ESP32

技术参考手册

版本 5.1
乐鑫信息科技

www.espressif.com

关于本手册

《ESP32 技术参考手册》的目标读者群体是使用 ESP32 芯片的应用开发工程师。本手册提供了关于 ESP32 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 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 下载最新版本文档。

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 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 下载产品证书。

目录

1 系统和存储器 24

2 中断矩阵 (INTERRUPT) 33

3 复位和时钟 38

4 IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX) 46

5 DPort 寄存器 86

6 DMA 控制器 (DMA) 114

7 SPI 控制器 (SPI) 119

8 SDIO 从机控制器 149

11 I2C 控制器 (I2C) 268

17 脉冲计数器 (PCNT) 457

18 定时器组 (TIMG) 466

19 看门狗定时器 (WDT) 476

20 eFuse 控制器 (eFuse) 478

21 双线汽车接口 (TWAI) 499

30 超低功耗协处理器 (ULP) 610

表格

插图

1 系统和存储器

1.1 概述

ESP32 采用两个哈佛结构 Xtensa LX6 CPU 构成双核系统。所有的片上存储器、片外存储器以及外设都分布在两个 CPU 的数据总线和 / 或指令总线上。

除下文列出的个别情况外，两个 CPU 的地址映射呈对称结构，即使用相同的地址访问同一目标。系统中多个外设能够通过 DMA 访问片上存储器。

两个 CPU 的名称分别是 “PRO_CPU” 和 “APP_CPU”。PRO 代表 “protocol（协议）”，APP 代表 “application（应用）”。在大多数情况下，两个 CPU 的功能是相同的。

1.2 主要特性

- 地址空间
    - 对称地址映射
    - 数据总线与指令总线分别有 4 GB (32-bit) 地址空间
    - 1296 KB 片上存储器地址空间
    - 19704 KB 片外存储器地址空间
    - 512 KB 外设地址空间
    - 部分片上存储器与片外存储器既能被数据总线也能被指令总线访问
    - 328 KB DMA 地址空间
- 片上存储器
    - 448 KB Internal ROM
    - 520 KB Internal SRAM
    - 8 KB RTC FAST Memory
    - 8 KB RTC SLOW Memory
- 片外存储器
  片外 SPI 存储器可作为片外存储器被映射到可用的地址空间。部分片上存储器可用作片外存储器的 Cache。
    - 最大支持 16 MB 片外 SPI Flash
    - 最大支持 8 MB 片外 SPI SRAM
- 外设
    - 41 个外设模块
- DMA
    - 13 个具有 DMA 功能的模块

图 1-1 描述了系统结构。图 1-2 描述了地址映射结构。

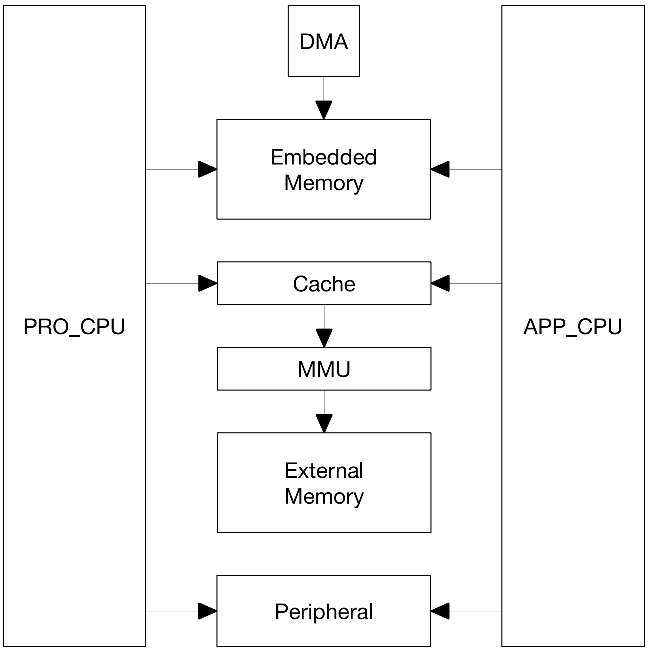


图 1-1. 系统结构

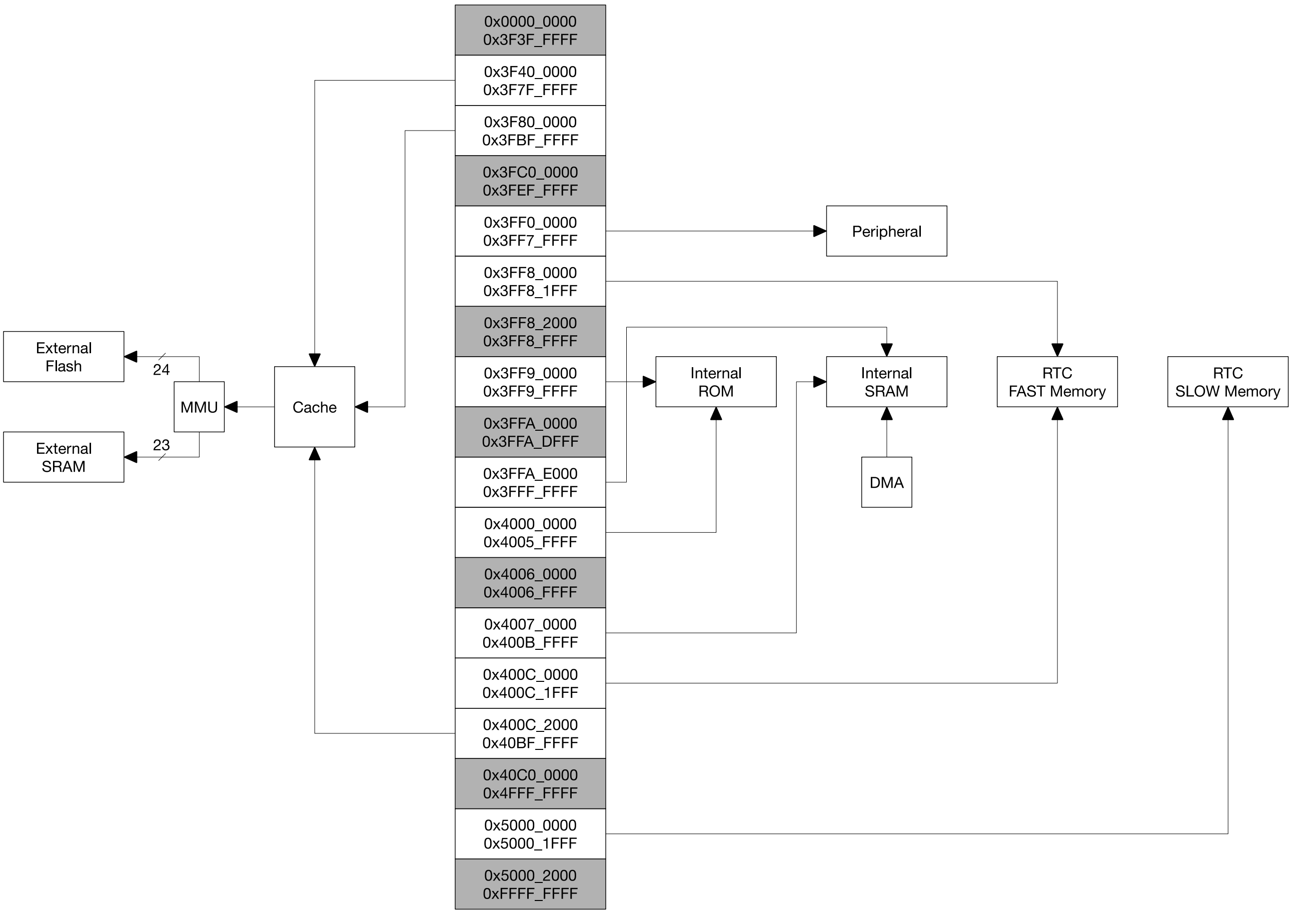


图 1-2. 地址映射结构

1.3 功能描述

1.3.1 地址映射

同构双核系统由两个哈佛结构 Xtensa LX6 CPU 构成，每个 CPU 都具有 4 GB（32-bit）的地址空间。两个 CPU 的地址映射是对称的。

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。即字节地址 0x0、0x1、0x2、0x3 访问的字节分别是 0x0 访问的 32-bit 字中的最低、次低、次高、最高字节。CPU 可以通过数据总线按照字节、半字、字进行对齐与非对齐的数据访问。CPU 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。

两个 CPU 都能够使用数据总线与指令总线直接访问片上存储器、使用 Cache 和 MMU 直接访问映射到地址空间的片外存储器、使用指令总线直接访问外设。当两个 CPU 访问同一目标时，其使用相同的地址，整个系统的地址映射呈对称结构。表 1-1 描述了两个 CPU 的数据总线与指令总线中的各段地址所能访问的目标。

系统中部分片上存储器与部分片外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，两个 CPU 都可以用多个地址访问到同一目标。

表 1-1. 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	0x3F3F_FFFF		保留
数据	0x3F40_0000	0x3F7F_FFFF	4 MB	片外存储器
数据	0x3F80_0000	0x3FBF_FFFF	4 MB	片外存储器
	0x3FC0_0000	0x3FEF_FFFF	3 MB	保留
数据	0x3FF0_0000	0x3FF7_FFFF	512 KB	外设
数据	0x3FF8_0000	0x3FFF_FFFF	512 KB	片上存储器
指令	0x4000_0000	0x400C_1FFF	776 KB	片上存储器
指令	0x400C_2000	0x40BF_FFFF	11512 KB	片外存储器
	0x40C0_0000	0x4FFF_FFFF	244 MB	保留
数据 / 指令	0x5000_0000	0x5000_1FFF	8 KB	片上存储器
	0x5000_2000	0xFFFF_FFFF		保留

1.3.2 片上存储器

片上存储器分为 Internal ROM、Internal SRAM、RTC FAST Memory、RTC SLOW Memory 四个部分，其容量分别为 448 KB、520 KB、8 KB、8 KB。其中 448 KB Internal ROM 分为 384 KB Internal ROM 0、64 KB Internal ROM 1 两部分；520 KB Internal SRAM 分为 192 KB Internal SRAM 0、128 KB Internal SRAM 1、200 KB Internal SRAM 2 三部分。

RTC FAST Memory 与 RTC SLOW Memory 都为 SRAM。

表 1-2 列出了所有片上存储器以及片上存储器的数据总线与指令总线地址段。

表 1-2. 片上寄存器地址映射

总线类型	边界地址		容量	目标	备注
	低位地址	高位地址			
数据	0x3FF8_0000	0x3FF8_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
	0x3FF8_2000	0x3FF8_FFFF	56 KB	保留	
数据	0x3FF9_0000	0x3FF9_FFFF	64 KB	Internal ROM 1	
	0x3FFA_0000	0x3FFA_DFFF	56 KB	保留	
数据	0x3FFA_E000	0x3FFD_FFFF	200 KB	Internal SRAM 2	DMA
数据	0x3FFE_0000	0x3FFF_FFFF	128 KB	Internal SRAM 1	DMA
总线类型	边界地址		容量	目标	备注
	低位地址	高位地址			
指令	0x4000_0000	0x4000_7FFF	32 KB	Internal ROM 0	Remap
指令	0x4000_8000	0x4005_FFFF	352 KB	Internal ROM 0	
	0x4006_0000	0x4006_FFFF	64 KB	保留	
指令	0x4007_0000	0x4007_FFFF	64 KB	Internal SRAM 0	Cache
指令	0x4008_0000	0x4009_FFFF	128 KB	Internal SRAM 0	
指令	0x400A_0000	0x400A_FFFF	64 KB	Internal SRAM 1	
指令	0x400B_0000	0x400B_7FFF	32 KB	Internal SRAM 1	Remap
指令	0x400B_8000	0x400B_FFFF	32 KB	Internal SRAM 1	
指令	0x400C_0000	0x400C_1FFF	8 KB	RTC FAST Memory	PRO_CPU Only
总线类型	边界地址		容量	目标	备注
	低位地址	高位地址			
数据指令	0x5000_0000	0x5000_1FFF	8 KB	RTC SLOW Memory	

1.3.2.1 Internal ROM 0

Internal ROM 0 的容量为 384 KB，可以被两个 CPU 通过指令总线 0x4000_0000 ~ 0x4005_FFFF 读取。

访问 ROM 0 的头 32 KB 的地址（0x4000_0000 ~ 0x4000_7FFF）可以被重新映射到 Internal SRAM 1 中的一部分，这部分原本被地址 0x400B_0000 ~ 0x400B_7FFF 访问。重映射时，这 32 KB SRAM 不能再被地址 0x400B_0000 ~ 0x400B_7FFF 访问，但是可以被数据总线 (0x3FFE_8000 ~ 0x3FFE_FFFF）访问。实现方式是分别为两个 CPU 配置一个寄存器，即为 PRO_CPU 置位 DPORT_PRO_BOOT_REMAP_CTRL_REG 寄存器的 bit 0 或者为 APP_CPU 置位 DPORT_APP_BOOT_REMAP_CTRL_REG 寄存器的 bit 0。

1.3.2.2 Internal ROM 1

Internal ROM 1 的容量为 64 KB，其可以被两个 CPU 通过数据总线 0x3FF9_0000 ~ 0x3FF9_FFFF 读取。

1.3.2.3 Internal SRAM 0

Internal SRAM 0 的容量为 192 KB，通过配置，硬件的头 64 KB 可以作为 Cache 来缓存片外存储器。不作为 Cache 使用时，头 64 KB 可以被两个 CPU 通过指令总线 0x4007_0000 ~ 0x4007_FFFF 读写，其余 128 KB 可以被两个 CPU 通过指令总线 0x4008_0000 ~ 0x4009_FFFF 读写。

1.3.2.4 Internal SRAM 1

Internal SRAM 1 的容量为 128 KB，其既可以被两个 CPU 通过数据总线 0x3FFE_0000 ~ 0x3FFF_FFFF 读写，也可以被两个 CPU 通过指令总线 0x400A_0000 ~ 0x400B_FFFF 读写。

指令总线地址和数据总线地址访问的 word 是逆序的。即地址：
0x3FFE_0000 与 0x400B_FFFC 访问到相同的 word
0x3FFE_0004 与 0x400B_FFF8 访问到相同的 word
0x3FFE_0008 与 0x400B_FFF4 访问到相同的 word
……
0x3FFF_FFF4 与 0x400A_0008 访问到相同的 word
0x3FFF_FFF8 与 0x400A_0004 访问到相同的 word
0x3FFF_FFFC 与 0x400A_0000 访问到相同的 word

CPU 的数据总线与指令总线都是小端序。因此地址空间访问每个 word 的字节顺序不是逆序的。即地址：
0x3FFE_0000 访问的字节等同于 0x400B_FFFC 访问的 word 中的最低字节
0x3FFE_0001 访问的字节等同于 0x400B_FFFC 访问的 word 中的次低字节
0x3FFE_0002 访问的字节等同于 0x400B_FFFC 访问的 word 中的次高字节
0x3FFE_0003 访问的字节等同于 0x400B_FFFC 访问的 word 中的最高字节
0x3FFE_0004 访问的字节等同于 0x400B_FFF8 访问的 word 中的最低字节
0x3FFE_0005 访问的字节等同于 0x400B_FFF8 访问的 word 中的次低字节
0x3FFE_0006 访问的字节等同于 0x400B_FFF8 访问的 word 中的次高字节
0x3FFE_0007 访问的字节等同于 0x400B_FFF8 访问的 word 中的最高字节
……
0x3FFF_FFF8 访问的字节等同于 0x400A_0004 访问的 word 中的最低字节
0x3FFF_FFF9 访问的字节等同于 0x400A_0004 访问的 word 中的次低字节
0x3FFF_FFFA 访问的字节等同于 0x400A_0004 访问的 word 中的次高字节
0x3FFF_FFFB 访问的字节等同于 0x400A_0004 访问的 word 中的最高字节
0x3FFF_FFFC 访问的字节等同于 0x400A_0000 访问的 word 中的最低字节
0x3FFF_FFFD 访问的字节等同于 0x400A_0000 访问的 word 中的次低字节
0x3FFF_FFFE 访问的字节等同于 0x400A_0000 访问的 word 中的次高字节
0x3FFF_FFFF 访问的字节等同于 0x400A_0000 访问的 word 中的最高字节

部分存储器可以被重新映射到 ROM 0 的地址空间。详细信息请参见 Internal Rom 0。

1.3.2.5 Internal SRAM 2

Internal SRAM 2 的容量为 200 KB，其可以被两个 CPU 通过数据总线 0x3FFA_E000 ~ 0x3FFD_FFFF 读写。

1.3.2.6 DMA

DMA 使用与 CPU 数据总线完全相同的地址读写 Internal SRAM 1 与 Internal SRAM 2。即 DMA 使用地址 0x3FFE_0000 ~ 0x3FFF_FFFF 读写 Internal SRAM 1，使用地址 0x3FFA_E000 ~ 0x3FFD_FFFF 读写 Internal SRAM 2。

系统中具有 DMA 功能的模块总共有 13 个。表 1-3 列出了所有具有 DMA 功能的模块。

表 1-3. 具有 DMA 功能的模块

UART0	UART1	UART2
SPI1	SPI2	SPI3
I2S0	I2S1	
SDIO Slave	SDMMC	
EMAC		
BT	WIFI	

1.3.2.7 RTC FAST Memory

RTC FAST Memory 为 8 KB SRAM，其只能被 PRO_CPU 通过数据总线 0x3FF8_0000 ~ 0x3FF8_1FFF 读写，或被 PRO_CPU 通过指令总线 0x400C_0000 ~ 0x400C_1FFF 读写。与其他存储器不同，APP_CPU 不能访问 RTC FAST Memory。

PRO_CPU 的这两段地址同序访问 RTC FAST Memory。即地址 0x3FF8_0000 与 0x400C_0000 访问到相同的 word，0x3FF8_0004 与 0x400C_0004 访问到相同的 word，0x3FF8_0008 与 0x400C_0008 访问到相同的 word，以此类推。APP_CPU 的这两段地址不能访问到 RTC FAST Memory，也不能访问到其他任何目标。

1.3.2.8 RTC SLOW Memory

RTC SLOW Memory 为 8 KB SRAM，其可以被两个 CPU 通过数据总线与指令总线共用地址段 0x5000_0000 ~ 0x5000_1FFF 读写。

1.3.3 片外存储器

ESP32 将 External Flash 与 External SRAM 作为片外存储器。表 1-4 列出了两个 CPU 的数据总线与指令总线中的各段地址通过 Cache 与 MMU 所能访问的片外存储器。两个 CPU 通过 Cache 与 MMU 对片外存储器进行访问时，Cache 将根据 MMU 中的设置把 CPU 的地址变换为 External Flash 与 External SRAM 的实地址。经过变换之后的实地址最大支持 16 MB 的 External Flash 与 8 MB 的 External SRAM。

表 1-4. 片外存储器地址映射

总线类型	边界地址		容量	目标	备注
	低位地址	高位地址			
数据	0x3F40_0000	0x3F7F_FFFF	4 MB	External Flash	读
数据	0x3F80_0000	0x3FBF_FFFF	4 MB	External SRAM	读 / 写
总线类型	边界地址		容量	目标	备注
	低位地址	高位地址			
指令	0x400C_2000	0x40BF_FFFF	11512 KB	External Flash	读

1.3.4 Cache

如下图 1-3 所示，ESP32 的 2 个 CPU 各有一组容量为 32 KB、块大小为 32 字节的 cache，用以访问外部存储器。PRO CPU 和 APP CPU 分别使用 DPORT_PRO_CACHE_CTRL_REG 的 PRO_CACHE_ENABLE 位和 DPORT_APP_CACHE_CTRL_REG 的 APP_CACHE_ENABLE 位使能 Cache 功能。

ESP32 Cache 采用两路组相连的映射方式。当只有 PRO CPU 使用 Cache 或只有 APP CPU 使用 Cache 时，可以通过配置寄存器 DPORT_CACHE_MUX_MODE_REG 的 CACHE_MUX_MODE[1:0] 位，选择使用 Internal

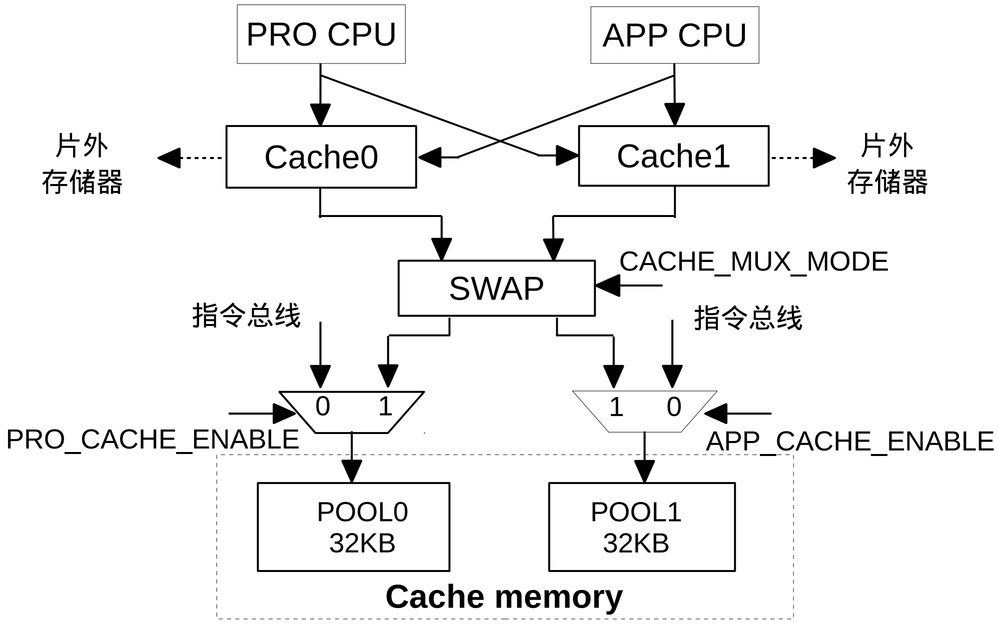


图 1-3. Cache 系统框图

SRAM0 的 POOL0 或 POOL1 作为 cache memory。当 PRO CPU 和 APP CPU 都使用 Cache 时，Internal SRAM0 的 POOL0 和 POOL1 可以复用作为 cache memory。详见表 1-5。

