reserved)		GPIO_PIN <sub>n</sub> _SYNC1_BYPASS		GPIO_PIN <sub>n</sub> _PAD_DRIVER		GPIO_PIN <sub>n</sub> _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	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset			

**GPIO\_PIN $n$ \_SYNC2\_BYPASS** 使能 GPIO 输入信号第二拍为上升沿或下降沿同步。0: 关闭同步; 1: 下降沿同步; 2 或 3: 上升沿同步。(读/写)

**GPIO\_PIN $n$ \_PAD\_DRIVER** pad 驱动选择。0: 正常输出; 1: 开漏输出。(读/写)

**GPIO\_PIN $n$ \_SYNC1\_BYPASS** 使能 GPIO 输入信号第一拍为上升沿或下降沿同步。0: 关闭同步; 1: 下降沿同步; 2 或 3: 上升沿同步。(读/写)

**GPIO\_PIN $n$ \_INT\_TYPE** 中断类型选择。(读/写)

- 0: 禁用 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

**GPIO\_PIN $n$ \_WAKEUP\_ENABLE** 使能 GPIO 唤醒, 仅能将 CPU 从 Light-sleep 模式唤醒。(读/写)

**GPIO\_PIN $n$ \_CONFIG** 保留。(读/写)

**GPIO\_PIN $n$ \_INT\_ENA** 中断使能位。bit13: 使能 CPU 中断; bit14: 使能 CPU 非屏蔽中断。(读/写)

Register 5.19: GPIO\_FUNC $n$ \_IN\_SEL\_CFG\_REG ( $n$ :0-255) (0x0154+4\* $n$ )

(reserved)																								GPIO_SIG <sub>n</sub> _IN_SEL			GPIO_FUNC <sub>n</sub> _IN_INV_SEL			GPIO_FUNC <sub>n</sub> _IN_SEL		
31																								8	7	6	5	0				
0 0																																

**GPIO\_FUNC $n$ \_IN\_SEL** 外设输入  $m$  的选择控制位。此位选择 1 个 GPIO 交换矩阵输入管脚与信号连接, 或者选择 0x38 与恒高电平输入信号连接, 或者选择 0x3C 与恒低电平输入信号连接。(读/写)

**GPIO\_FUNC $n$ \_IN\_INV\_SEL** 反转输入值。1: 反转; 0: 不反转。(读/写)

**GPIO\_SIG $n$ \_IN\_SEL** 旁路 GPIO 交换矩阵。1: 通过 GPIO 交换矩阵; 0: 直接通过 IO MUX 连接信号与外设。(读/写)

Register 5.20: GPIO\_FUNC $n$ \_OUT\_SEL\_CFG\_REG ( $n$ :0-53) (0x0554+4\* $n$ )

(reserved)																								GPIO_FUNC <sub>n</sub> _OEN_INV_SEL				GPIO_FUNC <sub>n</sub> _OEN_SEL				GPIO_FUNC <sub>n</sub> _OUT_INV_SEL				GPIO_FUNC <sub>n</sub> _OUT_SEL			
31												12				11	10	9	8	0																			
0 0 0 0 0 0 0 0 0 0 0 0												0 0 0 0				0x100												Reset											

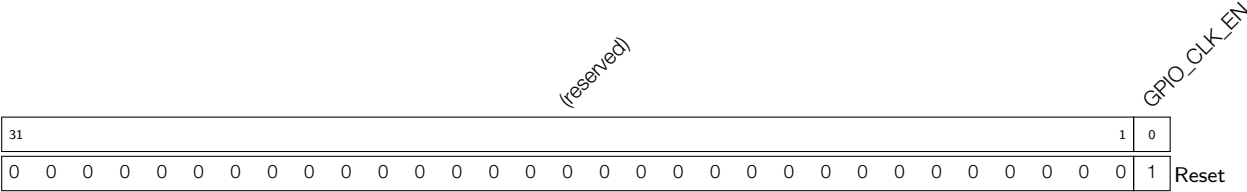
**GPIO\_FUNC $n$ \_OUT\_SEL** GPIO 输出  $n$  的选择控制位。值为  $s$  ( $0 \leq s < 256$ ) 连接外设输出  $s$  与 GPIO 输出  $n$ 。值为 256 选择 GPIO\_OUT\_REG/GPIO\_OUT1\_REG 和 GPIO\_ENABLE\_REG/GPIO\_ENABLE1\_REG 的 bit  $n$  作为输出值和输出使能。(读/写)

**GPIO\_FUNC $n$ \_OUT\_INV\_SEL** 0: 不反转输出值; 1: 反转输出值。(读/写)

**GPIO\_FUNC $n$ \_OEN\_SEL** 0: 采用外设的输出使能信号; 1: 强制使用 GPIO\_ENABLE\_REG 的 bit  $n$  用作输出使能信号。(读/写)

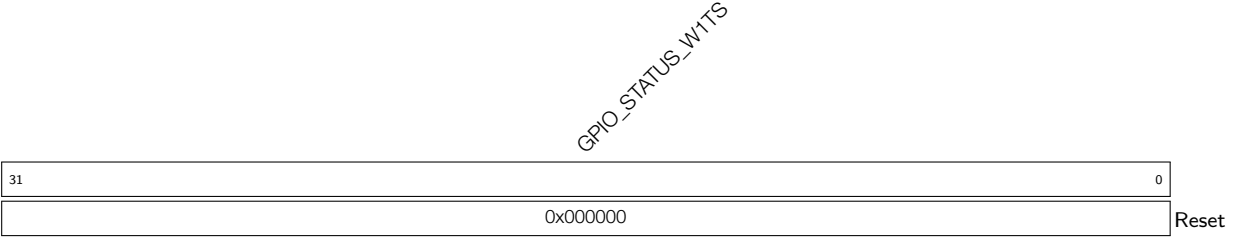
**GPIO\_FUNC $n$ \_OEN\_INV\_SEL** 0: 不反转输出使能信号; 1: 反转输出使能信号。(读/写)

Register 5.21: GPIO\_CLOCK\_GATE\_REG (0x062C)



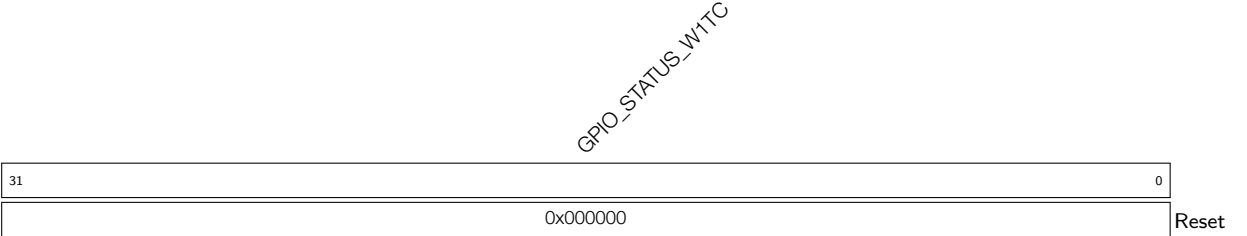
**GPIO\_CLK\_EN** 时钟门控使能。此位置 1，则时钟自由运转。(读/写)

Register 5.22: GPIO\_STATUS\_W1TS\_REG (0x0048)



**GPIO\_STATUS\_W1TS** GPIO0 ~ 31 中断状态置位寄存器。每位置 1，则 [GPIO\\_STATUS\\_INTERRUPT](#) 中的相应位也置 1。注：推荐使用此寄存器来置位 [GPIO\\_STATUS\\_INTERRUPT](#)。(只写)

Register 5.23: GPIO\_STATUS\_W1TC\_REG (0x004C)



**GPIO\_STATUS\_W1TC** GPIO0 ~ 31 中断状态清除寄存器。每一位置 1，则 [GPIO\\_STATUS\\_INTERRUPT](#) 中的相应位也会清零。注：推荐使用此寄存器来清零 [GPIO\\_STATUS\\_INTERRUPT](#)。(只写)

Register 5.24: GPIO\_STATUS1\_W1TS\_REG (0x0054)

(reserved)											GPIO_STATUS1_W1TS																					
3122											210																					
0000000000											0x0000																					Reset

**GPIO\_STATUS1\_W1TS** GPIO32 ~ 53 中断状态置位寄存器。每一位置 1，[GPIO\\_STATUS\\_INTERRUPT1](#) 中相应位也置 1。注：推荐使用此寄存器来置位 [GPIO\\_STATUS\\_INTERRUPT1](#)。（只写）

Register 5.25: GPIO\_STATUS1\_W1TC\_REG (0x0058)

(reserved)											GPIO_STATUS1_W1TC																				
31											22	21																			0
0	0	0	0	0	0	0	0	0	0	0	0x0000																				Reset

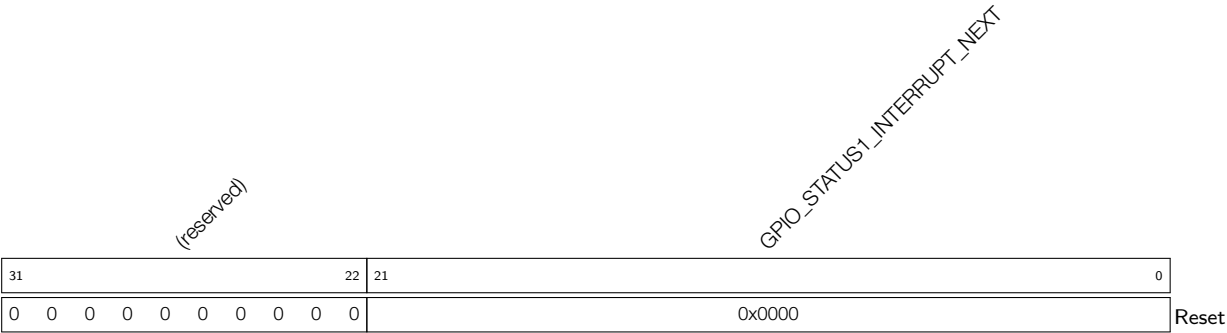
**GPIO\_STATUS1\_W1TC** GPIO32 ~ 53 中断状态清除寄存器。每一位置 1，则 [GPIO\\_STATUS\\_INTERRUPT1](#) 中的相应位也将清零。注：推荐使用此寄存器来清零 [GPIO\\_STATUS\\_INTERRUPT1](#)。（只写）

Register 5.26: GPIO\_STATUS\_NEXT\_REG (0x014C)

GPIO_STATUS_INTERRUPT_NEXT																															
31																															0
0x000000																															
Reset																															

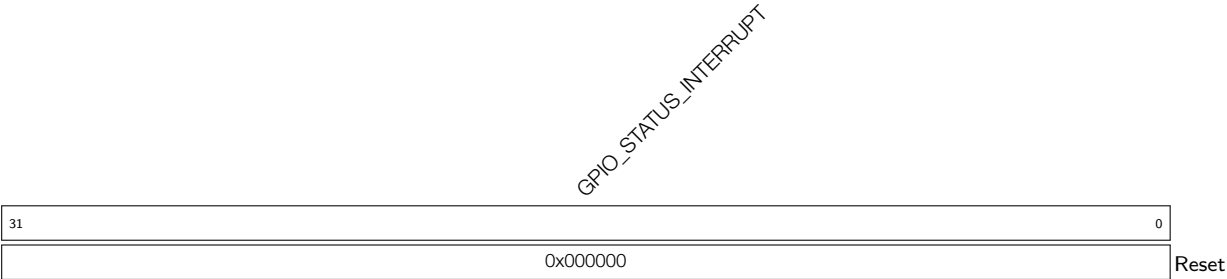
**GPIO\_STATUS\_INTERRUPT\_NEXT** GPIO0 ~ 31 中断源信号，可以设置为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。（只读）

Register 5.27: GPIO\_STATUS\_NEXT1\_REG (0x0150)



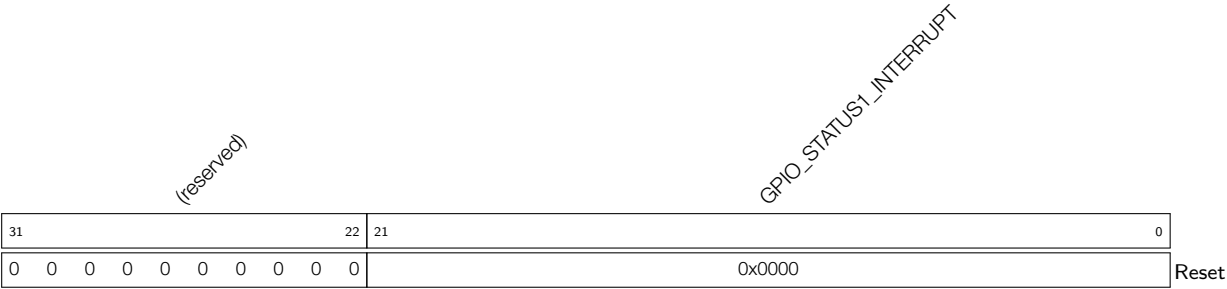
**GPIO\_STATUS1\_INTERRUPT\_NEXT** GPIO32 ~ 53 的中断源信号。(只读)

Register 5.28: GPIO\_STATUS\_REG (0x0044)



**GPIO\_STATUS\_INTERRUPT** GPIO0 ~ 31 中断状态寄存器。(读/写)

Register 5.29: GPIO\_STATUS1\_REG (0x0050)



**GPIO\_STATUS1\_INTERRUPT** GPIO32 ~ 53 中断状态寄存器。(读/写)



Register 5.30: GPIO\_PCPU\_INT\_REG (0x005C)

GPIO_PROCPU_INT																															
0																															
0x000000																															
Reset																															

**GPIO\_PROCPU\_INT** GPIO0 ~ 31 PRO\_CPU 中断状态。如果 [GPIO\\_PIN<sub>n</sub>\\_REG](#) 中 bit13 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 bit 的中断状态一致。(只读)

Register 5.31: GPIO\_PCPU\_NMI\_INT\_REG (0x0060)

GPIO_PROCPU_NMI_INT																															
31																															0
0x000000																															
Reset																															

**GPIO\_PROCPU\_NMI\_INT** GPIO0 ~ 31 PRO\_CPU 非屏蔽中断状态寄存器。如果 [GPIO\\_PIN<sub>n</sub>\\_REG](#) 中 bit14 高电平有效，即使能 CPU 非屏蔽中断，则此寄存器所示的中断状态应与 [GPIO\\_STATUS\\_REG](#) 中相应 bit 的中断状态一致。(只读)

Register 5.32: GPIO\_PCPU\_INT1\_REG (0x0068)

(reserved)										GPIO_PROCPU1_INT																					
31											22	21											0								
0	0	0	0	0	0	0	0	0	0	0	0x0000																				Reset

**GPIO\_PROCPU1\_INT** GPIO32 ~ 53 PRO\_CPU 中断状态寄存器。如果 [GPIO\\_PIN<sub>n</sub>\\_REG](#) 中 bit13 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\\_STATUS1\\_REG](#) 中相应 bit 的中断状态一致。(只读)

Register 5.33: GPIO\_PROCPU\_NMI\_INT1\_REG (0x006C)

(reserved)										GPIO_PROCPU_NMI1_INT																															
31										22										21																					0
0 0 0 0 0 0 0 0 0 0										0x0000																					Reset										

**GPIO\_PROCPU\_NMI1\_INT** GPIO32 ~ 53 PRO\_CPU 非屏蔽中断状态寄存器。如果 **GPIO\_PIN<sub>n</sub>\_REG** 中 bit14 高电平有效，即使能 CPU 非屏蔽中断，则此寄存器所示的中断状态应与 **GPIO\_STATUS1\_REG** 中相应 bit 的中断状态一致。（只读）

5.14.2 IO MUX 寄存器

Register 5.34: IO\_MUX\_PIN\_CTRL (0x0000)

(reserved)																IO_MUX_PAD_POWER_CTRL																IO_MUX_SWITCH_PRT_NUM																IO_MUX_PIN_CTRL_CLK3																IO_MUX_PIN_CTRL_CLK2																IO_MUX_PIN_CTRL_CLK1															
31																16																15				14				12				11				8				7				4				3				0																															
0x0																0x0				0x2				0x0				0x0				0x0				0x0				Reset																																																							

**IO\_MUX\_PIN\_CTRL\_CLK<sub>x</sub>** 配置 I2S0 外设时钟输出到：  
CLK\_OUT1，配置 IO\_MUX\_PIN\_CTRL[3:0] = 0x0；  
CLK\_OUT2，配置 IO\_MUX\_PIN\_CTRL[3:0] = 0x0 and IO\_MUX\_PIN\_CTRL[7:4] = 0x0；  
CLK\_OUT3，配置 IO\_MUX\_PIN\_CTRL[3:0] = 0x0 and IO\_MUX\_PIN\_CTRL[11:8] = 0x0。  
**说明：**  
只能有上述配置组合。  
CLK\_OUT1 ~ 3 可在 [IO\\_MUX Pad 列表](#) 中查询。

**IO\_MUX\_SWITCH\_PRT\_NUM** IO Pad 电源切换延时，延时单位为一个 APB 时钟周期。

**IO\_MUX\_PAD\_POWER\_CTRL** 选择 GPIO33 ~ 37 的电源电压。1：选择 VDD\_SPI 1.8 V 供电；0：选择 VDD3P3\_CPU 3.3 V 供电。

Register 5.35: IO\_MUX\_  $n$ \_REG ( $n$ : GPIO0-GPIO21, GPIO26-GPIO46) (0x0010+4\* $n$ )

(reserved)																IO_MUX_FILTER_EN		IO_MUX_MCU_SEL		IO_MUX_FUN_DRV		IO_MUX_FUN_IE		IO_MUX_FUN_WPU		(reserved)		IO_MUX_MCU_IE		IO_MUX_MCU_WPU		IO_MUX_MCU_WPD		IO_MUX_SLP_SEL		IO_MUX_MCU_OE											
31																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																0x0		0x0		0x2		0		0		0		00		0		0		0		0		0		0		Reset					

**IO\_MUX\_MCU\_OE** 睡眠模式下 pad 的输出使能。1: 输出使能; 0: 输出关闭。(读/写)

**IO\_MUX\_SLP\_SEL** Pad 的睡眠模式选择。置 1 将使能睡眠模式。(读/写)

**IO\_MUX\_MCU\_WPD** 睡眠模式下 pad 的下拉使能。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)

**IO\_MUX\_MCU\_WPU** 睡眠模式下 pad 的上拉使能。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)

**IO\_MUX\_MCU\_IE** 睡眠模式下 pad 的输入使能。1: 输入使能; 0: 输入关闭。(读/写)

**IO\_MUX\_FUN\_WPD** Pad 的下拉使能。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)

**IO\_MUX\_FUN\_WPU** Pad 的上拉使能。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)

**IO\_MUX\_FUN\_IE** Pad 的输入使能。1: 输入使能; 0: 输入关闭。(读/写)

**IO\_MUX\_FUN\_DRV** 选择 Pad 驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

**IO\_MUX\_MCU\_SEL** 为信号选择 IO MUX 功能。0: 选择 Function 1; 1: 选择 Function 2; 以此类推。(读/写)

**IO\_MUX\_FILTER\_EN** Pad 输入信号滤波使能。1: 滤波使能; 0: 滤波关闭。(读/写)

### 5.14.3 SDM 寄存器

Register 5.36: GPIOSD\_SIGMADELTA $n$ \_REG ( $n$ : 0-7) (0x0000+4\* $n$ )

(reserved)																GPIO_SD <sub>n</sub> _PRESCALE								GPIO_SD <sub>n</sub> _IN																																															
31																16																15																8								7								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0xff																0x0																Reset																							

**GPIOSD\_SD $n$ \_IN** 配置 SDM 输出信号的占空比。(读/写)

**GPIOSD\_SD $n$ \_PRESCALE** 配置 APB\_CLK 分频系数。(读/写)

### Register 5.37: GPIO\_SD\_SIGMADELTA\_CG\_REG (0x0020)

[illegible]

**GPIO\_SD\_CLK\_EN** 使能 SDM 配置寄存器的时钟。(读/写)

### Register 5.38: GPIO\_SD\_SIGMADELTA\_MISC\_REG (0x0024)

Diagram of the GPIO pin configuration register. The register is 32 bits wide. Bit 31 is labeled `GPIO0_SPL_SWAP`. Bit 30 is labeled `GPIO0_FUNCTION_CLK_EN`. Bits 29 through 0 are labeled `(reserved)`. The register value is shown as 00000000000000000000000000000000.

**GPIO\_SD\_FUNCTION\_CLK\_EN** 使能 SDM 的时钟。(读/写)

**GPIOSD\_SPI\_SWAP** 保留。(读/写)

### Register 5.39: GPIO\_SD\_SIGMADELTA\_VERSION\_REG (0x0028)

(reserved)				GPIO_SD_GPIO_SD_DATE																												
31	28	27	0																													
0	0	0	0	0x1802260																												Reset

**GPIO\_SD\_GPIO\_SD\_DATE** 版本控制寄存器。(读/写)

5.14.4 RTC IO MUX 寄存器

Register 5.40: RTCIO\_RTC\_GPIO\_OUT\_REG (0x0000)

RTCIO_GPIO_OUT_DATA										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_OUT\_DATA** GPIO0 ~ 21 输出寄存器。bit10 对应 GPIO0, bit11 对应 GPIO1, 以此类推。(读/写)

Register 5.41: RTCIO\_RTC\_GPIO\_OUT\_W1TS\_REG (0x0004)

RTCIO_GPIO_OUT_DATA_W1TS										(reserved)											
31										10	9									0	
0										0 0 0 0 0 0 0 0 0 0										0	Reset

**RTCIO\_GPIO\_OUT\_DATA\_W1TS** GPIO0 ~ 21 输出置位寄存器。每一位置 1, **RTCIO\_RTC\_GPIO\_OUT\_REG** 中相应位也置 1。注: 推荐使用此寄存器来置位 **RTCIO\_RTC\_GPIO\_OUT\_REG**。(只写)

Register 5.42: RTCIO\_RTC\_GPIO\_OUT\_W1TC\_REG (0x0008)

RTCIO_GPIO_OUT_DATA_W1TC										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_OUT\_DATA\_W1TC** GPIO0 ~ 21 输出清零寄存器。每一位置 1, 则 **RTCIO\_RTC\_GPIO\_OUT\_REG** 中相应位将被清零。注: 推荐使用此寄存器来清零 **RTCIO\_RTC\_GPIO\_OUT\_REG**。(只写)

Register 5.43: RTCIO\_RTC\_GPIO\_ENABLE\_REG (0x000C)

RTCIO_GPIO_ENABLE										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_ENABLE** GPIO0 ~ 21 输出使能。bit10 对应 GPIO0, bit11 对应 GPIO1, 以此类推。此位置 1, 即该 GPIO pad 为输出。(读/写)

Register 5.44: RTCIO\_RTC\_GPIO\_ENABLE\_W1TS\_REG (0x0010)

RTCIO_GPIO_ENABLE_W1TS										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_ENABLE\_W1TS** GPIO0 ~ 21 输出使能置位寄存器。每一位置 1, 则 [RTCIO\\_RTC\\_GPIO\\_ENABLE\\_REG](#) 中相应位也将置 1。注: 推荐使用此寄存器来置位 [RTCIO\\_RTC\\_GPIO\\_ENABLE\\_REG](#)。(只写)

Register 5.45: RTCIO\_RTC\_GPIO\_ENABLE\_W1TC\_REG (0x0014)

RTCIO_GPIO_ENABLE_W1TC										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_ENABLE\_W1TC** GPIO0 ~ 21 输出使能清零寄存器。每一位置 1, 则 [RTCIO\\_RTC\\_GPIO\\_ENABLE\\_REG](#) 中相应位将被清零。注: 推荐使用此寄存器来清零 [RTCIO\\_RTC\\_GPIO\\_ENABLE\\_REG](#)。(只写)

Register 5.46: RTCIO\_RTC\_GPIO\_STATUS\_REG (0x0018)

RTCIO_GPIO_STATUS_INT										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_STATUS\_INT** GPIO0 ~ 21 中断状态寄存器。bit10 对应 GPIO0，bit11 对应 GPIO1，以此类推。此寄存器应同时与 **RTCIO\_RTC\_GPIO\_PIN<sub>n</sub>\_REG** 寄存器中的 **RTCIO\_RTC\_GPIO\_PIN<sub>n</sub>\_INT\_TYPE** 中断类型配合使用。0：代表没有中断；1：代表有相应中断。（读/写）

Register 5.47: RTCIO\_RTC\_GPIO\_STATUS\_W1TS\_REG (0x001C)

RTCIO_GPIO_STATUS_INT_W1TS										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_STATUS\_INT\_W1TS** GPIO0 ~ 21 中断状态置位寄存器。每一位置 1，则 **RTCIO\_GPIO\_STATUS\_INT** 中相应位也将置 1。注：推荐使用此寄存器来置位 **RTCIO\_GPIO\_STATUS\_INT**。（只写）

Register 5.48: RTCIO\_RTC\_GPIO\_STATUS\_W1TC\_REG (0x0020)

RTCIO_GPIO_STATUS_INT_W1TC										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_STATUS\_INT\_W1TC** GPIO0 ~ 21 中断状态清零寄存器。每一位置 1，则 **RTCIO\_GPIO\_STATUS\_INT** 中的相应位也将清零。注：推荐使用此寄存器来清零 **RTCIO\_GPIO\_STATUS\_INT**。（只写）

Register 5.49: RTCIO\_RTC\_GPIO\_IN\_REG (0x0024)

RTCIO_GPIO_IN_NEXT										(reserved)										
31										10	9									0
0										0 0 0 0 0 0 0 0 0 0 0 0										Reset

**RTCIO\_GPIO\_IN\_NEXT** GPIO0 ~ 21 输入值。bit10 对应 GPIO0，bit11 对应 GPIO1，以此类推。每个 bit 代表 pad 的片外输入值，比如片外引脚为高电平，此 bit 值应为 1，片外引脚为低电平，此 bit 值应为 0。（只读）

Register 5.50: RTCIO\_RTC\_GPIO\_PIN<sub>n</sub>\_REG (n: 0-21) (0x0028+4\*n)

(reserved)										RTCIO_GPIO_PIN <sub>n</sub> _WAKEUP_ENABLE									
										RTCIO_GPIO_PIN <sub>n</sub> _INT_TYPE									
										(reserved)									
										RTCIO_GPIO_PIN <sub>n</sub> _PAD_DRIVER									
										(reserved)									
31										11	10	9	7	6	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

**RTCIO\_GPIO\_PIN<sub>n</sub>\_PAD\_DRIVER** Pad 驱动选择寄存器。0：正常输出；1：开漏输出。（读/写）

**RTCIO\_GPIO\_PIN<sub>n</sub>\_INT\_TYPE** GPIO 中断类型选择寄存器。（读/写）

- 0：禁用 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

**RTCIO\_GPIO\_PIN<sub>n</sub>\_WAKEUP\_ENABLE** GPIO 唤醒使能。只能将 ESP32-S2 从 Light-sleep 中唤醒。（读/写）



**Register 5.51: RTCIO\_TOUCH\_PAD $n$ \_REG ( $n$ : 0-14) (0x0084+4\* $n$ )**

[illegible]

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_FUN\_IE** 工作模式下输入使能。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_SLP\_OE** 睡眠模式下输出使能。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_SLP\_IE** 睡眠模式下输入使能。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_SLP\_SEL** 0: 无睡眠模式; 1: 使能睡眠模式。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_FUN\_SEL** Function 选择 (读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_MUX\_SEL** 连接 RTC pad 输入与数字 pad 输入，可以置 0。（读/写）

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_XPD** 触摸传感器上电。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_TIE\_OPT** 默认触摸传感器初始电压位。0：拉高；1：拉低。（读/写）

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_START** 启动触摸传感器。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_DAC** 触摸传感器斜率控制。每个触摸 pad 为 3-bit，默认为 0x4。（读/写）

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_RUE** 使能 pad 下拉。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)

**RTCIO\_TOUCH\_PAD<sub>n</sub>\_RDE** 使能 pad 上拉。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)

**RTCIO\_TOUCH\_PAD $n$ \_DRV** 选择 pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

Register 5.52: RTCIO\_XTAL\_32P\_PAD\_REG (0x00C0)

(reserved)				RTCIO_X32P_DRV				RTCIO_X32P_PDE				RTCIO_X32P_RUE				(reserved)				RTCIO_X32P_MUX_SEL				RTCIO_X32P_FUN_SEL				RTCIO_X32P_SLP_SEL				RTCIO_X32P_SLP_IE				RTCIO_X32P_SLP_OE				RTCIO_X32P_FUN_IE				(reserved)																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
31	30	29	28	27	26						20	19	18	17	16	15	14	13	12													0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
0		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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- RTCIO\_X32P\_FUN\_IE 使能工作模式下 pad 输入。(读/写)
- RTCIO\_X32P\_SLP\_OE 使能睡眠模式下 pad 输出。(读/写)
- RTCIO\_X32P\_SLP\_IE 使能睡眠模式下 pad 输入。(读/写)
- RTCIO\_X32P\_SLP\_SEL 1: 使能睡眠模式; 0: 关闭睡眠模式。(读/写)
- RTCIO\_X32P\_FUN\_SEL Function 选择。(读/写)
- RTCIO\_X32P\_MUX\_SEL 1: 选择使用 RTC GPIO; 0: 选择使用数字 GPIO。(读/写)
- RTCIO\_X32P\_RUE Pad 下拉使能。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)
- RTCIO\_X32P\_RDE Pad 上拉使能。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)
- RTCIO\_X32P\_DRV 选择 pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

### Register 5.53: RTCIO\_XTAL\_32N\_PAD\_REG (0x00C4)

[illegible]

**RTCIO\_X32N\_FUN\_IE** 工作模式下输入使能。(读/写)

**RTCIO\_X32N\_SLP\_OE** 睡眠模式下输出使能。(读/写)

**RTCIO\_X32N\_SLP\_IE** 睡眠模式下输入使能。(读/写)

**RTCIO\_X32N\_SLP\_SEL** 1: 使能睡眠模式; 0: 关闭睡眠模式。(读/写)

**RTCIO\_X32N\_FUN\_SEL** Function 选择。(读/写)

**RTCIO\_X32N\_MUX\_SEL** 1: 选择使用 RTC GPIO; 0: 选择使用数字 GPIO。(读/写)

**RTCIO\_X32N\_RUE** Pad 下拉使能。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)

**RTCIO\_X32N\_RDE** Pad 上拉使能。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)

**RTCIO\_X32N\_DRV** 选择 pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

## 124

ESP32-S2 TRM (预发布 V0.2)

## 反馈文档意见

**RTCIO\_PDAC1\_DRV** 选择 pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

## 125

ESP32-S2 TRM (预发布 V0.2)

## 反馈文档意见

**RTCIO\_PDAC2\_DRV** 选择 pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

## 126

ESP32-S2 TRM (预发布 V0.2)

## 反馈文档意见

**RTCIO\_RTC\_PAD19\_DRV** 选择 Pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

### Register 5.57: RTCIO\_RTC\_PAD20\_REG (0x00D4)

[illegible]

**RTCIO\_RTC\_PAD20\_FUN\_IE** 工作模式下输入使能。(读/写)

**RTCIO\_RTC\_PAD20\_SLP\_OE** 睡眠模式下输出使能。(读/写)

**RTCIO RTC PAD20 SLP IE** 睡眠模式下输入使能。(读/写)

**RTCIO\_RTC\_PAD20\_SLP\_SEL** 1: 使能睡眠模式; 0: 关闭睡眠模式。(读/写)

**RTCIO\_RTC\_PAD20\_FUN\_SEL** 功能选择。(读/写)

**RTCIO\_RTC\_PAD20\_MUX\_SEL** 1: 选择使用 RTC GPIO; 0: 选择使用数字 GPIO。(读/写)

**RTCIO\_RTC\_PAD20\_RUE** Pad 下拉使能。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)

**RTCIO\_RTC\_PAD20\_RDE** Pad 上拉使能。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)

**RTCIO\_RTC\_PAD20\_DRV** 选择 Pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

**Register 5.58: RTCIO\_RTC\_PAD21\_REG (0x00D8)**

[illegible]

**RTCIO\_RTC\_PAD21\_FUN\_IE** 工作模式下输入使能。(读/写)

**RTCIO\_RTC\_PAD21\_SLP\_OE** 睡眠模式下输出使能。(读/写)

**RTCIO\_RTC\_PAD21\_SLP\_IE** 睡眠模式下输入使能。(读/写)

**RTCIO\_RTC\_PAD21\_SLP\_SEL** 1: 使能睡眠模式; 0: 关闭睡眠模式。(读/写)

**RTCIO\_RTC\_PAD21\_FUN\_SEL** 功能选择。(读/写)

**RTCIO\_RTC\_PAD21\_MUX\_SEL** 1: 选择使用 RTC GPIO; 0: 选择使用数字 GPIO。(读/写)

**RTCIO\_RTC\_PAD21\_RUE** Pad 下拉使能。1: 内部下拉使能; 0: 内部下拉关闭。(读/写)

**RTCIO\_RTC\_PAD21\_RDE** Pad 上拉使能。1: 内部上拉使能; 0: 内部上拉关闭。(读/写)

**RTCIO\_RTC\_PAD21\_DRV** 选择 Pad 的驱动强度。0: ~5 mA; 1: ~10 mA; 2: ~20 mA; 3: ~40 mA。(读/写)

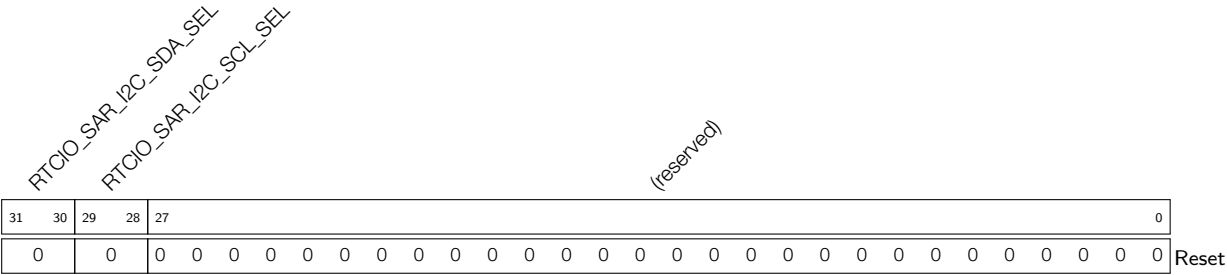
### Register 5.59: RTCIO\_XTL\_EXT\_CTR\_REG (0x00E0)

[illegible]

**RTCIO\_XTL\_EXT\_CTR\_SEL** 选择睡眠模式下外部晶振断电使能信号。0: 选择 GPIO0; 1: 选择 GPIO1, 以此类推。被选择管脚的值异或 RTC\_CNTL\_EXT\_XTL\_CONF\_REG[30] 上的逻辑值时晶振断电使能信号。(读/写)



Register 5.60: RTCIO\_SAR\_I2C\_IO\_REG (0x00E4)



**RTCIO\_SAR\_I2C\_SCL\_SEL** 选择 RTC I2C SCL 信号连接的 pad。0: 选择 TOUCH\_PAD0; 1: 选择 TOUCH\_PAD2。(读/写)

**RTCIO\_SAR\_I2C\_SDA\_SEL** 选择 RTC I2C SDA 信号连接的 pad。0: 选择 TOUCH\_PAD1; 1: 选择 TOUCH\_PAD3。(读/写)

## 6. 系统寄存器

### 6.1 概述

ESP32-S2 集成了丰富的外设，且允许对不同外设模块进行独立控制，从而在保持最佳性能的同时将功耗降至最低。具体来说，ESP32-S2 设计了一系列系统配置寄存器，用于芯片的时钟管理（时钟门控）、功耗管理、外设模块及核心模块配置。本章将简要例举这些系统配置寄存器及其功能。

### 6.2 主要特性

ESP32-S2 的系统寄存器可用于控制以下外设和模块：

- 系统和存储器
- 复位和时钟
- 中断矩阵
- eFuse 控制器
- 低功耗管理寄存器
- 外设时钟门控和复位

### 6.3 功能描述

#### 6.3.1 系统和存储器寄存器

以下系统寄存器用于系统和存储器的配置，例如 cache 配置和存储器掉电控制等。更多信息，请见章节 [1 系统和存储器](#)。

- [SYSTEM\\_ROM\\_CTRL\\_0\\_REG](#)
- [SYSTEM\\_ROM\\_CTRL\\_1\\_REG](#)
- [SYSTEM\\_SRAM\\_CTRL\\_0\\_REG](#)
- [SYSTEM\\_SRAM\\_CTRL\\_1\\_REG](#)
- [SYSTEM\\_SRAM\\_CTRL\\_2\\_REG](#)
- [SYSTEM\\_RSA\\_PD\\_CTRL\\_REG](#)
- [SYSTEM\\_MEM\\_PD\\_MASK\\_REG](#)
- [SYSTEM\\_CACHE\\_CONTROL\\_REG](#)
- [SYSTEM\\_BUSTOEXTMEM\\_ENA\\_REG](#)
- [SYSTEM\\_EXTERNAL\\_DEVICE\\_ENCRYPT\\_DECRYPT\\_CONTROL\\_REG](#)

## ROM 功耗控制

寄存器 `SYSTEM_ROM_CTRL_0_REG` 和 `SYSTEM_ROM_CTRL_1_REG` 用以控制 ESP32-S2 内部 ROM 存储的功耗，具体来说：

- 寄存器 `SYSTEM_ROM_CTRL_0_REG` 中 `SYSTEM_ROM_FO` 的相应位可用以控制 ROM 不同 block 的时钟门控。
- 寄存器 `SYSTEM_ROM_CTRL_1_REG` 中 `SYSTEM_ROM_FORCE_PD` 的相应位可用以控制 ROM 不同 block 的掉电。
- 寄存器 `SYSTEM_ROM_CTRL_1_REG` 中 `SYSTEM_ROM_FORCE_PU` 的相应位可用以控制 ROM 不同 block 的上电。

更多有关 ROM 控制位的对应关系，请见下方表 30。

表 30: ROM 控制位

ROM	低地址 1	高地址 1	低地址 2	高地址 2	控制位
Block0	0x4000_0000	0x4000_FFFF	-	-	Bit0
Block1	0x4001_2000	0x4001_FFFF	0x3FFA_0000	0x3FFA_FFFF	Bit1

## SRAM 功耗控制

寄存器 `SYSTEM_SRAM_CTRL_0_REG`、`SYSTEM_SRAM_CTRL_1_REG` 和 `SYSTEM_SRAM_CTRL_2_REG` 用以控制 ESP32-S2 内部 SRAM 存储的功耗，具体来说：

- 寄存器 `SYSTEM_SRAM_CTRL_0_REG` 中 `SYSTEM_SRAM_FO` 的相应位可用以控制 SRAM 不同 block 的时钟门控。
- 寄存器 `SYSTEM_SRAM_CTRL_1_REG` 中 `SYSTEM_SRAM_FORCE_PD` 的相应位可用以控制 SRAM 不同 block 的掉电。
- 寄存器 `SYSTEM_SRAM_CTRL_2_REG` 中 `SYSTEM_SRAM_FORCE_PU` 的相应位可用以控制 SRAM 不同 block 的上电。

更多有关 SRAM 控制位的对应关系，请见下方表 31。

表 31: SRAM 控制位

SRAM	低地址 1	高地址 1	低地址 2	高地址 2	控制位
Block0	0x4002_0000	0x4002_1FFF	0x3FFB_0000	0x3FFB_1FFF	Bit0
Block1	0x4002_2000	0x4002_3FFF	0x3FFB_2000	0x3FFB_3FFF	Bit1
Block2	0x4002_4000	0x4002_5FFF	0x3FFB_4000	0x3FFB_5FFF	Bit2
Block3	0x4002_6000	0x4002_7FFF	0x3FFB_6000	0x3FFB_7FFF	Bit3
Block4	0x4002_8000	0x4002_BFFF	0x3FFB_8000	0x3FFB_BFFF	Bit4
Block5	0x4002_C000	0x4002_FFFF	0x3FFB_C000	0x3FFB_FFFF	Bit5
Block6	0x4003_0000	0x4003_3FFF	0x3FFC_0000	0x3FFC_3FFF	Bit6
Block7	0x4003_4000	0x4003_7FFF	0x3FFC_4000	0x3FFC_7FFF	Bit7
Block8	0x4003_8000	0x4003_BFFF	0x3FFC_8000	0x3FFC_BFFF	Bit8

Block9	0x4003_C000	0x4003_FFFF	0x3FFC_C000	0x3FFC_FFFF	Bit9
Block10	0x4004_0000	0x4004_3FFF	0x3FFD_0000	0x3FFD_3FFF	Bit10
Block11	0x4004_4000	0x4004_7FFF	0x3FFD_4000	0x3FFD_7FFF	Bit11
Block12	0x4004_8000	0x4004_BFFF	0x3FFD_8000	0x3FFD_BFFF	Bit12
Block13	0x4004_C000	0x4004_FFFF	0x3FFD_C000	0x3FFD_FFFF	Bit13
Block14	0x4005_0000	0x4005_3FFF	0x3FFE_0000	0x3FFE_3FFF	Bit14
Block15	0x4005_4000	0x4005_7FFF	0x3FFE_4000	0x3FFE_7FFF	Bit15
Block16	0x4005_8000	0x4005_BFFF	0x3FFE_8000	0x3FFE_BFFF	Bit16
Block17	0x4005_C000	0x4005_FFFF	0x3FFE_C000	0x3FFE_FFFF	Bit17
Block18	0x4006_0000	0x4006_3FFF	0x3FFF_0000	0x3FFF_3FFF	Bit18
Block19	0x4006_4000	0x4006_7FFF	0x3FFF_4000	0x3FFF_7FFF	Bit19
Block20	0x4006_8000	0x4006_BFFF	0x3FFF_8000	0x3FFF_BFFF	Bit20
Block21	0x4006_C000	0x4006_FFFF	0x3FFF_C000	0x3FFF_FFFF	Bit21

### 6.3.2 复位和时钟寄存器

以下系统寄存器用于复位和时钟的配置。更多信息，请见章节 2 [复位和时钟](#)。

- [SYSTEM\\_CPU\\_PER\\_CONF\\_REG](#)
- [SYSTEM\\_SYSCLK\\_CONF\\_REG](#)
- [SYSTEM\\_BT\\_LPCK\\_DIV\\_FRAC\\_REG](#)

### 6.3.3 中断矩阵寄存器

以下系统寄存器用于产生中断矩阵中的 CPU 中断信号。更多信息，请见章节 4 [中断矩阵](#)。

- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_0\\_REG](#)
- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_1\\_REG](#)
- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_2\\_REG](#)
- [SYSTEM\\_CPU\\_INTR\\_FROM\\_CPU\\_3\\_REG](#)

### 6.3.4 JTAG 软件使能寄存器

以下系统寄存器用于撤销 eFuse 对 JTAG 的临时关闭功能。

- [SYSTEM\\_JTAG\\_CTRL\\_0\\_REG](#)
- [SYSTEM\\_JTAG\\_CTRL\\_1\\_REG](#)
- [SYSTEM\\_JTAG\\_CTRL\\_2\\_REG](#)
- [SYSTEM\\_JTAG\\_CTRL\\_3\\_REG](#)
- [SYSTEM\\_JTAG\\_CTRL\\_4\\_REG](#)
- [SYSTEM\\_JTAG\\_CTRL\\_5\\_REG](#)

- [SYSTEM\\_JTAG\\_CTRL\\_6\\_REG](#)
- [SYSTEM\\_JTAG\\_CTRL\\_7\\_REG](#)

### 6.3.5 低功耗管理寄存器

以下系统寄存器用于低功耗管理。

- [SYSTEM\\_RTC\\_FASTMEM\\_CONFIG\\_REG](#)
- [SYSTEM\\_RTC\\_FASTMEM\\_CRC\\_REG](#)

### 6.3.6 外设时钟门控和复位寄存器

以下系统寄存器用于控制外设时钟门控和复位，详见下方表 32。

- [SYSTEM\\_CPU\\_PERI\\_CLK\\_EN\\_REG](#)
- [SYSTEM\\_CPU\\_PERI\\_RST\\_EN\\_REG](#)
- [SYSTEM\\_PERIP\\_CLK\\_EN0\\_REG](#)
- [SYSTEM\\_PERIP\\_RST\\_EN0\\_REG](#)
- [SYSTEM\\_PERIP\\_CLK\\_EN1\\_REG](#)
- [SYSTEM\\_PERIP\\_RST\\_EN1\\_REG](#)

表 32: 外设时钟门控与复位控制位

外设	时钟使能位 <sup>1</sup>	复位使能位 <sup>23</sup>
<b>CPU 外设</b>	<b><a href="#">SYSTEM_CPU_PERI_CLK_EN_REG</a></b>	<b><a href="#">SYSTEM_CPU_PERI_RST_EN_REG</a></b>
DEDICATED GPIO	SYSTEM_CLK_EN_DEDICATED_GPIO	SYSTEM_RST_EN_DEDICATED_GPIO
<b>外设</b>	<b><a href="#">SYSTEM_PERIP_CLK_EN0_REG</a></b>	<b><a href="#">SYSTEM_PERIP_RST_EN0_REG</a></b>
Timers	SYSTEM_TIMERS_CLK_EN	SYSTEM_TIMERS_RST
Timer Group0	SYSTEM_TIMERGROUP_CLK_EN	SYSTEM_TIMERGROUP_RST
Timer Group1	SYSTEM_TIMERGROUP1_CLK_EN	SYSTEM_TIMERGROUP1_RST
System Timer	SYSTEM_SYSTIMER_CLK_EN	SYSTEM_SYSTIMER_RST
UART0	SYSTEM_UART_CLK_EN	SYSTEM_UART_RST
UART1	SYSTEM_UART1_CLK_EN	SYSTEM_UART1_RST
UART MEM	SYSTEM_UART_MEM_CLK_EN <sup>4</sup>	SYSTEM_UART_MEM_RST
SPI0, SPI1	SYSTEM_SPI01_CLK_EN	SYSTEM_SPI01_RST
SPI2	SYSTEM_SPI2_CLK_EN	SYSTEM_SPI2_RST
SPI3	SYSTEM_SPI3_DMA_CLK_EN	SYSTEM_SPI3_RST
SPI4	SYSTEM_SPI4_CLK_EN	SYSTEM_SPI4_RST
SPI2 DMA	SYSTEM_SPI2_DMA_CLK_EN	SYSTEM_SPI2_DMA_RST
SPI3 DMA	SYSTEM_SPI3_DMA_CLK_EN	SYSTEM_SPI3_DMA_RST
I2C0	SYSTEM_I2C_EXT0_CLK_EN	SYSTEM_I2C_EXT0_RST
I2C1	SYSTEM_I2C_EXT1_CLK_EN	SYSTEM_I2C_EXT1_RST
I2S0	SYSTEM_I2S0_CLK_EN	SYSTEM_I2S0_RST
I2S1	SYSTEM_I2S1_CLK_EN	SYSTEM_SPI2_DMA_RST

TWAI 控制器	SYSTEM_CAN_CLK_EN	SYSTEM_CAN_RST
UHCI0	SYSTEM_UHCI0_CLK_EN	SYSTEM_UHCI0_RST
UHCI1	SYSTEM_UHCI1_CLK_EN	SYSTEM_UHCI1_RST
USB	SYSTEM_USB_CLK_EN	SYSTEM_USB_RST
RMT	SYSTEM_RMT_CLK_EN	SYSTEM_RMT_RST
PCNT	SYSTEM_PCNT_CLK_EN	SYSTEM_PCNT_RST
PWM0	SYSTEM_PWM0_CLK_EN	SYSTEM_PWM0_RST
PWM1	SYSTEM_PWM1_CLK_EN	SYSTEM_PWM1_RST
PWM2	SYSTEM_PWM2_CLK_EN	SYSTEM_PWM2_RST
PWM3	SYSTEM_PWM3_CLK_EN	SYSTEM_PWM3_RST
LED_PWM 控制器	SYSTEM_LEDC_CLK_EN	SYSTEM_LEDC_RST
eFuse	SYSTEM_EFUSE_CLK_EN	SYSTEM_EFUSE_RST
APB SARADC	SYSTEM_APB_SARADC_CLK_EN	SYSTEM_APB_SARADC_RST
ADC2 ARB	SYSTEM_ADC2_ARB_CLK_EN	SYSTEM_ADC2_ARB_RST
WDG	SYSTEM_WDG_CLK_EN	SYSTEM_WDG_RST
<b>加速器</b>	<b>SYSTEM_PERIP_CLK_EN1_REG</b>	<b>SYSTEM_PERIP_RST_EN1_REG</b>
DMA	SYSTEM_CRYPT0_DMA_CLK_EN	SYSTEM_CRYPT0_DMA_RST <sup>5</sup>
HMAC	SYSTEM_CRYPT0_HMAC_CLK_EN	SYSTEM_CRYPT0_HMAC_RST <sup>6</sup>
数字签名	SYSTEM_CRYPT0_DS_CLK_EN	SYSTEM_CRYPT0_DS_RST <sup>7</sup>
RSA 加速器	SYSTEM_CRYPT0_RSA_CLK_EN	SYSTEM_CRYPT0_RSA_RST
SHA 加速器	SYSTEM_CRYPT0_SHA_CLK_EN	SYSTEM_CRYPT0_SHA_RST
AES 加速器	SYSTEM_CRYPT0_AES_CLK_EN	SYSTEM_CRYPT0_AES_RST

**说明：**

1. 时钟控制寄存器置 1 打开对应时钟，置 0 关闭对应时钟。
2. 复位寄存器置 1 使能复位状态，对应外设进行复位，置 0 关闭复位状态，对应外设正常工作。
3. 复位寄存器无法通过硬件清除。
4. UART 存储器为所有 UART 外设所共用，因此只要有一个 UART 在工作，UART 存储器就不能处于门控状态。
5. 该 Crypto DMA 为 AES 加速器和 SHA 加速器所共用。
6. 该位复位后，SHA 加速器也会同时被复位。
7. 该位复位后，AES 加速器、SHA 加速器和 RSA 加速器也会同时被复位。

## 6.4 基地址

用户可以通过两个不同的寄存器基地址访问系统寄存器，如表 33 所示。更多信息，请访问[章节 1 系统和存储器](#)。

表 33: 系统寄存器基地址

访问总线	基地址
PeriBUS1	0x3F4C0000

## 6.5 寄存器列表

请注意，这里的地址是相对于系统寄存器基地址的地址偏移量（相对地址）。请参阅[章节 6.4](#) 获取有关系统寄存器基地址的信息。

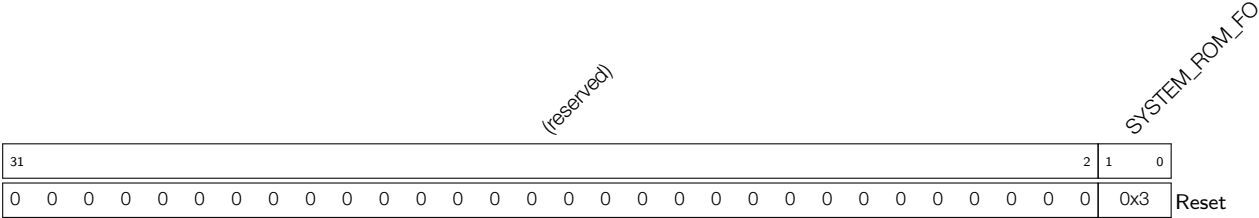
名称	描述	地址	访 问 权限
<b>系统和存储器寄存器</b>			
<a href="#">SYSTEM_ROM_CTRL_0_REG</a>	系统 ROM 配置寄存器 0	0x0000	读写
<a href="#">SYSTEM_ROM_CTRL_1_REG</a>	系统 ROM 配置寄存器 1	0x0004	读写
<a href="#">SYSTEM_SRAM_CTRL_0_REG</a>	系统 SRAM 配置寄存器 0	0x0008	读写
<a href="#">SYSTEM_SRAM_CTRL_1_REG</a>	系统 SRAM 配置寄存器 1	0x000C	读写
<a href="#">SYSTEM_SRAM_CTRL_2_REG</a>	系统 SRAM 配置寄存器 2	0x0088	读写
<a href="#">SYSTEM_RSA_PD_CTRL_REG</a>	RSA 存储掉电寄存器	0x0068	读写
<a href="#">SYSTEM_MEM_PD_MASK_REG</a>	存储器掉电屏蔽寄存器（low-sleep 状态下）	0x003C	读写
<a href="#">SYSTEM_CACHE_CONTROL_REG</a>	Cache 控制寄存器	0x0070	读写
<a href="#">SYSTEM_BUSTOEXTMEM_ENA_REG</a>	EDMA 使能寄存器	0x006C	读写
<a href="#">SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG</a>	外部设备加解密控制寄存器	0x0074	读写
<b>复位和时钟寄存器</b>			
<a href="#">SYSTEM_CPU_PER_CONF_REG</a>	CPU 外设时钟配置寄存器	0x0018	读写
<a href="#">SYSTEM_SYSCLK_CONF_REG</a>	系统时钟配置寄存器	0x008C	不定
<a href="#">SYSTEM_BT_LPCK_DIV_FRAC_REG</a>	低功耗时钟分频分数配置寄存器	0x0054	读写
<b>中断矩阵寄存器</b>			
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_0_REG</a>	CPU 中断控制寄存器 0	0x0058	读写
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_1_REG</a>	CPU 中断控制寄存器 1	0x005C	读写
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_2_REG</a>	CPU 中断控制寄存器 2	0x0060	读写
<a href="#">SYSTEM_CPU_INTR_FROM_CPU_3_REG</a>	CPU 中断控制寄存器 3	0x0064	读写
<b>JTAG 软件使能寄存器</b>			
<a href="#">SYSTEM_JTAG_CTRL_0_REG</a>	JTAG 配置寄存器 0	0x001C	只写
<a href="#">SYSTEM_JTAG_CTRL_1_REG</a>	JTAG 配置寄存器 1	0x0020	只写
<a href="#">SYSTEM_JTAG_CTRL_2_REG</a>	JTAG 配置寄存器 2	0x0024	只写
<a href="#">SYSTEM_JTAG_CTRL_3_REG</a>	JTAG 配置寄存器 3	0x0028	只写
<a href="#">SYSTEM_JTAG_CTRL_4_REG</a>	JTAG 配置寄存器 4	0x002C	只写
<a href="#">SYSTEM_JTAG_CTRL_5_REG</a>	JTAG 配置寄存器 5	0x0030	只写

名称	描述	地址	访 问 权限
SYSTEM_JTAG_CTRL_6_REG	JTAG 配置寄存器 6	0x0034	只写
SYSTEM_JTAG_CTRL_7_REG	JTAG 配置寄存器 7	0x0038	只写
低功耗管理寄存器			
SYSTEM_RTC_FASTMEM_CONFIG_REG	RTC 快速内存配置寄存器	0x0078	不定
SYSTEM_RTC_FASTMEM_CRC_REG	RTC 快速内存 CRC 校验寄存器	0x007C	只读
外设时钟门控和复位寄存器			
SYSTEM_CPU_PERI_CLK_EN_REG	CPU 外设时钟使能寄存器	0x0010	读写
SYSTEM_CPU_PERI_RST_EN_REG	CPU 外设复位寄存器	0x0014	读写
SYSTEM_PERIP_CLK_EN0_REG	系统外设时钟（硬件加速器）使能寄存器 0	0x0040	读写
SYSTEM_PERIP_CLK_EN1_REG	系统外设时钟（硬件加速器）使能寄存器 1	0x0044	读写
SYSTEM_PERIP_RST_EN0_REG	系统外设（硬件加速器）复位寄存器 0	0x0048	读写
SYSTEM_PERIP_RST_EN1_REG	系统外设（硬件加速器）复位寄存器 1	0x004C	读写
版本寄存器			
SYSTEM_REG_DATE_REG	版本控制寄存器	0x0FFC	读写

6.6 寄存器

请注意，这里的地址是相对于系统寄存器基地址的地址偏移量（相对地址）。请参阅章节 6.4 获取有关系统寄存器基地址的信息。

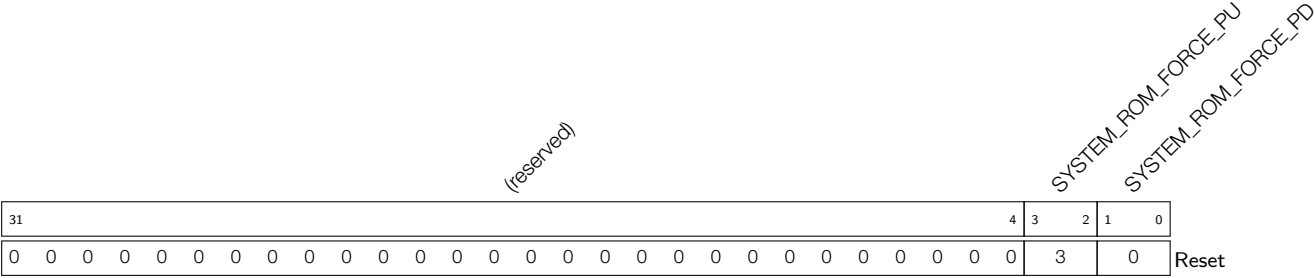
Register 6.1: SYSTEM\_ROM\_CTRL\_0\_REG (0x0000)



SYSTEM\_ROM\_FO 强制打开内部 ROM 的时钟门控。详见表 30。（读写）



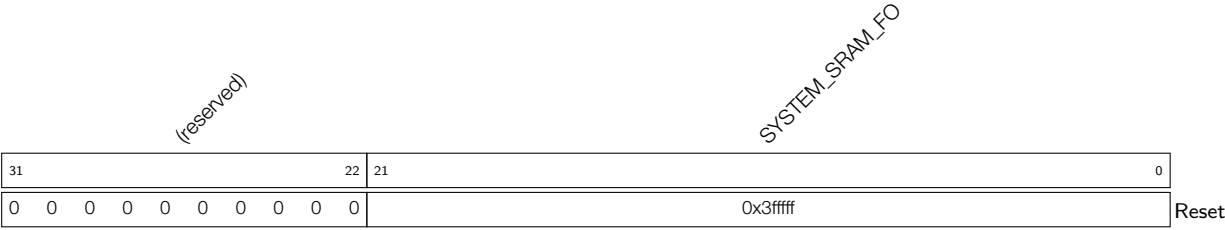
Register 6.2: SYSTEM\_ROM\_CTRL\_1\_REG (0x0004)



**SYSTEM\_ROM\_FORCE\_PD** 控制内部 ROM 的掉电。详见表 30。(读写)

**SYSTEM\_ROM\_FORCE\_PU** 控制内部 ROM 的上电。详见表 30。(读写)

Register 6.3: SYSTEM\_SRAM\_CTRL\_0\_REG (0x0008)



**SYSTEM\_SRAM\_FO** 强制打开内部 SRAM 的时钟门控。详见表 31。(读写)

Register 6.4: SYSTEM\_SRAM\_CTRL\_1\_REG (0x000C)



**SYSTEM\_SRAM\_FORCE\_PD** 控制内部 SRAM 的掉电。详见表 31。(读写)

Register 6.5: SYSTEM\_CPU\_PERI\_CLK\_EN\_REG (0x0010)

(reserved)																								SYSTEM_CLK_EN_DEDICATED_GPIO				(reserved)			
31																								8	7	6	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	0	0	Reset			

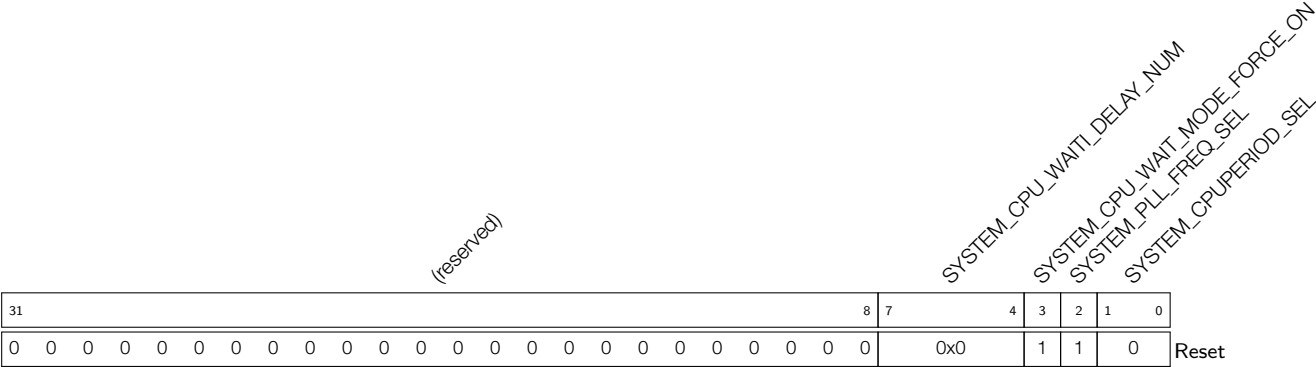
SYSTEM\_CLK\_EN\_DEDICATED\_GPIO 置 1 打开 DEDICATED GPIO 时钟。（读写）

Register 6.6: SYSTEM\_CPU\_PERI\_RST\_EN\_REG (0x0014)

(reserved)																								SYSTEM_RST_EN_DEDICATED_GPIO				(reserved)			
31																					8	7	6	5					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	0	0	0	0	Reset		

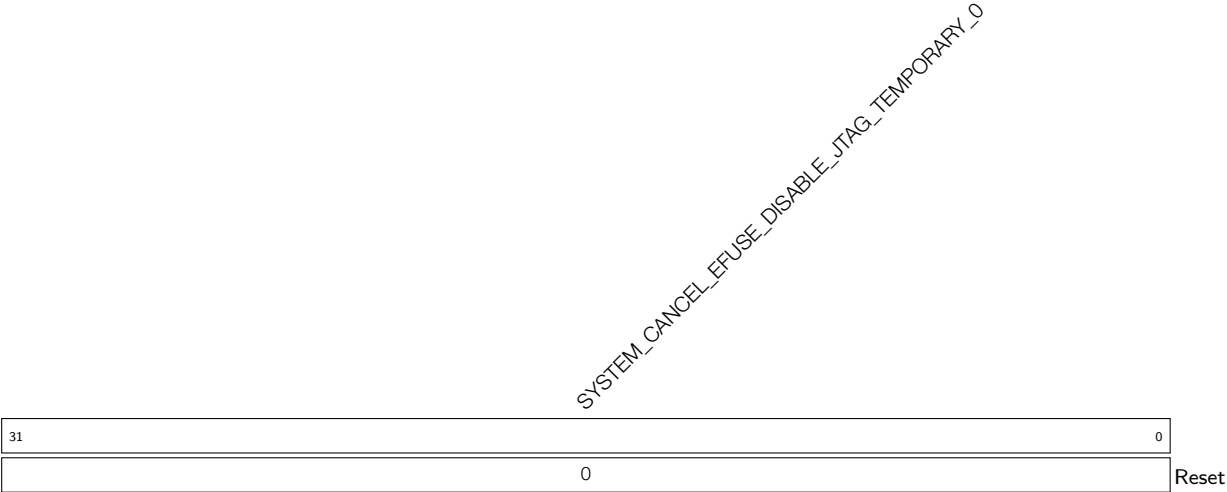
SYSTEM\_RST\_EN\_DEDICATED\_GPIO 置 1 复位 DEDICATED GPIO。（R/W）

Register 6.7: SYSTEM\_CPU\_PER\_CONF\_REG (0x0018)



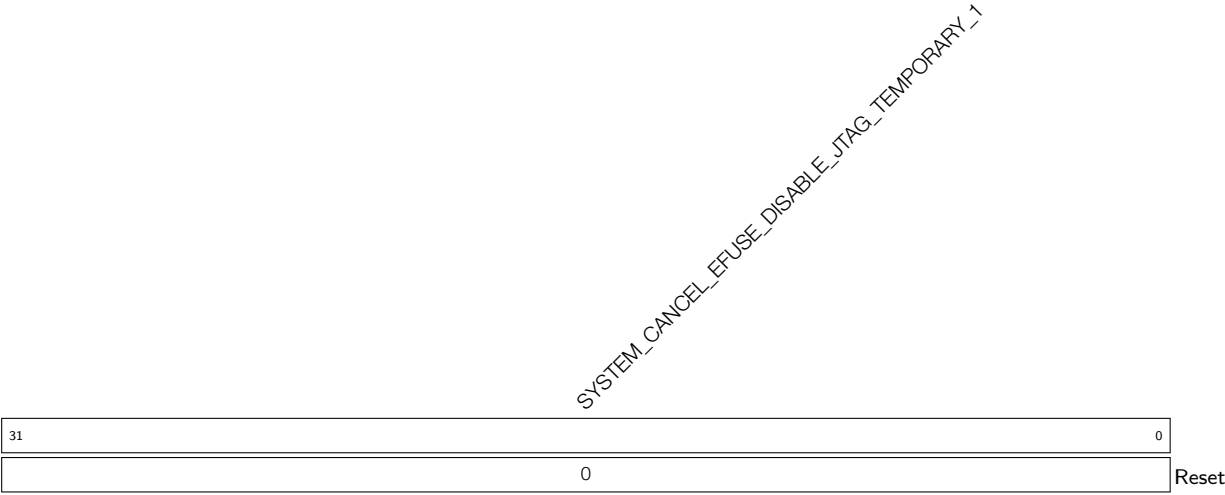
- SYSTEM\_CPUPERIOD\_SEL** 选择 CPU 时钟周期或时钟频率。详见章节 2 复位和时钟中表 8。(读写)
- SYSTEM\_PLL\_FREQ\_SEL** 选择基于 CPU 时钟周期的 PPL 时钟频率。详见章节 2 复位和时钟。(读写)
- SYSTEM\_CPU\_WAIT\_MODE\_FORCE\_ON** 置 1 强制打开 CPU 等待模式。在 CPU 等待模式下,CPU 时钟门控一直处于关闭状态，直到中断产生。用户也可通过 WAITI 指令强制打开 CPU 等待模式。(读写)
- SYSTEM\_CPU\_WAITI\_DELAY\_NUM** 设置 CPU 在收到 WAITI 指令后进入 CPU 等待模式前的等待周期。(读写)

Register 6.8: SYSTEM\_JTAG\_CTRL\_0\_REG (0x001C)



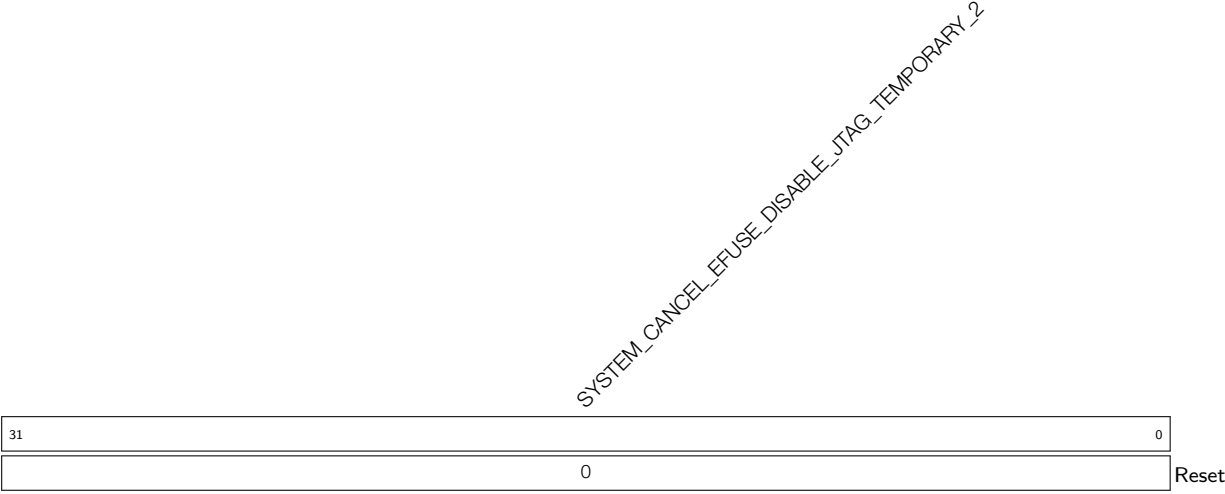
- SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_0** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 0 到 31 位。（只写）

Register 6.9: SYSTEM\_JTAG\_CTRL\_1\_REG (0x0020)



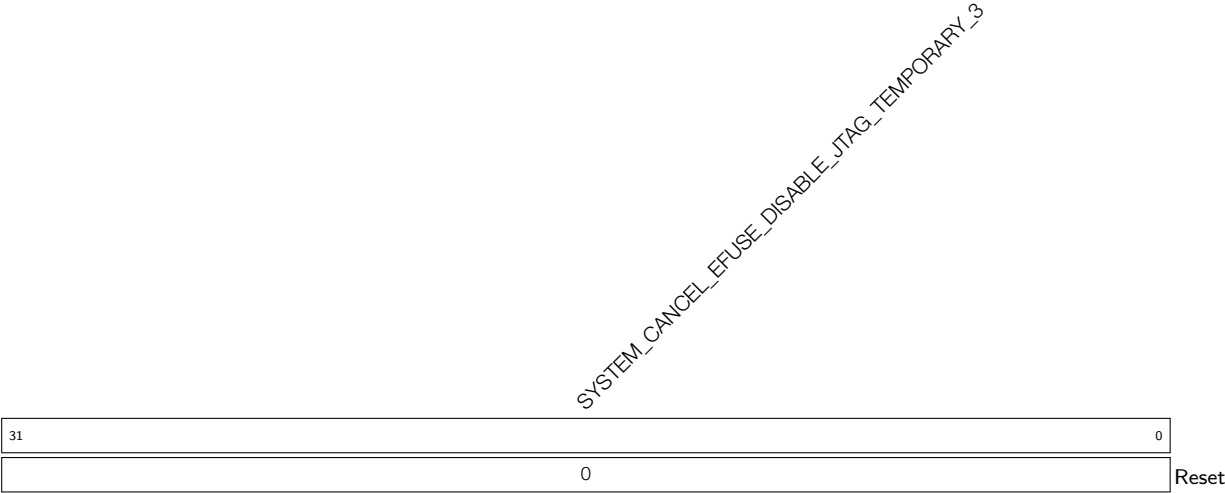
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_1** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 32 到 63 位。（只写）

Register 6.10: SYSTEM\_JTAG\_CTRL\_2\_REG (0x0024)



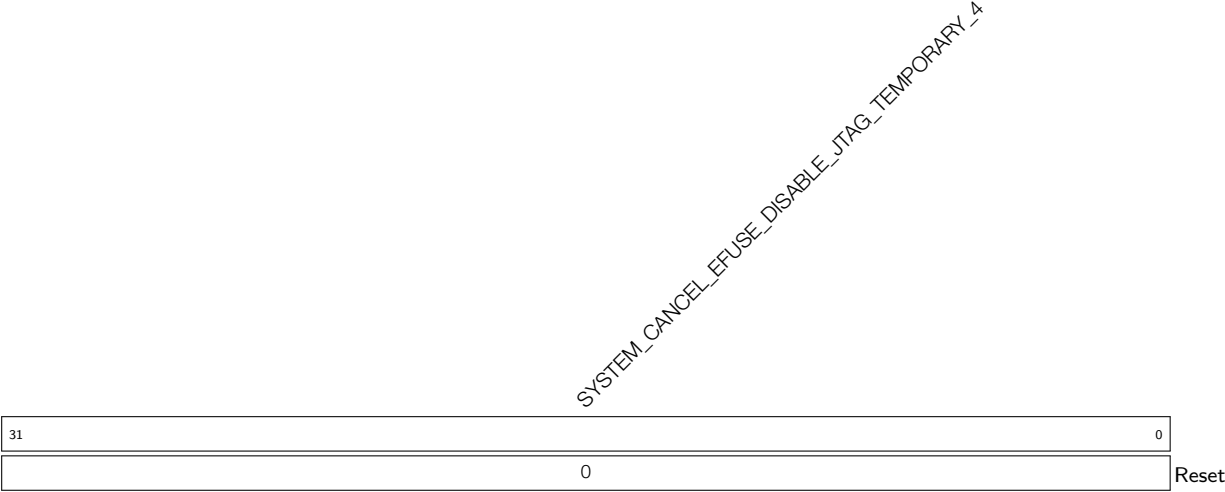
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_2** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 64 到 95 位。（只写）

Register 6.11: SYSTEM\_JTAG\_CTRL\_3\_REG (0x0028)



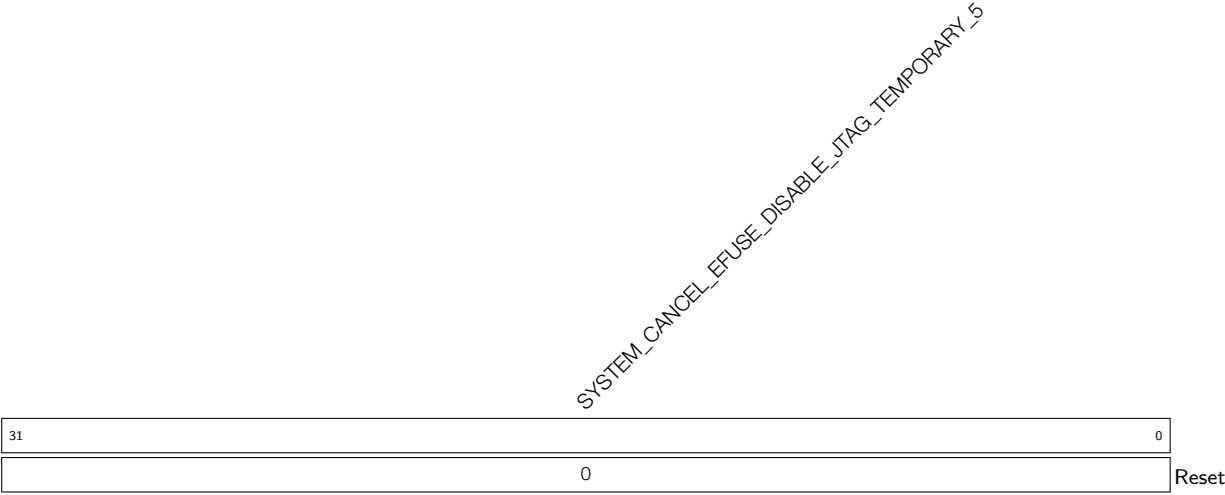
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_3** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 96 到 127 位。（只写）

Register 6.12: SYSTEM\_JTAG\_CTRL\_4\_REG (0x002C)



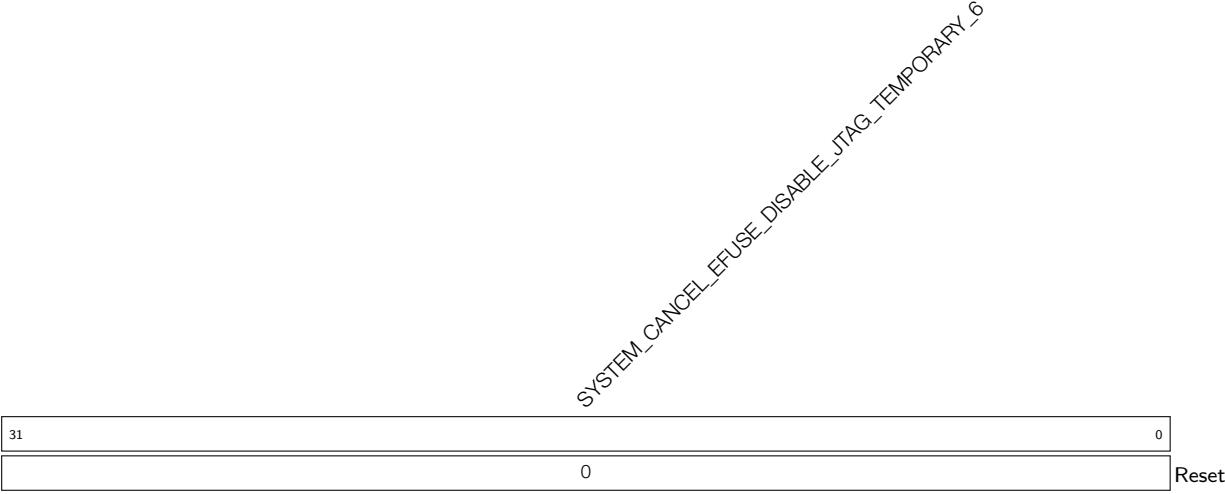
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_4** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 128 到 159 位。（只写）

Register 6.13: SYSTEM\_JTAG\_CTRL\_5\_REG (0x0030)



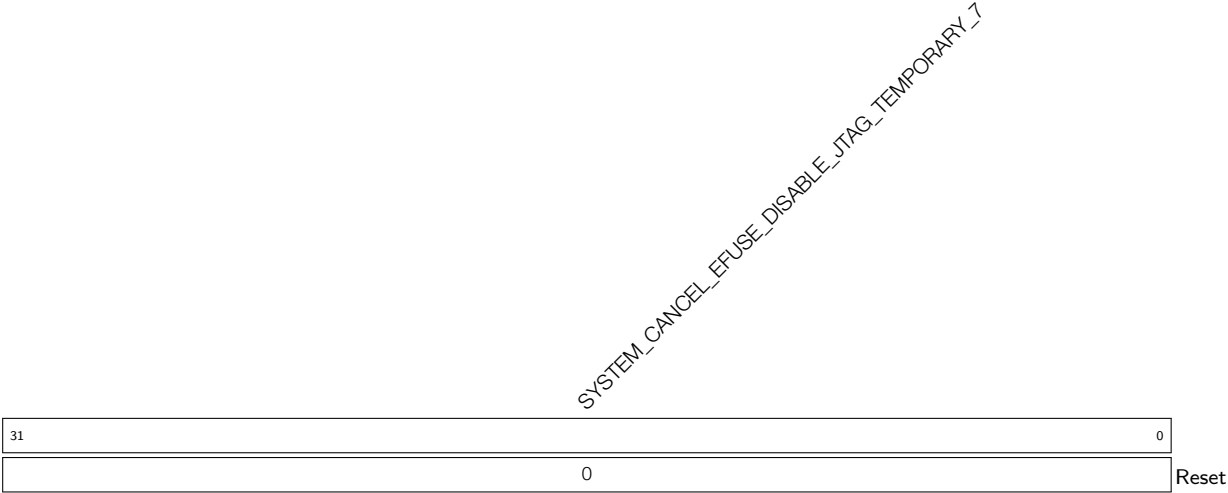
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_5** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 160 到 191 位。（只写）

Register 6.14: SYSTEM\_JTAG\_CTRL\_6\_REG (0x0034)



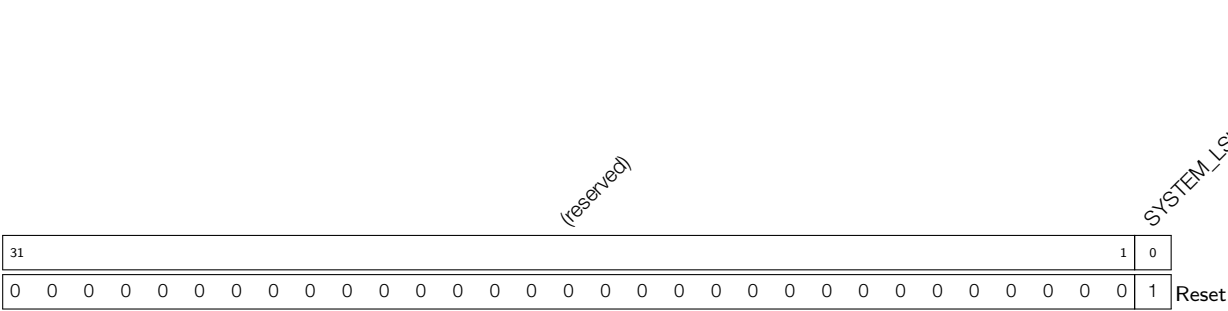
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_6** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 192 到 223 位。（只写）

Register 6.15: SYSTEM\_JTAG\_CTRL\_7\_REG (0x0038)



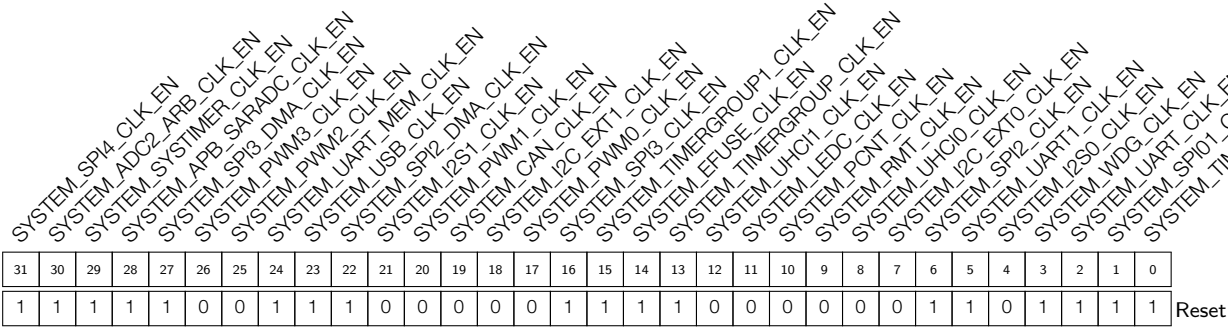
**SYSTEM\_CANCEL\_EFUSE\_DISABLE\_JTAG\_TEMPORARY\_7** 本组寄存器（共 256 位）用于撤销 eFuse 对 JTAG 的临时关闭，本寄存器为第 224 到 255 位。（只写）

Register 6.16: SYSTEM\_MEM\_PD\_MASK\_REG (0x003C)



**SYSTEM\_LSLP\_MEM\_PD\_MASK** 置 1 允许存储器在 light-sleep 模式下仍正常工作。（读写）

Register 6.17: SYSTEM\_PERIP\_CLK\_EN0\_REG (0x0040)



**SYSTEM\_CPU\_PERI\_CLK\_EN0\_REG** 使能不同的外设时钟，详见表 32。

Register 6.18: SYSTEM\_PERIP\_CLK\_EN1\_REG (0x0044)

(reserved)																								SYSTEM_CRYPTO_DMA_CLK_EN SYSTEM_CRYPTO_HMAC_CLK_EN SYSTEM_CRYPTO_DS_CLK_EN SYSTEM_CRYPTO_RSA_CLK_EN SYSTEM_CRYPTO_SHA_CLK_EN SYSTEM_CRYPTO_AES_CLK_EN (reserved)							
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	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

**SYSTEM\_PERIP\_CLK\_EN1\_REG** 使能不同的加速器时钟，详见表 32。

Register 6.19: SYSTEM\_PERIP\_RST\_EN0\_REG (0x0048)

SYSTEM_SPI4_RST SYSTEM_ADC2_ARB_RST SYSTEM_SYSTIMER_RST SYSTEM_APB_SARADC_RST SYSTEM_SPI3_DMA_RST SYSTEM_PWM3_RST SYSTEM_PWM2_RST SYSTEM_UART_MEM_RST SYSTEM_USB_RST SYSTEM_SPI2_DMA_RST SYSTEM_I2S1_RST SYSTEM_PWM1_RST SYSTEM_CAN_RST SYSTEM_I2C_EXT1_RST SYSTEM_PWM0_RST SYSTEM_SPI3_RST SYSTEM_TIMERGROUP1_RST SYSTEM_EFUSE_RST SYSTEM_TIMERGROUP_RST SYSTEM_UHC11_RST SYSTEM_LEDC_RST SYSTEM_PONT_RST SYSTEM_RMT_RST SYSTEM_UHC00_RST SYSTEM_I2C_EXT0_RST SYSTEM_SPI2_RST SYSTEM_UART1_RST SYSTEM_I2S0_RST SYSTEM_WDG_RST SYSTEM_UART_RST SYSTEM_SPI01_RST SYSTEM_TIMERS_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	
0	0	0	0	0	0	0	0	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

**SYSTEM\_PERIP\_RST\_EN0\_REG** 复位不同外设，详见表 32。

Register 6.20: SYSTEM\_PERIP\_RST\_EN1\_REG (0x004C)

(reserved)																												SYSTEM_CRYPTO_DMA_RST SYSTEM_CRYPTO_HMAC_RST SYSTEM_CRYPTO_DS_RST SYSTEM_CRYPTO_RSA_RST SYSTEM_CRYPTO_SHA_RST SYSTEM_CRYPTO_AES_RST (reserved)							
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	0	0	0	0	0	0	0	1	1	1	1	1	1	0	Reset			