表 1-5. Cache memory 模式

CACHE_MUX_MODE	POOL0	POOL1
0	PRO CPU	APP CPU
1	PRO CPU/APP CPU	-
2	-	PRO CPU/APP CPU
3	APP CPU	PRO CPU

由表 1-5 可知，当 CACHE_MUX_MODE 为 1 或 2 时，PRO CPU 和 APP CPU 不可同时开启 Cache 功能。开启 Cache 功能后，POOL0 或者 POOL1 只作为 cache memory 使用，不能复用作为指令总线的访问区域。

ESP32 Cache 具有 Flush 功能。需要注意的是，当使用 Flush 功能时，写入 cache 的数据将被丢弃，并不会写回到 External SRAM 中。实现 flush 操作的方法为：先将 DPORT_*x*_CACHE_CTRL_REG 的 *x*_CACHE_FLUSH_ENA 位清 0，再将该位置 1。此后，系统硬件会将寄存器中的 *x*_CACHE_FLUSH_DONE 位置为 1 时，表明 cache flush 操作已经完成，其中 *x* 表示 “PRO” 或 “APP”。

ESP32 Cache 的地址映射详见片上存储器章节和片外存储器章节。

1.3.5 外设

ESP32 共有 41 个外设模块。表 1-6 详细描述了两个 CPU 的数据总线中的各段地址所能访问的各个外设模块。除了 PID Controller 以外，其余外设模块都可以被两个 CPU 用相同地址访问到。

表 1-6. 外设地址映射

总线类型	总线类型		容量	目标	备注
	低位地址	高位地址			
数据	0x3FF0_0000	0x3FF0_0FFF	4 KB	DPort Register	
数据	0x3FF0_1000	0x3FF0_1FFF	4 KB	AES Accelerator	
数据	0x3FF0_2000	0x3FF0_2FFF	4 KB	RSA Accelerator	
数据	0x3FF0_3000	0x3FF0_3FFF	4 KB	SHA Accelerator	
数据	0x3FF0_4000	0x3FF0_4FFF	4 KB	Secure Boot	
	0x3FF0_5000	0x3FF0_FFFF	44 KB	保留	
数据	0x3FF1_0000	0x3FF1_3FFF	16 KB	Cache MMU Table	
	0x3FF1_4000	0x3FF1_EFFF	44 KB	保留	
数据	0x3FF1_F000	0x3FF1_FFFF	4 KB	PID Controller	每个 CPU 单独外设
	0x3FF2_0000	0x3FF3_FFFF	128 KB	保留	
数据	0x3FF4_0000	0x3FF4_0FFF	4 KB	UART0	
	0x3FF4_1000	0x3FF4_1FFF	4 KB	保留	
数据	0x3FF4_2000	0x3FF4_2FFF	4 KB	SPI1	
数据	0x3FF4_3000	0x3FF4_3FFF	4 KB	SPI0	
数据	0x3FF4_4000	0x3FF4_4FFF	4 KB	GPIO	
	0x3FF4_5000	0x3FF4_7FFF	12 KB	保留	
数据	0x3FF4_8000	0x3FF4_8FFF	4 KB	RTC	
数据	0x3FF4_9000	0x3FF4_9FFF	4 KB	IO MUX	
	0x3FF4_A000	0x3FF4_AFFF	4 KB	保留	
数据	0x3FF4_B000	0x3FF4_BFFF	4 KB	SDIO Slave	三个部分之一
数据	0x3FF4_C000	0x3FF4_CFFF	4 KB	UDMA1	
	0x3FF4_D000	0x3FF4_EFFF	8 KB	保留	
数据	0x3FF4_F000	0x3FF4_FFFF	4 KB	I2S0	
数据	0x3FF5_0000	0x3FF5_0FFF	4 KB	UART1	
	0x3FF5_1000	0x3FF5_2FFF	8 KB	保留	
数据	0x3FF5_3000	0x3FF5_3FFF	4 KB	I2C0	
数据	0x3FF5_4000	0x3FF5_4FFF	4 KB	UDMA0	
数据	0x3FF5_5000	0x3FF5_5FFF	4 KB	SDIO Slave	三个部分之一
数据	0x3FF5_6000	0x3FF5_6FFF	4 KB	RMT	
数据	0x3FF5_7000	0x3FF5_7FFF	4 KB	PCNT	
数据	0x3FF5_8000	0x3FF5_8FFF	4 KB	SDIO Slave	三个部分之一
数据	0x3FF5_9000	0x3FF5_9FFF	4 KB	LED PWM	
数据	0x3FF5_A000	0x3FF5_AFFF	4 KB	eFuse Controller	
数据	0x3FF5_B000	0x3FF5_BFFF	4 KB	Flash Encryption	
	0x3FF5_C000	0x3FF5_DFFF	8 KB	保留	
数据	0x3FF5_E000	0x3FF5_EFFF	4 KB	MCPWM0	
数据	0x3FF5_F000	0x3FF5_FFFF	4 KB	TIMG0	
数据	0x3FF6_0000	0x3FF6_0FFF	4 KB	TIMG1	
	0x3FF6_1000	0x3FF6_3FFF	12 KB	保留	
数据	0x3FF6_4000	0x3FF6_4FFF	4 KB	SPI2	
数据	0x3FF6_5000	0x3FF6_5FFF	4 KB	SPI3	

总线类型	总线类型		容量	目标	备注
	低位地址	高位地址			
数据	0x3FF6_6000	0x3FF6_6FFF	4 KB	SYSCON	
数据	0x3FF6_7000	0x3FF6_7FFF	4 KB	I2C1	
数据	0x3FF6_8000	0x3FF6_8FFF	4 KB	SDMMC	
数据	0x3FF6_9000	0x3FF6_AFFF	8 KB	EMAC	
数据	0x3FF6_B000	0x3FF6_BFFF	4KB	TWAI	
数据	0x3FF6_C000	0x3FF6_CFFF	4 KB	MCPWM1	
数据	0x3FF6_D000	0x3FF6_DFFF	4 KB	I2S1	
数据	0x3FF6_E000	0x3FF6_EFFF	4 KB	UART2	
数据	0x3FF6_F000	0x3FF6_FFFF	4 KB	保留	
数据	0x3FF7_0000	0x3FF7_0FFF	4 KB	保留	
	0x3FF7_1000	0x3FF7_4FFF	16 KB	保留	
数据	0x3FF7_5000	0x3FF7_5FFF	4 KB	RNG	
	0x3FF7_6000	0x3FF7_FFFF	40 KB	保留	

注意：

- 使用 0x3FF40000 ~ 0x3FF7FFFF 地址空间 (DPORT) 访问的外设，CPU 也可以使用 0x60000000 ~ 0x6003FFFF 地址空间 (AHB) 访问。即使用 (0x3FF40000 + n) 地址和使用 (0x60000000 + n) 地址访问的内容是相同的，其中 n = 0 ~ 0x3FFFF。
- 相比于通过 AHB 访问外设，CPU 通过 DPORT 访问外设效率更高。但是 DPORT 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。另外，DPORT 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。此外使用 AHB 总线读 FIFO 会出现无法预知的错误。因此使用时请严格遵循《ESP32 勘误表及解决办法》中第 3.3、3.10、3.16 和 3.17 章节的描述。

1.3.5.1 不对称 PID Controller 外设

系统中有两个 PID Controller 分别服务于 PRO_CPU 和 APP_CPU。**PRO_CPU 和 APP_CPU 都只能访问自己的 PID Controller，不能访问对方的 PID Controller**。两个 CPU 都使用数据总线 0x3FF1_F000 ~ 3FF1_FFFF 访问自己的 PID Controller。

1.3.5.2 不连续外设地址范围

外设模块 SDIO Slave 被划分为三部分。两个 CPU 访问这三部分的地址是不连续的。这三部分分别被两个 CPU 的数据总线 0x3FF4_B000 ~ 3FF4_BFFF、0x3FF5_5000 ~ 3FF5_5FFF、0x3FF5_8000 ~ 3FF5_8FFF 访问。和其他外设一样，SDIO Slave 能被两个 CPU 访问。

1.3.5.3 存储器速度

ROM 和 SRAM 的时钟源都是 CPU_CLK，CPU 可在单个时钟周期内访问这两个存储器。由于 RTC FAST Memory 的时钟源是 APB_CLOCK，RTC SLOW Memory 的时钟源是 FAST_CLOCK，所以 CPU 访问这两个存储器的速度稍慢。DMA 在 APB_CLK 时钟下访问存储器。

SRAM 每 32K 为一个块。只要同时访问的是不同的块，那么 CPU 和 DMA 可以同时以最快速度访问 SRAM。

2 中断矩阵 (INTERRUPT)

2.1 概述

ESP32 中断矩阵将任一外部中断源单独分配到每个 CPU 的任一外部中断上。这提供了强大的灵活性，能适应不同的应用需求。

2.2 主要特性

- 接受 71 个外部中断源作为输入
- 为两个 CPU 分别生成 26 个外部中断（总共 52 个）作为输出
- 屏蔽 CPU 的 NMI 类型中断
- 查询外部中断源当前的中断状态

中断矩阵的结构如图 2-1 所示。

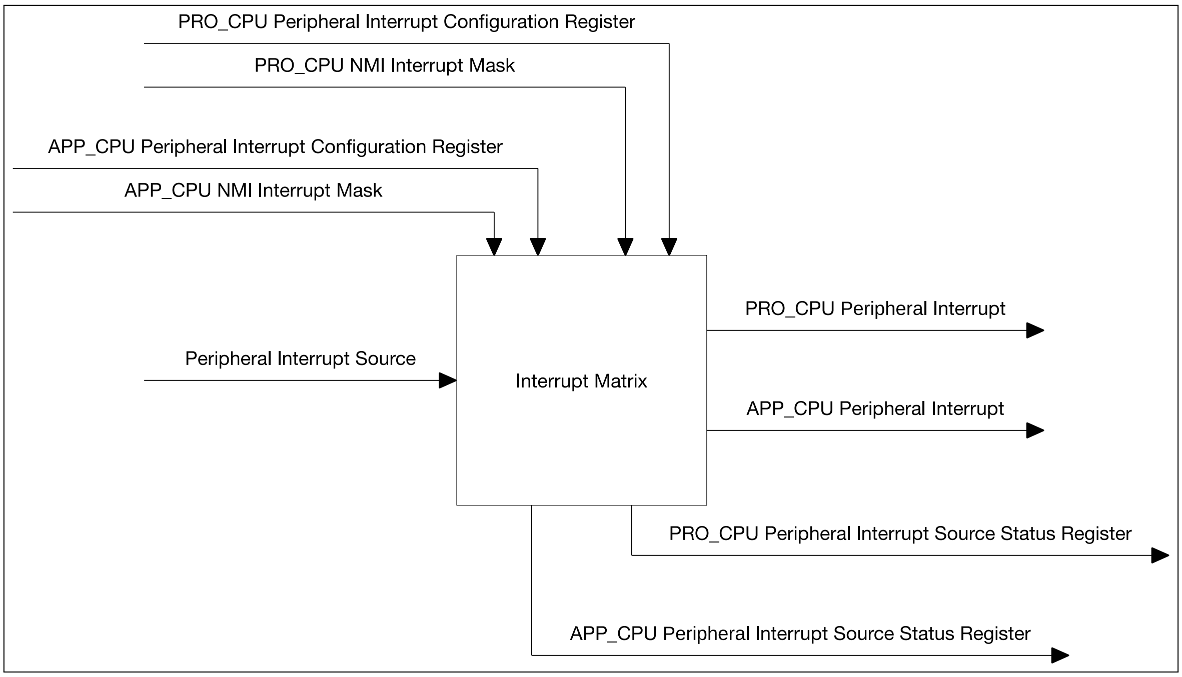


图 2-1. 中断矩阵结构图

2.3 功能描述

2.3.1 外部中断源

ESP32 总共有 71 个外部中断源。表 2-1 列出了所有外部中断源。ESP32 中的 71 个外部中断源中有 67 个可以分配给两个 CPU。其余的 4 个外部中断源只能分配给特定的 CPU，每个 CPU 2 个。GPIO_INTERRUPT_PRO 和 GPIO_INTERRUPT_PRO_NMI 只可以分配给 PRO_CPU，GPIO_INTERRUPT_APP 和 GPIO_INTERRUPT_APP_NMI 只可以分配给 APP_CPU。因此，PRO_CPU 与 APP_CPU 各可以分配到 69 个外部中断源。

表 2-1. PRO_CPU、APP_CPU 外部中断配置寄存器、外部中断源中断状态寄存器、外部中断源

PRO_CPU						APP_CPU		
Peripheral Interrupt Configuration Register	Status Register		Peripheral Interrupt Source			Status Register		Peripheral Interrupt Configuration Register
	Bit	Name	No.	Name	No.	Name	Bit	
DPORT_PRO_MAC_INTR_MAP_REG	0	DPORT_PRO_INTR_STATUS_REG_0_REG	0	MAC_INTR	0	DPORT_APP_INTR_STATUS_REG_0_REG	0	DPORT_APP_MAC_INTR_MAP_REG
DPORT_PRO_MAC_NMI_MAP_REG	1		1	MAC_NMI	1		1	DPORT_APP_MAC_NMI_MAP_REG
DPORT_PRO_BB_INT_MAP_REG	2		2	BB_INT	2		2	DPORT_APP_BB_INT_MAP_REG
DPORT_PRO_BT_MAC_INT_MAP_REG	3		3	BT_MAC_INT	3		3	DPORT_APP_BT_MAC_INT_MAP_REG
DPORT_PRO_BT_BB_INT_MAP_REG	4		4	BT_BB_INT	4		4	DPORT_APP_BT_BB_INT_MAP_REG
DPORT_PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5		5	DPORT_APP_BT_BB_NMI_MAP_REG
DPORT_PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6		6	DPORT_APP_RWBT_IRQ_MAP_REG
DPORT_PRO_RWBLE_IRQ_MAP_REG	7		7	RWBLE_IRQ	7		7	DPORT_APP_RWBLE_IRQ_MAP_REG
DPORT_PRO_RWBT_NMI_MAP_REG	8		8	RWBT_NMI	8		8	DPORT_APP_RWBT_NMI_MAP_REG
DPORT_PRO_RWBLE_NMI_MAP_REG	9		9	RWBLE_NMI	9		9	DPORT_APP_RWBLE_NMI_MAP_REG
DPORT_PRO_SLC0_INTR_MAP_REG	10		10	SLC0_INTR	10		10	DPORT_APP_SLC0_INTR_MAP_REG
DPORT_PRO_SLC1_INTR_MAP_REG	11		11	SLC1_INTR	11		11	DPORT_APP_SLC1_INTR_MAP_REG
DPORT_PRO_UHCI0_INTR_MAP_REG	12		12	UHCI0_INTR	12		12	DPORT_APP_UHCI0_INTR_MAP_REG
DPORT_PRO_UHCI1_INTR_MAP_REG	13		13	UHCI1_INTR	13		13	DPORT_APP_UHCI1_INTR_MAP_REG
DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG	14		14	TG_T0_LEVEL_INT	14		14	DPORT_APP_TG_T0_LEVEL_INT_MAP_REG
DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG	15		15	TG_T1_LEVEL_INT	15		15	DPORT_APP_TG_T1_LEVEL_INT_MAP_REG
DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG	16		16	TG_WDT_LEVEL_INT	16		16	DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG
DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG	17		17	TG_LACT_LEVEL_INT	17		17	DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG
DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG	18		18	TG1_T0_LEVEL_INT	18		18	DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG
DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG	19		19	TG1_T1_LEVEL_INT	19		19	DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG
DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	20		20	TG1_WDT_LEVEL_INT	20		20	DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG
DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	21		21	TG1_LACT_LEVEL_INT	21		21	DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG
DPORT_PRO_GPIO_INTERRUPT_MAP_REG	22		22	**GPIO_INTERRUPT_PRO** **GPIO_INTERRUPT_APP**	22		22	DPORT_APP_GPIO_INTERRUPT_MAP_REG
DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG	23		23	**GPIO_INTERRUPT_PRO_NMI** **GPIO_INTERRUPT_APP_NMI**	23		23	DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG
DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	24		24	CPU_INTR_FROM_CPU_0	24		24	DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG
DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	25		25	CPU_INTR_FROM_CPU_1	25		25	DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG
DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	26		26	CPU_INTR_FROM_CPU_2	26		26	DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG
DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	27		27	CPU_INTR_FROM_CPU_3	27		27	DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG
DPORT_PRO_SPI_INTR_0_MAP_REG	28		28	SPI_INTR_0	28		28	DPORT_APP_SPI_INTR_0_MAP_REG
DPORT_PRO_SPI_INTR_1_MAP_REG	29		29	SPI_INTR_1	29		29	DPORT_APP_SPI_INTR_1_MAP_REG
DPORT_PRO_SPI_INTR_2_MAP_REG	30		30	SPI_INTR_2	30		30	DPORT_APP_SPI_INTR_2_MAP_REG
DPORT_PRO_SPI_INTR_3_MAP_REG	31		31	SPI_INTR_3	31		31	DPORT_APP_SPI_INTR_3_MAP_REG
DPORT_PRO_I2S0_INT_MAP_REG	0	DPORT_PRO_INTR_STATUS_REG_1_REG	32	I2S0_INT	32	DPORT_APP_INTR_STATUS_REG_1_REG	0	DPORT_APP_I2S0_INT_MAP_REG
DPORT_PRO_I2S1_INT_MAP_REG	1		33	I2S1_INT	33		1	DPORT_APP_I2S1_INT_MAP_REG
DPORT_PRO_UART_INTR_MAP_REG	2		34	UART_INTR	34		2	DPORT_APP_UART_INTR_MAP_REG
DPORT_PRO_UART1_INTR_MAP_REG	3		35	UART1_INTR	35		3	DPORT_APP_UART1_INTR_MAP_REG
DPORT_PRO_UART2_INTR_MAP_REG	4		36	UART2_INTR	36		4	DPORT_APP_UART2_INTR_MAP_REG
DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG	5		37	SDIO_HOST_INTERRUPT	37		5	DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG
DPORT_PRO_EMAC_INT_MAP_REG	6		38	EMAC_INT	38		6	DPORT_APP_EMAC_INT_MAP_REG
DPORT_PRO_PWM0_INTR_MAP_REG	7		39	PWM0_INTR	39		7	DPORT_APP_PWM0_INTR_MAP_REG
DPORT_PRO_PWM1_INTR_MAP_REG	8		40	PWM1_INTR	40		8	DPORT_APP_PWM1_INTR_MAP_REG
Reserved	9		41	Reserved	41		9	Reserved
Reserved	10		42	Reserved	42		10	Reserved
DPORT_PRO_LEDC_INT_MAP_REG	11		43	LEDC_INT	43		11	DPORT_APP_LEDC_INT_MAP_REG
DPORT_PRO_EFUSE_INT_MAP_REG	12		44	EFUSE_INT	44		12	DPORT_APP_EFUSE_INT_MAP_REG
DPORT_PRO_TWAI_INT_MAP_REG	13		45	TWAI_INT	45		13	DPORT_APP_TWAI_INT_MAP_REG
DPORT_PRO_RTC_CORE_INTR_MAP_REG	14		46	RTC_CORE_INTR	46		14	DPORT_APP_RTC_CORE_INTR_MAP_REG
DPORT_PRO_RMT_INTR_MAP_REG	15		47	RMT_INTR	47		15	DPORT_APP_RMT_INTR_MAP_REG
DPORT_PRO_PCNT_INTR_MAP_REG	16		48	PCNT_INTR	48		16	DPORT_APP_PCNT_INTR_MAP_REG
DPORT_PRO_I2C_EXT0_INTR_MAP_REG	17		49	I2C_EXT0_INTR	49		17	DPORT_APP_I2C_EXT0_INTR_MAP_REG
DPORT_PRO_I2C_EXT1_INTR_MAP_REG	18		50	I2C_EXT1_INTR	50		18	DPORT_APP_I2C_EXT1_INTR_MAP_REG
DPORT_PRO_RSA_INTR_MAP_REG	19		51	RSA_INTR	51		19	DPORT_APP_RSA_INTR_MAP_REG
DPORT_PRO_SPI1_DMA_INT_MAP_REG	20		52	SPI1_DMA_INT	52		20	DPORT_APP_SPI1_DMA_INT_MAP_REG

PRO_CPU					APP_CPU			
Peripheral Interrupt Configuration Register	Peripheral Interrupt Source							Peripheral Interrupt Configuration Register
	Status Register		No.	Name	No.	Status Register		
	Bit	Name				Name	Bit	
DPORT_PRO_SPI2_DMA_INT_MAP_REG	21	DPORT_PRO_INTR_STATUS_REG_1_REG	53	SPI2_DMA_INT	53	DPORT_APP_INTR_STATUS_REG_1_REG	21	DPORT_APP_SPI2_DMA_INT_MAP_REG
DPORT_PRO_SPI3_DMA_INT_MAP_REG	22		54	SPI3_DMA_INT	54		22	DPORT_APP_SPI3_DMA_INT_MAP_REG
DPORT_PRO_WDG_INT_MAP_REG	23		55	WDG_INT	55		23	DPORT_APP_WDG_INT_MAP_REG
DPORT_PRO_TIMER_INT1_MAP_REG	24		56	TIMER_INT1	56		24	DPORT_APP_TIMER_INT1_MAP_REG
DPORT_PRO_TIMER_INT2_MAP_REG	25		57	TIMER_INT2	57		25	DPORT_APP_TIMER_INT2_MAP_REG
DPORT_PRO_TG_T0_EDGE_INT_MAP_REG	26		58	TG_T0_EDGE_INT	58		26	DPORT_APP_TG_T0_EDGE_INT_MAP_REG
DPORT_PRO_TG_T1_EDGE_INT_MAP_REG	27		59	TG_T1_EDGE_INT	59		27	DPORT_APP_TG_T1_EDGE_INT_MAP_REG
DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG	28		60	TG_WDT_EDGE_INT	60		28	DPORT_APP_TG_WDT_EDGE_INT_MAP_REG
DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG	29		61	TG_LACT_EDGE_INT	61		29	DPORT_APP_TG_LACT_EDGE_INT_MAP_REG
DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG	30		62	TG1_T0_EDGE_INT	62		30	DPORT_APP_TG1_T0_EDGE_INT_MAP_REG
DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG	31		63	TG1_T1_EDGE_INT	63		31	DPORT_APP_TG1_T1_EDGE_INT_MAP_REG
DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG	0	DPORT_PRO_INTR_STATUS_REG_2_REG	64	TG1_WDT_EDGE_INT	64	DPORT_APP_INTR_STATUS_REG_2_REG	0	DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG
DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG	1		65	TG1_LACT_EDGE_INT	65		1	DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG
DPORT_PRO_MMU_IA_INT_MAP_REG	2		66	MMU_IA_INT	66		2	DPORT_APP_MMU_IA_INT_MAP_REG
DPORT_PRO_MPU_IA_INT_MAP_REG	3		67	MPU_IA_INT	67		3	DPORT_APP_MPU_IA_INT_MAP_REG
DPORT_PRO_CACHE_IA_INT_MAP_REG	4		68	CACHE_IA_INT	68		4	DPORT_APP_CACHE_IA_INT_MAP_REG

2.3.2 CPU 中断

两个 CPU（PRO_CPU 和 APP_CPU）各有 32 个中断，其中 26 个为外部中断。表 2-2 列出了每个 CPU 所有的中断。

表 2-2. 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
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

2.3.3 分配外部中断源至 CPU 外部中断

在本小节中：

- 记号 Source_X 代表某个外部中断源。
- 记号 PRO_X_MAP_REG（或 APP_X_MAP_REG）表示 PRO_CPU（或 APP_CPU）的某个外部中断配置

寄存器，且此外部中断配置寄存器与外部中断源 Source_X 相对应。即表 2-1 中 “PRO_CPU（APP_CPU）- Peripheral Interrupt Configuration Register” 一列中与 “Peripheral Interrupt Source - Name” 一列中的某个外部中断源处于同一行的寄存器。

- 记号 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。
- 记号 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。

借助以上术语，可以这样描述中断矩阵控制器操作：

- **将外部中断源 Source_X 分配到 CPU（PRO_CPU 或 APP_CPU）**
  将寄存器 PRO_X_MAP_REG（APP_X_MAP_REG）配成 Num_P。Num_P 可以取任意 CPU 外部中断值。CPU 中断可以被多个外设共享（见下文）。
- **关闭 CPU（PRO_CPU 或 APP_CPU）外部中断源 Source_X**
  将寄存器 PRO_X_MAP_REG（APP_X_MAP_REG）配成任意 Num_I。由于任何被配成 Num_I 的中断都没有连接到 2 个 CPU 上，选择特定内部中断值不会造成影响。
- **将多个外部中断源 Source_X*n* ORed 分配到 PRO_CPU（APP_CPU）的外部中断**
  将各个寄存器 PRO_X*n*_MAP_REG (APP_X*n*_MAP_REG) 都配成同样的 Num_P。这些外设中断都会触发 CPU Interrupt_P。

2.3.4 屏蔽 CPU 的 NMI 类型中断

中断矩阵能够根据信号 PRO_CPU NMI Interrupt Mask（或 APP_CPU NMI Interrupt Mask）暂时屏蔽所有被分配到 PRO_CPU（或 APP_CPU）的外部中断源的 NMI 中断。信号 PRO_CPU NMI Interrupt Mask 和 APP_CPU NMI Interrupt Mask 分别来自外设 PID Controller。

2.3.5 查询外部中断源当前的中断状态

读寄存器 PRO_INTR_STATUS_REG_*n*（APP_INTR_STATUS_REG_*n*）中的特定 Bit 值就可以获知外部中断源当前的中断状态。寄存器 PRO_INTR_STATUS_REG_*n*（APP_INTR_STATUS_REG_*n*）与外部中断源的对应关系如表 2-1 所示。

3 复位和时钟

3.1 System 复位

3.1.1 概述

系统提供三种级别的复位方式，分别是 CPU 复位，内核复位，系统复位。

所有的复位都不会影响 MEM 中的数据。图 3-1 展示了整个子系统的结构以及每种复位方式：

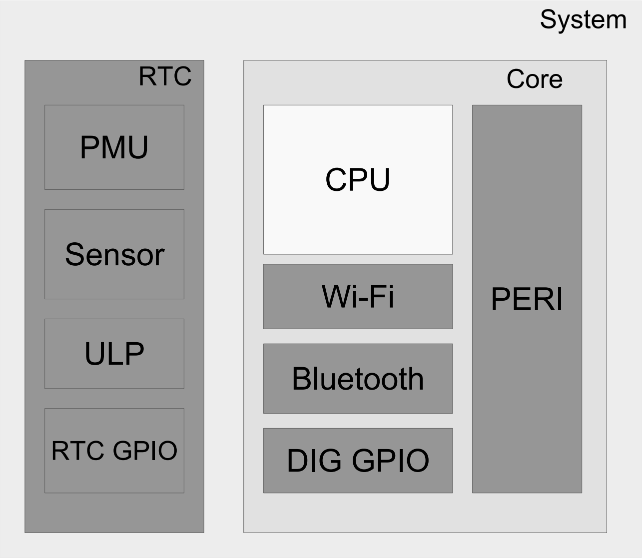


图 3-1. 系统复位

- CPU 复位：只复位 CPU 的所有寄存器。
- 内核复位：除了 RTC，会把整个 digital 的寄存器全部复位，包括 CPU、所有外设和数字 GPIO。
- 系统复位：会复位整个芯片所有的寄存器，包括 RTC。

3.1.2 复位源

大多数情况下，APP_CPU 和 PRO_CPU 将被立刻复位，有些复位源只能复位其中一个。APP_CPU 和 PRO_CPU 的复位原因也各自不同：当系统复位起来之后，PRO_CPU 可以通过读取寄存器 RTC_CNTL_RESET_CAUSE_PROCPU 来获取复位源，APP_CPU 则可以通过读取寄存器 RTC_CNTL_RESET_CAUSE_APPCPU 来获取复位源。

表 3-1 列出了从这些寄存器中可能读出的复位源。

表 3-1. PRO_CPU 和 APP_CPU 复位源

PRO	APP	源	复位方式	注释
0x01	0x01	芯片上电复位	系统复位	-
0x10	0x10	RWDT 系统复位	系统复位	详见 WDT 章节
0x0F	0x0F	欠压复位	系统复位	详见 Power Management 章节
0x03	0x03	软件系统复位	内核复位	配置 RTC_CNTL_SW_SYS_RST 寄存器
0x05	0x05	Deep Sleep Reset	内核复位	详见 Power Management 章节
0x07	0x07	MWDT0 全局复位	内核复位	详见 WDT 章节

PRO	APP	源	复位方式	注释
0x08	0x08	MWDT1 全局复位	内核复位	详见 WDT 章节
0x09	0x09	RWDT 内核复位	内核复位	详见 WDT 章节
0x0B	-	MWDT0 CPU 复位	CPU 复位	详见 WDT 章节
0x0C	-	软件 CPU 复位	CPU 复位	配置 RTC_CNTL_SW_APPCPU_RST 寄存器
-	0x0B	MWDT1 CPU 复位	CPU 复位	详见 WDT 章节
-	0x0C	软件 CPU 复位	CPU 复位	配置 RTC_CNTL_SW_APPCPU_RST 寄存器
0x0D	0x0D	RWDT CPU 复位	CPU 复位	详见 WDT 章节
-	0xE	PRO CPU 复位	CPU 复位	表明 PRO CPU 能够通过配置 DPORT_APPCPU_RESETTING 寄存器单独复位 APP CPU

3.2 系统时钟

3.2.1 概述

ESP32 提供了多种不同频率的时钟选择，可以灵活的配置 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 3-2 为系统时钟结构。

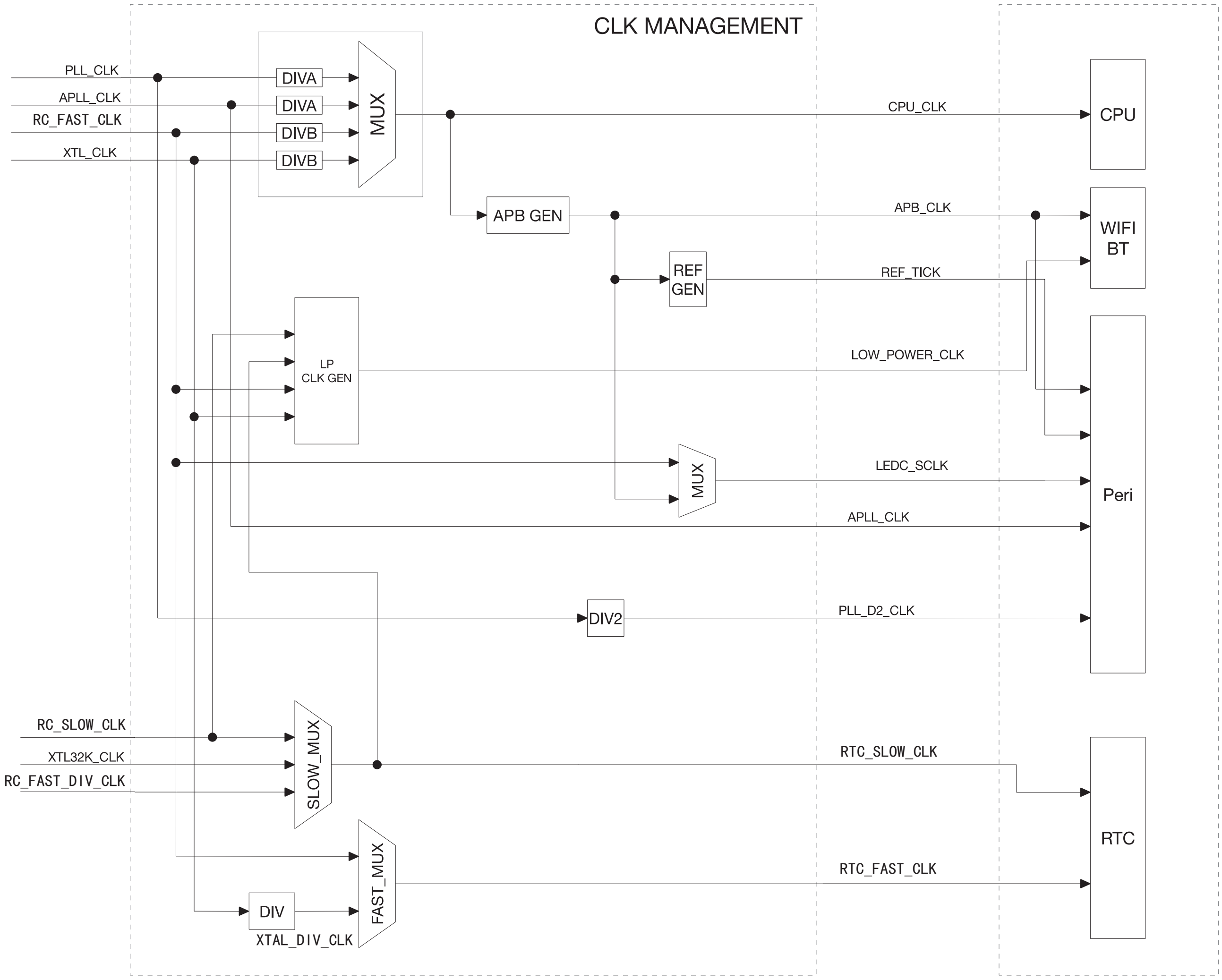


图 3-2. 系统时钟

3.2.2 时钟源

ESP32 的时钟源分别来自外部晶振、内部 PLL 或振荡电路。具体地说，这些时钟源为：

- 快速时钟
  - PLL_CLK，320 MHz 或 480 MHz 内部 PLL 时钟
  - XTL_CLK，2 ~ 40 MHz 外部晶振时钟
- 低功耗慢速时钟
  - XTL32K_CLK，32 KHz 外部晶振时钟
  - RC_FAST_CLK，8 MHz 内部时钟，频率可调
  - RC_FAST_DIV_CLK 由 RC_FAST_CLK 经 256 分频所得，频率为 (RC_FAST_CLK/256)。当 RC_FAST_CLK 的初始频率为 8 MHz 时，该时钟以 31.250 KHz 的频率运行。
  - RC_SLOW_CLK，150 KHz 内部低功耗时钟，频率可调
- 音频时钟
  - APLL_CLK，16 ~ 128 MHz 内部 Audio PLL 时钟

3.2.3 CPU 时钟

如图 3-2 所示，CPU_CLK 为 CPU 主时钟，它在高效工作模式下，主频可以达到 240 MHz。同时，CPU 能够在超低频下工作，以减少功耗。

CPU_CLK 由 RTC_CNTL_SOC_CLK_SEL 来选择时钟源，允许选择 PLL_CLK，APLL_CLK，RC_FAST_CLK，XTL_CLK 作为 CPU_CLK 的时钟源。具体请参考表 3-2 和表 3-3。

表 3-2. CPU_CLK 源

RTC_CNTL_SOC_CLK_SEL 值	时钟源
0	XTL_CLK
1	PLL_CLK
2	RC_FAST_CLK
3	APLL_CLK

表 3-3. CPU_CLK 源

时钟源	*SEL_0	*SEL_1	CPU 时钟
XTL_CLK	0	-	CPU_CLK = XTL_CLK / (SYSCON_PRE_DIV_CNT+1)
PLL_CLK (320 MHz)	1	0	CPU_CLK = PLL_CLK / 4 CPU_CLK 频率为 80 MHz
PLL_CLK (320 MHz)	1	1	CPU_CLK = PLL_CLK / 2 CPU_CLK 频率为 160 MHz
PLL_CLK (480 MHz)	1	2	CPU_CLK = PLL_CLK / 2 CPU_CLK 频率为 240 MHz
RC_FAST_CLK	2	-	CPU_CLK = RC_FAST_CLK / (SYSCON_PRE_DIV_CNT+1)
APLL_CLK	3	0	CPU_CLK = APLL_CLK / 4
APLL_CLK	3	1	CPU_CLK = APLL_CLK / 2

*SEL_0：寄存器 RTC_CNTL_SOC_CLK_SEL 的值
*SEL_1：寄存器 CPU_CPUPERIOD_SEL 的值

3.2.4 外设时钟

外设所需要的时钟包括 APB_CLK，REF_TICK，LEDC_SCLK，APLL_CLK 和 PLL_F160M_CLK。
下表 3-4 为接入各个外设的时钟。

表 3-4. 外设时钟用法

外设	APB_CLK	REF_TICK	LEDC_SCLK	APLL_CLK	PLL_F160M_CLK
EMAC	Y	N	N	Y	N
TIMG	Y	N	N	N	N
I2S	Y	N	N	Y	Y
UART	Y	Y	N	N	N
RMT	Y	Y	N	N	N
LED PWM	Y	Y	Y	N	N
PWM	Y	N	N	N	Y
I2C	Y	N	N	N	N
SPI	Y	N	N	N	N
PCNT	Y	N	N	N	N
eFuse Controller	Y	N	N	N	N
SDIO Slave	Y	N	N	N	N
SDMMC	Y	N	N	N	N

3.2.4.1 APB_CLK

APB_CLK 时钟频率由 CPU_CLK 源决定，具体见表 3-5：

表 3-5. APB_CLK

CPU_CLK 源	APB_CLK 时钟频率
PLL_CLK	80 MHz
APLL_CLK	CPU_CLK / 2
XTL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

3.2.4.2 REF_TICK

REF_TICK 时钟频率由 APB_CLK 分频产生，APB_CLK 时钟频率由 CPU_CLK 源决定。REF_TICK 的时钟频率应固定，因此当切换 CPU_CLK 源时，应配置分频寄存器，使其频率固定。分频寄存器如何配置见表 3-6：

表 3-6. REF_TICK

CPU_CLK 源	APB_CLK 时钟频率	REF_TICK 频率
PLL_CLK	80 MHz	APB_CLK / (SYSCON_PLL_TICK_NUM+1)
APLL_CLK	CPU_CLK / 2	APB_CLK / (SYSCON_APLL_TICK_NUM+1)
XTL_CLK	CPU_CLK	APB_CLK / (SYSCON_XTAL_TICK_NUM+1)
RC_FAST_CLK	CPU_CLK	APB_CLK / (SYSCON_CK8M_TICK_NUM+1)

例如，当 REF_TICK 时钟频率固定为 1 MHz 时，如果 CPU_CLK 源为 PLL_CLK，则 REF_TICK 频率 = 80 MHz / (SYSCON_PLL_TICK_NUM+1) = 1 MHz，那么 SYSCON_PLL_TICK_NUM 应配置为 79 (0x4F)。

3.2.4.3 LEDC_SCLK 源

LEDC_SCLK 时钟源由寄存器 LEDC_APB_CLK_SEL 决定，如表 3-7 所示。

表 3-7. LEDC_SCLK 源

LEDC_APB_CLK_SEL 值	LEDC_SCLK 源
0	RC_FAST_CLK
1	APB_CLK

3.2.4.4 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。

3.2.4.5 PLL_F160M_CLK 源

PLL_F160M_CLK 是 PLL_CLK 根据当前 PLL 的频率自动调节分频系数所得，频率时钟为 160 MHz。

3.2.4.6 时钟源注意事项

大多数外设一般在选择 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 3-4。

LED PWM 模块能将 RC_FAST_CLK 作为时钟源使用，即在 APB_CLK 关闭的时候，LED PWM 也可工作。换而言之，当系统处于低功耗模式时（参考章节 低功耗管理），所有正常外设都将停止工作（APB_CLK 关闭），但是 LED PWM 仍然可以通过 RC_FAST_CLK 来正常工作。

3.2.5 Wi-Fi BT 时钟

Wi-Fi 和 BT 必须在 APB_CLK 时钟源选择 PLL_CLK 下才能工作。只有当 Wi-Fi 和 BT 同时进入低功耗模式时，才能暂时关闭 PLL_CLK。

LOW_POWER_CLK 允许选择 RC_SLOW_CLK、RTC_SLOW_CLK、RC_FAST_CLK 或 XTL_CLK，用于 Wi-Fi 和 BT 的低功耗模式。

3.2.6 RTC 时钟

RTC_SLOW_CLK 和 RTC_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。

RTC_SLOW_CLK 允许选择 RC_SLOW_CLK，XTL32K_CLK 或 RC_FAST_DIV_CLK，用于驱动 Power Management 模块。

RTC_FAST_CLK 允许选择 XTL_CLK 的分频时钟或 RC_FAST_CLK，用于驱动 On-chip Sensor 模块。

3.2.7 音频 PLL

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。

为了通过集成的精密时钟源来最大限度地降低系统成本，ESP32 集成了音频 PLL。Audio PLL 公式如下：

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

其中，

- f_{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

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

注意：sdm1 和 sdm0 在 ESP32 的 revision0 版本中不支持。更多关于 ESP32 版本的信息，可在《ESP32 Bug 描述及解决方法》中查看。

Audio PLL 可通寄存器 RTC_CNTL_PLLA_FORCE_PU 强行打开，或者通过寄存器 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 的时候，PLL 会跟随系统状态，当系统进入睡眠模式的时候自动关闭，系统被唤醒的时候自动打开。