**SYSTEM\_PERIP\_RST\_EN1\_REG** 复位不同加速器，详见表 32。



Register 6.21: SYSTEM\_BT\_LPCK\_DIV\_FRAC\_REG (0x0054)

(reserved)				SYSTEM_LPCLK_RTC_EN				SYSTEM_LPCLK_SEL_XTAL32K				SYSTEM_LPCLK_SEL_XTAL				SYSTEM_LPCLK_SEL_8M				SYSTEM_LPCLK_SEL_RTC_SLOW				(reserved)								(reserved)			
31	29	28	27	26	25	24	23					12	11											0											
0	0	0	0	0	0	1	0	1				1										Reset													

SYSTEM\_LPCLK\_SEL\_RTC\_SLOW 置 1 选择 RTC 慢速时钟为低功耗时钟。(读写)

SYSTEM\_LPCLK\_SEL\_8M 置 1 选择 8m 时钟为低功耗时钟。(读写)

SYSTEM\_LPCLK\_SEL\_XTAL 置 1 选择 xtal 时钟为低功耗时钟。(读写)

SYSTEM\_LPCLK\_SEL\_XTAL32K 置 1 选择 xtal32k 时钟为低功耗时钟。(读写)

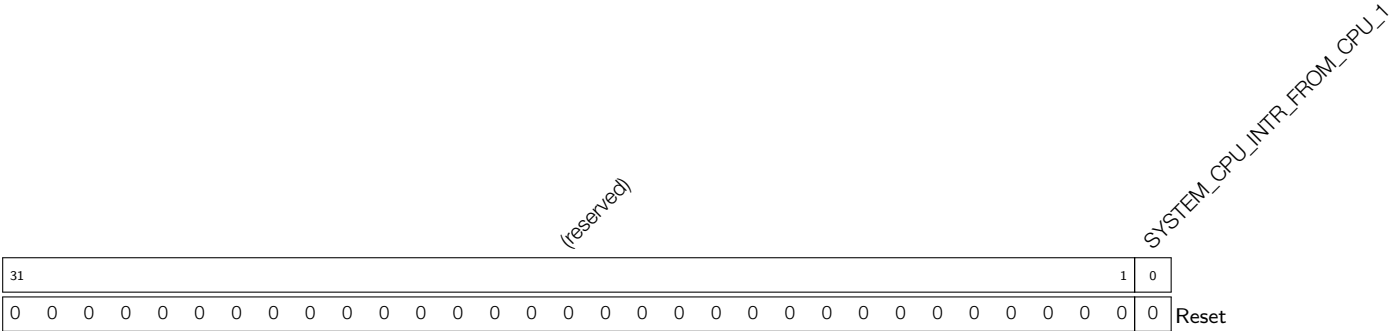
SYSTEM\_LPCLK\_RTC\_EN 置 1 使能 RTC 低功耗时钟。(读写)

Register 6.22: SYSTEM\_CPU\_INTR\_FROM\_CPU\_0\_REG (0x0058)

(reserved)																															SYSTEM_CPU_INTR_FROM_CPU_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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

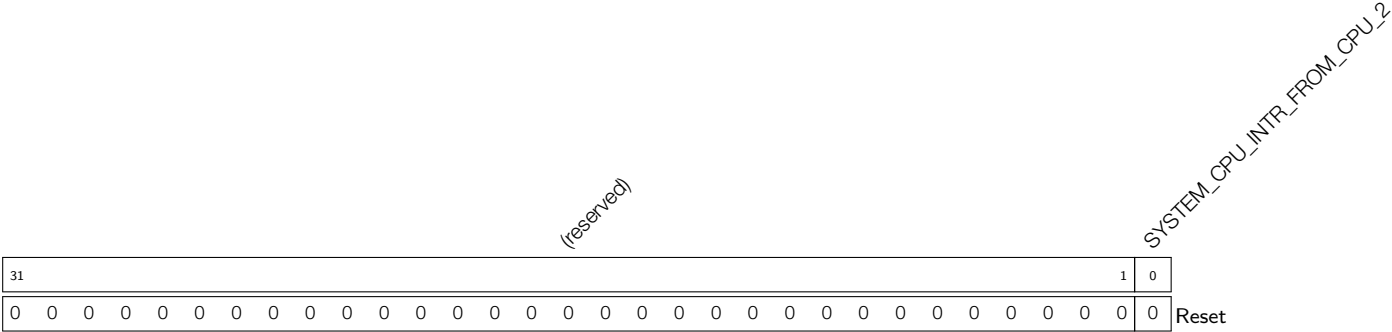
SYSTEM\_CPU\_INTR\_FROM\_CPU\_0 置 1 生成 CPU 中断 0。该位需在 ISR 过程中由软件清 0。(读写)

Register 6.23: SYSTEM\_CPU\_INTR\_FROM\_CPU\_1\_REG (0x005C)



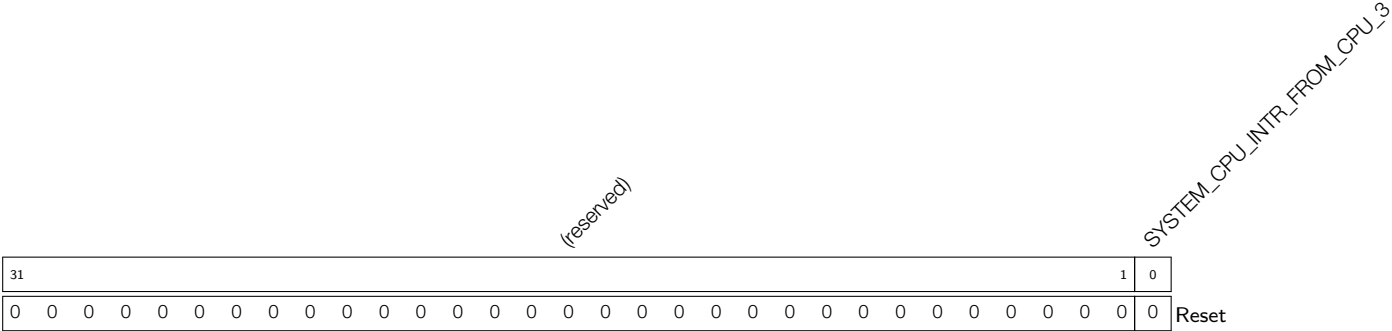
SYSTEM\_CPU\_INTR\_FROM\_CPU\_1 置 1 生成 CPU 中断 1。该位需在 ISR 过程中由软件清 0。（读写）

Register 6.24: SYSTEM\_CPU\_INTR\_FROM\_CPU\_2\_REG (0x0060)



SYSTEM\_CPU\_INTR\_FROM\_CPU\_2 置 1 生成 CPU 中断 2。该位需在 ISR 过程中由软件清 0。（读写）

Register 6.25: SYSTEM\_CPU\_INTR\_FROM\_CPU\_3\_REG (0x0064)



SYSTEM\_CPU\_INTR\_FROM\_CPU\_3 置 1 生成 CPU 中断 3。该位需在 ISR 过程中由软件清 0。（读写）

Register 6.26: SYSTEM\_RSA\_PD\_CTRL\_REG (0x0068)

(reserved)																																SYSTEM_RSA_MEM_FORCE_PD SYSTEM_RSA_MEM_FORCE_PU SYSTEM_RSA_MEM_PD				
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	1	Reset

- SYSTEM\_RSA\_MEM\_PD** 置 1 关闭 RSA 内存。该位优先级最低。当数字签名占用 RSA 加速器时，该位无效。(读写)
- SYSTEM\_RSA\_MEM\_FORCE\_PU** 置 1 关闭 RSA 内存。该位优先级第二高。(读写)
- SYSTEM\_RSA\_MEM\_FORCE\_PD** 置 1 关闭 RSA 内存。该位优先级最高。(读写)

Register 6.27: SYSTEM\_BUSTOEXTMEM\_ENA\_REG (0x006C)

(reserved)																															SYSTEM_BUSTOEXTMEM_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	1	Reset

- SYSTEM\_BUSTOEXTMEM\_ENA** 置 1 使能 EDMA 总线。(读写)

Register 6.28: SYSTEM\_CACHE\_CONTROL\_REG (0x0070)

(reserved)																																SYSTEM_PRO_CACHE_RESET SYSTEM_PRO_DCACHE_CLK_ON SYSTEM_PRO_ICACHE_CLK_ON		
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	1	Reset

- SYSTEM\_PRO\_ICACHE\_CLK\_ON** 置 1 使能 i-cache 时钟。(读写)
- SYSTEM\_PRO\_DCACHE\_CLK\_ON** 置 1 使能 d-cache 时钟。(读写)
- SYSTEM\_PRO\_CACHE\_RESET** 置 1 复位 cache。(读写)

Register 6.29: SYSTEM\_EXTERNAL\_DEVICE\_ENCRYPT\_DECRYPT\_CONTROL\_REG (0x0074)

(reserved)																																SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

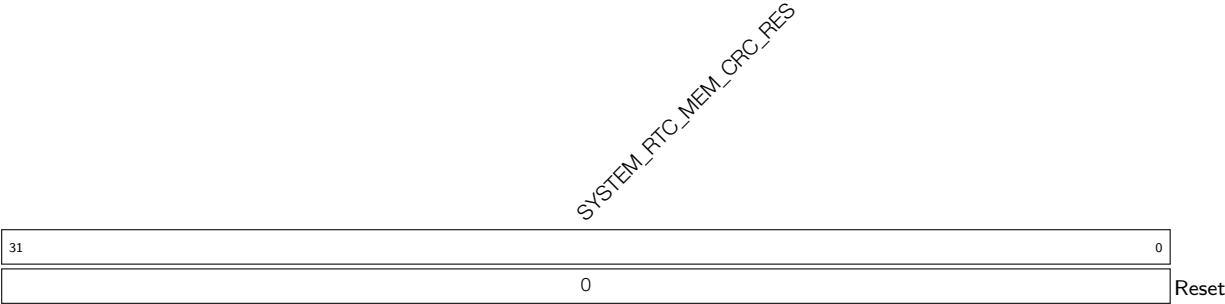
- SYSTEM\_ENABLE\_SPI\_MANUAL\_ENCRYPT** 置 1 在 SPI Boot 模式下使能手动加密 (Manual Encryption)。(读写)
- SYSTEM\_ENABLE\_DOWNLOAD\_DB\_ENCRYPT** 置 1 在 Download Boot 模式下使能自动加密 (Auto Encryption)。(读写)
- SYSTEM\_ENABLE\_DOWNLOAD\_G0CB\_DECRYPT** 置 1 在 Download Boot 模式下使能自动解密 (Auto Decryption)。(读写)
- SYSTEM\_ENABLE\_DOWNLOAD\_MANUAL\_ENCRYPT** 置 1 在 Download Boot 模式下使能手动加密 (Manual Encryption)。(读写)

Register 6.30: SYSTEM\_RTC\_FASTMEM\_CONFIG\_REG (0x0078)

SYSTEM_RTC_MEM_CRC_FINISH										SYSTEM_RTC_MEM_CRC_LEN										SYSTEM_RTC_MEM_CRC_ADDR										SYSTEM_RTC_MEM_CRC_START										(reserved)									
31	30																				9	8	7													0													
0			0x7ff																					0	0	0	0	0	0	0	0	0	0	0	0	0	Reset												

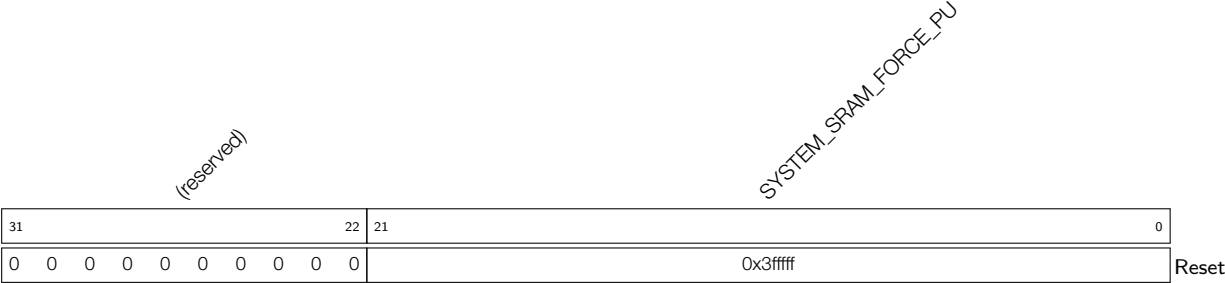
- SYSTEM\_RTC\_MEM\_CRC\_START** 置 1 启动 RTC 存储的 CRC 校验。(读写)
- SYSTEM\_RTC\_MEM\_CRC\_ADDR** 设置 CRC 校验的 RTC 存储地址。(读写)
- SYSTEM\_RTC\_MEM\_CRC\_LEN** 设置用于 CRC 校验的 RTC 存储长度 (基于起始地址)。(读写)
- SYSTEM\_RTC\_MEM\_CRC\_FINISH** 储存 RTC 存储 CRC 校验状态。高电平表示校验完成，低电平表示校验未完成。(只读)

Register 6.31: SYSTEM\_RTC\_FASTMEM\_CRC\_REG (0x007C)



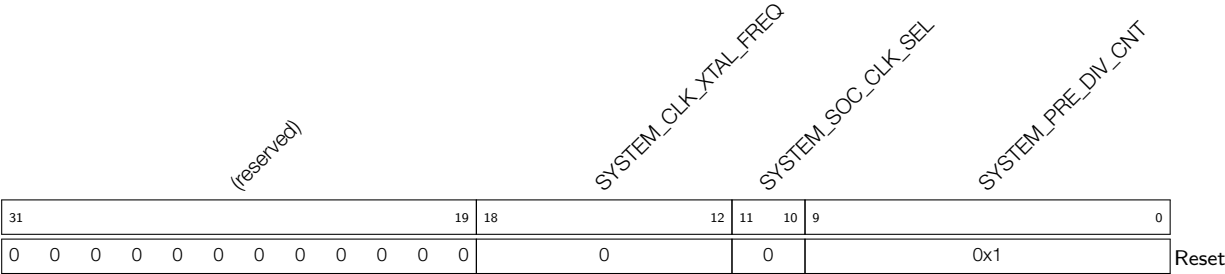
**SYSTEM\_RTC\_MEM\_CRC\_RES** 储存 RTC 存储的 CRC 校验结果。(只读)

Register 6.32: SYSTEM\_SRAM\_CTRL\_2\_REG (0x0088)



**SYSTEM\_SRAM\_FORCE\_PU** 控制内部 SRAM 的上电。详见表 31。(读写)

Register 6.33: SYSTEM\_SYSCLK\_CONF\_REG (0x008C)

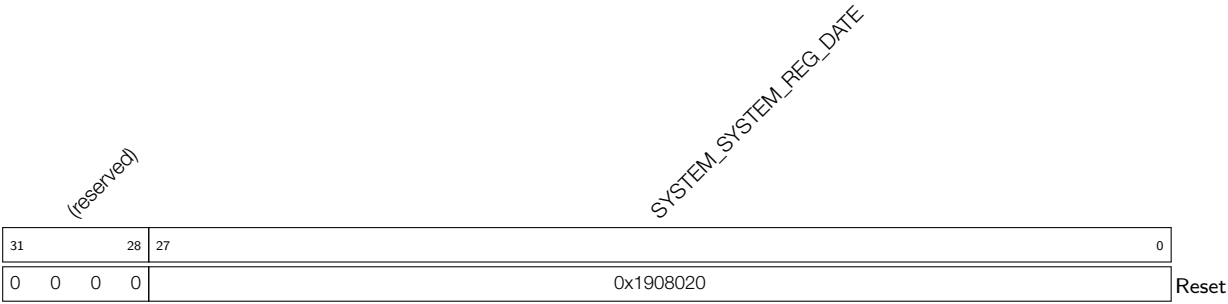


**SYSTEM\_PRE\_DIV\_CNT** 设置预分频器计数器。具体配置，请见章节 2 复位和时钟中表 10。(读写)

**SYSTEM\_SOC\_CLK\_SEL** 选择 SoC 时钟。具体配置，请见章节 2 复位和时钟中表 8。(读写)

**SYSTEM\_CLK\_XTAL\_FREQ** 读取晶振频率（单位：MHz）。(只读)

Register 6.34: SYSTEM\_REG\_DATE\_REG (0x0FFC)



**SYSTEM\_DATE** 版本控制寄存器。（读写）

## 7. DMA 控制器

### 7.1 概述

直接存储访问 (Direct Memory Access, DMA) 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。可以在无需任何 CPU 操作的情况下通过 DMA 快速移动数据，从而提高了 CPU 的效率。

ESP32-S2 包括三种 DMA：Internal DMA，EDMA 和 Copy DMA。Internal DMA 只能访问片内 RAM，用于内存与外设之间的数据传输；EDMA 既能访问片内 RAM 也能访问片外 RAM，用于存储器与外设之间的数据传输；Copy DMA 只能访问片内 RAM，用于内存与内存之间的高速数据传输。

ESP32-S2 中有 8 个外设具有 DMA 功能。如表 7-1 所示，UART0 与 UART1 共用一个 Internal DMA；SPI3 与 ADC Controller 共用一个 Internal DMA；AES 与 SHA 共用一个 EDMA；SPI2 与 I2S0 各有一个单独的 EDMA。除此之外，CPU Peripheral 模块下还包括一个 Copy DMA。

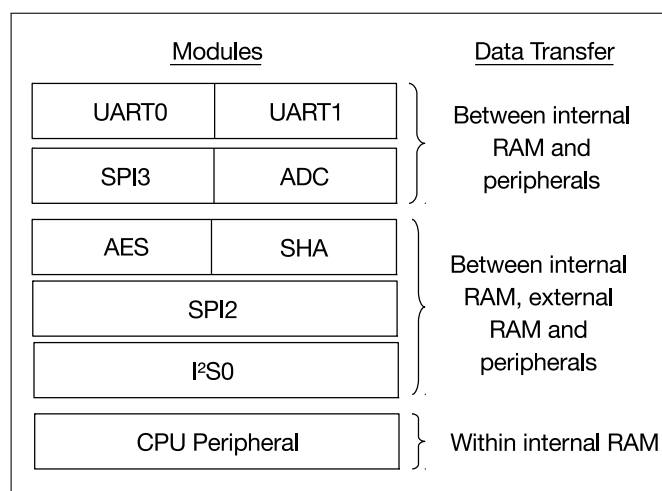


图 7-1. 具有 DMA 功能的模块和支持的数据传输类型

### 7.2 特性

DMA 控制器具有以下几个特点：

- AHB 总线架构
- 支持半双工和全双工收发数据
- 数据传输以字节为单位，传输数据量可软件编程
- 访问内部 RAM 时，支持 INCR burst 传输
- DMA 能够访问的内部 RAM 最大地址空间为 320 KB
- DMA 能够访问的最大外部地址空间为 10.5 MB
- 通过 DMA 实现高速数据传输

### 7.3 功能描述

ESP32-S2 中所有需要进行高速数据传输的模块都具有 DMA 功能。DMA 控制器与 CPU 的数据总线使用相同的地址空间访问内部和外部 RAM。根据各自模块的需求，各个模块的 DMA 控制器功能有所差别，但是 DMA 引擎 (DMA\_ENGINE) 的结构相同。

#### 7.3.1 DMA 引擎的架构

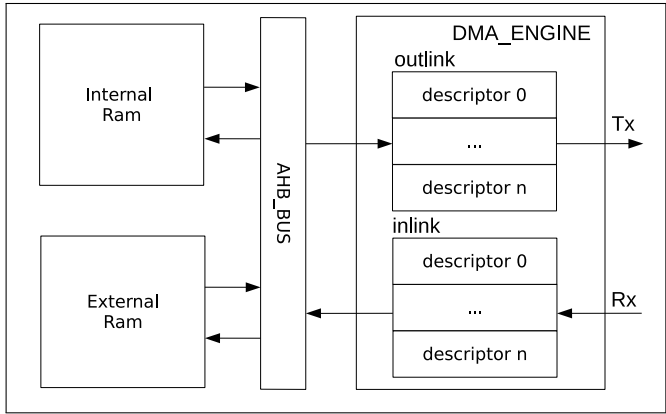


图 7-2. DMA 引擎的架构

DMA 引擎通过 AHB\_BUS 将数据存入外部或内部 RAM 或者将数据从外部或内部 RAM 取出。图 7-2 为 DMA 引擎基本架构图。RAM 的具体使用范围详见章节 1 系统和存储器。软件可以通过挂载链表的方式来使用 DMA 引擎。DMA\_ENGINE 根据 outlink 中的内容将相应 RAM 中的数据发送出去，也可根据 inlink 中的内容将接收的数据存入指定 RAM 地址空间。

#### 7.3.2 链表

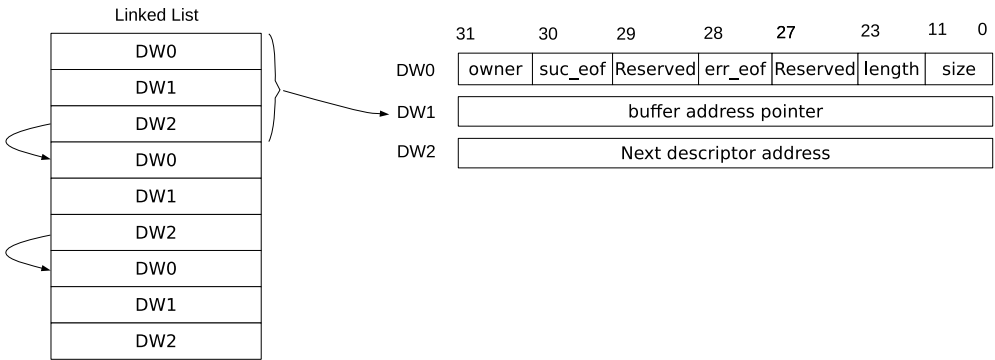


图 7-3. 链表结构图

图 7-3 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。链表应存放在内部 RAM 中，供 DMA 引擎使用。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。
  - 1'b0：允许的操作者为 CPU；
  - 1'b1：允许的操作者为 DMA 控制器。



在 DMA 使用完该描述符对应的 buffer 后，硬件会将该 bit 清零。也可通过置位 `PERI_IN_LOOP_TEST` 位关闭硬件自动清零的功能。软件在挂载链表时需要将该 bit 置 1。

**注意：**`PERI`指包含 DMA 的外设模块名，如 I<sup>2</sup>S, SPI, UHCI 等。

- `suc_eof` (DW0) [30]: 表示结束标志。  
1'b0: 当前描述符不是链表中的最后一个描述符;  
1'b1: 当前描述符为数据包的最后一个描述符。  
对于接收描述符，硬件会在收完 1 个包后将该 bit 置 1。对于发送描述符，需要软件在包的最后一个描述符中的该 bit 置 1。
- Reserved (DW0) [29]: 保留。
- `err_eof` (DW0) [28]: 表示接收结束错误标志。  
该 bit 只用于 UART DMA 接收数据。对于接收描述符，硬件在收完 1 个包并检测到接收数据错误会将该 bit 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 中读取数据时能够读取的字节数；对于接收描述符该段由硬件使用完该 buffer 后自动填写，向 buffer 中存储数据时表示已存数据的字节数。  
当 DMA 访问片外 RAM 时，该大小必须是 16/32/64 bytes 的整数倍，详情见章节 7.3.6 访问片外 RAM。
- `size` (DW0) [11:0]: 表示当前描述符对应的 buffer 的大小。  
当 DMA 访问片外 RAM 时，该大小必须是 16/32/64 bytes 的整数倍，详情见章节 7.3.6 访问片外 RAM。
- `buffer address pointer` (DW1): buffer 地址指针。  
当 DMA 访问片外 RAM 时，写地址必须根据 `PERI_EXT_MEM_BK_SIZE` 大小对齐，详情见章节 7.3.6 访问片外 RAM。
- `next descriptor address` (DW2): 下一个描述符的地址指针。当前描述符为链表中最后一个描述符时 (`suc_eof = 1`)，该值可以为 0。并且描述符的地址指针只能在片内 RAM。

用 DMA 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 长度，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。

### 7.3.3 启动 DMA

软件通过挂载链表的方式来使用 DMA。对于接收数据，软件挂载好接收链表，配置 `PERI_INLINK_ADDR` 字段指向第一个接收链表描述符。置位 `PERI_INLINK_START` 位启动 DMA。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 `PERI_OUTLINK_ADDR` 字段指向第一个发送链表描述符。置位 `PERI_OUTLINK_START` 位启动 DMA。`PERI_INLINK_START` 与 `PERI_OUTLINK_START` 位由硬件自动清零。

DMA 支持 Restart 功能。如果不确定已挂载链表的使用状态（使用完或正在使用中），又希望挂载新的链表，利用 Restart 功能，可以完成挂载新的链表而又不影响已挂载链表的正常使用。软件使用该功能时，需要重写已挂载链表的最后一个描述符，使其第三个 word 中的内容指向新链表的首地址；然后挂载新的链表，如图 7-4 所示；并置位 `PERI_INLINK_RESTART` 或者 `PERI_OUTLINK_RESTART`（这两个位由硬件自动清零）。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，从而继续读取新挂载的链表。

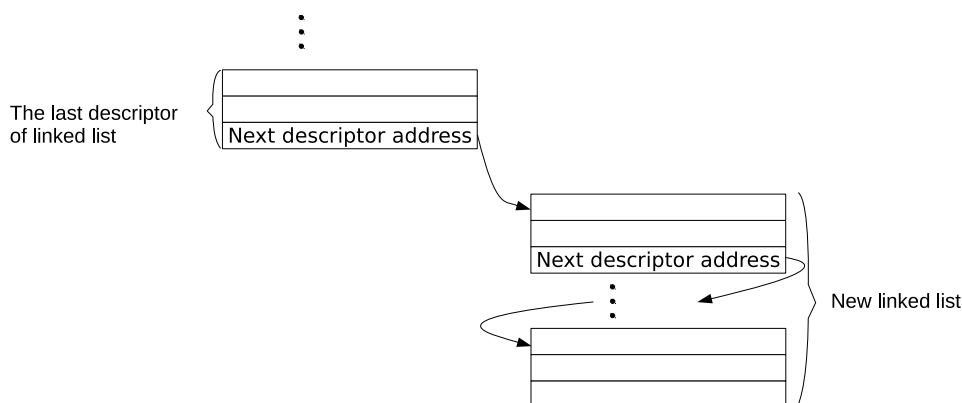


图 7-4. 链表关系图

### 7.3.4 读链表

软件在配置并启动 DMA 后，DMA 会从内部 RAM 读取链表。置位 *PERI\_IN\_DSCR\_ERR\_INT\_ENA* 或者 *PERI\_OUT\_DSCR\_ERR\_INT\_ENA* 位使能描述符错误中断。访问片内 RAM 时，当描述符中第二个 word 指示的地址不在 0x3FFB0000 ~ 0x3FFFFFFF；访问片外 RAM 时，当描述符中第二个 word 指示的地址不在 0x3F500000 ~ 0x3FF7FFFF 时，产生描述符错误中断。

**注意：**描述符的第三个 word 指示的地址只能在片内，指向下一个可用描述符；所有描述符都需存在内存中。

### 7.3.5 数据传输结束标志

DMA 通过 EOF 来指示数据传输结束。发送数据时，置位 *PERI\_OUT\_TOTAL\_EOF\_INT\_ENA* 位使能 *PERI\_OUT\_TOTAL\_EOF\_INT* 中断，当带有 EOF 标志的描述符对应 buffer 的数据传输完成后，DMA 会产生该中断。接收数据时，置位 *PERI\_IN\_SUC\_EOF\_INT\_ENA* 位使能 *PERI\_IN\_SUC\_EOF\_INT* 中断，表示数据接收完成。对于 UART DMA 还支持中断 *UHCL\_IN\_ERR\_EOF\_INT*，置位 *UHCL\_IN\_ERR\_EOF\_INT\_ENA* 使能该中断中断，表示接收完成但接收数据有错误。

软件在检测到 *PERI\_OUT\_TOTAL\_EOF\_INT* 或 *PERI\_IN\_SUC\_EOF\_INT* 中断时，可以记录 *PERI\_OUTLINK\_DSCR\_ADDR* 或 *PERI\_INLINK\_DSCR\_ADDR* 字段的值，即最后一个描述符的地址。需要注意，这两个地址为真实描述符地址右移两 bit 的结果。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。

### 7.3.6 访问片外 RAM

并不是所有外设的 DMA 都支持访问片外 RAM，ESP32-S2 只有 I<sup>2</sup>S0, SPI2, AES 和 SHA 的 DMA 支持访问片外 RAM。DMA 可访问的片外 RAM 地址空间为：0x3F500000 ~ 0x3FF7FFFF。软件配置地址时需要注意写地址（向片外 RAM 写入数据时的地址）必须按 16/32/64 字节大小对齐，表 35 显示了写地址分别为 16 位、32 位、64 位对齐时 *PERI\_EXT\_MEM\_BK\_SIZE* 位的值。

**注意：**读地址（从片外 RAM 读出数据时的地址）不需要进行地址对齐。

表 35: 配置寄存器与写地址的关系表

<i>PERI_EXT_MEM_BK_SIZE</i>	写地址对齐方式
0	16 位对齐

1	32 位对齐
2	64 位对齐

7.4 Copy DMA 控制器

Copy DMA 用于芯片内存与内存之间数据的传输，并且只能访问片内 RAM。图7-5所示为 Copy DMA 引擎的架构，与 Internal DMA 与 EDMA 不同的是，Copy DMA 通过发送链表将片内 RAM 的数据取出存入 DMA 内部 FIFO，并通过接收链表将内部 FIFO 中的数据写入目标片内 RAM。

Copy DMA 的软件配置流程如下：

- 1. 将 CP\_DMA\_IN\_RST、CP\_DMA\_OUT\_RST、CP\_DMA\_FIFO\_RST 和 CP\_DMA\_CMDFIFO\_RST 位先置 1 后置 0，复位 Copy DMA 状态机和 FIFO 指针；
- 2. 将 CP\_DMA\_FIFO\_RST 位先置 1 然后置 0，复位 FIFO 指针；
- 3. 挂载好发送链表，配置 CP\_DMA\_OUTLINK\_ADDR 指向第一个发送链表描述符；
- 4. 挂载好接收链表，配置 CP\_DMA\_INLINK\_ADDR 指向第一个接收链表描述符；
- 5. 置位 CP\_DMA\_OUTLINK\_START 启动 DMA 发送；
- 6. 置位 CP\_DMA\_INLINK\_START 启动 DMA 接收。

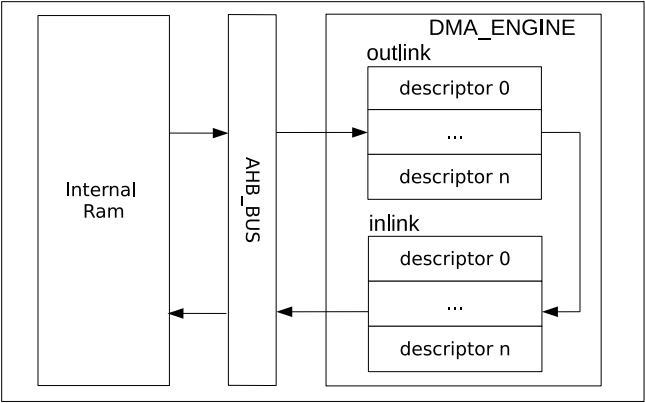


图 7-5. Copy DMA 引擎架构

7.5 UART DMA (UDMA) 控制器

ESP32-S2 中有 2 个 UART 接口，它们共用 1 个 UDMA 控制器。UHCL\_UART\_CE 寄存器用于选择哪个串口占用 UDMA。

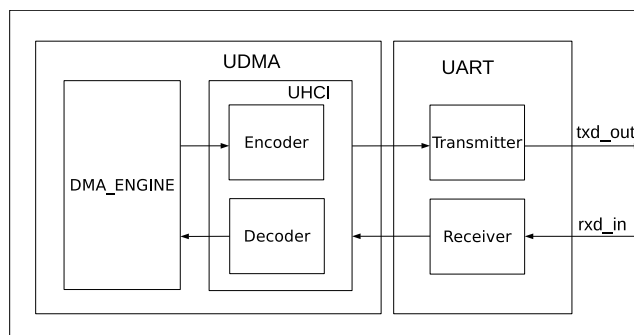


图 7-6. UDMA 模式数据传输

图 7-6 为 UDMA 方式数据传输图。在 UDMA 接收数据前，软件将接收链表准备好。UHCL\_INLINK\_ADDR 用于指向第一个接收链表描述符。置位 UHCL\_INLINK\_START 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 UDMA 的控制下存入接收链表指定的 RAM 空间。

在 UDMA 发送数据前，软件需要将发送链表和发送数据准备好，UHCL\_OUTLINK\_ADDR 用于指向第一个发送链表描述符。置位 UHCL\_OUTLINK\_START 之后，DMA 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。

UDMA 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用特殊字符替换。Decoder 用于去除数据包前后分隔符，并将数据中的特殊字符进行替换为分隔符。数据前后的分隔符可以有连续多个。分隔符可由 UHCL\_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 UHCL\_ESC\_SEQ0\_CHAR0（默认为 0xDB）和 UHCL\_ESC\_SEQ0\_CHAR1（默认为 0xDD）进行替换。当数据全部发送完成后，会产生 UHCL\_OUT\_TOTAL\_EOF\_INT 中断。当数据接收完成后，会产生 UHCL\_IN\_SUC\_EOF\_INT 中断。

## 7.6 SPI DMA 控制器

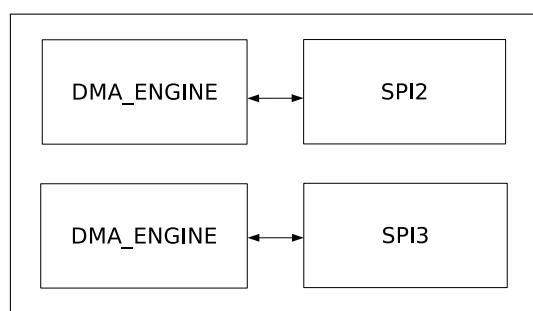


图 7-7. SPI DMA

ESP32-S2 SPI 除了使用 CPU 实现与外设的数据交换外，还可以使用 DMA。如图 7-7 所示，SPI2 和 SPI3 分别单独使用一个 DMA。

ESP32-S2 SPI DMA 使用链表接收/发送数据，发送数据支持 burst 操作，一次接收/发送的数据长度至少为 1 个字节。

寄存器 SPI\_DMA\_OUT\_LINK\_REG 的 SPI\_OUTLINK\_START 位和寄存器 SPI\_DMA\_IN\_LINK\_REG 的 SPI\_INLINK\_START 位用于使能 DMA 引擎，这两个位由硬件清零。当 SPI\_OUTLINK\_START 位被置为 1 时，DMA 引擎开始处理发送链表，并准备发送数据；当 SPI\_INLINK\_START 位被置为 1 时，DMA 引擎开始处理接收链表，并准备接收数据。

SPI DMA 接收数据时软件配置流程如下：

1. 将 SPI\_DMA\_IN\_RST、SPI\_AHBM\_FIFO\_RST 和 SPI\_AHBM\_RST 位先置 1 后置 0，复位 DMA 状态机和 FIFO 指针；
2. 挂载好接收链表，配置 SPI\_DMA\_INLINK\_ADDR 指向第一个接收链表描述符；
3. 置位 SPI\_DMA\_INLINK\_START 启动 DMA 接收。

SPI DMA 发送数据时的软件配置流程如下：

1. 将 SPI\_DMA\_OUT\_RST、SPI\_AHBM\_FIFO\_RST 和 SPI\_AHBM\_RST 位先置 1 后置 0，复位 DMA 状态机和 FIFO 指针；
2. 挂载好发送链表，配置寄存器 SPI\_DMA\_OUTLINK\_ADDR 指向第一个发送链表描述符；
3. 置位 SPI\_DMA\_OUTLINK\_START 启动 DMA 发送。

**注意：**在使用 SPI2 与 SPI3 DMA 实现片内 RAM 与片外 RAM 之间数据传输时，需要置位 SPI\_MEM\_TRANS\_EN。

SPI2 与 SPI3 DMA 还支持分段传输模式。

## 7.7 I<sup>2</sup>S DMA 控制器

ESP32-S2 I<sup>2</sup>S 单独使用一个 DMA。寄存器 I2S\_FIFO\_CONF\_REG 的 I2S\_DSCR\_EN 位用于使能 I<sup>2</sup>S 的 DMA 操作。ESP32-S2 I<sup>2</sup>S DMA 使用链表接收/发送数据，发送数据支持 burst 操作。寄存器 I2S\_RXEOF\_NUM\_REG 的 I2S\_RX\_EOF\_NUM[31:0] 位用于配置 DMA 一次接收数据个数，单位为字。

寄存器 I2S\_OUT\_LINK\_REG 的 I2S\_OUTLINK\_START 位和寄存器 I2S\_IN\_LINK\_REG 的 I2S\_INLINK\_START 位用于使能 DMA 引擎，这两个位由硬件清零。当 I2S\_OUTLINK\_START 位被置为 1 时，DMA 引擎开始处理发送链表，并准备发送数据，当 I2S\_INLINK\_START 位被置为 1 时，DMA 引擎开始处理接收链表，并准备接收数据。

I<sup>2</sup>S DMA 接收数据时的软件配置流程如下：

1. 将 I2S\_IN\_RST、I2S\_AHBM\_FIFO\_RST 和 I2S\_AHBM\_RST 位先置 1 后置 0，复位 DMA 状态机和 FIFO 指针；
2. 挂载好接收链表，配置 I2S\_INLINK\_ADDR 指向第一个接收链表描述符；
3. 置位 I2S\_INLINK\_START 启动 DMA 接收。

I<sup>2</sup>S DMA 发送数据时的软件配置流程如下：

1. 对 I2S\_OUT\_RST、I2S\_AHBM\_FIFO\_RST 和 I2S\_AHBM\_RST 位置 1 然后置 0，复位 DMA 状态机和 FIFO 指针；
2. 挂载好发送链表，配置 I2S\_OUTLINK\_ADDR 指向第一个发送链表描述符；

3. 置位 I2S\_OUTLINK\_START 启动 DMA 发送。

**注意:** 在使用 I<sup>2</sup>S DMA 实现片内 RAM 与片外 RAM 之间数据传输时，需要置位 I2S\_MEM\_TRANS\_EN。

## 8. 通用异步收发传输器 (UART)

### 8.1 概述

嵌入式应用通常要求一个简单的并且占用系统资源少的方法来传输数据。通用异步收发传输器 (UART) 即可以满足这些要求，它能够灵活地与外部设备进行全双工数据交换。芯片中有 2 个 UART 控制器可供使用，并且兼容不同的 UART 设备。另外，UART 还可以用作红外数据交换 (IrDA) 或 RS485 调制解调器。

2 个 UART 控制器分别有一组功能相同的寄存器。本文以 UART $n$  指代 2 个 UART 控制器， $n$  为 0、1。

### 8.2 主要特性

- 可编程收发波特率
- 2 个 UART 的发送 FIFO 以及接收 FIFO 共享 512 x 8-bit RAM
- 全双工异步通信
- 支持输入信号波特率自检功能
- 支持 5/6/7/8 位数据长度
- 支持 1/1.5/2/3 个停止位
- 支持奇偶校验位
- 支持 AT\_CMD 特殊字符检测
- 支持 RS485 协议
- 支持 IrDA 协议
- 支持 DMA 高速数据通信
- 支持 UART 唤醒模式
- 支持软件流控和硬件流控

### 8.3 功能描述

#### 8.3.1 UART 简介

UART 是一种以字符为导向的通用数据链，可以实现设备间的通信。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。芯片上的 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 DMA，可以实现无缝高速的数据传输。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。

### 8.3.2 UART 架构

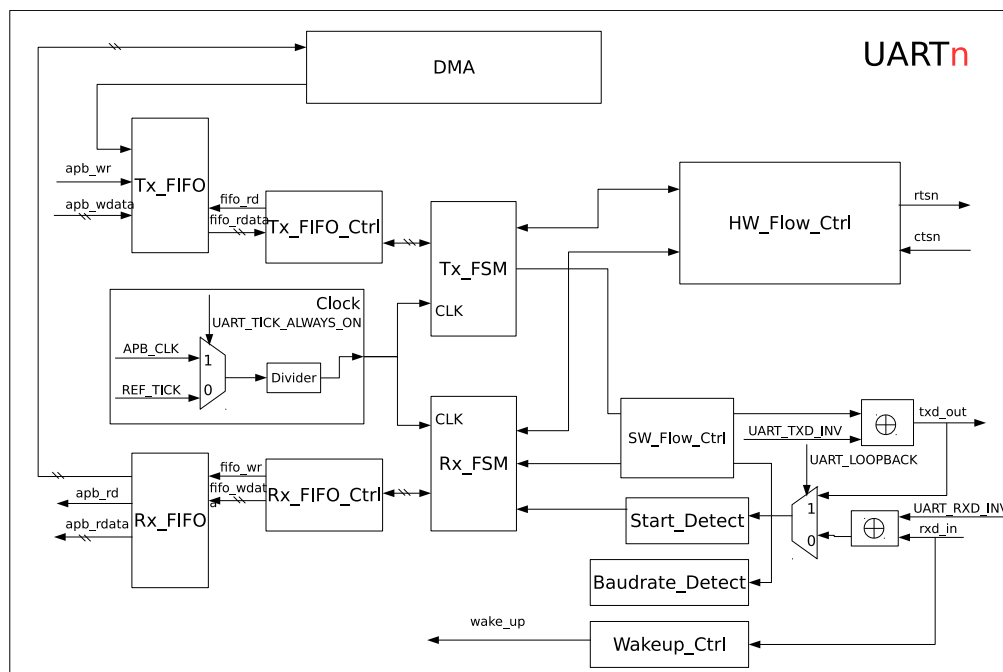


图 8-1. UART 基本架构图

图 8-1 为 UART 基本架构图。UART 有两个时钟源：80-MHz APB\_CLK 以及参考时钟 REF\_TICK（详情请参考章节 2 复位和时钟）。可以通过配置 `UART_TICK_REF_ALWAYS_ON` 来选择时钟源。分频器用于对时钟源进行分频，然后产生时钟信号来驱动 UART 模块。`UART_CLKDIV_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。

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 DMA 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 START 位，当检测到 START 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 DMA 方式进行数据接收。

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 状态时，Wakeup\_Ctrl 开始计算 rxd\_in 的上升沿个数，当上升沿个数大于 (`UART_ACTIVE_THRESHOLD + 2`) 时产生 wake\_up 信号给 RTC 模块，由 RTC 来唤醒芯片。



### 8.3.3 UART RAM

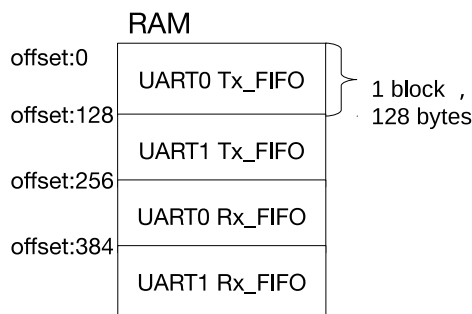


图 8-2. UART 共享 RAM 图

芯片中 2 个 UART 控制器共用 512x8-bit RAM 空间。如图 8-2 所示，RAM 以 block 为单位进行分配，1 block 为 128x8 bits。图 8-2 所示为默认情况下 2 个 UART 控制器的 Tx\_FIFO 和 Rx\_FIFO 占用 RAM 的情况。通过配置 `UART_TX_SIZE` 可以对 UART $n$  的 Tx\_FIFO 进行扩展，通过配置 `UART_RX_SIZE` 可以对 UART $n$  的 Rx\_FIFO 进行扩展，UART0 Tx\_FIFO 可以从地址 0 扩展到整个 RAM 空间，UART1 Tx\_FIFO 可以从地址 128 扩展到 RAM 的尾地址，UART0 Rx\_FIFO 可以从地址 256 扩展到 RAM 的尾地址，UART1 Rx\_FIFO 则不支持地址空间扩展，以 1 block 为单位进行扩展。需要注意的是当扩展某一个 UART 的 FIFO 空间时可能会占用其他 UART 的 FIFO 空间。比如，设置 UART0 的 `UART_TX_SIZE` 为 2，则 UART0 Tx\_FIFO 的地址从 0 扩展到 255。这时，UART1 Tx\_FIFO 的默认空间被占用，这时将不能使用 UART1 发送器功能。

当 2 个 UART 控制器都不工作时，可以通过置位 `UART_MEM_FORCE_PD` 来使 RAM 进入低功耗状态。

UART0 和 UART1 的 Tx\_FIFO 可以通过置位 `UART_TXFIFO_RST` 来复位，UART0 和 UART1 的 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 DMA 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 DMA 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 DMA。

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 `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。

### 8.3.4 波特率产生与检测

#### 8.3.4.1 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_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。

波特率分频器系数由  $\text{UART\_CLKDIV} + (\text{UART\_CLKDIV\_FRAG}/16)$  构成。也就是说，最终波特率为  $\text{INPUT\_FREQ}/(\text{UART\_CLKDIV} + (\text{UART\_CLKDIV\_FRAG}/16))$ 。例如，若 `UART_CLKDIV` = 694，

$\text{UART\_CLKDIV\_FRAG} = 7$ ，则分频系数为  $(694 + 7/16) = 694.4375$ 。

$\text{UART\_CLKDIV\_FRAG}$  为 0 时，分频器为整数分频，每  $\text{UART\_CLKDIV}$  个输入脉冲都会产生一个输出脉冲。

$\text{UART\_CLKDIV\_FRAG}$  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 8-3 所示，每 16 个输出脉冲，波特率发生器分频  $(\text{UART\_CLKDIV} + 1)$  个输入脉冲或  $\text{UART\_CLKDIV}$  个输入脉冲。分频  $(\text{UART\_CLKDIV} + 1)$  个输入脉冲产生  $\text{UART\_CLKDIV\_FRAG}$  个输出脉冲，分频  $\text{UART\_CLKDIV}$  个输入脉冲产生剩余的  $(16 - \text{UART\_CLKDIV\_FRAG})$  个输出脉冲。

如图 8-3 所示，输出脉冲相互交错，使得输出时序更加统一。

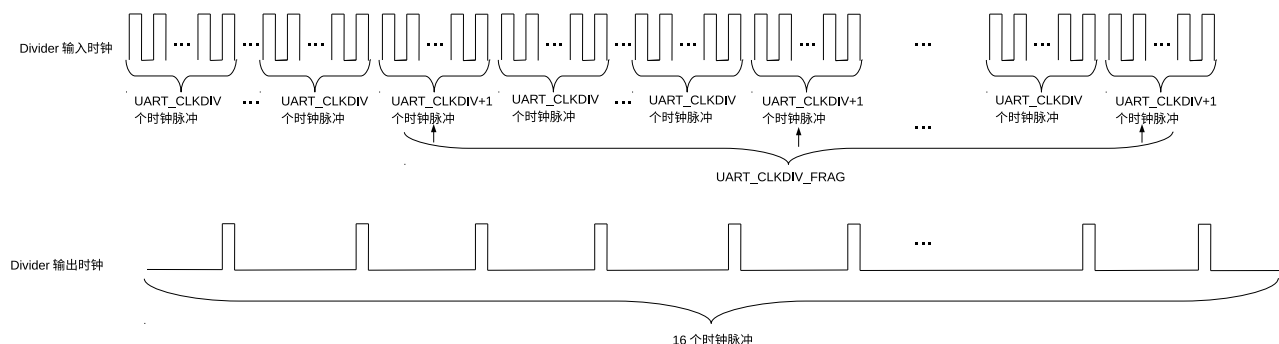


图 8-3. UART 控制器分频

为了支持 IrDA（详情见本节 8.3.7 IrDA），IrDA 小数分频器会产生  $16 \times \text{UART\_CLKDIV\_REG}$  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  $\text{UART\_CLKDIV}/16$  作为分频值的整数部分，取  $\text{UART\_CLKDIV}$  的低 4 比特作为小数部分。

### 8.3.4.2 波特率检测

置位  $\text{UART\_AUTOBAUD\_EN}$  可以开启 UART 波特率自检测功能。图 8-1 中的 Baudrate\_Detect 可以滤除信号脉宽小于  $\text{UART\_GLITCH\_FLT}$  的噪声。

在 UART 双方进行通信之前可以通过发送几个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。 $\text{UART\_LOWPULSE\_MIN\_CNT}$  存储了最小低电平脉冲宽度， $\text{UART\_HIGHPULSE\_MIN\_CNT}$  存储了最小高电平脉冲宽度， $\text{UART\_POSEDGE\_MIN\_CNT}$  存储了两个上升沿之间的最小脉冲宽度， $\text{UART\_NEGEDGE\_MIN\_CNT}$  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。

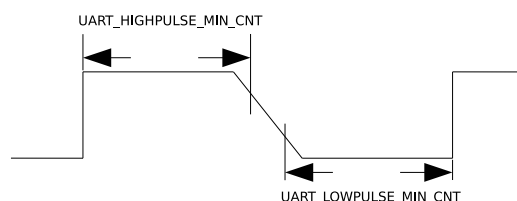


图 8-4. UART 信号下降沿较差时序图

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  $\text{UART\_LOWPULSE\_MIN\_CNT}$  或者  $\text{UART\_HIGHPULSE\_MIN\_CNT}$  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, 对于 UART 信号的下降沿信号比较差的情况, 如图8-4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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{\text{UART\_POSEDGE\_MIN\_CNT}/2}$$

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:

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{\text{UART\_NEGEDGE\_MIN\_CNT}/2}$$

8.3.5 UART 数据帧

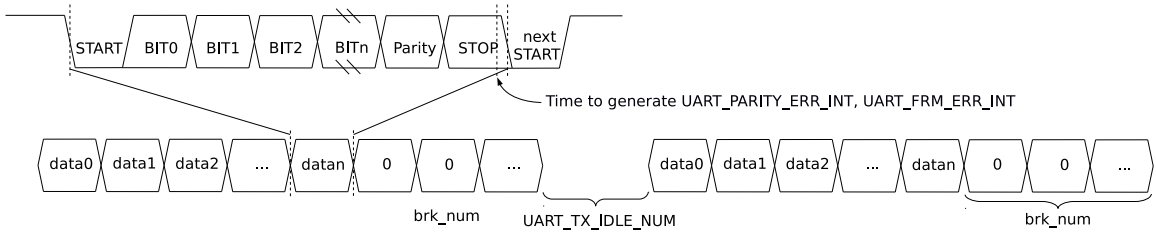


图 8-5. UART 数据帧结构

图 8-5 所示为基本数据帧格式, 数据帧从 START 位开始以 STOP 位结束。START 占用 1 bit, STOP 位可以通过配置 `UART_STOP_BIT_NUM`、`UART_DL1_EN` 和 `UART_DLO_EN` 实现 1/1.5/2/3 位宽。START 为低电平, STOP 为高电平。

数据位宽 (BIT0 ~ BITn) 为 5 ~ 8 bit, 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继续发送几个连续的特殊数据帧 NULL, 在 NULL 数据帧, 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。

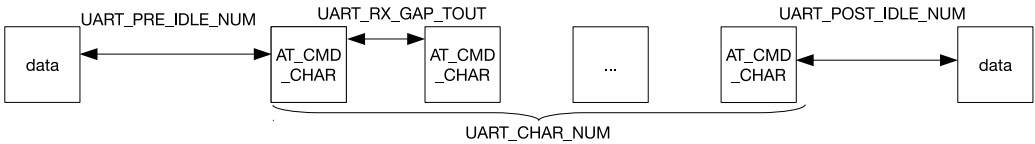


图 8-6. AT\_CMD 字符格式

图 8-6 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。

### 8.3.6 RS485

UART 支持 RS485 协议，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模式，UART 模块采用两线半双工模式，并支持侦听总线的功能。RS485 两线 multidrop 模式，最大可支持 32 个 slave。

#### 8.3.6.1 驱动控制

如图 8-7 所示，RS485 两线 multidrop 系统中，需要一个外部 RS485 传输器实现单端信号与差分信号的转换。RS485 传输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。本设计 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 8-7 所示，DE 与 UART 的 dtrn\_out 相连（详见本节 8.3.9.1 硬件流控）。

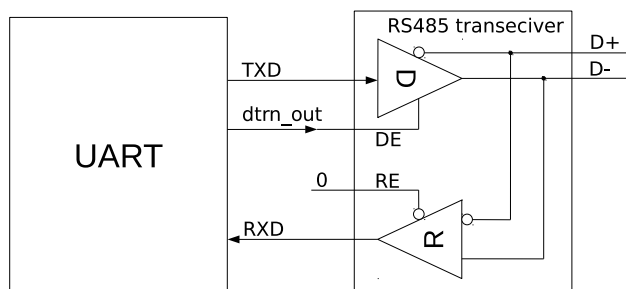


图 8-7. RS485 模式驱动控制结构图

#### 8.3.6.2 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 协议推荐在发送停止位之后增加一个波特率的 turn around dealy。UART 发送模块支持在停止位之后增加两个波特率的延时。置位 UART\_DL1\_EN，增加一个波特率周期延时；置位 DL0\_EN，增加另一个波特率周期延时。

### 8.3.6.3 总线侦听

RS485 两线 multidrop 系统中，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，配合外部 RS485 传输器的配置，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 接收的数据不同时，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；侦听到发送数据极性错误时，触发 `UART_RS485_PARITY_ERR_INT` 中断。

### 8.3.7 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 8-8 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，编码器输出极性与解码器输入极性相反。当检测到低脉冲表示接收到开始信号。

IrDA 使能时，一个比特被划分为 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。

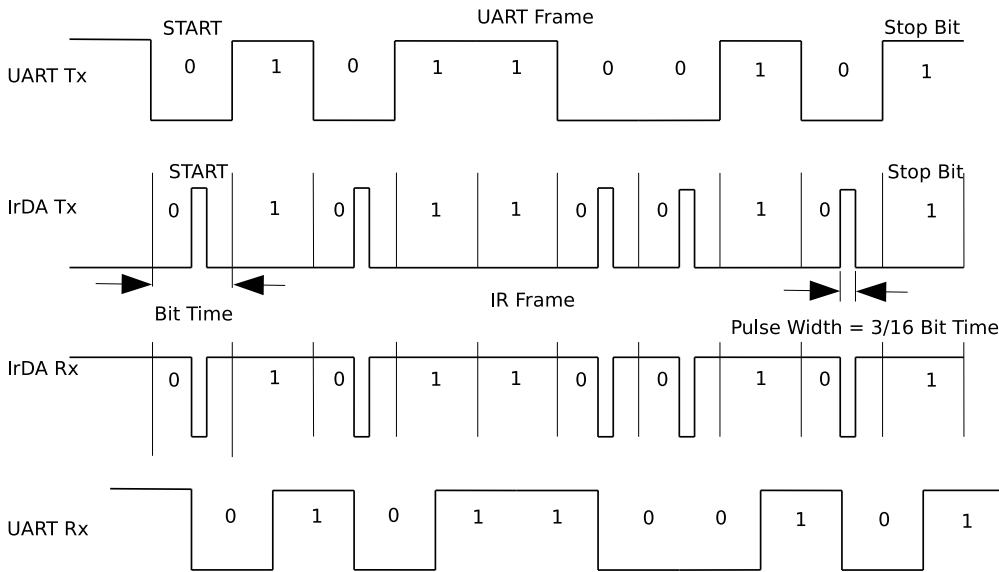


图 8-8. SIR 模式编解码时序图

IrDA 为半双工传输协议，不允许同时进行收发。如图8-9所示，置位 `UART_IRDA_EN` 使能 IrDA 功能。置位 `UART_IRDA_TX_EN`（拉高）使能 IrDA 发送数据，这时不允许 IrDA 接收数据；复位 `UART_IRDA_TX_EN`（拉低）使能 IrDA 接收数据，这时不允许 IrDA 发送数据。

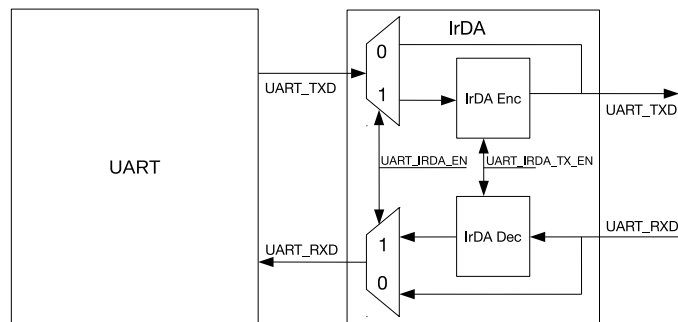


图 8-9. IrDA 编解码结构图

### 8.3.8 唤醒

UART0 和 UART1 支持唤醒功能。当 UART 处于 Light-sleep 状态时，Wakeup\_Ctrl 开始计算 rxd\_in 的上升沿个数，当上升沿个数大于 (`UART_ACTIVE_THRESHOLD` + 2) 时产生 wake\_up 信号给 RTC 模块，由 RTC 来唤醒芯片。

### 8.3.9 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 dsrn\_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。

8.3.9.1 硬件流控

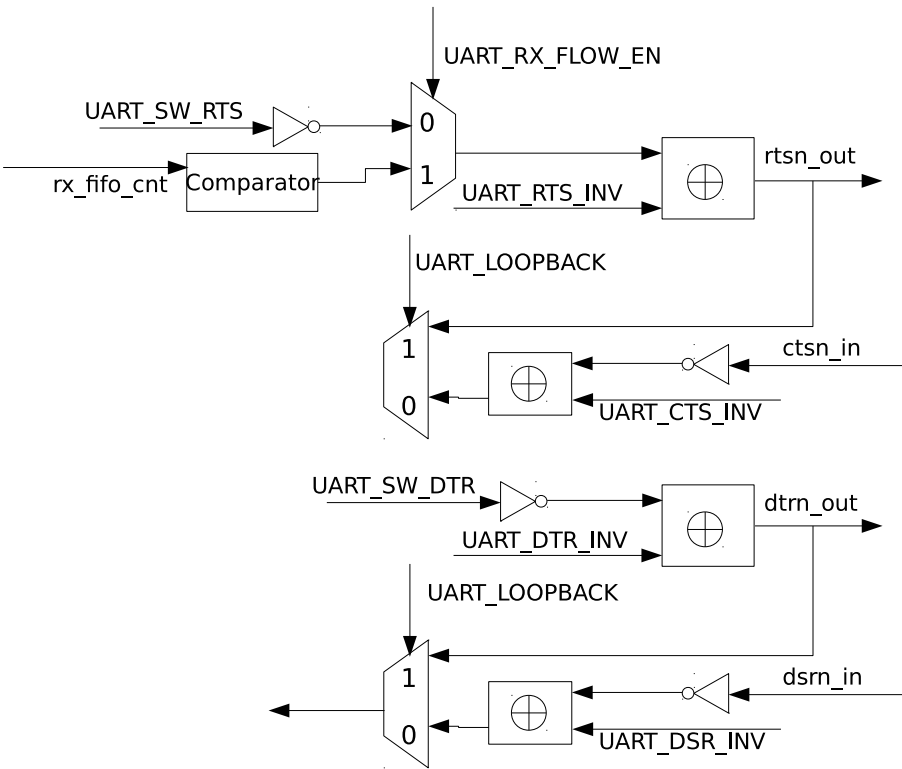


图 8-10. 硬件流控图

图 8-10 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 8-11 为两个 UART 之间硬件流控信号连接图。记 ESP32-S2 UART 为 IU0，External UART 为 EU0，下文将使用这两个标记来区分两个 UART。输出信号 rtsn\_out (IU0) 为低电平表示允许对方 (EU0) 发送数据，rtsn\_out (IU0) 为高电平表示通知对方 (EU0) 中止数据发送直到 rtsn\_out (IU0) 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。

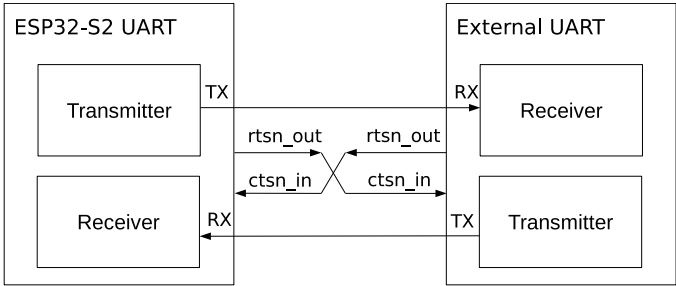


图 8-11. 硬件流控信号连接图

输入信号 ctsn\_in (IU0) 为低电平表示允许发送端 (IU0) 发送数据；ctsn\_in (IU0) 为高电平表示禁止发送端



(IU0) 发送数据。当 UART 检测到输入信号 `ctsn_in` (IU0) 的沿变化时会产生 `UART_CTS_CHG_INT` 中断。

UART 发送设备 (IU0) 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 (IU0) 在检测到输入信号 `dsrn_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 `dsrn_in` 的输入信号电平，`UART_DSRN` 为高电平时，表示对方设备 (EU0) 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，`rtn_out` 和 `ctsn_in` 相连，`dtrn_out` 和 `dsrn_out` 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。

### 8.3.9.2 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。

### 8.3.10 UDMA

UART 共用一个 UDMA (UART DMA)，UDMA 支持对 HCI 协议数据包的解析 (decoder) 及数据包封装 (encoder)。更多信息请见章节 7 DMA 控制器。

### 8.3.11 UART 中断

- `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 glitch 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: 当接收器接收一个字节的的时间大于 [UART\\_RX\\_TOUT\\_THRHD](#) 时触发此中断。
- UART\_BRK\_DET\_INT: 当接收器在停止位之后检测到 NULL 时触发此中断。
- UART\_CTS\_CHG\_INT: 当接收器检测到 CTSn 信号的沿变化时触发此中断。
- UART\_DSR\_CHG\_INT: 当接收器检测到 DSRn 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。

### 8.3.12 UCHI 中断

- UHCI\_DMA\_INFIFO\_FULL\_WM\_INT: 当 DMA 接收 FIFO 计数器的值大于 [UHCI\\_DMA\\_INFIFO\\_FULL\\_THRS](#) 时触发此中断。
- UHCI\_SEND\_A\_REG\_Q\_INT: 当使用 `always_send` 发送一串短包, DMA 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, DMA 发送了短包后触发此中断。
- UHCI\_OUT\_TOTAL\_EOF\_INT: 当所有数据都已发送时触发此中断。
- 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\_IN\_DSCR\_EMPTY\_INT: 当 DMA 没有足够的接收链表描述符时触发此中断。
- UHCI\_OUT\_DSCR\_ERR\_INT: 当发送链表描述符里有错误时触发此中断。
- UHCI\_IN\_DSCR\_ERR\_INT: 当接收链表描述符里有错误时触发此中断。
- UHCI\_OUT\_EOF\_INT: 当前描述符的 EOF 位为 1 时触发此中断。
- UHCI\_OUT\_DONE\_INT: 当发送链表描述符完成时触发此中断。

- UHCI\_IN\_ERR\_EOF\_INT: 当接收链表描述符中的 EOF 有错误时触发此中断。
- UHCI\_IN\_SUC\_EOF\_INT: 当接收一个数据包时触发此中断。
- UHCI\_IN\_DONE\_INT: 当一个接收链表描述符完成时触发此中断。
- UHCI\_TX\_HUNG\_INT: 当 DMA 从 RAM 中读取数据的时间过长时触发此中断。
- UHCI\_RX\_HUNG\_INT: 当 DMA 接收数据的时间过长时触发此中断。
- UHCI\_TX\_START\_INT: 当 DMA 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。

## 8.4 基地址

用户可以通过两个不同的寄存器基地址访问 UART，如表 36 所示。更多信息，请访问 [1 系统和存储器](#) 章节。

表 36: UART0、UART1 与 UHCI 基地址

模块名	访问总线	基地址
UART0	PeriBUS1	0x3F400000
	PeriBUS2	0x60000000
UART1	PeriBUS1	0x3F410000
	PeriBUS2	0x60010000
UHCI	PeriBUS1	0x3F414000
	PeriBUS2	0x60014000

## 8.5 寄存器列表

请注意，下表中的地址都是相对于 UART0、UART1 和 UHCI 基地址的地址偏移量（相对地址）。更多有关 UART0、UART1 和 UHCI 基地址的信息，请前往 [8.4 基地址](#) 章节。

名称	描述	地址	访问
<b>FIFO 配置</b>			
<a href="#">UART_FIFO_REG</a>	FIFO 数据寄存器	0x0000	只读
<a href="#">UART_MEM_CONF_REG</a>	UART 阈值和分配配置	0x005C	读/写
<b>中断寄存器</b>			
<a href="#">UART_INT_RAW_REG</a>	原始中断状态	0x0004	只读
<a href="#">UART_INT_ST_REG</a>	屏蔽中断状态	0x0008	只读
<a href="#">UART_INT_ENA_REG</a>	中断使能位	0x000C	读/写
<a href="#">UART_INT_CLR_REG</a>	中断清除位	0x0010	只写
<b>配置寄存器</b>			
<a href="#">UART_CLKDIV_REG</a>	时钟分频配置	0x0014	读/写
<a href="#">UART_CONF0_REG</a>	配置寄存器 0	0x0020	读/写
<a href="#">UART_CONF1_REG</a>	配置寄存器 1	0x0024	读/写
<a href="#">UART_FLOW_CONF_REG</a>	软件流控配置	0x0034	读/写

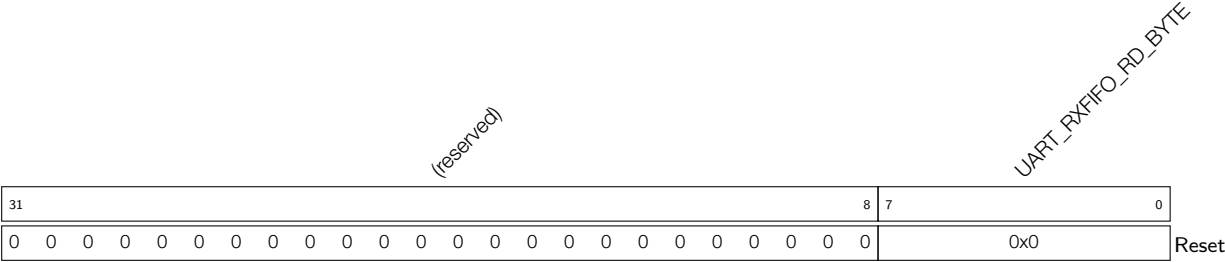
名称	描述	地址	访问
UART_SLEEP_CONF_REG	睡眠模式配置	0x0038	读/写
UART_SWFC_CONF0_REG	软件流控字符配置	0x003C	读/写
UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	读/写
UART_IDLE_CONF_REG	帧结束空闲配置	0x0044	读/写
UART_RS485_CONF_REG	RS485 模式配置	0x0048	读/写
<b>自动波特率检测寄存器</b>			
UART_AUTOBAUD_REG	自动波特率检测配置寄存器	0x0018	读/写
UART_LOWPULSE_REG	自动波特率检测最低电平脉冲持续时间寄存器	0x0028	只读
UART_HIGHPULSE_REG	自动波特率检测最短高电平脉冲持续时间寄存器	0x002C	只读
UART_RXD_CNT_REG	自动波特率检测沿变化计数寄存器	0x0030	只读
UART_POSPULSE_REG	自动波特率检测高电平脉冲寄存器	0x006C	只读
UART_NEGPULSE_REG	自动波特率检测低电平脉冲寄存器	0x0070	只读
<b>状态寄存器</b>			
UART_STATUS_REG	UART 状态寄存器	0x001C	只读
UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0060	只读
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x0064	只读
UART_FSM_STATUS_REG	UART 发送和接收状态	0x0068	只读
<b>AT 转义序列检测配置</b>			
UART_AT_CMD_PRECNT_REG	AT_CMD 字符序列发送前的时序配置	0x004C	读/写
UART_AT_CMD_POSTCNT_REG	AT_CMD 字符发送后的时序配置	0x0050	读/写
UART_AT_CMD_GAPTOOUT_REG	超时配置	0x0054	读/写
UART_AT_CMD_CHAR_REG	AT 转义序列检测配置	0x0058	读/写
<b>版本寄存器</b>			
UART_DATE_REG	UART 版本控制寄存器	0x0074	读/写

名称	描述	地址	访问
<b>配置寄存器</b>			
UHCI_CONF0_REG	UHCI 配置寄存器	0x0000	读/写
UHCI_CONF1_REG	UHCI 配置寄存器	0x002C	读/写
UHCI_AHB_TEST_REG	AHB 测试寄存器	0x0048	读/写
UHCI_ESCAPE_CONF_REG	转义符配置	0x0064	读/写
UHCI_HUNG_CONF_REG	超时配置	0x0068	读/写
UHCI_QUICK_SENT_REG	UHCI 快速发送配置寄存器	0x0074	读/写
UHCI_REG_Q0_WORD0_REG	Q0_WORD0 快速发送寄存器	0x0078	读/写
UHCI_REG_Q0_WORD1_REG	Q0_WORD1 快速发送寄存器	0x007C	读/写
UHCI_REG_Q1_WORD0_REG	Q1_WORD0 快速发送寄存器	0x0080	读/写
UHCI_REG_Q1_WORD1_REG	Q1_WORD1 快速发送寄存器	0x0084	读/写
UHCI_REG_Q2_WORD0_REG	Q2_WORD0 快速发送寄存器	0x0088	读/写
UHCI_REG_Q2_WORD1_REG	Q2_WORD1 快速发送寄存器	0x008C	读/写
UHCI_REG_Q3_WORD0_REG	Q3_WORD0 快速发送寄存器	0x0090	读/写
UHCI_REG_Q3_WORD1_REG	Q3_WORD1 快速发送寄存器	0x0094	读/写

名称	描述	地址	访问
UHCI_REG_Q4_WORD0_REG	Q4_WORD0 快速发送寄存器	0x0098	读/写
UHCI_REG_Q4_WORD1_REG	Q4_WORD1 快速发送寄存器	0x009C	读/写
UHCI_REG_Q5_WORD0_REG	Q5_WORD0 快速发送寄存器	0x00A0	读/写
UHCI_REG_Q5_WORD1_REG	Q5_WORD1 快速发送寄存器	0x00A4	读/写
UHCI_REG_Q6_WORD0_REG	Q6_WORD0 快速发送寄存器	0x00A8	读/写
UHCI_REG_Q6_WORD1_REG	Q6_WORD1 快速发送寄存器	0x00AC	读/写
UHCI_ESC_CONF0_REG	转义序列配置寄存器 0	0x00B0	读/写
UHCI_ESC_CONF1_REG	转义序列配置寄存器 1	0x00B4	读/写
UHCI_ESC_CONF2_REG	转义序列配置寄存器 2	0x00B8	读/写
UHCI_ESC_CONF3_REG	转义序列配置寄存器 3	0x00BC	读/写
UHCI_PKT_THRES_REG	包长度配置寄存器	0x00C0	读/写
<b>中断寄存器</b>			
UHCI_INT_RAW_REG	原始中断状态	0x0004	只读
UHCI_INT_ST_REG	屏蔽中断状态	0x0008	只读
UHCI_INT_ENA_REG	中断使能位	0x000C	读/写
UHCI_INT_CLR_REG	中断清除位	0x0010	只写
<b>DMA 状态</b>			
UHCI_DMA_OUT_STATUS_REG	DMA 数据输出状态寄存器	0x0014	只读
UHCI_DMA_IN_STATUS_REG	UHCI 数据输入状态寄存器	0x001C	只读
UHCI_STATE0_REG	UHCI 解码器状态寄存器	0x0030	只读
UHCI_DMA_OUT_EOF_DES_ADDR_REG	EOF 有效时发送链表描述符的地址	0x0038	只读
UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG	EOF 有效时接收链表描述符的地址	0x003C	只读
UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG	报错时接收链表描述符的地址	0x0040	只读
UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG	倒数第二个发送链表描述符	0x0044	只读
UHCI_DMA_IN_DSCR_REG	下一个接收链表描述符的第三个字	0x004C	只读
UHCI_DMA_IN_DSCR_BF0_REG	当前接收链表描述符的第三个字	0x0050	只读
UHCI_DMA_IN_DSCR_BF1_REG	当前接收链表描述符的第二个字	0x0054	只读
UHCI_DMA_OUT_DSCR_REG	下一个接收链表描述符的第三个字	0x0058	只读
UHCI_DMA_OUT_DSCR_BF0_REG	当前接收链表描述符的第三个字	0x005C	只读
UHCI_DMA_OUT_DSCR_BF1_REG	当前发送链表描述符的第二个字	0x0060	只读
UHCI_RX_HEAD_REG	UHCI 包报头寄存器	0x0070	只读
UHCI_STATE1_REG	UHCI 编码状态寄存器	0x0034	只读
<b>DMA 配置</b>			
UHCI_DMA_OUT_PUSH_REG	数据输出 FIFO 推送控制寄存器	0x0018	读/写
UHCI_DMA_IN_POP_REG	数据输入 FIFO 的弹出控制寄存器	0x0020	不定
UHCI_DMA_OUT_LINK_REG	链表描述符地址与控制	0x0024	不定
UHCI_DMA_IN_LINK_REG	链表描述符地址与控制	0x0028	不定
<b>版本寄存器</b>			
UHCI_DATE_REG	UHCI 版本控制寄存器	0x00FC	读/写

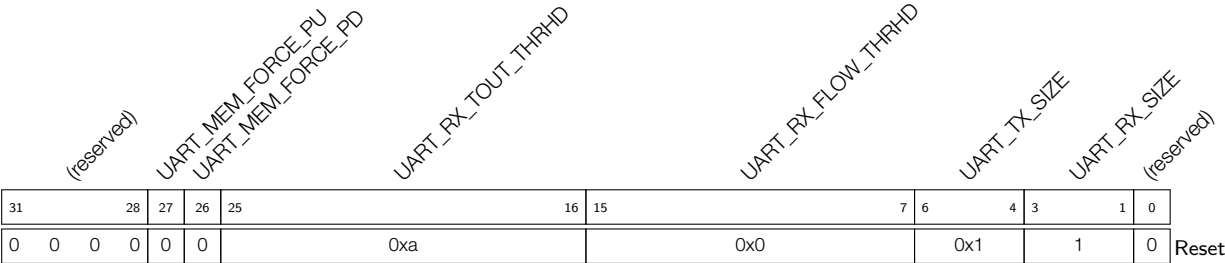
# 8.6 寄存器

Register 8.1: UART\_FIFO\_REG (0x0000)



**UART\_RXFIFO\_RD\_BYTE** 存储从 RX FIFO 读取的一个字节数据。(只读)

Register 8.2: UART\_MEM\_CONF\_REG (0x005C)



**UART\_RX\_SIZE** 配置存储器分配给 RX FIFO 的空间大小。默认为 128 字节。(读/写)

**UART\_TX\_SIZE** 配置存储器分配给 TX FIFO 的空间大小。默认为 128 字节。(读/写)

**UART\_RX\_FLOW\_THRHD** 配置使用硬件流控时接收数据的最大值。(读/写)

**UART\_RX\_TOUT\_THRHD** 配置接收器接收一个字节所需时间的阈值。接收器接收一个字节所需时间超过阈值且 UART\_RX\_TOUT\_EN 置 1 时触发 UART\_RXFIFO\_TOUT\_INT 中断。(读/写)

**UART\_MEM\_FORCE\_PD** 置位此位强制关闭 UART 存储器。(读/写)

**UART\_MEM\_FORCE\_PU** 置位此位强制开启 UART 存储器。(读/写)

Register 8.3: 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_DONE_INT_RAW UART_TX_BRK_IDLE_DONE_INT_RAW UART_GLITCH_DET_INT_RAW UART_SW_XON_INT_RAW UART_SW_XOFF_INT_RAW UART_RXFIFO_TOUT_INT_RAW UART_CTS_CHG_INT_RAW UART_DSR_CHG_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
0	0	0	0	0	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\_RAW** 接收器接收数据多于 UART\_RXFIFO\_FULL\_THRHD 的指定值时，该原始中断位翻转至高电平。(只读)

**UART\_TXFIFO\_EMPTY\_INT\_RAW** TX FIFO 中的数据少于 UART\_TXFIFO\_EMPTY\_THRHD 的指定值时，该原始中断位翻转至高电平。(只读)

**UART\_PARITY\_ERR\_INT\_RAW** 接收器检测到数据奇偶检验位错误时，该原始中断位翻转至高电平。(只读)

**UART\_FRM\_ERR\_INT\_RAW** 接收器检测到数据帧错误时，该原始中断位翻转至高电平。(只读)

**UART\_RXFIFO\_OVF\_INT\_RAW** 接收器接收数据超过 FIFO 的存储容量时，该原始中断位翻转至高电平。(只读)

**UART\_DSR\_CHG\_INT\_RAW** 接收器检测到 DSRn 信号的沿变化时，该原始中断位翻转至高电平。(只读)

**UART\_CTS\_CHG\_INT\_RAW** 接收器检测到 CTSn 信号的沿变化时，该原始中断位翻转至高电平。(只读)

**UART\_BRK\_DET\_INT\_RAW** 接收器在停止位后检测到 0 时，该原始中断位翻转至高电平。(只读)

**UART\_RXFIFO\_TOUT\_INT\_RAW** 接收器接收一个字节所需时间超过 UART\_RX\_TOUT\_THRHD 时，该原始中断位翻转至高电平。(只读)

**UART\_SW\_XON\_INT\_RAW** 接收器接收到 XON 字符且 UART\_SW\_FLOW\_CON\_EN 置 1 时，该原始中断位翻转至高电平。(只读)

**UART\_SW\_XOFF\_INT\_RAW** 接收器接收到 XOFF 字符且 UART\_SW\_FLOW\_CON\_EN 置 1 时，该原始中断位翻转至高电平。(只读)

**UART\_GLITCH\_DET\_INT\_RAW** 接收器在起始位的中点处检测到毛刺时，该原始中断位翻转至高电平。(只读)

**UART\_TX\_BRK\_DONE\_INT\_RAW** 发送器在发送完 TX FIFO 中所有数据后完成 NULL 字符的发送时，该原始中断位翻转至高电平。(只读)

**UART\_TX\_BRK\_IDLE\_DONE\_INT\_RAW** 发送器发送完最后一个数据后的间隔时间达到阈值时，该原始中断位翻转至高电平。(只读)

**UART\_TX\_DONE\_INT\_RAW** 发送器发完 FIFO 中的所有数据后，该原始中断位翻转至高电平。(只读)

**Register 8.3: UART\_INT\_RAW\_REG (0x0004)**

接上页...

**UART\_RS485\_PARITY\_ERR\_INT\_RAW** RS485 模式下接收器检测到发送器回音的数据检验位错误时，该原始中断位翻转至高电平。(只读)

**UART\_RS485\_FRM\_ERR\_INT\_RAW** RS485 模式下接收器检测到发送器回音的数据帧错误时，该原始中断位翻转至高电平。(只读)

**UART\_RS485\_CLASH\_INT\_RAW** RS485 模式下检测到发送器与接收器冲突时，该原始中断位翻转至高电平。(只读)

**UART\_AT\_CMD\_CHAR\_DET\_INT\_RAW** 接收器检测到配置的 UART\_AT\_CMD CHAR 时，该原始中断位翻转至高电平。(只读)

**UART\_WAKEUP\_INT\_RAW** 输入 RXD 沿变化次数超过 Light-sleep 模式指定的 UART\_ACTIVE\_THRESHOLD 值时，该原始中断位翻转至高电平。(只读)



## 176

ESP32-S2 TRM (预发布 V0.2)

## 反馈文档意见

见下页...



Register 8.4: UART\_INT\_ST\_REG (0x0008)

接上页...