3.3 寄存器列表

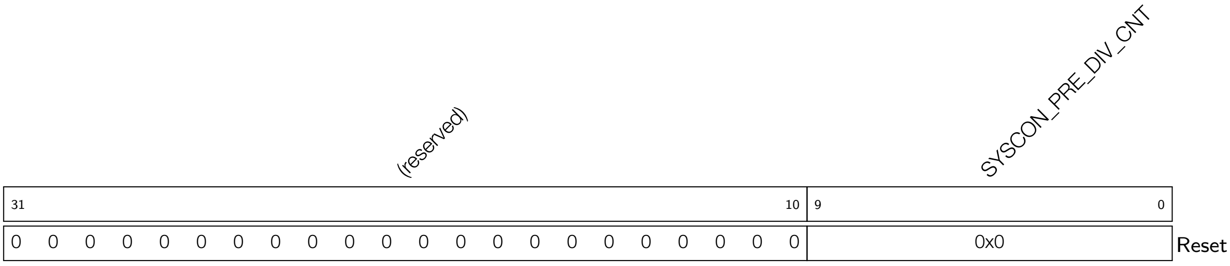
本小节的所有地址均为相对于 SYSCON 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。

名称	描述	地址	访问
配置寄存器			
SYSCON_SYSCLK_CONF_REG	配置系统时钟频率	0x0000	R/W
SYSCON_XTAL_TICK_CONF_REG	配置 REF_TICK 的分频系数	0x0004	R/W
SYSCON_PLL_TICK_CONF_REG	配置 REF_TICK 的分频系数	0x0008	R/W
SYSCON_CK8M_TICK_CONF_REG	配置 REF_TICK 的分频系数	0x000C	R/W
SYSCON_APLL_TICK_CONF_REG	配置 REF_TICK 的分频系数	0x003C	R/W
芯片版本寄存器			
SYSCON_DATE_REG	芯片版本寄存器	0x007C	R/W

3.4 寄存器

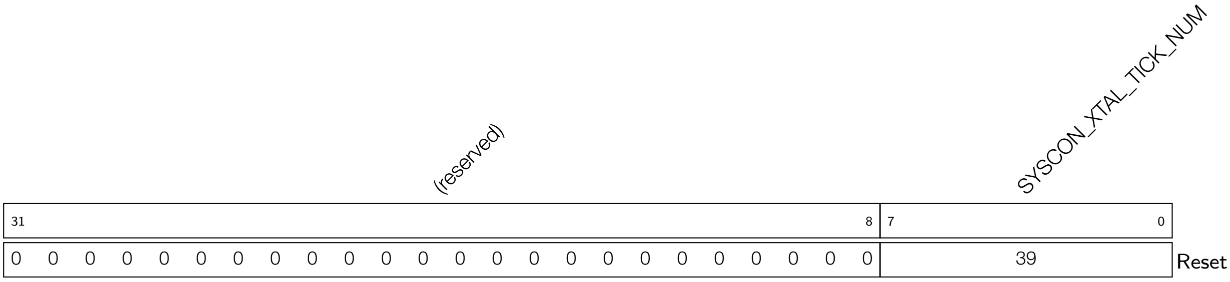
本小节的所有地址均为相对于 SYSCON 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。

Register 3.1. SYSCON_SYSCLK_CONF_REG (0x0000)



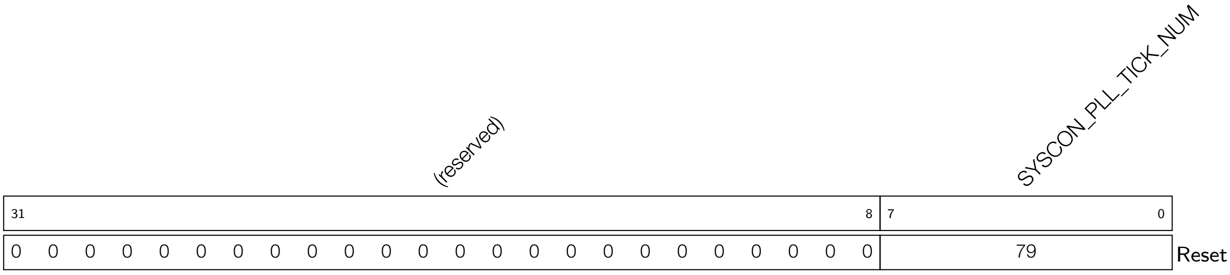
SYSCON_PRE_DIV_CNT 当 CPU_CLK 的时钟源为 XTL_CLK 或 RC_FAST_CLK 时，配置 CPU_CLK 的分频系数。取值范围为 0x0 ~ 0x3FF。CPU_CLK = XTL_CLK (或 RC_FAST_CLK) / (取值 +1)。(R/W)

Register 3.2. SYSCON_XTAL_TICK_CONF_REG (0x0004)



SYSCON_XTAL_TICK_NUM 当 APB_CLK 的时钟源为 XTL_CLK 时，配置 REF_TICK 的分频系数。取值范围为 0x0 ~ 0xFF。REF_TICK = APB_CLK /(取值 + 1)。(R/W)

Register 3.3. SYSCON_PLL_TICK_CONF_REG (0x0008)



SYSCON_PLL_TICK_NUM 当 APB_CLK 的时钟源为 PLL_CLK 时，配置 REF_TICK 的分频系数。取值范围为 0x0 ~ 0xFF。REF_TICK = APB_CLK /(取值 + 1)。(R/W)

Register 3.4. SYSCON_CK8M_TICK_CONF_REG (0x000C)

(reserved)	SYSCON_CK8M_TICK_NUM	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	11	Reset

SYSCON_CK8M_TICK_NUM 当 APB_CLK 的时钟源为 RC_FAST_CLK 时，配置 REF_TICK 的分频系数。取值范围为 0x0 ~ 0xFF。REF_TICK = APB_CLK /(取值 + 1)。(R/W)

Register 3.5. SYSCON_APLL_TICK_CONF_REG (0x003C)

(reserved)	SYSCON_APLL_TICK_NUM	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	99	Reset

SYSCON_APLL_TICK_NUM 当 APB_CLK 的时钟源为 APLL_CLK 时，配置 REF_TICK 的分频系数。取值范围为 0x0 ~ 0xFF。REF_TICK = APB_CLK /(取值 + 1)。(R/W)

Register 3.6. SYSCON_DATE_REG (0x007C)

SYSCON_DATE	
31 0	
0x16042000	Reset

SYSCON_DATE 芯片版本寄存器。更多信息请查看《ESP32 系列芯片勘误表》. (R/W)

4 IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)

4.1 概述

ESP32 芯片有 34 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 34 个物理 GPIO pad 的序列号为：0-19, 21-23, 25-27, 32-39。其中 GPIO 34-39 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。

本章描述了数字 pad（控制信号：FUN_SEL、IE、OE、WPU、WDU 等）和 162 个外设输入以及 176 个外设输出信号（控制信号：SIG_IN_SEL、SIG_OUT_SEL、IE、OE 等）和快速外设输入 / 输出信号（控制信号：IE、OE 等）以及 RTC IO_MUX 之间的信号选择和连接关系。

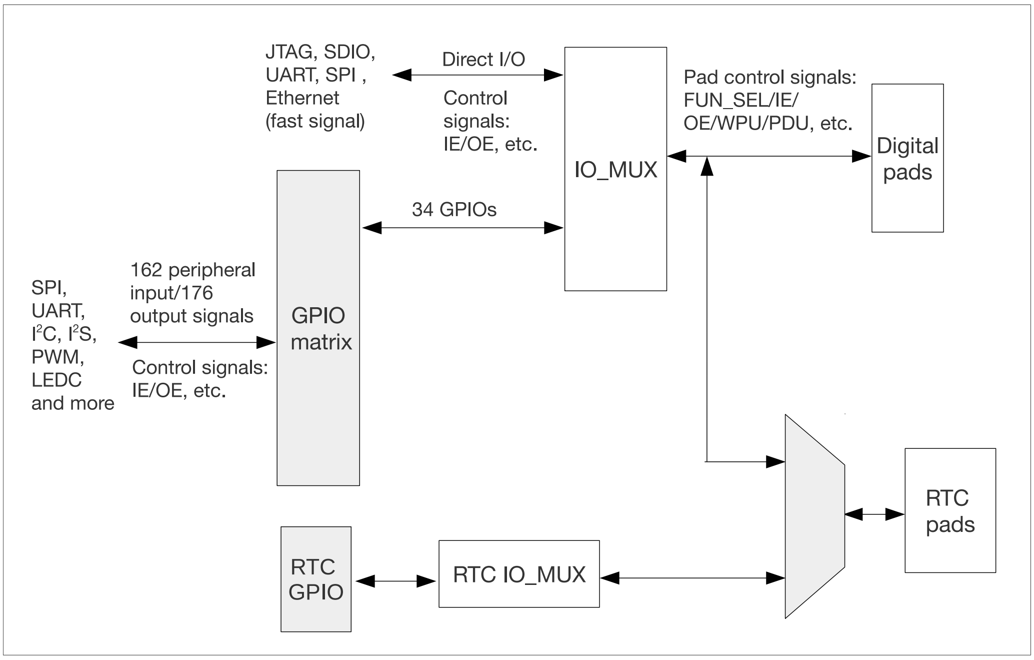


图 4-1. IO_MUX、RTC IO_MUX 和 GPIO 交换矩阵结构框图

1. IO_MUX 中每个 GPIO pad 有一组寄存器。每个 pad 可以配置成 GPIO 功能（连接 GPIO 交换矩阵）或者直连功能（旁路 GPIO 交换矩阵，快速信号如以太网、SDIO、SPI、JTAG、UART 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。）

   章节 4.10 列出了所有 GPIO pad 的 IO_MUX 功能。

2. GPIO 交换矩阵是外设输入和输出信号和 pad 之间的全交换矩阵。

   - 芯片输入方向：162 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 176 个外设输出信号中的任意一个。

   章节 4.9 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 4.11 列出了 RTC IO_MUX 功能。

4.2 通过 GPIO 交换矩阵的外设输入

4.2.1 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 34 个 GPIO（0-19，21-23，25-27，32-39）中获取外设输入信号的索引号（0-18，23-36，39-58，61-90，95-124，140-155，164-181，190-195，198-206）。

输入信号通过 IO_MUX 从 GPIO pad 中读取。IO_MUX 必须设置相应 pad 为 GPIO 功能。这样 GPIO pad 的输入信号就可进入 GPIO 交换矩阵然后通过 GPIO 交换矩阵进入选择的外设输入。

4.2.2 功能描述

图 4-2 为通过 GPIO 交换矩阵的外设输入的示意图。

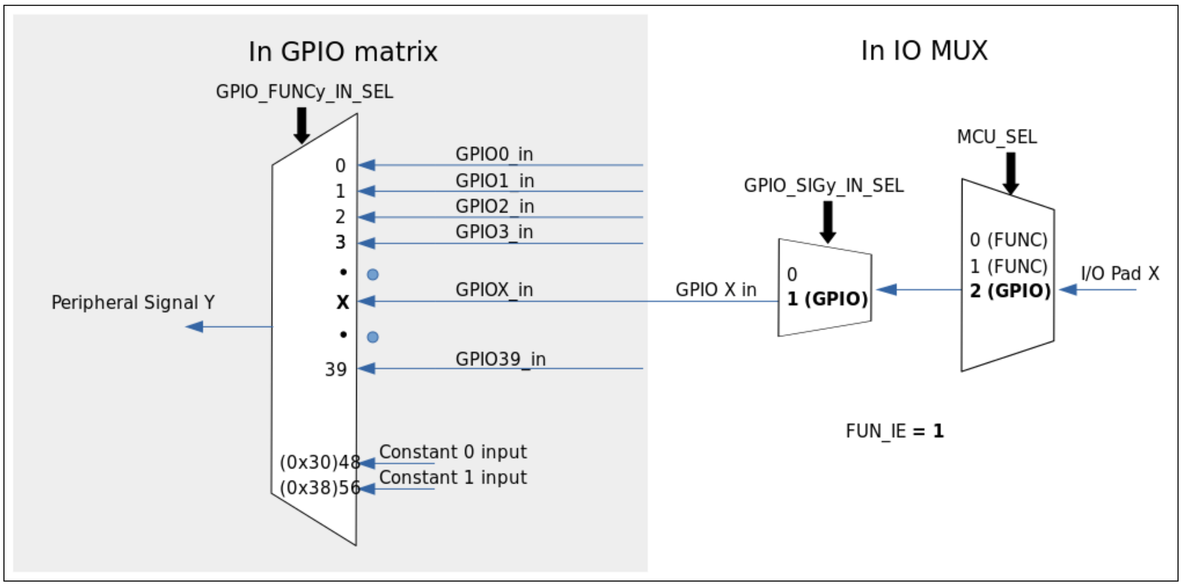


图 4-2. 通过 IO_MUX、GPIO 交换矩阵的外设输入

把某个外设信号 *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_FUNC*y*_IN_SEL_CFG 寄存器：
   - 置位 GPIO_SIG*y*_IN_SEL 选择通过 GPIO 交换矩阵接收外部输入信号；
   - 设置 GPIO_FUNC*y*_IN_SEL 字段为要读取的 GPIO pad *X* 的值。
2. 在 GPIO 交换矩阵中配置 GPIO pad *X* 的 GPIO_FUNC*x*_OUT_SEL_CFG 寄存器、清零 GPIO_ENABLE_DATA[*x*] 字段：
   - 要强制管脚的输出状态始终由 GPIO_ENABLE_DATA[*x*] 字段决定，则将 GPIO_FUNC*x*_OUT_SEL_CFG 寄存器的 GPIO_FUNC*x*_OEN_SEL 字段位置为 1。
   - GPIO_ENABLE_DATA[*x*] 字段在 GPIO_ENABLE_REG (GPIOs 0-31) 或 GPIO_ENABLE1_REG (GPIOs 32-39) 中，清零此位可以关闭 GPIO pad 的输出。
3. 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 (MCU_SEL) 为 GPIO *X* 的 IO_MUX 功能（所有管脚的 Function 2，数值为 2）。
   - 置位 FUN_IE 使能输入。

  - 置位或清零 FUN_WPU 和 FUN_WPD 位，使能或关闭内部上拉/下拉电阻器。

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 0x30 时，input_signal_*x* 始终为 0。
  - 当 GPIO_FUNC*y*_IN_SEL 是 0x38 时，input_signal_*x* 始终为 1。

例如，要把 RMT 外设通道 0 的输入信号 RMT_SIG_IN0_IDX（信号索引号 83）绑定到 GPIO15，请按照以下步骤操作（请注意 GPIO15 也叫做 MTDO 管脚）：

1. 将 GPIO_FUNC83_IN_SEL_CFG 寄存器的 GPIO_FUNC83_IN_SEL 字段设置为 15。
2. 因为此信号是纯输入信号，置位 GPIO_FUNC15_OUT_SEL_CFG_REG 寄存器中的 GPIO_FUNC15_OEN_SEL 位。
3. 清零 GPIO_ENABLE_REG 寄存器的 bit 15（GPIO_ENABLE_DATA[15] 字段）。
4. 配置 IO_MUX_GPIO15 寄存器的 MCU_SEL 字段为 2 (GPIO function)，同时置位 FUN_IE（使能输入模式）。

4.2.3 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 pin 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_*x*_REG 寄存器配置 FUN_IE 位以使能输入，如章节 4.2.2 所述。

4.3 通过 GPIO 交换矩阵的外设输出

4.3.1 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 0-18，23-37，61-121，140-215，224-228 的外设信号输出到 28 个 GPIO (0-19, 21-23, 25-27, 32-33)。

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 224-228 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。

4.3.2 功能描述

图 4-3 所示为 176 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。

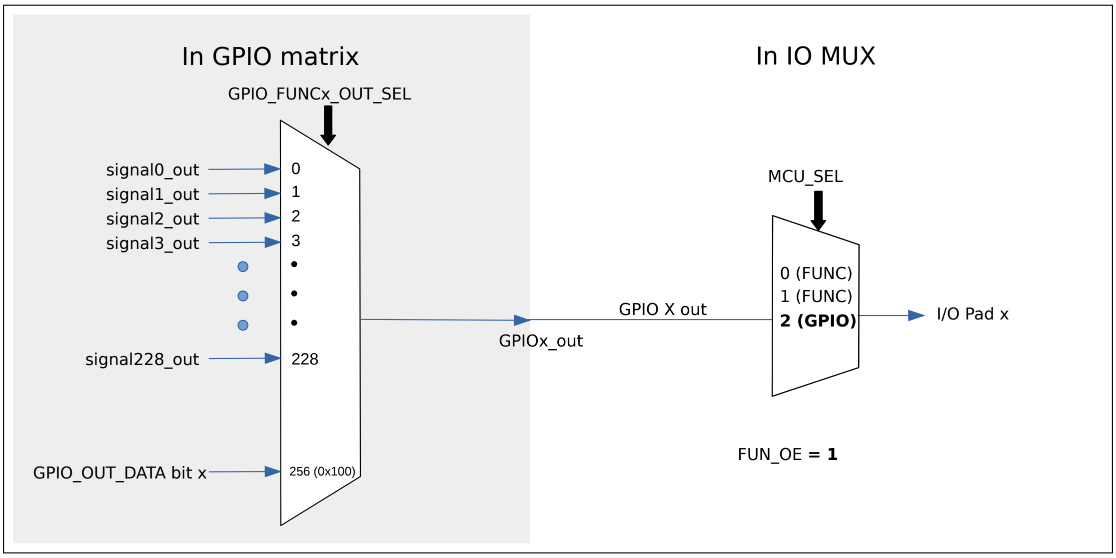


图 4-3. 通过 GPIO 交换矩阵输出信号

输出外设信号 *Y* 到某一 GPIO pad *X* 的步骤为：

1. 在 GPIO 交换矩阵里配置 GPIO *X* 的 GPIO_FUNC*x*_OUT_SEL_CFG 寄存器和 GPIO_ENABLE_DATA[*x*] 字段：
    - 设置 GPIO_FUNC*x*_OUT_SEL_CFG 寄存器的 GPIO_FUNC*x*_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，将 GPIO pad *X* 的 GPIO_FUNC*x*_OUT_SEL_CFG 寄存器的 GPIO_FUNC*x*_OEN_SEL 置位，并且将 GPIO_ENABLE_REG 寄存器的 GPIO_ENABLE_DATA[*x*] 字段置位。或者，将 GPIO_FUNC*x*_OEN_SEL 清零，此时输出使能信号由内部逻辑功能决定。
    - GPIO_ENABLE_DATA[*x*] 字段在 GPIO_ENABLE_REG (GPIOs 0-31) 或 GPIO_ENABLE1_REG (GPIOs 32-39) 中，清零此位可以关闭 GPIO pad 的输出。
2. 要选择以开漏方式输出，可以设置 GPIO *X* 的 GPIO_PIN*x* 寄存器中的 GPIO_PIN*x*_PAD_DRIVER 位。
3. 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 (MCU_SEL) 为 GPIO *X* 的 IO_MUX 功能（所有管脚的 Function 2，数值为 2）。
    - 设置 FUN_DRV 字段为特定的输出强度值 (0-3)，值越大，输出驱动能力越强。
    - 在开漏模式下，通过置位/清零 FUN_WPU 和 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。
- 只有 28 个 GPIO 管脚可用于输出信号。
- 置位 GPIO_FUNC*x*_OUT_INV_SEL 可以把输出的信号取反。

4.3.3 简单 GPIO 输出

GPIO 交换矩阵也可以用于简单 GPIO 输出。设置 GPIO_OUT_DATA 寄存器中某一位的值可以写入对应的 GPIO pad。

为实现某一 pad 的 GPIO 输出，设置 GPIO 交换矩阵 GPIO_FUNCx_OUT_SEL 寄存器为特定的外设索引值 256（0x100）。

4.4 IO_MUX 的直接 I/O 功能

4.4.1 概述

快速信号如以太网、SDIO、SPI、JTAG、UART 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。

4.4.2 功能描述

为实现外设 I/O 旁路 GPIO 交换矩阵必须配置两个寄存器：

1. GPIO pad 的 IO_MUX 必须设置为相应的 pad 功能，章节 4.10 列出了 pad 功能。
2. 对于输入信号，必须清零 SIG_IN_SEL 寄存器，直接将输入信号输出到外设。

4.5 RTC IO_MUX 的低功耗性能和模拟 I/O 功能

4.5.1 概述

18 个 GPIO 管脚具有低功耗（低功耗 RTC）性能和模拟功能，由 ESP32 的 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_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 睡眠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 4.11 列出了 RTC_MUX 管脚和功能。

4.5.2 模拟功能描述

RTC_GPIO 的 RTC 功能和模拟功能同一时刻只能选择一个，其中 RTC_GPIO8 ~ RTC_GPIO17 的模拟输出可以输出到 IO_MUX，该功能由寄存器 RTC_IO_TOUCH_PADn/m_TO_GPIO 位控制。该位为 1 时，pad 的输入信号通过模拟功能输出到 IO_MUX；该位为 0 时，则 pad 的输入信号通过数字功能输出到 IO_MUX。

4.6 Light-sleep 模式管脚功能

当 ESP32 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 寄存器中 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 4-1. IO_MUX Light-sleep 管脚功能寄存器

IO_MUX 功能	正常工作模式或者 SLP_SEL = 0	Light-sleep 模式并且 SLP_SEL = 1
Output Drive Strength	FUN_DRV	MCU_DRV
Pull-up Resistor	FUN_WPU	MCU_WPU
Pull-down Resistor	FUN_WPD	MCU_WPD
Output Enable	(From GPIO Matrix _OEN field)	MCU_OE

如果 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。

4.7 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时或 Deep-sleep 触发内核复位时 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 REG_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_CNTL_HOLD_FORCE_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 REG_DG_PAD_FORCE_UNHOLD 设置成 1。若想继续保持 pad 的值，可把 RTC_CNTL_HOLD_FORCE_REG 寄存器中相应的位设置成 1。

4.8 I/O Pad 供电

IO pad 供电如图 4-4 和图 4-5 所示。

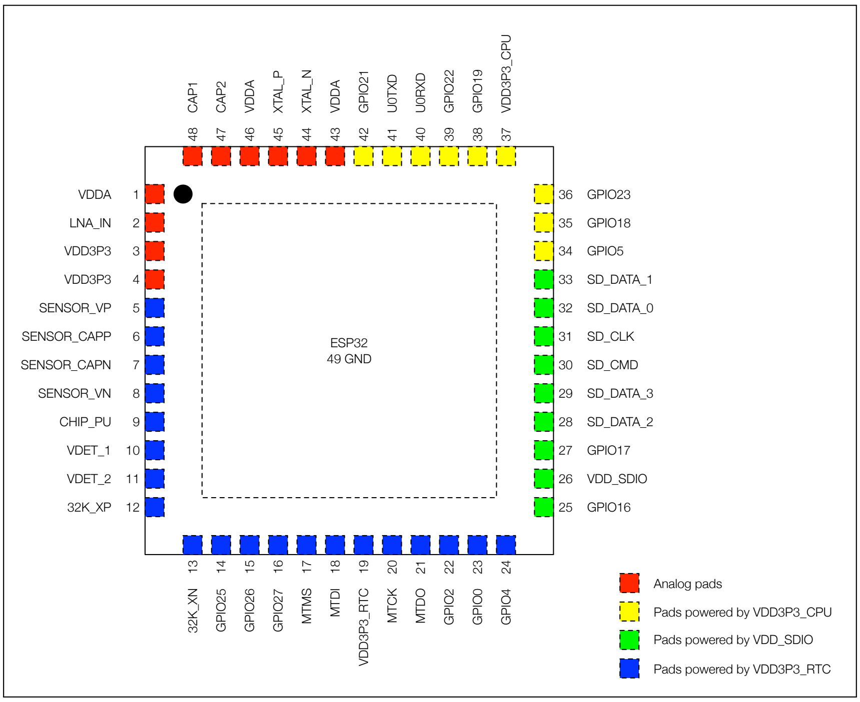


图 4-4. ESP32 I/O Pad 供电源（QFN 6*6，顶视图）

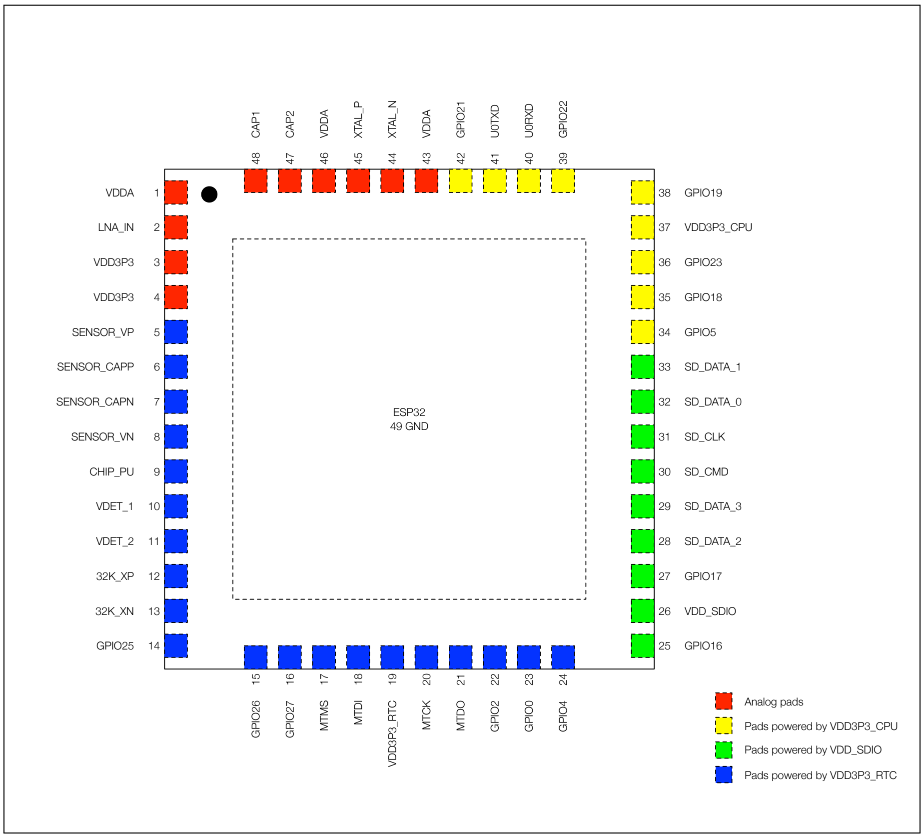


图 4-5. ESP32 I/O Pad 供电源（QFN 5*5，顶视图）

- 蓝色的 pad 为 RTC pad，它们都带有一种或几种模拟功能，也可以用作为正常数字 IO pad 使用，详见章节 4.11。
- 黄色和绿色的 pad 只有数字 IO 的功能。
- 绿色的 pad 可以通过 VDD_SDIO 由外部供电也可由芯片内部供电（详见下文）。

4.8.1 VDD_SDIO 电源域

VDD_SDIO 可以拉电流和灌电流，因此 VDD_SDIO 电源域可由外部或内部供电。若使用外部供电，必须使用和 VDD3P3_RTC 相同的电源。

如果外部不供电，则内部线性稳压器会给 VDD_SDIO 供电。VDD_SDIO 电压可以为 1.8 V 或与 VDD3P3_RTC 一致，这取决于 MTDI pad 在复位时的状态——高电平时为 1.8 V，低电平时为与 VDD3P3_RTC 一致。eFuse bit 置上后可强制决定 VDD_SDIO 的默认电压。此外，软件启动后软件还可以配置寄存器来强制改变 VDD_SDIO 的电压。

4.9 外设信号列表

表 4-2 列出了 GPIO 交换矩阵的外设输入 / 输出信号。

表 4-2. GPIO 交换矩阵外设信号

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
0	SPICLK_in	SPICLK_out	YES
1	SPIQ_in	SPIQ_out	YES
2	SPID_in	SPID_out	YES

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
3	SPIHD_in	SPIHD_out	YES
4	SPIWP_in	SPIWP_out	YES
5	SPICS0_in	SPICS0_out	YES
6	SPICS1_in	SPICS1_out	-
7	SPICS2_in	SPICS2_out	-
8	HSPICLK_in	HSPICLK_out	YES
9	HSPIQ_in	HSPIQ_out	YES
10	HSPID_in	HSPID_out	YES
11	HSPICS0_in	HSPICS0_out	YES
12	HSPIHD_in	HSPIHD_out	YES
13	HSPIWP_in	HSPIWP_out	YES
14	U0RXD_in	U0TXD_out	YES
15	U0CTS_in	U0RTS_out	YES
16	U0DSR_in	U0DTR_out	-
17	U1RXD_in	U1TXD_out	YES
18	U1CTS_in	U1RTS_out	YES
23	I2S0O_BCK_in	I2S0O_BCK_out	-
24	I2S1O_BCK_in	I2S1O_BCK_out	-
25	I2S0O_WS_in	I2S0O_WS_out	-
26	I2S1O_WS_in	I2S1O_WS_out	-
27	I2S0I_BCK_in	I2S0I_BCK_out	-
28	I2S0I_WS_in	I2S0I_WS_out	-
29	I2CEXT0_SCL_in	I2CEXT0_SCL_out	-
30	I2CEXT0_SDA_in	I2CEXT0_SDA_out	-
31	pwm0_sync0_in	sdio_tohost_int_out	-
32	pwm0_sync1_in	pwm0_out0a	-
33	pwm0_sync2_in	pwm0_out0b	-
34	pwm0_f0_in	pwm0_out1a	-
35	pwm0_f1_in	pwm0_out1b	-
36	pwm0_f2_in	pwm0_out2a	-
37	-	pwm0_out2b	-
39	pcnt_sig_ch0_in0	-	-
40	pcnt_sig_ch1_in0	-	-
41	pcnt_ctrl_ch0_in0	-	-
42	pcnt_ctrl_ch1_in0	-	-
43	pcnt_sig_ch0_in1	-	-
44	pcnt_sig_ch1_in1	-	-
45	pcnt_ctrl_ch0_in1	-	-
46	pcnt_ctrl_ch1_in1	-	-
47	pcnt_sig_ch0_in2	-	-
48	pcnt_sig_ch1_in2	-	-
49	pcnt_ctrl_ch0_in2	-	-
50	pcnt_ctrl_ch1_in2	-	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
51	pcnt_sig_ch0_in3	-	-
52	pcnt_sig_ch1_in3	-	-
53	pcnt_ctrl_ch0_in3	-	-
54	pcnt_ctrl_ch1_in3	-	-
55	pcnt_sig_ch0_in4	-	-
56	pcnt_sig_ch1_in4	-	-
57	pcnt_ctrl_ch0_in4	-	-
58	pcnt_ctrl_ch1_in4	-	-
61	HSPICS1_in	HSPICS1_out	-
62	HSPICS2_in	HSPICS2_out	-
63	VSPICLK_in	VSPICLK_out_mux	YES
64	VSPIQ_in	VSPIQ_out	YES
65	VSPID_in	VSPID_out	YES
66	VSPIHD_in	VSPIHD_out	YES
67	VSPIWP_in	VSPIWP_out	YES
68	VSPICS0_in	VSPICS0_out	YES
69	VSPICS1_in	VSPICS1_out	-
70	VSPICS2_in	VSPICS2_out	-
71	pcnt_sig_ch0_in5	ledc_hs_sig_out0	-
72	pcnt_sig_ch1_in5	ledc_hs_sig_out1	-
73	pcnt_ctrl_ch0_in5	ledc_hs_sig_out2	-
74	pcnt_ctrl_ch1_in5	ledc_hs_sig_out3	-
75	pcnt_sig_ch0_in6	ledc_hs_sig_out4	-
76	pcnt_sig_ch1_in6	ledc_hs_sig_out5	-
77	pcnt_ctrl_ch0_in6	ledc_hs_sig_out6	-
78	pcnt_ctrl_ch1_in6	ledc_hs_sig_out7	-
79	pcnt_sig_ch0_in7	ledc_ls_sig_out0	-
80	pcnt_sig_ch1_in7	ledc_ls_sig_out1	-
81	pcnt_ctrl_ch0_in7	ledc_ls_sig_out2	-
82	pcnt_ctrl_ch1_in7	ledc_ls_sig_out3	-
83	rmt_sig_in0	ledc_ls_sig_out4	-
84	rmt_sig_in1	ledc_ls_sig_out5	-
85	rmt_sig_in2	ledc_ls_sig_out6	-
86	rmt_sig_in3	ledc_ls_sig_out7	-
87	rmt_sig_in4	rmt_sig_out0	-
88	rmt_sig_in5	rmt_sig_out1	-
89	rmt_sig_in6	rmt_sig_out2	-
90	rmt_sig_in7	rmt_sig_out3	-
91	-	rmt_sig_out4	-
92	-	rmt_sig_out5	-
93	-	rmt_sig_out6	-
94	twai_rx	rmt_sig_out7	-
95	I2CEXT1_SCL_in	I2CEXT1_SCL_out	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
96	I2CEXT1_SDA_in	I2CEXT1_SDA_out	-
97	host_card_detect_n_1	host_ccmd_od_pullup_en_n	-
98	host_card_detect_n_2	host_rst_n_1	-
99	host_card_write_prt_1	host_rst_n_2	-
100	host_card_write_prt_2	gpio_sd0_out	-
101	host_card_int_n_1	gpio_sd1_out	-
102	host_card_int_n_2	gpio_sd2_out	-
103	pwm1_sync0_in	gpio_sd3_out	-
104	pwm1_sync1_in	gpio_sd4_out	-
105	pwm1_sync2_in	gpio_sd5_out	-
106	pwm1_f0_in	gpio_sd6_out	-
107	pwm1_f1_in	gpio_sd7_out	-
108	pwm1_f2_in	pwm1_out0a	-
109	pwm0_cap0_in	pwm1_out0b	-
110	pwm0_cap1_in	pwm1_out1a	-
111	pwm0_cap2_in	pwm1_out1b	-
112	pwm1_cap0_in	pwm1_out2a	-
113	pwm1_cap1_in	pwm1_out2b	-
114	pwm1_cap2_in	-	-
115	-	-	-
116	-	-	-
117	-	-	-
118	-	-	-
119	-	-	-
120	-	-	-
121	-	-	-
122	-	-	-
123	-	twai_tx	-
124	-	twai_bus_off_on	-
125	-	twai_clkout	-
140	I2S0I_DATA_in0	I2S0O_DATA_out0	-
141	I2S0I_DATA_in1	I2S0O_DATA_out1	-
142	I2S0I_DATA_in2	I2S0O_DATA_out2	-
143	I2S0I_DATA_in3	I2S0O_DATA_out3	-
144	I2S0I_DATA_in4	I2S0O_DATA_out4	-
145	I2S0I_DATA_in5	I2S0O_DATA_out5	-
146	I2S0I_DATA_in6	I2S0O_DATA_out6	-
147	I2S0I_DATA_in7	I2S0O_DATA_out7	-
148	I2S0I_DATA_in8	I2S0O_DATA_out8	-
149	I2S0I_DATA_in9	I2S0O_DATA_out9	-
150	I2S0I_DATA_in10	I2S0O_DATA_out10	-
151	I2S0I_DATA_in11	I2S0O_DATA_out11	-
152	I2S0I_DATA_in12	I2S0O_DATA_out12	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
153	I2S0I_DATA_in13	I2S0O_DATA_out13	-
154	I2S0I_DATA_in14	I2S0O_DATA_out14	-
155	I2S0I_DATA_in15	I2S0O_DATA_out15	-
156	-	I2S0O_DATA_out16	-
157	-	I2S0O_DATA_out17	-
158	-	I2S0O_DATA_out18	-
159	-	I2S0O_DATA_out19	-
160	-	I2S0O_DATA_out20	-
161	-	I2S0O_DATA_out21	-
162	-	I2S0O_DATA_out22	-
163	-	I2S0O_DATA_out23	-
164	I2S1I_BCK_in	I2S1I_BCK_out	-
165	I2S1I_WS_in	I2S1I_WS_out	-
166	I2S1I_DATA_in0	I2S1O_DATA_out0	-
167	I2S1I_DATA_in1	I2S1O_DATA_out1	-
168	I2S1I_DATA_in2	I2S1O_DATA_out2	-
169	I2S1I_DATA_in3	I2S1O_DATA_out3	-
170	I2S1I_DATA_in4	I2S1O_DATA_out4	-
171	I2S1I_DATA_in5	I2S1O_DATA_out5	-
172	I2S1I_DATA_in6	I2S1O_DATA_out6	-
173	I2S1I_DATA_in7	I2S1O_DATA_out7	-
174	I2S1I_DATA_in8	I2S1O_DATA_out8	-
175	I2S1I_DATA_in9	I2S1O_DATA_out9	-
176	I2S1I_DATA_in10	I2S1O_DATA_out10	-
177	I2S1I_DATA_in11	I2S1O_DATA_out11	-
178	I2S1I_DATA_in12	I2S1O_DATA_out12	-
179	I2S1I_DATA_in13	I2S1O_DATA_out13	-
180	I2S1I_DATA_in14	I2S1O_DATA_out14	-
181	I2S1I_DATA_in15	I2S1O_DATA_out15	-
182	-	I2S1O_DATA_out16	-
183	-	I2S1O_DATA_out17	-
184	-	I2S1O_DATA_out18	-
185	-	I2S1O_DATA_out19	-
186	-	I2S1O_DATA_out20	-
187	-	I2S1O_DATA_out21	-
188	-	I2S1O_DATA_out22	-
189	-	I2S1O_DATA_out23	-
190	I2S0I_H_SYNC	-	-
191	I2S0I_V_SYNC	-	-
192	I2S0I_H_ENABLE	-	-
193	I2S1I_H_SYNC	-	-
194	I2S1I_V_SYNC	-	-
195	I2S1I_H_ENABLE	-	-

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
196	-	-	-
197	-	-	-
198	U2RXD_in	U2TXD_out	YES
199	U2CTS_in	U2RTS_out	YES
200	emac_mdc_i	emac_mdc_o	-
201	emac_mdi_i	emac_mdo_o	-
202	emac_crs_i	emac_crs_o	-
203	emac_col_i	emac_col_o	-
204	pcmfsync_in	bt_audio0_irq	-
205	pcmclk_in	bt_audio1_irq	-
206	pcmdin	bt_audio2_irq	-
207	-	ble_audio0_irq	-
208	-	ble_audio1_irq	-
209	-	ble_audio2_irq	-
210	-	pcmfsync_out	-
211	-	pcmclk_out	-
212	-	pcmdout	-
213	-	ble_audio_sync0_p	-
214	-	ble_audio_sync1_p	-
215	-	ble_audio_sync2_p	-
224	-	sig_in_func224	-
225	-	sig_in_func225	-
226	-	sig_in_func226	-
227	-	sig_in_func227	-
228	-	sig_in_func228	-

Direct I/O in IO_MUX ”YES” 指此信号也可以通过 IO_MUX 直接连接 pad。如果这些信号要使用 GPIO 交换矩阵，则必须将相应的 SIG_IN_SEL 寄存器清零。

4.10 IO_MUX Pad 列表

表 4-3 列出了每个 I/O pad 的 IO_MUX 功能。

表 4-3. IO_MUX Pad 列表

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Reset	Notes
0	GPIO0	GPIO0	CLK_OUT1	GPIO0	-	-	EMAC_TX_CLK	3	R
1	U0TXD	U0TXD	CLK_OUT3	GPIO1	-	-	EMAC_RXD2	3	-
2	GPIO2	GPIO2	HSPIWP	GPIO2	HS2_DATA0	SD_DATA0	-	2	R
3	U0RXD	U0RXD	CLK_OUT2	GPIO3	-	-	-	3	-
4	GPIO4	GPIO4	HSPIHD	GPIO4	HS2_DATA1	SD_DATA1	EMAC_TX_ER	2	R
5	GPIO5	GPIO5	VSPICS0	GPIO5	HS1_DATA6	-	EMAC_RX_CLK	3	-
6	SD_CLK	SD_CLK	SPICLK	GPIO6	HS1_CLK	U1CTS	-	3	-
7	SD_DATA_0	SD_DATA0	SPIQ	GPIO7	HS1_DATA0	U2RTS	-	3	-
8	SD_DATA_1	SD_DATA1	SPID	GPIO8	HS1_DATA1	U2CTS	-	3	-
9	SD_DATA_2	SD_DATA2	SPIHD	GPIO9	HS1_DATA2	U1RXD	-	3	-
10	SD_DATA_3	SD_DATA3	SPIWP	GPIO10	HS1_DATA3	U1TXD	-	3	-
11	SD_CMD	SD_CMD	SPICS0	GPIO11	HS1_CMD	U1RTS	-	3	-

GPIO	Pad Name	Function 0	Function 1	Function 2	Function 3	Function 4	Function 5	Reset	Notes
12	MTDI	MTDI	HSPIQ	GPIO12	HS2_DATA2	SD_DATA2	EMAC_TXD3	2	R
13	MTCK	MTCK	HSPID	GPIO13	HS2_DATA3	SD_DATA3	EMAC_RX_ER	2	R
14	MTMS	MTMS	HSPICLK	GPIO14	HS2_CLK	SD_CLK	EMAC_TXD2	3	R
15	MTDO	MTDO	HSPICS0	GPIO15	HS2_CMD	SD_CMD	EMAC_RXD3	3	R
16	GPIO16	GPIO16	-	GPIO16	HS1_DATA4	U2RXD	EMAC_CLK_OUT	1	-
17	GPIO17	GPIO17	-	GPIO17	HS1_DATA5	U2TXD	EMAC_CLK_180	1	-
18	GPIO18	GPIO18	VSPICLK	GPIO18	HS1_DATA7	-	-	1	-
19	GPIO19	GPIO19	VSPIQ	GPIO19	U0CTS	-	EMAC_TXD0	1	-
21	GPIO21	GPIO21	VSPIHD	GPIO21	-	-	EMAC_TX_EN	1	-
22	GPIO22	GPIO22	VSPIWP	GPIO22	U0RTS	-	EMAC_TXD1	1	-
23	GPIO23	GPIO23	VSPID	GPIO23	HS1_STROBE	-	-	1	-
25	GPIO25	GPIO25	-	GPIO25	-	-	EMAC_RXD0	0	R
26	GPIO26	GPIO26	-	GPIO26	-	-	EMAC_RXD1	0	R
27	GPIO27	GPIO27	-	GPIO27	-	-	EMAC_RX_DV	0	R
32	32K_XP	GPIO32	-	GPIO32	-	-	-	0	R
33	32K_XN	GPIO33	-	GPIO33	-	-	-	0	R
34	VDET_1	GPIO34	-	GPIO34	-	-	-	0	R, I
35	VDET_2	GPIO35	-	GPIO35	-	-	-	0	R, I
36	SENSOR_VP	GPIO36	-	GPIO36	-	-	-	0	R, I
37	SENSOR_CAPP	GPIO37	-	GPIO37	-	-	-	0	R, I
38	SENSOR_CAPN	GPIO38	-	GPIO38	-	-	-	0	R, I
39	SENSOR_VN	GPIO39	-	GPIO39	-	-	-	0	R, 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）
- **3** - IE=1, WPU=1（输入使能，上拉电阻）