- UART\_TX\_DONE\_INT\_ST** UART\_TX\_DONE\_INT\_ENA 置 1 时 UART\_TX\_DONE\_INT\_RAW 的状态位。(只读)
- UART\_RS485\_PARITY\_ERR\_INT\_ST** UART\_RS485\_PARITY\_INT\_ENA 置 1 时 UART\_RS485\_PARITY\_ERR\_INT\_RAW 的状态位。(只读)
- UART\_RS485\_FRM\_ERR\_INT\_ST** UART\_RS485\_FM\_ERR\_INT\_ENA 置 1 时 UART\_RS485\_FRM\_ERR\_INT\_RAW 的状态位。(只读)
- UART\_RS485\_CLASH\_INT\_ST** UART\_RS485\_CLASH\_INT\_ENA 置 1 时 UART\_RS485\_CLASH\_INT\_RAW 的状态位。(只读)
- UART\_AT\_CMD\_CHAR\_DET\_INT\_ST** UART\_AT\_CMD\_CHAR\_DET\_INT\_ENA 置 1 时 UART\_AT\_CMD\_DET\_INT\_RAW 的状态位。(只读)
- UART\_WAKEUP\_INT\_ST** UART\_WAKEUP\_INT\_ENA 置 1 时 UART\_WAKEUP\_INT\_RAW 的状态位。(只读)

Register 8.5: 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_BRK_DET_INT_ENA UART_CTS_CHG_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	Reset			
0				0				0				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

- UART\_RXFIFO\_FULL\_INT\_ENA UART\_RXFIFO\_FULL\_INT\_ST 寄存器的使能位。(读/写)
- UART\_TXFIFO\_EMPTY\_INT\_ENA UART\_TXFIFO\_EMPTY\_INT\_ST 寄存器的使能位。(读/写)
- UART\_PARITY\_ERR\_INT\_ENA UART\_PARITY\_ERR\_INT\_ST 寄存器的使能位。(读/写)
- UART\_FRM\_ERR\_INT\_ENA UART\_FRM\_ERR\_INT\_ST 寄存器的使能位。(读/写)
- UART\_RXFIFO\_OVF\_INT\_ENA UART\_RXFIFO\_OVF\_INT\_ST 寄存器的使能位。(读/写)
- UART\_DSR\_CHG\_INT\_ENA UART\_DSR\_CHG\_INT\_ST 寄存器的使能位。(读/写)
- UART\_CTS\_CHG\_INT\_ENA UART\_CTS\_CHG\_INT\_ST 寄存器的使能位。(读/写)
- UART\_BRK\_DET\_INT\_ENA UART\_BRK\_DET\_INT\_ST 寄存器的使能位。(读/写)
- UART\_RXFIFO\_TOUT\_INT\_ENA UART\_RXFIFO\_TOUT\_INT\_ST 寄存器的使能位。(读/写)
- UART\_SW\_XON\_INT\_ENA UART\_SW\_XON\_INT\_ST 寄存器的使能位。(读/写)
- UART\_SW\_XOFF\_INT\_ENA UART\_SW\_XOFF\_INT\_ST 寄存器的使能位。(读/写)
- UART\_GLITCH\_DET\_INT\_ENA UART\_GLITCH\_DET\_INT\_ST 寄存器的使能位。(读/写)
- UART\_TX\_BRK\_DONE\_INT\_ENA UART\_TX\_BRK\_DONE\_INT\_ST 寄存器的使能位。(读/写)
- UART\_TX\_BRK\_IDLE\_DONE\_INT\_ENA UART\_TX\_BRK\_IDLE\_DONE\_INT\_ST 寄存器的使能位。  
(读/写)
- UART\_TX\_DONE\_INT\_ENA UART\_TX\_DONE\_INT\_ST 寄存器的使能位。(读/写)
- UART\_RS485\_PARITY\_ERR\_INT\_ENA UART\_RS485\_PARITY\_ERR\_INT\_ST 寄存器的使能位。  
(读/写)
- UART\_RS485\_FRM\_ERR\_INT\_ENA UART\_RS485\_PARITY\_ERR\_INT\_ST 寄存器的使能位。(读/写)
- UART\_RS485\_CLASH\_INT\_ENA UART\_RS485\_CLASH\_INT\_ST 寄存器的使能位。(读/写)
- UART\_AT\_CMD\_CHAR\_DET\_INT\_ENA UART\_AT\_CMD\_CHAR\_DET\_INT\_ST 寄存器的使能位。  
(读/写)
- UART\_WAKEUP\_INT\_ENA UART\_WAKEUP\_INT\_ST 寄存器的使能位。(读/写)

### Register 8.6: UART\_INT\_CLR\_REG (0x0010)

[illegible]

**UART\_RXFIFO\_FULL\_INT\_CLR** 置位此位清除 UART\_RXFIFO\_FULL\_INT\_RAW 中断。(只写)

**UART TXFIFO EMPTY INT CLR** 置位此位清除 UART TXFIFO EMPTY INT RAW 中断。(只写)

**UART PARITY ERR INT CLR** 置位此位清除 UART PARITY ERR INT RAW 中断。(只写)

**UART\_FRM\_ERR\_INT\_CLR** 置位此位清除 UART\_FRM\_ERR\_INT\_RAW 中断。(只写)

**UART\_RXFIFO\_OVF\_INT\_CLR** 置位此位清除 UART\_RXFIFO\_OVF\_INT\_RAW 中断。(只写)

**UART DSR CHG INT CLR** 置位此位清除 UART DSR CHG INT RAW 中断。(只写)

**UART\_CTS\_CHG\_INT\_CLR** 置位此位清除 UART\_CTS\_CHG\_INT\_RAW 中断。(只写)

**UART\_BRK\_DET\_INT\_CLR** 置位此位清除 UART\_BRK\_DET\_INT\_RAW 中断。(只写)

**UART RXFIFO TOUT INT CLR** 置位此位清除 UART RXFIFO TOUT INT RAW 中断。(只写)

**UART\_SW\_XON\_INT\_CLR** 置位此位清除 UART\_SW\_XON\_INT\_RAW 中断。(只写)

**UART\_SW\_XOFF\_INT\_CLR** 置位此位清除 UART\_SW\_XOFF\_INT\_RAW 中断。(只写)

**UART\_GLITCH\_DET\_INT\_CLR** 置位此位清除 UART\_GLITCH\_DET\_INT\_RAW 中断。(只写)

**UART\_TX\_BRK\_DONE\_INT\_CLR** 置位此位清除 UART\_TX\_BRK\_DONE\_INT\_RAW 中断。(只写)

**UART\_TX\_BRK\_IDLE\_DONE\_INT\_CLR** 置位此位清除 UART\_TX\_BRK\_IDLE\_DONE\_INT\_RAW 中断。(只写)

**UART\_TX\_DONE\_INT\_CLR** 置位此位清除 UART\_TX\_DONE\_INT\_RAW 中断。(只写)

**UART\_RS485\_PARITY\_ERR\_INT\_CLR** 置位此位清除 UART\_RS485\_PARITY\_ERR\_INT\_RAW 中断。(只写)

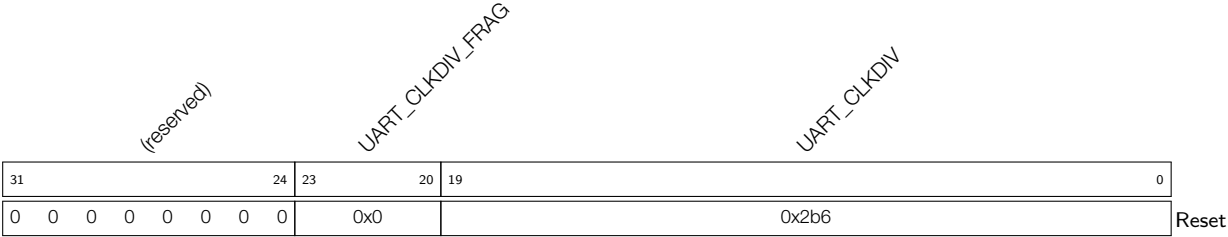
**UART\_RS485\_FRM\_ERR\_INT\_CLR** 置位此位清除 UART\_RS485\_FRM\_ERR\_INT\_RAW 中断。(只写)

**UART RS485 CLASH INT CLR** 置位此位清除 UART RS485 CLASH INT RAW 中断。(只写)

**UART\_AT\_CMD\_CHAR\_DET\_INT\_CLR** 置位此位清除 UART\_AT\_CMD\_CHAR\_DET\_INT\_RAW 中断。(只写)

**UART\_WAKEUP\_INT\_CLR** 置位此位清除 UART\_WAKEUP\_INT\_RAW 中断。(只写)

Register 8.7: UART\_CLKDIV\_REG (0x0014)



UART\_CLKDIV 分频系数的整数部分。(读/写)

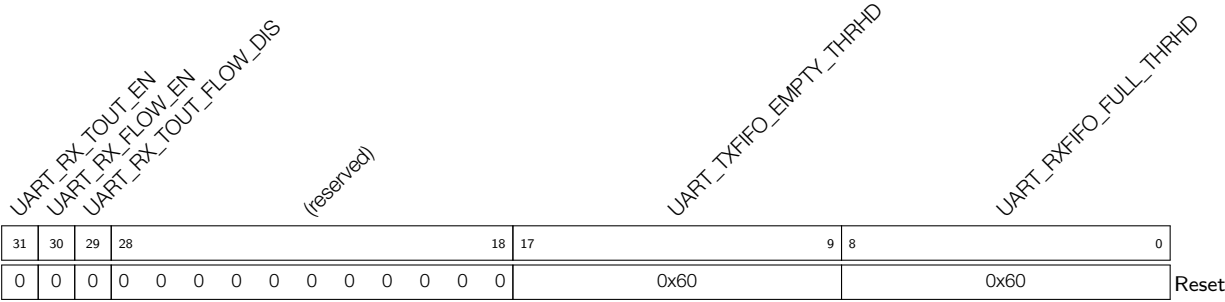
UART\_CLKDIV\_FRAG 分频系数的小数部分。(读/写)

Register 8.8: UART\_CONF0\_REG (0x0020)

(reserved)			UART_MEM_CLK_EN			UART_TICK_REF_ALWAYS_ON			(reserved)			UART_DTR_INV			UART_RTS_INV			UART_TXD_INV			UART_DSR_INV			UART_CTS_INV			UART_RXD_INV			UART_TXFIFO_RST			UART_RXFIFO_RST			UART_IRDA_EN			UART_LOOPBACK			UART_IRDA_RX_INV			UART_IRDA_TX_INV			UART_IRDA_WCTL			UART_IRDA_TX_EN			UART_TXD_DPLX			UART_SW_BRK			UART_SW_DTR			UART_STOP_BIT_NUM			UART_BIT_NUM			UART_PARITY_EN			UART_PARITY		
31	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	1	1	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。1'h0: 偶检验。1'h1: 奇检验。(读/写)
- UART\_PARITY\_EN** 置位此位使能 UART 奇偶检验。(读/写)
- UART\_BIT\_NUM** 设置数据长度。0: 5 位 1: 6 位 2: 7 位 3: 8 位。(读/写)
- UART\_STOP\_BIT\_NUM** 设置停止位的长度。1: 1 位 2: 1.5 位 3: 2 位 (读/写)
- UART\_SW\_RTS** 该寄存器用于配置软件流控使用的软件 RTS 信号。(读/写)
- UART\_SW\_DTR** 该寄存器用于配置软件流控使用的软件 DTR 信号。(读/写)
- UART\_TXD\_BRK** 置位此位，使能发送器在发完数据后发送 NULL。(读/写)
- UART\_IRDA\_DPLX** 置位此位开启 IrDA 回环测试模式。(读/写)
- UART\_IRDA\_TX\_EN** IrDA 发送器的启动使能位。(读/写)
- UART\_IRDA\_WCTL** 1'h1: IrDA 发送器的第 11 位与第 10 位相同。1'h0: 将 IrDA 发送器的第 11 位置 0。(读/写)
- UART\_IRDA\_TX\_INV** 置位此位翻转 IrDA 发送器的电平。(读/写)
- UART\_IRDA\_RX\_INV** 置位此位翻转 IrDA 接收器的电平。(读/写)
- UART\_LOOPBACK** 置位此位开启 UART 回环测试模式。(读/写)
- UART\_TX\_FLOW\_EN** 置位此位使能发送器的流控功能。(读/写)
- UART\_IRDA\_EN** 置位此位使能 IrDA 协议。(读/写)
- UART\_RXFIFO\_RST** 置位此位复位 UART RX FIFO。(读/写)
- UART\_TXFIFO\_RST** 置位此位复位 UART TX FIFO。(读/写)
- UART\_RXD\_INV** 置位此位翻转 UART RXD 信号电平。(读/写)
- UART\_CTS\_INV** 置位此位翻转 UART CTS 信号电平。(读/写)
- UART\_DSR\_INV** 置位此位翻转 UART DSR 信号电平。(读/写)
- UART\_TXD\_INV** 置位此位翻转 UART TXD 信号电平。(读/写)
- UART\_RTS\_INV** 置位此位翻转 UART RTS 信号电平。(读/写)
- UART\_DTR\_INV** 置位此位翻转 UART DTR 信号电平。(读/写)
- UART\_TICK\_REF\_ALWAYS\_ON** 该寄存器用于选取时钟。1'h1: APB\_CLK。1'h0: REF\_TICK。(读/写)

Register 8.9: UART\_CONF1\_REG (0x0024)



**UART\_RXFIFO\_FULL\_THRHD** 接收器接收数据多于该寄存器的值时产生 UART\_RXFIFO\_FULL\_INT 中断。(读/写)

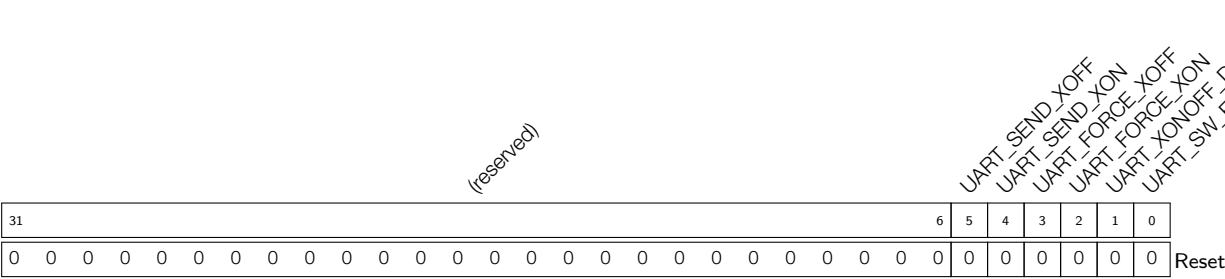
**UART\_TXFIFO\_EMPTY\_THRHD** TX FIFO 中的数据少于该寄存器的值时产生 UART\_TXFIFO\_EMPTY\_INT 中断。(读/写)

**UART\_RX\_TOUT\_FLOW\_DIS** 使用硬件流控时置位此位停止堆积 idle\_cnt。(读/写)

**UART\_RX\_FLOW\_EN** UART 接收器流控功能的使能位。1'h1: 配置 sw\_rts 信号选择软件流控。1'h0: 关闭软件流控。(读/写)

**UART\_RX\_TOUT\_EN** UART 接收器超时功能的使能位。(读/写)

Register 8.10: UART\_FLOW\_CONF\_REG (0x0034)



**UART\_SW\_FLOW\_CON\_EN** 置位此位使能软件流控，与 SW\_XON 或 SW\_XOFF 寄存器一起使用。(读/写)

**UART\_XONOFF\_DEL** 置位此位移除接收数据中的流控字符。(读/写)

**UART\_FORCE\_XON** 置位此位让发送器继续发送数据。(读/写)

**UART\_FORCE\_XOFF** 置位此位让发送器停止发送数据。(读/写)

**UART\_SEND\_XON** 置位此位发送 XON 字符。此位由硬件自动清除。(读/写)

**UART\_SEND\_XOFF** 置位此位发送 XOFF 字符。此位由硬件自动清除。(读/写)

Register 8.11: UART\_SLEEP\_CONF\_REG (0x0038)

(reserved)										UART_ACTIVE_THRESHOLD																			
31										9										0									
0 0										0xf0										Reset									

**UART\_ACTIVE\_THRESHOLD** 输入 RXD 沿变化次数超过该寄存器的值时，UART 从 Light-sleep 模式唤醒。(读/写)

Register 8.12: UART\_SWFC\_CONF0\_REG (0x003C)

(reserved)																UART_XOFF_CHAR								UART_XOFF_THRESHOLD																															
31																17								16								9								8								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x13								0xe0								Reset																							

**UART\_XOFF\_THRESHOLD** RX FIFO 中的数据超过该寄存器的值且 UART\_SW\_FLOW\_CON\_EN 置 1 时，发送 XOFF 字符。(读/写)

**UART\_XOFF\_CHAR** 存储 XOFF 流控字符。(读/写)

Register 8.13: UART\_SWFC\_CONF1\_REG (0x0040)

(reserved)															UART_XON_CHAR								UART_XON_THRESHOLD																																						
31															17															16								9								8								0							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0															0x11															0x0															Reset																

**UART\_XON\_THRESHOLD** RX FIFO 中的数据小于该寄存器的值且 UART\_SW\_FLOW\_CON\_EN 置 1 时，发送 XON 字符。(读/写)

**UART\_XON\_CHAR** 存储 XON 流控字符。(读/写)

### Register 8.14: UART\_IDLE\_CONF\_REG (0x0044)

31	28	27	20	19	10	9	0
(reserved)		UART_TX_BRK_NUM			UART_TX_IDLE_NUM		UART_RX_IDLE_THRHD
0 0 0 0		0xa			0x100		0x100
							Reset

**UART\_RX\_IDLE\_THRHD** 接收器接收一字节数据所需时间超过该寄存器的值时产生帧结束信号。  
(读/写)

**UART\_TX\_IDLE\_NUM** 配置两次数据传输的间隔时间。(读/写)

**UART\_TX\_BRK\_NUM** 配置数据发完后待发 NULL 字符的数量。UART\_TXD\_BRK 置 1 时有意义。  
(读/写)

### Register 8.15: UART\_RS485\_CONF\_REG (0x0048)

[illegible]

**UART\_RS485\_EN** 置位此位选择 RS485 模式。(读/写)

**UART\_DL0\_EN** 置位此位，延迟停止位 1 位。(读/写)

**UART\_DL1\_EN** 置位此位，延迟停止位 1 位。(读/写)

**UART\_RS485TX\_RX\_EN** 发送器在 RS485 模式下发送数据时，置位此位使能接收器接收数据。  
(读/写)

**UART\_RS485RXBY\_TX\_EN** 1'h1: RS485 接收器线路繁忙时使能 RS485 发送器发送数据。1' h0: RS485 接收器线路繁忙时 RS485 发送器不发送数据。(读/写)

**UART\_RS485\_RX\_DLY\_NUM** 延迟接收器的内部数据信号。(读/写)

**UART\_RS485\_TX\_DLY\_NUM** 延迟发送器的内部数据信号。(读/写)



Register 8.16: UART\_AUTOBAUD\_REG (0x0018)

(reserved)																UART_GLITCH_FILT								(reserved)								UART_AUTOBAUD_EN																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
31																16																15								8								7								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</							

**UART\_AUTOBAUD\_EN** 波特率检测的使能位。(读/写)

**UART\_GLITCH\_FILT** 宽度小于该寄存器值的输入脉冲会被忽略。该寄存器用于自动波特率检测。  
(读/写)

Register 8.17: UART\_LOWPULSE\_REG (0x0028)

(reserved)												UART_LOWPULSE_MIN_CNT																																									
31												20												19																		0											
0 0 0 0 0 0 0 0 0 0 0 0												0xffff																		Reset																							

**UART\_LOWPULSE\_MIN\_CNT** 存储低电平脉冲的最短持续时间，用于波特率检测。(只读)

Register 8.18: UART\_HIGHPULSE\_REG (0x002C)

(reserved)												UART_HIGHPULSE_MIN_CNT																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
31												20												19																			0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												0												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\_HIGHPULSE\_MIN\_CNT** 存储最长高电平脉冲持续时间。用于波特率检测。(只读)

Register 8.19: UART\_RXD\_CNT\_REG (0x0030)

(reserved)										UART_RXD_EDGE_CNT																			
31										9										0									
0 0										0x0										Reset									

UART\_RXD\_EDGE\_CNT 存储 RXD 沿变化的次数。用于波特率检测。(只读)

Register 8.20: UART\_POSPULSE\_REG (0x006C)

(reserved)												UART_POSEDGE_MIN_CNT																															
31												20												19																		0	
0 0 0 0 0 0 0 0 0 0 0 0												0xffff																		Reset													

UART\_POSEDGE\_MIN\_CNT 存储两个上升沿之间的最小输入时钟计数值。用于波特率检测。(只读)

Register 8.21: UART\_NEGPULSE\_REG (0x0070)

(reserved)												UART_NEGEDGE_MIN_CNT																																											
31												20												19																0															
0 0 0 0 0 0 0 0 0 0 0 0												0xffff																Reset																											

UART\_NEGEDGE\_MIN\_CNT 存储两个下降沿之间的最小输入时钟计数值。用于波特率检测。(只读)

Register 8.22: UART\_STATUS\_REG (0x001C)

UART_TXD UART_RTSN UART_DTRN (reserved)						UART_TXFIFO_CNT										UART_RXD UART_CTSN UART_DSRN (reserved)						UART_RXFIFO_CNT											
31	30	29	28	26	25											16	15	14	13	12	10	9											0
0x0	0	0	0	0	0	0x0										0	0	0	0	0	0	0x0										Reset	

UART\_RXFIFO\_CNT 存储 RX FIFO 中有效数据的字节数。(只读)

UART\_DSRN 该寄存器表示内部 UART DSR 信号的电平值。(只读)

UART\_CTSN 该寄存器表示内部 UART CTS 信号的电平值。(只读)

UART\_RXD 该寄存器表示内部 UART RXD 信号的电平值。(只读)

UART\_TXFIFO\_CNT 存储 TX FIFO 中数据的字节数。(只读)

UART\_DTRN 此位表示内部 UART DTR 信号的电平。(只读)

UART\_RTSN 此位表示内部 UART RTS 信号的电平。(只读)

UART\_TXD 此位表示内部 UART TXD 信号的电平。(只读)

Register 8.23: UART\_MEM\_TX\_STATUS\_REG (0x0060)

(reserved)												UART_TX_RADDR								(reserved)				UART_APB_TX_WADDR							
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

UART\_APB\_TX\_WADDR 在软件通过 APB 总线写 TX FIFO 时存储 TX FIFO 的偏移地址。(只读)

UART\_TX\_RADDR 在 TX FSM 通过 Tx\_FIFO\_Ctrl 读取数据时存储 TX FIFO 的偏移地址。(只读)

Register 8.24: UART\_MEM\_RX\_STATUS\_REG (0x0064)

(reserved)												UART_RX_WADDR										(reserved)										UART_APB_RX_RADDR									
31											21	20											11	10	9											0					
0	0	0	0	0	0	0	0	0	0	0	0	0x0										0	0x0										Reset								

**UART\_APB\_RX\_RADDR** 在软件通过 APB 总线读取 RX FIFO 数据时存储 RX FIFO 的偏移地址。(只读)

**UART\_RX\_WADDR** 在 Rx\_FIFO\_Ctrl 写 RX FIFO 时存储 RX FIFO 的偏移地址。(只读)

Register 8.25: UART\_FSM\_STATUS\_REG (0x0068)

(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	0x0	0x0	Reset						

**UART\_ST\_URX\_OUT** 接收器的状态寄存器。(只读)

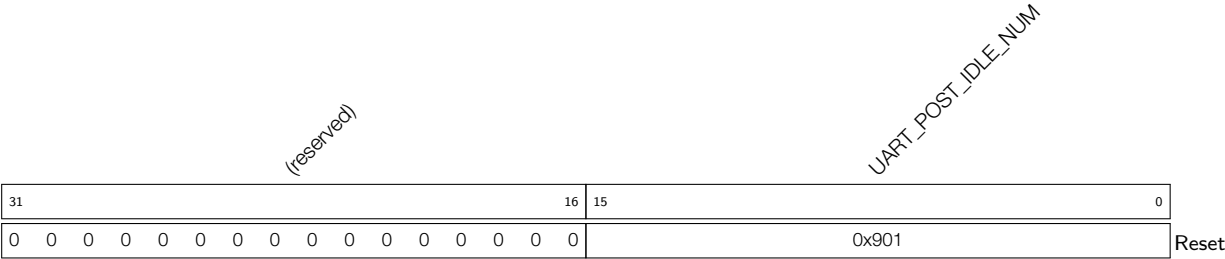
**UART\_ST\_UTX\_OUT** 发送器的状态寄存器。(只读)

Register 8.26: UART\_AT\_CMD\_PRECNT\_REG (0x004C)

(reserved)																UART_PRE_IDLE_NUM															
31											16	15																			0
0	0	0	0	0	0	0	0	0	0	0	0	0	0x901																		Reset

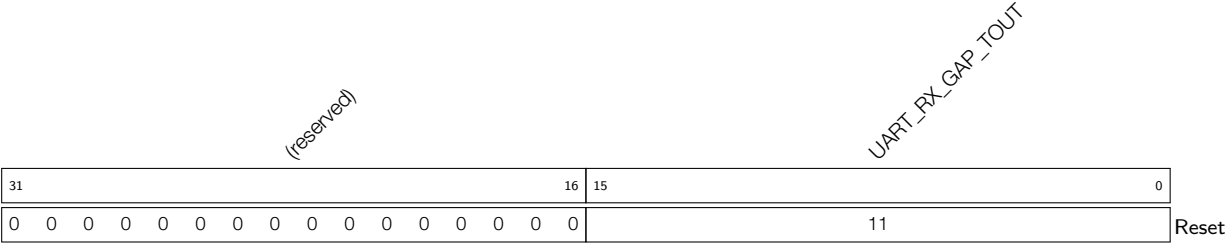
**UART\_PRE\_IDLE\_NUM** 配置接收器接收第一个 AT\_CMD 字符前的空闲时间。间隔时间小于该寄存器的值时，下一个接收数据不会被视作 AT\_CMD 字符。(读/写)

Register 8.27: UART\_AT\_CMD\_POSTCNT\_REG (0x0050)



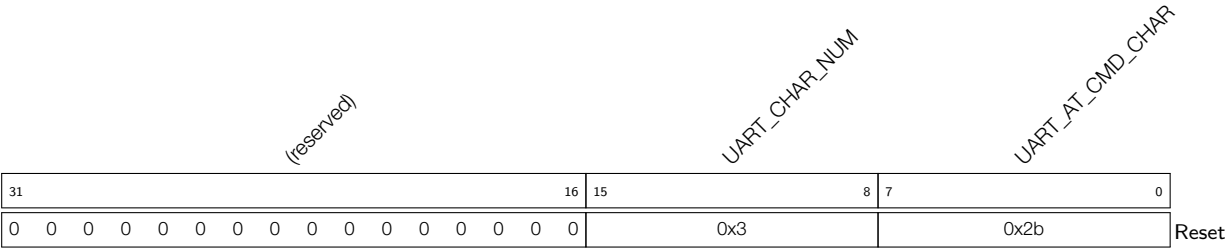
**UART\_POST\_IDLE\_NUM** 配置最后一个 AT\_CMD 字符和后续数据的间隔时间。间隔时间小于该寄存器的值时，前一个数据不会被视作 AT\_CMD 字符。(读/写)

Register 8.28: UART\_AT\_CMD\_GAPTOUT\_REG (0x0054)



**UART\_RX\_GAP\_TOUT** 配置 AT\_CMD 字符的间隔时间。持续/间隔时间小于该寄存器的值时，数据不会被视作连续的 AT\_CMD 字符。(读/写)

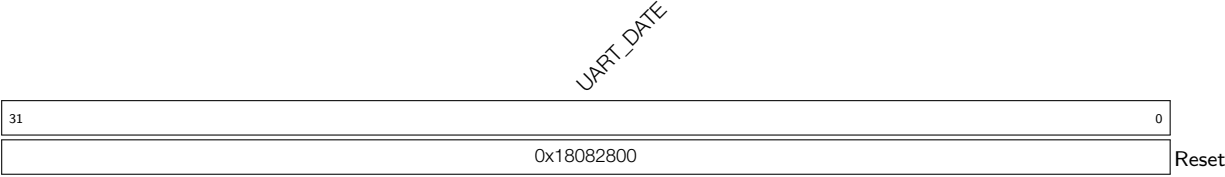
Register 8.29: UART\_AT\_CMD\_CHAR\_REG (0x0058)



**UART\_AT\_CMD\_CHAR** 配置 AT\_CMD 字符的内容。(读/写)

**UART\_CHAR\_NUM** 配置接收器接收连续 AT\_CMD 字符的个数。(读/写)

Register 8.30: UART\_DATE\_REG (0x0074)



**UART\_DATE** 版本控制寄存器。(读/写)

Register 8.31: UHCI\_CONF0\_REG (0x0000)

(reserved)								UHCI_UART_RX_BRK_EOF_EN								UHCI_UART0_CE								UHCI_OUT_NO_RESTART_CLR							
UHCI_LEN_EOF_EN								UHCI_UART1_CE								UHCI_OUT_AUTO_WBACK								UHCI_AHBM_RST							
UHCI_CRC_REC_EN								UHCI_OUT_LOOP_TEST								UHCI_IN_LOOP_TEST								UHCI_OUT_RST							
UHCI_UART_IDLE_EOF_EN								UHCI_OUT_EOF_MODE								UHCI_AHBM_FIFO_RST								UHCI_IN_RST							
UHCI_HEAD_EN								(reserved)								Reset															
UHCI_SEPER_EN								UHCI_OUT_DATA_BURST_EN																							
UHCI_MEM_TRANS_EN								UHCI_OUTDSCR_BURST_EN																							
UHCI_OUT_DATA_BURST_EN																															
UHCI_INDSCR_BURST_EN																															
UHCI_OUT_EOF_INT																															
UHCI_UART0_CE																															
UHCI_UART1_CE																															
UHCI_OUTDSCR_BURST_EN																															
UHCI_INDSCR_BURST_EN																															
UHCI_OUT_DATA_BURST_EN																															
UHCI_MEM_TRANS_EN																															
UHCI_SEPER_EN																															
UHCI_HEAD_EN																															
UHCI_CRC_REC_EN																															
UHCI_UART_IDLE_EOF_EN																															
UHCI_LEN_EOF_EN																															
UHCI_RX_BRK_EOF_EN																															
UHCI_TX_BRK_EOF_EN																															

**UHCI\_IN\_RST** 置位此位，复位接收 DMA FSM。(读/写)

**UHCI\_OUT\_RST** 置位此位，复位发送 DMA FSM。(读/写)

**UHCI\_AHBM\_FIFO\_RST** 置位此位，复位 DMA cmdFIFO 的 AHB 接口。(读/写)

**UHCI\_AHBM\_RST** 置位此位，复位 DMA 的 AHB 接口。(读/写)

**UHCI\_IN\_LOOP\_TEST** 保留。(读/写)

**UHCI\_OUT\_LOOP\_TEST** 保留。(读/写)

**UHCI\_OUT\_AUTO\_WBACK** 置位此位，在 TX Buffer 中所有数据传输完毕后自动回写发送链表。  
(读/写)

**UHCI\_OUT\_NO\_RESTART\_CLR** (读/写)

**UHCI\_OUT\_EOF\_MODE** 用于选择 UHCI\_OUT\_EOF\_INT 的产生条件。1: DMA 将所有数据从 FIFO 中弹出时产生中断。0: AHB 将所有数据推送到 FIFO 时产生中断。(读/写)

**UHCI\_UART0\_CE** 置位此位，将 HCI 和 UART0 相连。(读/写)

**UHCI\_UART1\_CE** 置位此位，将 HCI 和 UART1 相连。(读/写)

**UHCI\_OUTDSCR\_BURST\_EN** 用于选择 DMA 发送链表描述符传输模式。1: 突发模式。0: 字节模式。  
(读/写)

**UHCI\_INDSCR\_BURST\_EN** 用于选择 DMA 接收链表描述符传输模式。1: 突发模式。0: 字节模式。  
(读/写)

**UHCI\_OUT\_DATA\_BURST\_EN** 用于选择数据传输模式。1: 突发模式传输数据。0: 字节模式传输数据。(读/写)

**UHCI\_MEM\_TRANS\_EN** 1: UHCI 发送数据写回 DMA INFIFO。(读/写)

**UHCI\_SEPER\_EN** 置位此位，使用特殊字符分隔数据帧。(读/写)

**UHCI\_HEAD\_EN** 置位此位，用格式报头编码数据包。(读/写)

**UHCI\_CRC\_REC\_EN** 置位此位，使能 UHCI 接收 16 位 CRC。(读/写)

**UHCI\_UART\_IDLE\_EOF\_EN** 若此位置 1，UHCI 在 UART 空闲时停止接收有效载荷。(读/写)

**UHCI\_LEN\_EOF\_EN** 若此位置 1，UHCI 解码器接收字节数达到指定值时停止接收有效载荷数据。  
UHCI\_HEAD\_EN 为 1 时，该值是 UCHI 数据包报头明确的有效负载长度；UHCI\_HEAD\_EN 为 0 时，该值为配置值。若此位置 0，UHCI 解码器在接收到 0xc0 后停止接收有效载荷数据。(读/写)

Register 8.31: UHCI\_CONF0\_REG (0x0000)

接上页...

**UHCI\_ENCODE\_CRC\_EN** 置位此位，在有效载荷末尾加 16 位 CCITT-CRC 开始数据完整性检测。  
(读/写)

**UHCI\_CLK\_EN** 1'b1：强制为寄存器开启时钟。1'b0：仅在应用写寄存器时支持时钟。(读/写)

**UHCI\_UART\_RX\_BRK\_EOF\_EN** 若此位置 1，UART 收到 NULL 帧后 UHCI 结束 payload\_rec。(读/写)

Register 8.32: UHCI\_CONF1\_REG (0x002C)

(reserved)												UHCI_DMA_INFIFO_FULL_THRS												UHCI_SW_START UHCI_WAIT_SW_START UHCI_TX_CHECK_OWNER UHCI_TX_ACK_NUM_RE UHCI_SAVE_HEAD UHCI_CRC_DISABLE UHCI_CHECK_SEQ_EN UHCI_CHECK_SUM_EN							
31											21	20									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	1	1	0	0	1	1			

**UHCI\_CHECK\_SUM\_EN** UHCI 接收数据包时检查报头校验和的使能位。(读/写)

**UHCI\_CHECK\_SEQ\_EN** UHCI 接收数据包时检查序列号的使能位。(读/写)

**UHCI\_CRC\_DISABLE** 置位此位，支持 CRC 计算。UHCI 包中的数据完整性检测位应为 1。(读/写)

**UHCI\_SAVE\_HEAD** 置位此位，在 HCI 接收数据包时保存数据包报头。(读/写)

**UHCI\_TX\_CHECK\_SUM\_RE** 置位此位，用校验和编码数据包。(读/写)

**UHCI\_TX\_ACK\_NUM\_RE** 准备发送可靠数据包时，置位此位用 ACK 编码该数据包。(读/写)

**UHCI\_CHECK\_OWNER** 1：链表操作方为 DMA 控制器时检查链表描述符；0：总是检查链表描述符。(读/写)

**UHCI\_WAIT\_SW\_START** 此寄存器置 1 时，UHCI 编码器跳至 ST\_SW\_WAIT 状态。(读/写)

**UHCI\_SW\_START** 若当前 UHCI\_ENCODE\_STATE 为 ST\_SW\_WAIT 状态，此位置 1 时 UHCI 开始发送数据包。(读/写)

**UHCI\_DMA\_INFIFO\_FULL\_THRS** 接收 FIFO 的计数器值超过该寄存器的值时产生 UHCI\_DMA\_INFIFO\_FULL\_WM\_INT 中断。(读/写)



Register 8.33: UHCI\_AHB\_TEST\_REG (0x0048)

(reserved)																										UHCl_AHB_TESTADDR		(reserved)		UHCl_AHB_TESTMODE	
31																										6	5	4	3	2	0
0 0																										0	0	0		Reset	

UHCI\_AHB\_TESTMODE 保留。(读/写)

UHCI\_AHB\_TESTADDR 保留。(读/写)

Register 8.34: UHCI\_ESCAPE\_CONF\_REG (0x0064)

(reserved)																								UHCL_RX_13_ESC_EN								UHCL_RX_11_ESC_EN								UHCL_RX_DB_ESC_EN								UHCL_RX_C0_ESC_EN								UHCL_TX_13_ESC_EN								UHCL_TX_11_ESC_EN								UHCL_TX_DB_ESC_EN								UHCL_TX_C0_ESC_EN							
31																								8		7		6		5		4		3		2		1		0																																															
0																								0		0		1		1		0		0		1		1		Reset																																															

UHCI\_TX\_C0\_ESC\_EN 置位此位，在 DMA 接收数据时解析字符 0xc0。(读/写)

UHCI\_TX\_DB\_ESC\_EN 置位此位，在 DMA 接收数据时解析字符 0xdb。(读/写)

UHCI\_TX\_11\_ESC\_EN 置位此位，在 DMA 接收数据时解析流控字符 0x11。(读/写)

UHCI\_TX\_13\_ESC\_EN 置位此位，在 DMA 接收数据时解析流控字符 0x13。(读/写)

UHCI\_RX\_C0\_ESC\_EN 置位此位，在 DMA 发送数据时用特殊字符替换流控字符 0xc0。(读/写)

UHCI\_RX\_DB\_ESC\_EN 置位此位，在 DMA 发送数据时用特殊字符替换流控字符 0xdb。(读/写)

UHCI\_RX\_11\_ESC\_EN 置位此位，在 DMA 发送数据时用特殊字符替换流控字符 0x11。(读/写)

UHCI\_RX\_13\_ESC\_EN 置位此位，在 DMA 发送数据时用特殊字符替换流控字符 0x13。(读/写)

Register 8.35: UHCI\_HUNG\_CONF\_REG (0x0068)

(reserved)								UHCL_RXFIFO_TIMEOUT_ENA				UHCL_RXFIFO_TIMEOUT_SHIFT				UHCL_RXFIFO_TIMEOUT				UHCL_TXFIFO_TIMEOUT_ENA				UHCL_TXFIFO_TIMEOUT_SHIFT				UHCL_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** 存储超时值。DMA 接收数据时间过长时产生 UHCI\_TX\_HUNG\_INT 中断。  
(读/写)

**UHCI\_TXFIFO\_TIMEOUT\_SHIFT** 配置计数器的最大值。(读/写)

**UHCI\_TXFIFO\_TIMEOUT\_ENA** TX FIFO 接收数据超时使能位。(读/写)

**UHCI\_RXFIFO\_TIMEOUT** 存储超时值。DMA 读取 RAM 数据时间过长时产生 UHCI\_RX\_HUNG\_INT 中断。(读/写)

**UHCI\_RXFIFO\_TIMEOUT\_SHIFT** 配置计数器的最大值。(读/写)

**UHCI\_RXFIFO\_TIMEOUT\_ENA** DMA 发送数据超时的使能位。(读/写)

Register 8.36: UHCI\_QUICK\_SENT\_REG (0x0074)

(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																Reset
0 0																												0	0x0		0		0x0																		

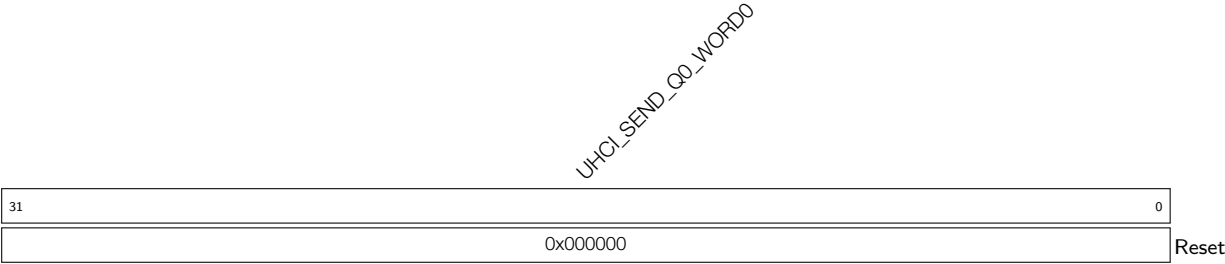
**UHCI\_SINGLE\_SEND\_NUM** 设定 single\_send 寄存器。(读/写)

**UHCI\_SINGLE\_SEND\_EN** 置位此位使能 single\_send 模式发送短包。(读/写)

**UHCI\_ALWAYS\_SEND\_NUM** 设定 always\_send 寄存器。(读/写)

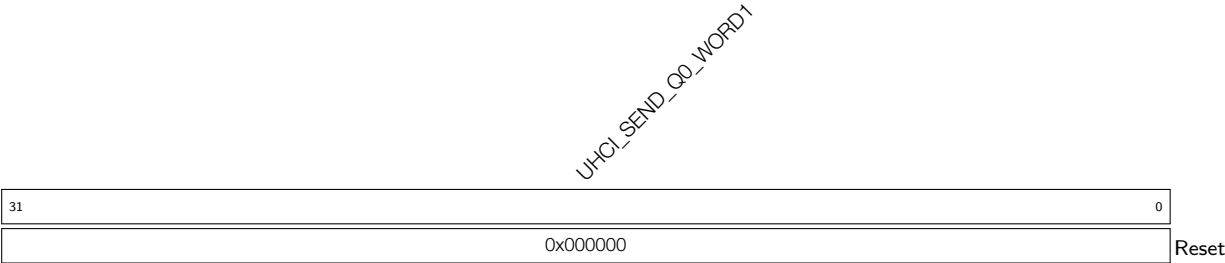
**UHCI\_ALWAYS\_SEND\_EN** 置位此位使能 always\_send 模式发送短包。(读/写)

Register 8.37: UHCI\_REG\_Q0\_WORD0\_REG (0x0078)



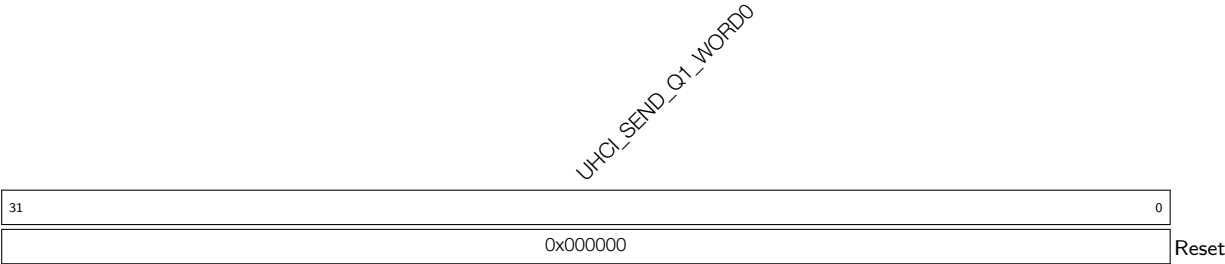
**UHCI\_SEND\_Q0\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.38: UHCI\_REG\_Q0\_WORD1\_REG (0x007C)



**UHCI\_SEND\_Q0\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.39: UHCI\_REG\_Q1\_WORD0\_REG (0x0080)



**UHCI\_SEND\_Q1\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.40: UHCI\_REG\_Q1\_WORD1\_REG (0x0084)

UHCI_SEND_Q1_WORD1	
31	0
0x000000	
	Reset

**UHCI\_SEND\_Q1\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.41: UHCI\_REG\_Q2\_WORD0\_REG (0x0088)

UHCI_SEND_Q2_WORD0	
31	0
0x000000	
	Reset

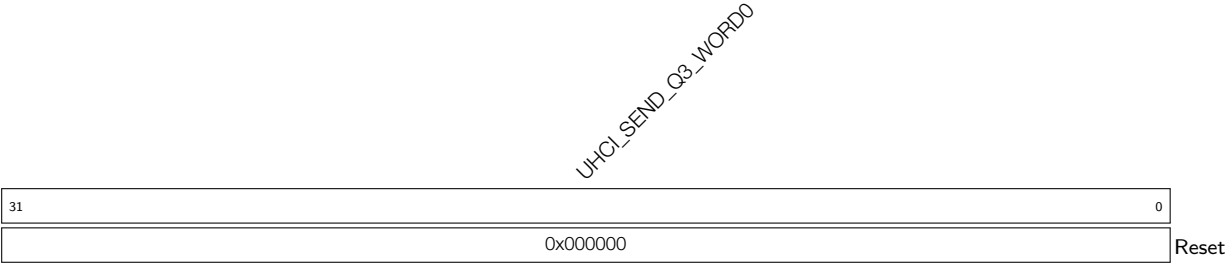
**UHCI\_SEND\_Q2\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.42: UHCI\_REG\_Q2\_WORD1\_REG (0x008C)

UHCI_SEND_Q2_WORD1	
31	0
0x000000	
	Reset

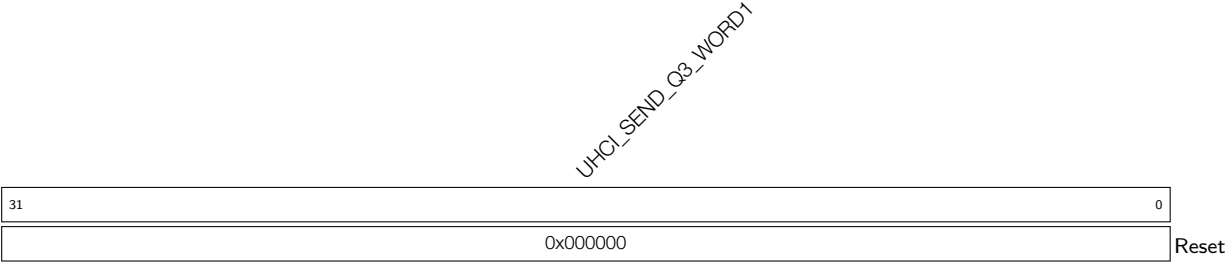
**UHCI\_SEND\_Q2\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.43: UHCI\_REG\_Q3\_WORD0\_REG (0x0090)



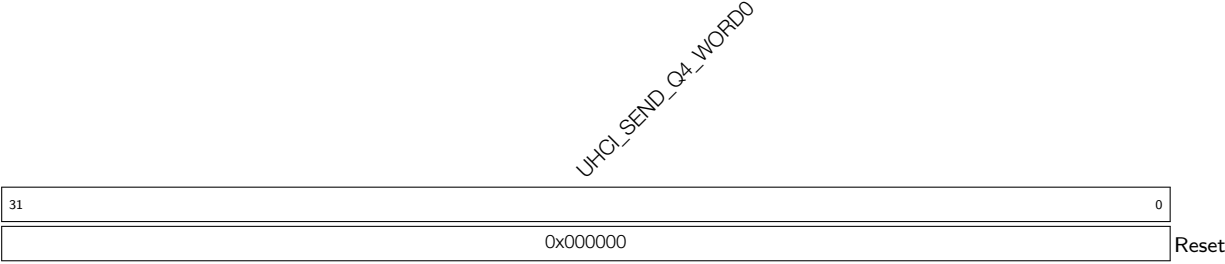
**UHCI\_SEND\_Q3\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.44: UHCI\_REG\_Q3\_WORD1\_REG (0x0094)



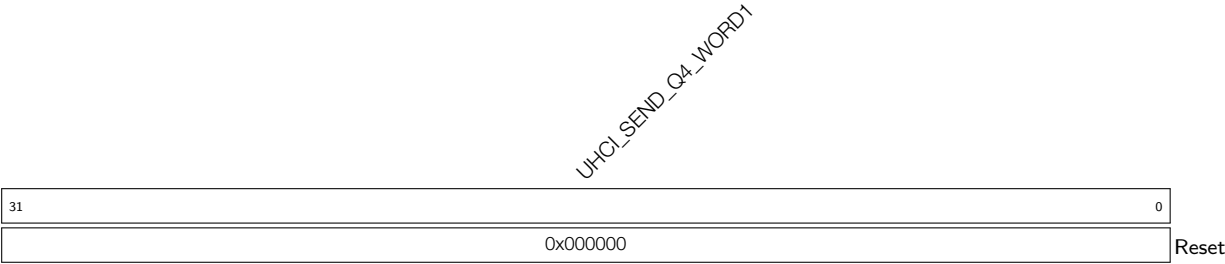
**UHCI\_SEND\_Q3\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.45: UHCI\_REG\_Q4\_WORD0\_REG (0x0098)



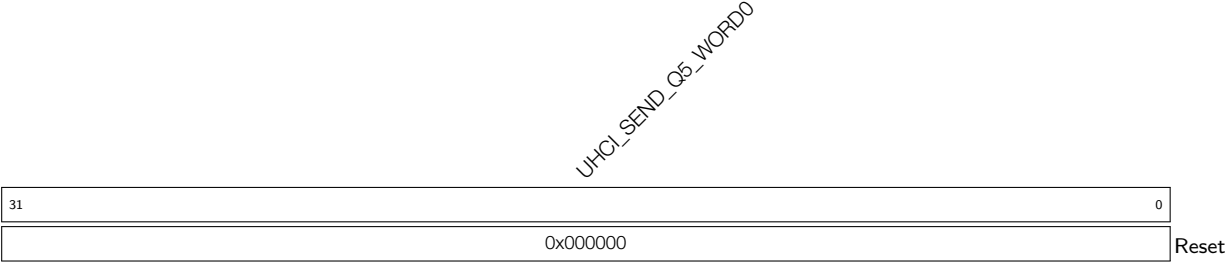
**UHCI\_SEND\_Q4\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.46: UHCI\_REG\_Q4\_WORD1\_REG (0x009C)



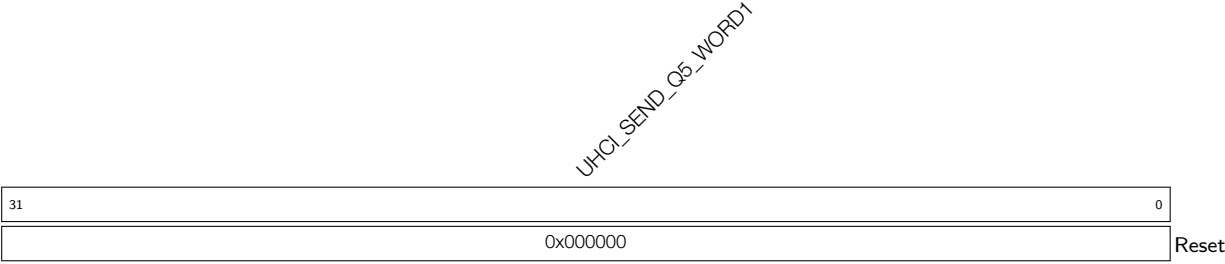
**UHCI\_SEND\_Q4\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.47: UHCI\_REG\_Q5\_WORD0\_REG (0x00A0)



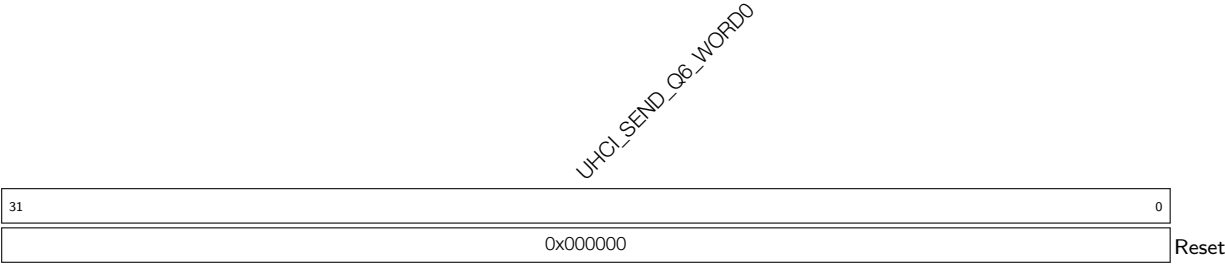
**UHCI\_SEND\_Q5\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.48: UHCI\_REG\_Q5\_WORD1\_REG (0x00A4)



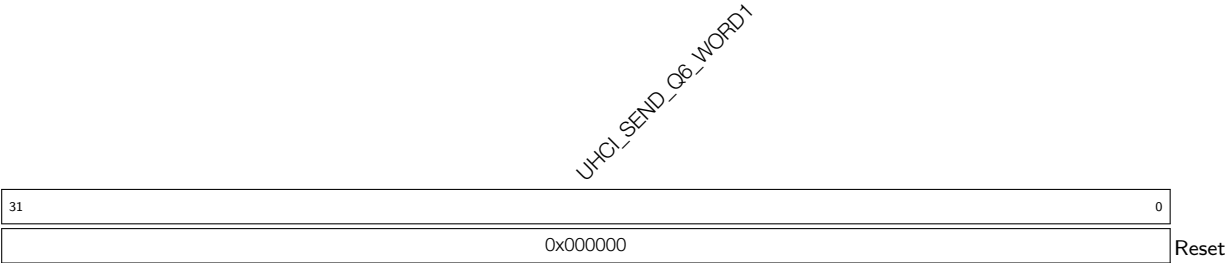
**UHCI\_SEND\_Q5\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.49: UHCI\_REG\_Q6\_WORD0\_REG (0x00A8)



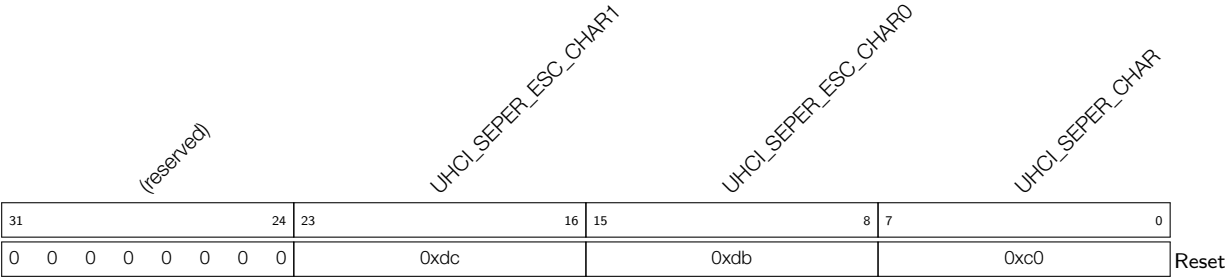
**UHCI\_SEND\_Q6\_WORD0** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.50: UHCI\_REG\_Q6\_WORD1\_REG (0x00AC)



**UHCI\_SEND\_Q6\_WORD1** 在 UHCI\_ALWAYS\_SEND\_NUM 或 UHCI\_SINGLE\_SEND\_NUM 指定时用作快速发送寄存器。(读/写)

Register 8.51: UHCI\_ESC\_CONF0\_REG (0x00B0)



- UHCI\_SEPER\_CHAR** 定义需编码的分隔符，默认为 0xc0。(读/写)
- UHCI\_SEPER\_ESC\_CHAR0** 编码分隔符时定义 Slip 转义序列的第一个字符，分隔符默认为 0xdb。(读/写)
- UHCI\_SEPER\_ESC\_CHAR1** 编码分隔符时定义 Slip 转义序列的第二个字符，分隔符默认为 0xdc。(读/写)

Register 8.52: UHCI\_ESC\_CONF1\_REG (0x00B4)

(reserved)								UHCI_ESC_SEQ_CHAR1								UHCI_ESC_SEQ_CHAR0								UHCI_ESC_SEQ0											
31								24	23								16	15								8	7								0
0	0	0	0	0	0	0	0	0xdd								0xdb								0xdb								Reset			

**UHCI\_ESC\_SEQ0** 定义需编码的字符，默认为用作 Slip 转义序列第一个字符的 0xdb。(读/写)

**UHCI\_ESC\_SEQ0\_CHAR0** 编码 UHCI\_ESC\_SEQ0 时定义 Slip 转义序列的第一个字符，UHCI\_ESC\_SEQ0 默认为 0xdb。(读/写)

**UHCI\_ESC\_SEQ0\_CHAR1** 编码 UHCI\_ESC\_SEQ0 时定义 Slip 转义序列的第二个字符，UHCI\_ESC\_SEQ0 默认为 0xdd。(读/写)

Register 8.53: UHCI\_ESC\_CONF2\_REG (0x00B8)

(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** 定义需编码的字符，默认为用作流控字符的 0x11。(读/写)

**UHCI\_ESC\_SEQ1\_CHAR0** 编码 UHCI\_ESC\_SEQ1 时定义 Slip 转义序列的第一个字符，UHCI\_ESC\_SEQ1 默认为 0xdb。(读/写)

**UHCI\_ESC\_SEQ1\_CHAR1** 编码 UHCI\_ESC\_SEQ1 时定义 Slip 转义序列的第二个字符，UHCI\_ESC\_SEQ1 默认为 0xde。(读/写)



Register 8.54: UHCI\_ESC\_CONF3\_REG (0x00BC)

(reserved)								UHCL_ESC_SEQ2_CHAR1								UHCL_ESC_SEQ2_CHAR0								UHCL_ESC_SEQ2								
31								24	23							16	15							8	7							0
0	0	0	0	0	0	0	0	0xdf								0xdb								0x13								Reset

- UHCI\_ESC\_SEQ2** 定义需编码的字符，默认为用作流控字符的 0x13。(读/写)
- UHCI\_ESC\_SEQ2\_CHAR0** 编码 UHCI\_ESC\_SEQ2 时定义 Slip 转义序列的第一个字符，UHCI\_ESC\_SEQ2 默认为 0xdb。(读/写)
- UHCI\_ESC\_SEQ2\_CHAR1** 编码 UHCI\_ESC\_SEQ2 时定义 Slip 转义序列的第二个字符，UHCI\_ESC\_SEQ2 默认为 0xdf。(读/写)

Register 8.55: UHCI\_PKT\_THRES\_REG (0x00C0)

(reserved)																UHCI_PKT_THRS																							
31																12												0											
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x80												Reset											

- UHCI\_PKT\_THRS** UHCI\_HEAD\_EN 为 0 时配置包长度的最大值。(读/写)

Register 8.56: UHCI\_INT\_RAW\_REG (0x0004)

(reserved)																																UHCI_DMA_INFIFO_FULL_WM_INT_RAW UHCI_SEND_A_REG_Q_INT_RAW UHCI_SEND_S_REG_Q_INT_RAW UHCI_OUT_TOTAL_EOF_INT_RAW UHCI_OUTLINK_EOF_INT_RAW UHCI_IN_DSCR_ERR_INT_RAW UHCI_OUT_DSCR_ERR_INT_RAW UHCI_OUT_EMPTY_INT_RAW UHCI_IN_DSCR_ERR_INT_RAW UHCI_OUT_EOF_INT_RAW UHCI_IN_DONE_INT_RAW UHCI_IN_SUC_EOF_INT_RAW UHCI_TX_HUNG_INT_RAW UHCI_RX_HUNG_INT_RAW UHCI_TX_START_INT_RAW UHCI_RX_START_INT_RAW																		
31																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	Reset

- UHCI\_RX\_START\_INT\_RAW** 原始中断位。分隔符成功发送时触发此中断。(只读)
- UHCI\_TX\_START\_INT\_RAW** 原始中断位。DMA 检测到分隔符时触发此中断。(只读)
- UHCI\_RX\_HUNG\_INT\_RAW** 原始中断位。DMA 接收数据所需时间超过配置值时触发此中断。(只读)
- UHCI\_TX\_HUNG\_INT\_RAW** 原始中断位。DMA 读取 RAM 数据所需时间超过配置值时触发此中断。(只读)
- UHCI\_IN\_DONE\_INT\_RAW** 原始中断位。接收链表描述符完成时触发此中断。(只读)
- UHCI\_IN\_SUC\_EOF\_INT\_RAW** 原始中断位。成功接收数据包时触发此中断。(只读)
- UHCI\_IN\_ERR\_EOF\_INT\_RAW** 原始中断位。接收链表描述符的 EOF 有错误时触发此中断。(只读)
- UHCI\_OUT\_DONE\_INT\_RAW** 原始中断位。发送链表描述符完成时触发此中断。(只读)
- UHCI\_OUT\_EOF\_INT\_RAW** 原始中断位。当前描述符的 EOF 有效时触发此中断。(只读)
- UHCI\_IN\_DSCR\_ERR\_INT\_RAW** 原始中断位。接收链表描述符存在错误时触发此中断。(只读)
- UHCI\_OUT\_DSCR\_ERR\_INT\_RAW** 原始中断位。发送链表描述符存在错误时触发此中断。(只读)
- UHCI\_IN\_DSCR\_EMPTY\_INT\_RAW** 原始中断位。DMA 没有足够的接收链表时触发此中断。(只读)
- UHCI\_OUTLINK\_EOF\_ERR\_INT\_RAW** 原始中断位。发送链表描述符的 EOF 有错误时触发此中断。(只读)
- UHCI\_OUT\_TOTAL\_EOF\_INT\_RAW** 原始中断位。最后一个缓存区中的所有数据发送完毕时触发此中断。(只读)
- UHCI\_SEND\_S\_REG\_Q\_INT\_RAW** 原始中断位。DMA 使用 single\_send 寄存器成功发送短包时触发此中断。(只读)
- UHCI\_SEND\_A\_REG\_Q\_INT\_RAW** 原始中断位。DMA 使用 always\_send 寄存器成功发送短包时触发此中断。(只读)
- UHCI\_DMA\_INFIFO\_FULL\_WM\_INT\_RAW** 原始中断位。DMA INFIFO 存储量达到配置阈值时触发此中断。(只读)

Register 8.57: UHCI\_INT\_ST\_REG (0x0008)

(reserved)																	UHCI_DMA_INFO_FULL_WM_INT_ST	UHCI_SEND_A_REG_Q_INT_ST	UHCI_SEND_S_REG_Q_INT_ST	UHCI_OUT_TOTAL_EOF_INT_ST	UHCI_OUTLINK_EOF_ERR_INT_ST	UHCI_IN_DSCR_ERR_INT_ST	UHCI_OUT_DSCR_EMPTY_INT_ST	UHCI_OUT_EOF_ERR_INT_ST	UHCI_IN_DONE_INT_ST	UHCI_IN_SUC_EOF_INT_ST	UHCI_TX_HUNG_INT_ST	UHCI_RX_HUNG_INT_ST	UHCI_TX_START_INT_ST	UHCI_RX_START_INT_ST				
31																	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	Reset							

**UHCI\_RX\_START\_INT\_ST** UHCI\_RX\_START\_INT\_ENA 置 1 时 UHCI\_RX\_START\_INT 中断的屏蔽中断位。(只读)

**UHCI\_TX\_START\_INT\_ST** UHCI\_TX\_START\_INT\_ENA 置 1 时 UHCI\_TX\_START\_INT 中断的屏蔽中断位。(只读)

**UHCI\_RX\_HUNG\_INT\_ST** UHCI\_RX\_HUNG\_INT\_ENA 置 1 时 UHCI\_RX\_HUNG\_INT 中断的屏蔽中断位。(只读)

**UHCI\_TX\_HUNG\_INT\_ST** UHCI\_TX\_HUNG\_INT\_ENA 置 1 时 UHCI\_TX\_HUNG\_INT 中断的屏蔽中断位。(只读)

**UHCI\_IN\_DONE\_INT\_ST** UHCI\_IN\_DONE\_INT\_ENA 置 1 时 UHCI\_IN\_DONE\_INT 中断的屏蔽中断位。(只读)

**UHCI\_IN\_SUC\_EOF\_INT\_ST** UHCI\_IN\_SUC\_EOF\_INT\_ENA 置 1 时 UHCI\_IN\_SUC\_EOF\_INT 中断的屏蔽中断位。(只读)

**UHCI\_IN\_ERR\_EOF\_INT\_ST** UHCI\_IN\_ERR\_EOF\_INT\_ENA 置 1 时 UHCI\_IN\_ERR\_EOF\_INT 中断的屏蔽中断位。(只读)

**UHCI\_OUT\_DONE\_INT\_ST** UHCI\_OUT\_DONE\_INT\_ENA 置 1 时 UHCI\_OUT\_DONE\_INT 中断的屏蔽中断位。(只读)

**UHCI\_OUT\_EOF\_INT\_ST** UHCI\_OUT\_EOF\_INT\_ENA 置 1 时 UHCI\_OUT\_EOF\_INT 中断的屏蔽中断位。(只读)

**UHCI\_IN\_DSCR\_ERR\_INT\_ST** UHCI\_IN\_DSCR\_ERR\_INT\_ENA 置 1 时 UHCI\_IN\_DSCR\_ERR\_INT 中断的屏蔽中断位。(只读)

**UHCI\_OUT\_DSCR\_ERR\_INT\_ST** UHCI\_OUT\_DSCR\_ERR\_INT\_ENA 置 1 时 UHCI\_OUT\_DSCR\_ERR\_INT 中断的屏蔽中断位。(只读)

**UHCI\_IN\_DSCR\_EMPTY\_INT\_ST** UHCI\_IN\_DSCR\_EMPTY\_INT\_ENA 置 1 时 UHCI\_IN\_DSCR\_EMPTY\_INT 中断的屏蔽中断位。(只读)

**UHCI\_OUTLINK\_EOF\_ERR\_INT\_ST** UHCI\_OUTLINK\_EOF\_ERR\_INT\_ENA 置 1 时 UHCI\_OUTLINK\_EOF\_ERR\_INT 中断的屏蔽中断位。(只读)

**UHCI\_OUT\_TOTAL\_EOF\_INT\_ST** UHCI\_OUT\_TOTAL\_EOF\_INT\_ENA 置 1 时 UHCI\_OUT\_TOTAL\_EOF\_INT 中断的屏蔽中断位。(只读)

见下页...

Register 8.57: UHCI\_INT\_ST\_REG (0x0008)

接上页...

UHCI_SEND_S_REG_Q_INT_ST	UHCI_SEND_S_REG_Q_INT_ENA	置	1	时
UHCI_SEND_S_REG_Q_INT 中断的屏蔽中断位。(只读)				
UHCI_SEND_A_REG_Q_INT_ST	UHCI_SEND_A_REG_Q_INT_ENA	置	1	时
UHCI_SEND_A_REG_Q_INT 中断的屏蔽中断位。(只读)				
UHCI_DMA_INFIFO_FULL_WM_INT_ST	UHCI_DMA_INFIFO_FULL_WM_INT_ENA	置	1	时
UHCI_DMA_INFIFO_FULL_WM_INT 中断的屏蔽中断位。(只读)				

Register 8.58: UHCI\_INT\_ENA\_REG (0x000C)

(reserved)																																UHCI_DMA_INFIFO_FULL_WM_INT_ENA UHCI_SEND_A_REG_Q_INT_ENA UHCI_SEND_S_REG_Q_INT_ENA UHCI_OUT_TOTAL_EOF_INT_ENA UHCI_OUTLINK_EOF_ERR_INT_ENA UHCI_IN_DSCR_EMPTY_INT_ENA UHCI_OUT_DSCR_ERR_INT_ENA UHCI_OUT_EOF_INT_ENA UHCI_IN_DSCR_EMPTY_INT_ENA UHCI_IN_ERR_EOF_INT_ENA UHCI_IN_DONE_INT_ENA UHCI_IN_SUC_EOF_INT_ENA UHCI_TX_HUNG_INT_ENA UHCI_RX_HUNG_INT_ENA UHCI_TX_START_INT_ENA UHCI_RX_START_INT_ENA																		
31																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	Reset

**UHCI\_RX\_START\_INT\_ENA** UHCI\_RX\_START\_INT 的中断使能位。(读/写)

**UHCI\_TX\_START\_INT\_ENA** UHCI\_TX\_START\_INT 的中断使能位。(读/写)

**UHCI\_RX\_HUNG\_INT\_ENA** UHCI\_RX\_HUNG\_INT 的中断使能位。(读/写)

**UHCI\_TX\_HUNG\_INT\_ENA** UHCI\_TX\_HUNG\_INT 的中断使能位。(读/写)

**UHCI\_IN\_DONE\_INT\_ENA** UHCI\_IN\_DONE\_INT 的中断使能位。(读/写)

**UHCI\_IN\_SUC\_EOF\_INT\_ENA** UHCI\_IN\_SUC\_EOF\_INT 的中断使能位。(读/写)

**UHCI\_IN\_ERR\_EOF\_INT\_ENA** UHCI\_IN\_ERR\_EOF\_INT 的中断使能位。(读/写)

**UHCI\_OUT\_DONE\_INT\_ENA** UHCI\_OUT\_DONE\_INT 的中断使能位。(读/写)

**UHCI\_OUT\_EOF\_INT\_ENA** UHCI\_OUT\_EOF\_INT 的中断使能位。(读/写)

**UHCI\_IN\_DSCR\_ERR\_INT\_ENA** UHCI\_IN\_DSCR\_ERR\_INT 的中断使能位。(读/写)

**UHCI\_OUT\_DSCR\_ERR\_INT\_ENA** UHCI\_OUT\_DSCR\_ERR\_INT 的中断使能位。(读/写)

**UHCI\_IN\_DSCR\_EMPTY\_INT\_ENA** UHCI\_IN\_DSCR\_EMPTY\_INT 的中断使能位。(读/写)

**UHCI\_OUTLINK\_EOF\_ERR\_INT\_ENA** UHCI\_OUTLINK\_EOF\_ERR\_INT 的中断使能位。(读/写)

**UHCI\_OUT\_TOTAL\_EOF\_INT\_ENA** UHCI\_OUT\_TOTAL\_EOF\_INT 的中断使能位。(读/写)

**UHCI\_SEND\_S\_REG\_Q\_INT\_ENA** UHCI\_SEND\_S\_REG\_Q\_INT 的中断使能位。(读/写)

**UHCI\_SEND\_A\_REG\_Q\_INT\_ENA** UHCI\_SEND\_A\_REG\_Q\_INT 的中断使能位。(读/写)

**UHCI\_DMA\_INFIFO\_FULL\_WM\_INT\_ENA** UHCI\_DMA\_INFIFO\_FULL\_WM\_INT 的中断使能位。  
(读/写)

Register 8.59: UHCI\_INT\_CLR\_REG (0x0010)

(reserved)																UHCI_DMA_INFIFO_FULL_WM_INT_CLR UHCI_SEND_A_REG_Q_INT_CLR UHCI_SEND_S_REG_Q_INT_CLR UHCI_OUT_TOTAL_EOF_INT_CLR UHCI_OUTLINK_EOF_ERR_INT_CLR UHCI_IN_DSCR_EMPTY_INT_CLR UHCI_OUT_DSCR_ERR_INT_CLR UHCI_OUT_EOF_ERR_INT_CLR UHCI_IN_ERR_DONE_INT_CLR UHCI_IN_SUC_EOF_INT_CLR UHCI_TX_HUNG_INT_CLR UHCI_RX_HUNG_INT_CLR UHCI_TX_START_INT_CLR UHCI_RX_START_INT_CLR																																	
31																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	Reset

**UHCI\_RX\_START\_INT\_CLR** 置位此位清除 UHCI\_RX\_START\_INT 中断。(只写)

**UHCI\_TX\_START\_INT\_CLR** 置位此位清除 UHCI\_TX\_START\_INT 中断。(只写)

**UHCI\_RX\_HUNG\_INT\_CLR** 置位此位清除 UHCI\_RX\_HUNG\_INT 中断。(只写)

**UHCI\_TX\_HUNG\_INT\_CLR** 置位此位清除 UHCI\_TX\_HUNG\_INT 中断。(只写)

**UHCI\_IN\_DONE\_INT\_CLR** 置位此位清除 UHCI\_IN\_DONE\_INT 中断。(只写)

**UHCI\_IN\_SUC\_EOF\_INT\_CLR** 置位此位清除 UHCI\_IN\_SUC\_EOF\_INT 中断。(只写)

**UHCI\_IN\_ERR\_EOF\_INT\_CLR** 置位此位清除 UHCI\_IN\_ERR\_EOF\_INT 中断。(只写)

**UHCI\_OUT\_DONE\_INT\_CLR** 置位此位清除 UHCI\_OUT\_DONE\_INT 中断。(只写)

**UHCI\_OUT\_EOF\_INT\_CLR** 置位此位清除 UHCI\_OUT\_EOF\_INT 中断。(只写)

**UHCI\_IN\_DSCR\_ERR\_INT\_CLR** 置位此位清除 UHCI\_IN\_DSCR\_ERR\_INT 中断。(只写)

**UHCI\_OUT\_DSCR\_ERR\_INT\_CLR** 置位此位清除 UHCI\_OUT\_DSCR\_ERR\_INT 中断。(只写)

**UHCI\_IN\_DSCR\_EMPTY\_INT\_CLR** 置位此位清除 UHCI\_IN\_DSCR\_EMPTY\_INT 中断。(只写)

**UHCI\_OUTLINK\_EOF\_ERR\_INT\_CLR** 置位此位清除 UHCI\_OUTLINK\_EOF\_ERR\_INT 中断。(只写)

**UHCI\_OUT\_TOTAL\_EOF\_INT\_CLR** 置位此位清除 UHCI\_OUT\_TOTAL\_EOF\_INT 中断。(只写)

**UHCI\_SEND\_S\_REG\_Q\_INT\_CLR** 置位此位清除 UHCI\_SEND\_S\_UHCI\_Q\_INT 中断。(只写)

**UHCI\_SEND\_A\_REG\_Q\_INT\_CLR** 置位此位清除 UHCI\_SEND\_A\_UHCI\_Q\_INT 中断。(只写)

**UHCI\_DMA\_INFIFO\_FULL\_WM\_INT\_CLR** 置位此位清除 UHCI\_DMA\_INFIFO\_FULL\_WM\_INT 中断。(只写)

Register 8.60: UHCI\_DMA\_OUT\_STATUS\_REG (0x0014)

(reserved)																															UHCI_OUT_EMPTY UHCI_OUT_FULL		
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	1	0	
																															Reset		

- UHCI\_OUT\_FULL** 1: DMA 数据输出 FIFO 为满。(只读)
- UHCI\_OUT\_EMPTY** 1: DMA 数据输出 FIFO 为空。(只读)

Register 8.61: UHCI\_DMA\_IN\_STATUS\_REG (0x001C)

(reserved)																										UHCI_RX_ERR_CAUSE		(reserved)		UHCI_IN_EMPTY		UHCI_IN_FULL	
31																										7	6	4	3	2	1	0	
0 0																										0x0	0	0	1	0	Reset		

- UHCI\_IN\_FULL** 数据输入 FIFO 满信号。(只读)
- UHCI\_IN\_EMPTY** 数据输入 FIFO 空信号。(只读)
- UHCI\_RX\_ERR\_CAUSE** 在 DMA 接收到错误帧时表示错误类型。3'b001: HCI 包校验和错误；3'b010: HCI 包序列号错误。3'b011: HCI 包 CRC 位错误；3'b100: 找到 0xc0 但接收的 HCI 包不完整；3'b101: 未找到 0xc0 但接收的 HCI 包完整；3'b110: CRC 检测错误。(只读)

### Register 8.62: UHCI\_STATE0\_REG (0x0030)

(reserved)																UHCI_DECODE_STATE																UHCI_INFIFO_CNT_DEBUG																UHCI_IN_STATE																UHCI_IN_DSCR_STATE																UHCI_INLINK_DSCR_ADDR															
31	30	28	27					23	22	20	19	18	17												0																																																																						
0	0	0				0				0	0	0											0	Reset																																																																							

**UHCI\_INLINK\_DSCR\_ADDR** 存储当前接收链表描述符的地址。(只读)

**UHCI\_IN\_DSCR\_STATE** 保留。(只读)

**UHCI\_IN\_STATE** 保留。(只读)

**UHCI\_INFIFO\_CNT\_DEBUG** 存储接收链表描述符 FIFO 中的数据字节数。(只读)

**UHCI\_DECODE\_STATE** UHCI 解码器状态。(只读)

**Register 8.63: UHCI\_DMA\_OUT\_EOF\_DES\_ADDR\_REG (0x0038)**

UHCI\_OUT\_EOF\_DES\_ADDR

31	0
0x000000	
Reset	

**UHCI\_OUT\_EOF\_DES\_ADDR** 存储发送链表描述符的 EOF 位为 1 时该描述符的地址。(只读)

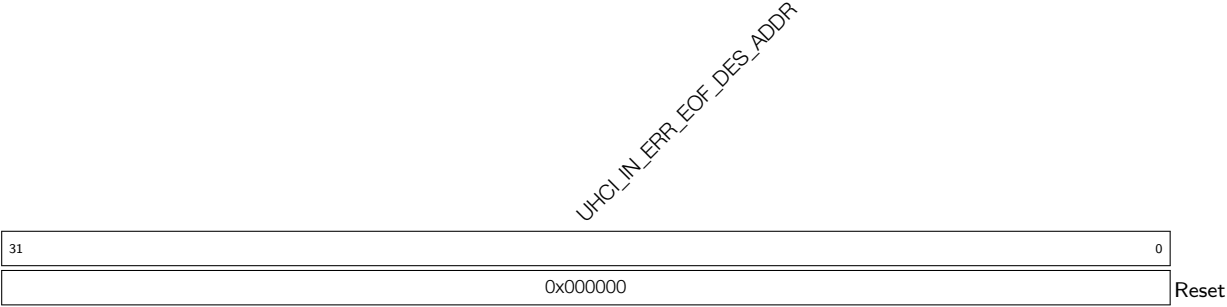
**Register 8.64: UHCI\_DMA\_IN\_SUC\_EOF\_DES\_ADDR\_REG (0x003C)**

Diagram of the UHCI\_IN\_SUC\_EOF\_DES\_ADDR register. The register is 32 bits wide, with bit 31 on the left and bit 0 on the right. The value 0x000000 is shown in the register, and the label 'Reset' is next to it.

**UHCI\_IN\_SUC\_EOF\_DES\_ADDR** 存储成功接收 EOF 时接收链表描述符的地址。(只读)

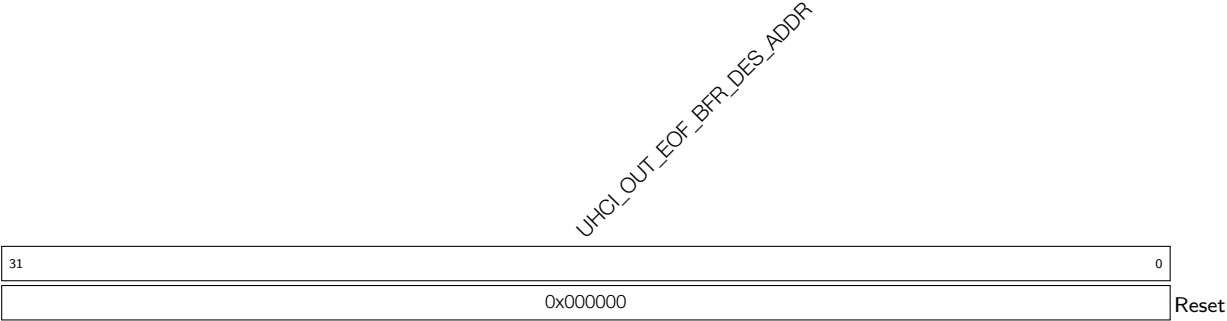


Register 8.65: UHCI\_DMA\_IN\_ERR\_EOF\_DES\_ADDR\_REG (0x0040)



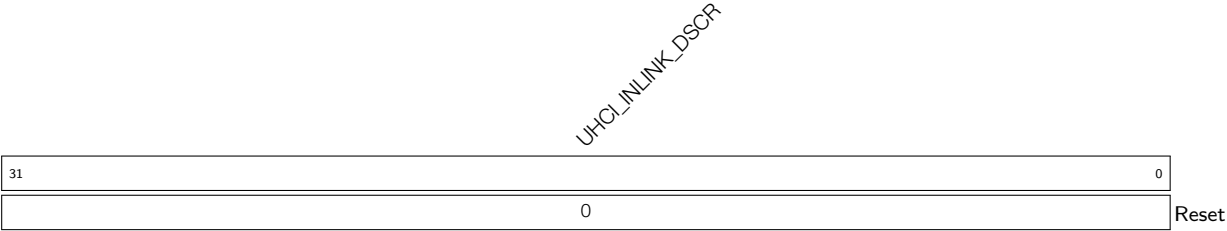
**UHCI\_IN\_ERR\_EOF\_DES\_ADDR** 存储接收链表描述符存在错误时该描述符的地址。(只读)

Register 8.66: UHCI\_DMA\_OUT\_EOF\_BFR\_DES\_ADDR\_REG (0x0044)



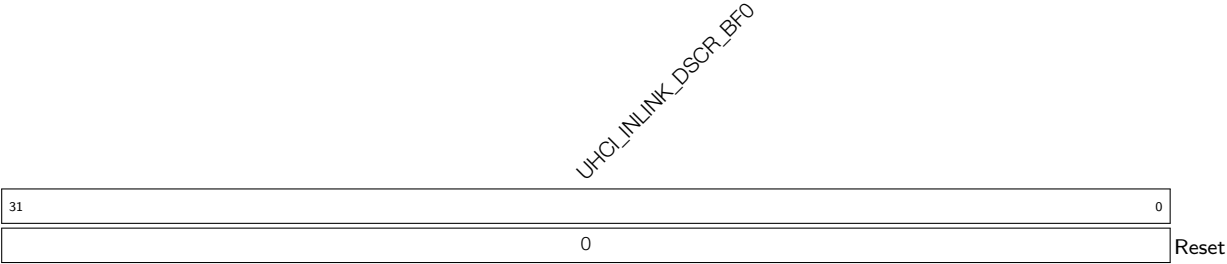
**UHCI\_OUT\_EOF\_BFR\_DES\_ADDR** 存储上一个发送链表描述符之前的描述符地址。(只读)

Register 8.67: UHCI\_DMA\_IN\_DSCR\_REG (0x004C)



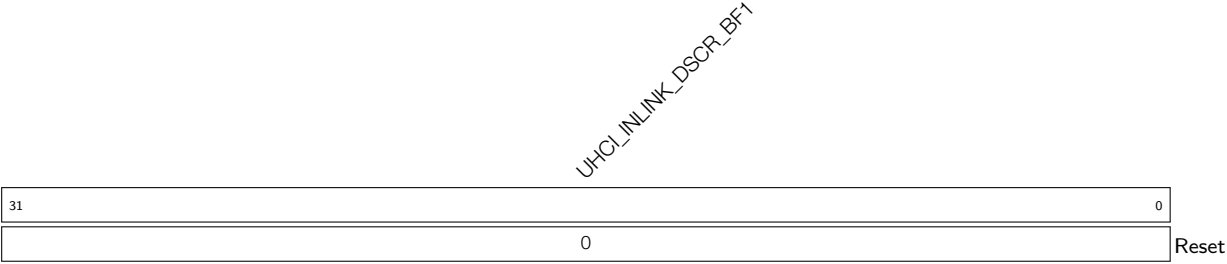
**UHCI\_INLINK\_DSCR** 存储下一个接收链表描述符的第三个字。(只读)

Register 8.68: UHCI\_DMA\_IN\_DSCR\_BF0\_REG (0x0050)



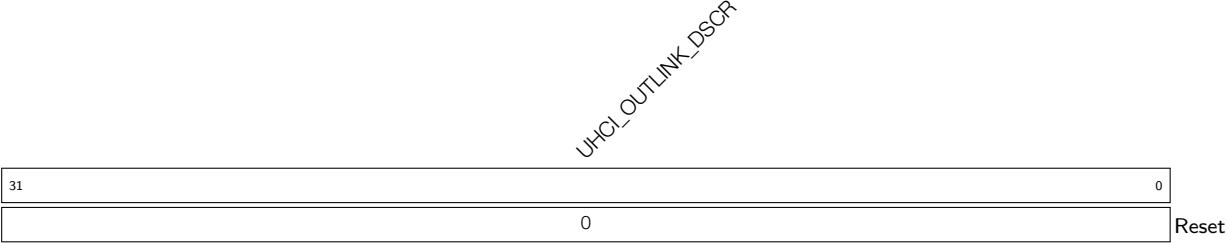
**UHCI\_INLINK\_DSCR\_BF0** 存储当前接收链表描述符的第三个字。(只读)

Register 8.69: UHCI\_DMA\_IN\_DSCR\_BF1\_REG (0x0054)



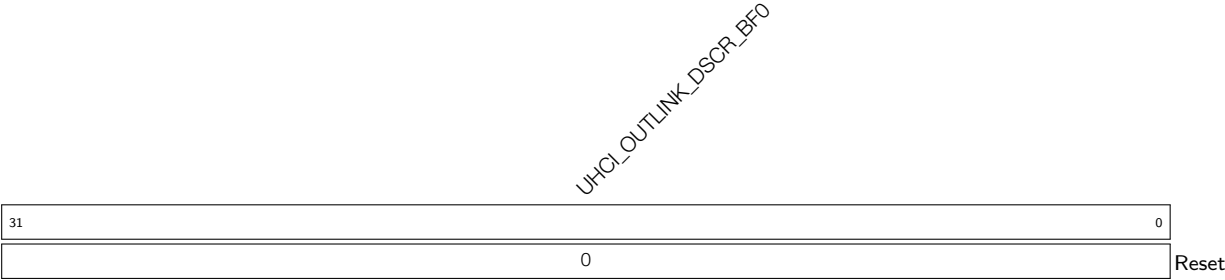
**UHCI\_INLINK\_DSCR\_BF1** 存储当前接收链表描述符的第二个字。(只读)

Register 8.70: UHCI\_DMA\_OUT\_DSCR\_REG (0x0058)



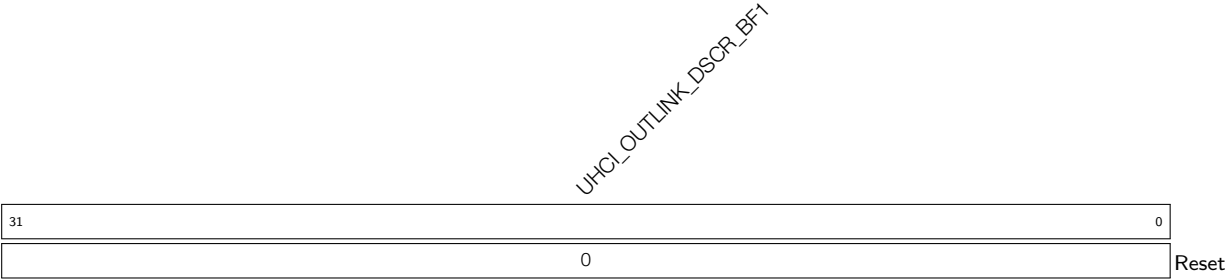
**UHCI\_OUTLINK\_DSCR** 存储下一个发送链表描述符的第三个字。(只读)

Register 8.71: UHCI\_DMA\_OUT\_DSCR\_BF0\_REG (0x005C)



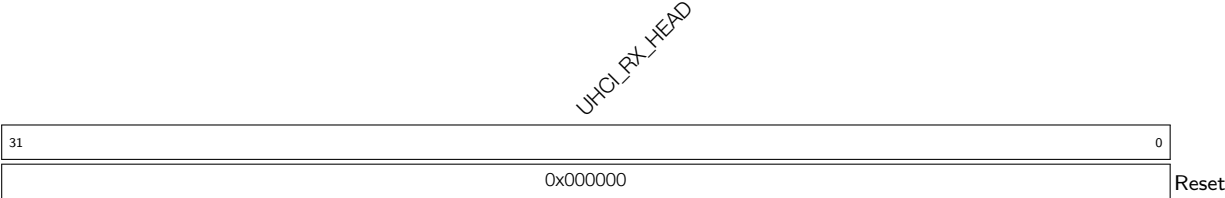
**UHCI\_OUTLINK\_DSCR\_BF0** 存储当前发送链表描述符的第三个字。(只读)

Register 8.72: UHCI\_DMA\_OUT\_DSCR\_BF1\_REG (0x0060)



**UHCI\_OUTLINK\_DSCR\_BF1** 存储当前发送链表描述符的第二个字。(只读)

Register 8.73: UHCI\_RX\_HEAD\_REG (0x0070)



**UHCI\_RX\_HEAD** 存储当前接收包的报头 (只读)



Register 8.77: UHCI\_DMA\_IN\_LINK\_REG (0x0028)

UHCI_INLINK_PARK				UHCI_INLINK_RESTART				UHCI_INLINK_START				UHCI_INLINK_STOP				(reserved)				UHCI_INLINK_AUTO_RET				UHCI_INLINK_ADDR						
31	30	29	28	27								21	20	19											0					
0	0	0	0	0	0	0	0	0	0	0	0	1	0x000																	Reset

- UHCI\_INLINK\_ADDR** 用于设定第一个接收链表描述符地址的低 20 位。(读/写)
- UHCI\_INLINK\_AUTO\_RET** 当前包存在错误时恢复当前接收链表描述符地址的使能位。(读/写)
- UHCI\_INLINK\_STOP** 置位此位，停止处理接收链表描述符。(读/写)
- UHCI\_INLINK\_START** 置位此位，开始处理接收链表描述符。(读/写)
- UHCI\_INLINK\_RESTART** 置位此位，启动新的接收链表描述符。(读/写)
- UHCI\_INLINK\_PARK** 1: 接收链表描述符的 FSM 空闲。0: 接收链表描述符的 FSM 正在工作。(只读)

Register 8.78: UHCI\_STATE1\_REG (0x0034)

(reserved)				UHCL_ENCODE_STATE				UHCL_OUTFIFO_CNT				UHCL_OUT_STATE				UHCL_OUT_DSCR_STATE				UHCL_OUTLINK_DSCR_ADDR			
31	30	28	27	23	22	20	19	18	17											0			
0	0	0	0	0	0	0	0										Reset						

- UHCI\_OUTLINK\_DSCR\_ADDR** 存储当前发送链表描述符的地址。(只读)
- UHCI\_OUT\_DSCR\_STATE** 保留。(只读)
- UHCI\_OUT\_STATE** 保留。(只读)
- UHCI\_OUTFIFO\_CNT** 存储发送链表描述符 FIFO 中的数据字节数。(只读)
- UHCI\_ENCODE\_STATE** UHCI 编码器状态。(只读)

Register 8.79: UHCI\_DATE\_REG (0x00FC)

UHCI_DATE	
31	0
0x18073001	
Reset	

UHCI\_DATE 版本控制寄存器。(读/写)

## 9. LED PWM 控制器

### 9.1 概述

LED PWM 控制器主要用于控制 LED 设备，也可以产生 PWM 信号用于控制其他开关设备。该控制器由 14 位定时器和波形发生器组成。

### 9.2 特性

LED PWM 控制器具有如下特性：

- 四个独立定时器，可实现小数分频
- 八个独立波形发生器，可产生 8 路 PWM 信号
- PWM 信号占空比可递增或递减渐变，无须处理器干预。渐变完成可产生中断事件
- 输出 PWM 信号相位可调
- 低功耗模式下可输出 PWM 信号

为方便描述，下文中八个 PWM 发生器统称为 PWM $n$ ，四个定时器统称为 Timer $x$ 。

### 9.3 功能描述

#### 9.3.1 架构

图 9-1 为 LED PWM 控制器的架构。图 9-2 为一个 PWM 生成器及其选取的定时器和计数器。

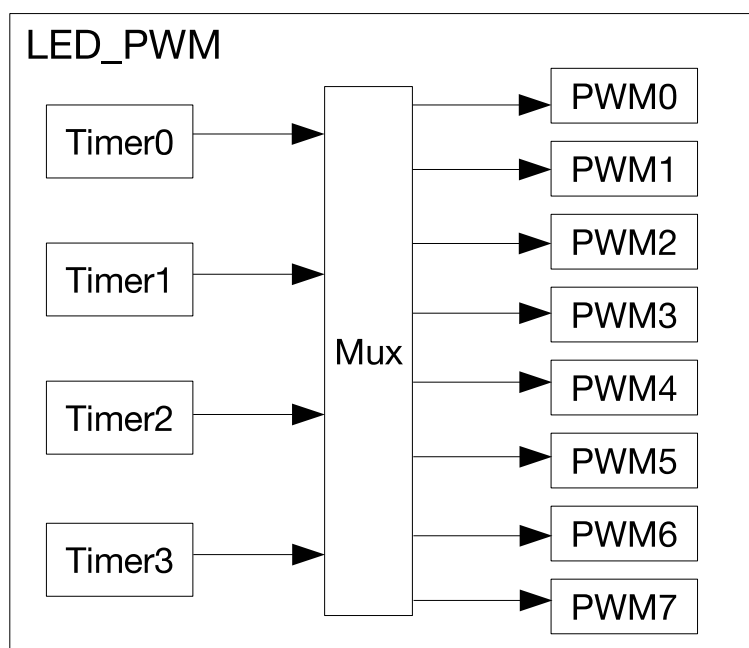


图 9-1. LED\_PWM 架构

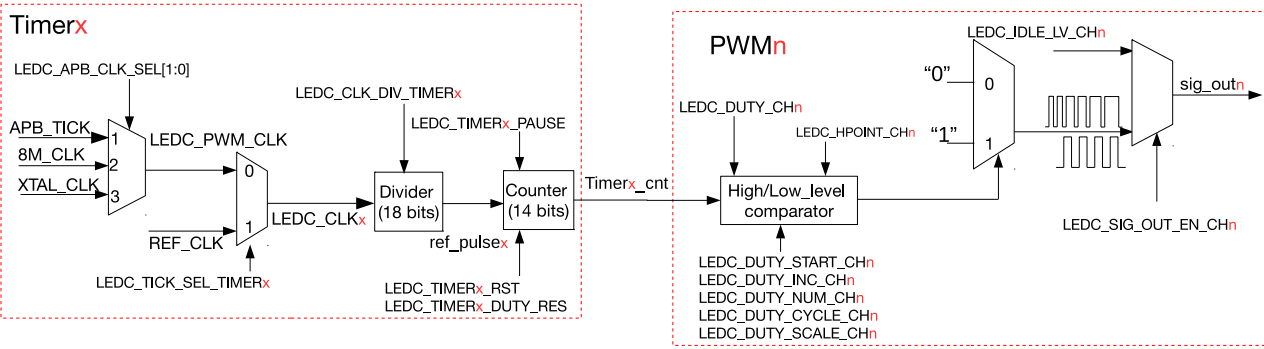


图 9-2. LED\_PWM 生成器图

### 9.3.2 定时器

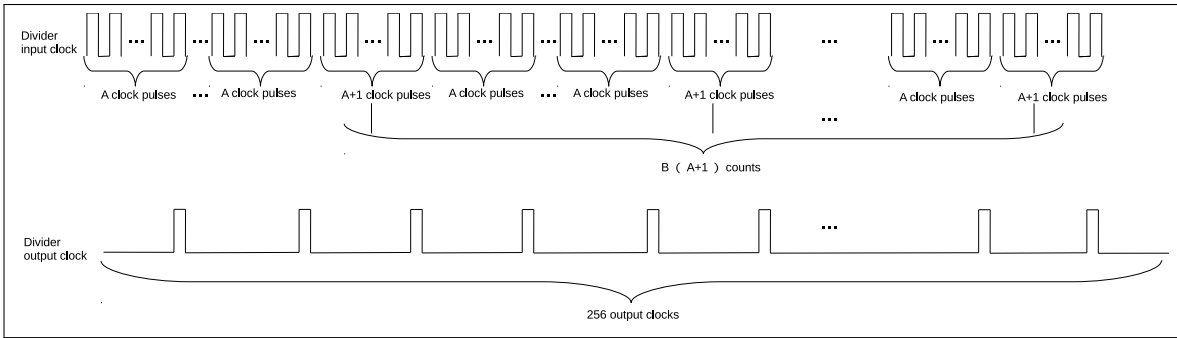


图 9-3. LED\_PWM 分频器

LED PWM 控制器的时钟 LEDC\_PWM\_CLK 有三个时钟源：APB\_CLK、RTC8M\_CLK 和 XTAL\_CLK，可通过配置 LEDC\_APB\_CLK\_SEL[1:0] 来选择。LED PWM 定时器的时钟 LEDC\_CLKx 有两个时钟源：LEDC\_PWM\_CLK 和 REF\_TICK。如使用 REF\_TICK 作定时器的时钟源，需将 LEDC\_APB\_CLK\_SEL[1:0] 置 1，REF\_TICK 的周期需为 APB\_CLK 周期的整数倍，否则定时器的时钟会不精准。更多关于时钟源的信息请参考 [复位与时钟](#) 一章。

LEDC\_CLKx 经分频器分频后的输出时钟用作计数器的基准时钟。分频器的分频系数通过 LEDC\_CLK\_DIV\_TIMERx 寄存器配置。该分频系数为定点数，高 10 位为整数部分 A，低 8 位为小数部分 B。分频系数 LEDC\_CLK\_DIVx 的公式为：

$$LEDC\_CLK\_DIVx = A + \frac{B}{256}$$

小数部分 B 不为 0 时，分频器的输入和输出时钟如图 9-3 所示。256 个输出时钟中有 B 个以 (A+1) 分频，有 (256-B) 个以 A 分频。以 (A+1) 分频的 B 个输出时钟均匀分布在 256 个输出时钟中。

LED PWM 控制器的计数器为 14 位，计数最大值为  $2^{LEDC\_TIMERx\_DUTY\_RES} - 1$ 。每当计数值达到  $2^{LEDC\_TIMERx\_DUTY\_RES} - 1$  时，计数器便会溢出，并重新从 0 开始计数。软件可以复位、暂停并读取计数器的计数值。计数器每次溢出时可产生 LEDC\_TIMERx\_OVF\_INT 中断，也可在溢出 (LEDC\_OVF\_NUM\_CHn + 1) 次时产生中断。寄存器配置流程如下：

1. 置位 LEDC\_OVF\_CNT\_EN\_CHn
2. 将 LEDC\_OVF\_NUM\_CHn 配置为溢出次数减 1
3. 置位 LEDC\_OVF\_CNT\_CHn\_INT\_ENA



4. 置位 `LEDC_TIMERx_DUTY_RES` 启动定时器，等待 `LEDC_OVF_CNT_CHn_INT` 中断

PWM 生成器输出信号 `sig_outn` 的频率取决于分频器的分频系数以及计数器的计数范围：

$$f_{\text{sig\_out}n} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV}x \cdot 2^{\text{LEDC\_TIMER}x\_DUTY\_RES}}$$

重新设置分频器的分频系数和计数器的溢出值，需分别配置 `LEDC_CLK_DIV_TIMERx` 和 `LEDC_TIMERx_DUTY_RES`，然后置位 `LEDC_TIMERx_PARA_UP`，否则配置无效。新的配置在计数器溢出时更新。

### 9.3.3 PWM 生成器

如图 9-2 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 生成器获取 LED PWM 定时器的 14 位计数值，与高低电平比较器的值 `Hpointn` 和 `Lpointn` 比较，来控制 PWM 信号的电平。

- 如果 `Timerx_cnt == Hpointn`，则 `sig_outn` 为 1。
- 如果 `Timerx_cnt == Lpointn`，则 `sig_outn` 为 0。

`Hpointn` 由 `LEDC_HPOINT_CHn` 在计数器溢出时更新。`Lpointn` 的初始值为计数器溢出时 `LEDC_DUTY_CHn[18..4]` 和 `LEDC_HPOINT_CHn` 的和。通过配置以上两个字段，可设置 PWM 输出的相对相位和占空比。

图 9-4 为占空比固定时的 PWM 波形。

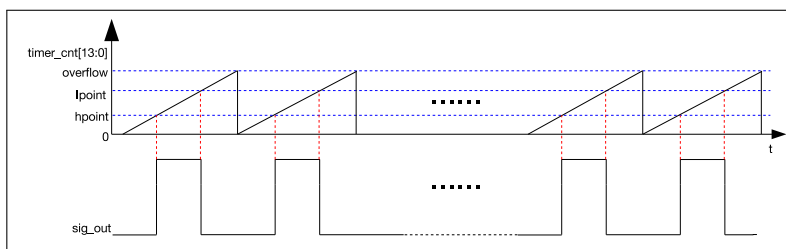


图 9-4. LED\_PWM 输出信号图

`LEDC_DUTY_CHn` 是一个具有四位小数的定点寄存器。`LEDC_DUTY_CHn[18..4]` 为整数部分，可直接用于 PWM 计算。`LEDC_DUTY_CHn[3..0]` 为小数部分，可用于调整输出信号。如 `LEDC_DUTY_CHn[3..0]` 不为 0，那么 `sig_outn` 每 16 个周期中，有 `LEDC_DUTY_CHn[3..0]` 个周期的 PWM 脉冲宽度要比  $(16 - \text{LEDC\_DUTY\_CH}n[3..0])$  个周期的脉冲宽度多一个定时器的计数周期。该功能可将 PWM 生成器的精度提至 18 位。

### 9.3.4 渐变占空比

LED PWM 输出信号可由一种占空比渐变为另一种占空比。渐变功能由 `LEDC_DUTY_CHn`、`LEDC_DUTY_START_CHn`、`LEDC_DUTY_INC_CHn`、`LEDC_DUTY_NUM_CHn` 和 `LEDC_DUTY_SCALE_CHn` 配置。

`LEDC_DUTY_START_CHn` 用于更新 `Lpointn` 的值。如置位该位，计数器溢出时 `Lpointn` 会自动递增或递减。

`LEDC_DUTY_INC_CHn` 位用于设置渐变模式。

sig\_out $n$  的占空比每隔 LEDC\_DUTY\_CYCLE\_CH $n$  个 PWM 脉冲周期，便会在当前占空比上加上或减去 LEDC\_DUTY\_SCALE\_CH $n$ 。

占空比渐变次数达到 LEDC\_DUTY\_NUM\_CH $n$  时停止渐变，并产生 LEDC\_DUTY\_CHNG\_END\_CH $n$ \_INT。图 9-5 为渐变的图例。该配置下，每 LEDC\_DUTY\_CYCLE\_CH $n$  个 PWM 脉冲周期，sig\_out $n$  的占空比会增加 LEDC\_DUTY\_SCALE\_CH $n$ 。

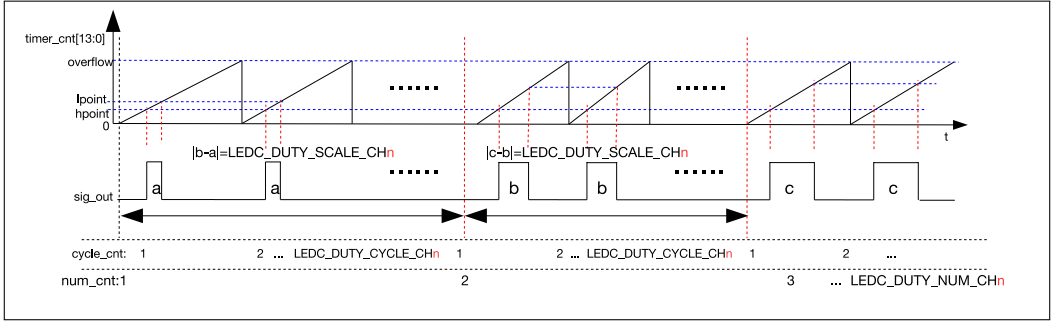


图 9-5. 渐变占空比输出信号图

LEDC\_SIG\_OUT\_EN\_CH $n$  用于启动 PWM 波形输出。LEDC\_SIG\_OUT\_EN\_CH $n$  为 0 时，sig\_out $n$  的电平值恒定为 LEDC\_IDLE\_LV\_CH $n$ 。

软件更新 LEDC\_HPOINT\_CH $n$ 、LEDC\_DUTY\_START\_CH $n$ 、LEDC\_SIG\_OUT\_EN\_CH $n$ 、LEDC\_TIMER\_SEL\_CH $n$ 、LEDC\_DUTY\_NUM\_CH $n$ 、LEDC\_DUTY\_CYCLE\_CH $n$ 、LEDC\_DUTY\_SCALE\_CH $n$ 、LEDC\_DUTY\_INC\_CH $n$  和 LEDC\_OVF\_CNT\_EN\_CH $n$  的配置后，需置位 LEDC\_PARA\_UP\_CH $n$  应用新配置。

### 9.3.5 中断

- 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：LED PWM 生成器渐变完成后触发中断。
- LEDC\_TIMER $x$ \_OVF\_INT：LED PWM 定时器达到最大计数值时触发中断。

## 9.4 基地址

用户可以通过两个不同的寄存器基地址访问 LED PWM，如表 39 所示。更多信息，请访问章节 1 系统和存储器。

表 39: LED\_PWM 基地址

访问总线	基地址
PeriBUS1	0x3F419000
PeriBUS2	0x60019000

## 9.5 寄存器列表

请注意，下表的地址指相对于 LED PWM 基地址的偏移量（相对地址）。请参阅章节 9.4 获取有关 LED PWM 基地址的信息。请注意，下表中的地址都是相对于 LED PWM 基地址的地址偏移量（相对地址）。更多有关 LED PWM 基地址的信息，请前往 9.4 章节。

名称	描述	地址	访问
<b>配置寄存器</b>			
LEDC_CH0_CONF0_REG	通道 0 的配置寄存器 0	0x0000	不定
LEDC_CH0_CONF1_REG	通道 0 的配置寄存器 1	0x000C	读/写
LEDC_CH1_CONF0_REG	通道 1 的配置寄存器 0	0x0014	不定
LEDC_CH1_CONF1_REG	通道 1 的配置寄存器 1	0x0020	读/写
LEDC_CH2_CONF0_REG	通道 2 的配置寄存器 0	0x0028	不定
LEDC_CH2_CONF1_REG	通道 2 的配置寄存器 1	0x0034	读/写
LEDC_CH3_CONF0_REG	通道 3 的配置寄存器 0	0x003C	不定
LEDC_CH3_CONF1_REG	通道 3 的配置寄存器 1	0x0048	读/写
LEDC_CH4_CONF0_REG	通道 4 的配置寄存器 0	0x0050	不定
LEDC_CH4_CONF1_REG	通道 4 的配置寄存器 1	0x005C	读/写
LEDC_CH5_CONF0_REG	通道 5 的配置寄存器 0	0x0064	不定
LEDC_CH5_CONF1_REG	通道 5 的配置寄存器 1	0x0070	读/写
LEDC_CH6_CONF0_REG	通道 6 的配置寄存器 0	0x0078	不定
LEDC_CH6_CONF1_REG	通道 6 的配置寄存器 1	0x0084	读/写
LEDC_CH7_CONF0_REG	通道 7 的配置寄存器 0	0x008C	不定
LEDC_CH7_CONF1_REG	通道 7 的配置寄存器 1	0x0098	读/写
LEDC_CONF_REG	LEDC 全局配置寄存器	0x00D0	读/写
<b>高位点寄存器</b>			
LEDC_CH0_HPOINT_REG	通道 0 的高位点寄存器	0x0004	读/写
LEDC_CH1_HPOINT_REG	通道 1 的高位点寄存器	0x0018	读/写
LEDC_CH2_HPOINT_REG	通道 2 的高位点寄存器	0x002C	读/写
LEDC_CH3_HPOINT_REG	通道 3 的高位点寄存器	0x0040	读/写
LEDC_CH4_HPOINT_REG	通道 4 的高位点寄存器	0x0054	读/写
LEDC_CH5_HPOINT_REG	通道 5 的高位点寄存器	0x0068	读/写
LEDC_CH6_HPOINT_REG	通道 6 的高位点寄存器	0x007C	读/写
LEDC_CH7_HPOINT_REG	通道 7 的高位点寄存器	0x0090	读/写
<b>占空比寄存器</b>			
LEDC_CH0_DUTY_REG	通道 0 的初始占空比	0x0008	读/写
LEDC_CH0_DUTY_R_REG	通道 0 的当前占空比	0x0010	只读
LEDC_CH1_DUTY_REG	通道 1 的初始占空比	0x001C	读/写
LEDC_CH1_DUTY_R_REG	通道 1 的当前占空比	0x0024	只读
LEDC_CH2_DUTY_REG	通道 2 的初始占空比	0x0030	读/写
LEDC_CH2_DUTY_R_REG	通道 2 的当前占空比	0x0038	只读
LEDC_CH3_DUTY_REG	通道 3 的初始占空比	0x0044	读/写
LEDC_CH3_DUTY_R_REG	通道 3 的当前占空比	0x004C	只读
LEDC_CH4_DUTY_REG	通道 4 的初始占空比	0x0058	读/写

名称	描述	地址	访问
LEDC_CH4_DUTY_R_REG	通道 4 的当前占空比	0x0060	只读
LEDC_CH5_DUTY_REG	通道 5 的初始占空比	0x006C	读/写
LEDC_CH5_DUTY_R_REG	通道 5 的当前占空比	0x0074	只读
LEDC_CH6_DUTY_REG	通道 6 的初始占空比	0x0080	读/写
LEDC_CH6_DUTY_R_REG	通道 6 的当前占空比	0x0088	只读
LEDC_CH7_DUTY_REG	通道 7 的初始占空比	0x0094	读/写
LEDC_CH7_DUTY_R_REG	通道 7 的当前占空比	0x009C	只读
<b>定时器寄存器</b>			
LEDC_TIMER0_CONF_REG	定时器 0 配置	0x00A0	不定
LEDC_TIMER0_VALUE_REG	定时器 0 的当前计数器值	0x00A4	只读
LEDC_TIMER1_CONF_REG	定时器 1 配置	0x00A8	不定
LEDC_TIMER1_VALUE_REG	定时器 1 的当前计数器值	0x00AC	只读
LEDC_TIMER2_CONF_REG	定时器 2 配置	0x00B0	不定
LEDC_TIMER2_VALUE_REG	定时器 2 的当前计数器值	0x00B4	只读
LEDC_TIMER3_CONF_REG	定时器 3 配置	0x00B8	不定
LEDC_TIMER3_VALUE_REG	定时器 3 的当前计数器值	0x00BC	只读
<b>中断寄存器</b>			
LEDC_INT_RAW_REG	原始中断状态	0x00C0	只读
LEDC_INT_ST_REG	屏蔽中断状态	0x00C4	只读
LEDC_INT_ENA_REG	中断使能位	0x00C8	读/写
LEDC_INT_CLR_REG	中断清除位	0x00CC	只写
<b>版本寄存器</b>			
LEDC_DATE_REG	版本控制寄存器	0x00FC	读/写

## 9.6 寄存器

Register 9.1: LEDC\_CH $n$ \_CONF0\_REG ( $n$ : 0-7) (0x0000+20\* $n$ )

(reserved)														LEDC_OVF_CNT_RESET_ST_CH <sub>n</sub>			LEDC_OVF_CNT_RESET_CH <sub>n</sub>			LEDC_OVF_CNT_EN_CH <sub>n</sub>			LEDC_OVF_NUM_CH <sub>n</sub>				LEDC_PARA_UP_CH <sub>n</sub>				LEDC_IDLE_LV_CH <sub>n</sub>		LEDC_SIG_OUT_EN_CH <sub>n</sub>		LEDC_TIMER_SEL_CH <sub>n</sub>	
31														18		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	0x0				0				0	0	0x0		Reset						

**LEDC\_TIMER\_SEL\_CH $n$**  用于选择通道  $n$  的定时器。0: 选择定时器 0; 1: 选择定时器 1; 2: 选择定时器 2; 3: 选择定时器 3。(读/写)

**LEDC\_SIG\_OUT\_EN\_CH $n$**  置位此位, 使能通道  $n$  的信号输出。(读/写)

**LEDC\_IDLE\_LV\_CH $n$**  控制通道  $n$  不工作时的输出电平。(读/写)

**LEDC\_PARA\_UP\_CH $n$**  用于更新通道  $n$  的 LEDC\_CH $n$ \_HPOINT 和 LEDC\_CH $n$ \_DUTY 寄存器。(只写)

**LEDC\_OVF\_NUM\_CH $n$**  用于配置定时器溢出次数的最大值减 1。通道  $n$  的定时器溢出次数达到 (LEDC\_OVF\_CNT\_CH $n$  + 1) 时触发 LEDC\_OVF\_CNT\_CH $n$ \_INT 中断。(读/写)

**LEDC\_OVF\_CNT\_EN\_CH $n$**  用于使能通道  $n$  的 ovf\_cnt。(读/写)

**LEDC\_OVF\_CNT\_RESET\_CH $n$**  置位此位, 复位通道  $n$  的 ovf\_cnt。(只写)

**LEDC\_OVF\_CNT\_RESET\_ST\_CH $n$**  LEDC\_OVF\_CNT\_RESET\_CH $n$  的状态位。(只读)

Register 9.2: LEDC\_CH $n$ \_CONF1\_REG ( $n$ : 0-7) (0x000C+20\* $n$ )

LEDC_DUTY_START_CH <sup>n</sup> LEDC_DUTY_INC_CH <sup>n</sup>										LEDC_DUTY_NUM_CH <sup>n</sup>										LEDC_DUTY_CYCLE_CH <sup>n</sup>										LEDC_DUTY_SCALE_CH <sup>n</sup>																																								
31	30	29																												20	19	10										9	0																											
0	1	0x0																0x0												0x0												Reset																												

**LEDC\_DUTY\_SCALE\_CH $n$**  用于配置通道  $n$  占空比的变化步长。(读/写)

**LEDC\_DUTY\_CYCLE\_CH $n$**  通道  $n$  占空比每隔 LEDC\_DUTY\_CYCLE\_CH $n$  变化一次。(读/写)

**LEDC\_DUTY\_NUM\_CH $n$**  用于控制占空比变化的次数。(读/写)

**LEDC\_DUTY\_INC\_CH $n$**  用于递增或递减通道  $n$  输出信号的占空比。1: 递增; 0: 递减。(读/写)

**LEDC\_DUTY\_START\_CH $n$**  此位置 1 时, LEDC\_CH $n$ \_CONF1\_REG 中的其他字段生效。(读/写)

## 222

## 反馈文档意见

**LEDC\_CLK\_EN** 用于控制时钟。1'b1: 强制开启寄存器时钟。1'b0: 仅在应用写寄存器时支持时钟。  
(读/写)

**LEDC\_HPOINT\_CH<sub>n</sub>** 所选定时器计数值达到该值时，输出信号翻转为高电平。(读/写)

**LEDC\_DUTY\_CH<sub>n</sub>** 通过控制低位点改变输出信号占空比。所选定时器达到低位点时，输出信号翻转为低电平。(读/写)

Register 9.6: LEDC\_CH $n$ \_DUTY\_R\_REG ( $n$ : 0-7) (0x0010+20\* $n$ )

(reserved)																LEDC_DUTY_R_0H																		
31																	18																	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x000																Reset		

LEDC\_DUTY\_R\_CH $n$  存储通道  $n$  输出信号的当前占空比。(只读)

Register 9.7: LEDC\_TIMER $x$ \_CONF\_REG ( $x$ : 0-3) (0x00A0+8\* $x$ )

(reserved)										LEDC_TIMER <sub>x</sub> _PARAM_UP LEDC_TIMER <sub>x</sub> _SEL_TIMER <sub>x</sub> LEDC_TIMER <sub>x</sub> _RST LEDC_TIMER <sub>x</sub> _PAUSE										LEDC_CLK_DIV_TIMER <sub>x</sub>										LEDC_TIMER <sub>x</sub> _DUTY_RES																								
31						26						25	24	23	22	21						4						3	0																									
0						0						0						0						0						0						0						0x000						0x0						Reset

- LEDC\_TIMER $x$ \_DUTY\_RES 用于控制定时器  $x$  计数器的计数范围。(读/写)
- LEDC\_CLK\_DIV\_TIMER $x$  用于配置定时器  $x$  分频器的分频系数。低 8 位为小数部分。(读/写)
- LEDC\_TIMER $x$ \_PAUSE 用于暂停定时器  $x$  的计数器。(读/写)
- LEDC\_TIMER $x$ \_RST 用于复位定时器  $x$ 。复位后计数器为 0。(读/写)
- LEDC\_TIMER $x$ \_SEL\_TIMER $x$  用于为定时器  $x$  选择时钟。此位置 1 时，LEDC\_APB\_CLK\_SEL[1:0] 应为 1，否则定时器时钟会不精准。1'h0: SLOW\_CLK 1'h1: REF\_TICK (读/写)
- LEDC\_TIMER $x$ \_PARAM\_UP 置位此位，更新 LEDC\_CLK\_DIV\_TIMER $x$  和 LEDC\_TIMER $x$ \_DUTY\_RES。  
(只写)

Register 9.8: LEDC\_TIMER $x$ \_VALUE\_REG ( $x$ : 0-3) (0x00A4+8\* $x$ )

(reserved)																LEDC_TIMER <del>x</del> _CNT																
31															14	13															0	Reset
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00													

LEDC\_TIMER $x$ \_CNT 存储定时器  $x$  的当前计数器值 (只读)

Register 9.9: LEDC\_INT\_RAW\_REG (0x00C0)

(reserved)																						LEDC_OVF_CNT_CH7_INT_RAW												LEDC_OVF_CNT_CH6_INT_RAW												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												LEDC_DUTY_CHNG_END_CH7_INT_RAW												LEDC_DUTY_CHNG_END_CH6_INT_RAW												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																						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																					

Reset

**LEDC\_TIMER $x$ \_OVF\_INT\_RAW** 定时器  $x$  达到最大计数值时触发中断。(只读)

**LEDC\_DUTY\_CHNG\_END\_CH $n$ \_INT\_RAW** 通道  $n$  的原始中断位。占空比渐变结束时触发。(只读)

**LEDC\_OVF\_CNT\_CH $n$ \_INT\_RAW** 通道  $n$  的原始中断位。ovf\_cnt 达到 LEDC\_OVF\_NUM\_CH $n$  指定值时触发。(只读)

Register 9.10: LEDC\_INT\_ST\_REG (0x00C4)

(reserved)																						LEDC_OVF_CNT_CH7_INT_ST												LEDC_OVF_CNT_CH6_INT_ST												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_DUTY_CHNG_END_CH7_INT_ST												LEDC_DUTY_CHNG_END_CH6_INT_ST												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												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																																																																																																																																																																																																		

Reset

**LEDC\_TIMER $x$ \_OVF\_INT\_ST** LEDC\_TIMER $x$ \_OVF\_INT\_ENA 置 1 时，LEDC\_TIMER $x$ \_OVF\_INT 中断的屏蔽中断状态位。(只读)

**LEDC\_DUTY\_CHNG\_END\_CH $n$ \_INT\_ST** LEDC\_DUTY\_CHNG\_END\_CH $n$ \_INT\_ENA 置 1 时，LEDC\_DUTY\_CHNG\_END\_CH $n$ \_INT 中断的屏蔽中断状态位。(只读)

**LEDC\_OVF\_CNT\_CH $n$ \_INT\_ST** LEDC\_OVF\_CNT\_CH $n$ \_INT\_ENA 置 1 时，LEDC\_OVF\_CNT\_CH $n$ \_INT 中断的屏蔽中断状态位。(只读)



Register 9.11: LEDC\_INT\_ENA\_REG (0x00C8)

(reserved)																				LEDC_OVF_CNT_CH7_INT_ENA LEDC_OVF_CNT_CH6_INT_ENA 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_CH0_INT_ENA LEDC_DUTY_CHNG_END_CH7_INT_ENA LEDC_DUTY_CHNG_END_CH6_INT_ENA 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																				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	0	0	0	0	0	0	0	0	0	0	0	0

Reset

**LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_ENA** LEDC\_TIMER<sub>x</sub>\_OVF\_INT 中断的使能位。(读/写)

**LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_ENA** LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT 中断的使能位。  
(读/写)

**LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_ENA** LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT 中断的使能位。(读/写)

Register 9.12: LEDC\_INT\_CLR\_REG (0x00CC)

(reserved)												LEDC_OVF_CNT_CH7_INT_CLR LEDC_OVF_CNT_CH6_INT_CLR LEDC_OVF_CNT_CH5_INT_CLR LEDC_OVF_CNT_CH4_INT_CLR LEDC_OVF_CNT_CH3_INT_CLR LEDC_OVF_CNT_CH2_INT_CLR LEDC_OVF_CNT_CH1_INT_CLR LEDC_OVF_CNT_CH0_INT_CLR LEDC_DUTY_CHNG_END_CH7_INT_CLR LEDC_DUTY_CHNG_END_CH6_INT_CLR LEDC_DUTY_CHNG_END_CH5_INT_CLR LEDC_DUTY_CHNG_END_CH4_INT_CLR LEDC_DUTY_CHNG_END_CH3_INT_CLR LEDC_DUTY_CHNG_END_CH2_INT_CLR LEDC_DUTY_CHNG_END_CH1_INT_CLR LEDC_DUTY_CHNG_END_CH0_INT_CLR LEDC_TIMER3_OVF_INT_CLR LEDC_TIMER2_OVF_INT_CLR LEDC_TIMER1_OVF_INT_CLR LEDC_TIMER0_OVF_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	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

**LEDC\_TIMER<sub>x</sub>\_OVF\_INT\_CLR** 置位此位，清除 LEDC\_TIMER<sub>x</sub>\_OVF\_INT 中断。(只写)

**LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT\_CLR** 置位此位，清除 LEDC\_DUTY\_CHNG\_END\_CH<sub>n</sub>\_INT 中  
断。(只写)

**LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT\_CLR** 置位此位，清除 LEDC\_OVF\_CNT\_CH<sub>n</sub>\_INT 中断。(只写)

Register 9.13: LEDC\_DATE\_REG (0x00FC)

LEDC_DATE																															
31																															0
0x19072601																															
Reset																															

Reset

**LEDC\_DATE** 版本控制寄存器。(读/写)

## 10. 红外遥控

### 10.1 概述

RMT（红外收发器）是一个红外发送/接收控制器，支持多种红外协议。RMT 模块可以实现将模块内置 RAM 中的脉冲编码转换为信号输出，或检测模块的输入信号转换为脉冲编码存入 RAM 中。此外，RMT 模块可以选择是否对输出信号进行载波调制，也可以选择是否对输入信号进行滤波处理。

RMT 共有四个通道，可独立用于信号发送或接收，编码为 0~3，每个通道有一组功能相同的寄存器。为了方便叙述，以  $n$  表示各个通道。

### 10.2 功能描述

#### 10.2.1 RMT 架构

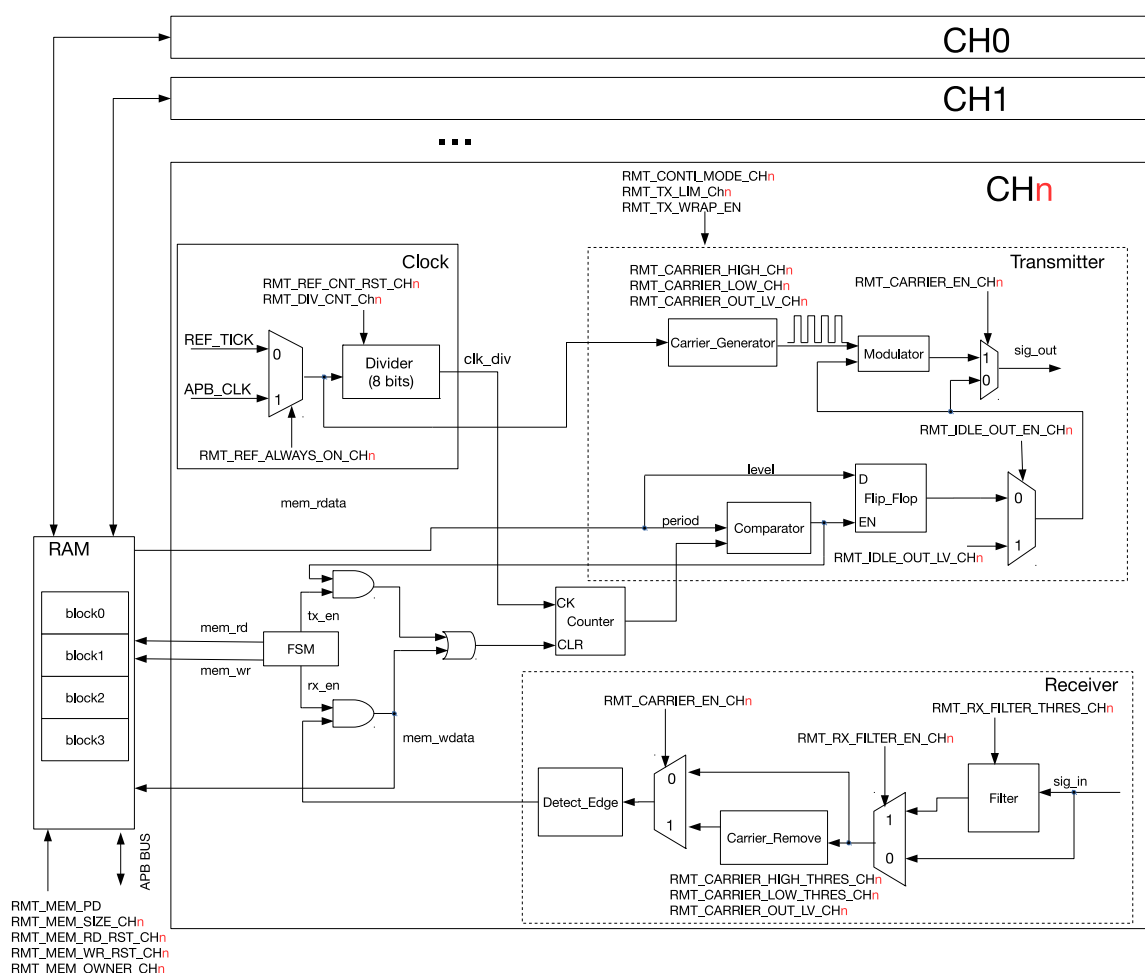


图 10-1. RMT 结构框图

RMT 模块有四个独立的通道，每个通道内部都有各自的一个时钟分频器，一个计数器，一个发射器和一个接收器。注意，同一个通道的发射器和接收器不可同时工作。四个通道共享一块 256 x 32 位的 RAM。

## 10.2.2 RMT RAM

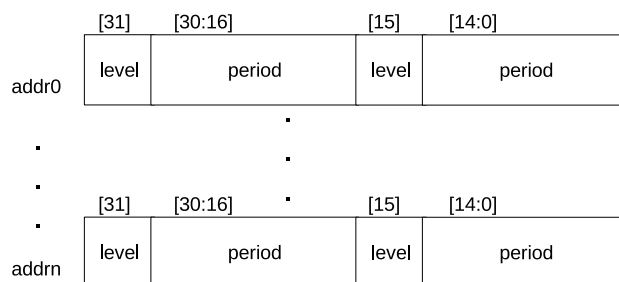


图 10-2. RAM 中脉冲编码结构

RAM 中脉冲编码结构如图 10-2 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（图 10-1 `clk_div`）周期数。Period 为 0 是传输结束标志。

RAM 按照  $64 \times 32$  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）。当通道  $n$  单次发送或接收的脉冲编码数大于一个 block 时，可以进行乒乓操作，或者通过配置 `RMT_MEM_SIZE_CH $n$`  寄存器，允许该通道占用多个 block。当设置 `RMT_MEM_SIZE_CH $n$`   $> 1$  时，通道  $n$  将占用 block  $(n) \sim \text{block}(n + \text{RMT\_MEM\_SIZE\_CH}_n - 1)$  的存储空间。通道  $n + 1 \sim n + \text{RMT\_MEM\_SIZE\_CH}_n - 1$  因为对应的 RAM block 被占用而无法使用。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 可以通过置位 `RMT_MEM_SIZE_CH $n$`  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。

RAM 可被 APB 总线及通道的发射器或接收器访问，为了防止发射器和接收器访问 RAM 时发生冲突，用户可以通过配置 `RMT_MEM_OWNER_CH $n$`  来决定当前 RAM 的使用权。当通道的发射器或接收器发生越权访问时会产生 `RMT_MEM_OWNER_ERR_CH $n$`  标志信号。

当 RMT 模块不工作时，可以通过配置 `RMT_MEM_FORCE_PD` 寄存器使 RAM 工作于低功耗模式。

## 10.2.3 时钟

用户可以通过配置 `RMT_REF_ALWAYS_ON_CH $n$`  选择时钟分频器的驱动时钟：APB\_CLK 或者 REF\_TICK，请参考复位和时钟章节。`RMT_DIV_CNT_CH $n$`  寄存器可以配置分频器的分频系数，除 0 表示 256 分频外，其他分频数等同于寄存器 `RMT_DIV_CNT_CH $n$`  的值。时钟分频器可以通过配置 `RMT_REF_CNT_RST_CH $n$`  进行复位。时钟分频器的分频时钟可供计数器使用。

## 10.2.4 发射器

当 `RMT_TX_START_CH $n$`  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_CH $n$ _TX_END_INT` 中断。配置 `RMT_TX_STOP_CH $n$`  寄存器可以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_CH $n$`  寄存器决定。用户可以配置 `RMT_IDLE_OUT_EN_CH $n$`  寄存器来选择这两种方式。

当发送的脉冲编码较多时，可通过置位 `RMT_MEM_TX_WRAP_EN` 寄存器使能乒乓操作。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，直到遇到结束标识为止。例如，当 `RMT_MEM_SIZE_CH $n$`  = 1 时，发射器将从  $64 * n$  地址开始发送，然后对应 RAM 的地址递增。发完  $64 * (n + 1)$

- 1) 地址的数据后，下次继续从  $64 * n$  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，可以更新已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

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 APB\_CLK 或者 REF\_TICK 时钟周期，低电平持续的时间为  $(RMT\_CARRIER\_LOW\_CHn + 1)$  个 APB\_CLK 或者 REF\_TICK 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。

置位  $RMT\_TX\_CONTI\_MODE\_CHn$  寄存器可以使能发射器的持续发送功能。置位该寄存器后，发射器会循环发送 RAM 中的脉冲编码。配置  $RMT\_TX\_LOOP\_CNT\_EN\_CHn$  后发射器会记录循环发送的次数。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。

置位  $RMT\_TX\_SIM\_EN$  可以使能发射器多个通道同步发送的功能，此时多个通道会同时启动发送。

$RMT\_TX\_SIM\_CHn$  用于选择同步发送的通道。

### 10.2.5 接收器

$RMT\_RX\_EN\_CHn$  置为 1 时接收器开始工作。接收器会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  $RMT\_IDLE\_THRES\_CHn$  时，接收器结束接收过程，返回空闲状态，并产生  $RMT\_CHn\_RX\_END\_INT$  中断。

每个通道都可以通过置位  $RMT\_RX\_FILTER\_EN\_CHn$  使能接收器对输入信号进行滤波的功能。滤波器的功能为连续采样输入信号，如果输入信号在连续  $RMT\_RX\_FILTER\_THRES\_CHn$  个 APB 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  $RMT\_RX\_FILTER\_THRES\_CHn$  个 APB 时钟周期的线路毛刺。

### 10.2.6 中断

- $RMT\_CHn\_ERR\_INT$ ：当通道  $n$  发生读写数据不正确，或内存空满错误时，即触发此中断。
- $RMT\_CHn\_TX\_THR\_EVENT\_INT$ ：发射器每发送  $RMT\_CHn\_TX\_LIM\_REG$  的数据，即触发一次此中断。
- $RMT\_CHn\_TX\_END\_INT$ ：当发射器停止发送信号时，即触发此中断。
- $RMT\_CHn\_RX\_END\_INT$ ：当接收器停止接收信号时，即触发此中断。
- $RMT\_CHn\_TX\_LOOP\_INT$ ：发射器处于循环发送模式时，当循环次数达到  $RMT\_TX\_LOOP\_NUM\_CHn$  的值后，会产生此中断。

## 10.3 基地址

用户可以通过两个不同的寄存器基地址访问 RMT，如表 41 所示。更多信息，请访问[章节 1 系统和存储器](#)。

表 41: RMT 基地址

访问总线	基地址
PeriBUS1	0x3F416000
PeriBUS2	0x60016000

## 10.4 寄存器列表

请注意，下表中的地址都是相对于 RMT 基地址的地址偏移量（相对地址）。更多有关 RMT 基地址的信息，请前往第 10.3 节。

名称	描述	地址	访问
<b>配置寄存器</b>			
<a href="#">RMT_CH0CONF0_REG</a>	通道 0 配置寄存器 0	0x0010	读/写
<a href="#">RMT_CH0CONF1_REG</a>	通道 0 配置寄存器 1	0x0014	不定
<a href="#">RMT_CH1CONF0_REG</a>	通道 1 配置寄存器 0	0x0018	读/写
<a href="#">RMT_CH1CONF1_REG</a>	通道 1 配置寄存器 1	0x001C	不定
<a href="#">RMT_CH2CONF0_REG</a>	通道 2 配置寄存器 0	0x0020	读/写
<a href="#">RMT_CH2CONF1_REG</a>	通道 2 配置寄存器 1	0x0024	不定
<a href="#">RMT_CH3CONF0_REG</a>	通道 3 配置寄存器 0	0x0028	读/写
<a href="#">RMT_CH3CONF1_REG</a>	通道 3 配置寄存器 1	0x002C	不定
<a href="#">RMT_APB_CONF_REG</a>	RMT APB 配置寄存器	0x0080	读/写
<a href="#">RMT_REF_CNT_RST_REG</a>	RMT 时钟分频器复位寄存器	0x0088	读/写
<a href="#">RMT_CH0_RX_CARRIER_RM_REG</a>	通道 0 载波清除寄存器	0x008C	读/写
<a href="#">RMT_CH1_RX_CARRIER_RM_REG</a>	通道 1 载波清除寄存器	0x0090	读/写
<a href="#">RMT_CH2_RX_CARRIER_RM_REG</a>	通道 2 载波清除寄存器	0x0094	读/写
<a href="#">RMT_CH3_RX_CARRIER_RM_REG</a>	通道 3 载波清除寄存器	0x0098	读/写
<b>载波占空比寄存器</b>			
<a href="#">RMT_CH0CARRIER_DUTY_REG</a>	通道 0 占空比配置寄存器	0x0060	读/写
<a href="#">RMT_CH1CARRIER_DUTY_REG</a>	通道 1 占空比配置寄存器	0x0064	读/写
<a href="#">RMT_CH2CARRIER_DUTY_REG</a>	通道 2 占空比配置寄存器	0x0068	读/写
<a href="#">RMT_CH3CARRIER_DUTY_REG</a>	通道 3 占空比配置寄存器	0x006C	读/写
<b>发送事件配置寄存器</b>			
<a href="#">RMT_CH0_TX_LIM_REG</a>	通道 0 Tx 配置寄存器	0x0070	不定
<a href="#">RMT_CH1_TX_LIM_REG</a>	通道 1 Tx 配置寄存器	0x0074	不定
<a href="#">RMT_CH2_TX_LIM_REG</a>	通道 2 Tx 配置寄存器	0x0078	不定
<a href="#">RMT_CH3_TX_LIM_REG</a>	通道 3 Tx 配置寄存器	0x007C	不定
<a href="#">RMT_TX_SIM_REG</a>	RMT 同步发送寄存器	0x0084	读/写
<b>状态寄存器</b>			
<a href="#">RMT_CH0STATUS_REG</a>	通道 0 状态寄存器	0x0030	只读

名称	描述	地址	访问
RMT_CH1STATUS_REG	通道 1 状态寄存器	0x0034	只读
RMT_CH2STATUS_REG	通道 2 状态寄存器	0x0038	只读
RMT_CH3STATUS_REG	通道 3 状态寄存器	0x003C	只读
RMT_CH0ADDR_REG	通道 0 地址寄存器	0x0040	只读
RMT_CH1ADDR_REG	通道 1 地址寄存器	0x0044	只读
RMT_CH2ADDR_REG	通道 2 地址寄存器	0x0048	只读
RMT_CH3ADDR_REG	通道 3 地址寄存器	0x004C	只读
<b>版本寄存器</b>			
RMT_DATE_REG	版本控制寄存器	0x00FC	读/写
<b>FIFO 读/写寄存器</b>			
RMT_CH0DATA_REG	配置 APB FIFO 对通道 0 进行读写操作	0x0000	只读
RMT_CH1DATA_REG	配置 APB FIFO 对通道 1 进行读写操作	0x0004	只读
RMT_CH2DATA_REG	配置 APB FIFO 对通道 2 进行读写操作	0x0008	只读
RMT_CH3DATA_REG	配置 APB FIFO 对通道 3 进行读写操作	0x000C	只读
<b>中断寄存器</b>			
RMT_INT_RAW_REG	原始中断状态寄存器	0x0050	只读
RMT_INT_ST_REG	隐蔽中断状态寄存器	0x0054	只读
RMT_INT_ENA_REG	中断使能寄存器	0x0058	读/写
RMT_INT_CLR_REG	中断清除寄存器	0x005C	只写

## 10.5 寄存器

Register 10.1: RMT\_CH $n$ CONF0\_REG ( $n$ : 0-3) (0x0010+8\* $n$ )

(reserved)				RMT_CARRIER_OUT_LV_CH $n$				RMT_CARRIER_EN_CH $n$				RMT_CARRIER_EFF_EN_CH $n$				RMT_MEM_SIZE_CH $n$				RMT_IDLE_THRES_CH $n$								RMT_DIV_CNT_CH $n$			
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	1	1	1	0x1	0x1000								0x2																Reset	

**RMT\_DIV\_CNT\_CH $n$**  用于配置通道  $n$  分频器的分频系数。(读/写)

**RMT\_IDLE\_THRES\_CH $n$**  当接收器长时间检测不到信号沿变化，即计数器的值大于等于此寄存器的值时，接收器结束接收过程。(读/写)

**RMT\_MEM\_SIZE\_CH $n$**  配置通道  $n$  可以使用的 block 数量，范围为 1 ~ 4- $n$ 。(读/写)

**RMT\_CARRIER\_EFF\_EN\_CH $n$**  1: 配置通道  $n$  仅在发送状态下对输出信号载波调制；0: 配置通道  $n$  对所有状态（空闲状态、从 RAM 中读取数据和发送数据阶段）均加载载波。仅在 **RMT\_CARRIER\_EN\_CH $n$**  为 1 时有效。(读/写)

**RMT\_CARRIER\_EN\_CH $n$**  通道  $n$  的载波调制使能位。1: 对输出信号进行载波调制；0: 禁止对输出信号进行载波调制。(读/写)

**RMT\_CARRIER\_OUT\_LV\_CH $n$**  用于配置通道  $n$  的载波调制方式。1'h0: 载波加载在输出信号低电平上；1'h1: 载波加载在输出信号高电平上。(读/写)

Register 10.2: RMT\_CH $n$ CONF1\_REG ( $n$ : 0-3) (0x0014+8\* $n$ )

(reserved)																RMT_TX_STOP_CH <sup>n</sup> RMT_IDLE_OUT_EN_CH <sup>n</sup> RMT_IDLE_OUT_LV_CH <sup>n</sup> RMT_REF_ALWAYS_ON_CH <sup>n</sup> RMT_CHK_RX_CARRIER_EN_CH <sup>n</sup>								RMT_RX_FILTER_THRES_CH <sup>n</sup> RMT_RX_FILTER_EN_CH <sup>n</sup> RMT_TX_CONTL_MODE_CH <sup>n</sup> RMT_MEM_OWNER_CH <sup>n</sup> RMT_APB_MEM_RST_CH <sup>n</sup> RMT_MEM_RD_RST_CH <sup>n</sup> RMT_MEM_WR_RST_CH <sup>n</sup> RMT_TX_START_CH <sup>n</sup>									
3121201918171615																876543210								Reset									
0000000000000000																0000000000000000								0xf	0010000000								

Reset

**RMT\_TX\_START\_CH $n$**  用于使能通道  $n$  的发射器。发射器开始发送数据。(读/写)

**RMT\_RX\_EN\_CH $n$**  用于使能通道  $n$  的接收器。接收器开始接收数据。(读/写)

**RMT\_MEM\_WR\_RST\_CH $n$**  用于复位通道  $n$  中接收器访问的 RAM 写地址。(只写)

**RMT\_MEM\_RD\_RST\_CH $n$**  用于复位通道  $n$  中发射器访问的 RAM 读地址。(只写)

**RMT\_MEM\_OWNER\_CH $n$**  标志通道  $n$  的 RAM 使用权。1'h1: 接收器有权使用该通道的 RAM; 1'h0: 发射器有权使用该通道的 RAM。(读/写)

**RMT\_TX\_CONTI\_MODE\_CH $n$**  设置通道  $n$  发送结束后, 发射器不进入空闲状态, 而是继续从第一个字节开始发送数据。(读/写)

**RMT\_RX\_FILTER\_EN\_CH $n$**  使能通道  $n$  接收器的滤波功能。(读/写)

**RMT\_RX\_FILTER\_THRES\_CH $n$**  接收模式下, 忽略宽度小于 RMT\_RX\_FILTER\_THRES\_CH $n$  个 APB 时钟周期的脉冲。(读/写)

**RMT\_CHK\_RX\_CARRIER\_EN\_CH $n$**  用于使能 RAM 读循环模式 (通道  $n$  中需使能载波调制模式)。(读/写)

**RMT\_REF\_ALWAYS\_ON\_CH $n$**  用于选择通道  $n$  的基础时钟。1'h1: APB\_CLK; 1'h0: REF\_TICK (读/写)

**RMT\_IDLE\_OUT\_LV\_CH $n$**  配置通道  $n$  处于空闲状态下的输出信号电平。(读/写)

**RMT\_IDLE\_OUT\_EN\_CH $n$**  通道  $n$  处于空闲状态下的输出使能位。(读/写)

**RMT\_TX\_STOP\_CH $n$**  通道  $n$  中发射器停止发送信息。(读/写)



### Register 10.3: RMT\_APB\_CONF\_REG (0x0080)

RMT\_CLK\_EN

(reserved)

RMT\_MEM\_FORCE\_PU

RMT\_MEM\_FORCE\_PD

RMT\_MEM\_CLK\_FORCE\_ON

RMT\_MEM\_TX\_WRAP\_EN

RMT\_APB\_FIFO\_MASK

31	30																							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	1	0	0

Reset

**RMT\_APB\_FIFO\_MASK** 1'h1: 直接访问 RAM; 1'h0: 经 APB FIFO 访问 RAM。(读/写)

**RMT\_MEM\_TX\_WRAP\_EN** 乒乓模式使能位。该模式下，当发射器发送的数据量大于 block 大小，发射器将继续从第一字节开始发送数据。（读/写）

**RMT\_MEM\_CLK\_FORCE\_ON** RMT 模块工作时, 使能 RAM 的时钟; RMT 模块不工作时, 关闭 RAM 的时钟, 从而实现低功耗。(读/写)

**RMT\_MEM\_FORCE\_PD** 降低 RMT RAM 的功耗。(读/写)

**RMT\_MEM\_FORCE\_PU** 1: 禁用 RMT RAM 的 Light-sleep 低功耗功能; 0: RMT 处于 Light-sleep 模式时, 设置 RAM 进入低功耗模式。(读/写)

**RMT\_CLK\_EN** RMT 寄存器的时钟门控使能位，用于 RMT 寄存器低功耗控制。1：使能 RMT 寄存器驱动时钟；0：禁用 RMT 寄存器驱动时钟。（读/写）

#### Register 10.4: RMT REF CNT RST REG (0x0088)

Diagram illustrating the RMT\_REF\_CNT\_RST\_CH0 register structure. The register is 32 bits wide. Bits 31-4 are reserved. Bits 3-0 are the RMT\_REF\_CNT\_RST\_CH0 field, which is split into four 1-bit fields: RMT\_REF\_CNT\_RST\_CH3 (bit 3), RMT\_REF\_CNT\_RST\_CH2 (bit 2), RMT\_REF\_CNT\_RST\_CH1 (bit 1), and RMT\_REF\_CNT\_RST\_CH0 (bit 0).

**RMT REF CNT RST CH<sub>*n*</sub>** 复位通道 *n* 的时钟分频器。(读/写)

Register 10.5: RMT\_CH $n$ \_RX\_CARRIER\_RM\_REG ( $n$ : 0-3) (0x008C+4\* $n$ )

RMT_CARRIER_HIGH_THRES_CH <sup>n</sup>																RMT_CARRIER_LOW_THRES_CH <sup>n</sup>																															
31																15																0															
0x00																0x00																Reset															

RMT\_CARRIER\_LOW\_THRES\_CH $n$  载波调制模式下，通道  $n$  低电平周期为 (RMT\_CARRIER\_LOW\_THRES\_CH $n$  + 1) 个时钟周期。(读/写)

RMT\_CARRIER\_HIGH\_THRES\_CH $n$  载波模式下，通道  $n$  高电平周期为 (RMT\_CARRIER\_HIGH\_THRES\_CH $n$  + 1) 个时钟周期。(读/写)

Register 10.6: RMT\_CH $n$ CARRIER\_DUTY\_REG ( $n$ : 0-3) (0x0060+4\* $n$ )

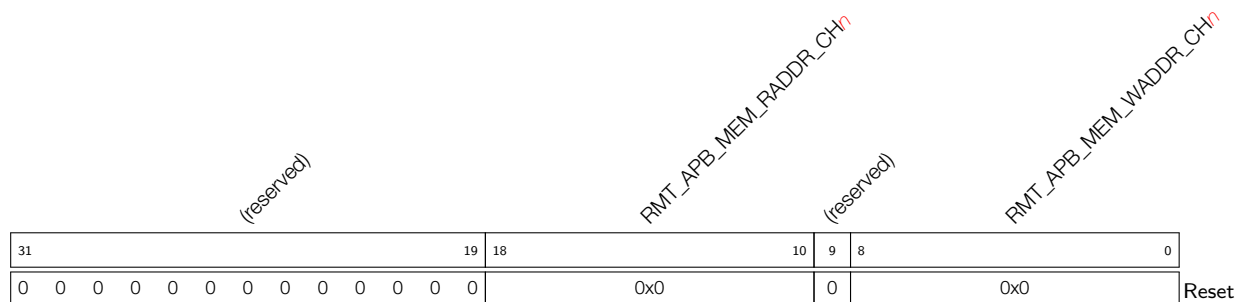
RMT_CARRIER_HIGH_CH <sup>n</sup>																RMT_CARRIER_LOW_CH <sup>n</sup>																																															
31																16																15																0															
0x40																0x40																Reset																															

RMT\_CARRIER\_LOW\_CH $n$  用于配置通道  $n$  载波的低电平时钟周期。(读/写)

RMT\_CARRIER\_HIGH\_CH $n$  用于配置通道  $n$  载波的高电平时钟周期。(读/写)



**Register 10.10: RMT\_CH $n$ ADDR\_REG ( $n$ : 0-3) (0x0040+4\* $n$ )**



**RMT\_APB\_MEM\_RD\_ERR\_CH<sub>n</sub>** 通过 APB 总线执行读操作时，如果偏移地址溢出 RAM block，将触发此状态位。（只读）

**RMT\_APB\_MEM\_RADDR\_CH<sub>n</sub>** 记录通过 APB 总线对 RAM 执行读操作时, 地址的偏移量。(只读)

Register 10.11: RMT\_DATE\_REG (0x00FC)

RMT_DATE																														
31																													0	
0x19072601																														Reset

RMT\_DATE 版本控制寄存器（读/写）

Register 10.12: RMT\_CH $n$ DATA\_REG ( $n$ : 0-3) (0x0000+4\* $n$ )

RMT_CH <sub>n</sub> _DATA_REG																															
31																															0
0x000000																															
Reset																															

RMT\_CH $n$ \_DATA\_REG 通过 APB FIFO 对通道  $n$  进行读写操作。（只读）

Register 10.13: RMT\_INT\_RAW\_REG (0x0050)

(reserved)																																RMT_CH3_TX_LOOP_INT_RAW																RMT_CH2_TX_LOOP_INT_RAW																RMT_CH1_TX_LOOP_INT_RAW																RMT_CH0_TX_LOOP_INT_RAW																RMT_CH3_TX_THR_EVENT_INT_RAW																RMT_CH2_TX_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_RX_END_INT_RAW																RMT_CH0_RX_END_INT_RAW																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															

RMT\_CH $n$ \_TX\_END\_INT\_RAW 通道  $n$  的原始中断状态位。发送结束即触发该中断。（只读）

RMT\_CH $n$ \_RX\_END\_INT\_RAW 通道  $n$  的原始中断状态位。接收结束即触发该中断。（只读）

RMT\_CH $n$ \_ERR\_INT\_RAW 通道  $n$  的原始中断状态位。发生错误即触发该中断。（只读）

RMT\_CH $n$ \_TX\_THR\_EVENT\_INT\_RAW 通道  $n$  的原始中断状态位。发送的数据量大于预先设定的值时即触发该中断。（只读）

RMT\_CH $n$ \_TX\_LOOP\_INT\_RAW 通道  $n$  的原始中断状态位。循环次数达到预先设定的阈值时即触发该中断。（只读）

Register 10.14: RMT\_INT\_ST\_REG (0x0054)

(reserved)																						RMT_CH3_TX_LOOP_INT_ST																RMT_CH2_TX_LOOP_INT_ST																RMT_CH1_TX_LOOP_INT_ST																RMT_CH0_TX_LOOP_INT_ST																RMT_CH3_TX_THR_EVENT_INT_ST																RMT_CH2_TX_THR_EVENT_INT_ST																RMT_CH1_TX_THR_EVENT_INT_ST																RMT_CH0_TX_THR_EVENT_INT_ST																RMT_CH3_RX_END_INT_ST																RMT_CH2_RX_END_INT_ST																RMT_CH1_RX_END_INT_ST																RMT_CH0_RX_END_INT_ST																RMT_CH3_ERR_INT_ST																RMT_CH2_ERR_INT_ST																RMT_CH1_ERR_INT_ST																RMT_CH0_ERR_INT_ST																RMT_CH3_TX_END_INT_ST																RMT_CH2_TX_END_INT_ST																RMT_CH1_TX_END_INT_ST																RMT_CH0_TX_END_INT_ST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																						
																						RMT_CH3_TX_LOOP_INT_ST																RMT_CH2_TX_LOOP_INT_ST																RMT_CH1_TX_LOOP_INT_ST																RMT_CH0_TX_LOOP_INT_ST																RMT_CH3_TX_THR_EVENT_INT_ST																RMT_CH2_TX_THR_EVENT_INT_ST																RMT_CH1_TX_THR_EVENT_INT_ST																RMT_CH0_TX_THR_EVENT_INT_ST																RMT_CH3_RX_END_INT_ST																RMT_CH2_RX_END_INT_ST																RMT_CH1_RX_END_INT_ST																RMT_CH0_RX_END_INT_ST																RMT_CH3_ERR_INT_ST																RMT_CH2_ERR_INT_ST																RMT_CH1_ERR_INT_ST																RMT_CH0_ERR_INT_ST																RMT_CH3_TX_END_INT_ST																RMT_CH2_TX_END_INT_ST																RMT_CH1_TX_END_INT_ST																RMT_CH0_TX_END_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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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 RMT\_CH $n$ \_TX\_END\_INT 的隐蔽中断状态位。(只读)

RMT\_CH $n$ \_RX\_END\_INT\_ST RMT\_CH $n$ \_RX\_END\_INT 的隐蔽中断状态位。(只读)

RMT\_CH $n$ \_ERR\_INT\_ST RMT\_CH $n$ \_ERR\_INT 的隐蔽中断状态位。(只读)

RMT\_CH $n$ \_TX\_THR\_EVENT\_INT\_ST RMT\_CH $n$ \_TX\_THR\_EVENT\_INT 的隐蔽中断状态位。(只读)

RMT\_CH $n$ \_TX\_LOOP\_INT\_ST RMT\_CH $n$ \_TX\_LOOP\_INT 的隐蔽中断状态位。(只读)

Register 10.15: RMT\_INT\_ENA\_REG (0x0058)

(reserved)																						RMT_CH3_TX_LOOP_INT_ENA																RMT_CH2_TX_LOOP_INT_ENA																RMT_CH1_TX_LOOP_INT_ENA																RMT_CH0_TX_LOOP_INT_ENA																RMT_CH3_TX_THR_EVENT_INT_ENA																RMT_CH2_TX_THR_EVENT_INT_ENA																RMT_CH1_TX_THR_EVENT_INT_ENA																RMT_CH0_TX_THR_EVENT_INT_ENA																RMT_CH3_RX_END_INT_ENA																RMT_CH2_RX_END_INT_ENA																RMT_CH1_RX_END_INT_ENA																RMT_CH0_RX_END_INT_ENA																RMT_CH3_TX_END_INT_ENA																RMT_CH2_TX_END_INT_ENA																RMT_CH1_TX_END_INT_ENA																RMT_CH0_TX_END_INT_ENA																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																									

Reset

RMT\_CH $n$ \_TX\_END\_INT\_ENA RMT\_CH $n$ \_TX\_END\_INT 中断使能位。(读/写)

RMT\_CH $n$ \_RX\_END\_INT\_ENA RMT\_CH $n$ \_RX\_END\_INT 中断使能位。(读/写)

RMT\_CH $n$ \_ERR\_INT\_ENA RMT\_CH $n$ \_ERR\_INT 中断使能位。(读/写)

RMT\_CH $n$ \_TX\_THR\_EVENT\_INT\_ENA RMT\_CH $n$ \_TX\_THR\_EVENT\_INT 中断使能位。(读/写)

RMT\_CH $n$ \_TX\_LOOP\_INT\_ENA RMT\_CH $n$ \_TX\_LOOP\_INT 中断使能位。(读/写)

Register 10.16: RMT\_INT\_CLR\_REG (0x005C)

(reserved)																																RMT_CH3_TX_LOOP_INT_CLR RMT_CH2_TX_LOOP_INT_CLR RMT_CH1_TX_LOOP_INT_CLR RMT_CH0_TX_LOOP_INT_CLR RMT_CH3_TX_THR_EVENT_INT_CLR RMT_CH2_TX_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_TX_END_INT_CLR RMT_CH2_TX_END_INT_CLR RMT_CH1_TX_END_INT_CLR RMT_CH0_TX_END_INT_CLR RMT_CH3_RX_END_INT_CLR RMT_CH2_RX_END_INT_CLR RMT_CH1_RX_END_INT_CLR RMT_CH0_RX_END_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	0	0	0	0	0	0	0	0	0	0	0	Reset											

**RMT\_CH<sub>n</sub>\_TX\_END\_INT\_CLR** 用于清除 **RMT\_CH<sub>n</sub>\_TX\_END\_INT** 中断。(只写)

**RMT\_CH<sub>n</sub>\_RX\_END\_INT\_CLR** 用于清除 **RMT\_CH<sub>n</sub>\_RX\_END\_INT** 中断。(只写)

**RMT\_CH<sub>n</sub>\_ERR\_INT\_CLR** 用于清除 **RMT\_CH<sub>n</sub>\_ERR\_INT** 中断。(只写)

**RMT\_CH<sub>n</sub>\_TX\_THR\_EVENT\_INT\_CLR** 用于清除 **RMT\_CH<sub>n</sub>\_TX\_THR\_EVENT\_INT** 中断。(只写)

**RMT\_CH<sub>n</sub>\_TX\_LOOP\_INT\_CLR** 用于清除 **RMT\_CH<sub>n</sub>\_TX\_LOOP\_INT** 中断。(只写)

## 11. 脉冲计数器

脉冲计数器 (Pulse Count Controller, PCNT) 用于对输入脉冲计数和产生中断，通过记录输入脉冲信号的上升沿或下降沿进行递增或递减计数。PCNT 有四个称为“单元”的独立脉冲计数器，这些单元拥有自己的寄存器。下文描述中  $n$  表示单元编号 0~3。

每个单元有两个通道 (ch0 和 ch1)，可以独立配置为递增或递减计数。两个通道功能相同，下文以通道 0 (ch0) 为例进行介绍。

如图 11-1 所示，每个通道有两个输入信号：

1. 一个控制信号（如 ctrl\_ch0\_ $n$  为 ch0 的控制信号）
2. 一个脉冲输入信号（如 sig\_ch0\_ $n$  为 ch0 的脉冲输入信号）

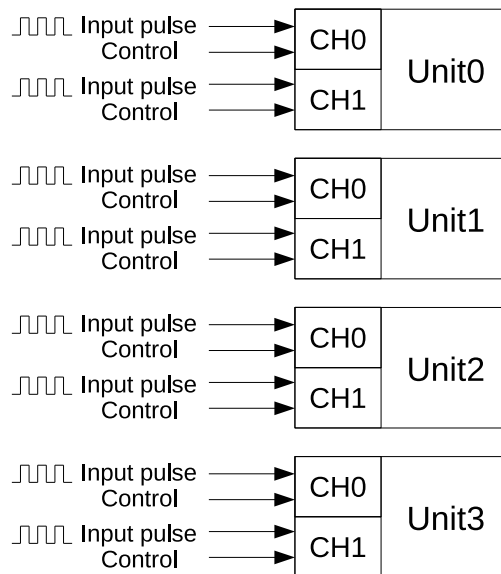


图 11-1. PCNT 框图

### 11.1 特性

PCNT 有如下特性：

- 四个脉冲计数器（单元），各自独立工作
- 每个单元有两个独立的通道，共用一个脉冲计数器
- 所有通道均有输入脉冲信号（如 sig\_ch0\_ $n$ ）和相应的控制信号（如 ctrl\_ch0\_ $n$ ）
- 滤波器独立工作，过滤每个单元的输入脉冲信号（sig\_ch0\_ $n$  和 sig\_ch1\_ $n$ ）和控制信号（ctrl\_ch0\_ $n$  和 ctrl\_ch1\_ $n$ ）
- 每个通道参数如下：
  1. 选择在输入脉冲信号的上升沿或下降沿计数
  2. 在控制信号为高电平或低电平时可将计数模式配置为递增、递减或停止计数。



## 11.2 功能描述

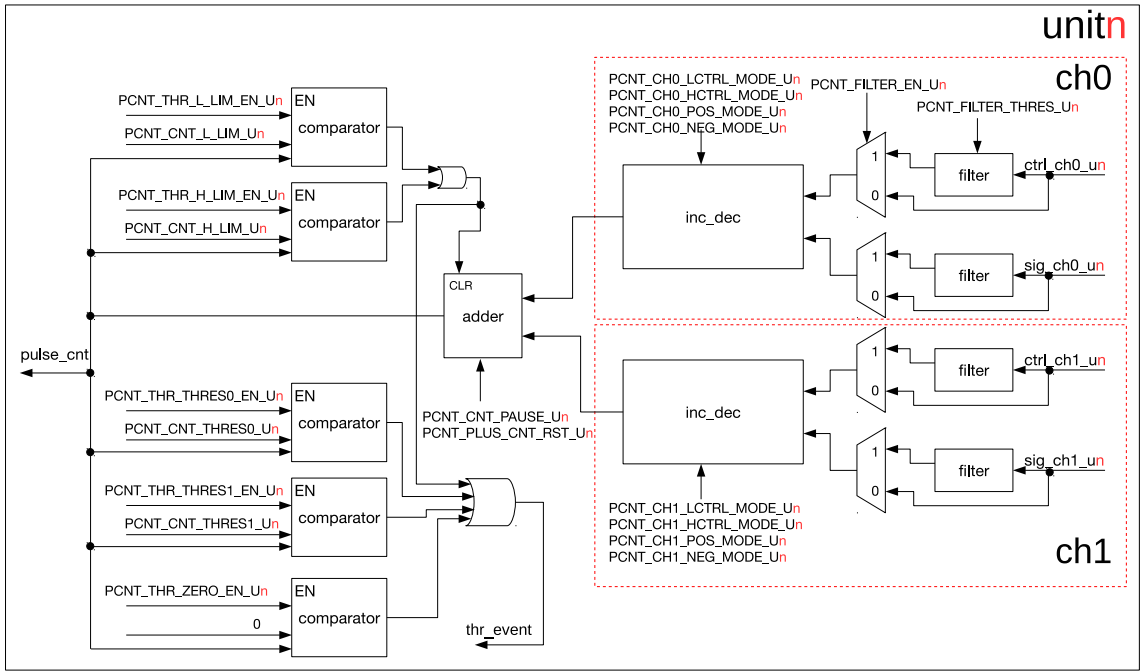


图 11-2. PCNT 单元基本架构图

图 11-2 为 PCNT 单元的基本架构图。如上所述，ctrl\_ch0\_un 为 ch0 的控制信号，当控制信号 ctrl\_ch0\_un 为高电平或低电平时，都可配置输入脉冲信号 sig\_ch0\_un 在上升沿和下降沿的不同计数模式。可选计数模式如下：

- 递增模式：通道检测到 sig\_ch0\_un 的有效边沿时，计数器的值 pulse\_cnt 加 1。pulse\_cnt 大于等于 PCNT\_CNT\_H\_LIM\_Un 时被清零。如果在 pulse\_cnt 达到 PCNT\_CNT\_H\_LIM\_Un 前，该通道的计数模式改变或置位 PCNT\_CNT\_PAUSE\_Un，则 pulse\_cnt 计数模式改变，且不被清零。
- 递减模式：通道检测到 sig\_ch0\_un 的有效边沿时，计数器的值 pulse\_cnt 减 1。pulse\_cnt 小于等于 PCNT\_CNT\_L\_LIM\_Un 时被清零。如果在 pulse\_cnt 达到 PCNT\_CNT\_H\_LIM\_Un 前，该通道的计数模式改变或置位 PCNT\_CNT\_PAUSE\_Un，则 pulse\_cnt 计数模式改变，且不被清零。
- 停止计数：计数停止，计数器的值 pulse\_cnt 保持不变。

表 43 至表 46 说明了如何配置通道 0 的计数模式。

表 43: 控制信号为低电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_Un	PCNT_CH0_LCTRL_MODE_Un	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 44: 控制信号为高电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_Un	PCNT_CH0_HCTRL_MODE_Un	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 45: 控制信号为低电平时输入脉冲信号下降沿的计数模式

PCNT_CH0_NEG_MODE_Un	PCNT_CH0_LCTRL_MODE_Un	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 46: 控制信号为高电平时输入脉冲信号下降沿的计数模式

PCNT_CH0_NEG_MODE_Un	PCNT_CH0_HCTRL_MODE_Un	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

每个单元均有一个滤波器，用于该单元的所有控制信号和输入脉冲信号。置位 `PCNT_FILTER_EN_Un` 位使能滤波器。滤波器监测信号，滤除脉冲宽度小于 `PCNT_FILTER_THRES_Un` 个 APB 时钟周期的线路毛刺。

如前文所述，每个单元有通道 0 和通道 1 两个通道，处理不同的输入脉冲信号，并通过各自的 inc\_dec 模块递增或递减计数值。之后，两个通道将计数值发送给 adder 模块。adder 模块是一个带符号位的 16 位加法器。软件可以通过置位 `PCNT_CNT_PAUSE_Un` 暂停 adder，也可以通过置位 `PCNT_PULSE_CNT_RST_Un` 清零 adder。

PCNT 可以设置五个观察点，五个观察点共用一个中断，可以通过每个观察点各自的中断使能信号开启或屏蔽中断。

- 最大计数值: 当 pulse\_cnt 大于等于 `PCNT_CNT_H_LIM_Un` 时，产生中断，同时 `PCNT_CNT_THR_H_LIM_LAT_Un` 为高。

- 最小计数值: 当 pulse\_cnt 小于等于 PCNT\_CNT\_L\_LIM\_Un 时, 产生中断, 同时 PCNT\_CNT\_THR\_L\_LIM\_LAT\_Un 为高。
- 两个中间阈值: 当 pulse\_cnt 等于 PCNT\_CNT\_THRES0\_Un 或者 PCNT\_CNT\_THRES1\_Un 时, 产生中断, 同时 PCNT\_CNT\_THR\_THRES0\_LAT\_Un 或 PCNT\_CNT\_THR\_THRES1\_LAT\_Un 为高。
- 零: 当 pulse\_cnt 等于 0 时, 产生中断, 同时 PCNT\_CNT\_THR\_ZERO\_LAT\_Un 有效。

## 11.3 应用实例

每个单元的通道 0 和通道 1 可配置为独立工作或一起工作。下文详细说明了通道 0 独自递增计数、通道 0 独自递减计数和两个通道一起递增计数的应用实例。本节中未详述的通道工作模式（如通道 1 递减或递减、双通道一增一减），可参考这三种模式。

### 11.3.1 通道 0 独自递增计数

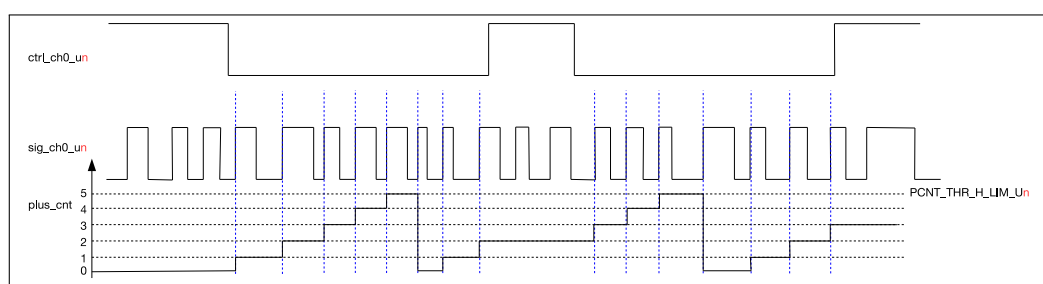


图 11-3. 通道 0 递增计数图

图 11-3 为通道 0 在 sig\_ch0\_un 上升沿独立递增计数的示意图, 此时通道 1 关闭（请参阅 11.2 一节查看如何关闭通道 1）。通道 0 的配置如下所示。

- PCNT\_CH0\_LCTRL\_MODE\_Un=0: 当 ctrl\_ch0\_un 为低电平时, 递增计数。
- PCNT\_CH0\_HCTRL\_MODE\_Un=2: 当 ctrl\_ch0\_un 为高电平时, 停止计数。
- PCNT\_CH0\_POS\_MODE\_Un=1: 在 sig\_ch0\_un 的上升沿递增计数。
- PCNT\_CH0\_NEG\_MODE\_Un=0: 在 sig\_ch0\_un 的下降沿不计数。
- PCNT\_CNT\_H\_LIM\_Un=5: pulse\_cnt 的值递增至 PCNT\_CNT\_H\_LIM\_Un 时被清零。

### 11.3.2 通道 0 独自递减计数

图 11-4 为通道 0 在 sig\_ch0\_un 上升沿独立递减计数的示意图, 此时通道 1 关闭。此时通道 0 的配置与图 11-3 相比有如下区别:

- PCNT\_CH0\_POS\_MODE\_Un=2: 即在 sig\_ch0\_un 的上升沿递减计数。
- PCNT\_CNT\_L\_LIM\_Un=-5: pulse\_cnt 的值递减到 PCNT\_CNT\_L\_LIM\_Un 时被清零。

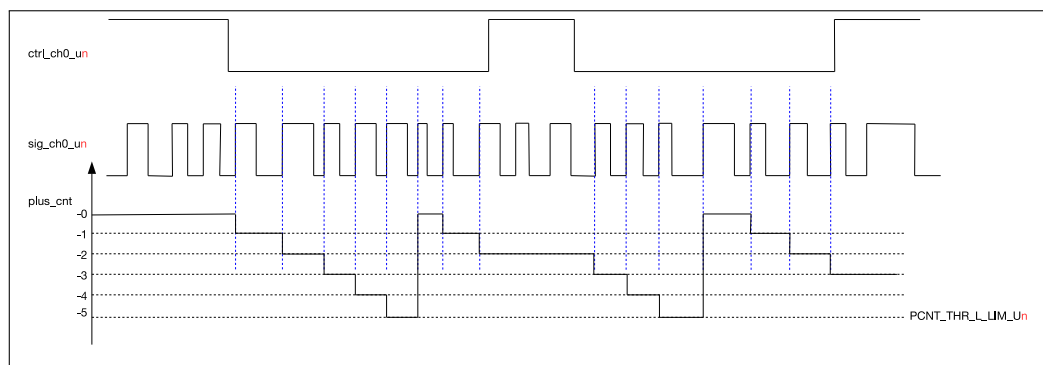


图 11-4. 通道 0 递减计数图

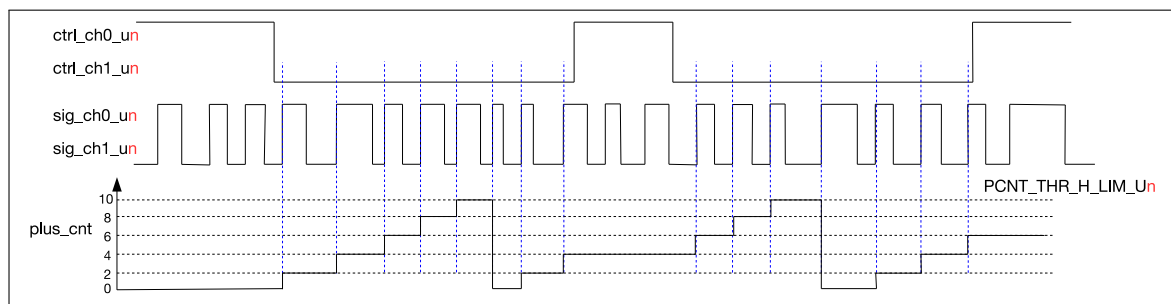


图 11-5. 双通道递增计数图

### 11.3.3 通道 0 和通道 1 同时递增计数

图 11-5 为通道 0 和通道 1 在 sig\_ch0\_un 和 sig\_ch1\_un 上升沿一同递增计数的示意图。如图 11-5 所示，控制信号 ctrl\_ch0\_un 与 ctrl\_ch1\_un 的波形一致，输入脉冲信号 sig\_ch0\_un 和 sig\_ch1\_un 波形一致。具体配置如下：

- 通道 0：
  - PCNT\_CH0\_LCTRL\_MODE\_Un=0：当 ctrl\_ch0\_un 为低电平时，递增计数。
  - PCNT\_CH0\_HCTRL\_MODE\_Un=2：当 ctrl\_ch0\_un 为高电平时，停止计数。
  - PCNT\_CH0\_POS\_MODE\_Un=1：在 sig\_ch0\_un 的上升沿递增计数。
  - PCNT\_CH0\_NEG\_MODE\_Un=0：在 sig\_ch0\_un 的下降沿不计数。
- 通道 1：
  - PCNT\_CH1\_LCTRL\_MODE\_Un=0：当 ctrl\_ch1\_un 为低电平时，递增计数。
  - PCNT\_CH1\_HCTRL\_MODE\_Un=2：当 ctrl\_ch1\_un 为高电平时，停止计数。
  - PCNT\_CH1\_POS\_MODE\_Un=1：在 sig\_ch1\_un 的上升沿递增计数。
  - PCNT\_CH1\_NEG\_MODE\_Un=0：在 sig\_ch1\_un 的下降沿不计数。
- PCNT\_CNT\_H\_LIM\_Un=10：pulse\_cnt 递增至 PCNT\_CNT\_H\_LIM\_Un 时被清零。

## 11.4 基地址

用户可以通过两个不同的寄存器基地址访问 PCNT，如表 47 所示。更多信息，请访问[章节 1 系统和存储器](#)。

表 47: PCNT 基地址

访问总线	基地址
PeriBUS1	0x3F417000
PeriBUS2	0x60017000

## 11.5 寄存器列表

请注意，下表中的地址都是相对于 PCNT 基地址的地址偏移量（相对地址）。更多有关 PCNT 基地址的信息，请前往 11.4 章节。

名称	描述	地址	访问
<b>配置寄存器</b>			
PCNT_U0_CONF0_REG	单元 0 的配置寄存器 0	0x0000	读/写
PCNT_U0_CONF1_REG	单元 0 的配置寄存器 1	0x0004	读/写
PCNT_U0_CONF2_REG	单元 0 的配置寄存器 2	0x0008	读/写
PCNT_U1_CONF0_REG	单元 1 的配置寄存器 0	0x000C	读/写
PCNT_U1_CONF1_REG	单元 1 的配置寄存器 1	0x0010	读/写
PCNT_U1_CONF2_REG	单元 1 的配置寄存器 2	0x0014	读/写
PCNT_U2_CONF0_REG	单元 2 的配置寄存器 0	0x0018	读/写
PCNT_U2_CONF1_REG	单元 2 的配置寄存器 1	0x001C	读/写
PCNT_U2_CONF2_REG	单元 2 的配置寄存器 2	0x0020	读/写
PCNT_U3_CONF0_REG	单元 3 的配置寄存器 0	0x0024	读/写
PCNT_U3_CONF1_REG	单元 3 的配置寄存器 1	0x0028	读/写
PCNT_U3_CONF2_REG	单元 3 的配置寄存器 2	0x002C	读/写
PCNT_CTRL_REG	所有计数器的控制寄存器	0x0060	读/写
<b>状态寄存器</b>			
PCNT_U0_CNT_REG	单元 0 的计数器值	0x0030	只读
PCNT_U1_CNT_REG	单元 1 的计数器值	0x0034	只读
PCNT_U2_CNT_REG	单元 2 的计数器值	0x0038	只读
PCNT_U3_CNT_REG	单元 3 的计数器值	0x003C	只读
PCNT_U0_STATUS_REG	脉冲计数器单元 0 的状态寄存器	0x0050	只读
PCNT_U1_STATUS_REG	脉冲计数器单元 1 的状态寄存器	0x0054	只读
PCNT_U2_STATUS_REG	脉冲计数器单元 2 的状态寄存器	0x0058	只读
PCNT_U3_STATUS_REG	脉冲计数器单元 3 的状态寄存器	0x005C	只读
<b>中断寄存器</b>			
PCNT_INT_RAW_REG	原始中断状态寄存器	0x0040	只读
PCNT_INT_ST_REG	中断状态寄存器	0x0044	只读
PCNT_INT_ENA_REG	中断使能寄存器	0x0048	读/写