说明

- **R** - Pad 通过 RTC_MUX 具有 RTC / 模拟功能。
- **I** - Pad 只能配置为输入 GPIO。此类 Pad 不具备输出驱动或内置上拉/下拉电路。

请参考《ESP32 技术规格书》的附录查看管脚功能的完整表格。

4.11 RTC_MUX 管脚清单

表 4-4 列出了 RTC 管脚和对应 GPIO pad。

表 4-4. RTC_MUX 管脚清单

RTC GPIO Num	GPIO Num	Pad Name	Analog Function			RTC Function	
			0	1	2	Function 0 (FUN_SEL = 0)	Function 1 (FUN_SEL = 3)
0	36	SENSOR_VP	ADC_H	ADC1_CH0	-	RTC_GPIO0	-
1	37	SENSOR_CAPP	ADC_H	ADC1_CH1	-	RTC_GPIO1	-
2	38	SENSOR_CAPN	ADC_H	ADC1_CH2	-	RTC_GPIO2	-
3	39	SENSOR_VN	ADC_H	ADC1_CH3	-	RTC_GPIO3	-

RTC GPIO Num	GPIO Num	Pad Name	Analog Function			RTC Function	
			0	1	2	Function 0 (FUN_SEL = 0)	Function 1 (FUN_SEL = 3)
4	34	VDET_1	-	ADC1_CH6	-	RTC_GPIO4	-
5	35	VDET_2	-	ADC1_CH7	-	RTC_GPIO5	-
6	25	GPIO25	DAC_1	ADC2_CH8	-	RTC_GPIO6	-
7	26	GPIO26	DAC_2	ADC2_CH9	-	RTC_GPIO7	-
8	33	32K_XN	XTAL_32K_N	ADC1_CH5	TOUCH8	RTC_GPIO8	-
9	32	32K_XP	XTAL_32K_P	ADC1_CH4	TOUCH9	RTC_GPIO9	-
10	4	GPIO4	-	ADC2_CH0	TOUCH0	RTC_GPIO10	I2C_SCL*
11	0	GPIO0	-	ADC2_CH1	TOUCH1	RTC_GPIO11	I2C_SDA*
12	2	GPIO2	-	ADC2_CH2	TOUCH2	RTC_GPIO12	I2C_SCL*
13	15	MTDO	-	ADC2_CH3	TOUCH3	RTC_GPIO13	I2C_SDA*
14	13	MTCK	-	ADC2_CH4	TOUCH4	RTC_GPIO14	-
15	12	MTDI	-	ADC2_CH5	TOUCH5	RTC_GPIO15	-
16	14	MTMS	-	ADC2_CH6	TOUCH6	RTC_GPIO16	-
17	27	GPIO27	-	ADC2_CH7	TOUCH7	RTC_GPIO17	-

说明：

有关 RTC I2C 的配置信息，请参考章节 30 *超低功耗协处理器 (ULP)*：RTC I2C 控制器。

4.12 寄存器列表

4.12.1 GPIO 交换矩阵寄存器列表

名称	描述	地址	访问
GPIO_OUT_REG	GPIO 0-31 输出寄存器	0x3FF44004	读/写
GPIO_OUT_W1TS_REG	GPIO 0-31 输出寄存器 _WITS	0x3FF44008	只写
GPIO_OUT_W1TC_REG	GPIO 0-31 输出寄存器 _WITC	0x3FF4400C	只写
GPIO_OUT1_REG	GPIO 32-39 输出寄存器	0x3FF44010	读/写
GPIO_OUT1_W1TS_REG	GPIO 32-39 输出置位寄存器	0x3FF44014	只写
GPIO_OUT1_W1TC_REG	GPIO 32-39 输出清除寄存器	0x3FF44018	只写
GPIO_ENABLE_REG	GPIO 0-31 输出使能寄存器	0x3FF44020	读/写
GPIO_ENABLE_W1TS_REG	GPIO 0-31 输出使能寄存器 _W1TS	0x3FF44024	只写
GPIO_ENABLE_W1TC_REG	GPIO 0-31 输出使能寄存器 _W1TC	0x3FF44028	只写
GPIO_ENABLE1_REG	GPIO 32-39 输出使能寄存器	0x3FF4402C	读/写
GPIO_ENABLE1_W1TS_REG	GPIO 32-39 输出使能置位寄存器	0x3FF44030	只写
GPIO_ENABLE1_W1TC_REG	GPIO 32-39 输出使能清除寄存器	0x3FF44034	只写
GPIO_STRAP_REG	Bootstrap 管脚值寄存器	0x3FF44038	只读
GPIO_IN_REG	GPIO 0-31 输入寄存器	0x3FF4403C	只读
GPIO_IN1_REG	GPIO 32-39 输入寄存器	0x3FF44040	只读
GPIO_STATUS_REG	GPIO 0-31 中断状态寄存器	0x3FF44044	读/写
GPIO_STATUS_W1TS_REG	GPIO 0-31 中断状态寄存器 _W1TS	0x3FF44048	只写
GPIO_STATUS_W1TC_REG	GPIO 0-31 中断状态寄存器 _W1TC	0x3FF4404C	只写
GPIO_STATUS1_REG	GPIO 32-39 中断状态寄存器 1	0x3FF44050	读/写

名称	描述	地址	访问
GPIO_STATUS1_W1TS_REG	GPIO 32-39 中断状态置位寄存器	0x3FF44054	只写
GPIO_STATUS1_W1TC_REG	GPIO 32-39 中断状态清除寄存器	0x3FF44058	只写
GPIO_ACPU_INT_REG	GPIO 0-31 APP_CPU 中断状态	0x3FF44060	只读
GPIO_ACPU_NMI_INT_REG	GPIO 0-31 APP_CPU 非屏蔽中断状态	0x3FF44064	只读
GPIO_PCPU_INT_REG	GPIO 0-31 PRO_CPU 中断状态	0x3FF44068	只读
GPIO_PCPU_NMI_INT_REG	GPIO 0-31 PRO_CPU 非屏蔽中断状态	0x3FF4406C	只读
GPIO_ACPU_INT1_REG	GPIO 31-39 APP_CPU 中断状态	0x3FF44074	只读
GPIO_ACPU_NMI_INT1_REG	GPIO 32-39 APP_CPU 非屏蔽中断状态	0x3FF44078	只读
GPIO_PCPU_INT1_REG	GPIO 32-39 PRO_CPU 中断状态	0x3FF4407C	只读
GPIO_PCPU_NMI_INT1_REG	GPIO 32-39 PRO_CPU 非屏蔽中断状态	0x3FF44080	只读
GPIO_PIN0_REG	配置 GPIO 管脚 0	0x3FF44088	读/写
GPIO_PIN1_REG	配置 GPIO 管脚 1	0x3FF4408C	读/写
GPIO_PIN2_REG	配置 GPIO 管脚 2	0x3FF44090	读/写
...	...	...	...
GPIO_PIN38_REG	配置 GPIO 管脚 38	0x3FF44120	读/写
GPIO_PIN39_REG	配置 GPIO 管脚 39	0x3FF44124	读/写
GPIO_FUNC0_IN_SEL_CFG_REG	外设功能 0 输入选择寄存器	0x3FF44130	读/写
GPIO_FUNC1_IN_SEL_CFG_REG	外设功能 1 输入选择寄存器	0x3FF44134	读/写
...	...	...	...
GPIO_FUNC254_IN_SEL_CFG_REG	外设功能 254 输入选择寄存器	0x3FF44528	读/写
GPIO_FUNC255_IN_SEL_CFG_REG	外设功能 255 输入选择寄存器	0x3FF4452C	读/写
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 的外设输出选择	0x3FF44530	读/写
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 的外设输出选择	0x3FF44534	读/写
...	...	...	...
GPIO_FUNC38_OUT_SEL_CFG_REG	GPIO38 的外设输出选择	0x3FF445C8	读/写
GPIO_FUNC39_OUT_SEL_CFG_REG	GPIO39 的外设输出选择	0x3FF445CC	读/写

4.12.2 IO MUX 寄存器列表

名称	描述	地址	访问
IO_MUX_PIN_CTRL	外设时钟配置寄存器	0x3FF49000	读/写
IO_MUX_GPIO36_REG	管脚 GPIO36 的配置寄存器	0x3FF49004	读/写
IO_MUX_GPIO37_REG	管脚 GPIO37 的配置寄存器	0x3FF49008	读/写
IO_MUX_GPIO38_REG	管脚 GPIO38 的配置寄存器	0x3FF4900C	读/写
IO_MUX_GPIO39_REG	管脚 GPIO39 的配置寄存器	0x3FF49010	读/写
IO_MUX_GPIO34_REG	管脚 GPIO34 的配置寄存器	0x3FF49014	读/写
IO_MUX_GPIO35_REG	管脚 GPIO35 的配置寄存器	0x3FF49018	读/写
IO_MUX_GPIO32_REG	管脚 GPIO32 的配置寄存器	0x3FF4901C	读/写
IO_MUX_GPIO33_REG	管脚 GPIO33 的配置寄存器	0x3FF49020	读/写
IO_MUX_GPIO25_REG	管脚 GPIO25 的配置寄存器	0x3FF49024	读/写
IO_MUX_GPIO26_REG	管脚 GPIO26 的配置寄存器	0x3FF49028	读/写
IO_MUX_GPIO27_REG	管脚 GPIO27 的配置寄存器	0x3FF4902C	读/写
IO_MUX_MTMS_REG	管脚 MTMS 的配置寄存器	0x3FF49030	读/写
IO_MUX_MTDI_REG	管脚 MTDI 的配置寄存器	0x3FF49034	读/写

名称	描述	地址	访问
IO_MUX_MTCK_REG	管脚 MTCK 的配置寄存器	0x3FF49038	读/写
IO_MUX_MTDO_REG	管脚 MTDO 的配置寄存器	0x3FF4903C	读/写
IO_MUX_GPIO2_REG	管脚 GPIO2 的配置寄存器	0x3FF49040	读/写
IO_MUX_GPIO0_REG	管脚 GPIO0 的配置寄存器	0x3FF49044	读/写
IO_MUX_GPIO4_REG	管脚 GPIO4 的配置寄存器	0x3FF49048	读/写
IO_MUX_GPIO16_REG	管脚 GPIO16 的配置寄存器	0x3FF4904C	读/写
IO_MUX_GPIO17_REG	管脚 GPIO17 的配置寄存器	0x3FF49050	读/写
IO_MUX_SD_DATA2_REG	管脚 SD_DATA2 的配置寄存器	0x3FF49054	读/写
IO_MUX_SD_DATA3_REG	管脚 SD_DATA3 的配置寄存器	0x3FF49058	读/写
IO_MUX_SD_CMD_REG	管脚 SD_CMD 的配置寄存器	0x3FF4905C	读/写
IO_MUX_SD_CLK_REG	管脚 SD_CLK 的配置寄存器	0x3FF49060	读/写
IO_MUX_SD_DATA0_REG	管脚 SD_DATA0 的配置寄存器	0x3FF49064	读/写
IO_MUX_SD_DATA1_REG	管脚 SD_DATA1 的配置寄存器	0x3FF49068	读/写
IO_MUX_GPIO5_REG	管脚 GPIO5 的配置寄存器	0x3FF4906C	读/写
IO_MUX_GPIO18_REG	管脚 GPIO18 的配置寄存器	0x3FF49070	读/写
IO_MUX_GPIO19_REG	管脚 GPIO19 的配置寄存器	0x3FF49074	读/写
IO_MUX_GPIO20_REG	管脚 GPIO20 的配置寄存器	0x3FF49078	读/写
IO_MUX_GPIO21_REG	管脚 GPIO21 的配置寄存器	0x3FF4907C	读/写
IO_MUX_GPIO22_REG	管脚 GPIO22 的配置寄存器	0x3FF49080	读/写
IO_MUX_U0RXD_REG	管脚 U0RXD 的配置寄存器	0x3FF49084	读/写
IO_MUX_U0TXD_REG	管脚 U0TXD 的配置寄存器	0x3FF49088	读/写
IO_MUX_GPIO23_REG	管脚 GPIO23 的配置寄存器	0x3FF4908C	读/写
IO_MUX_GPIO24_REG	管脚 GPIO24 的配置寄存器	0x3FF49090	读/写

4.12.3 RTC IO MUX 寄存器列表

名称	描述	地址	访问
GPIO 配置 / 数据寄存器			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO 输出寄存器	0x3FF48400	读/写
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO 输出置位寄存器	0x3FF48404	只写
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO 输出清除寄存器	0x3FF48408	只写
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO 输出使能寄存器	0x3FF4840C	读/写
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO 输出使能置位寄存器	0x3FF48410	只写
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO 输出使能清除寄存器	0x3FF48414	只写
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO 中断状态寄存器	0x3FF48418	读/写
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO 中断状态置位寄存器	0x3FF4841C	只写
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO 中断状态清除寄存器	0x3FF48420	只写
RTCIO_RTC_GPIO_IN_REG	RTC GPIO 输入寄存器	0x3FF48424	只读
RTCIO_RTC_GPIO_PIN0_REG	配置 RTC 管脚 0	0x3FF48428	读/写
RTCIO_RTC_GPIO_PIN1_REG	配置 RTC 管脚 1	0x3FF4842C	读/写
RTCIO_RTC_GPIO_PIN2_REG	配置 RTC 管脚 2	0x3FF48430	读/写
RTCIO_RTC_GPIO_PIN3_REG	配置 RTC 管脚 3	0x3FF48434	读/写
RTCIO_RTC_GPIO_PIN4_REG	配置 RTC 管脚 4	0x3FF48438	读/写
RTCIO_RTC_GPIO_PIN5_REG	配置 RTC 管脚 5	0x3FF4843C	读/写

名称	描述	地址	访问
RTCIO_RTC_GPIO_PIN6_REG	配置 RTC 管脚 6	0x3FF48440	读/写
RTCIO_RTC_GPIO_PIN7_REG	配置 RTC 管脚 7	0x3FF48444	读/写
RTCIO_RTC_GPIO_PIN8_REG	配置 RTC 管脚 8	0x3FF48448	读/写
RTCIO_RTC_GPIO_PIN9_REG	配置 RTC 管脚 9	0x3FF4844C	读/写
RTCIO_RTC_GPIO_PIN10_REG	配置 RTC 管脚 10	0x3FF48450	读/写
RTCIO_RTC_GPIO_PIN11_REG	配置 RTC 管脚 11	0x3FF48454	读/写
RTCIO_RTC_GPIO_PIN12_REG	配置 RTC 管脚 12	0x3FF48458	读/写
RTCIO_RTC_GPIO_PIN13_REG	配置 RTC 管脚 13	0x3FF4845C	读/写
RTCIO_RTC_GPIO_PIN14_REG	配置 RTC 管脚 14	0x3FF48460	读/写
RTCIO_RTC_GPIO_PIN15_REG	配置 RTC 管脚 15	0x3FF48464	读/写
RTCIO_RTC_GPIO_PIN16_REG	配置 RTC 管脚 16	0x3FF48468	读/写
RTCIO_RTC_GPIO_PIN17_REG	配置 RTC 管脚 17	0x3FF4846C	读/写
RTCIO_DIG_PAD_HOLD_REG	RTC GPIO Hold 寄存器	0x3FF48474	读/写
GPIO RTC 功能配置寄存器			
RTCIO_SENSOR_PADS_REG	传感器管脚配置寄存器	0x3FF4847C	读/写
RTCIO_ADC_PAD_REG	ADC 配置寄存器	0x3FF48480	读/写
RTCIO_PAD_DAC1_REG	DAC1 配置寄存器	0x3FF48484	读/写
RTCIO_PAD_DAC2_REG	DAC2 配置寄存器	0x3FF48488	读/写
RTCIO_XTAL_32K_PAD_REG	32 KHz 晶振管脚配置寄存器	0x3FF4848C	读/写
RTCIO_TOUCH_CFG_REG	触摸传感器配置寄存器	0x3FF48490	读/写
RTCIO_TOUCH_PAD0_REG	触摸管脚配置寄存器	0x3FF48494	读/写
...	...	...	...
RTCIO_TOUCH_PAD9_REG	触摸管脚配置寄存器	0x3FF484B8	读/写
RTCIO_EXT_WAKEUP0_REG	外部唤醒配置寄存器	0x3FF484BC	读/写
RTCIO_XTL_EXT_CTR_REG	晶振断电 GPIO 使能源	0x3FF484C0	读/写
RTCIO_SAR_I2C_IO_REG	RTC I2C 管脚选择	0x3FF484C4	读/写

4.13 寄存器

4.13.1 GPIO 交换矩阵寄存器

本小节括号中的地址均为相对于 GPIO 基地址的地址偏移量（相对地址），具体基地址见章节 1 *系统和存储器*中的表 1-6 *外设地址映射*。寄存器绝对地址见章节 4.12.1 *GPIO 交换矩阵寄存器列表*。

Register 4.1. GPIO_OUT_REG (0x0004)

31																															0	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reset

GPIO_OUT_REG GPIO0-31 输出值。（读/写）

Register 4.2. GPIO_OUT_W1TS_REG (0x0008)

31																															0	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reset

GPIO_OUT_W1TS_REG GPIO0-31 输出置位寄存器。每一位置 1，则 GPIO_OUT_REG 中的相应位也会置 1。（只写）

Register 4.3. GPIO_OUT_W1TC_REG (0x000c)

31																															0	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	Reset

GPIO_OUT_W1TC_REG 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。（只写）

Register 4.4. GPIO_OUT1_REG (0x0010)

(reserved)																								GPIO_OUT_DATA								
31																							8	7							0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	x	x	x	x	x	x	x	x	Reset

GPIO_OUT_DATA GPIO32-39 输出值。（只读）

Register 4.5. GPIO_OUT1_W1TS_REG (0x0014)

(reserved)	GPIO_OUT_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_OUT_DATA GPIO32-39 输出值置位寄存器。每一位置 1，则 GPIO_OUT1_DATA 中的相应位也会置 1。（只读）

Register 4.6. GPIO_OUT1_W1TC_REG (0x0018)

(reserved)	GPIO_OUT_DATA	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_OUT_DATA GPIO32-39 输出值清零寄存器。每一位置 1，则 GPIO_OUT1_DATA 中的相应位会清零。（只写）

Register 4.7. GPIO_ENABLE_REG (0x0020)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

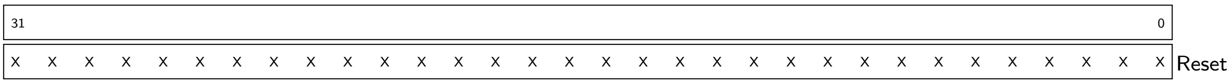
GPIO_ENABLE_REG GPIO0-31 输出使能。（读/写）

Register 4.8. GPIO_ENABLE_W1TS_REG (0x0024)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

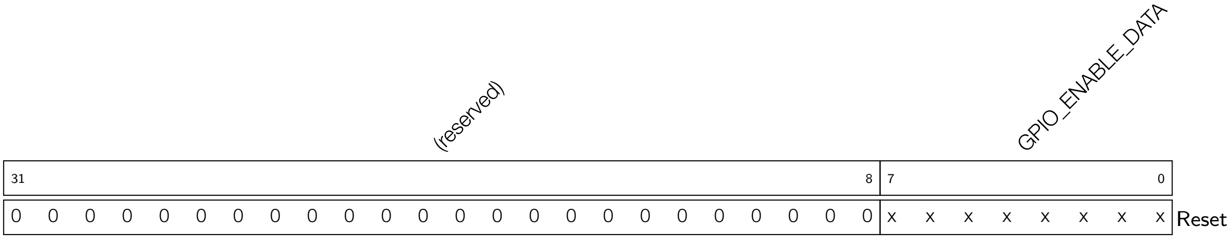
GPIO_ENABLE_W1TS_REG GPIO0-31 输出使能置位寄存器。每一位置 1，则 GPIO_ENABLE 中的相应位也置 1。（只写）

Register 4.9. GPIO_ENABLE_W1TC_REG (0x0028)