名称	描述	地址	访问
<a href="#">PCNT_INT_CLR_REG</a>	中断清除寄存器	0x004C	只写
<b>版本寄存器</b>			
<a href="#">PCNT_DATE_REG</a>	脉冲计数器的版本控制寄存器	0x00FC	读/写

## 11.6 寄存器

Register 11.1: PCNT\_U $n$ \_CONF0\_REG ( $n$ : 0-3) (0x0000+12\* $n$ )

PCNT_CH1_LCTRL_MODE_U0				PCNT_CH1_HCTRL_MODE_U0				PCNT_CH1_POS_MODE_U0				PCNT_CH1_NEG_MODE_U0				PCNT_CH0_LCTRL_MODE_U0				PCNT_CH0_HCTRL_MODE_U0				PCNT_CH0_POS_MODE_U0				PCNT_CH0_NEG_MODE_U0				PCNT_THR_THRES1_EN_U0				PCNT_THR_THRES0_EN_U0				PCNT_THR_L_LIM_EN_U0				PCNT_THR_H_LIM_EN_U0				PCNT_FILTER_EN_U0				PCNT_FILTER_THRES_U0				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9													0																					
0x0				0x0				0x0				0x0				0x0				0				0				1				1				1				1				0x10												Reset

**PCNT\_FILTER\_THRES\_U $n$**  滤波器设置的最大阈值，以 APB\_CLK 时钟周期为单位。滤波器启动时，任何小于该值的脉冲都会被过滤。(读/写)

**PCNT\_FILTER\_EN\_U $n$**  单元  $n$  输入滤波器的使能位。(读/写)

**PCNT\_THR\_ZERO\_EN\_U $n$**  单元  $n$  过零比较器的使能位。(读/写)

**PCNT\_THR\_H\_LIM\_EN\_U $n$**  单元  $n$  上限比较器的使能位。(读/写)

**PCNT\_THR\_L\_LIM\_EN\_U $n$**  单元  $n$  下限比较器的使能位。(读/写)

**PCNT\_THR\_THRES0\_EN\_U $n$**  单元  $n$  阈值 0 比较器的使能位。(读/写)

**PCNT\_THR\_THRES1\_EN\_U $n$**  单元  $n$  阈值 1 比较器的使能位。(读/写)

**PCNT\_CH0\_NEG\_MODE\_U $n$**  用于设置通道 0 输入信号检测下降沿的工作模式。1: 增加计数器；2: 减少计数器；0、3: 对计数器无任何影响。(读/写)

**PCNT\_CH0\_POS\_MODE\_U $n$**  用于设置通道 0 输入信号检测上升沿的工作模式。1: 增加计数器；2: 减少计数器；0、3: 对计数器无任何影响。(读/写)

**PCNT\_CH0\_HCTRL\_MODE\_U $n$**  控制信号为高电平时，用于改变 CH $n$ \_POS\_MODE 和 CH $n$ \_NEG\_MODE 的设置。0: 不做修改；1: 反转（增加转为减少，减少转为增加）；2、3: 禁止计数器修改。(读/写)

**PCNT\_CH0\_LCTRL\_MODE\_U $n$**  控制信号为低电平时，用于改变 CH $n$ \_POS\_MODE 和 CH $n$ \_NEG\_MODE 的设置。0: 不做修改；1: 反转（增加转为减少，减少转为增加）；2、3: 禁止计数器修改。(读/写)

**PCNT\_CH1\_NEG\_MODE\_U $n$**  用于设置通道 1 输入信号检测下降沿的工作模式。1: 计数器递增；2: 计数器递减；0、3: 对计数器无任何影响。(读/写)

**PCNT\_CH1\_POS\_MODE\_U $n$**  用于设置通道 1 输入信号检测上升沿的工作模式。1: 计数器递增；2: 计数器递减；0、3: 对计数器无任何影响。(读/写)

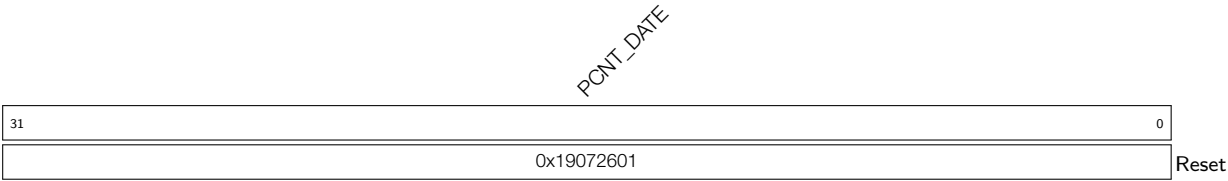
**PCNT\_CH1\_HCTRL\_MODE\_U $n$**  控制信号为高电平时，用于改变 CH $n$ \_POS\_MODE 和 CH $n$ \_NEG\_MODE 的设置。0: 不做修改；1: 反转（增加转为减少，减少转为增加）；2、3: 禁止计数器修改。(读/写)

**PCNT\_CH1\_LCTRL\_MODE\_U $n$**  控制信号为低电平时，用于改变 CH $n$ \_POS\_MODE 和 CH $n$ \_NEG\_MODE 的设置。0: 不做修改；1: 反转（增加转为减少，减少转为增加）；2、3: 禁止计数器修改。(读/写)



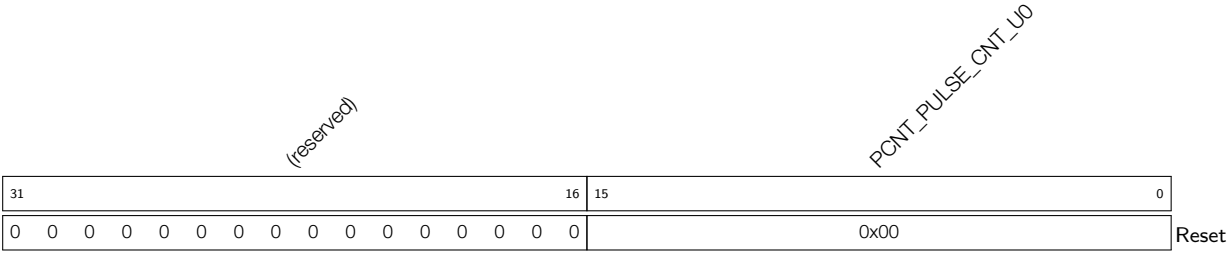


Register 11.5: PCNT\_DATE\_REG (0x00FC)



PCNT\_DATE 脉冲计数器的版本控制寄存器。(读/写)

Register 11.6: PCNT\_U<sub>n</sub>\_CNT\_REG (*n*: 0-3) (0x0030+4\**n*)



PCNT\_PULSE\_CNT\_U<sub>n</sub> 存储单元 *n* 脉冲计数器的当前值。(只读)

Register 11.7: PCNT\_U $n$ \_STATUS\_REG ( $n$ : 0-3) (0x0050+4\* $n$ )

(reserved)																								PCNT_CNT_THR_ZERO_LAT_U0 PCNT_CNT_THR_H_LIM_LAT_U0 PCNT_CNT_THR_L_LIM_LAT_U0 PCNT_CNT_THR_THRES0_LAT_U0 PCNT_CNT_THR_THRES1_LAT_U0 PCNT_CNT_THR_ZERO_MODE_U0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**PCNT\_CNT\_THR\_ZERO\_MODE\_U $n$**  PCNT\_U $n$  为 0 时的脉冲计数器状态。0: 脉冲计数器的值由正数减至 0。1: 脉冲计数器的值由负数增至 0。2: 脉冲计数器为负。3: 脉冲计数器为正。(只读)

**PCNT\_CNT\_THR\_THRES1\_LAT\_U $n$**  阈值中断有效时, PCNT\_U $n$  阈值 1 的锁存值。1: 脉冲计数器的当前值与阈值 1 相等, 阈值 1 有效。0: 其他。(只读)

**PCNT\_CNT\_THR\_THRES0\_LAT\_U $n$**  阈值中断有效时, PCNT\_U $n$  阈值 0 的锁存值。1: 脉冲计数器的当前值与阈值 0 相等, 阈值 0 有效。0: 其他。(只读)

**PCNT\_CNT\_THR\_L\_LIM\_LAT\_U $n$**  阈值中断有效时, PCNT\_U $n$  下限的锁存值。1: 脉冲计数器的当前值与下限阈值相等, 下限有效。0: 其他。(只读)

**PCNT\_CNT\_THR\_H\_LIM\_LAT\_U $n$**  阈值中断有效时, PCNT\_U $n$  上限的锁存值。1: 脉冲计数器的当前值与上限阈值相等, 上限有效。0: 其他。(只读)

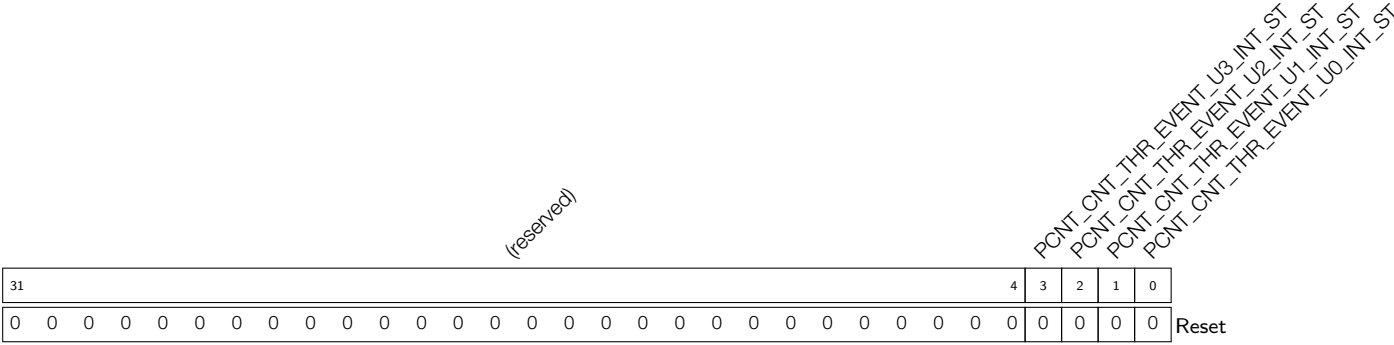
**PCNT\_CNT\_THR\_ZERO\_LAT\_U $n$**  阈值中断有效时, PCNT\_U $n$  阈值 0 的锁存值。1: 脉冲计数器的当前值为 0, 阈值 0 有效。0: 其他。(只读)

Register 11.8: PCNT\_INT\_RAW\_REG (0x0040)

(reserved)																												PCNT_CNT_THR_EVENT_U3_INT_RAW PCNT_CNT_THR_EVENT_U2_INT_RAW PCNT_CNT_THR_EVENT_U1_INT_RAW PCNT_CNT_THR_EVENT_U0_INT_RAW																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																		
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

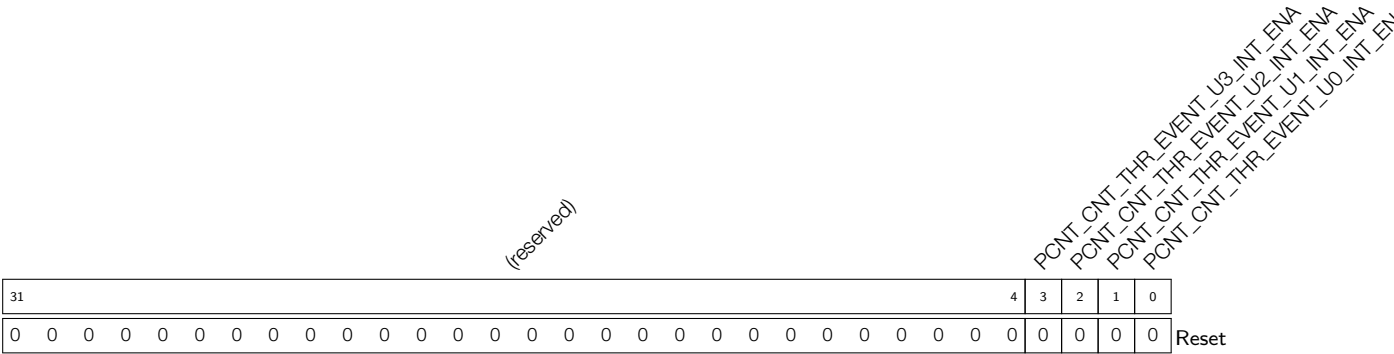
**PCNT\_CNT\_THR\_EVENT\_U $n$ \_INT\_RAW** 单元  $n$  事件中断的原始中断状态位。(只读)

Register 11.9: PCNT\_INT\_ST\_REG (0x0044)



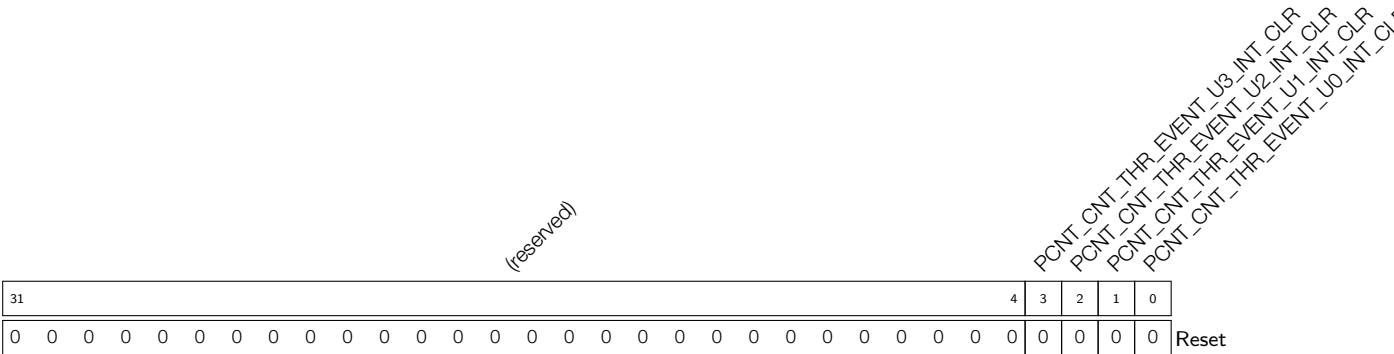
PCNT\_CNT\_THR\_EVENT\_U $n$ \_INT\_ST 单元  $n$  事件中断的屏蔽中断状态位。(只读)

Register 11.10: PCNT\_INT\_ENA\_REG (0x0048)



PCNT\_CNT\_THR\_EVENT\_U $n$ \_INT\_ENA 单元  $n$  事件中断的中断使能位。(读/写)

Register 11.11: PCNT\_INT\_CLR\_REG (0x004C)



PCNT\_CNT\_THR\_EVENT\_U $n$ \_INT\_CLR 置位此位，清除单元  $n$  事件中断。(只写)

## 12. 64 位定时器

### 12.1 概述

通用定时器可用于准确设定时间间隔、在一定间隔后触发中断（周期或非周期的）或充当硬件时钟。如图12-1所示，ESP32-S2 包含两个定时器组，即定时器组 0 和定时器组 1。每个定时器组有两个通用定时器（下文用  $T_x$  表示， $x$  为 0 或 1）和一个主系统看门狗定时器。所有通用定时器均基于 16 位预分频器和 64 位可自动重新加载向上 / 向下计数器。

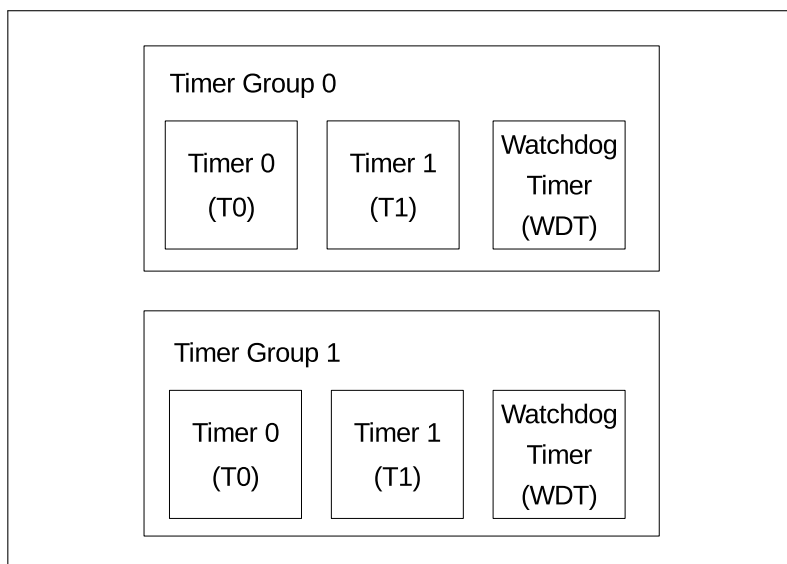


图 12-1. 定时器组

本章包含主系统看门狗定时器的寄存器描述，其功能描述请参阅章节 13 看门狗定时器。本章中“定时器”指代通用定时器。

定时器具有如下功能：

- 16 位时钟预分频器，分频系数为 1-65536
- 64 位时基计数器可配置成递增或递减
- 可读取时基计数器的实时值
- 暂停和恢复时基计数器器
- 可配置的报警产生机制
- 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）
- 电平触发中断和边沿触发中断机制

## 12.2 功能描述

### 12.2.1 16 位预分频器与时钟选择

每个定时器可通过配置寄存器 `TIMG_TxCONFIG_REG` 的 `TIMG_Tx_USE_XTAL` 字段，选择 APB 时钟 (APB\_CLK) 或外部时钟 (XTAL\_CLK) 作为源时钟。源时钟经 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 位预分频器可通过 `TIMG_Tx_DIVIDER` 器字段配置，选取从 1 到 65536 之间的任意值。注意，将 `TIMG_Tx_DIVIDER` 置 0 后，分频系数会变为 65536。定时器必须关闭（即 `TIMG_Tx_EN` 必须清零），才能更改 16 位预分频器。在定时器使能时更改 16 位预分频器会造成不可预知的结果。

### 12.2.2 64 位时基计数器

64 位时基计数器基于 TB\_CLK，可通过 `TIMG_Tx_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_Tx_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，`TIMG_Tx_INCREASE` 置位后，`TIMG_Tx_EN` 字段还可以更改，时基计数器可立即改变计数方向。

时基计数器 64 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）。在 `TIMG_TxUPDATE_REG` 上写任意值，64 位定时器的值可立即锁入寄存器 `TIMG_TxLO_REG` 和 `TIMG_TxHI_REG`，两个寄存器分别锁存低 32 位和高 32 位。在 `TIMG_TxUPDATE_REG` 写入新值之前，寄存器 `TIMG_TxLO_REG` 和 `TIMG_TxHI_REG` 将保持不变，以便 CPU 读值。

### 12.2.3 报警产生

配置后，定时器的当前值与报警值相同时可触发报警。报警会产生中断，（可选择）让定时器的当前值自动重新加载（详见第 12.2.4 节）。

64 位报警值可在 `TIMG_TxALARMLO_REG` 和 `TIMG_TxALARMHI_REG` 配置，两者分别代表报警值的低 32 位和高 32 位。但是，只有置位 `TIMG_Tx_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），定时器的当前值高于向上计数器或低于向下计数器的报警值时也会立即触发报警。

报警时，`TIMG_Tx_ALARM_EN` 字段自动清零，在置位 `TIMG_Tx_ALARM_EN` 前不会再次报警。

### 12.2.4 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 32 位分别更新为寄存器 `TIMG_Tx_LOAD_LO` 和 `TIMG_Tx_LOAD_HI` 存储的重新加载值。但是，把重新加载值写入 `TIMG_Tx_LOAD_LO` 和 `TIMG_Tx_LOAD_HI` 寄存器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 在寄存器上 `TIMG_TxLOAD_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。如置位 `TIMG_Tx_EN`，定时器会继续从新数值开始递增或递减计数。如清零 `TIMG_Tx_EN`，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。`TIMG_Tx_AUTORELOAD` 字段置 1 可以使能报警时自动重新加载。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。

### 12.2.5 中断

每个定时器都有一组连接至 CPU 的中断线（针对边沿中断和电平中断）。因此，每组定时器共有 6 个中断，命名如下：

- TIMG\_WDT\_LEVEL\_INT：组内看门狗定时器的电平中断，在看门狗定时器中断阶段超时后产生
- TIMG\_WDT\_EDGE\_INT：组内看门狗定时器的边沿中断，在看门狗定时器中断阶段超时后产生
- TIMG\_T<sub>x</sub>\_LEVEL\_INT：通用定时器的电平中断，在报警时产生
- TIMG\_T<sub>x</sub>\_EDGE\_INT：通用定时器的边沿中断，在报警时产生

中断在报警（或看门狗定时器阶段超时）时触发。报警（或阶段超时）后，电平中断线会被拉高，直至手动清除。边沿中断则会在报警（或阶段超时）后产生一个短脉冲。要使能定时器的电平中断或边沿中断，应分别置位 TIMG\_T<sub>x</sub>\_LEVEL\_INT\_EN 或 TIMG\_T<sub>x</sub>\_EDGE\_INT\_EN 位。

每个定时器组的中断由一组寄存器控制。组内的每个定时器在该组寄存器中都有对应的位：

- TIMG\_T<sub>x</sub>\_INT\_RAW：报警时置 1。该位在写值到对应的 TIMG\_T<sub>x</sub>\_INT\_CLR 位后才会被清零。
- TIMG\_WDT\_INT\_RAW：阶段超时时置 1。该位在写值到对应的 TIMG\_WDT\_INT\_CLR 位后才会被清零。
- TIMG\_T<sub>x</sub>\_INT\_ST：反映每个定时器中断的状态，通过用 TIMG\_T<sub>x</sub>\_INT\_ENA 屏蔽 TIMG\_T<sub>x</sub>\_INT\_RAW 位来生成。对于电平中断而言，TIMG\_T<sub>x</sub>\_INT\_ST 反映了 TIMG\_T<sub>x</sub>\_INT\_RAW 的电平。
- TIMG\_WDT\_INT\_ST：反映每个看门狗定时器中断的状态，通过用 TIMG\_WDT\_INT\_ENA 屏蔽 TIMG\_WDT\_INT\_RAW 位来生成。对于电平中断而言，TIMG\_WDT\_INT\_ST 反映了 TIMG\_WDT\_INT\_RAW 的电平。
- TIMG\_T<sub>x</sub>\_INT\_ENA：用于使能或屏蔽组内定时器的中断状态位。
- TIMG\_WDT\_INT\_ENA：用于使能或屏蔽组内看门狗定时器的中断状态位。
- TIMG\_T<sub>x</sub>\_INT\_CLR：置 1 此位清除定时器中断，定时器对应的 TIMG\_T<sub>x</sub>\_INT\_RAW 和 TIMG\_T<sub>x</sub>\_INT\_ST 位会清零。注意，使用电平中断前，必须清除定时器中断。
- TIMG\_WDT\_INT\_CLR：置 1 此位清除定时器中断，看门狗定时器对应的 TIMG\_WDT\_INT\_RAW 和 TIMG\_WDT\_INT\_ST 位会清零。注意，使用电平中断前，必须清除看门狗定时器中断。

## 12.3 配置与使用

### 12.3.1 定时器用作简单时钟

1. 配置时基计数器。

- 置位 TIMG\_T<sub>x</sub>\_USE\_XTAL 字段选择源时钟。
- 置位 TIMG\_T<sub>x</sub>\_DIVIDER 配置 16 位预分频器。
- 置位或清除 TIMG\_T<sub>x</sub>\_INCREASE 配置定时器方向。
- 在 TIMG\_T<sub>x</sub>\_LOAD\_LO 和 TIMG\_T<sub>x</sub>\_LOAD\_HI 上写初始值设置定时器的初始值，然后在 TIMG\_T<sub>x</sub>\_LOAD\_REG 上写任意值将初始值重新加载进定时器。

2. 置位 `TIMG_Tx_EN` 开启定时器。
3. 获得定时器的当前值。
  - 在 `TIMG_TxUPDATE_REG` 上写任意值锁存定时器的当前值。
  - 从 `TIMG_TxLO_REG` 和 `TIMG_TxHI_REG` 读取锁存的定时器值。

### 12.3.2 定时器用于一次性报警

1. 按照第 12.3.1 节的第 1 步配置时基计数器。
2. 配置报警。
  - 置位 `TIMG_TxALARMLO_REG` 和 `TIMG_TxALARMHI_REG` 配置报警值。
  - 置位 `TIMG_Tx_LEVEL_INT_EN` 或 `TIMG_Tx_EDGE_INT_EN` 分别使能电平中断和边沿中断。
3. 清零 `TIMG_Tx_AUTORELOAD` 关闭自动重新加载。
4. 置位 `TIMG_Tx_EN` 开启定时器。
5. 处理报警中断。
  - 置位定时器对应的 `TIMG_Tx_INT_CLR` 位清除中断。
  - 清零 `TIMG_Tx_EN` 关闭定时器。

### 12.3.3 定时器用于周期性报警

1. 按照第 12.3.1 节的第 1 步配置时基计数器。
2. 按照第 12.3.2 节的第 2 步配置报警。
3. 置位 `TIMG_Tx_AUTORELOAD` 使能自动重新加载，将重新加载值写入 `TIMG_Tx_LOAD_LO` 和 `TIMG_Tx_LOAD_HI`。
4. 置位 `TIMG_Tx_EN` 开启定时器。
5. 处理报警中断（每次报警时重复）。
  - 置位定时器对应的 `TIMG_Tx_INT_CLR` 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_TxALARMLO_REG`、`TIMG_TxALARMHI_REG`、`TIMG_Tx_LOAD_LO` 和 `TIMG_Tx_LOAD_HI`。否则，上述寄存器应保持不变。
  - 置位 `TIMG_Tx_ALARM_EN` 重新使能报警。
- 6.（最后一次报警时）关闭定时器。
  - 置位定时器对应的 `TIMG_Tx_INT_CLR` 位清除中断。
  - 清零 `TIMG_Tx_EN` 关闭定时器。

## 12.4 基地址

用户可通过四个不同的寄存器基地址访问 64 位定时器，如表 49 所示。更多信息，请访问[章节 1 系统和存储器](#)。

表 49: 64 位定时器基地址

	访问总线	基地址
TIMG0	PeriBUS1	0x3F41F000
	PeriBUS2	0x6001F000
TIMG1	PeriBUS1	0x3F420000
	PeriBUS2	0x60020000

## 12.5 寄存器列表

请注意，下表的地址指相对于 64 位定时器基地址的偏移量（相对地址）。请参阅第 12.4 节获取有关 64 位定时器基地址的信息。

名称	描述	地址	访问
<b>定时器 0 配置和控制寄存器</b>			
<a href="#">TIMG_T0CONFIG_REG</a>	定时器 0 配置寄存器	0x0000	读/写
<a href="#">TIMG_T0LO_REG</a>	定时器 0 的当前值，低 32 位	0x0004	只读
<a href="#">TIMG_T0HI_REG</a>	定时器 0 的当前值，高 32 位	0x0008	只读
<a href="#">TIMG_T0UPDATE_REG</a>	将当前定时器的值写到 <a href="#">TIMG_T0LO_REG</a> 和 <a href="#">TIMG_T0HI_REG</a>	0x000C	读/写
<a href="#">TIMG_T0ALARMLO_REG</a>	定时器 0 的报警值，低 32 位	0x0010	读/写
<a href="#">TIMG_T0ALARMHI_REG</a>	定时器 0 的报警值，高位	0x0014	读/写
<a href="#">TIMG_T0LOADLO_REG</a>	定时器 0 的重新加载值，低 32 位	0x0018	读/写
<a href="#">TIMG_T0LOADHI_REG</a>	定时器 0 的重新加载值，高 32 位	0x001C	读/写
<a href="#">TIMG_T0LOAD_REG</a>	在 <a href="#">TIMG_T0LOADLO_REG</a> 和 <a href="#">TIMG_T0LOADHI_REG</a> 上写值重新加载定时器	0x0020	只写
<b>定时器 1 配置和控制寄存器</b>			
<a href="#">TIMG_T1CONFIG_REG</a>	定时器 1 配置寄存器	0x0024	读/写
<a href="#">TIMG_T1LO_REG</a>	定时器 1 的当前值，低 32 位	0x0028	只读
<a href="#">TIMG_T1HI_REG</a>	定时器 1 的当前值，高 32 位	0x002C	只读
<a href="#">TIMG_T1UPDATE_REG</a>	将当前定时器的值写到 <a href="#">TIMG_T1LO_REG</a> 和 <a href="#">TIMG_T1HI_REG</a>	0x0030	读/写
<a href="#">TIMG_T1ALARMLO_REG</a>	定时器 1 的报警值，低 32 位	0x0034	读/写
<a href="#">TIMG_T1ALARMHI_REG</a>	定时器 1 的报警值，高位	0x0038	读/写
<a href="#">TIMG_T1LOADLO_REG</a>	定时器 1 的重新加载值，低 32 位	0x003C	读/写
<a href="#">TIMG_T1LOADHI_REG</a>	定时器 1 的重新加载值，高 32 位	0x0040	读/写
<a href="#">TIMG_T1LOAD_REG</a>	在 <a href="#">TIMG_T1LOADLO_REG</a> 和 <a href="#">TIMG_T1LOADHI_REG</a> 上写值重新加载定时器	0x0044	只写



名称	描述	地址	访问
<b>看门狗定时器配置和控制寄存器</b>			
TIMG_WDTCONFIG0_REG	看门狗定时器配置寄存器	0x0048	读/写
TIMG_WDTCONFIG1_REG	看门狗定时器预分频器寄存器	0x004C	读/写
TIMG_WDTCONFIG2_REG	看门狗定时器阶段 0 超时值	0x0050	读/写
TIMG_WDTCONFIG3_REG	看门狗定时器阶段 1 超时值	0x0054	读/写
TIMG_WDTCONFIG4_REG	看门狗定时器阶段 2 超时值	0x0058	读/写
TIMG_WDTCONFIG5_REG	看门狗定时器阶段 3 超时值	0x005C	读/写
TIMG_WDTFEED_REG	写值驱动看门狗定时器	0x0060	只写
TIMG_WDTWPROTECT_REG	看门狗写保护寄存器	0x0064	读/写
<b>RTC CALI 配置和控制寄存器</b>			
TIMG_RTCCALICFG2_REG	定时器组校准寄存器	0x00A8	不定
<b>中断寄存器</b>			
TIMG_INT_ENA_TIMERS_REG	中断使能位	0x0098	读/写
TIMG_INT_RAW_TIMERS_REG	原始中断状态	0x009C	只读
TIMG_INT_ST_TIMERS_REG	屏蔽中断状态	0x00A0	只读
TIMG_INT_CLR_TIMERS_REG	中断清除位	0x00A4	只写
<b>版本寄存器</b>			
TIMG_TIMERS_DATE_REG	版本控制寄存器	0x00F8	读/写
<b>配置寄存器</b>			
TIMG_REGCLK_REG	定时器组时钟门控寄存器	0x00FC	读/写

## 12.6 寄存器

Register 12.1: TIMG\_T $\times$ CONFIG\_REG ( $\times$ : 0-1) (0x0000+36\* $\times$ )

TIMG_T <del>x</del> _EN TIMG_T <del>x</del> _INCREASE TIMG_T <del>x</del> _AUTORELOAD				TIMG_T <del>x</del> _DIVIDER								TIMG_T <del>x</del> _EDGE_INT_EN TIMG_T <del>x</del> _LEVEL_INT_EN TIMG_T <del>x</del> _ALARM_EN TIMG_T <del>x</del> _USE_XTAL								(reserved)								
31	30	29	28	13								12	11	10	9	8	0											
0	1	1	0x01								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

Reset

**TIMG\_T $\times$ \_USE\_XTAL** 2: 使用 XTAL\_CLK 作为定时器组的源时钟。0: 使用 APB\_CLK 作为定时器组的源时钟。(读/写)

**TIMG\_T $\times$ \_ALARM\_EN** 置 1 后, 报警使能。报警使能后, 此位自动清零。(读/写)

**TIMG\_T $\times$ \_LEVEL\_INT\_EN** 置 1 后, 报警触发电平中断。(读/写)

**TIMG\_T $\times$ \_EDGE\_INT\_EN** 置 1 后, 报警触发边沿中断。(读/写)

**TIMG\_T $\times$ \_DIVIDER** 定时器  $\times$  时钟 (T $\times$ \_clk) 的预分频器值。(读/写)

**TIMG\_T $\times$ \_AUTORELOAD** 置 1 后, 定时器  $\times$  报警时自动重新加载使能。(读/写)

**TIMG\_T $\times$ \_INCREASE** 置 1 后, 定时器  $\times$  的时基计数器在每个时钟周期递增计数。清零后, 定时器  $\times$  的时基计数器递减计数。(读/写)

**TIMG\_T $\times$ \_EN** 置 1 后, 定时器  $\times$  时基计数器使能。(读/写)

Register 12.2: TIMG\_T $\times$ LO\_REG ( $\times$ : 0-1) (0x0004+36\* $\times$ )

TIMG_Tx_LO																															
31																															0
0x000000																															
Reset																															

Reset

**TIMG\_T $\times$ \_LO** 在 TIMG\_T $\times$ UPDATE\_REG 上写值后, 可读取定时器  $\times$  时基计数器的低 32 位。(只读)

Register 12.3: TIMG\_T $\times$ HI\_REG ( $\times$ : 0-1) (0x0008+36\* $\times$ )

TIMG_T <del>X</del> _HI																															
31																															0
0x000000																															
Reset																															

Reset

**TIMG\_T $\times$ \_HI** 在 TIMG\_T $\times$ UPDATE\_REG 上写值后, 可读取定时器  $\times$  时基计数器的高 32 位。(只读)

TIMING\_TX\_UPDATE

(reserved)

**TIMG\_Tx\_UPDATE** 在 TIMG\_TxUPDATE\_REG 上写 0 或 1，计数器的值被锁存 (读/写)

TIMING\_ALARM\_LO

**TIMG\_Tx\_ALARM\_LO** 定时器  $x$  时基计数器低 32 位的警报触发值。(读/写)

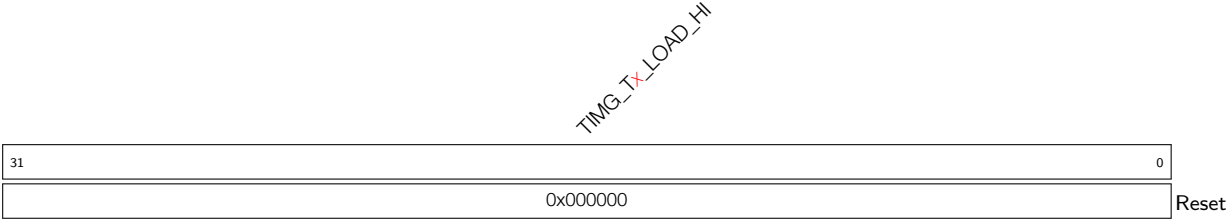
TIMG\_T<sub>x</sub>\_ALARM\_HI

**TIMG\_Tx\_ALARM\_HI** 定时器 x 时基计数器高 32 位的警报触发值。(读/写)

TIMG\_T~~x~~\_LOAD\_LO

**TIMG\_T<sub>x</sub>\_LOAD\_LO** 定时器 x 时基计数器低 32 位的重新加载值。(读/写)

Register 12.8: TIMG\_T<sub>x</sub>LOADHI\_REG (x: 0-1) (0x001C+36\*x)



**TIMG\_T<sub>x</sub>LOAD\_HI** 定时器 x 时基计数器高 32 位的重新加载值。(读/写)

Register 12.9: TIMG\_T<sub>x</sub>LOAD\_REG (x: 0-1) (0x0020+36\*x)



**TIMG\_T<sub>x</sub>LOAD** 写任意值重新加载定时器 x 时基计数器。(只写)

Register 12.10: TIMG\_WDTCONFIG0\_REG (0x0048)

TIMG_WDT_EN																															TIMG_WDT_STG0																															TIMG_WDT_STG1																															TIMG_WDT_STG2																															TIMG_WDT_STG3																															TIMG_WDT_EDGE_INT_EN																															TIMG_WDT_LEVEL_INT_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																					0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
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	0	0	0	0	0	0	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** WDT 复位 CPU 使能。(读/写)

**TIMG\_WDT\_FLASHBOOT\_MOD\_EN** 置 1 后, flash 启动保护使能。(读/写)

**TIMG\_WDT\_SYS\_RESET\_LENGTH** 系统复位信号长度选择。0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 us, 7: 3.2 us。(读/写)

**TIMG\_WDT\_CPU\_RESET\_LENGTH** CPU 复位信号长度选择。0: 100 ns, 1: 200 ns, 2: 300 ns, 3: 400 ns, 4: 500 ns, 5: 800 ns, 6: 1.6 us, 7: 3.2 us。(读/写)

**TIMG\_WDT\_LEVEL\_INT\_EN** 置 1 后, 如超过设置的阶段中断产生时间, 会产生电平中断。(读/写)

**TIMG\_WDT\_EDGE\_INT\_EN** 置 1 后, 如超过设置的阶段中断产生时间, 会产生边沿中断。(读/写)

**TIMG\_WDT\_STG3** 阶段 3 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

**TIMG\_WDT\_STG2** 阶段 2 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

**TIMG\_WDT\_STG1** 阶段 1 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

**TIMG\_WDT\_STG0** 阶段 0 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

**TIMG\_WDT\_EN** 置 1 后, MWDt 使能。(读/写)

Register 12.11: TIMG\_WDTCONFIG1\_REG (0x004C)

TIMG_WDT_CLK_PRESCALER																(reserved)																
31															16	15											0					
0x01																0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

**TIMG\_WDT\_CLK\_PRESCALER** MWDt 时钟预分频器值。MWDt 时钟长度 = 12.5 ns \* TIMGn\_WDT\_CLK\_PRESCALE。(读/写)

Register 12.12: TIMG\_WDTCONFIG2\_REG (0x0050)

TIMG_WDT_STG0_HOLD	
31	0
0x18cba80	
Reset	

**TIMG\_WDT\_STG0\_HOLD** MWDT 时钟周期中阶段 0 超时时间。(读/写)

Register 12.13: TIMG\_WDTCONFIG3\_REG (0x0054)

TIMG_WDT_STG1_HOLD	
31	0
0x7fffff	
Reset	

**TIMG\_WDT\_STG1\_HOLD** MWDT 时钟周期中阶段 1 超时时间。(读/写)

Register 12.14: TIMG\_WDTCONFIG4\_REG (0x0058)

TIMG_WDT_STG2_HOLD	
31	0
0x0ffff	
Reset	

**TIMG\_WDT\_STG2\_HOLD** MWDT 时钟周期中阶段 2 超时时间。(读/写)

Register 12.15: TIMG\_WDTCONFIG5\_REG (0x005C)

TIMG_WDT_STG3_HOLD	
31	0
0x0ffff	
Reset	

**TIMG\_WDT\_STG3\_HOLD** MWDAT 时钟周期中阶段 3 超时时间。(读/写)

Register 12.16: TIMG\_WDTFEED\_REG (0x0060)

TIMG_WDT_FEED	
31	0
0x000000	
Reset	

**TIMG\_WDT\_FEED** 写任意值驱动 MWDAT。(只写)

Register 12.17: TIMG\_WDTWPROTECT\_REG (0x0064)

TIMG_WDT_WKEY	
31	0
0x50d83aa1	
Reset	

**TIMG\_WDT\_WKEY** 如果寄存器中有和复位值不同的值，写保护使能。(读/写)

Register 12.18: TIMG\_RTCCALICFG2\_REG (0x00A8)

																TIMG_RTC_CALL_TIMEOUT_THRES				TIMG_RTC_CALL_TIMEOUT_RST_CNT				(reserved)		TIMG_RTC_CALL_TIMEOUT	
31																7	6	3		2	1	0					
0x1 fffff																0x3		0	0	0	Reset						

**TIMG\_RTC\_CALI\_TIMEOUT** RTC 校准超时指示器 (只读)

**TIMG\_RTC\_CALI\_TIMEOUT\_RST\_CNT** 校准超时复位周期 (读/写)

**TIMG\_RTC\_CALI\_TIMEOUT\_THRES** RTC 校准定时器的界限值。校准定时器的值超过此界限值时触发超时。(读/写)

Register 12.19: TIMG\_INT\_ENA\_TIMERS\_REG (0x0098)

																								<div>TIMG_WDT_INT_ENA TIMG_T1_INT_ENA TIMG_TO_INT_ENA</div>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																								
																								(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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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\_T<sub>x</sub>\_INT\_ENA** TIMG\_T<sub>x</sub>\_INT 中断的中断使能位 (读/写)

**TIMG\_WDT\_INT\_ENA** TIMG\_WDT\_INT 中断的中断使能位 (读/写)

Register 12.20: TIMG\_INT\_RAW\_TIMERS\_REG (0x009C)

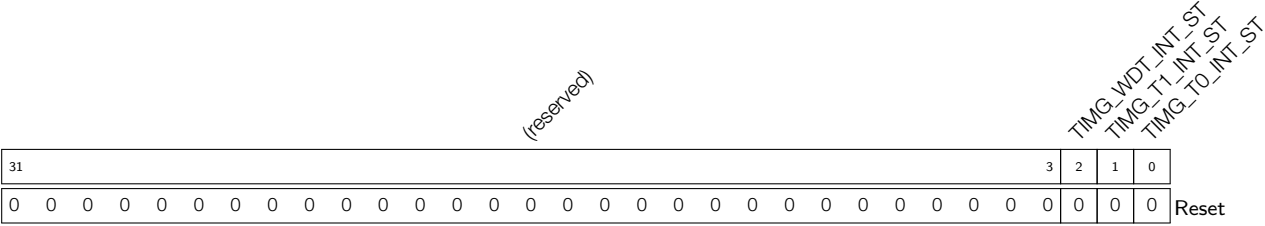
																												<div>TIMG_WDT_INT_RAW TIMG_T1_INT_RAW TIMG_TO_INT_RAW</div>																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																			
																												(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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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\_T<sub>x</sub>\_INT\_RAW** TIMG\_T<sub>x</sub>\_INT 中断的原始中断状态位。(只读)

**TIMG\_WDT\_INT\_RAW** TIMG\_WDT\_INT 中断的原始中断状态位。(只读)



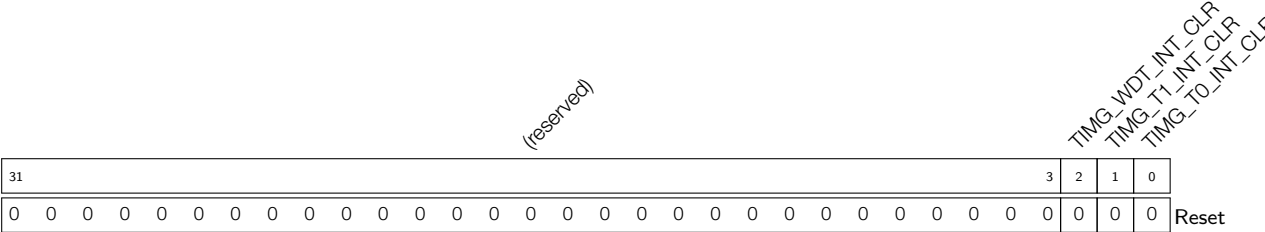
Register 12.21: TIMG\_INT\_ST\_TIMERS\_REG (0x00A0)



**TIMG\_T<sub>x</sub>\_INT\_ST** TIMG\_T<sub>x</sub>\_INT 中断的屏蔽中断状态位。(只读)

**TIMG\_WDT\_INT\_ST** TIMG\_WDT\_INT 中断的屏蔽中断状态位。(只读)

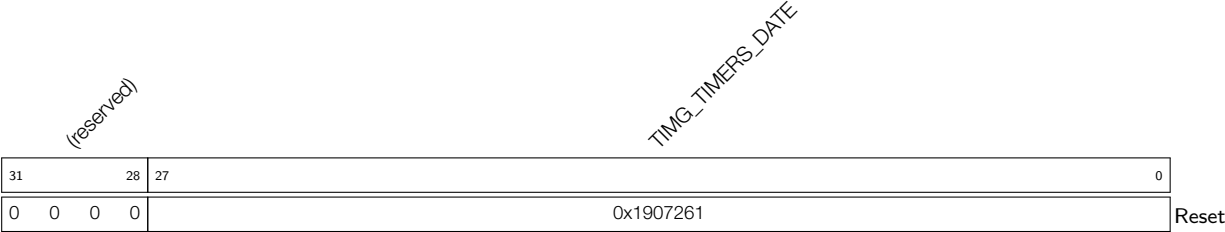
Register 12.22: TIMG\_INT\_CLR\_TIMERS\_REG (0x00A4)



**TIMG\_T<sub>x</sub>\_INT\_CLR** 置位此位，清除 TIMG\_T<sub>x</sub>\_INT 中断。(只写)

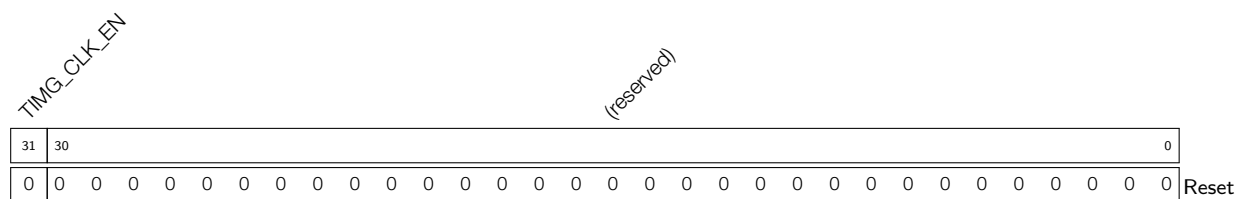
**TIMG\_WDT\_INT\_CLR** 置位此位，清除 TIMG\_WDT\_INT 中断。(只写)

Register 12.23: TIMG\_TIMERS\_DATE\_REG (0x00F8)



**TIMG\_TIMERS\_DATE** 版本控制寄存器。(读/写)

### Register 12.24: TIMG\_REGCLK\_REG (0x00FC)



**TIMG\_CLK\_EN** 寄存器时钟门控信号。1: 可以通过软件读取、写值的寄存器 0: 无法通过软件读取、写值的寄存器 (读/写)

## 13. 看门狗定时器

### 13.1 概述

系统/软件若出现不可预知的问题（比如软件卡在某个循环中或事件逾期）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/软件的错误行为。

ESP32-S2 中有三个看门狗定时器：两个定时器组中各一个（称作主系统看门狗定时器，缩写为 MWDT），RTC 模块中一个（称作 RTC 看门狗定时器，缩写为 RWDT）。看门狗在运行期间会经历四个阶段（除非看门狗被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，其中除了 RWDT 支持四种超时动作外，其它两个看门狗仅支持三种。超时动作包括：中断、CPU 复位、内核复位和系统复位。其中，只有 RWDT 能够触发系统复位，即复位芯片内部所有的数字电路，包括 RTC 和主系统。每个阶段的超时时间都可单独设置。

在引导加载 flash 固件期间，RWDT 和第一个 MWDT 会自动使能，以检测引导过程中发生的错误，并恢复运行。

请注意，本章节仅包含看门狗定时器的功能描述，其寄存器部分详见第 12 章：64 位定时器。

### 13.2 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。每阶段都可单独配置、使能和关闭。
- 如在某个阶段发生超时，则会采取三或四种（分别针对 MWDT 和 RWDT）动作中的一种（中断、CPU 复位、内核复位和系统复位）。
- 保护 32 位超时计数器，防止 RWDT 和 MWDT 的配置被无意间更改。
- 写保护，防止 RWDT 和 MWDT 配置无意间改动
- Flash 启动保护

如果在预定时间内 SPI flash 的引导过程没有完成，看门狗会重启整个主系统。

### 13.3 功能描述

#### 13.3.1 时钟源与 32 位计数器

每个看门狗定时器的核心是一个 32 位计数器。APB 时钟经过可配置的 16 位预分频器后会得到 MWDT 的时钟源。而 RWDT 的时钟源则直接取自于 RTC 慢速时钟（没有预分频器），频率通常为 32 kHz。MWDT 的 16 位预分频器可通过 `TIMG_WDTCONFIG1_REG` 寄存器的 `TIMG_WDT_CLK_PRESCALER` 字段配置。

MWDT 和 RWDT 看门狗可分别通过设置 `TIMG_WDT_EN` 和 `RTC_CNTL_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。如果软件在规定的时间内成功喂狗，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` 和 `RTC_CNTL_RTC_WDT_FEED` 寄存器内写入任意值，便可分别为 MDWT 和 RWDT 喂狗。

### 13.3.2 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在 `TIMG_WDTCONFIGi_REG`（*i* 的范围是 2 到 5）寄存器中配置，RWDt 的超时时间可在 `RTC_CNTL_WDT_STGGj_HOLD_REG`（*j* 的范围是 0 到 3）寄存器中配置。值得注意的是，RWDt 在阶段 0 的超时时间 ( $T_{hold0}$ ) 受 eFuse 寄存器 `EFUSE_WDT_DELAY_SEL` 字段和 `RTC_CTRL_WDT_STG0_HOLD_REG` 寄存器共同影响，关系如下：

$$T_{hold0} = \text{RTC\_CTRL\_WDT\_STG0\_HOLD\_REG} \ll (\text{EFUSE\_WDT\_DELAY\_SEL} + 1)$$

如某个阶段超时，下列超时动作之一将会执行：

- 触发中断  
如阶段超时，中断被触发。
- 复位 CPU 内核  
如阶段超时，复位 CPU 内核。
- 复位主系统  
如阶段超时，包括 MWDt 在内的主系统都会复位。RTC 不会复位。
- 复位主系统和 RTC  
如阶段超时，主系统和 RTC 同时复位。此动作仅可在 RWDt 中实现。
- 关闭  
该阶段对系统不产生影响。

MWDt 所有阶段的超时动作均在 `TIMG_WDTCONFIG0_REG` 寄存器中配置。与之类似，RWDt 的超时动作可在 `RTC_WDTCONFIG0` 寄存器配置。

### 13.3.3 写保护

看门狗定时器对于检测和处理系统/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，防止看门狗因偶然的错误访问而被关闭或篡改。

每个看门狗定时器都有一个写密钥寄存器，运行写保护机制（MWDt 看门狗使用 `TIMG_WDT_WKEY`，RWDt 看门狗使用 `RTC_CNTL_WDT_WKEY`）。必须向看门狗定时器的写密钥保护寄存器写入 `0x50D83AA1`，才能修改其它看门狗寄存器。如果写密钥保护寄存器的值不是 `0x50D83AA1`，任何试图在看门狗定时器寄存器（除了密钥保护寄存器本身）上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 `0x50D83AA1` 写入看门狗定时器的写密钥保护寄存器，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保护寄存器上写入除 `0x50D83AA1` 以外的任意值，重新使能写保护。

### 13.3.4 Flash 引导保护

在 flash 引导过程中，定时器组 0（`TIMG0`）中的 MWDt 和 RWDt 会自动使能。MWDt 使能后，阶段 0 的默认超时动作为系统复位。RWDt 的阶段 0 超时动作为主系统和 RTC 复位。引导后，应将

[TIMG\\_WDT\\_FLASHBOOT\\_MOD\\_EN](#) 和 [RTC\\_CNTL\\_WDT\\_FLASHBOOT\\_MOD\\_EN](#) 位清零，分别关闭 MWDT 和 RWDT 的 flash 引导保护。然后，软件可以配置 MWDT 和 RWDT。

### 13.4 寄存器

MWDT “寄存器” 是定时器模块的一部分，在第 12 章：64 位定时器 [定时器寄存器](#) 中有详细描述。

## 14. eFuse 控制器

### 14.1 概述

ESP32-S2 系统中有一块 4096 位的 eFuse，其中存储着参数内容。eFuse 的各个位一旦被烧写为 1，则不能再恢复为 0。eFuse 控制器按照软件配置完成对 eFuse 中各参数中的各个位的烧写。这些参数有些可以通过 eFuse 控制器被软件读取，有些直接由硬件模块使用。

### 14.2 主要特性

- 一次性可编程存储
- 烧写保护可配置
- 软件读取保护可配置
- 使用多种硬件编码方式保护参数内容

### 14.3 功能描述

#### 14.3.1 结构

eFuse 从结构上分成 11 个块 (BLOCK0 ~ BLOCK10)。

BLOCK0 存储大部分核心系统参数，其中 25 位供硬件使用，软件不可见；还有 37 位处于保留状态，留作未来使用。

表 51 列出了 BLOCK0 中的参数名称、偏移地址、位宽、是否可供硬件使用、烧写保护，以及描述信息。

在这些参数中，**EFUSE\_WR\_DIS** 用于控制其他参数的烧写，**EFUSE\_RD\_DIS** 用于控制软件读取 BLOCK4 ~ BLOCK10。更多关于这两个参数的信息请见章节 14.3.1.1、14.3.1.2。

表 51: BLOCK0 参数

参数	偏移	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
<b>EFUSE_WR_DIS</b>	0	32	Y	N/A	禁止 eFuse 烧写
<b>EFUSE_RD_DIS</b>	32	7	Y	0	禁止软件读取 eFuse BLOCK4-10 的内容
EFUSE_DIS_RTC_RAM_BOOT	39	1	N	1	禁止从 RTC RAM 启动
EFUSE_DIS_ICACHE	40	1	Y	2	关闭 ICache
EFUSE_DIS_DCACHE	41	1	Y	2	关闭 DCache
EFUSE_DIS_DOWNLOAD_ICACHE	42	1	Y	2	在 Download 模式下关闭 ICache
EFUSE_DIS_DOWNLOAD_DCACHE	43	1	Y	2	在 Download 模式下关闭 DCache
EFUSE_DIS_FORCE_DOWNLOAD	44	1	Y	2	禁止强制芯片进入 Download 模式

参数	偏移	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_DIS_USB	45	1	Y	2	关闭 USB OTG 功能
EFUSE_DIS_CAN	46	1	Y	2	关闭 TWAI 控制器功能
EFUSE_DIS_BOOT_REMAP	47	1	Y	2	REMAP 指代 RAM 空间可以映射到 ROM 空间，此功能可被关闭
EFUSE_SOFT_DIS_JTAG	49	1	Y	2	软件禁用 JTAG，可通过 HMAC 重新启动
EFUSE_HARD_DIS_JTAG	50	1	Y	2	硬件永远禁用 JTAG
EFUSE_DIS_DOWNLOAD_MAN- UAL_ENCRYPT	51	1	Y	2	在 download boot 模式下禁用 flash 加密功能
EFUSE_USB_EXCHG_PINS	56	1	Y	30	交换 USB D+ / D- 管脚
EFUSE_EXT_PHY_ENABLE	57	1	N	30	使能外部 USB PHY
EFUSE_USB_FORCE_NOPERSIST	58	1	N	30	强制设置 USB BVALID 为 1
EFUSE_VDD_SPI_XPD	68	1	Y	3	若 VDD_SPI_FORCE 为 1，控制 VDD_SPI 调节器上电
EFUSE_VDD_SPI_TIEH	69	1	Y	3	若 VDD_SPI_FORCE 为 1，选择 VDD_SPI 电压。0: VDD_SPI 连接 1.8 V LDO；1: VDD_SPI 连接 VDD_RTC_IO
EFUSE_VDD_SPI_FORCE	70	1	Y	3	置位使用 XPD_VDD_PSI_REG 和 VDD_SPI_TIEH 配置 VDD_SPI LDO
EFUSE_WDT_DELAY_SEL	80	2	Y	3	选择 RTC WDT 超时阈值
EFUSE_SPI_BOOT_CRYPT_CNT	82	3	Y	4	使能 SPI boot 加解密，奇数个 1: 使能；偶数个 1: 关闭
EFUSE_SECURE_BOOT_KEY_ REVOKE0	85	1	N	5	使能撤销第一个 secure boot（安全启动）密钥
EFUSE_SECURE_BOOT_KEY_ REVOKE1	86	1	N	6	使能撤销第二个 secure boot 密钥
EFUSE_SECURE_BOOT_KEY_ REVOKE2	87	1	N	7	使能撤销第三个 secure boot 密钥
EFUSE_KEY_PURPOSE_0	88	4	Y	8	Key0 用途 (purpose)，见表 52
EFUSE_KEY_PURPOSE_1	92	4	Y	9	Key1 用途，见表 52
EFUSE_KEY_PURPOSE_2	96	4	Y	10	Key2 用途，见表 52
EFUSE_KEY_PURPOSE_3	100	4	Y	11	Key3 用途，见表 52
EFUSE_KEY_PURPOSE_4	104	4	Y	12	Key4 用途，见表 52
EFUSE_KEY_PURPOSE_5	108	4	Y	13	Key5 用途，见表 52
EFUSE_SECURE_BOOT_EN	116	1	N	15	使能 secure boot
EFUSE_SECURE_BOOT_ AG- GRESSIVE_REVOKE	117	1	N	16	Secure boot 的撤销采用激进策略
EFUSE_FLASH_TPUW	124	4	N	18	上电后 flash 等待时间，单位为 (ms/2)，值为 15 时，等待时间为 7.5 ms
EFUSE_DIS_DOWNLOAD_MODE	128	1	N	18	关闭所有 download boot 模式

参数	偏移	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_DIS_LEGACY_SPI_BOOT	129	1	N	18	关闭 Legacy SPI boot 模式
EFUSE_UART_PRINT_CONTROL	130	1	N	18	选择打印 boot 信息的 UART 通道。 0: UART0; 1: UART1
EFUSE_DIS_USB_DOWNLOAD_MODE	132	1	N	18	在 UART download boot 模式下关闭 USB OTG 下载功能
EFUSE_ENABLE_SECURITY_DOWNLOAD	133	1	N	18	使能 UART 安全下载模式 (仅支持读写 flash)
EFUSE_UART_PRINT_CONTROL	134	2	N	18	控制 UART boot 信息输出模式。 2'b00: 强制打印; 2'b01: 由 GPIO 46 控制, 低电平打印; 2'b10: 由 GPIO 46 控制, 高电平打印; 2'b11: 强制关闭打印
EFUSE_PIN_POWER_SELECTION	136	1	N	18	选择 GPIO33 ~ GPIO37 的电源; 0: VDD3P3_CPU; 1: VDD_SPI
EFUSE_FLASH_TYPE	137	1	N	18	Flash 类型; 0: 4 根数据线; 1: 8 根数据线
EFUSE_FORCE_SEND_RESUME	138	1	N	18	强制 ROM 代码在 SPI 启动过程中发送 SPI flash 继续指令
EFUSE_SECURE_VERSION	139	16	N	18	安全版本 (用于 ESP-IDF 的防回滚功能)

表 52 为密钥用途各个数值对应的含义。通过配置参数 EFUSE\_KEY\_PURPOSE\_*n* 来声明 KEY*n* 用途 (*n*: 0 ~ 5)。

表 52: 密钥用途数值对应的含义

密钥用途数值	含义
0	指定为用户使用 (仅为软件使用)
1	保留
2	指定为 XTS_AES_256_KEY_1 使用 (用于 flash/SRAM 加解密)
3	指定为 XTS_AES_256_KEY_2 使用 (用于 flash/SRAM 加解密)
4	指定为 XTS_AES_128_KEY 使用 (用于 flash/SRAM 加解密)
5	指定为 HMAC Downstream (下行) 模式使用
6	指定为 HMAC Downstream 模式下的 JTAG 使用
7	指定为 HMAC Downstream 模式下的数字签名使用
8	指定为 HMAC Upstream (上行) 模式使用
9	指定为 SECURE_BOOT_DIGEST0 使用 (secure boot 密钥摘要)
10	指定为 SECURE_BOOT_DIGEST1 使用 (secure boot 密钥摘要)
11	指定为 SECURE_BOOT_DIGEST2 使用 (secure boot 密钥摘要)

表 53 列出了 BLOCK1 ~ BLOCK10 中存储的参数的信息。



表 53: BLOCK1-10 参数

块	参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	EFUSE_RD_DIS 软件读取保护位	描述
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC 地址
	EFUSE_SPI_PAD_CONFIGURE	[0:5]	N	20	N/A	CLK
		[6:11]	N	20	N/A	Q (D1)
		[12:17]	N	20	N/A	D (D0)
		[18:23]	N	20	N/A	CS
		[24:29]	N	20	N/A	HD (D3)
		[30:35]	N	20	N/A	WP (D2)
		[36:41]	N	20	N/A	DQS
		[42:47]	N	20	N/A	D4
		[48:53]	N	20	N/A	D5
		[54:59]	N	20	N/A	D6
		[60:65]	N	20	N/A	D7
	EFUSE_SYS_DATA_PART0	78	N	20	N/A	系统数据
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	系统数据
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	用户数据
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 或用户数据
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 或用户数据
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 或用户数据
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 或用户数据
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 或用户数据
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 或用户数据
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	系统数据

其中，BLOCK4 ~ 9 分别存储 KEY0~5，表示 eFuse 中至多可以烧写 6 个 256 位的密钥。每烧写一个密钥，还需要烧写该密钥用途的数值（见表 52）。例如，软件将用于 HMAC Downstream 模式下的 JTAG 功能的密钥烧写到 KEY3（即 BLOCK7），还需要将密钥用途的数值 6 烧写到 EFUSE\_KEY\_PURPOSE\_3。

BLOCK1 ~ BLOCK10 均采用 RS 编码方式，因此参数烧写受到一定的限制，具体请参考章节 14.3.1.3：数据存储方式，和章节 14.3.2：软件烧写参数。

### 14.3.1.1 EFUSE\_WR\_DIS

参数 EFUSE\_WR\_DIS 决定了 eFuse 中所有的参数是否处于烧写保护状态。烧写完 EFUSE\_WR\_DIS 参数后，需要更新 eFuse 读寄存器才能生效（参考章节 14.3.3 中：更新 eFuse 读寄存器）。

表 51 以及表 53 中的“EFUSE\_WR\_DIS 烧写保护位”列描述了各参数的烧写保护状态具体由 EFUSE\_WR\_DIS 的哪个位决定。

当某个参数对应的烧写保护位为 0 时，表示此参数未处于烧写保护状态，可以烧写该参数，但已经被烧写的参数不能被重复烧写。

当某个参数对应的烧写保护位为 1 时，表示此参数处于烧写保护状态，此参数的每一个位都无法被更改，未被烧写的位永远为 0，已经被烧写的位永远为 1。所以如果某个参数已经处于烧写保护状态了，则会一直处在

状态，无法再更改。

### 14.3.1.2 EFUSE\_RD\_DIS

所有参数中，只有 BLOCK4 ~ BLOCK10 的参数受软件读取保护状态的约束，即表 53 中“EFUSE\_RD\_DIS 软件读取保护”列非“N/A”的参数。烧写完 EFUSE\_RD\_DIS 参数后，需要更新 eFuse 读寄存器才能生效（参考章节 14.3.3 中：更新 eFuse 读寄存器）。

参数 EFUSE\_RD\_DIS 中的某个位为 0，表示此位管理的参数未处于软件读取保护状态；某个位为 1，表示此位管理的参数处于软件读取保护状态。

除 BLOCK4 ~ BLOCK10 之外，其他参数不受软件读取保护状态的约束，均可被软件读取。

BLOCK4 ~ BLOCK10 即使被配置处于读取保护状态，仍然可以通过设置 EFUSE\_KEY\_PURPOSE<sub>n</sub> 被硬件使用。

### 14.3.1.3 数据存储方式

eFuse 使用硬件编码机制保护数据，对用户不可见。

BLOCK0 使用 4 备份方式存储参数，即 BLOCK0 中的所有参数（除了 EFUSE\_WR\_DIS）均在 eFuse 中存储了 4 份。4 备份机制对软件不可见。

BLOCK1 ~ BLOCK10 使用 RS (44, 32) 编码方式，最多支持自动校正 5 个字节。本文 RS (44, 32) 使用的本源多项式为  $p(x) = x^8 + x^4 + x^3 + x^2 + 1$ ，产生校验码的移位寄存器电路如图 14-1 所示，其中 gf\_mul\_n (n 为一个整数) 为  $GF(2^8)$  域中某一字节数据与元素  $\alpha^n$  相乘的结果。

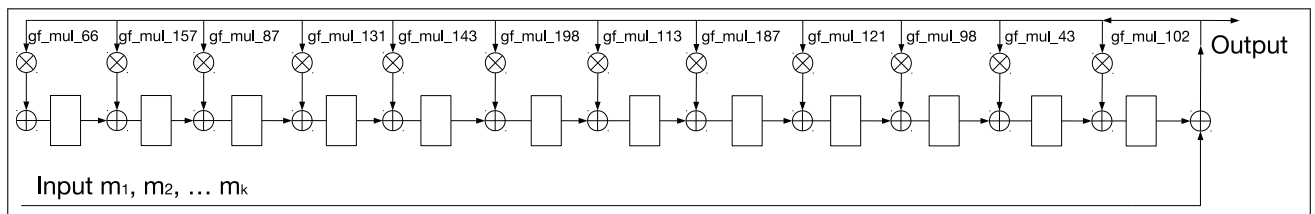


图 14-1. 移位寄存器电路图

软件需要先对 32 字节参数进行 RS (44, 32) 编码得到 12 字节验证码，然后将参数及验证码一起烧入 eFuse。

eFuse 控制器会在读 eFuse 的过程中自动完成解码和自动校正。

由于 RS 校验码是在整个 256 位的 eFuse block 上生成的，因此每个 block 只能写入一次。

## 14.3.2 软件烧写参数

烧写 eFuse 参数时，需要按块烧写。BLOCK0 ~ BLOCK10 共用同一段地址来存储即将烧写的参数。通过配置 EFUSE\_BLK\_NUM 参数表明当前需要烧写的是哪一个块。

### 烧写 BLOCK0

当 EFUSE\_BLK\_NUM = 0 时，烧写 BLOCK0。EFUSE\_PGM\_DATA0\_REG 寄存器存储着 EFUSE\_WR\_DIS。

EFUSE\_PGM\_DATA1\_REG ~ EFUSE\_PGM\_DATA5\_REG 用来存储即将烧写的参数的有效信息，其中 25 位为硬件可用、软件不可见的有效信息，必须写入 0，对应位置为：

- EFUSE\_PGM\_DATA1\_REG[29:31]
- EFUSE\_PGM\_DATA1\_REG[20:23]
- EFUSE\_PGM\_DATA1\_REG[16]
- EFUSE\_PGM\_DATA2\_REG[7:15]
- EFUSE\_PGM\_DATA2\_REG[0:3]
- EFUSE\_PGM\_DATA3\_REG[16:19]

EFUSE\_PGM\_DATA6\_REG ~ EFUSE\_PGM\_DATA7\_REG 以及 EFUSE\_PGM\_CHECK\_VALUE0\_REG ~ EFUSE\_PGM\_CHECK\_VALUE2\_REG 中的数据不影响 BLOCK0 的烧写。

### 烧写 BLOCK1

当 EFUSE\_BLK\_NUM = 1 时，EFUSE\_PGM\_DATA0\_REG ~ EFUSE\_PGM\_DATA5\_REG 存储着 BLOCK1 即将烧写的参数，EFUSE\_PGM\_CHECK\_VALUE0\_REG ~ EFUSE\_PGM\_CHECK\_VALUE2\_REG 中存储着对应的 RS 校验码。EFUSE\_PGM\_DATA6\_REG ~ EFUSE\_PGM\_DATA7\_REG 中的数据不影响 BLOCK1 的烧写。RS 校验码的计算视这些位为 0。

### 烧写 BLOCK2 ~ 10

当 EFUSE\_BLK\_NUM = 2 ~ 10 时，EFUSE\_PGM\_DATA0\_REG ~ EFUSE\_PGM\_DATA7\_REG 存储着即将烧写的参数，EFUSE\_PGM\_CHECK\_VALUE0\_REG ~ EFUSE\_PGM\_CHECK\_VALUE2\_REG 中存储着对应的 RS 校验码。

### 烧写流程

烧写参数的流程如下：

1. 配置 EFUSE\_BLK\_NUM 参数，决定烧写哪一个块。
2. 将需要烧写的参数填写到寄存器 EFUSE\_PGM\_DATA0\_REG ~ EFUSE\_PGM\_DATA7\_REG 和 EFUSE\_PGM\_CHECK\_VALUE0\_REG ~ EFUSE\_PGM\_CHECK\_VALUE2\_REG 中。
3. 确保 eFuse 时钟寄存器的配置正确，具体请参考章节 14.3.4.1：eFuse 烧写时序。
4. 确保 eFuse 烧写电压 VDDQ 的配置正确，具体请参考章节 14.3.4.2：eFuse VDDQ 时序。
5. 配置寄存器 EFUSE\_CONF\_REG 的 EFUSE\_OP\_CODE 位段为 0x5A5A。
6. 配置寄存器 EFUSE\_CMD\_REG 的 EFUSE\_PGM\_CMD 位段为 1。
7. 轮询寄存器 EFUSE\_CMD\_REG 直到其为 0x0，或者等待烧写完成中断产生。识别烧写/读取完成中断产生的方法详见章节 14.3.3 最后的说明。
8. 将寄存器中写入的参数清零。
9. 执行更新 eFuse 读寄存器操作使写入的新值生效，具体请参考章节 14.3.3：软件读取参数。

### 限制

BLOCK0 中不同的参数，甚至对于同一个参数中的不同位可以在多次烧写中分别完成。但是并不推荐这样做，而是建议尽量减少烧写次数。我们建议对于某个参数中的所有需要烧写的位都在一次烧写中完成。并且当 EFUSE\_WR\_DIS 的某个位管理的所有参数都烧写之后，就立即烧写 EFUSE\_WR\_DIS 的这个位。甚至可以在同

一次烧写中既烧写 EFUSE\_WR\_DIS 的某个位管理的所有参数，同时也烧写 EFUSE\_WR\_DIS 的这个位。另外严禁对已经烧写了的位重复烧写，否则将发生烧写错误。

BLOCK1 中数据信息在出厂时已经烧写完毕，不允许再次烧写。

BLOCK2 ~ 10 中每一个 BLOCK 都只能烧写一次，不允许重复烧写。

### 14.3.3 软件读取参数

软件不能直接读取 eFuse 中烧写的信息内容。eFuse 控制器能够将烧写的信息读取到对应的地址段的寄存器内，软件再通过读取以 EFUSE\_RD\_ 开始的寄存器来获取 eFuse 信息。下表列出了读取数据的寄存器名称以及对应烧写时的烧写寄存器名称。

BLOCK	读寄存器	烧写寄存器
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SPI_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_PART1_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY <sub>n</sub> _DATA0 ~ 7_REG ( <i>n</i> : 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_PART2_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

#### 更新 eFuse 读寄存器

eFuse 控制器读取内部 eFuse 来更新相应寄存器的数据。读取操作在系统复位时进行，也可以根据需要由软件手动触发（例如在需要读取新烧写 eFuse 中的数据内容时）。软件触发 eFuse 读取操作的流程如下：

1. 配置 eFuse 读取时序，具体请参考章节 14.3.4.3: eFuse 读取时序。
2. 配置寄存器 EFUSE\_CONF\_REG 的 EFUSE\_OP\_CODE 位段为 0x5AA5。
3. 配置寄存器 EFUSE\_CMD\_REG 的 EFUSE\_READ\_CMD 位段为 1。
4. 轮询寄存器 EFUSE\_CMD\_REG 直到其为 0x0，或者等待 read\_done interrupt（读取完成中断）产生，识别烧写/读取完成中断产生的方法详见下方说明。
5. 软件从 eFuse 存储器中读取参数的值。

eFuse 读寄存器中的数值将一直保持到下一次执行更新 eFuse 读操作。

#### 烧写错误检测

烧写错误记录寄存器允许软件检测 eFuse 参数的备份是否有不一致的错误。

EFUSE\_RD\_REPEAT\_ERR0 ~ 3\_REG 寄存器用于指示 BLOCK0 中除了 EFUSE\_WR\_DIS 外的其他参数的烧写是否出错（对应位为 1 代表烧写出错，此位作废；为 0 代表烧写正确）。

EFUSE\_RD\_RS\_ERR0 ~ 1\_REG 寄存器记录 eFuse 读 BLOCK1 ~ BLOCK10 过程中，纠错的字节数目以及 RS 解码是否失败的信息。

软件只可以在更新 eFuse 读寄存器操作完成之后才可以读取上面几个寄存器的值。

#### 识别烧写/读取操作完成

识别烧写/读取操作完成的方法如下。位 1 对应烧写操作，位 0 对应读取操作。

- 方法 1:
  1. 轮询寄存器 `EFUSE_INT_RAW_REG` 的位 1/0，直到位 1/0 为 1，表示烧写/读取操作完成。
- 方法 2:
  1. 对寄存器 `EFUSE_INT_ENA_REG` 的位 1/0 置 1，使 eFuse 控制器能够产生烧写/读取完成中断。
  2. 配置中断矩阵使 CPU 能够响应 EFUSE 的中断信号。
  3. 等待烧写/读取完成中断产生。
  4. 对寄存器 `EFUSE_INT_CLR_REG` 的位 1/0 置 1 以清除烧写/读取完成中断。

14.3.4 时序

14.3.4.1 eFuse 烧写时序

图 14-2 是 eFuse 烧写的时序图。硬件提供 `EFUSE_TSUP_A`、`EFUSE_TPGM`、`EFUSE_THP_A` 和 `EFUSE_TPGM_INACTIVE` 四个寄存器用于控制 eFuse 的烧写时序。图中 CSB、VDDQ、PGENB 的含义如下：

- CSB：片选信号，低电平有效
- VDDQ：eFuse 烧写电压
- PGENB：烧写使能信号，低电平有效

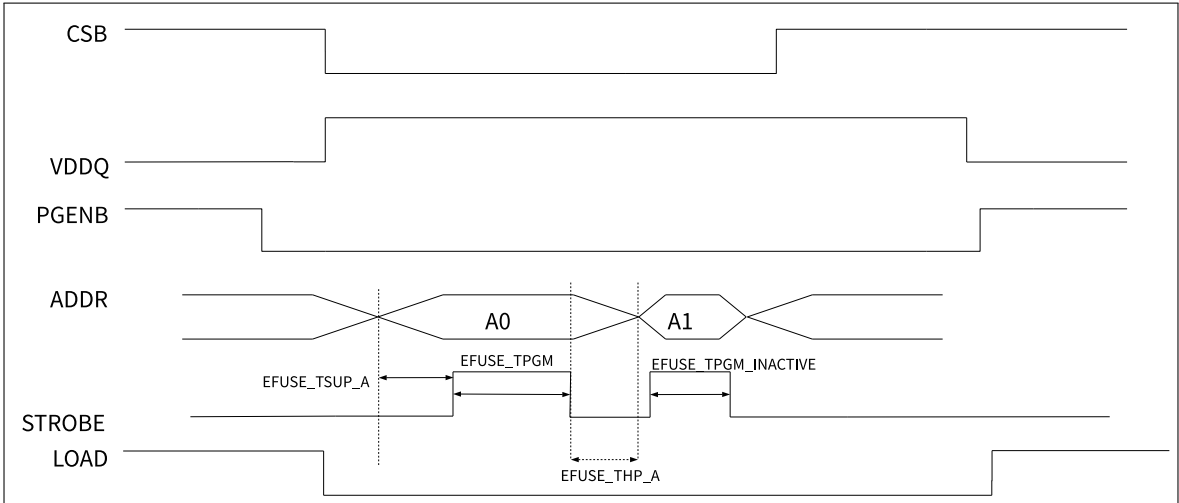


图 14-2. eFuse 烧写时序图

eFuse 模块使用的是 CLK\_APB 时钟。由于 CLK\_APB 时钟频率是可变的，需要根据不同的时钟频率来配置上述时序参数，具体请见表 55。复位后，eFuse 默认的参数配置对应的是 20 MHz 时钟频率。

表 55: eFuse 烧写时序参数配置

APB Frequency	<code>EFUSE_TSUP_A</code> (> 6.669 ns)	<code>EFUSE_TPGM</code> (9-11 $\mu$ s, usually 10 $\mu$ s)	<code>EFUSE_THP_A</code> (> 6.166 ns)	<code>EFUSE_TPGM_INACTIVE</code> (> 35.96 ns)
---------------	---	---	--	--

APB Frequency	EFUSE_TSUP_A (> 6.669 ns)	EFUSE_TPGM (9-11 $\mu$ s, usually 10 $\mu$ s)	EFUSE_THP_A (> 6.166 ns)	EFUSE_TPGM_INACTIVE (> 35.96 ns)
80 MHz	0x2	0x320	0x2	0x4
40 MHz	0x1	0x190	0x1	0x2
20 MHz	0x1	0xC8	0x1	0x1

图 14-2 中 A0 地址烧写 1，即 A0 对应的 eFuse 位为 1；A1 地址处于不烧写状态，A1 对应的 eFuse 位为 0。

14.3.4.2 eFuse VDDQ 时序

用户需要根据不同的 APB 时钟频率来配置 eFuse 的烧写电压 VDDQ 的时序参数，具体配置如下：

表 56: VDDQ 时序参数配置

APB Frequency	EFUSE_DAC_CLK_DIV (> 1 $\mu$ s)	EFUSE_PWR_ON_NUM (> EFUSE_DAC_CLK_DIV*255)	EFUSE_PWR_OFF_NUM (> 3 $\mu$ s)
80 MHz	0xA0	0xA200	0x100
40 MHz	0x50	0x5100	0x80
20 MHz	0x28	0x2880	0x40

14.3.4.3 eFuse 读取时序

图 14-3 是 eFuse 读取的时序图。硬件提供 EFUSE\_TSUR\_A、EFUSE\_TRD 和 EFUSE\_THR\_A 三个寄存器用于控制 eFuse 的读时序。

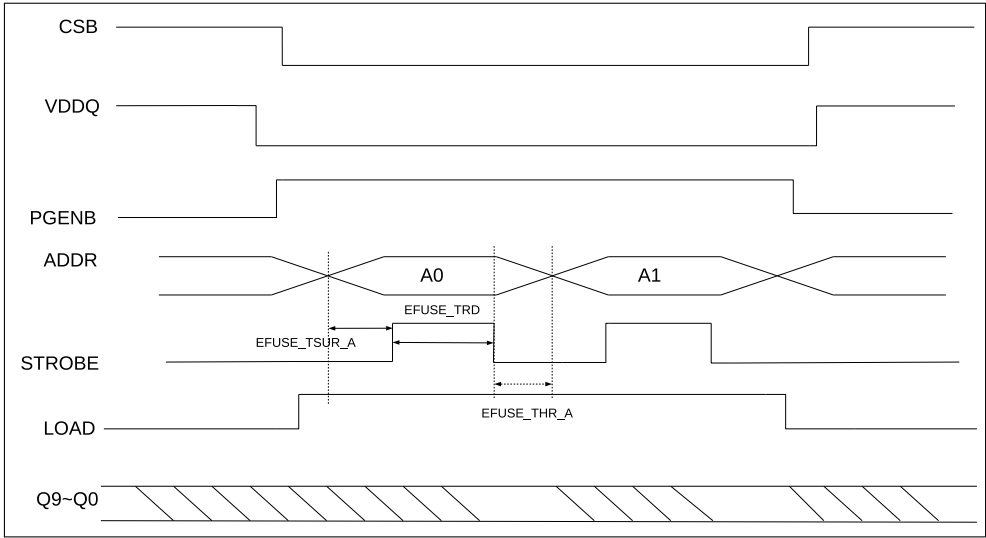


图 14-3. eFuse 读取时序图

用户需要根据不同的 APB 时钟频率来配置上述时序参数，具体请见表 57。

表 57: eFuse 读取时序参数配置

APB Frequency	EFUSE_TSR_A (> 6.669 ns)	EFUSE_TRD (> 35.96 ns)	EFUSE_THR_A (> 6.166 ns)
80 MHz	0x2	0x4	0x2
40 MHz	0x1	0x2	0x1
20 MHz	0x1	0x1	0x1

### 14.3.5 硬件模块使用参数

硬件模块使用参数是通过电路连接实现的，软件无法干预这个过程。硬件使用的参数为表 51 和 53 “硬件使用”一栏中标记为“Y”的参数。

### 14.3.6 中断

- 烧写完成中断：当 eFuse 烧写完成后，此中断被触发。如果要启动该中断信号，需将寄存器 EFUSE\_PGM\_DONE\_INT\_ENA 置 1。
- 读取完成中断：当 eFuse 读取完成后，此中断被触发。如果要启动该中断信号，需将寄存器 EFUSE\_READ\_DONE\_INT\_ENA 置 1。

## 14.4 基地址

用户可以通过两个不同的寄存器基地址访问 eFuse 控制器，如表 58 所示。更多有关通过不同总线访问外设的信息，请参考章节 1 系统和存储器。

表 58: eFuse 控制器基地址

访问外设总线	地址值
PeriBUS1	0x6001A000
PeriBUS2	0x3FC1A000

## 14.5 寄存器列表

请注意，这里的地址是相对于 eFuse 控制器基地址的地址偏移量（相对地址）。请参阅章节 14.4 获取有关 eFuse 控制器的基地址的信息。

名称	描述	地址	访问
<b>烧写数据寄存器</b>			
EFUSE_PGM_DATA0_REG	存放待烧写数据的第 0 个寄存器内容。	0x0000	读/写
EFUSE_PGM_DATA1_REG	存放待烧写数据的第 1 个寄存器内容。	0x0004	读/写
EFUSE_PGM_DATA2_REG	存放待烧写数据的第 2 个寄存器内容。	0x0008	读/写
EFUSE_PGM_DATA3_REG	存放待烧写数据的第 3 个寄存器内容。	0x000C	读/写
EFUSE_PGM_DATA4_REG	存放待烧写数据的第 4 个寄存器内容。	0x0010	读/写
EFUSE_PGM_DATA5_REG	存放待烧写数据的第 5 个寄存器内容。	0x0014	读/写



名称	描述	地址	访问
EFUSE_PGM_DATA6_REG	存放待烧写数据的第 6 个寄存器内容。	0x0018	读/写
EFUSE_PGM_DATA7_REG	存放待烧写数据的第 7 个寄存器内容。	0x001C	读/写
EFUSE_PGM_CHECK_VALUE0_REG	存放待烧写 RS 代码的第 0 个寄存器数据。	0x0020	读/写
EFUSE_PGM_CHECK_VALUE1_REG	存放待烧写 RS 代码的第 1 个寄存器数据。	0x0024	读/写
EFUSE_PGM_CHECK_VALUE2_REG	存放待烧写 RS 代码的第 2 个寄存器数据。	0x0028	读/写
<b>读取数据寄存器</b>			
EFUSE_RD_WR_DIS_REG	BLOCK0 的第 0 个寄存器内容。	0x002C	只读
EFUSE_RD_REPEAT_DATA0_REG	BLOCK0 的第 1 个寄存器内容。	0x0030	只读
EFUSE_RD_REPEAT_DATA1_REG	BLOCK0 的第 2 个寄存器内容。	0x0034	只读
EFUSE_RD_REPEAT_DATA2_REG	BLOCK0 的第 3 个寄存器内容。	0x0038	只读
EFUSE_RD_REPEAT_DATA3_REG	BLOCK0 的第 4 个寄存器内容。	0x003C	只读
EFUSE_RD_REPEAT_DATA4_REG	BLOCK0 的第 5 个寄存器内容。	0x0040	只读
EFUSE_RD_MAC_SPI_SYS_0_REG	BLOCK1 的第 0 个寄存器内容。	0x0044	只读
EFUSE_RD_MAC_SPI_SYS_1_REG	BLOCK1 的第 1 个寄存器内容。	0x0048	只读
EFUSE_RD_MAC_SPI_SYS_2_REG	BLOCK1 的第 2 个寄存器内容。	0x004C	只读
EFUSE_RD_MAC_SPI_SYS_3_REG	BLOCK1 的第 3 个寄存器内容。	0x0050	只读
EFUSE_RD_MAC_SPI_SYS_4_REG	BLOCK1 的第 4 个寄存器内容。	0x0054	只读
EFUSE_RD_MAC_SPI_SYS_5_REG	BLOCK1 的第 5 个寄存器内容。	0x0058	只读
EFUSE_RD_SYS_DATA_PART1_0_REG	BLOCK2 (system) 的第 0 个寄存器内容。	0x005C	只读
EFUSE_RD_SYS_DATA_PART1_1_REG	BLOCK2 (system) 的第 1 个寄存器内容。	0x0060	只读
EFUSE_RD_SYS_DATA_PART1_2_REG	BLOCK2 (system) 的第 2 个寄存器内容。	0x0064	只读
EFUSE_RD_SYS_DATA_PART1_3_REG	BLOCK2 (system) 的第 3 个寄存器内容。	0x0068	只读
EFUSE_RD_SYS_DATA_PART1_4_REG	BLOCK2 (system) 的第 4 个寄存器内容。	0x006C	只读
EFUSE_RD_SYS_DATA_PART1_5_REG	BLOCK2 (system) 的第 5 个寄存器内容。	0x0070	只读
EFUSE_RD_SYS_DATA_PART1_6_REG	BLOCK2 (system) 的第 6 个寄存器内容。	0x0074	只读
EFUSE_RD_SYS_DATA_PART1_7_REG	BLOCK2 (system) 的第 7 个寄存器内容。	0x0078	只读
EFUSE_RD_USR_DATA0_REG	BLOCK3 (user) 的第 0 个寄存器内容。	0x007C	只读
EFUSE_RD_USR_DATA1_REG	BLOCK3 (user) 的第 1 个寄存器内容。	0x0080	只读
EFUSE_RD_USR_DATA2_REG	BLOCK3 (user) 的第 2 个寄存器内容。	0x0084	只读
EFUSE_RD_USR_DATA3_REG	BLOCK3 (user) 的第 3 个寄存器内容。	0x0088	只读
EFUSE_RD_USR_DATA4_REG	BLOCK3 (user) 的第 4 个寄存器内容。	0x008C	只读
EFUSE_RD_USR_DATA5_REG	BLOCK3 (user) 的第 5 个寄存器内容。	0x0090	只读
EFUSE_RD_USR_DATA6_REG	BLOCK3 (user) 的第 6 个寄存器内容。	0x0094	只读
EFUSE_RD_USR_DATA7_REG	BLOCK3 (user) 的第 7 个寄存器内容。	0x0098	只读
EFUSE_RD_KEY0_DATA0_REG	BLOCK4 (KEY0) 的第 0 个寄存器内容。	0x009C	只读
EFUSE_RD_KEY0_DATA1_REG	BLOCK4 (KEY0) 的第 1 个寄存器内容。	0x00A0	只读
EFUSE_RD_KEY0_DATA2_REG	BLOCK4 (KEY0) 的第 2 个寄存器内容。	0x00A4	只读
EFUSE_RD_KEY0_DATA3_REG	BLOCK4 (KEY0) 的第 3 个寄存器内容。	0x00A8	只读
EFUSE_RD_KEY0_DATA4_REG	BLOCK4 (KEY0) 的第 4 个寄存器内容。	0x00AC	只读
EFUSE_RD_KEY0_DATA5_REG	BLOCK4 (KEY0) 的第 5 个寄存器内容。	0x00B0	只读
EFUSE_RD_KEY0_DATA6_REG	BLOCK4 (KEY0) 的第 6 个寄存器内容。	0x00B4	只读
EFUSE_RD_KEY0_DATA7_REG	BLOCK4 (KEY0) 的第 7 个寄存器内容。	0x00B8	只读