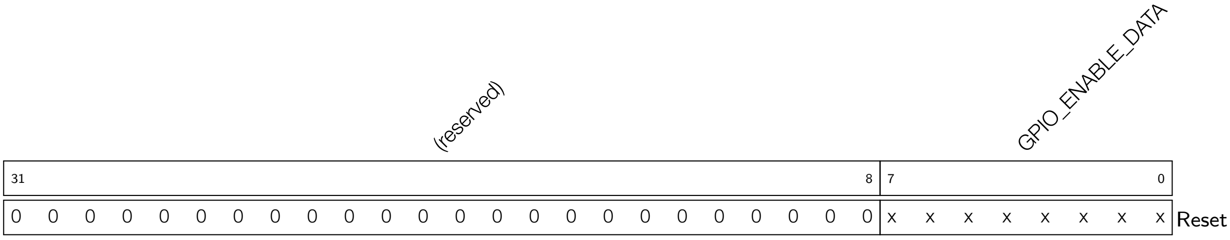
GPIO_ENABLE_W1TC_REG GPIO0-31 输出使能清零寄存器。每一位置 1，则 GPIO_ENABLE 中的相应位会清零。（只写）

Register 4.10. GPIO_ENABLE1_REG (0x002c)



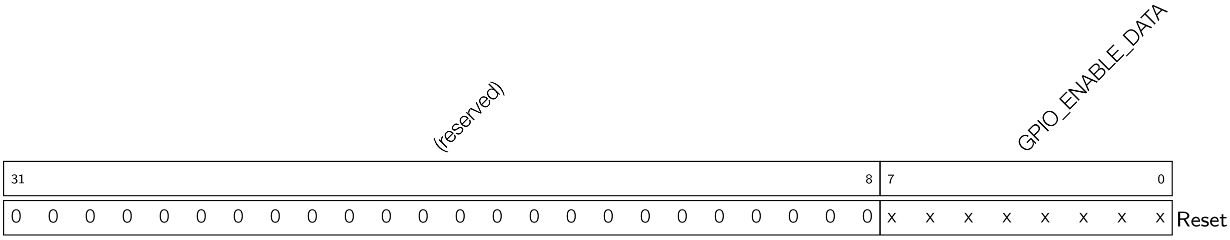
GPIO_ENABLE_DATA GPIO32-39 输出使能。（读/写）

Register 4.11. GPIO_ENABLE1_W1TS_REG (0x0030)



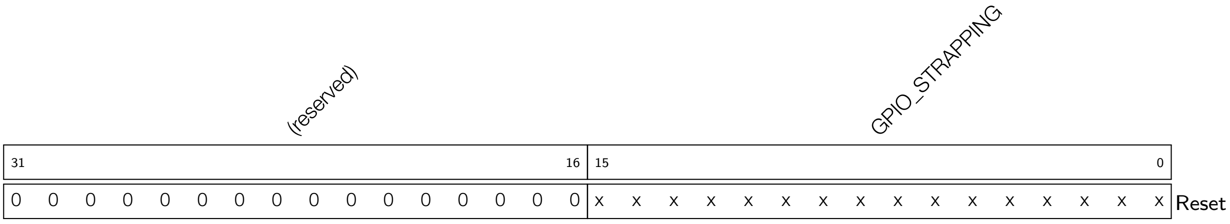
GPIO_ENABLE_DATA GPIO32-39 输出使能置位寄存器。每一位置 1，则 GPIO_ENABLE1 中的相应位也置 1。（只写）

Register 4.12. GPIO_ENABLE1_W1TC_REG (0x0034)



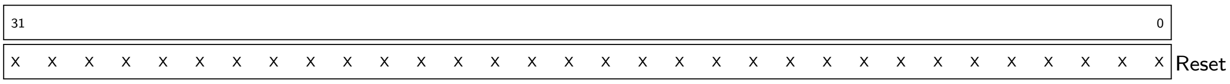
GPIO_ENABLE_DATA GPIO32-39 输出使能清零寄存器。每一位置 1，则 GPIO_ENABLE1 中的相应位会清零。（只写）

Register 4.13. GPIO_STRAP_REG (0x0038)



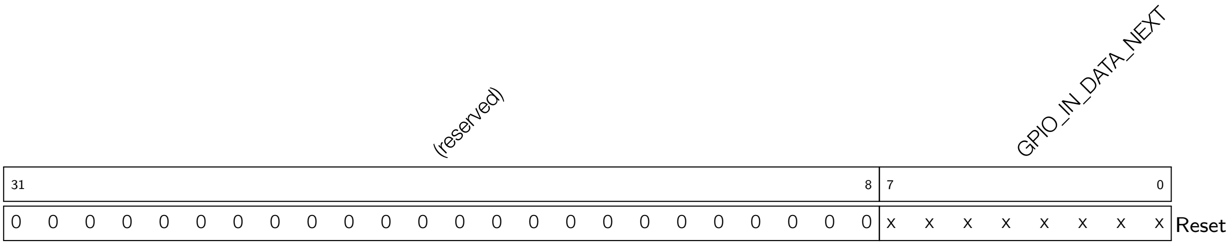
GPIO_STRAPPING GPIO strapping 结果：boot_sel_chip[5:0] 的 bit5 到 bit0 分别对应 MTDI, GPIO0, GPIO2, GPIO4, MTDO, GPIO5。

Register 4.14. GPIO_IN_REG (0x003c)



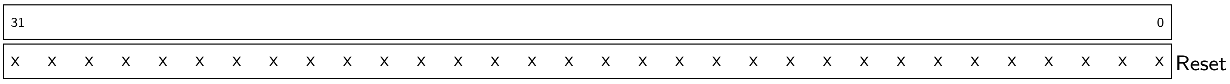
GPIO_IN_REG GPIO0-31 输入值。每个 bit 代表 pad 的片外输入值，比如片外引脚为高电平，此 bit 值应为 1，片外引脚为低电平，此 bit 值应为 0。（只读）

Register 4.15. GPIO_IN1_REG (0x0040)



GPIO_IN_DATA_NEXT GPIO32-39 输入值。每个 bit 代表 pad 的片外输入值。（只读）

Register 4.16. GPIO_STATUS_REG (0x0044)



GPIO_STATUS_REG GPIO0-31 中断状态寄存器。每个 bit 都可以作为两个 CPU 的两种中断源，同时应该把 GPIO_PIN*n*_REG 的 0-4 bit 相应的 GPIO_STATUS_INTERRUPT 的使能位置为 1。（读/写）

Register 4.17. GPIO_STATUS_W1TS_REG (0x0048)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_STATUS_W1TS_REG GPIO0-31 中断状态置位寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT 中的相应位也置 1。（只读）

Register 4.18. GPIO_STATUS_W1TC_REG (0x004c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_STATUS_W1TC_REG GPIO0-31 中断状态清除寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT 中的相应位会清零。（只写）

Register 4.19. GPIO_STATUS1_REG (0x0050)

(reserved)	GPIO_STATUS_INTERRUPT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_STATUS_INTERRUPT GPIO32-39 中断状态。（读/写）

Register 4.20. GPIO_STATUS1_W1TS_REG (0x0054)

(reserved)	GPIO_STATUS_INTERRUPT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_STATUS_INTERRUPT GPIO32-39 中断状态置位寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT1 中的相应位也置 1。（只写）

Register 4.21. GPIO_STATUS1_W1TC_REG (0x0058)

(reserved)	GPIO_STATUS_INTERRUPT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_STATUS_INTERRUPT GPIO32-39 中断状态清除寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT1 中的相应位会清零。（只写）

Register 4.22. GPIO_ACPU_INT_REG (0x0060)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ACPU_INT_REG GPIO0-31 APP CPU 中断状态寄存器。（只读）

Register 4.23. GPIO_ACPU_NMI_INT_REG (0x0064)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_ACPU_NMI_INT_REG GPIO0-31 APP CPU 非屏蔽中断状态寄存器。（只读）

Register 4.24. GPIO_PCPU_INT_REG (0x0068)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_PCPU_INT_REG GPIO0-31 PRO CPU 中断状态寄存器。（只读）

Register 4.25. GPIO_PCPU_NMI_INT_REG (0x006c)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

GPIO_PCPU_NMI_INT_REG GPIO0-31 PRO CPU 非屏蔽中断状态寄存器。（只读）

Register 4.26. GPIO_ACPU_INT1_REG (0x0074)

(reserved)	GPIO_APPCPU_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_APPCPU_INT GPIO32-39 APP CPU 中断状态寄存器。（只读）

Register 4.27. GPIO_ACPU_NMI_INT1_REG (0x0078)

(reserved)	GPIO_APPCPU_NMI_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_APPCPU_NMI_INT GPIO32-39 APP CPU 非屏蔽中断状态寄存器。（只读）

Register 4.28. GPIO_PCPU_INT1_REG (0x007c)

(reserved)	GPIO_PROCPU_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

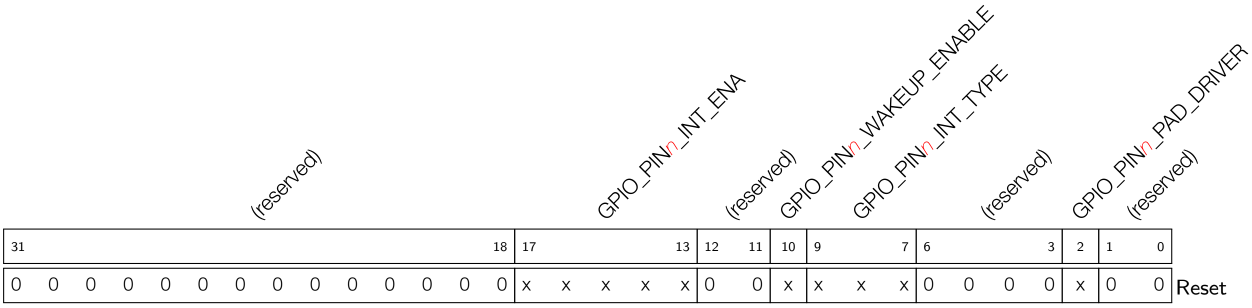
GPIO_PROCPU_INT GPIO32-39 PRO CPU 中断状态寄存器。（只读）

Register 4.29. GPIO_PCPU_NMI_INT1_REG (0x0080)

(reserved)	GPIO_PROCPU_NMI_INT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	x x x x x x x x	Reset

GPIO_PROCPU_NMI_INT GPIO32-39 PRO CPU 非屏蔽中断状态寄存器。（只读）

Register 4.30. GPIO_PIN*n*_REG (*n*: 0-39) (0x88+0x4**n*)



GPIO_PIN*n*_INT_ENA bit0：APP CPU 中断使能；（读/写）

bit1：APP CPU 非屏蔽中断使能；

bit2：PRO CPU 中断使能；

bit3：PRO CPU 非屏蔽中断使能。

GPIO_PIN*n*_WAKEUP_ENABLE GPIO 唤醒使能。只能将 CPU 从 Light-sleep 中唤醒。（读/写）

GPIO_PIN*n*_INT_TYPE 0: GPIO 中断类型；（读/写）

1：上升沿触发；

2：下降沿触发；

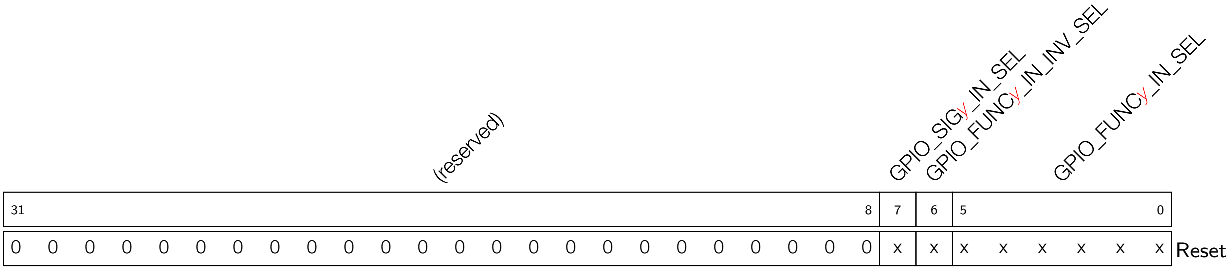
3：任一沿触发；

4：低电平触发；

5：高电平触发。

GPIO_PIN*n*_PAD_DRIVER 0：正常输出；1：开漏方式输出。（读/写）

Register 4.31. GPIO_FUNC*y*_IN_SEL_CFG_REG (*y*: 0-255) (0x130+0x4**y*)



GPIO_SIG*y*_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）

GPIO_FUNC*y*_IN_INV_SEL 反转输入值。1: 反转；0: 不反转。（读/写）

GPIO_FUNC*y*_IN_SEL 外设输入 *y* 选择控制。此位选择 1 个 GPIO 交换矩阵输入管脚中与信号连接，或者选择 0x38 与恒高电平输入信号连接或者选择 0x30 与恒低电平输入信号连接。（读/写）

Register 4.32. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-19, 21-23, 25-27, 32-33) (0x530+0x4n*)**

(reserved)	GPIO_FUNC*n*_OEN_INV_SEL	GPIO_FUNC*n*_OEN_SEL	GPIO_FUNC*n*_OUT_INV_SEL	GPIO_FUNC*n*_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 0 0 0 0	x	x	x	x x x x x x x x x	Reset

GPIO_FUNC*n*_OEN_INV_SEL 1：反转输出使能信号；0：不反转输出使能信号。（读/写）

GPIO_FUNC*n*_OEN_SEL 1：强制从 GPIO_ENABLE_REG bit *n* 中选取输出使能信号。0：使用从外设选取的输出使能信号。（读/写）

GPIO_FUNC*n*_OUT_INV_SEL 1：反转输出值；0：不反转输出值。（读/写）

GPIO_FUNC*n*_OUT_SEL GPIO 输出 *n* 的选择控制。值为 *s* (0<=*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* 做为输出值和输出使能。（读/写）

4.13.2 IO MUX 寄存器

本小节括号中的地址均为相对于 IO MUX 基地址的地址偏移量（相对地址），具体基地址见章节 1 *系统和存储器*中的表 1-6 外设地址映射。寄存器绝对地址见章节 4.12.2 *IO MUX 寄存器列表*。

Register 4.33. IO_MUX_PIN_CTRL (0x3FF49000)

(reserved)	PIN_CTRL_CLK3	PIN_CTRL_CLK2	PIN_CTRL_CLK1	
31 12	11 8	7 4	3 0	
0x0	0x0	0x0	0x0	Reset

要将 I2S*0* 外设时钟 (I2S*0*_CLK) 输出到：

CLK_OUT1，配置 PIN_CTRL[3:0] = 0x0；

CLK_OUT2，配置 PIN_CTRL[3:0] = 0x0 且 PIN_CTRL[7:4] = 0x0；

CLK_OUT3，配置 PIN_CTRL[3:0] = 0x0 且 PIN_CTRL[11:8] = 0x0。

要将 I2S*1* 外设时钟 (I2S*1*_CLK) 输出到：

CLK_OUT1，配置 PIN_CTRL[3:0] = 0xF；

CLK_OUT2，配置 PIN_CTRL[3:0] = 0xF 且 PIN_CTRL[7:4] = 0x0；

CLK_OUT3，配置 PIN_CTRL[3:0] = 0xF 且 PIN_CTRL[11:8] = 0x0。

要将 APLL 输出到

CLK_OUT1，配置 PIN_CTRL[3:0] = 0x6；

CLK_OUT2，配置 PIN_CTRL[3:0] = 0x6 且 PIN_CTRL[7:4] = 0x6；

CLK_OUT3，配置 PIN_CTRL[3:0] = 0x6 且 PIN_CTRL[11:8] = 0x6。（读/写）

说明：

- 仅支持上述配置组合，即；
  - 仅支持输出时钟：I2S0/1_CLK 和 APLL;
  - 仅支持输出管脚：CLK_OUT1-3
- CLK_OUT1-3 可在 IO_MUX Pad 列表中查询。

Register 4.34. IO_MUX_x_REG (x: GPIO0-GPIO39) (0x10+4*x)

(reserved)	MCU_SEL	FUN_DRV	FUN_IE	FUN_WPU	FUN_WPD	MCU_DRV	MCU_IE	MCU_WPU	MCU_WPD	SLP_SEL	MCU_OE	
31 15	14 12	11 10	9	8	7	6 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x2	0	0	0	0x0	0	0	0	0	0	Reset

MCU_SEL 为信号选择 IO_MUX 功能。0：选择功能 0；1：选择功能 1；以此类推。（读/写）

FUN_DRV 选择驱动强度，值越大，强度越大。对于 GPIO34-39，FUN_DRV 恒为 0。具体驱动强度请参考《ESP32 技术规格书》中“管脚清单说明”的第 8 条。（读/写）

FUN_IE pad 的输入使能。1：输入使能；0：输入关闭。（读/写）

FUN_WPU pad 的上拉使能。1：内部上拉使能；0：内部上拉关闭。由于 GPIO34-39 仅可用作输入管脚，不具备输出驱动或内置上拉/下拉电路，因此这些管脚的 FUN_WPU 恒为 0。（读/写）

FUN_WPD pad 的下拉使能。1：内部下拉使能；0：内部下拉关闭。由于 GPIO34-39 仅可用作输入，不具备输出驱动或内置上拉/下拉电路，因此这些管脚的 FUN_WPD 恒为 0。（读/写）

MCU_DRV 睡眠模式下选择 pad 的驱动强度，值越大，强度越大。（读/写）

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）

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）

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）

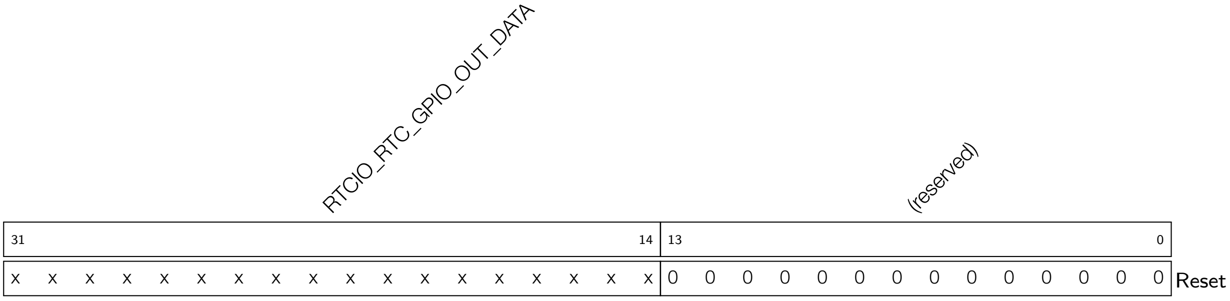
SLP_SEL pad 的睡眠模式选择。置 1 将使能睡眠模式。（读/写）

MCU_OE 睡眠模式下 pad 的输出使能。1: 输出使能；2: 输出关闭。（读/写）

4.13.3 RTC IO MUX 寄存器

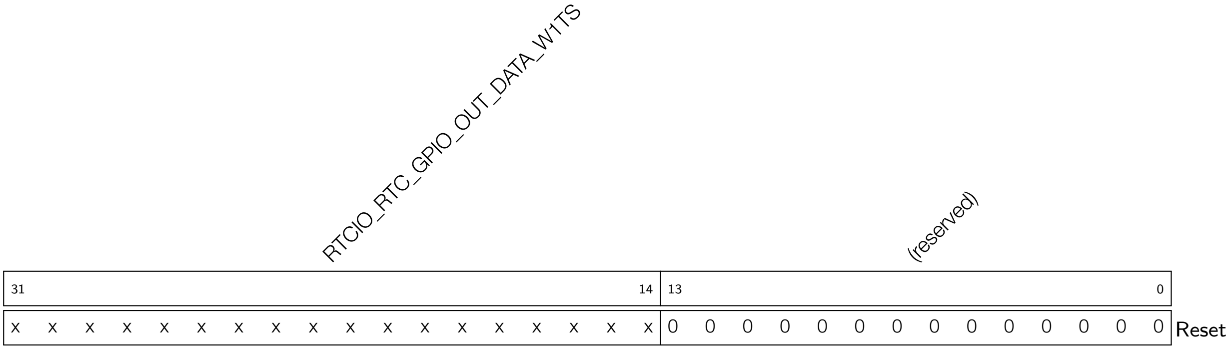
本小节括号中的地址均为相对于 (RTC 基地址 + 0x0400) 的地址偏移量（相对地址）。RTC 基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 4.12.3 *RTC IO MUX 寄存器列表*。

Register 4.35. RTCIO_RTC_GPIO_OUT_REG (0x0000)



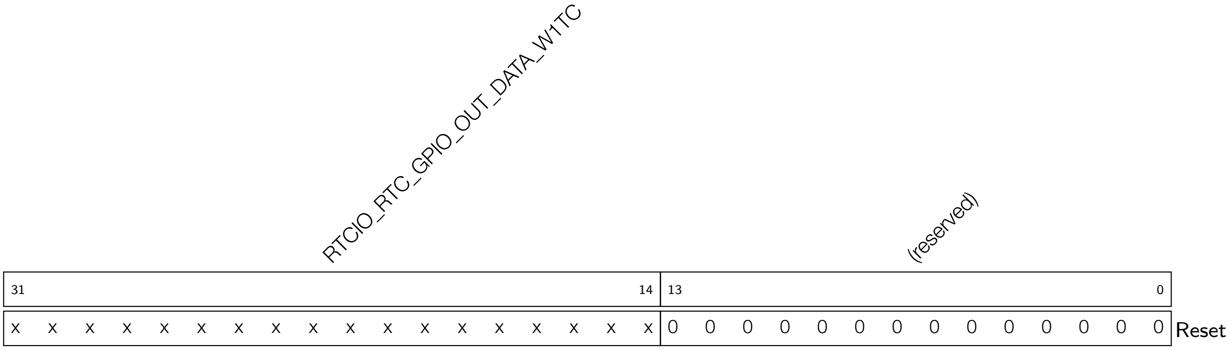
RTCIO_RTC_GPIO_OUT_DATA GPIO0-17 输出寄存器。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，以此类推。(读/写)