名称	描述	地址	访问
EFUSE_RD_KEY1_DATA0_REG	BLOCK5 (KEY1) 的第 0 个寄存器内容。	0x00BC	只读
EFUSE_RD_KEY1_DATA1_REG	BLOCK5 (KEY1) 的第 1 个寄存器内容。	0x00C0	只读
EFUSE_RD_KEY1_DATA2_REG	BLOCK5 (KEY1) 的第 2 个寄存器内容。	0x00C4	只读
EFUSE_RD_KEY1_DATA3_REG	BLOCK5 (KEY1) 的第 3 个寄存器内容。	0x00C8	只读
EFUSE_RD_KEY1_DATA4_REG	BLOCK5 (KEY1) 的第 4 个寄存器内容。	0x00CC	只读
EFUSE_RD_KEY1_DATA5_REG	BLOCK5 (KEY1) 的第 5 个寄存器内容。	0x00D0	只读
EFUSE_RD_KEY1_DATA6_REG	BLOCK5 (KEY1) 的第 6 个寄存器内容。	0x00D4	只读
EFUSE_RD_KEY1_DATA7_REG	BLOCK5 (KEY1) 的第 7 个寄存器内容。	0x00D8	只读
EFUSE_RD_KEY2_DATA0_REG	BLOCK6 (KEY2) 的第 0 个寄存器内容。	0x00DC	只读
EFUSE_RD_KEY2_DATA1_REG	BLOCK6 (KEY2) 的第 1 个寄存器内容。	0x00E0	只读
EFUSE_RD_KEY2_DATA2_REG	BLOCK6 (KEY2) 的第 2 个寄存器内容。	0x00E4	只读
EFUSE_RD_KEY2_DATA3_REG	BLOCK6 (KEY2) 的第 3 个寄存器内容。	0x00E8	只读
EFUSE_RD_KEY2_DATA4_REG	BLOCK6 (KEY2) 的第 4 个寄存器内容。	0x00EC	只读
EFUSE_RD_KEY2_DATA5_REG	BLOCK6 (KEY2) 的第 5 个寄存器内容。	0x00F0	只读
EFUSE_RD_KEY2_DATA6_REG	BLOCK6 (KEY2) 的第 6 个寄存器内容。	0x00F4	只读
EFUSE_RD_KEY2_DATA7_REG	BLOCK6 (KEY2) 的第 7 个寄存器内容。	0x00F8	只读
EFUSE_RD_KEY3_DATA0_REG	BLOCK7 (KEY3) 的第 0 个寄存器内容。	0x00FC	只读
EFUSE_RD_KEY3_DATA1_REG	BLOCK7 (KEY3) 的第 1 个寄存器内容。	0x0100	只读
EFUSE_RD_KEY3_DATA2_REG	BLOCK7 (KEY3) 的第 2 个寄存器内容。	0x0104	只读
EFUSE_RD_KEY3_DATA3_REG	BLOCK7 (KEY3) 的第 3 个寄存器内容。	0x0108	只读
EFUSE_RD_KEY3_DATA4_REG	BLOCK7 (KEY3) 的第 4 个寄存器内容。	0x010C	只读
EFUSE_RD_KEY3_DATA5_REG	BLOCK7 (KEY3) 的第 5 个寄存器内容。	0x0110	只读
EFUSE_RD_KEY3_DATA6_REG	BLOCK7 (KEY3) 的第 6 个寄存器内容。	0x0114	只读
EFUSE_RD_KEY3_DATA7_REG	BLOCK7 (KEY3) 的第 7 个寄存器内容。	0x0118	只读
EFUSE_RD_KEY4_DATA0_REG	BLOCK8 (KEY4) 的第 0 个寄存器内容。	0x011C	只读
EFUSE_RD_KEY4_DATA1_REG	BLOCK8 (KEY4) 的第 1 个寄存器内容。	0x0120	只读
EFUSE_RD_KEY4_DATA2_REG	BLOCK8 (KEY4) 的第 2 个寄存器内容。	0x0124	只读
EFUSE_RD_KEY4_DATA3_REG	BLOCK8 (KEY4) 的第 3 个寄存器内容。	0x0128	只读
EFUSE_RD_KEY4_DATA4_REG	BLOCK8 (KEY4) 的第 4 个寄存器内容。	0x012C	只读
EFUSE_RD_KEY4_DATA5_REG	BLOCK8 (KEY4) 的第 5 个寄存器内容。	0x0130	只读
EFUSE_RD_KEY4_DATA6_REG	BLOCK8 (KEY4) 的第 6 个寄存器内容。	0x0134	只读
EFUSE_RD_KEY4_DATA7_REG	BLOCK8 (KEY4) 的第 7 个寄存器内容。	0x0138	只读
EFUSE_RD_KEY5_DATA0_REG	BLOCK9 (KEY5) 的第 0 个寄存器内容。	0x013C	只读
EFUSE_RD_KEY5_DATA1_REG	BLOCK9 (KEY5) 的第 1 个寄存器内容。	0x0140	只读
EFUSE_RD_KEY5_DATA2_REG	BLOCK9 (KEY5) 的第 2 个寄存器内容。	0x0144	只读
EFUSE_RD_KEY5_DATA3_REG	BLOCK9 (KEY5) 的第 3 个寄存器内容。	0x0148	只读
EFUSE_RD_KEY5_DATA4_REG	BLOCK9 (KEY5) 的第 4 个寄存器内容。	0x014C	只读
EFUSE_RD_KEY5_DATA5_REG	BLOCK9 (KEY5) 的第 5 个寄存器内容。	0x0150	只读
EFUSE_RD_KEY5_DATA6_REG	BLOCK9 (KEY5) 的第 6 个寄存器内容。	0x0154	只读
EFUSE_RD_KEY5_DATA7_REG	BLOCK9 (KEY5) 的第 7 个寄存器内容。	0x0158	只读
EFUSE_RD_SYS_DATA_PART2_0_REG	BLOCK10 (system) 的第 0 个寄存器内容。	0x015C	只读
EFUSE_RD_SYS_DATA_PART2_1_REG	BLOCK10 (system) 的第 1 个寄存器内容。	0x0160	只读

名称	描述	地址	访问
EFUSE_RD_SYS_DATA_PART2_2_REG	BLOCK10 (system) 的第 2 个寄存器内容。	0x0164	只读
EFUSE_RD_SYS_DATA_PART2_3_REG	BLOCK10 (system) 的第 3 个寄存器内容。	0x0168	只读
EFUSE_RD_SYS_DATA_PART2_4_REG	BLOCK10 (system) 的第 4 个寄存器内容。	0x016C	只读
EFUSE_RD_SYS_DATA_PART2_5_REG	BLOCK10 (system) 的第 5 个寄存器内容。	0x0170	只读
EFUSE_RD_SYS_DATA_PART2_6_REG	BLOCK10 (system) 的第 6 个寄存器内容。	0x0174	只读
EFUSE_RD_SYS_DATA_PART2_7_REG	BLOCK10 (system) 的第 7 个寄存器内容。	0x0178	只读
<b>错误状态寄存器</b>			
EFUSE_RD_REPEAT_ERR0_REG	记录 BLOCK0 参数烧写错误第 0 个寄存器。	0x017C	只读
EFUSE_RD_REPEAT_ERR1_REG	记录 BLOCK0 参数烧写错误第 1 个寄存器。	0x0180	只读
EFUSE_RD_REPEAT_ERR2_REG	记录 BLOCK0 参数烧写错误第 2 个寄存器。	0x0184	只读
EFUSE_RD_REPEAT_ERR3_REG	记录 BLOCK0 参数烧写错误第 3 个寄存器。	0x0188	只读
EFUSE_RD_REPEAT_ERR4_REG	记录 BLOCK0 参数烧写错误第 4 个寄存器。	0x0190	只读
EFUSE_RD_RS_ERR0_REG	记录 BLOCK1-10 参数烧写错误信息的第 0 个寄存器。	0x01C0	只读
EFUSE_RD_RS_ERR1_REG	记录 BLOCK1-10 参数烧写错误信息的第 1 个寄存器。	0x01C4	只读
<b>控制/状态寄存器</b>			
EFUSE_CLK_REG	eFuse 时钟配置寄存器。	0x01C8	读/写
EFUSE_CONF_REG	eFuse 运行模式寄存器。	0x01CC	读/写
EFUSE_CMD_REG	eFuse 命令寄存器。	0x01D4	读/写
EFUSE_DAC_CONF_REG	eFuse 烧写电压控制寄存器。	0x01E8	读/写
EFUSE_STATUS_REG	eFuse 运行状态寄存器。	0x01D0	只读
<b>中断寄存器</b>			
EFUSE_INT_RAW_REG	eFuse 原始中断寄存器。	0x01D8	只读
EFUSE_INT_ST_REG	eFuse 中断状态寄存器。	0x01DC	只读
EFUSE_INT_ENA_REG	eFuse 中断使能寄存器。	0x01E0	读/写
EFUSE_INT_CLR_REG	eFuse 中断清除寄存器。	0x01E4	只写
<b>配置寄存器</b>			
EFUSE_RD_TIM_CONF_REG	eFuse 读取时序参数配置寄存器。	0x01EC	读/写
EFUSE_WR_TIM_CONF0_REG	eFuse 烧写时序参数第 0 个配置寄存器。	0x01F0	读/写
EFUSE_WR_TIM_CONF1_REG	eFuse 烧写时序参数第 1 个配置寄存器。	0x01F4	读/写
EFUSE_WR_TIM_CONF2_REG	eFuse 烧写时序参数第 2 个配置寄存器。	0x01F8	读/写
<b>版本寄存器</b>			
EFUSE_DATE_REG	版本控制寄存器。	0x01FC	读/写

## 14.6 寄存器

Register 14.1: EFUSE\_PGM\_DATA $_n$ \_REG ( $n$ : 0-7) (0x0000+4\* $n$ )

EFUSE_PGM_DATA $_n$	
31	0
0x000000	
Reset	

EFUSE\_PGM\_DATA $_n$  存放待烧写数据的第  $n$  个 32 位数据内容。(读/写)

Register 14.2: EFUSE\_PGM\_CHECK\_VALUE $_n$ \_REG ( $n$ : 0-2) (0x0020+4\* $n$ )

EFUSE_PGM_RS_DATA $_n$	
31	0
0x000000	
Reset	

EFUSE\_PGM\_RS\_DATA $_n$  存放待烧写 RS 代码的第  $n$  个 32 位数据内容。(读/写)

Register 14.3: EFUSE\_RD\_WR\_DIS\_REG (0x002C)

EFUSE_WR_DIS	
31	0
0x000000	
Reset	

EFUSE\_WR\_DIS 置位禁用 eFuse 烧写。(只读)

#### Register 14.4: EFUSE\_RD\_REPEAT\_DATA0\_REG (0x0030)

[illegible]

**EFUSE\_RD\_DIS** 置位禁止软件读取 eFuse Block4-10 的内容。(只读)

**EFUSE\_DIS\_RTC\_RAM\_BOOT** 置位禁止从 RTC RAM 启动。(只读)

**EFUSE\_DIS\_ICACHE** 置位禁用 ICache。(只读)

**EFUSE\_DIS\_DCACHE** 置位禁用 DCache。(只读)

**EFUSE\_DIS\_DOWNLOAD\_ICACHE** 置位在下载模式下关闭 ICache。(只读)

**EFUSE\_DIS\_DOWNLOAD\_DCACHE** 置位在下载模式下关闭 DCache。(只读)

**EFUSE DIS FORCE DOWNLOAD** 置位禁止强制芯片进入下载模式。(只读)

**EFUSE\_DIS\_USB** 置位关闭 USB OTG 功能。(只读)

**EFUSE\_DIS\_CAN** 置位关闭 TWAI 控制器功能。(只读)

**EFUSE\_DIS\_BOOT\_REMAP** 置位禁止 RAM 空间映射到 ROM 空间。(只读)

**EFUSE\_SOFT\_DIS\_JTAG** 软关断 JTAG 功能，用户还可以通过 HAMC 模块再次打开 JTAG。(只读)

**EFUSE\_HARD\_DIS\_JTAG** 硬关断 JTAG 功能，永久关断。（只读）

**EFUSE\_DIS\_DOWNLOAD\_MANUAL\_ENCRYPT** 置位在 download boot 模式下关闭 flash 加密功能。(只读)

**EFUSE\_USB\_EXCHG\_PINS** 置位交換 USB D+ 和 D- 管腳。(只讀)

**EFUSE\_EXT\_PHY\_ENABLE** 置位使能外部 USB PHY。(只读)

**EFUSE\_USB\_FORCE\_NOPERSIST** 置位强制设置 USB BAVLID 为 1。(只读)

**EFUSE\_RPT4\_RESERVED0** 保留（采用 4 备份编码）。（只读）

### Register 14.5: EFUSE\_RD\_REPEAT\_DATA1\_REG (0x0034)

[illegible]

**EFUSE\_VDD\_SPI\_XPD** 当 EFUSE\_VDD\_SPI\_FORCE 为 1 时，控制 SPI 调节器上电。（只读）

**EFUSE\_VDD\_SPI\_TIEH** 当 EFUSE\_VDD\_SPI\_FORCE 为 1 时，选择 VDD\_SPI 电压。0: VDD\_SPI 连接 1.8 V LDO；1: VDD SPI 连接 VDD RTC IO。（只读）

**EFUSE\_VDD\_SPI\_FORCE** 使用 EFUSE\_VDD\_SPI\_XPD 和 EFUSE\_VDD\_SPI\_TIEH 配置 VDD\_SPI。(只读)

**EFUSE\_WDT\_DELAY\_SEL** 选择 RTC 看门狗超时阈值。00: 40,000 个慢速时钟周期；01: 80,000 个慢速时钟周期；10: 160,000 个慢速时钟周期；11: 320,000 个慢速时钟周期。（只读）

**EFUSE\_SPI\_BOOT\_CRYPT\_CNT** 置位使能 SPI boot 加解密。奇数个 1：使能；偶数个 1：禁能。（只读）

**EFUSE\_SECURE\_BOOT\_KEY\_REVOKE0** 置位使能撤销第一个安全启动密钥。(只读)

**EFUSE\_SECURE\_BOOT\_KEY\_REVOKE1** 置位使能撤销第二个安全启动密钥。(只读)

**EFUSE\_SECURE\_BOOT\_KEY\_REVOKE2** 置位使能撤销第三个安全启动密钥。(只读)

**EFUSE\_KEY\_PURPOSE\_0** KEY0 purpose, 详见表 52: 密钥用途数值对应的含义。(只读)

**EFUSE KEY PURPOSE 1** KEY1 purpose, 详见表 52: 密钥用途数值对应的含义。(只读)

Register 14.6: EFUSE\_RD\_REPEAT\_DATA2\_REG (0x0038)

EFUSE_FLASH_TPUW				EFUSE_RPT4_RESERVED1				EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE				EFUSE_SECURE_BOOT_EN				(reserved)				EFUSE_KEY_PURPOSE_5				EFUSE_KEY_PURPOSE_4				EFUSE_KEY_PURPOSE_3				EFUSE_KEY_PURPOSE_2			
31	28	27	22	21	20	19	16	15	12	11	8	7	4	3	0																				
0x0				0x0				0	0	0	0	0	0	0	0	0	0x0	0x0		0x0		0x0		0x0		Reset									

**EFUSE\_KEY\_PURPOSE\_2** KEY2 purpose, 详见表 52: 密钥用途数值对应的含义。(只读)

**EFUSE\_KEY\_PURPOSE\_3** KEY3 purpose, 详见表 52: 密钥用途数值对应的含义。(只读)

**EFUSE\_KEY\_PURPOSE\_4** KEY4 purpose, 详见表 52: 密钥用途数值对应的含义。(只读)

**EFUSE\_KEY\_PURPOSE\_5** KEY5 purpose, 详见表 52: 密钥用途数值对应的含义。(只读)

**EFUSE\_SECURE\_BOOT\_EN** 置位使能安全启动。(只读)

**EFUSE\_SECURE\_BOOT\_AGGRESSIVE\_REVOKE** 置位使能密钥失效的激进策略。(只读)

**EFUSE\_RPT4\_RESERVED1** 保留 (采用 4 备份编码)。(只读)

**EFUSE\_FLASH\_TPUW** 配置上电后 flash 等待时间, 单位为 ms/2, 值为 15 时, 等待时间为 7.5 ms。(只读)

Register 14.7: EFUSE\_RD\_REPEAT\_DATA3\_REG (0x003C)

EFUSE_RPT4_RESERVED2																EFUSE_SECURE_VERSION																EFUSE_FORCE_SEND_RESUME																EFUSE_FLASH_TYPE																EFUSE_PIN_POWER_SELECTION																EFUSE_ENABLE_SECURITY_DOWNLOAD																EFUSE_DIS_USB_DOWNLOAD_MODE																EFUSE_RPT4_RESERVED3																EFUSE_UART_PRINT_CHANNEL																EFUSE_DIS_LEGACY_SPI_BOOT																EFUSE_DIS_DOWNLOAD_MODE															
31				27				26								11				10		9		8		7		6		5		4		3		2		1		0																																																																																																																																							
0x0								0x00								0		0		0		0x0		0		0		0		0		0		0		0		0		Reset																																																																																																																																							

**EFUSE\_DIS\_DOWNLOAD\_MODE** 置位关闭下载模式。(只读)

**EFUSE\_DIS\_LEGACY\_SPI\_BOOT** 置位关闭 Legacy SPI boot 模式。(只读)

**EFUSE\_UART\_PRINT\_CHANNEL** 选择打印 boot 信息的 UART 通道。0: UART0; 1: UART1。(只读)

**EFUSE\_RPT4\_RESERVED3** 保留 (采用 4 备份编码)。(只读)

**EFUSE\_DIS\_USB\_DOWNLOAD\_MODE** 置位在 UART download boot 模式下关闭 USB 功能。(只读)

**EFUSE\_ENABLE\_SECURITY\_DOWNLOAD** 置位使能安全 UART 下载模式 (仅支持读写 flash) (只读)

**EFUSE\_UART\_PRINT\_CONTROL** 控制 UART 打印方式。00: 强制打印; 01: 由 GPIO 46 控制, 低电平打印; 10: 由 GPIO 46 控制, 高电平打印; 11: 强制关闭打印。(只读)

**EFUSE\_PIN\_POWER\_SELECTION** SPI flash 启动时选择 GPIO33-GPIO37 的电源。0: VDD3P3\_CPU; 1: VDD\_SPI。(只读)

**EFUSE\_FLASH\_TYPE** Flash 类型。0: 4 根数据线; 1: 8 根数据线。(只读)

**EFUSE\_FORCE\_SEND\_RESUME** 置位强制 ROM 代码在 SPI 启动过程中发送恢复指令。(只读)

**EFUSE\_SECURE\_VERSION** 表明 IDF 安全版本 (用于 ESP-IDF 的防回滚功能)。(只读)

**EFUSE\_RPT4\_RESERVED2** 保留 (采用 4 备份编码)。(只读)

Register 14.8: EFUSE\_RD\_REPEAT\_DATA4\_REG (0x0040)

(reserved)								EFUSE_RPT4_RESERVED4																							
31								24	23																					0	
0	0	0	0	0	0	0	0	0x0000																							Reset

**EFUSE\_RPT4\_RESERVED4** 保留（采用 4 备份编码）。（只读）

Register 14.9: EFUSE\_RD\_MAC\_SPI\_SYS\_0\_REG (0x0044)

EFUSE_MAC_0																															
31																															0
0x000000																															
Reset																															

**EFUSE\_MAC\_0** 存储 MAC 地址低 32 位的内容。（只读）

Register 14.10: EFUSE\_RD\_MAC\_SPI\_SYS\_1\_REG (0x0048)

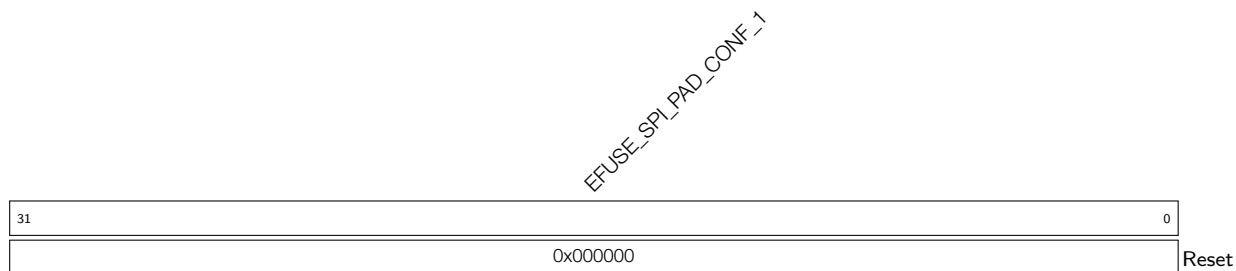
EFUSE_SPI_PAD_CONF_0																EFUSE_MAC_1														
31															16	15													0	
0x00																0x00														Reset

**EFUSE\_MAC\_1** 存储 MAC 地址高 16 位的内容。（只读）

**EFUSE\_SPI\_PAD\_CONF\_0** 存储 SPI\_PAD\_CONF 第 0 部分的内容。（只读）

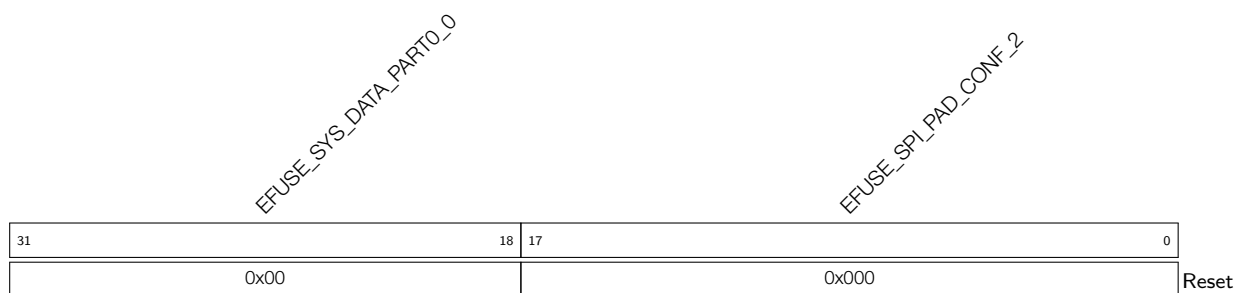


### Register 14.11: EFUSE\_RD\_MAC\_SPI\_SYS\_2\_REG (0x004C)



**EFUSE\_SPI\_PAD\_CONF\_1** 存储 SPI\_PAD\_CONF 第 1 部分的内容。(只读)

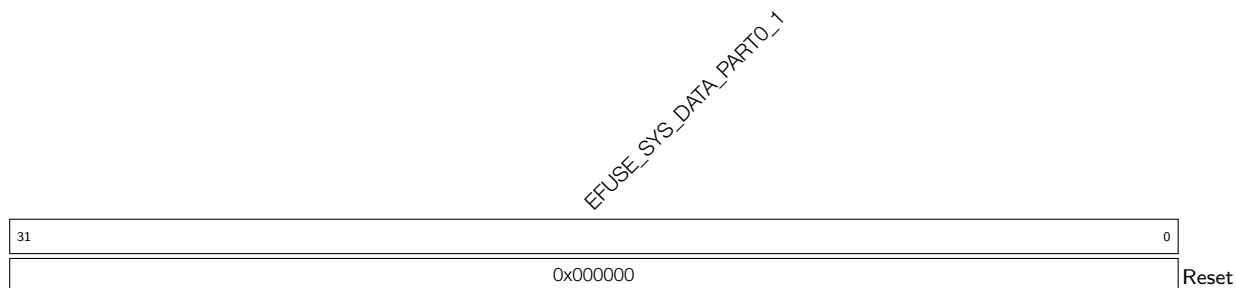
### Register 14.12: EFUSE\_RD\_MAC\_SPI\_SYS\_3\_REG (0x0050)



**EFUSE\_SPI\_PAD\_CONF\_2** 存儲 SPI\_PAD\_CONF 第 2 部分的内容。(只讀)

**EFUSE\_SYS\_DATA\_PART0\_0** 存储系统数据第 0 部分的第 0 部分的内容。(只读)

### Register 14.13: EFUSE\_RD\_MAC\_SPI\_SYS\_4\_REG (0x0054)



**EFUSE\_SYS\_DATA\_PART0\_1** 存储系统数据第 0 部分的第 1 部分的内容。(只读)

Register 14.14: EFUSE\_RD\_MAC\_SPI\_SYS\_5\_REG (0x0058)

EFUSE_SYS_DATA_PART0_2	
31	0
0x000000	
Reset	

EFUSE\_SYS\_DATA\_PART0\_2 存储系统数据第 0 部分的第 2 部分的内容。(只读)

Register 14.15: EFUSE\_RD\_SYS\_DATA\_PART1\_n\_REG ( $n$ : 0-7) (0x005C+4\*n)

EFUSE_SYS_DATA_PART1_n	
31	0
0x000000	
Reset	

EFUSE\_SYS\_DATA\_PART1\_n 存储系统数据第 1 部分的第  $n$  个 32 位的内容。(只读)

Register 14.16: EFUSE\_RD\_USR\_DATA\_n\_REG ( $n$ : 0-7) (0x007C+4\*n)

EFUSE_USR_DATA_n	
31	0
0x000000	
Reset	

EFUSE\_USR\_DATA\_n 存储 BLOCK3 (user) 的第  $n$  个 32 位的内容。(只读)

Register 14.17: EFUSE\_RD\_KEY0\_DATA $n$ \_REG ( $n$ : 0-7) (0x009C+4\* $n$ )

EFUSE_KEY0_DATA $n$	
31	0
0x000000	
	Reset

EFUSE\_KEY0\_DATA $n$  存储 KEY0 的第  $n$  个 32 位的内容。(只读)

Register 14.18: EFUSE\_RD\_KEY1\_DATA $n$ \_REG ( $n$ : 0-7) (0x00BC+4\* $n$ )

EFUSE_KEY1_DATA $n$	
31	0
0x000000	
	Reset

EFUSE\_KEY1\_DATA $n$  存储 KEY1 的第  $n$  个 32 位的内容。(只读)

Register 14.19: EFUSE\_RD\_KEY2\_DATA $n$ \_REG ( $n$ : 0-7) (0x00DC+4\* $n$ )

EFUSE_KEY2_DATA $n$	
31	0
0x000000	
	Reset

EFUSE\_KEY2\_DATA $n$  存储 KEY2 的第  $n$  个 32 位的内容。(只读)

Register 14.20: EFUSE\_RD\_KEY3\_DATA $n$ \_REG ( $n$ : 0-7) (0x00FC+4\* $n$ )

EFUSE_KEY3_DATA $n$	
31	0
0x000000	
	Reset

EFUSE\_KEY3\_DATA $n$  存储 KEY3 的第  $n$  个 32 位的内容。(只读)

Register 14.21: EFUSE\_RD\_KEY4\_DATA<sub>*n*</sub>\_REG (*n*: 0-7) (0x011C+4\**n*)

EFUSE_KEY4_DATA <sub><i>n</i></sub>	
31	0
0x000000	
Reset	

EFUSE\_KEY4\_DATA<sub>*n*</sub> 存储 KEY4 的第 *n* 个 32 位的内容。(只读)

Register 14.22: EFUSE\_RD\_KEY5\_DATA<sub>*n*</sub>\_REG (*n*: 0-7) (0x013C+4\**n*)

EFUSE_KEY5_DATA <sub><i>n</i></sub>	
31	0
0x000000	
Reset	

EFUSE\_KEY5\_DATA<sub>*n*</sub> 存储 KEY5 的第 *n* 个 32 位的内容。(只读)

Register 14.23: EFUSE\_RD\_SYS\_DATA\_PART2\_<sub>*n*</sub>\_REG (*n*: 0-7) (0x015C+4\**n*)

EFUSE_SYS_DATA_PART2_ <sub><i>n</i></sub>	
31	0
0x000000	
Reset	

EFUSE\_SYS\_DATA\_PART2\_<sub>*n*</sub> 存储系统数据第 2 部分的第 *n* 个 32 位的内容。(只读)

293

ESP32-S2 TRM (预发布 V0.2)

## 反馈文档意见

寄存器描述下一页继续。

## Register 14.24: EFUSE\_RD\_REPEAT\_ERR0\_REG (0x017C)

继上一页寄存器描述。

**EFUSE\_DIS\_DOWNLOAD\_MANUAL\_ENCRYPT\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_DIS\\_DOWNLOAD\\_MANUAL\\_ENCRYPT](#) 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_USB\_EXCHG\_PINS\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_USB\\_EXCHG\\_PINS](#) 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_EXT\_PHY\_ENABLE\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_EXT\\_PHY\\_ENABLE](#) 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_USB\_FORCE\_NOPERSIST\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_USB\\_FORCE\\_B](#) 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_RPT4\_RESERVED0\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_RPT4\\_RESERVED0](#) 中该位的四备份不一致，参数不可靠。（只读）

### Register 14.25: EFUSE RD REPEAT ERR1 REG (0x0180)

[illegible]

**EFUSE\_VDD\_SPI\_XPD\_ERR** 若该参数中任意位为 1, 表明对应 **EFUSE\_VDD\_SPI\_XPD** 中该位的四备份不一致, 参数不可靠。(只读)

**EFUSE\_VDD\_SPI\_TIEH\_ERR** 若该参数中任意位为 1，表明对应 EFUSE\_VDD\_SPI\_TIEH 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_VDD\_SPI\_FORCE\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_VDD\_SPI\_FORCE** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_WDT\_DELAY\_SEL\_ERR** 若该参数中任意位为 1，表明对应 EFUSE\_WDT\_DELAY\_SEL 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_SPI\_BOOT\_CRYPT\_CNT\_ERR** 若该参数中任意位为 1，表明对应 EFUSE SPI BOOT CRYPT CNT 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_SECURE\_BOOT\_KEY\_REVOKE0\_ERR** 若该参数中任意位为 1，表明对应 EFUSE\_SECURE\_BOOT\_KEY\_REVOKE0 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_SECURE\_BOOT\_KEY\_REVOKE1\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_SECURE\_BOOT\_KEY\_REVOKE1** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_SECURE\_BOOT\_KEY\_REVOKE2\_ERR** 若该参数中任意位为 1，表明对应 EFUSE\_SECURE\_BOOT\_KEY\_REVOKE2 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_KEY\_PURPOSE\_0\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_KEY\_PURPOSE\_0** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_KEY\_PURPOSE\_1\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_KEY\_PURPOSE\_1** 中该位的四备份不一致，参数不可靠。（只读）

Register 14.26: EFUSE\_RD\_REPEAT\_ERR2\_REG (0x0184)

EFUSE_FLASH_TPUW_ERR				EFUSE_RPT4_RESERVED1_ERR				EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR				EFUSE_SECURE_BOOT_EN_ERR				(reserved)				EFUSE_KEY_PURPOSE_5_ERR				EFUSE_KEY_PURPOSE_4_ERR				EFUSE_KEY_PURPOSE_3_ERR				EFUSE_KEY_PURPOSE_2_ERR			
31	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						
0x0				0x0				0	0	0	0	0	0	0	0x0				0x0				0x0				0x0				Reset				

**EFUSE\_KEY\_PURPOSE\_2\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_KEY\\_PURPOSE\\_2](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_KEY\_PURPOSE\_3\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_KEY\\_PURPOSE\\_3](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_KEY\_PURPOSE\_4\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_KEY\\_PURPOSE\\_4](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_KEY\_PURPOSE\_5\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_KEY\\_PURPOSE\\_5](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_SECURE\_BOOT\_EN\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_SECURE\\_BOOT\\_EN](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_SECURE\_BOOT\_AGGRESSIVE\_REVOKE\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_SECURE\\_BOOT\\_AGGRESSIVE\\_REVOKE](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_RPT4\_RESERVED1\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_RPT4\\_RESERVED1](#) 中该位的四备份不一致，参数不可靠。(只读)

**EFUSE\_FLASH\_TPUW\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_FLASH\\_TPUM](#) 中该位的四备份不一致，参数不可靠。(只读)



### Register 14.27: EFUSE RD REPEAT ERR3 REG (0x0188)

31		27		26		11		10	9	8	7	6	5	4	3	2	1	0
0x0				0x00				0	0	0	0x0	0	0	0	0	0	0	0
EFUSE_RPT4_RESERVED2_ERR				EFUSE_SECURE_VERSION_ERR				EFUSE_FORCE_SEND_RESUME_ERR	EFUSE_FLASH_TYPE_ERR	EFUSE_PIN_POWER_ERR	EFUSE_UART_PRINT_SELECTION_ERR	EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR	EFUSE_DIS_USB_DOWNLOAD_MODE_ERR	EFUSE_RPT4_RESERVED3_ERR	EFUSE_UART_PRINT_CHANNEL_ERR	EFUSE_DIS_LEGACY_SPI_BOOT_ERR	EFUSE_DIS_DOWNLOAD_MODE_ERR	
Reset																		

**EFUSE\_DIS\_DOWNLOAD\_MODE\_ERR** 若该参数中任意位为 1，表明对应 EFUSE DIS DOWNLOAD MODE 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_DIS\_LEGACY\_SPI\_BOOT\_ERR** 若该参数中任意位为 1，表明对应 EFUSE DIS LEGACY SPI BOOT 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_UART\_PRINT\_CHANNEL\_ERR** 若该参数中任意位为 1，表明对应 EFUSE UART PRINT CHANNEL 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_RPT4\_RESERVED3\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_RPT4\_RESERVED3** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_DIS\_USB\_DOWNLOAD\_MODE\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_DIS\_USB\_DOWNLOAD\_MODE** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_ENABLE\_SECURITY\_DOWNLOAD\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_ENABLE\_SECURITY\_DOWNLOAD** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_UART\_PRINT\_CONTROL\_ERR** 若该参数中任意位为 1，表明对应 EFUSE UART PRINT CONTROL 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_PIN\_POWER\_SELECTION\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_PIN\_POWER\_SELECTION** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_FLASH\_TYPE\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_FLASH\_TYPE** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_FORCE\_SEND\_RESUME\_ERR** 若该参数中任意位为 1，表明对应 [EFUSE\\_FORCE\\_SEND\\_RESUME](#) 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_SECURE\_VERSION\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_SECURE\_VERSION** 中该位的四备份不一致，参数不可靠。（只读）

**EFUSE\_RPT4\_RESERVED2\_ERR** 若该参数中任意位为 1，表明对应 EFUSE\_RPT4\_RESERVED2 中该位的四备份不一致，参数不可靠。（只读）

### Register 14.28: EFUSE\_RD\_REPEAT\_ERR4\_REG (0x0190)

(reserved)								EFUSE_PPT4_RESERVED4_ERR																																							
31								24								23																								0							
0 0 0 0 0 0 0 0								0x0000																								Reset															

**EFUSE\_RPT4\_RESERVED4\_ERR** 若该参数中任意位为 1，表明对应 **EFUSE\_RPT4\_RESERVED4** 中该位的四备份不一致，参数不可靠。（只读）

### Register 14.29: EFUSE\_RD\_RS\_ERR0\_REG (0x01C0)

[illegible]

**EFUSE\_MAC\_SPI\_8M\_ERR\_NUM** 指示 BLOCK1 中的错误字节的个数。(只读)

**EFUSE\_MAC\_SPI\_8M\_FAIL** 0: 代表没有烧写错误, BLOCK1 里的数据是可靠的; 1: 代表烧写 BLOCK1 失败, 错误字节数超过 5 个。(只读)

**EFUSE SYS PART1 NUM** 指示 BLOCK2 中的错误字节的个数。(只读)

**EFUSE\_SYS\_PART1\_FAIL** 0: 代表没有烧写错误, BLOCK2 里的数据是可靠的; 1: 代表烧写 BLOCK2 失败, 错误字节数超过 5 个。(只读)

**EFUSE\_USR\_DATA\_ERR\_NUM** 指示 BLOCK3 中的错误字节的个数。(只读)

**EFUSE\_USR\_DATA\_FAIL** 0: 代表没有烧写错误, BLOCK3 里的数据是可靠的; 1: 代表烧写 BLOCK3 失败, 错误字节数超过 5 个。(只读)

**EFUSE\_KEY<sub>*n*</sub>\_ERR\_NUM** 指示 KEY<sub>*n*</sub> 中的错误字节的个数。(只读)

**EFUSE\_KEY $n$ \_FAIL** 0: 代表没有烧写错误, key $n$  数据是可靠的; 1: 代表 key $n$  烧写失败, 错误字节数超过 5 个。(只读)

Register 14.30: EFUSE\_RD\_RS\_ERR1\_REG (0x01C4)

(reserved)																								EFUSE_SYS_PART2_FAIL				EFUSE_SYS_PART2_ERR_NUM				EFUSE_KEY5_FAIL				EFUSE_KEY5_ERR_NUM						
31																								8	7	6	4		3	2	0											
0 0																								0	0x0		0	0x0		Reset												

**EFUSE\_KEY5\_ERR\_NUM** 指示 KEY5 中的错误字节的个数。(只读)

**EFUSE\_KEY5\_FAIL** 0: 代表没有烧写错误, KEY5 里的数据是可靠的; 1: 代表烧写 KEY5 失败, 错误字节数超过 5 个。(只读)

**EFUSE\_SYS\_PART2\_ERR\_NUM** 指示 BLOCK10 中的错误字节的个数。(只读)

**EFUSE\_SYS\_PART2\_FAIL** 0: 代表没有烧写错误, BLOCK10 里的数据是可靠的; 1: 代表烧写 BLOCK10 失败, 错误字节数超过 5 个。(只读)

Register 14.31: EFUSE\_CLK\_REG (0x01C8)

(reserved)																EFUSE_CLK_EN																(reserved)																EFUSE_EFUSE_MEM_FORCE_PU				EFUSE_MEM_CLK_FORCE_ON				EFUSE_EFUSE_MEM_FORCE_PD			
31																17	16	15															3	2	1	0																							
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0	0 0																0	1	0	Reset																							

**EFUSE\_EFUSE\_MEM\_FORCE\_PD** 置位强制使 eFuse SRAM 进入低功耗模式。(读/写)

**EFUSE\_MEM\_CLK\_FORCE\_ON** 置位强制激活 eFuse SRAM 的时钟信号。(读/写)

**EFUSE\_EFUSE\_MEM\_FORCE\_PU** 置位强制使 eFuse SRAM 进入工作模式。(读/写)

**EFUSE\_CLK\_EN** 置位强制使能 eFuse memory 的时钟信号。(读/写)

### Register 14.32: EFUSE\_CONF\_REG (0x01CC)

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

**EFUSE\_OP\_CODE** 0x5A5A: 运行烧写指令; 0x5AA5: 运行读取指令。(读/写)

### Register 14.33: EFUSE\_CMD\_REG (0x01D4)

[illegible]

**EFUSE\_READ\_CMD** 置位发送读取指令。(读/写)

**EFUSE\_PGM\_CMD** 置位发送烧写指令。(读/写)

**EFUSE\_BLK\_NUM** 表明烧写哪个块，值 0-10 分别对应 BLOCK0-10。（读/写）

### Register 14.34: EFUSE\_DAC\_CONF\_REG (0x01E8)

(reserved)																EFUSE_OE_CLR	EFUSE_DAC_NUM								EFUSE_DAC_CLK_PAD_SEL								EFUSE_DAC_CLK_DIV							
31																18	17	16									9	8	7									0		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	255	0	28									Reset												

**EFUSE\_DAC\_CLK\_DIV** 烧写电压的爬升时钟分频系数。(读/写)

**EFUSE\_DAC\_CLK\_PAD\_SEL** 无关项。(读/写)

**EFUSE\_DAC\_NUM** 烧写供电的上升周期。(读/写)

**EFUSE\_OE\_CLR** 降低烧写电压的供电能力。(读/写)

Register 14.35: EFUSE\_STATUS\_REG (0x01D0)

(reserved)																EFUSE_REPEAT_ERR_CNT					(reserved)					EFUSE_STATE		
311817109430																												
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x0					0 0 0 0 0 0 0					0x0		Reset

**EFUSE\_STATE** 表明 eFuse 状态机所处的状态。(只读)

**EFUSE\_REPEAT\_ERR\_CNT** 表明烧写 BLOCK0 时的错误位的个数。(只读)

Register 14.36: EFUSE\_INT\_RAW\_REG (0x01D8)

(reserved)																															EFUSE_PGM_DONE_INT_RAW		EFUSE_READ_DONE_INT_RAW	
31																															2	1	0	
0 0																															0	0	Reset	

**EFUSE\_READ\_DONE\_INT\_RAW** 读取完成中断的原始中断状态位。(只读)

**EFUSE\_PGM\_DONE\_INT\_RAW** 烧写完成中断的原始中断状态位。(只读)

Register 14.37: EFUSE\_INT\_ST\_REG (0x01DC)

(reserved)																															EFUSE_PGM_DONE_INT_ST			EFUSE_READ_DONE_INT_ST																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0</

**EFUSE\_READ\_DONE\_INT\_ST** 读取完成中断的状态位。(只读)

**EFUSE\_PGM\_DONE\_INT\_ST** 烧写完成中断的状态位。(只读)

Register 14.38: EFUSE\_INT\_ENA\_REG (0x01E0)

(reserved)																															EFUSE_PGM_DONE EFUSE_READ_DONE				
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

**EFUSE\_READ\_DONE\_INT\_ENA** 读取完成中断的使能位。(读/写)

**EFUSE\_PGM\_DONE\_INT\_ENA** 烧写完成中断的使能位。(读/写)

Register 14.39: EFUSE\_INT\_CLR\_REG (0x01E4)

(reserved)																															EFUSE_PGM_DONE EFUSE_READ_DONE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																										
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**EFUSE\_READ\_DONE\_INT\_CLR** 读取完成中断的清除位。(只写)

**EFUSE\_PGM\_DONE\_INT\_CLR** 烧写完成中断的清除位。(只写)

Register 14.40: EFUSE\_RD\_TIM\_CONF\_REG (0x01EC)

EFUSE_READ_INIT_NUM																EFUSE_TSR_A																EFUSE_TRD																EFUSE_THR_A																															
31								24								23								16								15								8								7								0																							
0x12																0x1																0x1																0x1																Reset															

**EFUSE\_THR\_A** 配置读取操作的保持时间。(读/写)

**EFUSE\_TRD** 配置读取操作的脉冲长度。(读/写)

**EFUSE\_TSR\_A** 配置读取操作的建立时间。(读/写)

**EFUSE\_READ\_INIT\_NUM** 配置 eFuse 的初始读取时间。(读/写)

Register 14.41: EFUSE\_WR\_TIM\_CONF0\_REG (0x01F0)

EFUSE_TPGM																EFUSE_TPGM_INACTIVE								EFUSE_THP_A										
31																16	15								8	7								0
0xc8																0x1								0x1								Reset		

**EFUSE\_THP\_A** 配置烧写操作的保持时间。(读/写)

**EFUSE\_TPGM\_INACTIVE** 配置烧写 eFuse 为 0 时的脉冲长度。(读/写)

**EFUSE\_TPGM** 配置烧写 eFuse 为 1 时的脉冲长度。(读/写)

Register 14.42: EFUSE\_WR\_TIM\_CONF1\_REG (0x01F4)

(reserved)																EFUSE_PWR_ON_NUM																EFUSE_TSUP_A																																															
31								24								23								8								7								0																																							
0								0								0								0								0								0								0								0x2880								0x1								Reset							

**EFUSE\_TSUP\_A** 配置烧写操作的建立时间。(读/写)

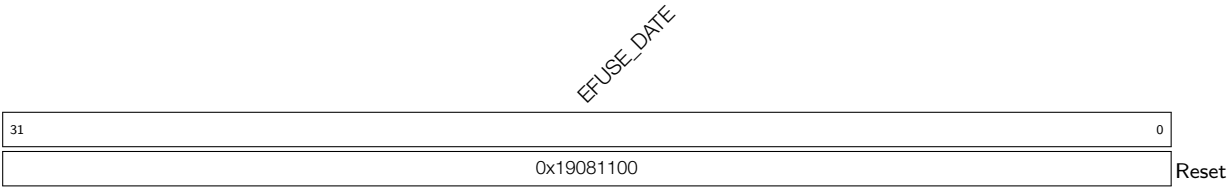
**EFUSE\_PWR\_ON\_NUM** 配置烧写电压 VDDQ 的上升时间。(读/写)

Register 14.43: EFUSE\_WR\_TIM\_CONF2\_REG (0x01F8)

(reserved)																EFUSE_PWR_OFF_NUM																																																																																																																																																															
31																16																15																0																																																																																																																															
0																0																0																0																0																0																0																0																0																0x190																Reset															

**EFUSE\_PWR\_OFF\_NUM** 配置烧写电压 VDDQ 的下降时间。(读/写)

Register 14.44: EFUSE\_DATE\_REG (0x01FC)



**EFUSE\_DATE** 版本控制寄存器。(读/写)



## 15. I<sup>2</sup>C 控制器

### 15.1 概述

I<sup>2</sup>C (Inter-Integrated Circuit) 总线用于使 ESP32-S2 和多个外部设备进行通信。多个外部设备可以共用一个 I<sup>2</sup>C 总线。

### 15.2 主要特性

I<sup>2</sup>C 具有以下几个特点。

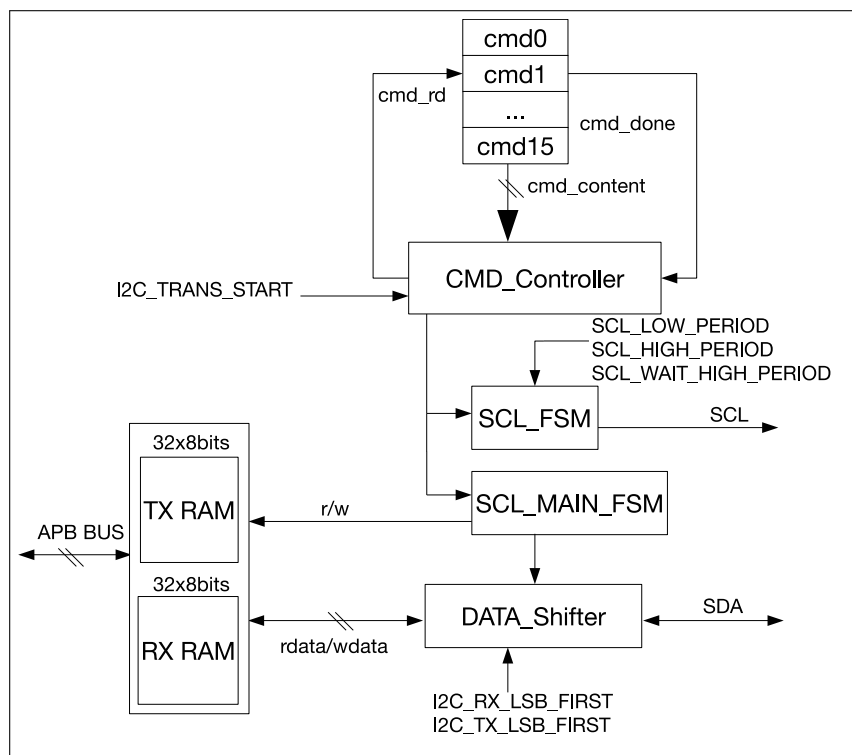
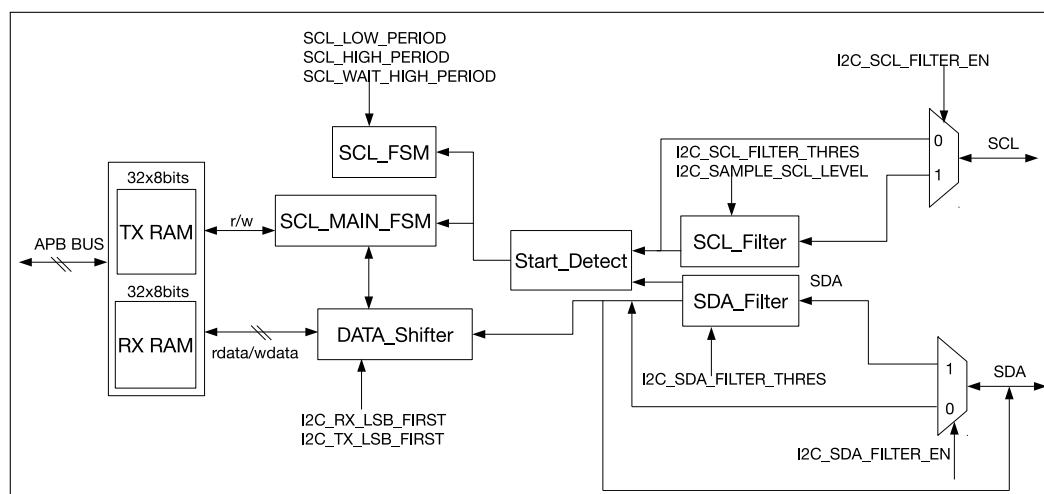
- 支持主机模式以及从机模式
- 支持多主机多从机通信
- 支持标准模式 (100 kbit/s)
- 支持快速模式 (400 kbit/s)
- 支持 7-bit 以及 10-bit 模式寻址
- 支持拉低 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能
- 支持双寻址模式

### 15.3 I<sup>2</sup>C 功能描述

#### 15.3.1 I<sup>2</sup>C 简介

I<sup>2</sup>C 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，I<sup>2</sup>C 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，则通讯开始：在 SCL 为高电平时拉低 SDA 线，主机将通过 SCL 线发出 9 个时钟脉冲。前 8 个脉冲用于按位传输，该字节包括 7-bit 地址和 1 个读写位。如果从机地址与该 7-bit 地址一致，那么从机可以通过在第 9 个脉冲上拉低 SDA 线来应答。接下来，根据读 / 写标志位，主机和从机可以发送 / 接收更多的数据。根据应答位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，发送一个停止标志：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。

15.3.2 I<sup>2</sup>C 架构图 15-1. I<sup>2</sup>C Master 基本架构图 15-2. I<sup>2</sup>C Slave 基本架构

I<sup>2</sup>C 控制器可以工作于 Master 模式或者 Slave 模式，`I2C_MS_MODE` 寄存器用于模式选择。图 15-1 为 I<sup>2</sup>C Master 基本架构图，图 15-2 为 I<sup>2</sup>C Slave 基本架构图。I<sup>2</sup>C 控制器内部包括的模块主要有 TX/RX RAM、CMD\_Controller、SCL\_FSM、SCL\_MAIN\_FSM、DATA\_Shifter、SCL\_Filter 和 SDA\_Filter。

### 15.3.2.1 TX/RX RAM

TX/RX RAM 大小均为 32 x 8 bits。TX RAM 用于存储 I<sup>2</sup>C 控制器需要发送的数据。在 I<sup>2</sup>C 通信的过程中，当 I<sup>2</sup>C 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 I<sup>2</sup>C 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 I<sup>2</sup>C 控制器工作于从机模式时，TX RAM 中只存放写数据。

RX RAM 存储的是 I<sup>2</sup>C 通信过程中，I<sup>2</sup>C 控制器接收到的数据。当 I<sup>2</sup>C 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 I<sup>2</sup>C 通信结束后，读出 RX RAM 的值。

TX RAM 和 RX RAM 均可以通过 FIFO 和直接地址 (non-FIFO) 两种方式访问，具体访问方式通过 I2C\_NONFIFO\_EN 位配置。

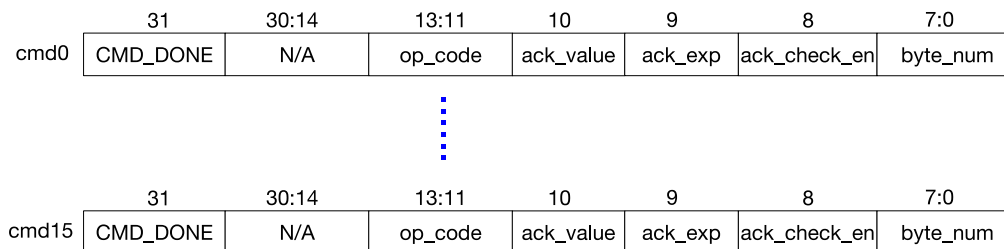
TX RAM 可被 CPU 读写。CPU 可通过两种方式写 TX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 `I2C_DATA_REG` 写 TX RAM，硬件自动进行 TX RAM 写地址自增。直接地址访问是通过地址段 (`I2C_基地址 + 0x100`) ~ (`I2C_基地址 + 0x17C`) 直接访问 TX RAM。TX RAM 的每一个字节占据一个 word 的地址。因此，第一个字节访问地址为 `I2C_基地址 + 0x100`，第二字节访问地址为 `I2C_基地址 + 0x104`，第三字节访问地址为 `I2C_基地址 + 0x108`，以此类推。CPU 只可通过直接地址访问方式读 TX RAM，读 TX RAM 的地址相比于写 TX RAM 地址需要减去 0x80 的偏移。

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](#) + 0x100) ~ ([I2C 基地址](#) + 0x17C) 直接访问 RX RAM。RX RAM 的每一个字节占据一个 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。

TX RAM 的写地址和 RX RAM 的读地址范围一样，因此可以把 TX RAM 和 RX RAM 看成一块 32 x 8 bits 的 RAM。本文在后续章节中都以 RAM 来代替 TX RAM 和 RX RAM。

### 15.3.2.2 CMD Controller

I<sup>2</sup>C 控制器工作于主机模式时，CMD\_Controller 会依次从 16 个命令寄存器中读出命令并按照命令来控制 SCL FSM 及 SDA FSM。

图 15-3. I<sup>2</sup>C 命令寄存器结构

命令寄存器只在 I<sup>2</sup>C 控制器工作于主机模式时才有效，其内部结构如图 15-3 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 5 种命令。
  - RSTART: op\_code 等于 0 时为 RSTART 命令, 该命令指示 I<sup>2</sup>C 控制器发送 I<sup>2</sup>C 协议中的 START 位以及 RSTART 位。
  - WRITE: op\_code 等于 1 时为 WRITE 命令, 该命令指示 I<sup>2</sup>C 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。
  - READ: op\_code 等于 2 时为 READ 命令, 该命令指示 I<sup>2</sup>C 控制器从从机读取数据。
  - STOP: op\_code 等于 3 时为 STOP 命令, 该命令指示 I<sup>2</sup>C 控制器发送 I<sup>2</sup>C 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。
  - END: op\_code 等于 4 时为 END 命令, 该命令指示 I<sup>2</sup>C 控制器将 SCL 信号拉低, 暂停 I<sup>2</sup>C 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 I<sup>2</sup>C 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。
3. ack\_value: 该位设置读操作时 I<sup>2</sup>C 控制器在 I<sup>2</sup>C 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 I<sup>2</sup>C 控制器在 I<sup>2</sup>C 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 I<sup>2</sup>C 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, I<sup>2</sup>C Master 会产生 I2C\_NACK\_INT 中断, 停止发送数据并产生 STOP。1: 检测从机发送的 ACK 位电平; 0: 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 16 个命令寄存器中必须有 STOP 或 END 命令。

一次完整的 I<sup>2</sup>C 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 I<sup>2</sup>C 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、数据长度和发送数据等。这样可以弥补 RAM 大小不足的问题, 也可以实现更灵活的 I<sup>2</sup>C 通信。

### 15.3.2.3 SCL\_FSM

SCL\_FSM 单元模块控制 SCL 时钟线。I2C\_SCL\_LOW\_PERIOD\_REG、I2C\_SCL\_HIGH\_PERIOD\_REG 和 I2C\_SCL\_WAIT\_HIGH\_PERIOD 用于配置 SCL 的频率和占空比。当 SCL\_FSM 长时间处于非空闲状态, 且时间超过 I2C\_SCL\_ST\_TO 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。

### 15.3.2.4 SCL\_MAIN\_FSM

SCL\_MAIN\_FSM 单元模块控制 SDA 数据线以及数据的存取。当 SCL\_MAIN\_FSM 长时间处于非空闲状态，且时间超过 I2C\_SCL\_MAIN\_ST\_TO 个模块时钟后，会触发 I2C\_SCL\_MAIN\_ST\_TO\_INT 中断，状态机会回到空闲状态。

### 15.3.2.5 DATA\_Shifter

DATA\_Shifter 模块用于串并转换，将字节数据转化成比特流或者将比特流转化成字节数据。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。

### 15.3.2.6 SCL\_Filter 和 SDA\_Filter

SCL\_Filter 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，SCL\_Filter 滤波器的功能为连续采样输入信号 SCL，如果输入信号在连续 I2C\_SCL\_FILTER\_THRES 个 APB 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 滤波器会过滤脉冲宽度小于 I2C\_SCL\_FILTER\_THRES 以及 I2C\_SDA\_FILTER\_THRES 个 APB 时钟周期的线路毛刺。

## 15.3.3 I<sup>2</sup>C 总线时序

I<sup>2</sup>C 控制器计时的时钟源可以用 APB\_CLK，也可以用 REF\_TICK。I2C\_REF\_ALWAYS\_ON 置 1 选择 APB\_CLK，清零选择 REF\_TICK。

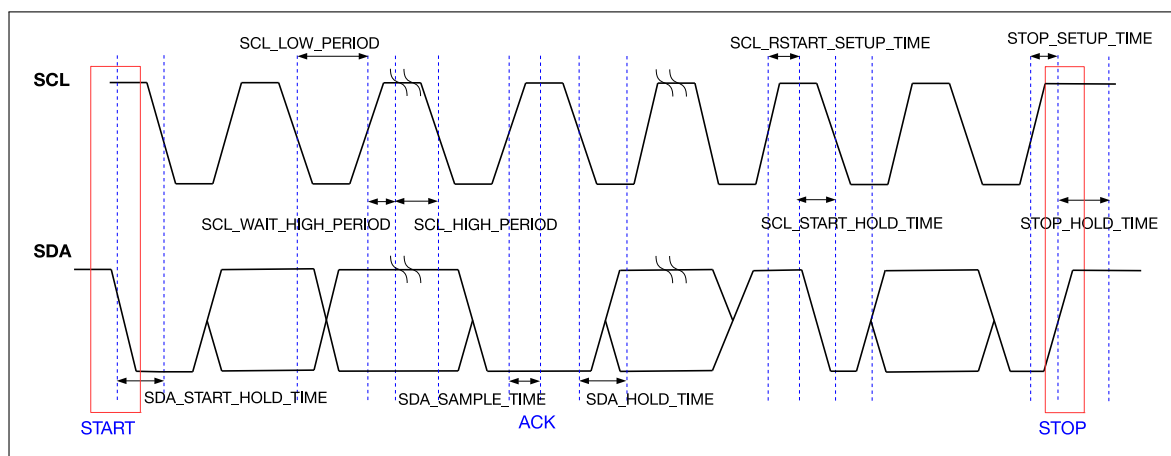


图 15-4. I<sup>2</sup>C 时序图

图 15-4 为 I<sup>2</sup>C 主机的时序图，图中的参数是以模块时钟 (I2C\_CLK) 为单位的，即 I2C\_REF\_ALWAYS\_ON 为 1 时，以  $T_{APB\_CLK}$  为单位；I2C\_REF\_ALWAYS\_ON 为 0 时，以  $T_{REF\_TICK}$  为单位。I<sup>2</sup>C 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 15-4 中所示的寄存器进行配置。如图 15-4 所示，各参数的含义如下：

1. `I2C_SCL_START_HOLD_TIME` 生成 I<sup>2</sup>C 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。但是如果外设拉低 SCL，I<sup>2</sup>C 控制器执行 END 命令拉低 SCL，或者控制器发生 SCL 延展传输则可能会导致 SCL 低电平时间变长。仅控制器工作在主机模式时有意义。
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：

$$f_{scl} = \frac{f_{I2C\_CLK}}{I2C\_SCL\_LOW\_PERIOD + 1 + I2C\_SCL\_HIGH\_PERIOD + I2C\_SCL\_WAIT\_HIGH\_PERIOD}$$

5. `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。
6. `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。

SCL 及 SDA 线采用 open-drain 的驱动方式。I<sup>2</sup>C 控制器有两种配置方式实现 open-drain 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_PINn_PAD_DRIVER` 寄存器为 open-drain 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，I<sup>2</sup>C 的输出频率受限于 SCL 和 SDA 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。

## 15.4 典型应用

为了便于描述，下文所有图示中的 I<sup>2</sup>C Master 和 Slave 都假定为 I<sup>2</sup>C 外设控制器。



### 15.4.1 I<sup>2</sup>C 主机写入从机，7-bit 寻址，单次命令序列

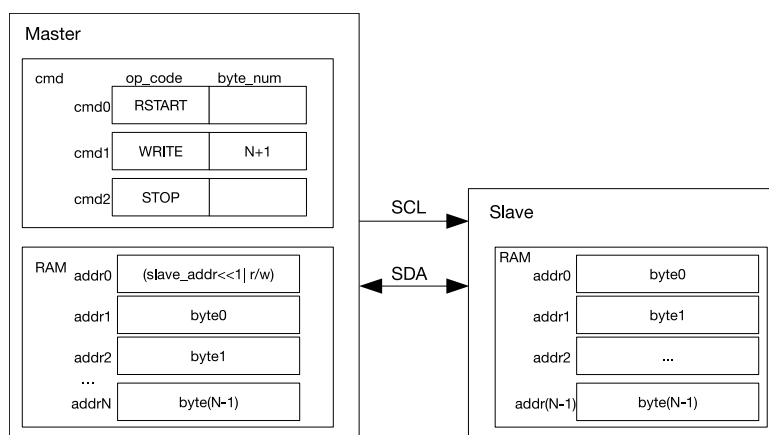


图 15-5. I<sup>2</sup>C Master 写 7-bit 寻址的 Slave

图 15-5 为 I<sup>2</sup>C Master 采用 7-bit 寻址写 N 个字节数据到 I<sup>2</sup>C Slave 的命令寄存器及 RAM 的值。如图 15-5 所示，主机 RAM 中第一个字节数据为 7-bit Slave 地址 + 1-bit 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. 控制器会等待 SCL 线为高电平，以避免 SCL 线被其他 Master 或者 Slave 占用。
2. 控制器执行 RSTART 命令发送 START 位。
3. 控制器执行 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 I<sup>2</sup>C Master 完成 STOP 位的传输后，会产生 `I2C_TRANS_COMPLETE_INT` 中断。

当需要传输的数据量太大，超过 32 字节时，可以对 RAM 使用乒乓操作。在控制器发送数据的同时，软件更新已发送的数据。当 I<sup>2</sup>C Master 的 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。

软件在检测到该中断后，向使用完的 RAM 区域更新新的数据。当 RAM 采用 non-FIFO 访问时，软件可以配置 `I2C_TX_UPDATE` 锁存已发送数据在 RAM 中的首末地址，通过读取 `I2C_FIFO_ST_REG` 寄存器中 `I2C_TXFIFO_START_ADDR` 段及 `I2C_TXFIFO_END_ADDR` 得到已发送数据在 RAM 中的首末地址，从而更新 RAM 中的旧数据。当 RAM 采用 FIFO 访问时，直接通过 `I2C_DATA_REG` 寄存器写入新的数据就可以。

有如下两种情况会打断控制器执行命令序列：

1. 当 I<sup>2</sup>C Master WRITE 命令中的 `ack_check_en` 配置为 1 时，I<sup>2</sup>C Master 会在发送完每个字节之后进行 ACK 检测。如果接收的 ACK 值与 WRITE 命令中的 `ack_exp` 电平不一致时，I<sup>2</sup>C Master 会产生 `I2C_NACK_INT` 中断，停止发送数据并且产生 STOP。
2. I<sup>2</sup>C Master 在 SCL 为高电平期间，检测到 SDA 输入值与 SDA 输出值不等时，则 I<sup>2</sup>C Master 会产生 `I2C_ARBITRATION_LOST_INT` 中断，并停止命令序列的执行返回 IDLE 状态，释放对 SCL 及 SDA 线的控制。

I<sup>2</sup>C Slave 在检测到 I<sup>2</sup>C Master 发送的 START 位之后，开始接收地址并进行地址匹配。当 I<sup>2</sup>C Slave 接收的地址与其 `I2C_SLAVE_ADDR[6:0]` 的值不匹配时，I<sup>2</sup>C Slave 停止接收数据。当地址匹配后，I<sup>2</sup>C Slave 将接下来接收的数据按照顺序存储到 RAM 中。

当需要接收的数据量太大，超过 32 字节时，可以对 RAM 使用乒乓操作。在控制器接收数据的同时，软件回收已接收的数据。当 I<sup>2</sup>C Slave 的 RAM 中接收的待回收的数据字节数大于等于 `I2C_RXFIFO_WM_THRHD` 时，会产生 `I2C_RXFIFO_WM_INT` 中断。

软件在检测到该中断后，需要回收相应的 RAM 区域数据。当 RAM 采用 non-FIFO 访问时，软件可以配置 I2C

`_RX_UPDATE` 锁存已待回收数据在 RAM 中的首末地址，通过读取 `RD_FIFO_ST_REG` 寄存器中 `RXFIFO_START_ADDR` 段及 `RXFIFO_END_ADDR` 得到已发送数据在 RAM 中的首末地址，从而回收 RAM 中的接收数据。当 RAM 采用 FIFO 访问时，直接通过 `I2C_DATA_REG` 寄存器回收数据就可以。

#### 15.4.2 I<sup>2</sup>C 主机写入从机，10-bit 寻址，单次命令序列

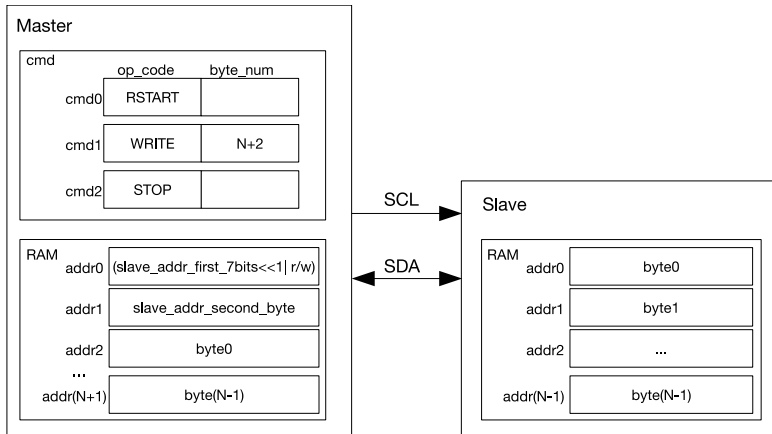


图 15-6. I<sup>2</sup>C Master 写 10-bit 寻址的 Slave

假设从机地址为 `SLV_ADDR`。ESP32-S2 I<sup>2</sup>C 控制器可以使用 7-bit 寻址 (`SLV_ADDR[6:0]`)，也可以使用 10-bit 寻址 (`SLV_ADDR[9:0]`)。

图 15-6 为 I<sup>2</sup>C Master 写 N 个字节到 10-bit 地址 I<sup>2</sup>C Slave 的配置图。主机模式下，相比于 7-bit 寻址，10-bit 寻址需要发送两字节地址段。将从机地址的第一个 7 bits `slave_addr_first_7bits` 和读写标志位存入 RAM 的 `addr0` 地址，`slave_addr_first_7bits` 的值应该配置为 `(0x78 | SLV_ADDR[9:8])`。接着将 `slave_addr_second_byte` 存入 RAM 的 `addr1` 地址，`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-bit 寻址模式。`I2C_SLAVE_ADDR` 用于配置 I<sup>2</sup>C Slave 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-bit Slave 地址比 7-bit 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。



### 15.4.3 I<sup>2</sup>C 主机写入从机，7-bit 双地址寻址，单次命令序列

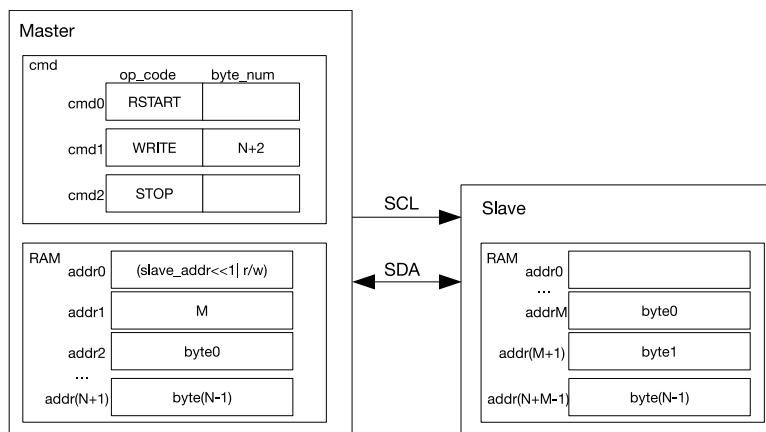


图 15-7. I<sup>2</sup>C Master 写 7-bit 寻址 Slave 的 M 地址 RAM

控制器处于 slave 模式时，还支持双地址访问方式。双地址的第一个地址是 I<sup>2</sup>C 从机地址，第二个地址是 I<sup>2</sup>C 从机的内存地址。双地址模式下，RAM 必须采用 non-FIFO 方式访问。通过置位 I2C\_FIFO\_ADDR\_CFG\_EN 来使能双地址访问功能。如图 15-7 所示，I<sup>2</sup>C Slave 将接收到的数据 byte0 ~ byte(N-1) 从 Slave RAM 中的 addrM 开始依次存储。当超出地址 31 后会从地址 0 开始继续存储。

### 15.4.4 I<sup>2</sup>C 主机写入从机，7-bit 寻址，多次命令序列

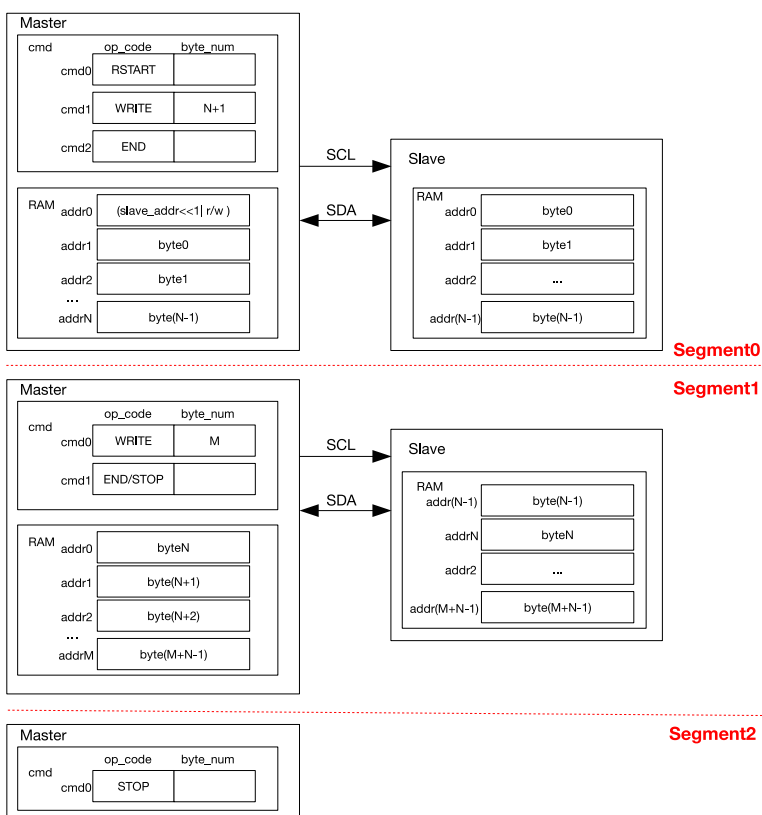


图 15-8. I<sup>2</sup>C Master 分段写 7-bit 寻址的 Slave

RAM 的大小只有 32 字节，对于大量的数据传输当 RAM 乒乓操作也不能满足要求时，建议使用多次命令序列进行分段传输。每次命令序列以 END 命令结尾，这样控制器会执行 END 命令拉低 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 15-8 所示为 I<sup>2</sup>C Master 分成三段或者两段写 Slave。首先配置 I<sup>2</sup>C Master 的命令序列如第一段所示，并且在 Master 的 RAM 中准备好数据，置位 I2C\_TRANS\_START，I<sup>2</sup>C Master 即开始数据传输。在执行到 END 命令后，I<sup>2</sup>C Master 会关闭 SCL 时钟，并将 SCL 线拉低来防止其他设备占用 I<sup>2</sup>C 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 Slave。置位 I2C\_TRANS\_START 后，I<sup>2</sup>C Master 继续发送数据，并在最后发送 STOP 位。当为三段写 Slave 时，I<sup>2</sup>C Master 在第二段发送完数据，并检测到 I<sup>2</sup>C Master 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，I<sup>2</sup>C Master 即产生 STOP 位，从而停止传输。

请注意，在两个分段之间，I<sup>2</sup>C 总线上的其他 Master 设备不会占用总线。只有在发送了 STOP 信号后总线才会被释放。任何情况下，置位 I2C\_FSM\_RST 可复位 I<sup>2</sup>C 控制器，硬件自清 I2C\_FSM\_RST。

在 I<sup>2</sup>C Master 处于空闲状态时，置位 I2C\_SCL\_RST\_SLV\_EN，硬件会发送 I2C\_SCL\_RST\_SLV\_NUM 个 SCL 脉冲，之后硬件会自清 I2C\_SCL\_RST\_SLV\_EN 位。

需要注意的是，总线上其他 Master 或者 Slave 的操作可能与 ESP32-S2 I<sup>2</sup>C 外设有所不同，具体请参考各个 I<sup>2</sup>C 设备的技术规格书。

#### 15.4.5 I<sup>2</sup>C 主机读取从机，7-bit 寻址，单次命令序列

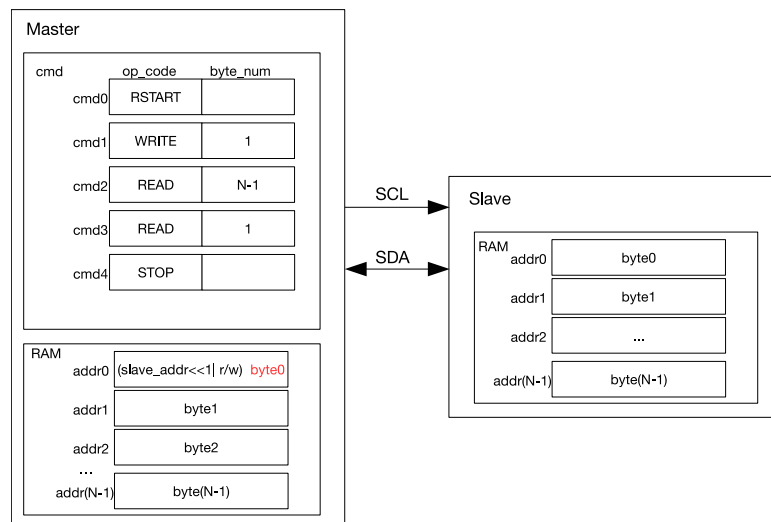


图 15-9. I<sup>2</sup>C Master 读 7-bit 寻址的 Slave

图 15-9 I<sup>2</sup>C Master 从 7-bit 寻址 I<sup>2</sup>C Slave 读取 N 个字节数据的命令寄存器及 RAM 的值。cmd1 为 WRITE 命令，I<sup>2</sup>C Master 会将 I<sup>2</sup>C Slave 的地址发送出去。该命令发送的字节是 7-bit I<sup>2</sup>C Slave 地址以及读写标志位。读写标志位为 1 表示读操作。I<sup>2</sup>C Slave 在地址匹配成功之后即开始发送数据给 I<sup>2</sup>C Master。I<sup>2</sup>C Master 根据 READ 命令中设置的 ack\_value，在接收完一个字节的数据之后回复 ACK。

图 15-9 中 READ 分成两次，I<sup>2</sup>C Master 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，I<sup>2</sup>C Master 从 RAM 的首地

址开始存储，图中红色 byte0 会覆盖第一个字节的内容（Slave 地址 +1-bit 读写位）。

#### 15.4.6 I<sup>2</sup>C 主机读取从机，10-bit 寻址，单次命令序列

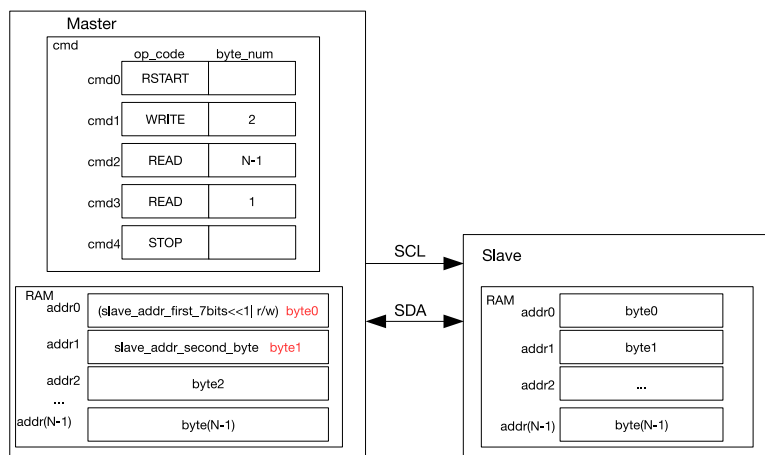


图 15-10. I<sup>2</sup>C Master 读 10-bit 寻址的 Slave

图 15-10 为 I<sup>2</sup>C Master 从 10-bit 寻址的 I<sup>2</sup>C Slave 中读取数据的命令寄存器及 RAM 的值。相比于 7-bit 寻址，I<sup>2</sup>C Master 的第一写命令的字节数为 2 字节，相应 RAM 中存储 2 个字节的 I<sup>2</sup>C Slave 10-bit 地址。相比于 7-bit 寻址，I<sup>2</sup>C Slave 需要置位 `I2C_ADDR_10BIT_EN` 和 `I2C_SLAVE_ADDR[14: 0]`。具体配置方式与 15.4.2 小节的相同。

#### 15.4.7 I<sup>2</sup>C 主机读取从机，7-bit 双寻址，单次命令序列

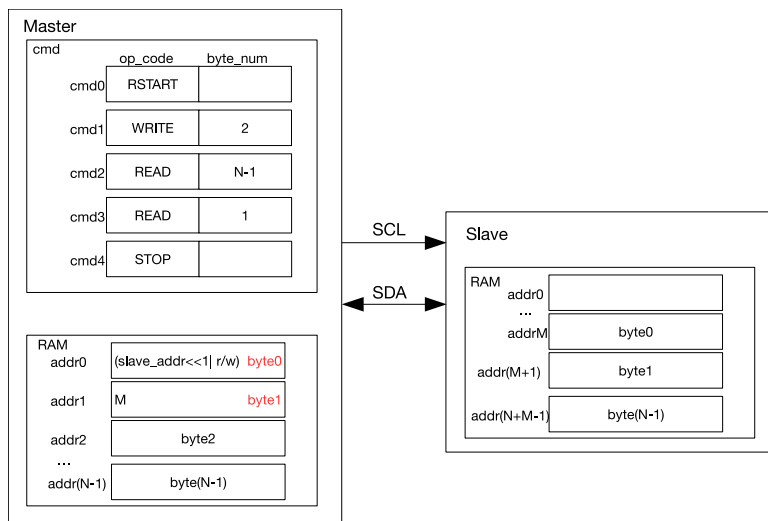


图 15-11. I<sup>2</sup>C Master 从 7-bit 寻址 Slave 的 M 地址读取 N 个数据

图 15-11 为 I<sup>2</sup>C Master 从 I<sup>2</sup>C Slave 中指定地址读取数据的命令寄存器及 RAM 的值。配置流程如下：

1. 在 I<sup>2</sup>C Slave 中置位 `I2C_FIFO_ADDR_CFG_EN` 并在其 RAM 中准备好待发送的数据。
2. 在 I<sup>2</sup>C Master 中准备好 I<sup>2</sup>C Slave 的地址以及其指定的寄存器地址 M。

3. 置位 I<sup>2</sup>C Master 的 `I2C_TRANS_START`，I<sup>2</sup>C Slave 会将从 RAM 中 M 地址开始取 N 个数据发送给 I<sup>2</sup>C Master。

#### 15.4.8 I<sup>2</sup>C 主机读取从机，7-bit 寻址，多次命令序列

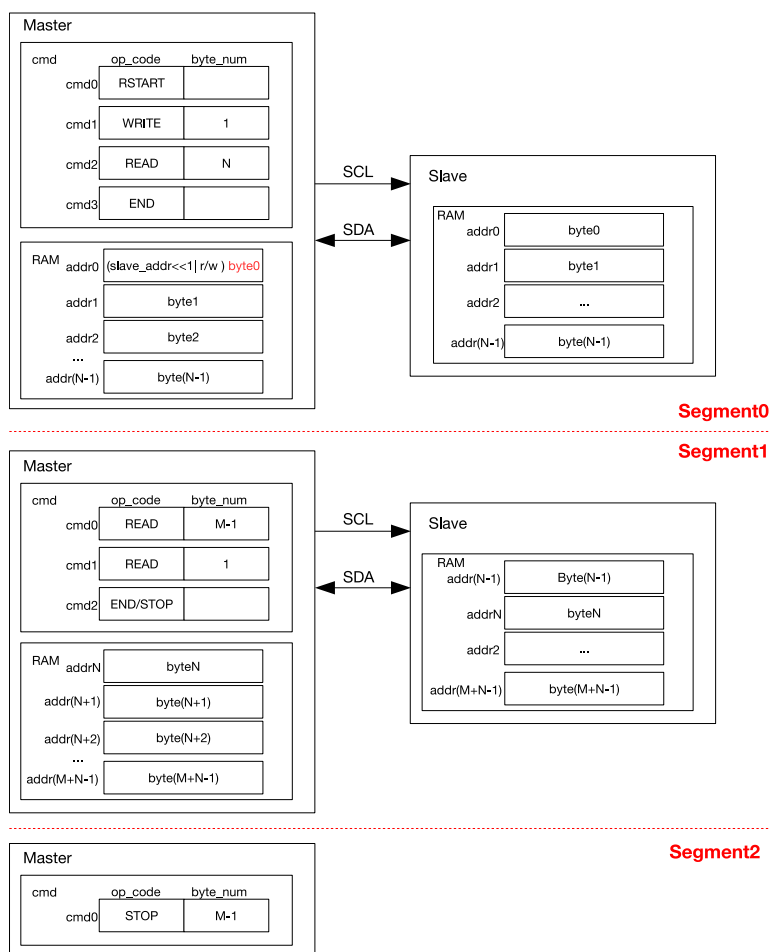


图 15-12. I<sup>2</sup>C Master 分段读 7-bit 寻址的 Slave

图 15-12 为 I<sup>2</sup>C Master 通过 END 命令分三段或者分两段，从 I<sup>2</sup>C Slave 读取 N+M 个数据的配置图。配置的流程如下：

1. 首先配置命令寄存器和 RAM 的内容，如第一段所示。
2. 接着在 Slave 的 RAM 中准备好数据，置位 `I2C_TRANS_START`，I<sup>2</sup>C 即开始工作。当执行到 END 命令时，I<sup>2</sup>C Master 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 I<sup>2</sup>C Slave，置位 `I2C_TRANS_START`，I<sup>2</sup>C Master 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 I<sup>2</sup>C Master 完成第二次数据传输，并检测到 I<sup>2</sup>C Master 的 `I2C_END_DETECT_INT` 中断后，配置 cmd 如第三段所示。置位 `I2C_TRANS_START`，I<sup>2</sup>C Master 发送 STOP 位停止传输。

## 15.5 SCL 延展传输

从机模式下，可以通过 SCL 线拉低，以给软件足够的时间进行处理。置位 [I2C\\_SLAVE\\_SCL\\_STRETCH\\_EN](#) 位使能延展传输，置位 [I2C\\_STRETCH\\_PROTECT\\_NUM](#) 位配置延展传输时长。出现以下三种情况会拉低 SCL 线：

1. 地址命中：从机模式下，I<sup>2</sup>C 控制器的地址与 SDA 线发送的地址相匹配。
2. 写满：从机模式下，I<sup>2</sup>C 控制器的 RX RAM 为满。
3. 读空：从机模式下，I<sup>2</sup>C 控制器的 TX RAM 为空。

SCL 线拉低后，可读取 [I2C\\_STRETCH\\_CAUSE](#) 位获取延展传输的原因。置位 [I2C\\_SLAVE\\_SCL\\_STRETCH\\_CLR](#) 位关闭 SCL 延展传输。

## 15.6 中断

- [I2C\\_SLAVE\\_STRETCH\\_INT](#): 当 I<sup>2</sup>C 从机将 SCL 线拉低时产生此中断。
- [I2C\\_DET\\_START\\_INT](#): 当检测到 I<sup>2</sup>C START 位时，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 [I2C\\_SCL\\_MAIN\\_ST\\_TO](#)[23:0] 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 [I2C\\_SCL\\_ST\\_TO](#)[23:0] 个模块时钟周期时，触发此中断。
- [I2C\\_RXFIFO\\_UDF\\_INT](#): 直接地址访问模式下，当 I<sup>2</sup>C 接收 [I2C\\_NONFIFO\\_RX\\_THRES](#) 个数据，即触发该中断。
- [I2C\\_TXFIFO\\_OVF\\_INT](#): 每当 I<sup>2</sup>C 发送 [I2C\\_NONFIFO\\_TX\\_THRES](#) 个数据，即触发该中断。
- [I2C\\_NACK\\_INT](#): 当 I<sup>2</sup>C 配置为 Master 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 I<sup>2</sup>C 配置为 Slave 时，接收到的 ACK 值为 1 时即触发该中断。
- [I2C\\_TRANS\\_START\\_INT](#): 当 I<sup>2</sup>C 发送一个 START 位时，即触发该中断。
- [I2C\\_TIME\\_OUT\\_INT](#): 在传输过程中，当 I<sup>2</sup>C SCL 保持为高或为低电平的时间超过 [I2C\\_TIME\\_OUT](#) 个模块时钟后，即触发该中断。
- [I2C\\_TRANS\\_COMPLETE\\_INT](#): 当 I<sup>2</sup>C 检测到 STOP 位时，即触发该中断。
- [I2C\\_MST\\_TXFIFO\\_UDF\\_INT](#): 当 I<sup>2</sup>C 主机的 TX FIFO 下溢时，触发此中断。
- [I2C\\_ARBITRATION\\_LOST\\_INT](#): 当 I<sup>2</sup>C Master 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 I<sup>2</sup>C 发送或接收一个字节，即触发该中断。
- [I2C\\_END\\_DETECT\\_INT](#): 当 I<sup>2</sup>C 主机命令的 op\_code 为 END，且检测到 I<sup>2</sup>C END 状态时，触发此中断。
- [I2C\\_RXFIFO\\_OVF\\_INT](#): 当 I<sup>2</sup>C RX FIFO 上溢时，触发此中断。
- [I2C\\_TXFIFO\\_WM\\_INT](#): I<sup>2</sup>C 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: I<sup>2</sup>C 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。

## 15.7 基地址

用户可通过不同的寄存器基地址访问 I<sup>2</sup>C 控制器, 如表 60 所示。更多信息, 请访问章节 1 系统和存储器。

表 60: I<sup>2</sup>C 控制器基地址

	访问总线	基地址
I <sup>2</sup> C0	PeriBUS1	0x3F413000
	PeriBUS2	0x60013000
I <sup>2</sup> C1	PeriBUS1	0x3F427000
	PeriBUS2	0x60027000

## 15.8 寄存器列表

请注意, 下表的地址指相对于 I<sup>2</sup>C 控制器基地址的偏移量 (相对地址)。更多有关 I<sup>2</sup>C 控制器基地址的信息, 请前往 15.7 章节。

名称	描述	地址	访问
<b>时序寄存器</b>			
I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平宽度	0x0000	读/写
I2C_SDA_HOLD_REG	配置 SCL 下降沿后的保持时间	0x0030	读/写
I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	读/写
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平宽度	0x0038	读/写
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	读/写
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟	0x0044	读/写
I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	读/写
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	读/写
I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0098	读/写
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x009C	读/写
<b>配置寄存器</b>			
I2C_CTR_REG	传输设置	0x0004	读/写
I2C_TO_REG	设置接收数据超时控制	0x000C	读/写
I2C_SLAVE_ADDR_REG	本地从机地址设置	0x0010	读/写
I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	读/写
I2C_SCL_SP_CONF_REG	电源配置寄存器	0x00A0	读/写
I2C_SCL_STRETCH_CONF_REG	配置 I2C 从机 SCL 延展传输	0x00A4	不定
<b>状态寄存器</b>			
I2C_SR_REG	描述 I2C 的工作状态	0x0008	只读
I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	不定

名称	描述	地址	访问
I2C_DATA_REG	RX FIFO 读取数据	0x001C	只读
<b>中断寄存器</b>			
I2C_INT_RAW_REG	原始中断状态	0x0020	只读
I2C_INT_CLR_REG	中断清除位	0x0024	只写
I2C_INT_ENA_REG	中断使能位	0x0028	读/写
I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	只读
<b>滤波寄存器</b>			
I2C_SCL_FILTER_CFG_REG	SCL 滤波配置寄存器	0x0050	读/写
I2C_SDA_FILTER_CFG_REG	SDA 滤波配置寄存器	0x0054	读/写
<b>命令寄存器</b>			
I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	读/写
I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	读/写
I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	读/写
I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	读/写
I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	读/写
I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	读/写
I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	读/写
I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	读/写
I2C_COMD8_REG	I2C 命令寄存器 8	0x0078	读/写
I2C_COMD9_REG	I2C 命令寄存器 9	0x007C	读/写
I2C_COMD10_REG	I2C 命令寄存器 10	0x0080	读/写
I2C_COMD11_REG	I2C 命令寄存器 11	0x0084	读/写
I2C_COMD12_REG	I2C 命令寄存器 12	0x0088	读/写
I2C_COMD13_REG	I2C 命令寄存器 13	0x008C	读/写
I2C_COMD14_REG	I2C 命令寄存器 14	0x0090	读/写
I2C_COMD15_REG	I2C 命令寄存器 15	0x0094	读/写
<b>版本寄存器</b>			
I2C_DATE_REG	版本控制寄存器	0x00F8	读/写

## 15.9 寄存器

Register 15.1: I2C\_SCL\_LOW\_PERIOD\_REG (0x0000)

(reserved)														I2C_SCL_LOW_PERIOD																															
31														14														13																	0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0														0x00																	Reset														

**I2C\_SCL\_LOW\_PERIOD** 用于配置 SCL 低电平的保持时间，以 I2C 模块时钟周期数为单位。(读/写)

Register 15.2: I2C\_SDA\_HOLD\_REG (0x0030)

(reserved)																						I2C_SDA_HOLD_TIME																																			
31																																10		9		0																					
0 0																																0x0																						Reset			

**I2C\_SDA\_HOLD\_TIME** 用于配置 SCL 下降沿后的数据保持时间，以 I2C 模块时钟周期数为单位。  
(读/写)

Register 15.3: I2C\_SDA\_SAMPLE\_REG (0x0034)

(reserved)																						I2C_SDA_SAMPLE_TIME																															
31																																10		9		0																	
0 0																																0x0																		Reset			

**I2C\_SDA\_SAMPLE\_TIME** 用于配置采样 SDA 的时间，以 I2C 模块时钟周期数为单位。(读/写)



Register 15.4: I2C\_SCL\_HIGH\_PERIOD\_REG (0x0038)

(reserved)				I2C_SCL_WAIT_HIGH_PERIOD										I2C_SCL_HIGH_PERIOD																
31	28	27											14	13											0					
0	0	0	0	0x00										0x00										Reset						

**I2C\_SCL\_HIGH\_PERIOD** 用于配置 SCL 在主机模式下保持高电平的时间，以 I2C 模块时钟周期数为单位。(读/写)

**I2C\_SCL\_WAIT\_HIGH\_PERIOD** 用于配置 SCL\_FSM 等待 SCL 在主机模式下翻转至高电平的时间，以 I2C 模块时钟周期数为单位。(读/写)

Register 15.5: I2C\_SCL\_START\_HOLD\_REG (0x0040)

(reserved)																	I2C_SCL_START_HOLD_TIME											
31																	10	9	0									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8										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，以 I2C 模块时钟周期数为单位。(读/写)

Register 15.6: I2C\_SCL\_RSTART\_SETUP\_REG (0x0044)

(reserved)																	I2C_SCL_RSTART_SETUP_TIME											
31																	10	9	0									
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8										Reset	

**I2C\_SCL\_RSTART\_SETUP\_TIME** 配置 RESTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间，以 I2C 模块的时钟周期数为单位。(读/写)

Register 15.7: I2C\_SCL\_STOP\_HOLD\_REG (0x0048)

(reserved)																I2C_SCL_STOP_HOLD_TIME																																															
31																14																13																0															
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																0x00																Reset																															

**I2C\_SCL\_STOP\_HOLD\_TIME** 配置 STOP 命令后的延迟，以 I2C 模块时钟周期数为单位。(读/写)

Register 15.8: I2C\_SCL\_STOP\_SETUP\_REG (0x004C)

(reserved)																						I2C_SCL_STOP_SETUP_TIME																																
31																						10										9											0											
0 0																						0x0																						Reset										

**I2C\_SCL\_STOP\_SETUP\_TIME** 配置 SCL 上升沿和 SDA 上升沿的间隔时间，以 I2C 模块时钟周期数为单位。(读/写)

Register 15.9: I2C\_SCL\_ST\_TIME\_OUT\_REG (0x0098)

(reserved)																I2C_SCL_ST_TO																							
31								24								23								0															
0								0								0								0x0100								Reset							

**I2C\_SCL\_ST\_TO** SCL\_FSM 状态不变的阈值。(读/写)

Register 15.10: I2C\_SCL\_MAIN\_ST\_TIME\_OUT\_REG (0x009C)

(reserved)								I2C_SCL_MAIN_ST_TO																					
31								24	23																				0
0	0	0	0	0	0	0	0	0x0100																					Reset

**I2C\_SCL\_MAIN\_ST\_TO** SCL\_MAIN\_FSM 状态不变的阈值。(读/写)

Register 15.11: I2C\_CTR\_REG (0x0004)

(reserved)																								I2C_REF_ALWAYS_ON I2C_FSM_RST I2C_ARBITRATION_EN I2C_CLK_EN I2C_RX_LSB_FIRST I2C_TX_LSB_FIRST I2C_TRANS_START I2C_MS_MODE I2C_SAMPLE_SCL_LEVEL I2C_SDA_FORCE_OUT																				
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	1	0	1	0	0	0	0	0	1	0	1	1													

**I2C\_SDA\_FORCE\_OUT** 0: 直接输出; 1: 漏极开路输出。(读/写)

**I2C\_SCL\_FORCE\_OUT** 0: 直接输出; 1: 漏极开路输出。(读/写)

**I2C\_SAMPLE\_SCL\_LEVEL** 用于选择采样模式。1: SCL 为低电平时采样 SDA 数据。0: SCL 为高电平时采样 SDA 数据。(读/写)

**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。(读/写)

**I2C\_MS\_MODE** 置位此位, 将模块配置为 I2C 主机。清零此位, 将模块配置为 I2C 从机。(读/写)

**I2C\_TRANS\_START** 置位此位, 开始发送 TX FIFO 中的数据。(读/写)

**I2C\_TX\_LSB\_FIRST** 用于控制待发送数据的发送模式。1: 从最低有效位开始发送数据; 0: 从最高有效位开始发送数据。(读/写)

**I2C\_RX\_LSB\_FIRST** 用于控制接收数据的存储模式。1: 从最低有效位开始接收数据; 0: 从最高有效位开始接收数据。(读/写)

**I2C\_CLK\_EN** 保留 (读/写)

**I2C\_ARBITRATION\_EN** I2C 总线仲裁的使能位。(读/写)

**I2C\_FSM\_RST** 用于复位 SCL\_FSM。(读/写)

**I2C\_REF\_ALWAYS\_ON** 用于控制 REF\_TICK。(读/写)

## 324

ESP32-S2 TRM (预发布 V0.2)

### 反馈文档意见

**I2C\_ADDR\_10BIT\_EN** 用于在主机模式下使能从机的 10 位寻址模式。(读/写)

31	30	15	14	0
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00		

Reset

### Register 15.14: I2C\_FIFO\_CONF\_REG (0x0018)

(reserved)					I2C_FIFO_PRT_EN					I2C_NONFIFO_TX_THRES					I2C_NONFIFO_RX_THRES					I2C_TX_FIFO_RST					I2C_RX_FIFO_RST					I2C_FIFO_ADDR_RST					I2C_NONFIFO_EN					I2C_TXFIFO_WM_THRHD					I2C_RXFIFO_WM_THRHD				
31					27		26	25	20					19	14					13	12	11	10	9	5					4	0																		
0					0		0	0	0	1					0x15					0x15					0		0	0		0	0x4					0xb					Reset								

**I2C\_RXFIFO\_WM\_THRHD** non-FIFO 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\_TXFIFO\_WM\_THRHD[4:0] 时，I2C\_TXFIFO\_WM\_INT\_RAW 位有效。  
(读/写)

**I2C\_TXFIFO\_WM\_THRHD** non-FIFO 访问模式下，TX FIFO 的水标阈值。I2C\_FIFO\_PRT\_EN 为 1 且 TX FIFO 计数值小于 I2C\_TXFIFO\_WM\_THRHD[4:0] 时，I2C\_TXFIFO\_WM\_INT\_RAW 位有效。  
(读/写)

**I2C\_NONFIFO\_EN** 置位此位，使能 APB non-FIFO 访问。(读/写)

**I2C\_FIFO\_ADDR\_CFG\_EN** 此位置 1 时，从机接收地址字节的后一个字节为从机 RAM 中的偏移地址。(读/写)

**I2C\_RX\_FIFO\_RST** 置位此位，复位 RX FIFO。(读/写)

**I2C\_TX\_FIFO\_RST** 置位此位，复位 TX FIFO。(读/写)

**I2C\_NONFIFO\_RX\_THRES** I2C 接收的数据超过 I2C\_NONFIFO\_TX\_THRES 字节时，生成 I2C\_RXFIFO\_UDF\_INT 中断，更新接收数据的当前偏移地址。(读/写)

**I2C\_NONFIFO\_TX\_THRES** I2C 发送的数据超过 I2C\_NONFIFO\_TX\_THRES 个字节时，生成 I2C\_TXFIFO\_OVF\_INT 中断，更新发送数据的当前偏移地址。(读/写)

**I2C\_FIFO\_PRT\_EN** non-FIFO 访问模式下 FIFO 指针的控制使能位。该位控制 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的有效位和中断。(读/写)

Register 15.15: I2C\_SCL\_SP\_CONF\_REG (0x00A0)

(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	0x0		0	Reset	

Reset

**I2C\_SCL\_RST\_SLV\_EN** 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[4:0]。(读/写)

**I2C\_SCL\_RST\_SLV\_NUM** 配置主机模式下生成的 SCL 脉冲。I2C\_SCL\_RST\_SLV\_EN 为 1 时有效。(读/写)

**I2C\_SCL\_PD\_EN** 降低 I2C SCL 输出功耗的使能位。1：不工作，降低功耗。0：正常工作。将 I2C\_SCL\_FORCE\_OUT 和 I2C\_SCL\_PD\_EN 置 1 延长 SCL 的低电平时间。(读/写)

**I2C\_SDA\_PD\_EN** 降低 I2C SDA 输出功耗的使能位。1：不工作，降低功耗。0：正常工作。将 I2C\_SDA\_FORCE\_OUT 和 I2C\_SDA\_PD\_EN 置 1 延长 SDA 的低电平时间。(读/写)

Register 15.16: I2C\_SCL\_STRETCH\_CONF\_REG (0x00A4)

(reserved)																				I2C_SLAVE_SCL_STRETCH_CLR I2C_SLAVE_SCL_STRETCH_EN		I2C_STRETCH_PROTECT_NUM			
31																				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																				0	0	0x0		Reset	

Reset

**I2C\_STRETCH\_PROTECT\_NUM** 配置 I2C 从机延长 SCL 低电平时间，以时钟周期为单位。(读/写)

**I2C\_SLAVE\_SCL\_STRETCH\_EN** 从机 SCL 延展传输功能的使能位。1：使能。0：关闭。I2C\_SLAVE\_SCL\_STRETCH\_EN 为 1 时延展时钟，延长 SCL 输出线的低电平时间。延展传输的原因可见 I2C\_STRETCH\_CAUSE。(读/写)

**I2C\_SLAVE\_SCL\_STRETCH\_CLR** 置位此位，清除 I2C 从机的 SCL 延展传输功能。(只写)

Register 15.17: 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_BYTE_TRANS		I2C_SLAVE_ADDRESSED		I2C_BUS_BUSY		I2C_ARB_LOST		I2C_TIME_OUT		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	0x0	0	0x0	0x0			0	0	0x0	0x0				0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**I2C\_RESP\_REC** 主机模式或从机模式下接收的 ACK 电平值。0: ACK, 1: NACK。(只读)

**I2C\_SLAVE\_RW** 从机模式下, 1: 主机读取从机数据; 0: 主机向从机写入数据。(只读)

**I2C\_TIME\_OUT** I2C 控制器接收一位数据的时间超过 I2C\_TIME\_OUT 周期时, 该字段变为 1。(只读)

**I2C\_ARB\_LOST** I2C 控制器不控制 SCL 线时, 该寄存器变为 1。(只读)

**I2C\_BUS\_BUSY** 1: I2C 总线正在传输数据; 0: I2C 总线处于空闲状态。(只读)

**I2C\_SLAVE\_ADDRESSED** 配置成 I2C 从机、且主机发送地址与从机地址匹配时, 该位翻转为高电平。(只读)

**I2C\_BYTE\_TRANS** 传输一个字节后, 该字段变为 1。(只读)

**I2C\_RXFIFO\_CNT** 该字段为需发送数据的字节数。(只读)

**I2C\_STRETCH\_CAUSE** 从机模式下延长 SCL 低电平时间的原因。0: I2C 开始读取数据时延长 SCL 的低电平时间。1: 从机模式下 TX FIFO 为空时延长 SCL 的低电平时间。2: 从机模式下 RX FIFO 为满时延长 SCL 的低电平时间。(只读)

**I2C\_TXFIFO\_CNT** 该字段存储 RAM 接收数据的字节数。(只读)

**I2C\_SCL\_MAIN\_STATE\_LAST** 该字段为 I2C 模块状态机的状态。0: 空闲; 1: 地址转换; 2: ACK 地址; 3: 接收数据; 4: 发送数据; 5: 发送 ACK; 6: 等待 ACK (只读)

**I2C\_SCL\_STATE\_LAST** 该字段为生成 SCL 的状态机状态。0: 空闲状态; 1: 开始; 2: 下降沿; 3: 低电平; 4: 上升沿; 5: 高电平; 6: 停止 (只读)

Register 15.18: I2C\_FIFO\_ST\_REG (0x0014)

(reserved)			I2C_SLAVE_RW_POINT				I2C_TX_UPDATE I2C_RX_UPDATE		I2C_TXFIFO_END_ADDR		I2C_TXFIFO_START_ADDR		I2C_RXFIFO_END_ADDR		I2C_RXFIFO_START_ADDR					
31	30	29	22				21	20	19	15		14	10		9	5		4	0	
0	0	0x0				0	0	0x0		0x0		0x0		0x0		0x0		Reset		

**I2C\_RXFIFO\_START\_ADDR** 最后接收数据的偏移地址，如寄存器 I2C\_NONFIFO\_RX\_THRES 所述。  
(只读)

**I2C\_RXFIFO\_END\_ADDR** 最后接收数据的偏移地址，如寄存器 I2C\_NONFIFO\_RX\_THRES 所述。该值在 I2C\_RX\_REC\_FULL\_INT 中断或 I2C\_TRANS\_COMPLETE\_INT 中断产生时更新。(只读)

**I2C\_TXFIFO\_START\_ADDR** 最先发送数据的偏移地址，如寄存器 I2C\_NONFIFO\_TX\_THRES 所述。  
(只读)

**I2C\_TXFIFO\_END\_ADDR** 最后发送数据的偏移地址，如寄存器 I2C\_NONFIFO\_TX\_THRES 所述。该值在 I2C\_TX\_SEND\_EMPTY\_INT 中断或 I2C\_TRANS\_COMPLETE\_INT 中断产生时更新。(只读)

**I2C\_RX\_UPDATE** 在 I2C\_RX\_UPDATE 上写 0 或 1，更新 I2C\_RXFIFO\_END\_ADDR 和 I2C\_RXFIFO\_START\_ADDR 的值。(只写)

**I2C\_TX\_UPDATE** 在 I2C\_TX\_UPDATE 上写 0 或 1，更新 I2C\_TXFIFO\_END\_ADDR 和 I2C\_TXFIFO\_START\_ADDR 的值。(只写)

**I2C\_SLAVE\_RW\_POINT** 从机模式下接收的数据。(只读)

Register 15.19: I2C\_DATA\_REG (0x001C)

(reserved)																I2C_FIFO_RDATA																	
31																8	7	0															
0 0																0x0																Reset	

**I2C\_FIFO\_RDATA** RX FIFO 读取数据的值。(只读)



Register 15.20: I2C\_INT\_RAW\_REG (0x0020)

(reserved)																	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_OVF_INT_RAW	I2C_NACK_INT_RAW	I2C_TRANS_START_INT_RAW	I2C_TIME_OUT_INT_RAW	I2C_TRANS_COMPLETE_INT_RAW	I2C_ARBITRATION_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																	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\_RAW** I2C\_RXFIFO\_WM\_INT 的原始中断位。(只读)

**I2C\_TXFIFO\_WM\_INT\_RAW** I2C\_TXFIFO\_WM\_INT 的原始中断位。(只读)

**I2C\_RXFIFO\_OVF\_INT\_RAW** I2C\_RXFIFO\_OVF\_INT 的原始中断位。(只读)

**I2C\_END\_DETECT\_INT\_RAW** I2C\_END\_DETECT\_INT 的原始中断位。(只读)

**I2C\_BYTE\_TRANS\_DONE\_INT\_RAW** I2C\_END\_DETECT\_INT 的原始中断位。(只读)

**I2C\_ARBITRATION\_LOST\_INT\_RAW** I2C\_ARBITRATION\_LOST\_INT 的原始中断位。(只读)

**I2C\_MST\_TXFIFO\_UDF\_INT\_RAW** I2C\_TRANS\_COMPLETE\_INT 的原始中断位。(只读)

**I2C\_TRANS\_COMPLETE\_INT\_RAW** I2C\_TRANS\_COMPLETE\_INT 的原始中断位。(只读)

**I2C\_TIME\_OUT\_INT\_RAW** I2C\_TIME\_OUT\_INT 的原始中断位。(只读)

**I2C\_TRANS\_START\_INT\_RAW** I2C\_TRANS\_START\_INT 的原始中断位。(只读)

**I2C\_NACK\_INT\_RAW** I2C\_SLAVE\_STRETCH\_INT 的原始中断位。(只读)

**I2C\_TXFIFO\_OVF\_INT\_RAW** I2C\_TXFIFO\_OVF\_INT 的原始中断位。(只读)

**I2C\_RXFIFO\_UDF\_INT\_RAW** I2C\_RXFIFO\_UDF\_INT 的原始中断位。(只读)

**I2C\_SCL\_ST\_TO\_INT\_RAW** I2C\_SCL\_ST\_TO\_INT 的原始中断位。(只读)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_RAW** I2C\_SCL\_MAIN\_ST\_TO\_INT 的原始中断位。(只读)

**I2C\_DET\_START\_INT\_RAW** I2C\_DET\_START\_INT 的原始中断位。(只读)

**I2C\_SLAVE\_STRETCH\_INT\_RAW** I2C\_SLAVE\_STRETCH\_INT 的原始中断位。(只读)

Register 15.21: I2C\_INT\_CLR\_REG (0x0024)

(reserved)																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_TXFIFO_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_ARBTRATION_LOST_INT_CLR I2C_BYTE_TRANS_DONE_INT_CLR I2C_END_DETECT_INT_CLR I2C_RXFIFO_OVF_INT_CLR I2C_TXFIFO_WM_INT_CLR I2C_RXFIFO_WM_INT_CLR																				
31																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

**I2C\_RXFIFO\_WM\_INT\_CLR** 置位此位，清除 I2C\_RXFIFO\_WM\_INT 中断。(只写)

**I2C\_TXFIFO\_WM\_INT\_CLR** 置位此位，清除 I2C\_TXFIFO\_WM\_INT 中断。(只写)

**I2C\_RXFIFO\_OVF\_INT\_CLR** 置位此位，清除 I2C\_RXFIFO\_OVF\_INT 中断。(只写)

**I2C\_END\_DETECT\_INT\_CLR** 置位此位，清除 I2C\_END\_DETECT\_INT 中断。(只写)

**I2C\_BYTE\_TRANS\_DONE\_INT\_CLR** 置位此位，清除 I2C\_END\_DETECT\_INT 中断。(只写)

**I2C\_ARBTRATION\_LOST\_INT\_CLR** 置位此位，清除 I2C\_ARBTRATION\_LOST\_INT 中断。(只写)

**I2C\_MST\_TXFIFO\_UDF\_INT\_CLR** 置位此位，清除 I2C\_TRANS\_COMPLETE\_INT 中断。(只写)

**I2C\_TRANS\_COMPLETE\_INT\_CLR** 置位此位，清除 I2C\_TRANS\_COMPLETE\_INT 中断。(只写)

**I2C\_TIME\_OUT\_INT\_CLR** 置位此位，清除 I2C\_TIME\_OUT\_INT 中断。(只写)

**I2C\_TRANS\_START\_INT\_CLR** 置位此位，清除 I2C\_TRANS\_START\_INT 中断。(只写)

**I2C\_NACK\_INT\_CLR** 置位此位，清除 I2C\_SLAVE\_STRETCH\_INT 中断。(只写)

**I2C\_TXFIFO\_OVF\_INT\_CLR** 置位此位，清除 I2C\_TXFIFO\_OVF\_INT 中断。(只写)

**I2C\_RXFIFO\_UDF\_INT\_CLR** 置位此位，清除 I2C\_RXFIFO\_UDF\_INT 中断。(只写)

**I2C\_SCL\_ST\_TO\_INT\_CLR** 置位此位，清除 I2C\_SCL\_ST\_TO\_INT 中断。(只写)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_CLR** 置位此位，清除 I2C\_SCL\_MAIN\_ST\_TO\_INT 中断。(只写)

**I2C\_DET\_START\_INT\_CLR** 置位此位，清除 I2C\_DET\_START\_INT 中断。(只写)

**I2C\_SLAVE\_STRETCH\_INT\_CLR** 置位此位，清除 I2C\_SLAVE\_STRETCH\_INT 中断。(只写)

Register 15.22: I2C\_INT\_ENA\_REG (0x0028)

(reserved)																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_OVF_INT_ENA I2C_NACK_INT_ENA I2C_TRANS_START_INT_ENA I2C_TIME_OUT_INT_ENA I2C_TRANS_COMPLETE_INT_ENA I2C_ARBTRATION_LOST_INT_ENA I2C_BYTE_TRANS_DONE_INT_ENA I2C_END_DETECT_INT_ENA I2C_RXFIFO_WM_INT_ENA I2C_TXFIFO_WM_INT_ENA																			
31																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	Reset

**I2C\_RXFIFO\_WM\_INT\_ENA** I2C\_RXFIFO\_WM\_INT 的原始中断位。(读/写)

**I2C\_TXFIFO\_WM\_INT\_ENA** I2C\_TXFIFO\_WM\_INT 的原始中断位。(读/写)

**I2C\_RXFIFO\_OVF\_INT\_ENA** I2C\_RXFIFO\_OVF\_INT 的原始中断位。(读/写)

**I2C\_END\_DETECT\_INT\_ENA** I2C\_END\_DETECT\_INT 的原始中断位。(读/写)

**I2C\_BYTE\_TRANS\_DONE\_INT\_ENA** I2C\_END\_DETECT\_INT 的原始中断位。(读/写)

**I2C\_ARBITRATION\_LOST\_INT\_ENA** I2C\_ARBITRATION\_LOST\_INT 的原始中断位。(读/写)

**I2C\_MST\_TXFIFO\_UDF\_INT\_ENA** I2C\_TRANS\_COMPLETE\_INT 的原始中断位。(读/写)

**I2C\_TRANS\_COMPLETE\_INT\_ENA** I2C\_TRANS\_COMPLETE\_INT 的原始中断位。(读/写)

**I2C\_TIME\_OUT\_INT\_ENA** I2C\_TIME\_OUT\_INT 的原始中断位。(读/写)

**I2C\_TRANS\_START\_INT\_ENA** I2C\_TRANS\_START\_INT 的原始中断位。(读/写)

**I2C\_NACK\_INT\_ENA** I2C\_SLAVE\_STRETCH\_INT 的原始中断位。(读/写)

**I2C\_TXFIFO\_OVF\_INT\_ENA** I2C\_TXFIFO\_OVF\_INT 的原始中断位。(读/写)

**I2C\_RXFIFO\_UDF\_INT\_ENA** I2C\_RXFIFO\_UDF\_INT 的原始中断位。(读/写)

**I2C\_SCL\_ST\_TO\_INT\_ENA** I2C\_SCL\_ST\_TO\_INT 的原始中断位。(读/写)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_ENA** I2C\_SCL\_MAIN\_ST\_TO\_INT 的原始中断位。(读/写)

**I2C\_DET\_START\_INT\_ENA** I2C\_DET\_START\_INT 的原始中断位。(读/写)

**I2C\_SLAVE\_STRETCH\_INT\_ENA** I2C\_SLAVE\_STRETCH\_INT 的原始中断位。(读/写)

Register 15.23: I2C\_INT\_STATUS\_REG (0x002C)

(reserved)																I2C_SLAVE_STRETCH_INT_ST	I2C_DET_START_INT_ST	I2C_SCL_MAIN_ST_TO_INT_ST	I2C_SCL_ST_TO_INT_ST	I2C_RXFIFO_UDF_INT_ST	I2C_TXFIFO_OVF_INT_ST	I2C_NACK_INT_ST	I2C_TRANS_START_INT_ST	I2C_TIME_OUT_INT_ST	I2C_TRANS_COMPLETE_INT_ST	I2C_MST_TXFIFO_UDF_INT_ST	I2C_ARBTRATION_LOST_INT_ST	I2C_BYTE_TRANS_DONE_INT_ST	I2C_END_DETECT_INT_ST	I2C_RXFIFO_OVF_INT_ST	I2C_TXFIFO_WM_INT_ST	I2C_RXFIFO_WM_INT_ST		
31																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

**I2C\_RXFIFO\_WM\_INT\_ST** I2C\_RXFIFO\_WM\_INT 的屏蔽中断状态位。(只读)

**I2C\_TXFIFO\_WM\_INT\_ST** I2C\_TXFIFO\_WM\_INT 的屏蔽中断状态位。(只读)

**I2C\_RXFIFO\_OVF\_INT\_ST** I2C\_RXFIFO\_OVF\_INT 的屏蔽中断状态位。(只读)

**I2C\_END\_DETECT\_INT\_ST** I2C\_END\_DETECT\_INT 的屏蔽中断状态位。(只读)

**I2C\_BYTE\_TRANS\_DONE\_INT\_ST** I2C\_END\_DETECT\_INT 的屏蔽中断状态位。(只读)

**I2C\_ARBTRATION\_LOST\_INT\_ST** I2C\_ARBTRATION\_LOST\_INT 的屏蔽中断状态位。(只读)

**I2C\_MST\_TXFIFO\_UDF\_INT\_ST** I2C\_TRANS\_COMPLETE\_INT 的屏蔽中断状态位。(只读)

**I2C\_TRANS\_COMPLETE\_INT\_ST** I2C\_TRANS\_COMPLETE\_INT 的屏蔽中断状态位。(只读)

**I2C\_TIME\_OUT\_INT\_ST** I2C\_TIME\_OUT\_INT 的屏蔽中断状态位。(只读)

**I2C\_TRANS\_START\_INT\_ST** I2C\_TRANS\_START\_INT 的屏蔽中断状态位。(只读)

**I2C\_NACK\_INT\_ST** I2C\_SLAVE\_STRETCH\_INT 的屏蔽中断状态位。(只读)

**I2C\_TXFIFO\_OVF\_INT\_ST** I2C\_TXFIFO\_OVF\_INT 的屏蔽中断状态位。(只读)

**I2C\_RXFIFO\_UDF\_INT\_ST** I2C\_RXFIFO\_UDF\_INT 的屏蔽中断状态位。(只读)

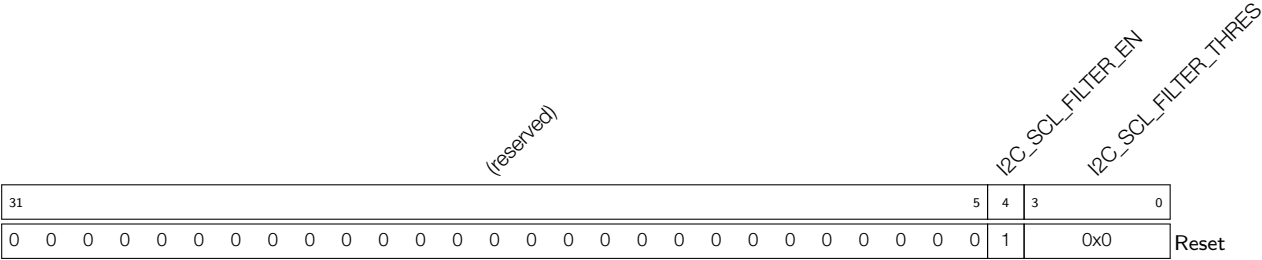
**I2C\_SCL\_ST\_TO\_INT\_ST** I2C\_SCL\_ST\_TO\_INT 的屏蔽中断状态位。(只读)

**I2C\_SCL\_MAIN\_ST\_TO\_INT\_ST** I2C\_SCL\_MAIN\_ST\_TO\_INT 的屏蔽中断状态位。(只读)

**I2C\_DET\_START\_INT\_ST** I2C\_DET\_START\_INT 的屏蔽中断状态位。(只读)

**I2C\_SLAVE\_STRETCH\_INT\_ST** I2C\_SLAVE\_STRETCH\_INT 的屏蔽中断状态位。(只读)

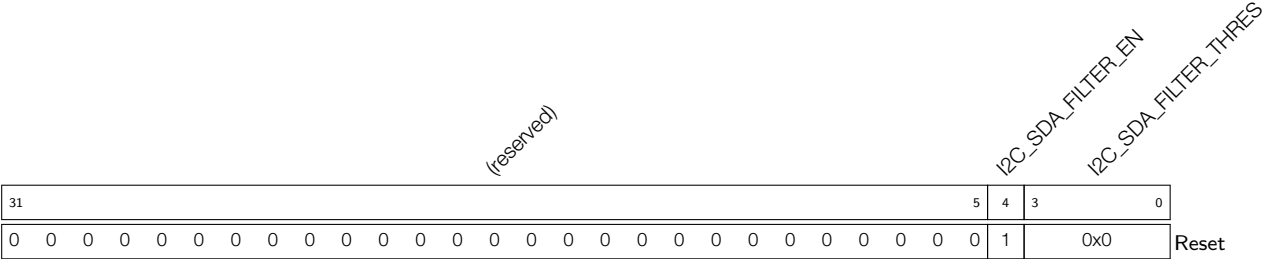
Register 15.24: I2C\_SCL\_FILTER\_CFG\_REG (0x0050)



**I2C\_SCL\_FILTER\_THRES** SCL 输入信号的脉冲宽度小于该寄存器的值时，I2C 控制器忽略此脉冲。  
该寄存器的值以 I2C 模块时钟周期数为单位。(读/写)

**I2C\_SCL\_FILTER\_EN** SCL 的滤波使能位。(读/写)

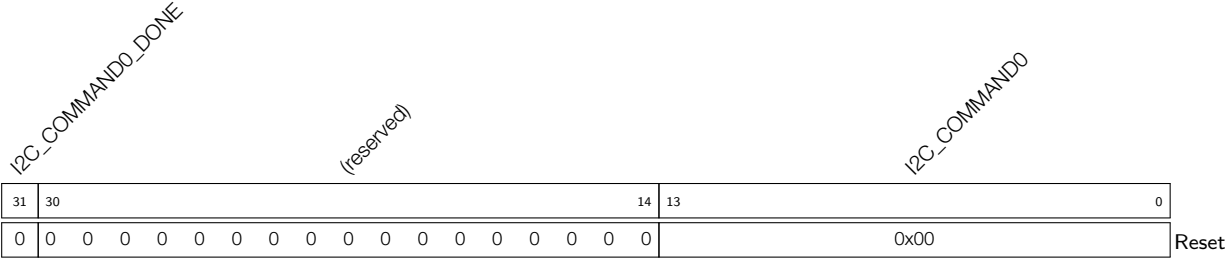
Register 15.25: I2C\_SDA\_FILTER\_CFG\_REG (0x0054)



**I2C\_SDA\_FILTER\_THRES** SDA 输入信号的脉冲宽度小于该寄存器的值时，I2C 控制器忽略此脉冲。  
该寄存器的值以 I2C 模块时钟周期数为单位。(读/写)

**I2C\_SDA\_FILTER\_EN** SDA 的滤波使能位。(读/写)

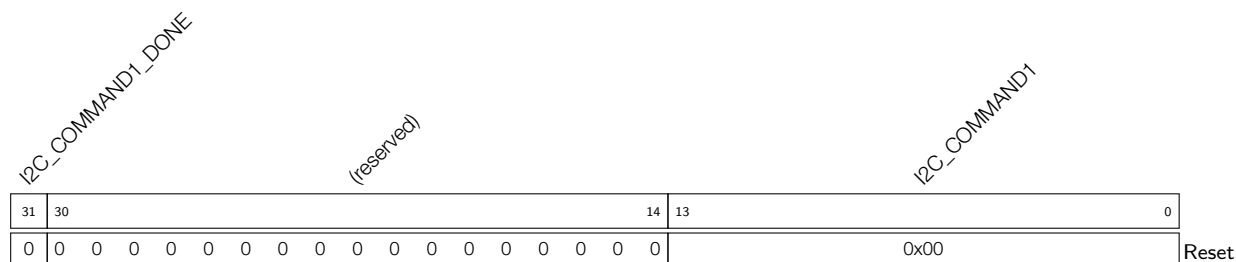
Register 15.26: I2C\_CMD0\_REG (0x0058)



**I2C\_COMMAND0** 命令 0 的内容。该命令包括三个部分：op\_code 为命令，0: START; 1: WRITE; 2: READ; 3: STOP; 4: END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND0\_DONE** 在 I2C 主机模式下完成命令 0 时，该位翻转为高电平。(读/写)

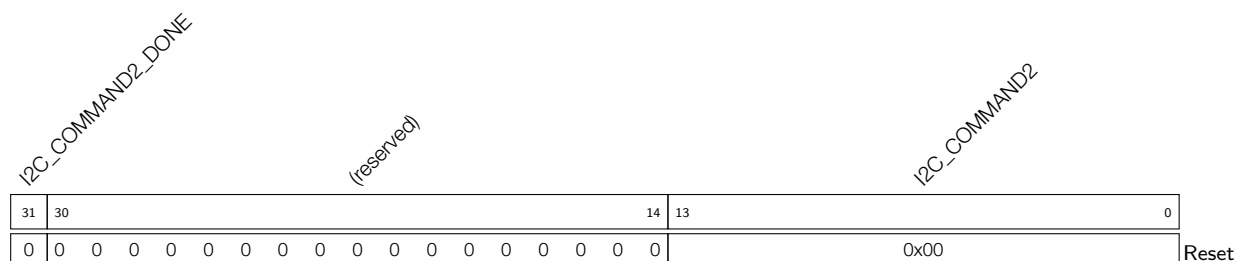
### Register 15.27: I2C\_COMMD1\_REG (0x005C)



**I2C\_COMMAND1** 命令 1 的内容。该命令包括三个部分：op\_code 为命令，0: START; 1: WRITE; 2: READ; 3: STOP; 4: END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND1\_DONE** 在 I2C 主机模式下完成命令 1 时，该位翻转为高电平。(读/写)

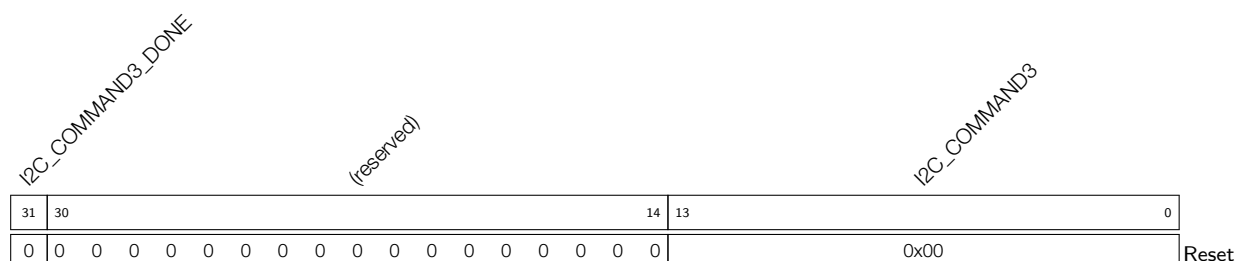
### Register 15.28: I2C\_CMD2\_REG (0x0060)



**I2C\_COMMAND2** 命令 2 的内容。该命令包括三个部分：op\_code 为命令，0：START；1：WRITE；2：READ；3：STOP；4：END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C COMMAND2 DONE** 在 I2C 主机模式下完成命令 2 时，该位翻转为高电平。(读/写)

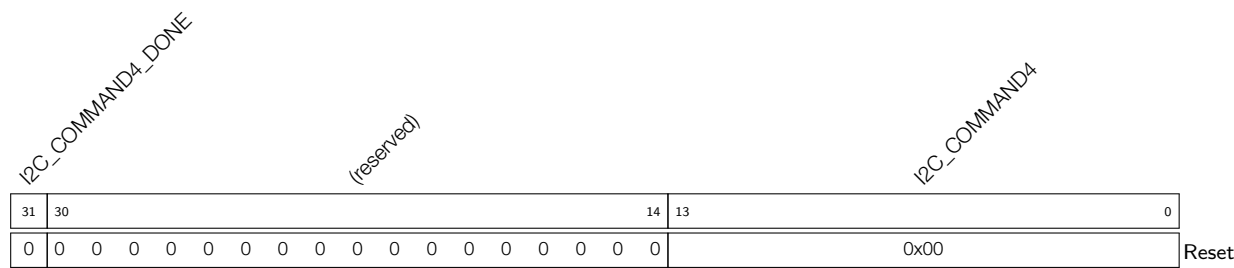
### Register 15.29: I2C COMD3 REG (0x0064)



**I2C\_COMMAND3** 命令 3 的内容。该命令包括三个部分：op\_code 为命令，0：START；1：WRITE；2：READ；3：STOP；4：END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C COMMAND3 DONE** 在 I2C 主机模式下完成命令 3 时，该位翻转为高电平。(读/写)

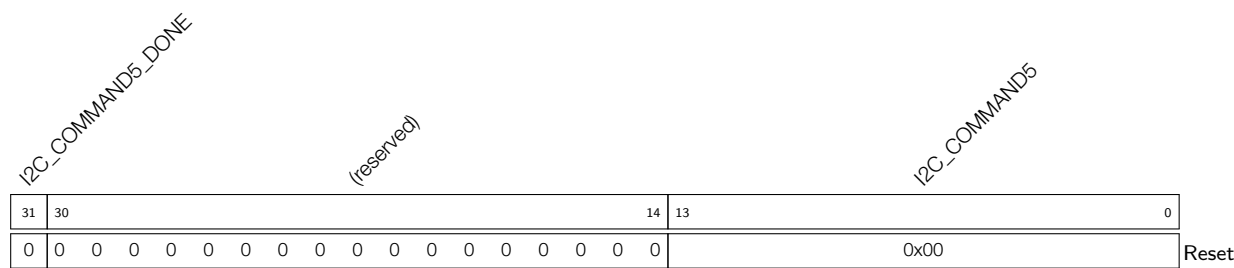
Register 15.30: I2C\_COMD4\_REG (0x0068)



**I2C\_COMMAND4** 命令 4 的内容。该命令包括三个部分：op\_code 为命令，0： START；1： WRITE；2： READ；3： STOP；4： END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND4\_DONE** 在 I2C 主机模式下完成命令 4 时，该位翻转为高电平。(读/写)

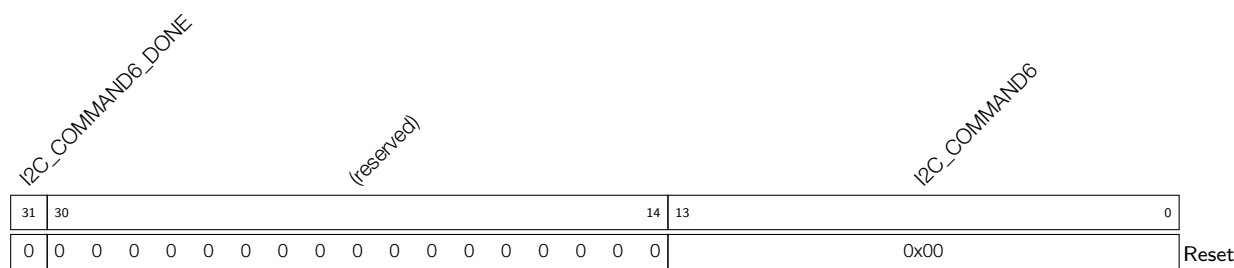
Register 15.31: I2C\_COMD5\_REG (0x006C)



**I2C\_COMMAND5** 命令 5 的内容。该命令包括三个部分：op\_code 为命令，0： START；1： WRITE；2： READ；3： STOP；4： END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND5\_DONE** 在 I2C 主机模式下完成命令 5 时，该位翻转为高电平。(读/写)

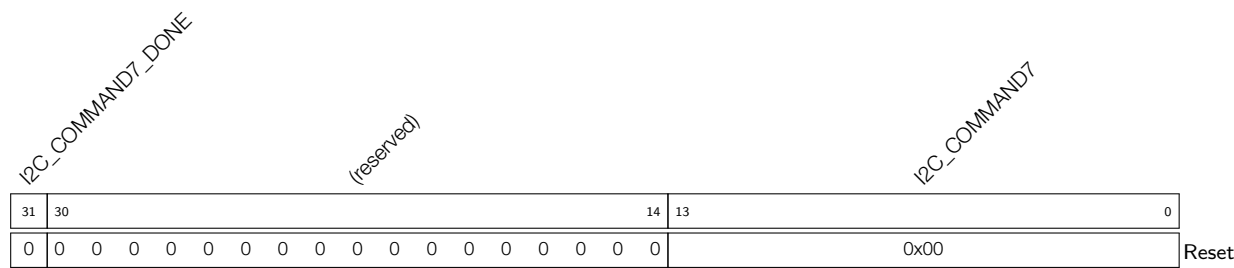
Register 15.32: I2C\_COMD6\_REG (0x0070)



**I2C\_COMMAND6** 命令 6 的内容。该命令包括三个部分：op\_code 为命令，0： START；1： WRITE；2： READ；3： STOP；4： END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND6\_DONE** 在 I2C 主机模式下完成命令 6 时，该位翻转为高电平。(读/写)

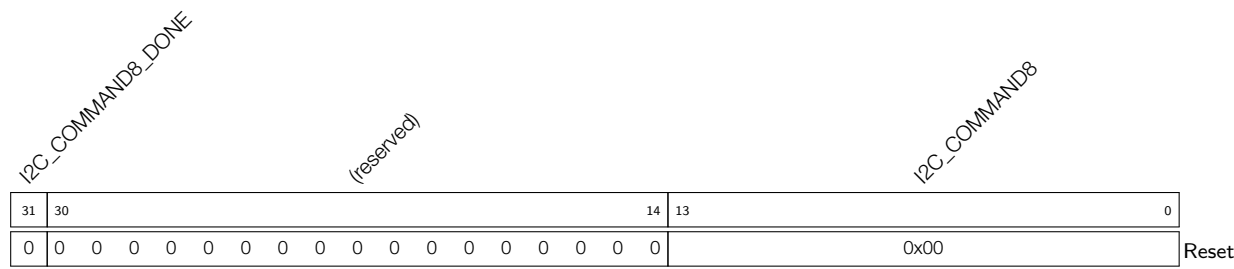
Register 15.33: I2C\_COMD7\_REG (0x0074)



**I2C\_COMMAND7** 命令 7 的内容。该命令包括三个部分：op\_code 为命令，0： START；1： WRITE；2： READ；3： STOP；4： END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND7\_DONE** 在 I2C 主机模式下完成命令 7 时，该位翻转为高电平。(读/写)

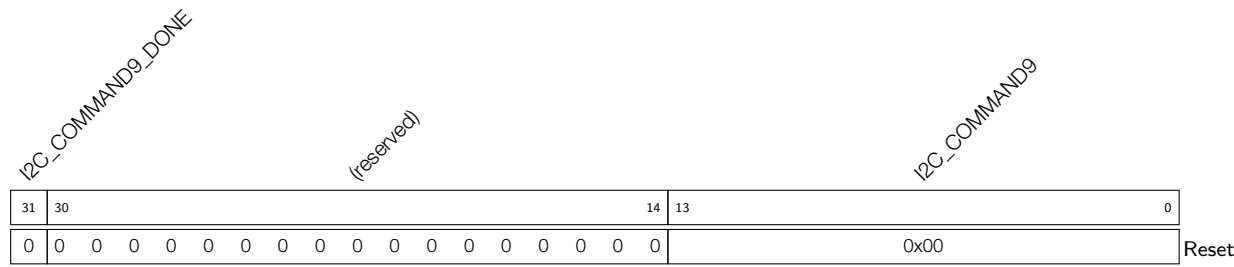
Register 15.34: I2C\_COMD8\_REG (0x0078)



**I2C\_COMMAND8** 命令 8 的内容。该命令包括三个部分：op\_code 为命令，0： START；1： WRITE；2： READ；3： STOP；4： END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND8\_DONE** 在 I2C 主机模式下完成命令 8 时，该位翻转为高电平。(读/写)

Register 15.35: I2C\_COMD9\_REG (0x007C)



**I2C\_COMMAND9** 命令 9 的内容。该命令包括三个部分：op\_code 为命令，0： START；1： WRITE；2： READ；3： STOP；4： END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND9\_DONE** 在 I2C 主机模式下完成命令 9 时，该位翻转为高电平。(读/写)



### Register 15.36: I2C\_COMMD10\_REG (0x0080)

Diagram illustrating the structure of the I2C\_COMMAND10 register:

- I2C\_COMMAND10\_DONE**: Bits 31-30 (2 bits)
- (reserved)**: Bits 30-14 (17 bits)
- I2C\_COMMAND10**: Bits 13-0 (14 bits)

The register value is shown as 0x00.

**I2C\_COMMAND10** 命令 10 的内容。该命令包括三个部分: op\_code 为命令, 0: START; 1: WRITE; 2: READ; 3: STOP; 4: END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND10\_DONE** 在 I2C 主机模式下完成命令 10 时，该位翻转为高电平。(读/写)

### Register 15.37: I2C\_COMMD11\_REG (0x0084)

I2C_COMMAND11_DONE																(reserved)																I2C_COMMAND11															
31																14	13																0														
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x00																Reset														

**I2C\_COMMAND11** 命令 11 的内容。该命令包括三个部分: op\_code 为命令, 0: START; 1: WRITE; 2: READ; 3: STOP; 4: END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND11\_DONE** 在 I2C 主机模式下完成命令 11 时，该位翻转为高电平。(读/写)

### Register 15.38: I2C\_COMMD12\_REG (0x0088)

Diagram illustrating the structure of the I2C\_COMMAND12 register:

- Bit 31: I2C\_COMMAND12\_DONE
- Bits 30-14: (reserved)
- Bits 13-0: I2C\_COMMAND12 (Value: 0x00)

**I2C\_COMMAND12** 命令 12 的内容。该命令包括三个部分: op\_code 为命令, 0: START; 1: WRITE; 2: READ; 3: STOP; 4: END。byte\_num 表示需发送或接收的字节数。ack\_check\_en、ack\_exp 和 ack 用于控制 ACK 位。参阅 I2C cmd 结构获取更多信息。(读/写)

**I2C\_COMMAND12\_DONE** 在 I2C 主机模式下完成命令 12 时，该位翻转为高电平。(读/写)

## 338

ESP32-S2 TRM (预发布 V0.2)

## 反馈文档意见

**I2C\_COMMAND15\_DONE** 在 I2C 主机模式下完成命令 15 时，该位翻转为高电平。(读/写)

Register 15.42: I2C\_DATE\_REG (0x00F8)

I2C_DATE	
31	0
0x19052000	
Reset	

I2C\_DATE 版本控制寄存器。(读/写)

## 16. AES 加速器

### 16.1 概述

ESP32-S2 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-S2 支持以下特性：

- Typical AES 工作模式
  - AES-128/AES-192/AES-256 加解密运算
  - 4 种密钥字节序和 4 种文本字节序
- DMA-AES 工作模式
  - 块模式
    - \* ECB (Electronic Codebook)
    - \* CBC (Cipher Block Chaining)
    - \* OFB (Output Feedback)
    - \* CTR (Counter)
    - \* CFB8 (8-bit Cipher Feedback)
    - \* CFB128 (128-bit Cipher Feedback)
  - GCM (Galois/Counter Mode)
  - 中断发生

### 16.3 工作模式简介

ESP32-S2 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 [NIST FIPS 197](#)，能够实现 AES-128、AES-192、AES-256 加密与解密运算。这种情况下，明文/密文的读/写操作统一通过 CPU 访问完成。
- DMA-AES 工作模式：支持 [NIST SP 800-38A](#) 标准中的 ECB/CBC/OFB/CTR/CFB8/CFB128 等块加密模式运算以及 [NIST SP 800-38D](#) 标准中的 GCM 运算。在这种情况下，明文/密文的传输通过硬件上的 crypto DMA 完成，计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 62。

表 62: 工作模式

<a href="#">AES_DMA_ENABLE_REG</a>	工作模式
0	Typical AES
1	DMA-AES

有关 Typical AES 和 DMA-AES 两种工作模式的具体介绍，请见下方 16.4 章节和 16.5 章节。

**注意：**  
ESP32-S2 的 [数字签名](#)和 [片外存储器加密与解密](#)模块也会调用 AES 加速器。此时，用户无法正常访问 AES 加速器。

16.4 Typical AES 工作模式

在 Typical AES 工作模式下，AES 加速器支持 AES-128/AES-192/AES-256 加解密共 6 种运算类型。用户可通过配置 [AES\\_MODE\\_REG](#) 寄存器选择具体运算类型，具体可参考表 63。

表 63: 运算模式选择

<a href="#">AES_MODE_REG</a> [2:0]	运算模式
0	AES-128 加密
1	AES-192 加密
2	AES-256 加密
4	AES-128 解密
5	AES-192 解密
6	AES-256 解密

AES 加速器的状态值可查看寄存器 [AES\\_STATE\\_REG](#)，具体见表 64 所示：

表 64: 状态返回值

返回值	描述	状态说明
0	IDLE	加速器空闲或计算完成
1	WORK	加速器忙于计算

在 Typical AES 工作模式下，AES 加速器每加密一个信息块需要 11 ~ 15 个时钟周期，每解密一个信息块需要 21 或 22 个时钟周期。

16.4.1 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-192 加解密运算，则 192 位密钥在寄存器 [AES\\_KEY\\_0\\_REG ~ AES\\_KEY\\_5\\_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](#) 用于存放明文或密文，各由 4 个 32 位寄存器组成。

- 如果为 AES-128/192/256 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-128/192/256 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](#)。

## 16.4.2 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 block 进行加解密。其中，输入文本的字节序由寄存器 [AES\\_ENDIAN\\_REG](#) 的 Bit 2 和 Bit 3 控制，输出文本字节序由 Bit 4 和 Bit 5 控制。具体来说，Bit 2 和 Bit 4 控制每个 word 内 4 个 byte 的顺序，Bit 3 和 Bit 5 控制每个 block 中 4 个 word 的顺序。

不难看出，通过配置 [AES\\_ENDIAN\\_REG](#) 寄存器，AES 加速器可允许四种文本字节序。表 65 指定了在四种不同字节序下，寄存器 [AES\\_TEXT\\_IN\\_m\\_REG](#) 和 [AES\\_TEXT\\_OUT\\_m\\_REG](#) 中的每个 word 所存放的明文或密文将如何构成 State。

表 65: Typical AES 文本字节序

Word Endian 控制位	Byte Endian 控制位	明文/密文 <sup>2</sup>							
0	0	State <sup>1</sup>		c					
				0	1	2	3		
		r	0	AES_TEXT_x_3_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_0_REG[31:24]		
			1	AES_TEXT_x_3_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_0_REG[23:16]		
			2	AES_TEXT_x_3_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_0_REG[15:8]		
			3	AES_TEXT_x_3_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_0_REG[7:0]		
0	1	State		c					
				0	1	2	3		
		r	0	AES_TEXT_x_3_REG[7:0]	AES_TEXT_x_2_REG[7:0]	AES_TEXT_x_1_REG[7:0]	AES_TEXT_x_0_REG[7:0]		
			1	AES_TEXT_x_3_REG[15:8]	AES_TEXT_x_2_REG[15:8]	AES_TEXT_x_1_REG[15:8]	AES_TEXT_x_0_REG[15:8]		
			2	AES_TEXT_x_3_REG[23:16]	AES_TEXT_x_2_REG[23:16]	AES_TEXT_x_1_REG[23:16]	AES_TEXT_x_0_REG[23:16]		
			3	AES_TEXT_x_3_REG[31:24]	AES_TEXT_x_2_REG[31:24]	AES_TEXT_x_1_REG[31:24]	AES_TEXT_x_0_REG[31:24]		
1	0	State		c					
				0	1	2	3		
		r	0	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	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]		
			2	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]		
			3	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	1	State		c					
				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]		

#### 说明:

1. 有关“State”的详细定义，请参考 [NIST FIPS 197](#) 中“3.4 The State”章节。
2. 其中，
  - $x = \text{IN}$  时，[AES\\_TEXT\\_IN\\_m\\_REG](#) 的 Word Endian 和 Byte Endian 控制位分别为 [AES\\_ENDIAN\\_REG](#) 的 Bit 2 和 Bit 3；
  - $x = \text{OUT}$  时，[AES\\_TEXT\\_OUT\\_m\\_REG](#) 的 Word Endian 和 Byte Endian 控制位分别为 [AES\\_ENDIAN\\_REG](#) 的 Bit 4 和 Bit 5。

### 密钥字节序

在 Typical AES 工作模式下，AES 加速器的密钥字节序由寄存器 [AES\\_ENDIAN\\_REG](#) 的 Bit 0 和 Bit 1 控制，共可组成四种密钥字节序。

表 66、表 67、表 68 描述了在四种密钥字节序下，寄存器 [AES\\_KEY\\_n\\_REG](#) 中的每个 word 将如何构成 “the first Nk words of the expanded key”。

表 66: AES-128 密钥字节序

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit <sup>2</sup>	w[0]	w[1]	w[2]	w[3] <sup>1</sup>
0	0	[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]
		[23:16]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
		[15:8]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
		[7:0]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
1	1	[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]

说明:

1. 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。
2. Bit 列代表 w[0] ~ w[3] 每个 word 中的各个字节。

表 67: AES-192 密钥字节序

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit <sup>2</sup>	w[0]	w[1]	w[2]	w[3]	w[4]	w[5] <sup>1</sup>
0	0	[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
		[23:16]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[15:8]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[7:0]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
1	0	[31:24]	AES_KEY_0_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]
		[23:16]	AES_KEY_0_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]
		[15:8]	AES_KEY_0_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]
		[7:0]	AES_KEY_0_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]
1	1	[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]	AES_KEY_4_REG[7:0]	AES_KEY_5_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]	AES_KEY_4_REG[15:8]	AES_KEY_5_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]	AES_KEY_4_REG[23:16]	AES_KEY_5_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]	AES_KEY_4_REG[31:24]	AES_KEY_5_REG[31:24]

说明:

1. w[0] ~ w[5] 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。
2. Bit 列代表 w[0] ~ w[5] 每个 word 中的各个字节。



表 68: AES-256 密钥字节序

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit <sup>2</sup>	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7] <sup>1</sup>
0	0	[31:24]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
0	1	[31:24]	AES_KEY_7_REG[31:24]	AES_KEY_6_REG[31:24]	AES_KEY_5_REG[31:24]	AES_KEY_4_REG[31:24]	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
		[23:16]	AES_KEY_7_REG[23:16]	AES_KEY_6_REG[23:16]	AES_KEY_5_REG[23:16]	AES_KEY_4_REG[23:16]	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
		[15:8]	AES_KEY_7_REG[15:8]	AES_KEY_6_REG[15:8]	AES_KEY_5_REG[15:8]	AES_KEY_4_REG[15:8]	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
		[7:0]	AES_KEY_7_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
1	0	[31:24]	AES_KEY_0_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]
		[23:16]	AES_KEY_0_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]
		[15:8]	AES_KEY_0_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]
		[7:0]	AES_KEY_0_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]
1	1	[31:24]	AES_KEY_0_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]
		[23:16]	AES_KEY_0_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]
		[15:8]	AES_KEY_0_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]
		[7:0]	AES_KEY_0_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]

说明:

1. 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。
2. Bit 列代表 w[0] ~ w[7] 每个 word 中的各个字节。

### 16.4.3 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`、`AES_ENDIAN_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_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`、`AES_ENDIAN_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`、`AES_KEY_n_REG`、`AES_ENDIAN_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。

## 16.5 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 等 6 种块模式运算和 GCM 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 69。

表 69: 运算模式选择

<a href="#">AES_BLOCK_MODE_REG</a> [2:0]	运算模式
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	GCM (Galois/Counter Mode)

AES 加速器的状态值可查看寄存器 [AES\\_STATE\\_REG](#)，具体见表 70 所示：

表 70: 状态返回值

返回值	描述	状态说明
0	IDLE	加速器空闲
1	WORK	加速器忙于计算
2	DONE	加速器计算完成

AES 加速器在 DMA-AES 工作模式下允许中断发生，软件清零。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 加速器在完成计算时，中断发生。

### 16.5.1 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 (in\_stream) 来自 DMA，结果数据 (out\_stream) 也将被写入 DMA。

- 如果为加密运算，则 DMA 从 memory 中读取明文数据流并将其传给 AES。AES 计算出密文后将密文写入 DMA。DMA 再将密文写入 memory。
- 如果为解密运算，则 DMA 从 memory 中读取密文数据流并将其传给 AES。AES 计算出明文后将明文写入 DMA。DMA 再将明文写入 memory。

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 的计算过程有所交叠，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 71 所示。

表 71: TEXT-PADDING

Function : TEXT-PADDING()	
<b>Input</b>	: $X$ , bit string.
<b>Output</b>	: $Y = \text{TEXT-PADDING}(X)$ , whose length is the nearest integral multiples of 128 bits.
<b>Steps</b> Let us assume that $X$ is a data-stream that can be split into $n$ parts as following: $X = X_1    X_2    \cdots    X_{n-1}    X_n$ Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equals to 128 bits, and the length of $X_n$ is $t$ ( $0 \leq t \leq 127$ ). If $t = 0$ , then $\text{TEXT-PADDING}(X) = X;$ If $0 < t \leq 127$ , define a 128-bit block, $X_n^*$ , and let $X_n^* = X_n    0^{128-t}$ , then $\text{TEXT-PADDING}(X) = X_1    X_2    \cdots    X_{n-1}    X_n^* = X    0^{128-t}$	

### 16.5.2 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 memory 中以一定的方式来存放，且要求数据量必须是 block 的整数倍。

举例说明，假设 DMA 需要搬运 2 个 block 大小的源数据：

- 十六进制：0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20

假设起始地址为 0x0280，则源数据在 memory 中的存放位置如表 72 所示。结果数据也遵从相同的存放规则，在此不多做介绍。

表 72: 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

另外，值得注意的是，DMA 既可以访问片内存储空间，又可以访问片外 PSRAM。当访问片外 PSRAM 时，基地址必须满足 DMA 对地址的相关要求，但当访问片内存储器空间时则没有限制。

### 16.5.3 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：INC<sub>32</sub> 和 INC<sub>128</sub>。用户可通过将寄存器 AES\_INC\_SEL\_REG 置为 0 或 1 选择 INC<sub>32</sub> 或 INC<sub>128</sub> 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。

### 16.5.4 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。

### 16.5.5 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 等操作，[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-bit 长的比特串，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 72 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。

### 16.5.6 块运算模式的流程

1. 对寄存器 [CRYPTO\\_DMA\\_AES\\_SHA\\_SELECT\\_REG](#) 写入 0。
2. 配置 Crypto DMA 链表并启动 DMA。
3. 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。
  - 初始化寄存器 [AES\\_MODE\\_REG](#)、[AES\\_KEY\\_n\\_REG](#)、[AES\\_ENDIAN\\_REG](#)。
  - 初始化寄存器 [AES\\_BLOCK\\_MODE\\_REG](#)，请参照表 69。
  - 初始化寄存器 [AES\\_BLOCK\\_NUM\\_REG](#)，请参照章节 16.5.4。
  - 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）。
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 2。如果开启了中断功能，也可以等待 [AES\\_INT](#) 中断产生。
6. 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。
7. 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。
8. 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 5 之后。

### 16.5.7 GCM 运算模式的流程

1. 对寄存器 `CRYPTO_DMA_AES_SHA_SELECT_REG` 写入 0。
2. 配置 Crypto DMA 链表并启动 DMA。
3. 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。
  - 初始化寄存器 `AES_MODE_REG`、`AES_KEY_n_REG`、`AES_ENDIAN_REG`。
  - 初始化寄存器 `AES_BLOCK_MODE_REG` 为 6。
  - 初始化寄存器 `AES_BLOCK_NUM_REG`，请参照章节 16.5.4。
  - 初始化寄存器 `AES_AAD_BLOCK_NUM_REG`，请参照章节 16.6.4。
  - 初始化寄存器 `AES_REMAINDER_BIT_NUM_REG`，请参照章节 16.6.5。解密时不使用。
4. 启动运算。对寄存器 `AES_TRIGGER_REG` 写入 1。
5. 轮询寄存器 `AES_STATE_REG`，直到读到 0。请参照表 70。此处无中断产生。
6. 读存储器 `AES_H_MEM` 获取  $H$ 。
7. 软件计算  $J_0$ ，然后将其写入存储器 `AES_J0_MEM`。
8. 继续运算。对寄存器 `AES_CONTINUE_REG` 写入 1。
9. 等待运算完成。轮询寄存器 `AES_STATE_REG`，直到读到 2。请参照表 70。如果开启了中断功能，也可以等待 `AES_INT` 中断产生。
10. 此时， $T_0$  已经准备好。读存储器 `AES_T0_MEM` 获取  $T_0$ 。
11. 确认 DMA 完成数据传输。此时，结果数据已经被 DMA 写入 memory，可以直接从中读取。
12. 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。
13. 对寄存器 `AES_DMA_EXIT_REG` 写入 1。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 9 之后。

## 16.6 GCM 算法

ESP32-S2 中 AES 加速器直接支持 GCM 算法（更多信息，请见 [NIST SP 800-38D](#) 标准）。值得注意的是，考虑到实际使用中很少会使用长度超过  $2^{32} - 1$  比特的  $AAD$ 、 $C$  和  $P$ ，我们在算法中将  $AAD$ 、 $C$  和  $P$  的长度限制在  $2^{32} - 1$  比特之内。图 16-1 显示了 ESP32-S2 内置 AES 加速器实现 GCM 加密算法的流程。

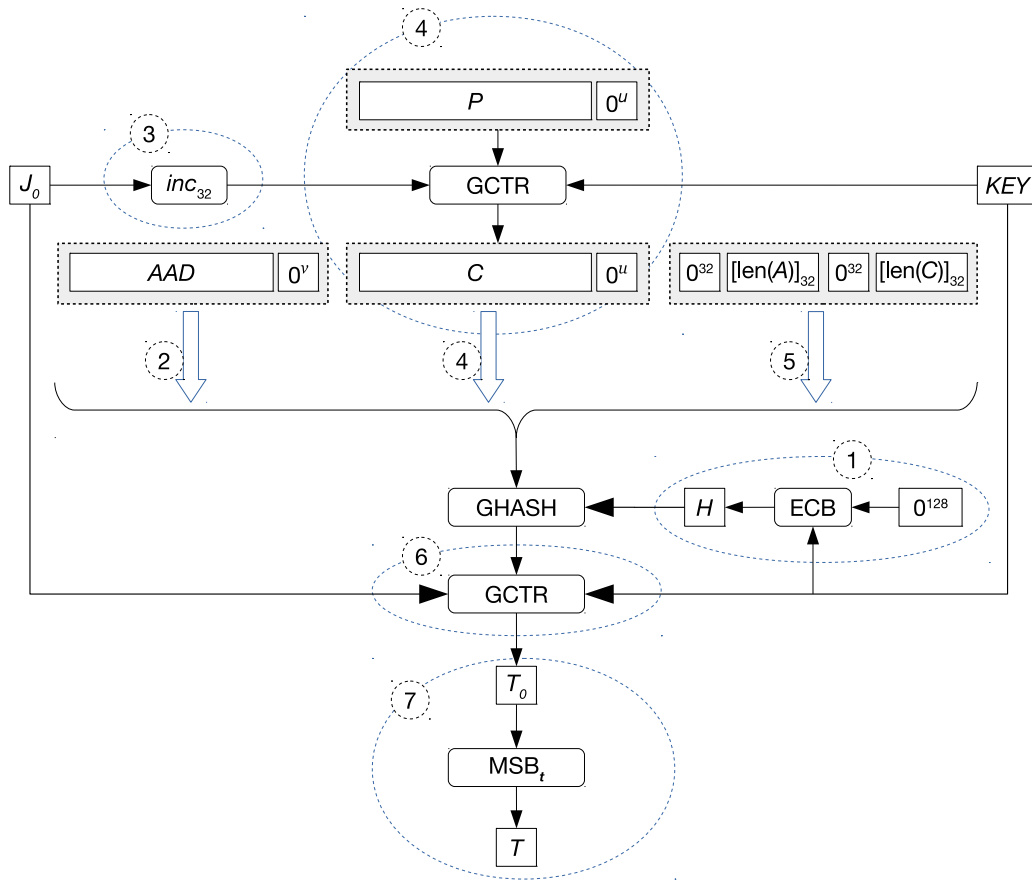


图 16-1. GCM 加密操作流程图

具体步骤描述见下：

1. 硬件调用 ECB 算法求解哈希子密钥 ( $H$ ) 的值，为哈希计算做准备；
2. 硬件调用 GHASH 算法对封装后的  $AAD$  作哈希计算；
3. 硬件调用标准增量函数  $INC_{32}$  对  $J_0$  作运算，为 CTR 加密做准备；
4. 硬件调用 GCTR 算法对封装后的明文  $P$  做加密运算，同时调用 GHASH 算法对封装后的密文  $C$  作哈希计算；
5. 硬件调用 GHASH 算法对追加块做哈希计算，获得最终的 128 位哈希结果；
6. 硬件调用 GCTR 算法对  $J_0$  做加密运算，获得  $T_0$ ；
7. 软件读取硬件结果  $T_0$ ，并调用  $MSB_t$  算法计算最终的认证标签  $T$ 。

GCM 解密运算与 GCM 加密运算基本相同，仅有一处差别：在图 16-1 中的步骤 4 中，使用 GCTR 算法对封装后的密文做解密运算，得出明文。此处不再赘述。

### 16.6.1 哈希子密钥 (Hash subkey)

在 GCM 操作中，哈希子密钥 ( $H$ ) 是一个 128-bit 的值，由硬件计算求出，即图 16-1 中的步骤 1，对应 [NIST SP 800-38D](#) 标准中“7 GCM Specification”中的“Step 1. Let  $H = \text{CIPH}_K(0^{128})$ ”描述。

哈希子密钥 ( $H$ ) 的值以字节序的形式存放在存储器 [AES\\_H\\_MEM](#) 中，且需遵循表 72 中的字节序规，即最左侧字节存放在 [AES\\_H\\_MEM](#) 中的最低地址，最右侧字节存放在 [AES\\_H\\_MEM](#) 中的最高地址。

### 16.6.2 $J_0$

在 GCM 操作中， $J_0$  是一个 128-bit 的值，由软件计算求出，将参与到图 16-1 中的步骤 3 和步骤 6 的计算中。具体的计算方式在 [NIST SP 800-38D](#) 标准中的“7 GCM Specification”章节部分有明确定义。请参阅 [NIST SP 800-38D](#) 获取更多算法标准的相关信息。

$J_0$  值以字节序的形式存放在存储器 [AES\\_J0\\_MEM](#) 中，且需遵循表 72 中的字节序规，即最左侧字节存放在 [AES\\_J0\\_MEM](#) 中的最低地址，最右侧字节存放在 [AES\\_J0\\_MEM](#) 中的最高地址。

### 16.6.3 认证标签 (Authenticated Tag)

认证标签 (Authenticated Tag，简称为 Tag) 是 GCM 计算的最终结果之一，由软件完成计算，对应图 16-1 中的步骤 7，具体取值由长度参数  $t$  ( $1 \leq t \leq 128$ ) 决定：

- 当  $t = 128$  时，Tag 的值记为  $T_0$ 。 $T_0$  是一个 128-bit 的比特串，以字节序的形式存放在存储器 [AES\\_T0\\_MEM](#) 中，且需遵循表 72 中的字节序规，即最左侧字节存放在 [AES\\_T0\\_MEM](#) 中的最低地址，最右侧字节存放在 [AES\\_T0\\_MEM](#) 中的最高地址。
- 当  $1 \leq t < 128$  时，Tag 的值为  $T_0$  中最左侧的  $t$  个比特位，可以用  $\text{MSB}_t(T_0)$  函数表示。例如， $\text{MSB}_4(111011010) = 1110$ ， $\text{MSB}_5(11010011010) = 11010$ 。有关  $\text{MSB}_t()$  函数的具体定义，请见 [NIST SP 800-38D](#) 的“6 Mathematical Components of GCM”章节。

### 16.6.4 附加认证消息块个数 (AAD Block Number)

寄存器 [AES\\_AAD\\_BLOCK\\_NUM\\_REG](#) 存放附加认证消息 (Additional Authenticated Data, AAD) 的块个数，其值等于  $\text{length}(\text{TEXT-PADDING}(\text{AAD}))/128$ 。该寄存器仅在 DMA-AES 工作模式下的 GCM 操作中有意义。

### 16.6.5 不完整块的有效比特数 (Remainder Bit Number)

寄存器 [AES\\_REMAINDER\\_BIT\\_NUM\\_REG](#) 存放明文不完整块的有效比特数，其值等于  $\text{length}(P) \% 128$ 。该寄存器仅在 DMA-AES 工作模式下的 GCM 加密操作中有意义。寄存器 [AES\\_REMAINDER\\_BIT\\_NUM\\_REG](#) 的值不会影响到密文或明文结果，但会影响 Tag 结果值 (包含在  $T_0$  中)。依据 GCM 算法标准，GCM 运算是 GCTR 运算和 GHASH 运算的结合，GCTR 运算用于执行加解密计算，GHASH 运算用于求解 Tag。

- 对于 GCM 加密，硬件首先计算  $C$ ，而后将其封装为  $\text{TEXT-PADDING}(C)$ ，作为 GHASH 运算的输入。此时，硬件需要根据寄存器 [AES\\_REMAINDER\\_BIT\\_NUM\\_REG](#) 的值决定追加 0 的个数。
- 对于 GCM 解密，封装操作  $\text{TEXT-PADDING}(C)$  交由软件完成，此时寄存器 [AES\\_REMAINDER\\_BIT\\_NUM\\_REG](#) 的值不起作用。



## 16.7 基地址

用户可以通过两个不同的寄存器基地址访问 AES，如表 73 所示。更多信息，请前往章节 1 [系统和存储器](#)。

表 73: AES 基地址

访问总线	基地址
PeriBUS1	0x3F43A000
PeriBUS2	0x6003A000

## 16.8 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 AES 基地址的地址偏移量（相对地址）。请参阅章节 16.7 获取有关 AES 基地址的信息。

名称	描述	大小	起始地址	结束地址	访问
<a href="#">AES_IV_MEM</a>	存储器 IV	16 字节	0x0050	0x005F	读 / 写
<a href="#">AES_H_MEM</a>	存储器 H	16 字节	0x0060	0x006F	只读
<a href="#">AES_J0_MEM</a>	存储器 J0	16 字节	0x0070	0x007F	读 / 写
<a href="#">AES_T0_MEM</a>	存储器 T0	16 字节	0x0080	0x008F	只读

## 16.9 寄存器列表

请注意，这里的地址是相对于 AES 基地址的地址偏移量（相对地址）。请参阅章节 16.7 获取有关 AES 基地址的信息。

名称	描述	地址	访问
<b>密钥寄存器</b>			
AES_KEY_0_REG	AES 密钥寄存器 0	0x0000	读 / 写
AES_KEY_1_REG	AES 密钥寄存器 1	0x0004	读 / 写
AES_KEY_2_REG	AES 密钥寄存器 2	0x0008	读 / 写
AES_KEY_3_REG	AES 密钥寄存器 3	0x000C	读 / 写
AES_KEY_4_REG	AES 密钥寄存器 4	0x0010	读 / 写
AES_KEY_5_REG	AES 密钥寄存器 5	0x0014	读 / 写
AES_KEY_6_REG	AES 密钥寄存器 6	0x0018	读 / 写
AES_KEY_7_REG	AES 密钥寄存器 7	0x001C	读 / 写
<b>TEXT_IN 寄存器</b>			
AES_TEXT_IN_0_REG	源数据寄存器 0	0x0020	读 / 写
AES_TEXT_IN_1_REG	源数据寄存器 1	0x0024	读 / 写
AES_TEXT_IN_2_REG	源数据寄存器 2	0x0028	读 / 写
AES_TEXT_IN_3_REG	源数据寄存器 3	0x002C	读 / 写
<b>TEXT_OUT 寄存器</b>			
AES_TEXT_OUT_0_REG	结果数据寄存器 0	0x0030	只读
AES_TEXT_OUT_1_REG	结果数据寄存器 1	0x0034	只读
AES_TEXT_OUT_2_REG	结果数据寄存器 2	0x0038	只读
AES_TEXT_OUT_3_REG	结果数据寄存器 3	0x003C	只读
<b>配置寄存器</b>			
AES_MODE_REG	工作模式选择寄存器	0x0040	读 / 写
AES_ENDIAN_REG	字节序配置寄存器	0x0044	读 / 写
AES_DMA_ENABLE_REG	DMA 使能寄存器	0x0090	读 / 写
AES_BLOCK_MODE_REG	块模式配置寄存器	0x0094	读 / 写
AES_BLOCK_NUM_REG	块数量配置寄存器	0x0098	读 / 写
AES_INC_SEL_REG	标准增量函数选择寄存器	0x009C	读 / 写
AES_AAD_BLOCK_NUM_REG	AAD 块数量配置寄存器	0x00A0	读 / 写
AES_REMAINDER_BIT_NUM_REG	明文或密文的不完整块的有效比特位数	0x00A4	读 / 写
<b>控制 / 状态寄存器</b>			
AES_TRIGGER_REG	开始运算寄存器	0x0048	只写
AES_STATE_REG	运算状态寄存器	0x004C	只读
AES_CONTINUE_REG	继续运算寄存器	0x00A8	只写
AES_DMA_EXIT_REG	退出运算寄存器	0x00B8	只写
<b>中断寄存器</b>			
AES_INT_CLR_REG	DMA-AES 中断清除	0x00AC	只写
AES_INT_ENA_REG	DMA-AES 中断使能寄存器	0x00B0	读 / 写

## 16.10 寄存器

Register 16.1: AES\_KEY\_ *n* \_REG (*n*: 0-7) (0x0000+4\**n*)

31	0
0x00000000	
Reset	

AES\_KEY\_ *n* \_REG (*n*: 0-7) AES 密钥寄存器。(读 / 写)

Register 16.2: AES\_TEXT\_IN\_ *m* \_REG (*m*: 0-3) (0x0020+4\**m*)

31	0
0x00000000	
Reset	

AES\_TEXT\_IN\_ *m* \_REG (*m*: 0-3) Typical AES 文本输入寄存器。(读 / 写)

Register 16.3: AES\_TEXT\_OUT\_ *m* \_REG (*m*: 0-3) (0x0030+4\**m*)

31	0
0x00000000	
Reset	

AES\_TEXT\_OUT\_ *m* \_REG (*m*: 0-3) Typical AES 文本输出寄存器。(只读)

Register 16.4: AES\_MODE\_REG (0x0040)

(reserved)																AES_MODE		
31															3	2	0	
0x00000000																0		Reset

AES\_MODE 选择 AES 加速器的工作模式。详情请见表 63。(读 / 写)

Register 16.5: AES\_ENDIAN\_REG (0x0044)

(reserved)																AES_ENDIAN						
31															6	5	0					
0x00000000																0	0	0	0	0	0	Reset

AES\_ENDIAN 字节序选择寄存器。详情请见表 65。(读 / 写)

Register 16.6: AES\_DMA\_ENABLE\_REG (0x0090)

(reserved)															AES_DMA_ENABLE	
31															1	0
0x00000000															0	Reset

**AES\_DMA\_ENABLE** 选择工作模式。详情请见表 62。(读 / 写)

Register 16.7: AES\_BLOCK\_MODE\_REG (0x0094)

(reserved)															AES_BLOCK_MODE			
31											3	2	0	Reset				
0x00000000												0						

**AES\_BLOCK\_MODE** 选择 DMA-AES 使用的块模式。详情请见表 69。(读 / 写)

Register 16.8: AES\_BLOCK\_NUM\_REG (0x0098)

31																																0	
0x00000000																																	Reset

**AES\_BLOCK\_NUM** 在 DMA-AES 运算中待加解密的文本块数。详情请见章节 16.5.4。(读 / 写)

Register 16.9: AES\_INC\_SEL\_REG (0x009C)

(reserved)															AES_INC_SEL	
31											1	0				Reset
0x00000000												0				

**AES\_INC\_SEL** 选择 CTR 块模式使用的标准增量函数。置 0 选择 INC<sub>32</sub> 标准增量函数，置 1 选择 INC<sub>128</sub> 标准增量函数。(读 / 写)

Register 16.10: AES\_AAD\_BLOCK\_NUM\_REG (0x00A0)

31	0
0x00000000	
Reset	

**AES\_AAD\_BLOCK\_NUM** 在 GCM 运算中 AAD 的块数。详情请见章节 16.6.4。(读 / 写)

Register 16.11: AES\_REMAINDER\_BIT\_NUM\_REG (0x00A4)

(reserved)							AES_REMAINDER_BIT_NUM	
31	7	6	0					
0x00000000							0	Reset

**AES\_REMAINDER\_BIT\_NUM** 在 GCM 运算中输入文本的不完整块的有效比特数。详情请见章节 16.6.5。(读 / 写)

Register 16.12: AES\_TRIGGER\_REG (0x0048)

(reserved)															AES_TRIGGER		
31														1	0		
0x00000000															x	Reset	

**AES\_TRIGGER** 写入 1 使能 AES 运算。(只写)

Register 16.13: AES\_STATE\_REG (0x004C)

(reserved)																															AES_STATE	
31																														2	1	0
0x00000000																													0x0		Reset	

**AES\_STATE** AES 状态寄存器。详见表 64 (Typical AES 工作模式) 和表 70 (DMA-AES 工作模式)。(只读)

Register 16.14: AES\_CONTINUE\_REG (0x00A8)

(reserved)																AES_CONTINUE	
31															1	0	Reset
0x00000000																x	

**AES\_CONTINUE** 在 GCM 运算中，写入 1 继续运算。(只写)

Register 16.15: AES\_DMA\_EXIT\_REG (0x00B8)

(reserved)																															AES_DMA_EXIT	
31																															1	0
0x00000000																															x	Reset

**AES\_DMA\_EXIT** 在 DMA-AES 运算完成后，在下一次配置 AES 任何寄存器之前，写入 1 使 AES 回到空闲状态。(只写)

Register 16.16: AES\_INT\_CLR\_REG (0x00AC)

(reserved)																															AES_INT_CLR	
31																															1	0
0x00000000																															x	Reset

**AES\_INT\_CLR** 写入 1 清除 AES 中断。(只写)

Register 16.17: AES\_INT\_ENA\_REG (0x00B0)

(reserved)																															AES_INT_ENA	
31																															1	0
0x00000000																															0	Reset

**AES\_INT\_ENA** 写入 1 使能 AES 中断功能，写入 0 关闭 AES 中断功能。(读 / 写)

## 17. SHA 加速器

### 17.1 概述

ESP32-S2 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。

### 17.2 主要特性

ESP32-S2 的 SHA 硬件加速器：

- 支持 [FIPS PUB 180-4 规范](#) 的全部运算标准
  - SHA-1 运算
  - SHA-224 运算
  - SHA-256 运算
  - SHA-384 运算
  - SHA-512 运算
  - SHA-512/224 运算
  - SHA-512/256 运算
  - SHA-512/t 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 (interleave) 功能（仅限 Typical SHA 工作模式）
- 允许中断功能（仅限 DMA-SHA 工作模式）

### 17.3 工作模式简介

ESP32-S2 内置的 SHA 加速器支持 [Typical SHA](#) 和 [DMA-SHA](#) 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 crypto 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。

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，先配置的工作模式生效，具体请见表 76。

表 76: 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 77。

表 77: 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
SHA-384	3
SHA-512	4
SHA-512/224	5
SHA-512/256	6
SHA-512/t	7

**注意：**

ESP32-S2 的数字签名和 HMAC 模块也会调用 SHA 加速器。此时，用户无法正常访问 SHA 加速器。

## 17.4 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](#)。

### 17.4.1 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](#)。

#### 17.4.1.1 附加填充比特

SHA 加速器仅能处理长度为 512 位及其倍数或 1024 位及其倍数的信息。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，则针对不同运算标准的填充步骤见下：

- SHA-1、SHA-224 和 SHA-256

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  $m + 1 + k \equiv 448 \mod 512$  的最小非负数解；



- 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。

- **SHA-384、SHA-512、SHA-512/224、SHA-512/256 和 SHA-512/t**

- 首先，在待处理信息后填充 1 个“1”；
- 随后，再填充  $k$  个“0”。其中， $k$  为满足  $m + 1 + k \equiv 896 \bmod 1024$  的最小非负数解；
- 最后，在末尾填充一个 128 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](#) 中的“5.1 Padding the Message”章节。

### 17.4.1.2 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 位或 1024 位的信息块。

- 对于 **SHA-1、SHA-224 和 SHA-256**：待处理信息（及其填充）应解析为  $N$  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  $M_{15}^{(i)}$ 。
- 对于 **SHA-384、SHA-512、SHA-512/224、SHA-512/256 和 SHA-512/t**：待处理信息（及其填充）应解析为  $N$  个 1024 位的信息块，即  $M^{(1)}$ 、 $M^{(2)}$ 、...、 $M^{(N)}$ 。一个 1024 位信息块包括 16 个 64 位的字 (word)，则第  $i$  个信息块的第一个 64 位字表示为  $M_0^{(i)}$ ，第二个 64 位字表示为  $M_1^{(i)}$ ，...，第 16 个 64 位字表示为  $M_{15}^{(i)}$ 。

在 Typical SHA 工作模式下，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：

- **SHA-1、SHA-224、SHA-256** 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。
- **SHA-384、SHA-512、SHA-512/224、SHA-512/256** 将  $M_0^{(i)}$  的高 32 位存放在 [SHA\\_M\\_0\\_REG](#) 中，低 32 位存放在 [SHA\\_M\\_1\\_REG](#)， $M_1^{(i)}$  的高 32 位存放在 [SHA\\_M\\_2\\_REG](#) 中，低 32 位存放在 [SHA\\_M\\_3\\_REG](#)，...， $M_{15}^{(i)}$  的高 32 位存放在 [SHA\\_M\\_30\\_REG](#) 中，低 32 位存放在 [SHA\\_M\\_31\\_REG](#)。

**说明：**

有关“信息块”及相关概念的描述，请参考 [FIPS PUB 180-4 规范](#) 中“2.1 Glossary of Terms and Acronyms”章节。

在 DMA-SHA 工作模式下，需要预先完成如下配置：

- 创建外发链表；
- 根据章节 *DMA Controller* 完成链表配置，包括但不限于将字节数组（完成填充后的信息）的首地址配置给发送链表的 buffer 地址指针；
- 配置寄存器 [CRYPTO\\_DMA\\_OUTLINK\\_ADDR](#) 指向第一个发送链表；
- 将数值 1 写入寄存器 [CRYPTO\\_DMA\\_OUTLINK\\_START](#)，启动 DMA 模块开始搬运数据；
- 将数值 1 写入寄存器 [CRYPTO\\_DMA\\_AES\\_SHA\\_SELECT\\_REG](#)，将 AES 和 SHA 共用的 DMA 资源分配给 SHA 加速器使用。

### 17.4.1.3 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  $H^{(0)}$ 。不同运算标准的哈希初始值设置要求不同，其中 SHA-1、SHA-224、SHA-256、SHA-384、SHA-512、SHA-512/224、SHA-512/256 等运算的哈希初始值为常量 C，且已经固定在硬件中，无需专门计算。

然而，SHA-512/ $t$  对于不同的  $t$  均需要一个不同的哈希初始值。简单来说，SHA-512/ $t$  是一种基于 SHA-512 的  $t$  位运算标准，其运算结果将截断至  $t$  位。其中，运算标准对  $t$  值的要求为“大于 0 小于 512 且不等于 384 的正整数”。对于不同  $t$  值的 SHA-512/ $t$  运算，其哈希初始值可通过对“SHA-512/ $t$ ”字符串的十六进制表示进行 SHA-512 运算获得。不难看出，对于  $t$  取值不同的 SHA-512/ $t$  运算标准，其不同之处仅在于  $t$  值不同。

因此，为了简化 SHA-512/ $t$  的哈希初始值计算，我们特别提出了以下方法：

1. **计算  $t\_string$  和  $t\_length$** ：其中， $t\_string$  为  $t$  的字符串信息，长度为 32-bit。 $t\_length$  指明字符串长度信息，长度为 7-bit。根据  $t$  的取值范围不同， $t\_string$  和  $t\_length$  的计算过程如下：

- 如果  $1 \leq t \leq 9$ ，则  $t\_length = 7'h48$ ， $t\_string$  需要按照如下格式封装：

8'h3t <sub>0</sub>	1'b1	23'b0
--------------------	------	-------

其中， $t_0 = t$ 。

举例，如果  $t = 8$ ，则  $t_0 = 8$ ， $t\_string = 32'h38800000$ 。

- 如果  $10 \leq t \leq 99$ ，则  $t\_length = 7'h50$ ， $t\_string$  需要按照如下格式封装：

8'h3t <sub>1</sub>	8'h3t <sub>0</sub>	1'b1	15'b0
--------------------	--------------------	------	-------

其中， $t_0 = t \% 10$ ， $t_1 = t / 10$ 。

举例，如果  $t = 56$ ，则  $t_0 = 6$ ， $t_1 = 5$ ， $t\_string = 32'h35368000$ 。

- 如果  $100 \leq t < 512$ ，则  $t\_length = 7'h58$ ， $t\_string$  需要按照如下格式封装：

8'h3t <sub>2</sub>	8'h3t <sub>1</sub>	8'h3t <sub>0</sub>	1'b1	7'b0
--------------------	--------------------	--------------------	------	------

其中， $t_0 = t \% 10$ ， $t_1 = (t / 10) \% 10$ ， $t_2 = t / 100$ 。

举例，如果  $t = 231$ ，则  $t_0 = 1$ ， $t_1 = 3$ ， $t_2 = 2$ ， $t\_string = 32'h32333180$ 。

2. **配置计算哈希初始值所需的寄存器**：用  $t\_string$  和  $t\_length$  初始化文本寄存器 [SHA\\_T\\_STRING\\_REG](#) 和 [SHA\\_T\\_LENGTH\\_REG](#)。
3. **计算得到哈希初始值**：对 [SHA\\_MODE\\_REG](#) 寄存器置 7 选择 SHA-512/ $t$  运算，并对 [SHA\\_START\\_REG](#) 寄存器置 1，启动 SHA 加速器的运算即可。最后，轮询寄存器 [SHA\\_BUSY\\_REG](#) 结果为 0，则哈希初始值已计算完毕。

此外，您也可以按照 [FIPS PUB 180-4 Spec](#) 中“5.3.6 SHA-512/ $t$ ”章节的描述计算 SHA-512/ $t$  的哈希初始值，也就是对“SHA-512/ $t$ ”字符串的十六进制表示进行一次“特殊”的 SHA-512 运算，其运算得到的信息摘要即为所需的哈希初始值。这里的“特殊”指本次 SHA-512 运算的哈希初始值为“SHA-512 运算标准的初始值常量 C 与 0xa5 每 8 位进行一次异或位运算后得到的结果”。

### 17.4.2 哈希运算流程

在完成信息预处理后，ESP32-S2 SHA 加速器将正式开始哈希运算，并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-S2 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。

#### 17.4.2.1 Typical SHA 模式下的运算流程

ESP32-S2 SHA 加速器在 Typical SHA 工作模式下支持两种运算方法：

- “alone” 运算方法：在计算完所有信息块并得到全部信息摘要前，用户不会插入其他运算任务。
- “interleave” 运算方法：在计算完每一个信息块后，用户都可以将存储在 [SHA\\_H\\_n\\_REG](#) 寄存器中的信息摘要暂存起来，然后插入优先级更高的运算任务，包括 Typical SHA 运算和 DMA-SHA 运算。当临时任务结束后，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 (SHA-512/t 除外)

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 77。
2. 处理当前信息块。
  - (a) 将当前信息块写入 [SHA\\_M\\_n\\_REG](#) 寄存器堆；
  - (b) 启动 SHA 加速器<sup>1</sup>：
    - 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，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。
  - (c) 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。然后执行步骤 3。
3. 选择是否要插入其他运算。
  - 如需插入，则准备移交 SHA 加速器使用权：
    - (a) 读取并保存寄存器 [SHA\\_MODE\\_REG](#) 中的运算标准类型；
    - (b) 读取并保存寄存器堆 [SHA\\_H\\_n\\_REG](#) 中的信息摘要；
    - (c) 最后，跳转至相应流程进行插入运算。具体按照插入运行类型的不同，请见 [Typical SHA](#) 或 [DMA-SHA](#) 章节。
  - 如无需插入，则继续执行步骤 4。
4. 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，进入步骤 5。

5. 判断是否需要交还 SHA 加速器使用权（即判断本次运算是否为插入运算）：

- 如果是插入运算，则应交还 SHA 加速器的使用权：
  - (a) 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`;
  - (b) 将获得使用权前保存的信息摘要写入寄存器堆 `SHA_H_n_REG`;
  - (c) 然后执行步骤 2。
- 如果不是插入运算，则无需交还 SHA 加速器的使用权。此时，请执行步骤 6。

6. 获取信息摘要：

- 从寄存器堆 `SHA_H_n_REG` 取出信息摘要。

### Typical SHA 的具体运算流程 (SHA-512/t)

1. 选择运算标准。

- 配置 `SHA_MODE_REG` 寄存器为 7 选择 SHA-512/t 运算标准。

2. 计算哈希初始值。

- (a) 配置 `t_string` 和 `t_length`，并将其写入 `SHA_T_STRING_REG` 和 `SHA_T_LENGTH_REG` 寄存器。具体请见 17.4.1.3 章节。
- (b) 对 `SHA_START_REG` 寄存器置 1，启动 SHA 加速器的运算。
- (c) 轮询寄存器 `SHA_BUSY_REG` 一直到读回的值为 0，代表 SHA 硬件加速器已完成哈希初始值的计算。

3. 处理当前信息块。

- (a) 将当前信息块写入 `SHA_M_n_REG` 寄存器堆；
- (b) 启动 SHA 加速器<sup>1</sup>：
  - 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。
- (c) 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。然后执行步骤 4。

4. 选择是否要插入其他运算。

- 如需插入，则准备移交 SHA 加速器使用权：
  - (a) 读取并保存寄存器 `SHA_MODE_REG` 中的运算标准类型；
  - (b) 读取并保存寄存器堆 `SHA_H_n_REG` 中的信息摘要；
  - (c) 最后，跳转至相应流程进行插入运算。具体按照插入运行类型的不同，请见 Typical SHA 或 DMA-SHA 章节。
- 如无需插入，则继续执行步骤 5。

## 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 3。
- 否则，进入步骤 6。

## 6. 判断是否需要交还 SHA 加速器使用权（即判断本次运算是否为插入运算）：

- 如果是插入运算，则应交还 SHA 加速器的使用权：
  - (a) 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`;
  - (b) 将获得使用权前保存的信息摘要写入寄存器堆 `SHA_Hn_REG`;
  - (c) 然后执行步骤 3。
- 如果不是插入运算，则无需交还 SHA 加速器的使用权。此时，请执行步骤 7。

## 7. 获取信息摘要：

- 从寄存器堆 `SHA_Hn_REG` 取出信息摘要。

**说明：**

1. 在第 2b 步时，在 SHA 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_Mn_REG` 寄存器堆，以节省时间。

### 17.4.2.2 DMA-SHA 模式下的运算流程

ESP32-S2 SHA 加速器在 DMA-SHA 工作模式下不支持“interleave”运算方法，即在每次运算全部完成前，不允许插入其他运算任务。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-SHA 运算。每次 DMA-SHA 运算之间允许插入其他运算标准的计算任务。

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硬件完成。用户先按照 17.4.1.2 章节配置 DMA 控制器。

**DMA-SHA 的具体工作流程（SHA-512/t 除外）**

## 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 77。

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 运算为接着另一次 DMA-SHA 的运算，需要提前将另一次计算得到的信息摘要写入寄存器堆 `SHA_Hn_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<sub>n</sub>\\_REG](#) 取出信息摘要。

#### **DMA-SHA 的具体工作流程 (SHA-512/t)**

1. 选择运算标准。

- 配置 [SHA\\_MODE\\_REG](#) 寄存器为 7 选择 SHA-512/t 运算标准。

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. 计算哈希初始值。

- (a) 配置 `t_string` 和 `t_length`，并将其写入 [SHA\\_T\\_STRING\\_REG](#) 和 [SHA\\_T\\_LENGTH\\_REG](#) 寄存器。具体请见 17.4.1.3 章节。
- (b) 对 [SHA\\_START\\_REG](#) 寄存器置 1，启动 SHA 加速器的运算。
- (c) 轮询寄存器 [SHA\\_BUSY\\_REG](#) 一直到读回的值为 0，代表 SHA 硬件加速器已完成哈希初始值的计算。

4. 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。

5. 开始 DMA-SHA 运算。

- 对 [SHA\\_DMA\\_CONTINUE\\_REG](#) 置 1 启动 SHA 加速器。

6. 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。

7. 获取信息摘要

- 从寄存器堆 [SHA\\_H<sub>n</sub>\\_REG](#) 取出信息摘要。

### 17.4.3 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` ( $n$ : 0~15) 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 81：

表 81: 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 <sup>1</sup>
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
SHA-384	384	SHA_H_0_REG ~ SHA_H_11_REG
SHA-512	512	SHA_H_0_REG ~ SHA_H_15_REG
SHA-512/224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-512/256	256	SHA_H_0_REG ~ SHA_H_7_REG
SHA-512/ $t$ <sup>2</sup>	$t$	SHA_H_0_REG ~ SHA_H_x_REG

**说明：**

- 信息摘要从左至右存放，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。
- SHA-512/ $t$  运算标准使用的寄存器与  $t$  的取值有关。 $x+1$  代表用于存储  $t$  位信息摘要的 32 位寄存器个数，因此  $x = \text{roundup}(t/32)-1$ 。举例：
  - 当  $t = 8$  时，则  $x = 0$ ，代表最终的信息摘要长度为 8 位，存放在寄存器 `SHA_H_0_REG` 的高 8 位中；
  - 当  $t = 32$  时，则  $x = 0$ ，代表最终的信息摘要长度为 32 位，存放在寄存器 `SHA_H_0_REG` 中；
  - 当  $t = 132$  时，则  $x = 4$ ，代表最终的信息摘要长度为 132 位，存放在寄存器 `SHA_H_0_REG`、`SHA_H_1_REG`、`SHA_H_2_REG`、`SHA_H_3_REG`，及 `SHA_H_4_REG` 中。

### 17.4.4 中断

SHA 加速器在 DMA-SHA 工作模式下允许中断发生。用户可通过将 `SHA_INT_ENA_REG` 寄存器配置为 1 开启中断。如开启中断功能，SHA 加速器在完成运算时，中断发生。注意，该中断必须由软件将 `SHA_INT_CLEAR_REG` 寄存器置为 1 进行清除。由于 SHA 加速器在 Typical SHA 工作模式下的时间开销较小，因此不支持中断功能。

## 17.5 基地址

用户可以通过两个不同的寄存器基地址访问 SHA，如表 82 所示。更多信息，请前往章节 1 系统和存储器。

表 82: SHA 基地址

基地址	地址值
PeriBUS1	0x3F43B000
PeriBUS2	0x6003B000



## 17.6 寄存器列表

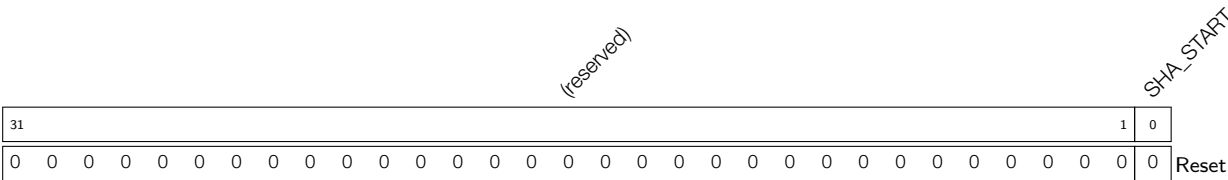
名称	描述	地址	访问权限
<b>控制与状态寄存器</b>			
SHA_CONTINUE_REG	继续 SHA 运算（仅用于 Typical SHA 模式）	0x0014	WO
SHA_BUSY_REG	指示 SHA 加速器是否处于“忙碌”状态	0x0018	RO
SHA_DMA_START_REG	启动 SHA 加速器的 DMA-SHA 模式	0x001C	WO
SHA_START_REG	启动 SHA 加速器的 Typical SHA 模式	0x0010	WO
SHA_DMA_CONTINUE_REG	继续 SHA 运算（仅用于 DMA-SHA 模式）	0x0020	WO
SHA_INT_CLEAR_REG	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
<b>版本寄存器</b>			
SHA_DATE_REG	版本控制寄存器	0x002C	R/W
<b>配置寄存器</b>			
SHA_MODE_REG	配置 SHA 加速器的运算标准	0x0000	R/W
SHA_T_STRING_REG	哈希字符串内容寄存器（仅用于计算 SHA-512/t 的哈希初始值）	0x0004	R/W
SHA_T_LENGTH_REG	哈希字符串长度寄存器（仅用于计算 SHA-512/t 的哈希初始值）	0x0008	R/W
<b>存储器</b>			
SHA_DMA_BLOCK_NUM_REG	信息块个数寄存器（仅用于 DMA-SHA 工作模式）	0x000C	R/W
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_H_8_REG	哈希值	0x0060	R/W
SHA_H_9_REG	哈希值	0x0064	R/W
SHA_H_10_REG	哈希值	0x0068	R/W
SHA_H_11_REG	哈希值	0x006C	R/W
SHA_H_12_REG	哈希值	0x0070	R/W
SHA_H_13_REG	哈希值	0x0074	R/W
SHA_H_14_REG	哈希值	0x0078	R/W
SHA_H_15_REG	哈希值	0x007C	R/W
SHA_M_0_REG	输入信息	0x0080	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



名称	描述	地址	访 问 权限
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
SHA_M_16_REG	输入信息	0x00C0	R/W
SHA_M_17_REG	输入信息	0x00C4	R/W
SHA_M_18_REG	输入信息	0x00C8	R/W
SHA_M_19_REG	输入信息	0x00CC	R/W
SHA_M_20_REG	输入信息	0x00D0	R/W
SHA_M_21_REG	输入信息	0x00D4	R/W
SHA_M_22_REG	输入信息	0x00D8	R/W
SHA_M_23_REG	输入信息	0x00DC	R/W
SHA_M_24_REG	输入信息	0x00E0	R/W
SHA_M_25_REG	输入信息	0x00E4	R/W
SHA_M_26_REG	输入信息	0x00E8	R/W
SHA_M_27_REG	输入信息	0x00EC	R/W
SHA_M_28_REG	输入信息	0x00F0	R/W
SHA_M_29_REG	输入信息	0x00F4	R/W
SHA_M_30_REG	输入信息	0x00F8	R/W
SHA_M_31_REG	输入信息	0x00FC	R/W

17.7 寄存器

Register 17.1: SHA\_START\_REG (0x0010)



SHA\_START 置 1 启动 SHA 加速器的 Typical SHA 模式。(只写)

Register 17.2: SHA\_CONTINUE\_REG (0x0014)

(reserved)																														SHA_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

Reset

**SHA\_CONTINUE** 置 1 继续 SHA 加速器的 Typical SHA 运算。(只写)

Register 17.3: SHA\_BUSY\_REG (0x0018)

(reserved)																														SHA_BUSY_STATE	
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

Reset

**SHA\_BUSY\_STATE** 指示 SHA 是否处于“忙碌”状态。(只读) 1'h0: 空闲 1'h1: 忙碌

Register 17.4: 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

Reset

**SHA\_DMA\_START** 置 1 启动 SHA 加速器的 DMA-SHA 模式。(只写)

Register 17.5: 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

Reset

**SHA\_DMA\_CONTINUE** 置 1 继续 SHA 加速器的 DMA-SHA 运算。(只写)

Register 17.6: 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

Reset

SHA\_CLEAR\_INTERRUPT 清除 DMA-SHA 中断。(只写)

Register 17.7: 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

Reset

SHA\_INTERRUPT\_ENA 使能 DMA-SHA 中断。(读写)

Register 17.8: SHA\_DATE\_REG (0x002C)

(reserved)			SHA_DATE																											
31	30	29																											0	
0	0	0x20190402																										Reset		

SHA\_DATE 版本控制寄存器。(读写)

Register 17.9: SHA\_MODE\_REG (0x0000)

(reserved)																												SHA_MODE			
31																												3	2	0	
0 0																												0x0		Reset	

SHA\_MODE 选择 SHA 加速器的运算标准，详见表 77。(R/W)

## SHA1\_STRING

**SHA\_T\_STRING** 存储哈希字符串内容（仅用于计算 SHA-512/t 的哈希初始值）。（读写）

(reserved)

SHA\_T\_LENGTH

**SHA\_DMA\_BLOCK\_NUM** 定义 DMA-SHA 工作模式下的信息块个数。(读写)

(reserved)

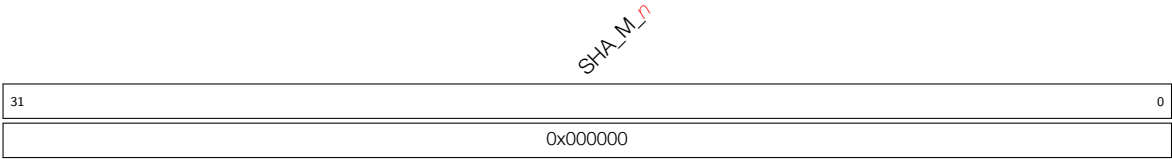
SHA\_DMA\_BLOCK\_NUM

**SHA\_DMA\_BLOCK\_NUM** 定义 DMA-SHA 工作模式下的信息块个数。(读写)

## SHA\_H\_m

**SHA\_H\_***n* 存储第 *n* 个 32 位哈希值。(读写)

Register 17.14: SHA\_M\_0\_REG (0: 0-31) (0x0080+4\*0)



SHA\_M\_0 存储第 0 个 32 位输入信息。(读写)

## 18. RSA 加速器

### 18.1 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软件复杂度，且支持多种“运算子长度”，具有很高的运算效率。

### 18.2 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
- 中断功能

### 18.3 功能描述

RSA 加速器的激活仅需使能 DPORT\_CRYPTORSA\_CLK\_EN 外围时钟的 DPORT\_PERIP\_CLK\_EN1\_REG 位，并同时清零 DPORT\_RSA\_PD\_CTRL\_REG 寄存器中的 DPORT\_RSA\_PD 位。

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，寄存器 [RSA\\_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，软件需先查询寄存器 [RSA\\_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\\_INTERRUPT\\_ENA\\_REG](#) 写 1 / 0 以开启 / 关闭中断。RSA 加速器的中断功能默认开启。

#### 注意：

ESP32-S2 的 [数字签名](#) 模块也会调用 RSA 加速器。此时，用户无法正常访问 RSA 加速器。

### 18.3.1 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 Montgomery Multiplication（蒙哥马利乘法）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 加速器支持长度为  $N = 32 \times x$  ( $x \in \{1, 2, 3, \dots, 128\}$ ) 的大数模幂运算。 $Z$ 、 $X$ 、 $Y$ 、 $M$  和  $\bar{r}$  的位宽为这 128 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  $\bar{r} = R^2 \bmod M$ 。

$M'$  可使用下方公式计算：

$$M^{-1} \times M + 1 = R \times R^{-1}$$

$$M' = M^{-1} \bmod b$$

注意，上方公式适用于使用扩展二进制 GCD 算法的运算。

大数模幂运算的软件流程为：

1. 对寄存器 [RSA\\_INTERRUPT\\_ENA\\_REG](#) 写 1 / 0 以开启 / 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 [18.3.4](#) 获取详细信息。

3. 将  $X_i$ 、 $Y_i$ 、 $M_i$ 、 $\bar{r}_i (i \in \{0, 1, \dots, n\})$  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 128 字 (word)。每块存储器的每一个字刚好存放一个  $b$  进制数。这些存储器都是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. 对寄存器 [RSA\\_MODEXP\\_START\\_REG](#) 写入 1 启动计算。
5. 等待运算结束。轮询寄存器 [RSA\\_IDLE\\_REG](#) 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  $Z_i (i \in \{0, 1, \dots, n\})$ 。
7. 若中断功能已开启，对寄存器 [RSA\\_CLEAR\\_INTERRUPT\\_REG](#) 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。

### 18.3.2 大数模乘运算

大数模乘运算也是基于 Montgomery Multiplication 实现运算  $Z = X \times Y \bmod M$ ，所以也需要预先通过软件计算得到  $\bar{r}$  和  $M'$ 。

RSA 加速器也支持 128 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\\_INTERRUPT\\_ENA\\_REG](#) 写 1 / 0 以开启 / 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  $X_i$ 、 $Y_i$ 、 $M_i$ 、 $\bar{r}_i (i \in \{0, 1, \dots, n\})$  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 128 字 (word)。

每块存储器的每一个字刚好存放一个  $b$  进制数。这些存储器都是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. 对寄存器 [RSA\\_MODMULT\\_START\\_REG](#) 写入 1。
5. 等待运算结束。轮询寄存器 [RSA\\_IDLE\\_REG](#) 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  $Z_i (i \in \{0, 1, \dots, n\})$ 。
7. 若中断功能已开启，对寄存器 [RSA\\_CLEAR\\_INTERRUPT\\_REG](#) 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，只需要更新必需的寄存器与存储器即可。



### 18.3.3 大数乘法运算

大数乘法运算实现了  $Z = X \times Y$ 。其中  $Z$  的长度是运算符  $X$ 、 $Y$  长度的两倍。所以 RSA 加速器只支持运算符长度为  $N = 32 \times x$  ( $x \in \{1, 2, \dots, 64\}$ ) 的大数乘法运算。运算符  $Z$  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_INTERRUPT_ENA_REG` 写 1 / 0 以开启 / 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  $X_i$ 、 $Y_i$  ( $i \in \{0, 1, \dots, n\}$ ) 分别写入存储器 `RSA_X_MEM`、`RSA_Z_MEM`。每块存储器的容量都是 128 字。每块存储器的每一个字刚好存放一个  $b$  进制数。这些存储器都是低地址存放运算符的低位进制数，高地址存放运算符的高位进制数。 $n$  为  $\frac{N}{32}$ 。

$X_i$  ( $i \in \{0, 1, \dots, n\}$ ) 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， $Y_i$  ( $i \in \{0, 1, \dots, n\}$ ) 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  $4 \times (n+i)$ 。

只需要根据运算符长度，将这两个运算符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. 对寄存器 `RSA_MULT_START_REG` 写入 1。
5. 等待运算结束。轮询寄存器 `RSA_IDLE_REG` 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  $Z_i$  ( $i \in \{0, 1, \dots, n\}$ )。 $\hat{n}$  为  $2 \times n$ 。
7. 若中断功能已开启，对寄存器 `RSA_CLEAR_INTERRUPT_REG` 写入 1 以清除中断。

运算结束后，寄存器 `RSA_MODE_REG` 中存储的运算符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。

### 18.3.4 加速选项

ESP32-S2 RSA 提供了两种加速选项，分别为 SEARCH 选项和 CONSTANT\_TIME 选项，专用于加速大数模幂运算。这两个加速选项默认关闭，但可以同时使用。

当两个加速选项均关闭时，求解  $Z = X^Y \bmod M$  的时间开销完全由运算符长度决定。然后，当任一加速选项开启时，运算的时间开销还与  $Y$  的 0/1 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，当启动加速选项时：

- SEARCH 选项
  - RSA 加速器将忽略所有  $\tilde{Y}_i$  ( $i > \alpha$ ) 位。其中，加速位置  $\alpha$  可通过 `RSA_SEARCH_POS_REG` 寄存器配置。 $\alpha$  的最大值不能超过  $N-1$ ，否则相当于没有加速；且不建议小于  $t$ ，否则无法正确求解  $Z = X^Y \bmod M$ 。当设置  $\alpha$  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。
- CONSTANT\_TIME 选项
  - RSA 加速器在运算过程中将简化对  $Y$  中 0 位的处理。因此不难想象， $Y$  中的 0 越多，加速效果越明显。

为了直观地展示加速选项带来的加速效果，下面通过一个典型实例加以说明。在  $Z = X^Y \bmod M$  中， $N$  等于 3072， $Y$  等于 65537。表 84 展示了 4 种加速选项组合对应的时间开销。可以看到，相比于不开启任何加速选项，开启加速选项时的时间开销明显大幅度降低。注意，这里 SEARCH 选项开启时设定  $\alpha$  为 16。

表 84: 加速效果

SEARCH 选项	CONSTANT_TIME 选项	时间开销	提速比例
关闭	关闭	376.405 ms	0%
开启	关闭	2.260 ms	99.41%
关闭	开启	1.203 ms	99.68%
开启	开启	1.165 ms	99.69%

## 18.4 基地址

用户可以通过两个不同的寄存器基地址访问 RSA，如表 85 所示。更多有关总线的信息，请见章节 1 系统和存储器。

表 85: RSA 基地址

外设访问方式	地址值
PeriBUS1	0x3F43C000
PeriBUS2	0x6003C000

## 18.5 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 RSA 基地址的地址偏移量（相对地址）。请参阅章节 18.4 获取有关 RSA 基地址的信息。

名称	描述	大小（字节）	起始地址	结束地址	访问
<a href="#">RSA_M_MEM</a>	存储器 M	512	0x0000	0x01FF	只写
<a href="#">RSA_Z_MEM</a>	存储器 Z	512	0x0200	0x03FF	读 / 写
<a href="#">RSA_Y_MEM</a>	存储器 Y	512	0x0400	0x05FF	只写

RSA_X_MEM	存储器 X	512	0x0600	0x07FF	只写
-----------	-------	-----	--------	--------	----

### 18.6 寄存器列表

请注意，这里的地址是相对于 RSA 基地址的地址偏移量（相对地址）。请参阅章节 18.4 获取有关 RSA 基地址的信息。

名称	描述	地址	访问
<b>配置寄存器</b>			
RSA_M_PRIME_REG	M' 存储器	0x0800	读 / 写
RSA_MODE_REG	RSA 长度模式	0x0804	读 / 写
RSA_CONSTANT_TIME_REG	固定时间选项	0x0820	读 / 写
RSA_SEARCH_ENABLE_REG	使能 search 加速选项	0x0824	读 / 写
RSA_SEARCH_POS_REG	search 起始位置	0x0828	读 / 写
<b>状态/控制寄存器</b>			
RSA_CLEAN_REG	RSA 清除寄存器	0x0808	只读
RSA_MODEXP_START_REG	模幂运算起始位	0x080C	只写
RSA_MODMULT_START_REG	模乘运算起始位	0x0810	只写
RSA_MULT_START_REG	乘法运算起始位	0x0814	只写
RSA_IDLE_REG	RSA 闲置寄存器	0x0818	只读
<b>中断寄存器</b>			
RSA_CLEAR_INTERRUPT_REG	RSA 中断清除寄存器	0x081C	只写
RSA_INTERRUPT_ENA_REG	RSA 中断使能寄存器	0x082C	读 / 写
<b>版本寄存器</b>			
RSA_DATE_REG	RSA 日期与版本寄存器	0x0830	读 / 写

### 18.7 寄存器

请注意，本章节的地址是相对于 RSA 基地址的地址偏移量（相对地址）。请参阅章节 18.4 获取有关 RSA 基地址的信息。

Register 18.1: RSA\_M\_PRIME\_REG (0x0800)

31	0
0x00000000	
Reset	

**RSA\_M\_PRIME\_REG** 此寄存器存储 M'。（读 / 写）

Register 18.2: RSA\_MODE\_REG (0x0804)

(reserved)																												RSA_MODE																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
31																												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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**RSA\_MODE** 此寄存器存储模幂运算的模式。(读 / 写)

Register 18.3: RSA\_CLEAN\_REG (0x0808)

(reserved)																																RSA_CLEAN																
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	Reset														

**RSA\_CLEAN** 一旦存储器初始化结束，此位为 1。(只读)

Register 18.4: RSA\_MODEXP\_START\_REG (0x080C)

(reserved)																																RSA_MODE		
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	Reset

**RSA\_MODEXP\_START** 写入 1 以开始模幂运算。(只写)

Register 18.5: RSA\_MODMULT\_START\_REG (0x0810)

(reserved)																															RSA_MODMULT_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	Reset

**RSA\_MODMULT\_START** 写入 1 以开始模乘运算。(只写)

Register 18.6: RSA\_MULT\_START\_REG (0x0814)

(reserved)																														RSA_MULT_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

Reset

**RSA\_MULT\_START** 写入 1 以开始乘法运算。(只写)

Register 18.7: RSA\_IDLE\_REG (0x0818)

(reserved)																														RSA_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

Reset

**RSA\_IDLE** 当 RSA 空闲时，此位为 1。(只读)

Register 18.8: RSA\_CLEAR\_INTERRUPT\_REG (0x081C)

(reserved)																														RSA_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

Reset

**RSA\_CLEAR\_INTERRUPT** RSA 中断清除寄存器。写入 1 清除中断。(只写)

Register 18.9: RSA\_CONSTANT\_TIME\_REG (0x0820)

(reserved)																														RSA_CONSTANT_TIME	
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	1	1

Reset

**RSA\_CONSTANT\_TIME\_REG** 模幂运算中的 constant\_time 加速选项。0: 开启; 1: 关闭 (默认)。(读 / 写)

Register 18.10: RSA\_SEARCH\_ENABLE\_REG (0x0824)

(reserved)																															RSA_SEARCH_ENABLE			
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	Reset

**RSA\_SEARCH\_ENABLE** 模幂运算中的 search 加速选项。1：开启；0：关闭（默认）。（读 / 写）

Register 18.11: RSA\_SEARCH\_POS\_REG (0x0828)

(reserved)																RSA_SEARCH_POS																																															
31																12																11																0															
0 0																0x000																Reset																															

**RSA\_SEARCH\_POS** 模幂运算中的 search 加速选项。用于配置 search 的起始位置（读 / 写）。

Register 18.12: RSA\_INTERRUPT\_ENA\_REG (0x082C)

(reserved)																															RSA_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	1	Reset

**RSA\_INTERRUPT\_ENA** RSA 中断使能寄存器。写入 1 开启中断，默认开启。（读 / 写）

Register 18.13: RSA\_DATE\_REG (0x0830)

(reserved)																															RSA_DATE																														
31	30	29																																																								0			
0	0	0x20190425																																																							Reset				

**RSA\_DATE** 版本控制寄存器（读 / 写）。

## 19. 随机数发生器

### 19.1 概述

ESP32-S2 内置一个真随机数发生器，其生成的随机数可作为加密等操作的基础。

### 19.2 主要特性

该随机数发生器可生成真随机数，即所有生成的随机数在特定范围内出现的概率完全一致。

### 19.3 功能描述

在正确使用的情况下，系统每次从随机数发生器中的寄存器 `RNG_DATA_REG` 读到的每一个 32 位值都是真随机数。这些真随机数来自系统中的热噪声。

噪声源可以来自高速 ADC 或 SAR ADC 或两者兼有。

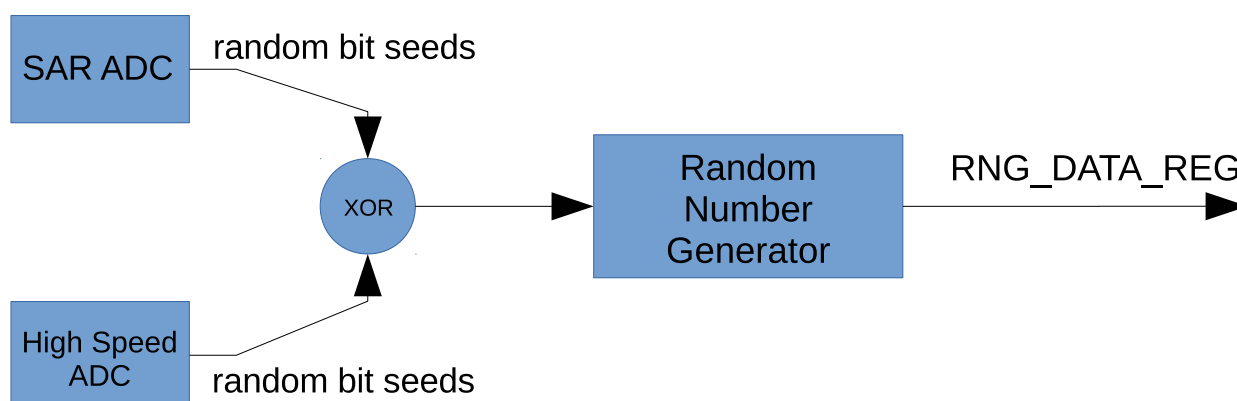


图 19-1. 噪声源

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 `RNG_DATA_REG` 寄存器时的速度不超过 5 MHz。

当 SAR ADC 打开时，每个 8 MHz 时钟周期内（来自内部 RC 振荡器，详情可见 [复位和时钟](#)），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 `RNG_DATA_REG` 寄存器时的速度不超过 500 kHz。

我们在仅高速 ADC 打开的状态下，以 5 MHz 的速率从 `RNG_DATA_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。

**说明：**

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。所以建议在 Wi-Fi 开启时，使用 SAR ADC 产生随机数。

未来，我们将提供一个生成随机数的系统 API，建议客户直接调用该 API 生成随机数。

## 19.4 基地址

用户可以通过两个不同的寄存器基地址访问随机数发生器，如表 88 所示。更多信息，请访问[章节 1 系统和存储器](#)。

表 88: 随机数发生器基地址

访问总线	基地址
PeriBUS1	0x3FF75000
PeriBUS2	0x60035000

## 19.5 寄存器列表

请注意，下表中的地址都是相对于随机数发生器基地址的地址偏移量（相对地址）。更多有关随机数发生器基地址的信息，请前往[19.4 章节](#)。

名称	描述	地址	访问
<a href="#">RNG_DATA_REG</a>	随机数数据	0x0110	只读

## 19.6 寄存器

请注意，这里的地址都是相对于随机数发生器基地址的地址偏移量（相对地址）。更多有关随机数发生器基地址的信息，请前往[19.4 章节](#)。

Register 19.1: RNG\_DATA\_REG (0x0110)

31	0
0x00000000	
Reset	

**RNG\_DATA** 随机数来源。（只读）



## 20. 片外存储器加密与解密

### 20.1 概述

ESP32-S2 芯片集成了片外存储器加密与解密模块，采用符合 [IEEE Std 1619-2007](#) 指定的 XTS-AES 标准的算法，为用户存放在片外存储器（flash 与片外 RAM）的应用代码和数据提供了安全保障。用户可以将专有软件、敏感的用户数据（如用来访问私有网络的凭据）存放在片外 flash 中，或将通用数据存放在片外 RAM 中。

### 20.2 主要特性

- 通用 XTS-AES 算法，符合 IEEE Std 1619-2007
- 手动加密过程需要软件参与
- 高速的自动加密过程，无需软件参与
- 高速的自动解密过程，无需软件参与
- 寄存器配置、eFuse 参数、启动 (boot) 模式共同决定加解密功能

### 20.3 功能描述

片外存储器加解密模块包含三个部分：手动加密 (Manual Encryption) 模块、自动加密 (Auto Encryption) 模块、自动解密 (Auto Decryption) 模块。结构图如图 20-1 所示。

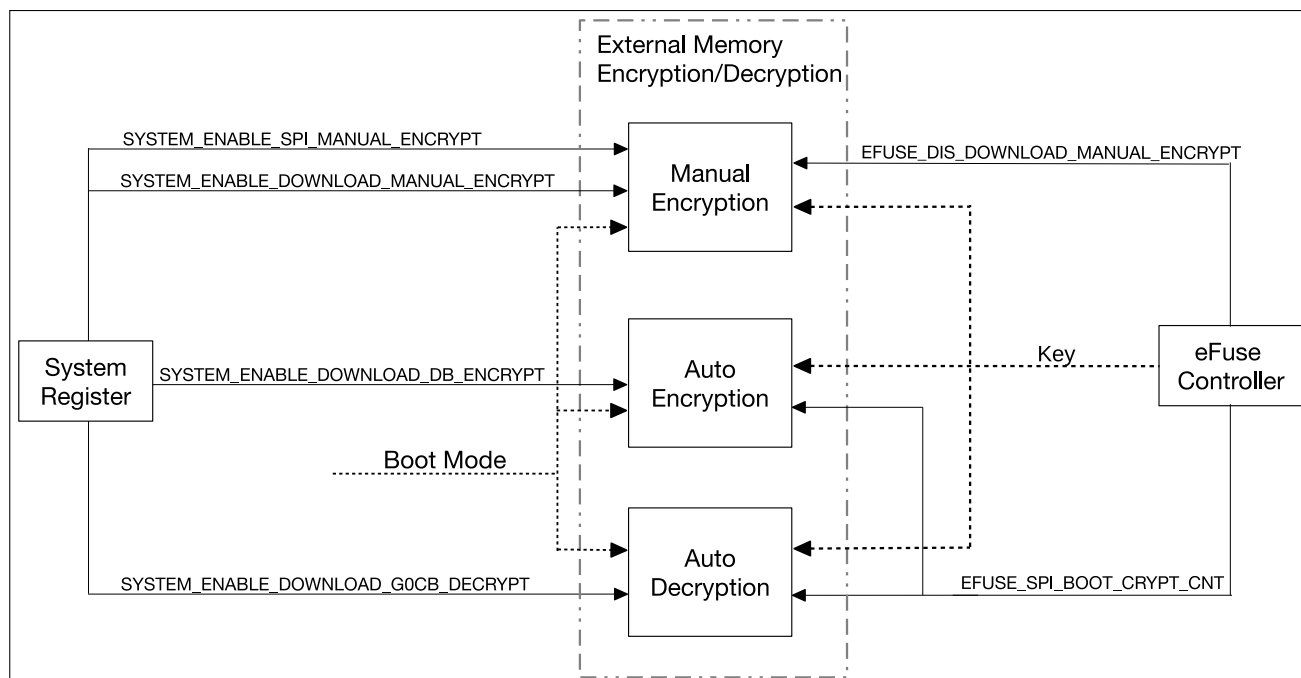


图 20-1. 片外存储器加解密模块架构

手动加密模块能够对指令/数据进行加密，指令/数据将以密文状态通过 SPI1 被写入片外 flash。

当 CPU 通过 cache 写片外 RAM 时，自动加密模块会先对数据自动进行加密，数据将以密文状态被写入片外 RAM。

当 CPU 通过 cache 读片外 flash 或片外 RAM 时，自动解密模块将对读取到的密文自动进行解密以恢复指令和数据。

外设 System 寄存器中与片外存储器加解密相关的是寄存器

SYSTEM\_EXTERNAL\_DEVICE\_ENCRYPT\_DECRYPT\_CONTROL\_REG 中的下面 4 个位：

- SYSTEM\_ENABLE\_DOWNLOAD\_MANUAL\_ENCRYPT
- SYSTEM\_ENABLE\_DOWNLOAD\_GOCB\_DECRYPT
- SYSTEM\_ENABLE\_DOWNLOAD\_DB\_ENCRYPT
- SYSTEM\_ENABLE\_SPI\_MANUAL\_ENCRYPT

片外存储器加解密模块还会从外设 eFuse 控制器中获取 2 个参数：

EFUSE\_DIS\_DOWNLOAD\_MANUAL\_ENCRYPT 和 EFUSE\_SPI\_BOOT\_CRYPT\_CNT。

### 20.3.1 XTS 算法

不论是手动加密，还是自动加/解密，使用的是同一种算法——XTS 算法。根据算法特征，在具体实现中，使用 1024 位为一个数据单元 (data unit)，此处“data unit”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 标准](#)。

### 20.3.2 密钥 Key

在执行 XTS 运算时，手动加密模块、自动加密模块和自动解密模块使用完全相同的密钥 Key。密钥 Key 来自硬件 eFuse，且无法被用户访问获取。

密钥 Key 支持两种长度：256 位、512 位。Key 值取决于 eFuse 中的 BLOCK4 ~ BLOCK9 的某个 BLOCK 或两个 BLOCK 的内容。为方便描述，现作如下约定：

- Block<sub>A</sub>：指 BLOCK4 ~ BLOCK9 中 key purpose（密钥用途）的值等于 EFUSE\_KEY\_PURPOSE\_XTS\_AES\_256\_KEY\_1 的 BLOCK。Block<sub>A</sub> 中存放 256 位的 Key<sub>A</sub>。
- Block<sub>B</sub>：指 BLOCK4 ~ BLOCK9 中密钥用途的值等于 EFUSE\_KEY\_PURPOSE\_XTS\_AES\_256\_KEY\_2 的 BLOCK。Block<sub>B</sub> 中存放 256 位的 Key<sub>B</sub>。
- Block<sub>C</sub>：指 BLOCK4 ~ BLOCK9 中密钥用途的值等于 EFUSE\_KEY\_PURPOSE\_XTS\_AES\_128\_KEY 的 BLOCK。Block<sub>C</sub> 中存放 256 位的 Key<sub>C</sub>。

根据 Block<sub>A</sub>、Block<sub>B</sub> 和 Block<sub>C</sub> 存在与否，不同的组合将产生不同的 Key 值，如表 90 所示。

表 90: Key 值映射表

Block <sub>A</sub>	Block <sub>B</sub>	Block <sub>C</sub>	Key 值	Key 长度 (位)
Yes	Yes	无关项	Key <sub>A</sub>   Key <sub>B</sub>	512
Yes	No	无关项	Key <sub>A</sub>   0 <sup>256</sup>	512
No	Yes	无关项	0 <sup>256</sup>   Key <sub>B</sub>	512
No	No	Yes	Key <sub>C</sub>	256

Block <sub>A</sub>	Block <sub>B</sub>	Block <sub>C</sub>	Key 值	Key 长度 (位)
No	No	No	0 <sup>256</sup>	256

注：表 90 中的 “Yes” 指存在，“No” 指不存在，“0<sup>256</sup>” 表示 256 个位“0” 组成的位串，“||” 表示将两个比特串按照前后顺序合成一个更长的新位串。

更多有关密钥用途的信息，请参考章节 14 eFuse 控制器。

### 20.3.3 目标空间

目标空间是指单次加密后的密文将要存放在片外存储器中的哪一段连续的地址空间中。目标空间可以由目标类型、目标尺寸、目标基地址这三个参数唯一确定。这三个参数的定义如下：

- 目标类型：目标空间的类型 (*type*)，片外 flash 或片外 RAM。目标类型的值为 0 时指 flash，值为 1 时指片外 RAM。
- 目标尺寸：目标空间的大小 (*size*)，以字节为单位，即单次对多少数据进行加密。只有 16、32 和 64 可选。
- 目标基地址：目标空间的基地址 (*base\_addr*)，这是一个物理地址，要求以目标尺寸为单位对齐，即  $base\_addr \% size == 0$ 。

如某一次加密操作，要将 16 字节的指令数据加密后存放在 flash 中的地址段 0x130 ~ 0x13F 中，则目标空间为 0x130 ~ 0x13F，目标类型为 0 (flash)，目标尺寸为 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/数据的加密，可以将整个加密过程拆分成多次进行，每次都有各自的目标空间参数。

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，目标空间等参数需要用户主动配置。

### 20.3.4 数据填充

对于自动加/解密模块，数据的填充由硬件自动完成。对于手动加密模块，数据的填充需要用户主动配置。手动加密模块中由 16 个寄存器 XTS\_AES\_PLAIN\_*n*\_REG (*n*: 0-15) 构成的寄存器堆专用于数据填充，一次可以存放最多 512 位明文指令/数据。

实际上，手动加密模块不在乎明文来自什么地方，只注重密文将要存放在什么地方。考虑到明文和密文之间呈严格的对应关系，为了更好地描述明文如何封装在寄存器堆中，现假设明文从一开始就放在目标空间中，并在加密完成后被密文替换。因此，接下来的描述不再出现“明文”这个概念，而用“目标空间”来代替。但请注意，在真正使用时，明文可以来自任何地方，但用户必须清晰知道明文如何封装在寄存器堆中。

**目标空间映射到寄存器堆的方法为：**

假设目标空间中某一 word 的存放地址为 *address*，记  $offset = address \% 64$ ， $n = \frac{offset}{4}$ ，那么该 word 将被存放在编号为 *n* 的寄存器 XTS\_AES\_PLAIN\_*n*\_REG 中。

例如，当目标尺寸为 64 时，寄存器堆中的所有寄存器都将被使用，目标空间中的地址与寄存器堆之间的填充映射关系如表 91 所示。

表 91: 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	<a href="#">XTS_AES_PLAIN_0_REG</a>	0x20	<a href="#">XTS_AES_PLAIN_8_REG</a>
0x04	<a href="#">XTS_AES_PLAIN_1_REG</a>	0x24	<a href="#">XTS_AES_PLAIN_9_REG</a>
0x08	<a href="#">XTS_AES_PLAIN_2_REG</a>	0x28	<a href="#">XTS_AES_PLAIN_10_REG</a>
0x0C	<a href="#">XTS_AES_PLAIN_3_REG</a>	0x2C	<a href="#">XTS_AES_PLAIN_11_REG</a>
0x10	<a href="#">XTS_AES_PLAIN_4_REG</a>	0x30	<a href="#">XTS_AES_PLAIN_12_REG</a>
0x14	<a href="#">XTS_AES_PLAIN_5_REG</a>	0x34	<a href="#">XTS_AES_PLAIN_13_REG</a>
0x18	<a href="#">XTS_AES_PLAIN_6_REG</a>	0x38	<a href="#">XTS_AES_PLAIN_14_REG</a>
0x1C	<a href="#">XTS_AES_PLAIN_7_REG</a>	0x3C	<a href="#">XTS_AES_PLAIN_15_REG</a>

### 20.3.5 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、外设 System 寄存器、eFuse 参数、boot 模式共同配置并使用这一模块。请注意，手动加密模块只能加密 flash。

此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](#) 的值设置为  $type = 0$ 。
- 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  $type$ 、 $base\_addr$ 、 $size$  的定义，请参考章节 20.3.3。

2. 用明文填充寄存器堆 [XTS\\_AES\\_PLAIN\\_n\\_REG](#) ( $n$ : 0-15)。请参考章节 20.3.4 获取相关信息。  
只需要根据需要填充寄存器，不需要使用的寄存器可以是任意值。
3. 轮询寄存器 [XTS\\_AES\\_STATE\\_REG](#) 直到读到 0，确保手动加密模块是空闲的。
4. 启动计算。对寄存器 [XTS\\_AES\\_TRIGGER\\_REG](#) 写入 1。
5. 等待加密完成。轮询寄存器 [XTS\\_AES\\_STATE\\_REG](#)，直到读到 2。  
步骤 1 至 5 操作手动加密模块对明文指令进行加密。加密算法使用的密钥就是  $Key$ 。
6. 下放密文访问权限给 SPI1。对寄存器 [XTS\\_AES\\_RELEASE\\_REG](#) 写入 1，使得加密结果允许被 SPI1 获取。  
之后如果读取寄存器 [XTS\\_AES\\_STATE\\_REG](#) 将读到 3。
7. 调用 SPI1，将加密结果写入片外 flash。
8. 销毁加密结果。对寄存器 [XTS\\_AES\\_DESTROY\\_REG](#) 写入 1。之后如果读取寄存器 [XTS\\_AES\\_STATE\\_REG](#) 将读到 0。

重复上述步骤，即可满足明文指令/数据的加密需求。

当且仅当手动加密模块拥有工作权限时，才允许手动加密。手动加密模块是否拥有工作权限取决于：

- SPI Boot 模式下

当寄存器 [SYSTEM\\_EXTERNAL\\_DEVICE\\_ENCRYPT\\_DECRYPT\\_CONTROL\\_REG](#) 的

`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 `SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` 的

`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：**

- 虽然 CPU 可以不通过 cache 而直接读片外存储器从而得到加密指令/数据，但软件还是绝对无法获取到密钥 *Key*。
- 手动加密模块通过调用 AES 加速器完成计算，在此期间禁止用户访问 AES。

### 20.3.6 自动加密模块

自动加密并非外设模块，自身不带寄存器，不能被 CPU 直接访问。外设 System 寄存器、eFuse 参数、boot 模式共同配置并使用这一模块。

**当且仅当自动加密模块拥有工作权限时，才允许自动加密。** 自动加密模块是否拥有工作权限取决于：

- SPI Boot 模式下

当参数 `SPI_BOOT_CRYPT_CNT`（3 位宽）中有奇数个位为 1 时，自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 `SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` 的

`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 通过 cache 写访问片外 RAM，自动加密模块将自动对数据进行加密，然后将加密结果写到片外 RAM。加密的整个过程无需软件参与并且对 cache 是透明的。加密算法使用的密钥 *Key* 同样无法被软件获取。
- 当自动加密模块没有工作权限时，自动加密模块将不理睬 CPU 对 cache 的访问请求，也不对数据做任何处理，因此数据将以明文状态被直接写到片外 RAM。

### 20.3.7 自动解密模块

自动解密并非外设模块，自身不带寄存器，不能被 CPU 直接访问。外设 System 寄存器、eFuse 参数、boot 模式共同配置并使用这一模块。

**当且仅当自动解密模块拥有工作权限时，才允许自动解密。** 自动解密模块是否拥有工作权限取决于：

- SPI Boot 模式下

当参数 `SPI_BOOT_CRYPT_CNT`（3 位宽）中有奇数个位为 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 `SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG` 的 `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：**

- 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并且对 cache 是透明的。解密算法使用的密钥 *Key* 同样无法被软件获取。
- 当自动解密模块没有工作权限时，自动解密模块不对片外存储器中的内容产生作用，无论是加密内容还是未加密内容，因此 CPU 通过 cache 读取到的是片外存储器中的原始内容。

## 20.4 基地址

用户可以通过两个不同的寄存器基地址访问手动加密模块，如表 92 所示。更多有关通过不同总线访问外设的信息，请参考章节 1 [系统和存储器](#)。

**表 92: 手动加密模块基地址**

访问外设总线	地址值
PeriBUS1	0x3F43A000
PeriBUS2	0x6003A000

## 20.5 寄存器列表

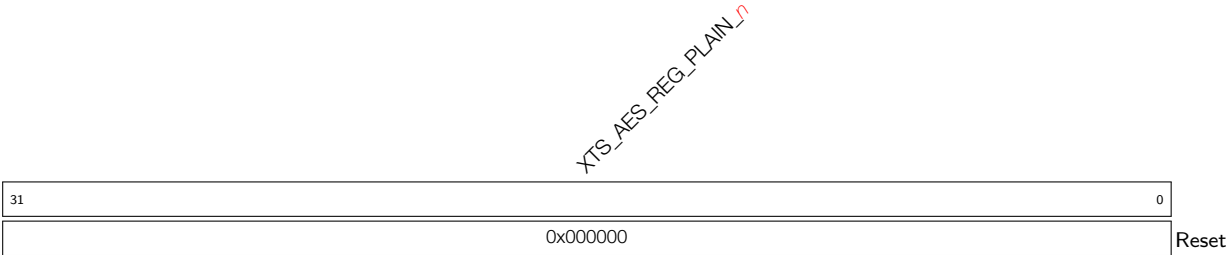
请注意，这里的地址是相对于手动加密模块基地址的地址偏移量（相对地址）。请参阅章节 20.4 获取有关手动加密模块的基地址的信息。

名称	描述	地址	访问
<b>明文寄存器堆</b>			
<code>XTS_AES_PLAIN_0_REG</code>	明文寄存器 0	0x0100	读/写
<code>XTS_AES_PLAIN_1_REG</code>	明文寄存器 1	0x0104	读/写
<code>XTS_AES_PLAIN_2_REG</code>	明文寄存器 2	0x0108	读/写
<code>XTS_AES_PLAIN_3_REG</code>	明文寄存器 3	0x010C	读/写
<code>XTS_AES_PLAIN_4_REG</code>	明文寄存器 4	0x0110	读/写
<code>XTS_AES_PLAIN_5_REG</code>	明文寄存器 5	0x0114	读/写
<code>XTS_AES_PLAIN_6_REG</code>	明文寄存器 6	0x0118	读/写
<code>XTS_AES_PLAIN_7_REG</code>	明文寄存器 7	0x011C	读/写
<code>XTS_AES_PLAIN_8_REG</code>	明文寄存器 8	0x0120	读/写
<code>XTS_AES_PLAIN_9_REG</code>	明文寄存器 9	0x0124	读/写
<code>XTS_AES_PLAIN_10_REG</code>	明文寄存器 10	0x0128	读/写
<code>XTS_AES_PLAIN_11_REG</code>	明文寄存器 11	0x012C	读/写
<code>XTS_AES_PLAIN_12_REG</code>	明文寄存器 12	0x0130	读/写

名称	描述	地址	访问
<a href="#">XTS_AES_PLAIN_13_REG</a>	明文寄存器 13	0x0134	读/写
<a href="#">XTS_AES_PLAIN_14_REG</a>	明文寄存器 14	0x0138	读/写
<a href="#">XTS_AES_PLAIN_15_REG</a>	明文寄存器 15	0x013C	读/写
配置寄存器			
<a href="#">XTS_AES_LINESIZE_REG</a>	加密块大小寄存器	0x0140	读/写
<a href="#">XTS_AES_DESTINATION_REG</a>	加密类型寄存器	0x0144	读/写
<a href="#">XTS_AES_PHYSICAL_ADDRESS_REG</a>	物理地址寄存器	0x0148	读/写
控制/状态寄存器			
<a href="#">XTS_AES_TRIGGER_REG</a>	启动运算	0x014C	只写
<a href="#">XTS_AES_RELEASE_REG</a>	释放控制	0x0150	只写
<a href="#">XTS_AES_DESTROY_REG</a>	销毁控制	0x0154	只写
<a href="#">XTS_AES_STATE_REG</a>	状态寄存器	0x0158	只读
版本寄存器			
<a href="#">XTS_AES_DATE_REG</a>	版本控制寄存器	0x015C	只读

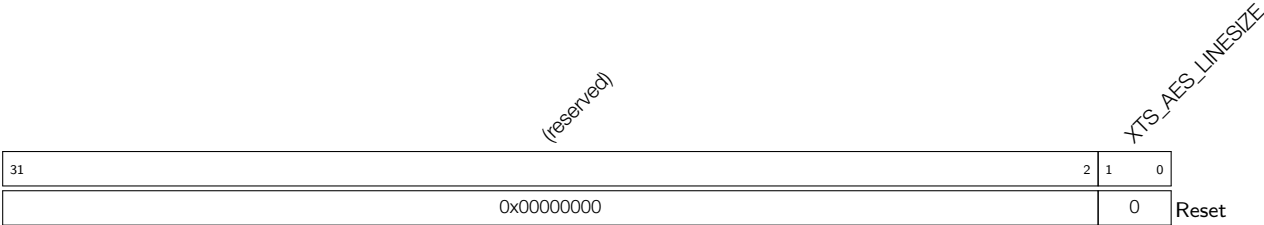
20.6 寄存器

Register 20.1: XTS\_AES\_PLAIN\_ *n* \_REG (*n*: 0-15) (0x0100+4\**n*)



XTS\_AES\_REG\_PLAIN\_ *n* 存储明文的第 *n* 个 32 位部分。(读/写)

Register 20.2: XTS\_AES\_LINESIZE\_REG (0x0140)

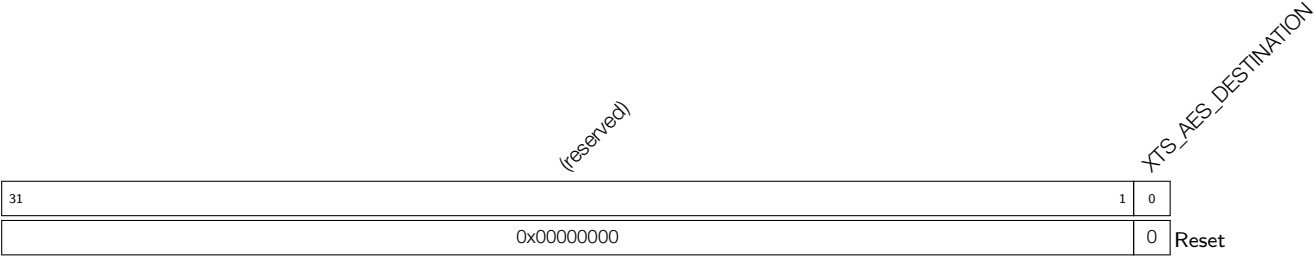


XTS\_AES\_LINESIZE 块大小寄存器，决定单次加密的数据量。(读/写)

- 0: 加密 128 位;
- 1: 加密 256 位;
- 2: 加密 512 位。

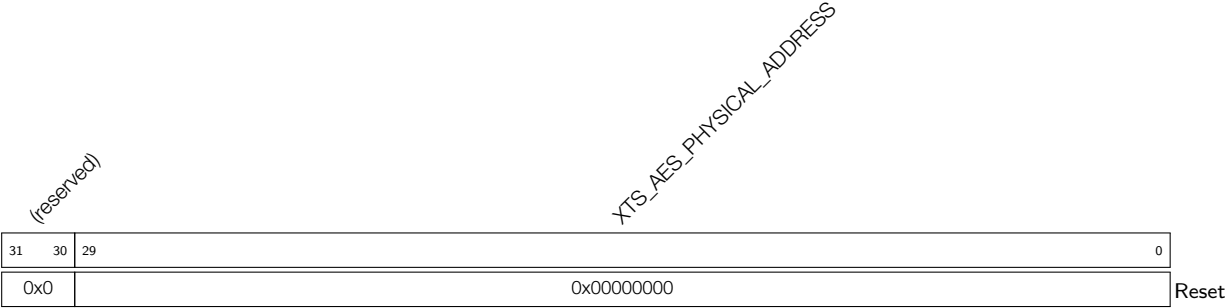


Register 20.3: XTS\_AES\_DESTINATION\_REG (0x0144)



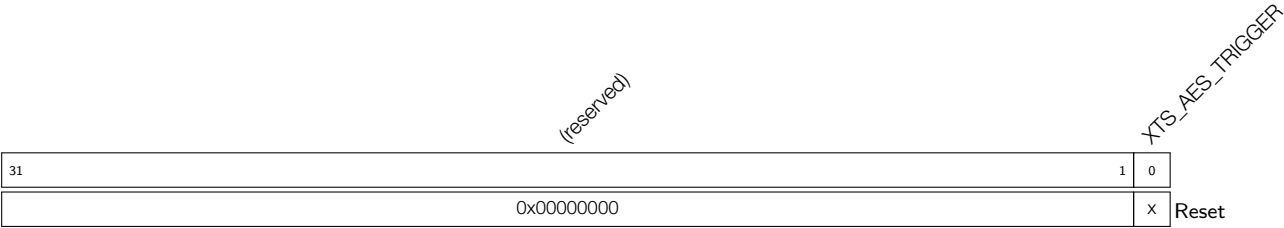
**XTS\_AES\_DESTINATION** 决定手动加密类型，目前只能手动加密 flash，所以只能为 0。用户不能写入 1，否则将发生错误。0：加密 flash；1：加密片外 RAM。（读/写）

Register 20.4: XTS\_AES\_PHYSICAL\_ADDRESS\_REG (0x0148)



**XTS\_AES\_PHYSICAL\_ADDRESS** 物理地址。（读/写）

Register 20.5: XTS\_AES\_TRIGGER\_REG (0x014C)



**XTS\_AES\_TRIGGER** 写入 1 使能手动加密运算。（只写）



Register 20.6: XTS\_AES\_RELEASE\_REG (0x0150)

(reserved)															XTS_AES_RELEASE	
31														1	0	
0x00000000														x	Reset	

**XTS\_AES\_RELEASE** 写入 1 使加密结果对 SPI1 可见，因而 SPI1 可以获取到加密结果。（只写）

Register 20.7: XTS\_AES\_DESTROY\_REG (0x0154)

(reserved)															XTS_AES_DESTROY	
31														1	0	
0x00000000														x	Reset	

**XTS\_AES\_DESTROY** 写入 1 销毁加密结果。（只写）

Register 20.8: XTS\_AES\_STATE\_REG (0x0158)

(reserved)															XTS_AES_STATE			
31															2	1	0	
0x00000000															0x0			Reset

**XTS\_AES\_STATE** 手动加密模块状态寄存器。（只读）

- 0x0 (XTS\_AES\_IDLE): 空闲；
- 0x1 (XTS\_AES\_BUSY): 忙于计算；
- 0x2 (XTS\_AES\_DONE): 计算完成，但手动加密结果数据对 SPI 不可见；
- 0x3 (XTS\_AES\_RELEASE): 手动加密结果对 SPI 可见。

Register 20.9: XTS\_AES\_DATE\_REG (0x015C)

(reserved)		XTS_AES_DATE	
31	30	29	0
0	0	0x20190416	
		Reset	

**XTS\_AES\_DATE** 版本控制寄存器。(只读)

## 21. 数字签名

### 21.1 概述

数字签名提供了一种使用私钥对消息进行加密、再使用公钥对消息进行验证的方法。数字签名可用于向服务器验证设备自身身份，或验证消息是否未被篡改。

ESP32-S2 包含一个数字签名 (DS) 模块，可高效生成 RSA 数字签名，而软件无法访问 RSA 私钥。

### 21.2 主要特性

- RSA 数字签名支持密钥长度最大为 4096 位
- 私钥数据已加密，并且只能由 DS 读取
- SHA-256 摘要用于保护私钥数据免遭攻击者篡改

### 21.3 功能描述

#### 21.3.1 概述

DS 模块计算 RSA 加密操作  $Z = X^Y \bmod M$ ，其中  $Z$  是签名， $X$  是输入消息， $Y$  和  $M$  是 RSA 私钥参数。

私钥参数以加密形式存储在 flash 或其他存储器中。这些参数使用一个密钥进行加密，该密钥只能由 DS 模块通过 HMAC 模块获取，并且，求解该密钥所需的一切输入信息只存放在 eFuse 中且只允许被 HMAC 模块访问。这意味着只有 DS 硬件才能解密私钥，软件绝对不会获取私钥明文。

需要签名时，软件直接将输入消息  $X$  发送到 DS 外设。加密操作之后，软件将读取签名结果  $Z$ 。

#### 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 4096 位）。相应的公钥也将另外生成和存储，可独立用于验证 DS 签名。

加密操作还需要两个运算符，即参数  $\bar{r}$  和  $M'$ 。这两个参数由  $Y$  和  $M$  得出，但需要通过软件提前运算得到。

运算符  $Y$ 、 $M$ 、 $\bar{r}$  和  $M'$  与验证摘要一起由用户加密并以密文  $C$  的形式存储。密文  $C$  输入到 DS 模块之后由硬件解密并使用密钥生成签名。具体的加密过程请参考章节 21.3.4。

DS 模块计算 RSA 加密操作  $Z = X^Y \bmod M$ ，更多信息请参考章节 18 RSA 加速器中第 18.3.1 节大数模幂运算。

### 21.3.3 约定

为方便描述，这里约定几个符号和函数，它们的作用域局限于本章。

- $1^s$  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。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， $[x]$  右边可以加上 0，使长度变成  $s$  位。例如： $[0x5]_4 = 0101$ ， $[0x5]_8 = 00000101$ ， $[0x5]_{16} = 0000010100000000$ ， $[0x13]_8 = 00010011$ ， $[0x13]_{16} = 0001001100000000$ 。
- $||$  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。

### 21.3.4 软件存储私钥

软件要存储用于 DS 加密操作的私钥，需要做以下准备工作：

- 按照章节 21.3.2 所述准备 RSA 私钥  $(Y, M)$  和运算符  $\bar{r}$  和  $M'$ 。
- 准备一个 256 位 HMAC 密钥 ( $[HMAC\_KEY]_{256}$ )，存储在 eFuse 中。HMAC 模块通过读取 HMAC 密钥来生成一个密钥，即  $DS\_KEY = \text{HMAC-SHA256}([HMAC\_KEY]_{256}, 1^{256})$ ，该密钥用于加解密 RSA 私钥。
- 准备密文  $C$  形式的加密密钥参数，长度为 1584 字节。

下图描述了软件层面的准备工作和硬件层面的工作流程。

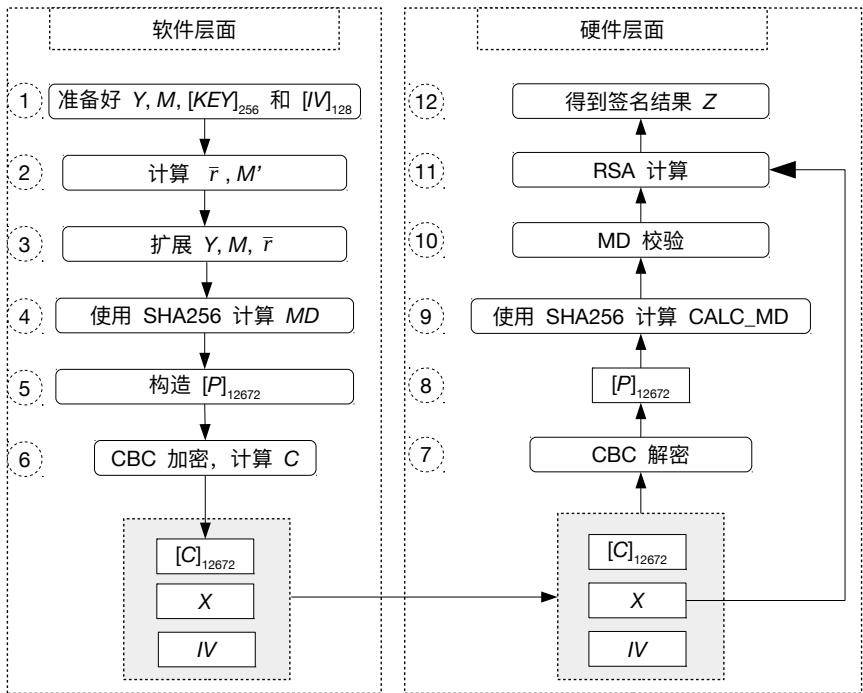


图 21-1. 软件准备工作与硬件工作流程

图 21-1 中左半边给出了计算  $C$  的过程。用户需要依照图 21-1 指定的步骤来计算  $C$ 。更加详细的过程描述如下：

- **步骤 1:** 准备好大数  $Y$  和  $M$ ，它们应符合运算子的长度要求。记  $[L]_{32} = \frac{N}{32}$ （比如，对于 RSA 4096， $[L]_{32} == [0x80]_{32}$ ）。另外，准备好  $[DS\_KEY]_{256}$  和一个随机的  $[IV]_{128}$ ，他们都要符合 AES-CBC 块加密算法的要求。有关 AES 更多信息，请参考章节 16 AES 加速器。
- **步骤 2:** 根据  $M$  求解  $\bar{r}$  和  $M'$ 。
- **步骤 3:** 扩展  $Y$ 、 $M$  和  $\bar{r}$ ，得到  $[Y]_{4096}$ 、 $[M]_{4096}$  和  $[\bar{r}]_{4096}$ 。由于  $Y$ 、 $M$  和  $\bar{r}$  的最大位宽为 4096，运算子位宽小于 4096 需要扩展为 4096，位宽等于 4096 则不需要扩展。
- **步骤 4:** 使用 SHA-256 计算 MD 校验码： $[MD]_{256} = \text{SHA256}([Y]_{4096} || [M]_{4096} || [\bar{r}]_{4096} || [M']_{32} || [L]_{32} || [IV]_{128})$
- **步骤 5:** 构造  $[P]_{12672} = ([Y]_{4096} || [M]_{4096} || [\bar{r}]_{4096} || [MD]_{256} || [M']_{32} || [L]_{32} || [\beta]_{64})$ ，其中  $[\beta]_{64}$  是符合 PKCS#7 封装方式的追加码，由 8 个值为 0x08 的字节组成的一个 64 位的位串  $[0x0808080808080808]_{64}$ ，目的在于使  $P$  的长度为 128 位的整数倍。
- **步骤 6:** 计算  $C = [C]_{12672} = \text{AES-CBC-ENC}([P]_{12672}, [DS\_KEY]_{256}, [IV]_{128})$ 。 $C$  以密文状态包含诸多信息，包括 RSA 运算子  $Y$ 、 $M$ 、 $\bar{r}$ 、 $M'$  和  $L$ ，用于校验的 MD 校验码和追加码  $[\beta]_{64}$  等其他信息。如前文 21.3.4 所述， $DS\_KEY$  由 eFuse 中的  $HMAC\_KEY$  得出。

### 21.3.5 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。输入信息是预先生成的私钥密文  $C$ 、唯一的消息  $X$ ，和  $IV$ 。

DS 模块的工作流程可以视为 21.3.4 准备工作中计算  $C$  的逆过程。可以分为如下三个阶段：

#### 1. 解析阶段，即图 21-1 中的步骤 7 和步骤 8

解析过程是图 21-1 中步骤 6 的逆过程。DS 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  $P = \text{AES-CBC-DEC}(C, DS\_KEY, IV)$ ，其中  $IV$  就是  $[IV]_{128}$ ，由用户直接指定； $[DS\_KEY]_{256}$  由硬件 HMAC 提供，由存储在 eFuse 中的  $HMAC\_KEY$  得到，软件无法获取。

显然，DS 模块能够通过  $P$  解析出  $[Y]_{4096}$ 、 $[M]_{4096}$ 、 $[\bar{r}]_{4096}$ 、 $[M']_{32}$ 、 $[L]_{32}$ 、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 21-1 中的步骤 9 和步骤 10

DS 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，因此填充校验不在图 21-1 中体现。

- MD 校验——DS 模块调用 SHA-256 进行哈希计算获取哈希结果值  $[CALC\_MD]_{256}$ （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——DS 模块将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。

如果 MD 校验通过，DS 模块将执行后续计算；否则 DS 模块拒绝执行。如果填充校验失败，将生成警告信息，但不会影响 DS 模块的后续操作。

#### 3. 计算阶段，即图 21-1 中的步骤 11 和步骤 12

DS 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。DS 模块调用 RSA 加速器完成大数模幂运算  $Z = X^Y \bmod M$ ， $Z$  为签名结果。

### 21.3.6 软件层面的 DS 操作

每次需要计算数字签名时，都应执行以下软件操作。输入消息是预先生成的私钥密文  $C$ 、唯一的消息  $X$ ，和  $IV$ 。这些软件步骤触发章节 21.3.5 中描述的硬件工作流程。

1. **启动 DS**：对寄存器 `DS_SET_START_REG` 写 1。
2. **检查  $DS\_KEY$  是否已经准备好**：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 未被调用。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 DS 模块没有拿到  $DS\_KEY$ ，原因可能是有其他程序的干扰。
3. **配置寄存器**：将  $IV$  block 写入寄存器 `DS_IV_m_REG` ( $m$ : 0-3)。有关  $IV$  block 的更多信息，请参考章节 16 AES 加速器。
  4. **将  $X$  写入存储器  $DS\_X\_MEM$** ：将  $X_i$  ( $i \in [0, n) \cap \mathbb{N}$ ) 写入存储器 `DS_X_MEM`，容量为 128 个字 (word)。每一个字刚好存放一个  $b$  进制数。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 128 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。
  5. **将  $C$  写入存储器  $DS\_C\_MEM$** ：将  $C_i$  ( $i \in [0, 396) \cap \mathbb{N}$ ) 写入存储器 `DS_C_MEM`，容量为 396 个字。每一个字刚好存放一个  $b$  进制数。
  6. **启动计算**：对寄存器 `DS_SET_ME_REG` 写入 1。
  7. **等待运算结束**：轮询寄存器 `DS_QUERY_BUSY_REG` 直到读到 0。
  8. **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。 $Z$  结果值全零无效，跳至步骤 10。
    - 如果返回值为 2，则说明填充校验通过失败，但 MD 校验通过，用户可以继续读取  $Z$  结果值。
    - 如果返回值为 3，则说明填充填充校验失败，且 MD 校验失败， $Z$  结果值全零无效，跳至步骤 10。
  9. **读出运算结果**：从存储器 `DS_Z_MEM` 读出运算结果  $Z_i$  ( $i \in \{0, 1, 2, \dots, n\}$ )。 $Z$  以小端字节序存储在存储器中。
  10. **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。

DS 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 基地址

用户可以通过两个不同的寄存器基地址访问 DS 模块，如表 94 所示。更多有关通过不同总线访问外设的信息，请参考章节 1 系统和存储器。

表 94: 基地址

访问外设总线	地址值
PeriBUS1	0x3F43D000
PeriBUS2	0x6003D000

## 21.5 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基地址的地址偏移量（相对地址）。请参阅表 94 查询 DS 模块的基地址。

名称	描述	大小（字节）	起始地址	结束地址	访问
DS_C_MEM	存储器 C	1584	0x0000	0x062F	只写
DS_X_MEM	存储器 X	512	0x0800	0x09FF	只写
DS_Z_MEM	存储器 Z	512	0x0A00	0x0BFF	只读

## 21.6 寄存器列表

请注意，这里的地址是相对于基地址的地址偏移量（相对地址）。请参阅表 94 查询 DS 模块的基地址。

名称	描述	地址	访问
<b>配置寄存器</b>			
DS_IV_0_REG	IV block 数据	0x0630	只写
DS_IV_1_REG	IV block 数据	0x0634	只写
DS_IV_2_REG	IV block 数据	0x0638	只写
DS_IV_3_REG	IV block 数据	0x063C	只写
<b>状态/控制寄存器</b>			
DS_SET_START_REG	启动 DS 模块	0x0E00	只写
DS_SET_ME_REG	开始计算	0x0E04	只写
DS_SET_FINISH_REG	结束计算	0x0E08	只写
DS_QUERY_BUSY_REG	DS 模块状态	0x0E0C	只读
DS_QUERY_KEY_WRONG_REG	查询 DS_KEY 未准备好的原因	0x0E10	只读
DS_QUERY_CHECK_REG	查询校验结果	0x0814	只读
<b>版本寄存器</b>			
DS_DATE_REG	版本控制寄存器	0x0820	读/写

21.7 寄存器

Register 21.1: DS\_IV\_m\_REG (m: 0-3) (0x0630+4\*m)

31	0
0x00000000	
Reset	

DS\_IV\_m\_REG (m: 0-3) IV block 数据。(只写)

Register 21.2: DS\_SET\_START\_REG (0x0E00)

(reserved)																															DS_SET_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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

DS\_SET\_START 写入 1 启动 DS 模块。(只写)

Register 21.3: DS\_SET\_ME\_REG (0x0E04)

(reserved)																															DS_SET_ME																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

DS\_SET\_ME 写入 1 以开始计算。(只写)

Register 21.4: DS\_SET\_FINISH\_REG (0x0E08)

(reserved)																															DS_SET_FINISH																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

DS\_SET\_FINISH 写入 1 以结束运算。(只写)



Register 21.5: DS\_QUERY\_BUSY\_REG (0x0E0C)

(reserved)																														DS_QUERY_BUSY	
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

**DS\_QUERY\_BUSY** 1: DS 模块正在忙; 0: DS 模块空闲。(只读)

Register 21.6: DS\_QUERY\_KEY\_WRONG\_REG (0x0E10)

(reserved)																												DS_QUERY_KEY_WRONG					
31																												4	3	0			
0 0																												0x0				Reset	

**DS\_QUERY\_KEY\_WRONG** 1-15: HMAC 被调用, 但 DS 模块未拿到 *DS\_KEY* (最大值为 15);  
0: HMAC 未被调用。(只读)

Register 21.7: DS\_QUERY\_CHECK\_REG (0x0E14)

(reserved)																															DS_PADDING_BAD DS_MD_ERROR																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																											
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**DS\_PADDING\_BAD** 1: 填充校验失败; 0: 填充校验通过。(只读)

**DS\_MD\_ERROR** 1: MD 校验失败; 0: MD 校验通过。(只读)

Register 21.8: DS\_DATE\_REG (0x0E20)

(reserved)			DS_DATE																											0		
31	30	29																												0		
0	0	0x20190418																												Reset		

**DS\_DATE** 版本控制寄存器 (读/写)。

# 修订历史

日期	版本	发布说明
2020.01	V0.2	增加以下章节： <ul style="list-style-type: none"><li>第 5 章 <i>IO MUX</i> 和 <i>GPIO</i> 交换矩阵</li><li>第 7 章 <i>DMA</i> 控制器</li><li>第 8 章 通用异步收发传输器 (<i>UART</i>)</li></ul> 更新 eFuse 烧写时序、读取时序参数配置。
2019.11	V0.1	预发布。