Register 4.36. RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0004)



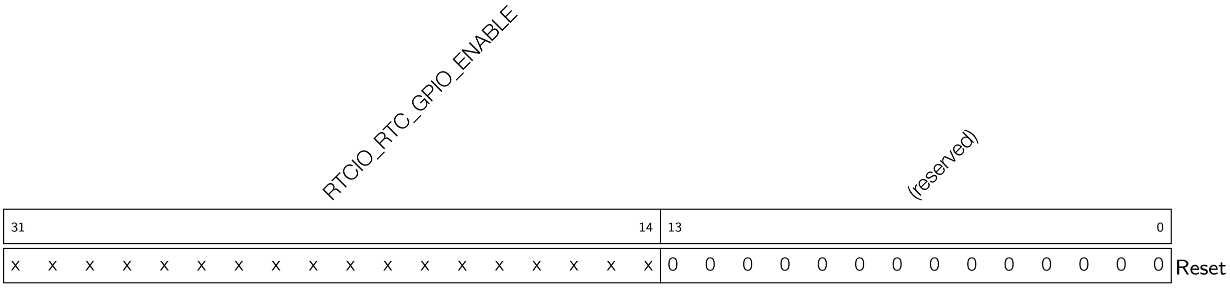
RTCIO_RTC_GPIO_OUT_DATA_W1TS GPIO0-17 输出设置寄存器。每一位置 1，则 RTCIO_RTC_GPIO_OUT 中的相应位也会置 1。(只写)

Register 4.37. RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0008)



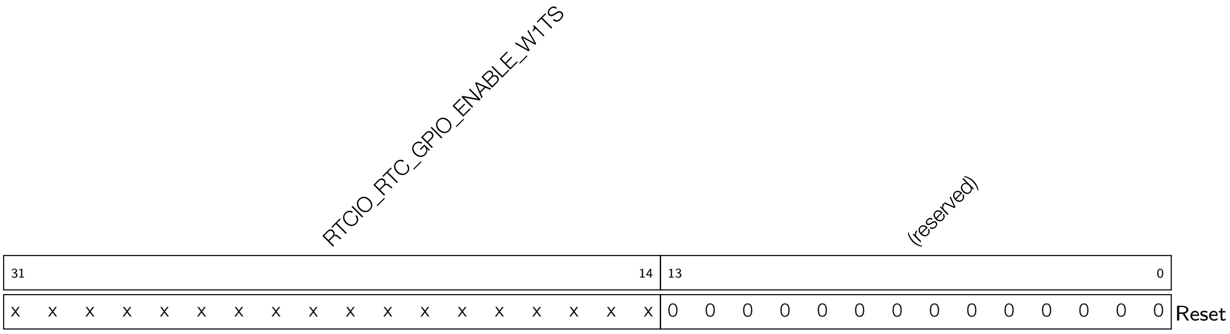
RTCIO_RTC_GPIO_OUT_DATA_W1TC GPIO0-17 输出清除寄存器。每一位置 1，则 RTCIO_RTC_GPIO_OUT 中的相应位会清零。(只写)

Register 4.38. RTCIO_RTC_GPIO_ENABLE_REG (0x000C)



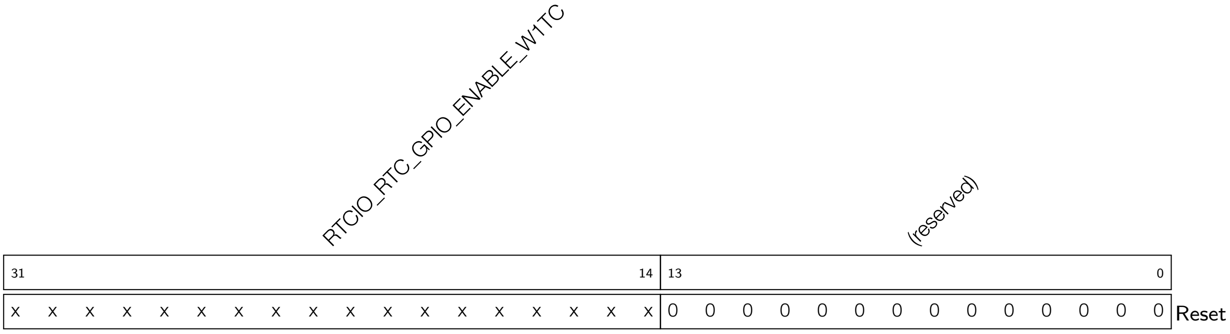
RTCIO_RTC_GPIO_ENABLE GPIO0-17 输出使能。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，以此类推。1 代表此 GPIO 为输出口。（读/写）

Register 4.39. RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0010)



RTCIO_RTC_GPIO_ENABLE_W1TS GPIO0-17 输出使能设置寄存器。每一位置 1，则 RTCIO_RTC_GPIO_ENABLE 中的相应位也会置 1。（只写）

Register 4.40. RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0014)



RTCIO_RTC_GPIO_ENABLE_W1TC GPIO0-17 输出使能清除寄存器。每一位置 1，则 RTCIO_RTC_GPIO_ENABLE 中的相应位会清零。（只写）

Register 4.41. RTCIO_RTC_GPIO_STATUS_REG (0x0018)

RTCIO_RTC_GPIO_STATUS_INT	(reserved)	
31 … 14	13 … 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_STATUS_INT GPIO0-17 中断状态。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，以此类推。此寄存器应同时和 RTCIO_RTC_GPIO_PIN*n*_REG 的 RTCIO_RTC_GPIO_PIN*n*_INT_TYPE 中断类型配合使用，1 代表有相应中断，0 代表没有中断。(读/写)

Register 4.42. RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x001C)

RTCIO_RTC_GPIO_STATUS_INT_W1TS	(reserved)	
31 … 14	13 … 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_STATUS_INT_W1TS GPIO0-17 中断设置寄存器。每一位置 1，则 RTCIO_RTC_GPIO_STATUS_INT 中的相应位也会置 1。(只写)

Register 4.43. RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0020)

RTCIO_RTC_GPIO_STATUS_INT_W1TC	(reserved)	
31 … 14	13 … 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_STATUS_INT_W1TC GPIO0-17 中断清除寄存器。每一位置 1，则 RTCIO_RTC_GPIO_STATUS_INT 中的相应位会清零。(只写)

Register 4.44. RTCIO_RTC_GPIO_IN_REG (0x0024)

RTCIO_RTC_GPIO_IN_NEXT	(reserved)	
31 14	13 0	
x x x x x x x x x x x x x x x x x x	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_RTC_GPIO_IN_NEXT GPIO0-17 输入值。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，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 4.45. RTCIO_RTC_GPIO_PIN*n*_REG (*n*: 0-17) (28+4**n*)

(reserved)	RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE	RTCIO_RTC_GPIO_PIN*n*_INT_TYPE	(reserved)	RTCIO_RTC_GPIO_PIN*n*_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 0 0	x	x x x	0 0 0 0	x	0 0	Reset

RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE GPIO 唤醒使能。只能将处于 Light-sleep 的 ESP32 唤醒。（读/写）

RTCIO_RTC_GPIO_PIN*n*_INT_TYPE GPIO 中断类型选择。（读/写）

0：GPIO 中断关闭；
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_RTC_GPIO_PIN*n*_PAD_DRIVER Pad 驱动器选择。0：正常输出；1：开漏模式。（读/写）

Register 4.46. RTCIO_DIG_PAD_HOLD_REG (0x0074)

31 0	
0	Reset

RTCIO_DIG_PAD_HOLD_REG 选择哪一个数字 pad 置于 hold 状态。0: 允许正常操作；1: 置于 hold 状态。（读/写）

名称	描述
Bit[0]	pad U0RTD Hold 功能使能位
Bit[1]	pad U0TXD Hold 功能使能位
Bit[2]	pad SD_CLK Hold 功能使能位
Bit[3]	pad SD_DATA0 Hold 功能使能位
Bit[4]	pad SD_DATA1 Hold 功能使能位
Bit[5]	pad SD_DATA2 Hold 功能使能位
Bit[6]	pad SD_DATA3 Hold 功能使能位
Bit[7]	pad SD_CMD Hold 功能使能位
Bit[8]	pad GPIO5 Hold 功能使能位
Bit[9]	pad GPIO16 Hold 功能使能位
Bit[10]	pad GPIO17 Hold 功能使能位
Bit[11]	pad GPIO18 Hold 功能使能位
Bit[12]	pad GPIO19 Hold 功能使能位
Bit[13]	pad GPIO20 Hold 功能使能位
Bit[14]	pad GPIO21 Hold 功能使能位
Bit[15]	pad GPIO22 Hold 功能使能位
Bit[16]	pad GPIO23 Hold 功能使能位

Register 4.47. RTCIO_SENSOR_PADS_REG (0x007C)

RTCIO_SENSOR_SENSE1_HOLD	RTCIO_SENSOR_SENSE2_HOLD	RTCIO_SENSOR_SENSE3_HOLD	RTCIO_SENSOR_SENSE4_HOLD	RTCIO_SENSOR_SENSE1_MUX_SEL	RTCIO_SENSOR_SENSE2_MUX_SEL	RTCIO_SENSOR_SENSE3_MUX_SEL	RTCIO_SENSOR_SENSE4_MUX_SEL	RTCIO_SENSOR_SENSE1_FUN_SEL	RTCIO_SENSOR_SENSE1_SLP_SEL	RTCIO_SENSOR_SENSE1_SLP_IE	RTCIO_SENSOR_SENSE1_FUN_IE	RTCIO_SENSOR_SENSE2_FUN_SEL	RTCIO_SENSOR_SENSE2_SLP_SEL	RTCIO_SENSOR_SENSE2_SLP_IE	RTCIO_SENSOR_SENSE2_FUN_IE	RTCIO_SENSOR_SENSE3_FUN_SEL	RTCIO_SENSOR_SENSE3_SLP_SEL	RTCIO_SENSOR_SENSE3_SLP_IE	RTCIO_SENSOR_SENSE3_FUN_IE	RTCIO_SENSOR_SENSE4_FUN_SEL	RTCIO_SENSOR_SENSE4_SLP_SEL	RTCIO_SENSOR_SENSE4_SLP_IE	RTCIO_SENSOR_SENSE4_FUN_IE	(reserved)
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 0
0	0	0	0	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

RTCIO_SENSOR_SENSE*n*_HOLD 置 1 保持传感器 *n* 的输出值。0: 正常操作。(读/写)

RTCIO_SENSOR_SENSE*n*_MUX_SEL 1: 连接传感器 *n* 与 RTC 模块；0: 连接传感器 *n* 与数字 IO_MUX。(读/写)

RTCIO_SENSOR_SENSE*n*_FUN_SEL 选择 RTC IO_MUX 功能。0: 选择 Function 0。(读/写)

RTCIO_SENSOR_SENSE*n*_SLP_SEL pad 的睡眠模式选择信号。pad 进入睡眠模式时应将此 bit 置 1 。(读/写)

RTCIO_SENSOR_SENSE*n*_SLP_IE 睡眠模式下 pad 的输入使能。1: 使能；0: 关闭。(读/写)

RTCIO_SENSOR_SENSE*n*_FUN_IE pad 的输入使能。1: 使能；0: 关闭。(读/写)

Register 4.48. RTCIO_ADC_PAD_REG (0x0080)

RTCIO_ADC_ADC1_HOLD	RTCIO_ADC_ADC2_HOLD	RTCIO_ADC_ADC1_MUX_SEL	RTCIO_ADC_ADC2_MUX_SEL	RTCIO_ADC_ADC1_FUN_SEL	RTCIO_ADC_ADC1_SLP_SEL	RTCIO_ADC_ADC1_SLP_IE	RTCIO_ADC_ADC1_FUN_IE	RTCIO_ADC_ADC2_FUN_SEL	RTCIO_ADC_ADC2_SLP_SEL	RTCIO_ADC_ADC2_SLP_IE	RTCIO_ADC_ADC2_FUN_IE	(reserved)
31	30	29	28	27 26	25	24	23	22 21	20	19	18	17 0
0	0	0	0	0	0	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

RTCIO_ADC_ADC*n*_HOLD 置 1 将保留 pad 的输出值，0: 正常操作。(读/写)

RTCIO_ADC_ADC*n*_MUX_SEL 0: 连接 pad 与数字 IO_MUX。1: 连接 pad 与 RTC 模块。(读/写)

RTCIO_ADC_ADC*n*_FUN_SEL 选择 RTC 功能。0: 选择 Function 0；3: 选择 Function 1。(读/写)

RTCIO_ADC_ADC*n*_SLP_SEL pad 的睡眠模式选择信号。置 1 pad 将进入睡眠模式。(读/写)

RTCIO_ADC_ADC*n*_SLP_IE 睡眠模式下 pad 的输入使能。1: 使能；0: 关闭。(读/写)

RTCIO_ADC_ADC*n*_FUN_IE pad 的输入使能。1: 使能；0: 关闭。(读/写)

Register 4.49. RTCIO_PAD_DAC1_REG (0x0084)

RTCIO_PAD_PDAC1_DRV	RTCIO_PAD_PDAC1_HOLD	RTCIO_PAD_PDAC1_RDE	RTCIO_PAD_PDAC1_RUE	RTCIO_PAD_PDAC1_DAC	RTCIO_PAD_PDAC1_XPD_DAC	RTCIO_PAD_PDAC1_MUX_SEL	RTCIO_PAD_PDAC1_FUN_SEL	RTCIO_PAD_PDAC1_SLP_SEL	RTCIO_PAD_PDAC1_SLP_IE	RTCIO_PAD_PDAC1_SLP_OE	RTCIO_PAD_PDAC1_FUN_IE	RTCIO_PAD_PDAC1_DAC_XPD_FORCE	(reserved)	
31 30	29	28	27	26 19	18	17	16 15	14	13	12	11	10	9 0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_PAD_PDAC1_DRV 选择 pad 的驱动强度。（读/写）

RTCIO_PAD_PDAC1_HOLD 置 1 将保留 pad 的输出值，0: 正常操作。（读/写）

RTCIO_PAD_PDAC1_RDE 1：下拉使能；0: 下拉关闭。（读/写）

RTCIO_PAD_PDAC1_RUE 1：上拉使能；0: 上拉关闭。（读/写）

RTCIO_PAD_PDAC1_DAC PAD DAC1 输出值。（读/写）

RTCIO_PAD_PDAC1_XPD_DAC 给 DAC1 上电。一般而言，DAC 上电时，PDAC1 应该置为三态，即 IE=0、OE=0、RDE=0、RUE=0。（读/写）

RTCIO_PAD_PDAC1_MUX_SEL 0：连接 pad 与数字 IO_MUX。1：连接 RTC 模块。（读/写）

RTCIO_PAD_PDAC1_FUN_SEL pad 的功能选择信号。（读/写）

RTCIO_PAD_PDAC1_SLP_SEL pad 的睡眠模式选择信号。置 1 则 pad 进入睡眠模式。（读/写）

RTCIO_PAD_PDAC1_SLP_IE 睡眠模式下 pad 的输入使能。1: 使能；0: 关闭。（读/写）

RTCIO_PAD_PDAC1_SLP_OE pad 的输出使能。1: 使能；0: 关闭。（读/写）

RTCIO_PAD_PDAC1_FUN_IE pad 的输入使能。1: 使能；0: 关闭。（读/写）

RTCIO_PAD_PDAC1_DAC_XPD_FORCE 给 DAC1 上电。一般而言，DAC 上电时，PDAC1 应该置为三态，即 IE=0、OE=0、RDE=0、RUE=0。（读/写）

Register 4.50. RTCIO_PAD_DAC2_REG (0x0088)

RTCIO_PAD_PDAC2_DRV	RTCIO_PAD_PDAC2_HOLD	RTCIO_PAD_PDAC2_RDE	RTCIO_PAD_PDAC2_RUE	RTCIO_PAD_PDAC2_DAC	RTCIO_PAD_PDAC2_XPD_DAC	RTCIO_PAD_PDAC2_MUX_SEL	RTCIO_PAD_PDAC2_FUN_SEL	RTCIO_PAD_PDAC2_SLP_SEL	RTCIO_PAD_PDAC2_SLP_IE	RTCIO_PAD_PDAC2_SLP_OE	RTCIO_PAD_PDAC2_FUN_IE	RTCIO_PAD_PDAC2_DAC_XPD_FORCE	(reserved)	
31 30	29	28	27	26 19	18	17	16 15	14	13	12	11	10	9 0	
2	0	0	0	0	0	0	0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_PAD_PDAC2_DRV 选择 pad 的驱动强度。(读/写)

RTCIO_PAD_PDAC2_HOLD 置 1 将保留 pad 的输出值，0：正常操作。(读/写)

RTCIO_PAD_PDAC2_RDE 1：下拉使能；0：下拉关闭。(读/写)

RTCIO_PAD_PDAC2_RUE 1：上拉使能；0：上拉关闭。(读/写)

RTCIO_PAD_PDAC2_DAC PAD DAC2 输出值。(读/写)

RTCIO_PAD_PDAC2_XPD_DAC 给 DAC2 上电。一般而言，DAC 上电时，PDAC2 应该置为三态，即 IE=0, OE=0, RDE=0, RUE=0。(读/写)

RTCIO_PAD_PDAC2_MUX_SEL 0：连接 pad 与数字 IO_MUX。1：连接 RTC 模块。(读/写)

RTCIO_PAD_PDAC2_FUN_SEL 选择 pad 的 RTC 功能，0：选择 Function 0。(读/写)

RTCIO_PAD_PDAC2_SLP_SEL pad 的睡眠模式选择信号。置 1 则 pad 进入睡眠模式。(读/写)

RTCIO_PAD_PDAC2_SLP_IE 睡眠模式下 pad 的输入使能。1：使能；0：关闭。(读/写)

RTCIO_PAD_PDAC2_SLP_OE pad 的输出使能。1：使能；0：关闭。(读/写)

RTCIO_PAD_PDAC2_FUN_IE pad 的输入使能。1：使能；0: 关闭。(读/写)

RTCIO_PAD_PDAC2_DAC_XPD_FORCE 给 DAC2 上电。一般而言，DAC 上电时，PDAC2 应该置为三态，即 IE=0, OE=0, RDE=0, RUE=0。(读/写)

Register 4.51. RTCIO_XTAL_32K_PAD_REG (0x008C)

RTCIO_XTAL_X32N_DRV	RTCIO_XTAL_X32N_HOLD	RTCIO_XTAL_X32N_RDE	RTCIO_XTAL_X32N_RUE	RTCIO_XTAL_X32P_DRV	RTCIO_XTAL_X32P_HOLD	RTCIO_XTAL_X32P_RDE	RTCIO_XTAL_X32P_RUE	RTCIO_XTAL_DAC_XTAL_32K	RTCIO_XTAL_XPD_XTAL_32K	RTCIO_XTAL_X32N_MUX_SEL	RTCIO_XTAL_X32P_MUX_SEL	RTCIO_XTAL_X32N_FUN_SEL	RTCIO_XTAL_X32N_SLP_SEL	RTCIO_XTAL_X32N_SLP_IE	RTCIO_XTAL_X32N_SLP_OE	RTCIO_XTAL_X32N_FUN_IE	RTCIO_XTAL_X32P_FUN_SEL	RTCIO_XTAL_X32P_SLP_SEL	RTCIO_XTAL_X32P_SLP_IE	RTCIO_XTAL_X32P_SLP_OE	RTCIO_XTAL_X32P_FUN_IE	RTCIO_XTAL_DRES_XTAL_32K	RTCIO_XTAL_DBIAS_XTAL_32K	(reserved)	
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	
2	0	0	0	2	0	0	0	0 1	0	0	0	0	0	0	0	0	0	0	0	0	0	1 0	0 0	0	Reset

RTCIO_XTAL_X32N_DRV 选择 pad 的驱动强度。（读/写）

RTCIO_XTAL_X32N_HOLD 置 1 将保留 pad 的输出值，0：正常操作。（读/写）

RTCIO_XTAL_X32N_RDE 1：下拉使能；0: 下拉关闭。（读/写）

RTCIO_XTAL_X32N_RUE 1：上拉使能；0: 上拉关闭。（读/写）

RTCIO_XTAL_X32P_DRV 选择 pad 的驱动强度。（读/写）

RTCIO_XTAL_X32P_HOLD 置 1 将保留 pad 的输出值，0：正常操作。（读/写）

RTCIO_XTAL_X32P_RDE 1：下拉使能；0：下拉关闭。（读/写）

RTCIO_XTAL_X32P_RUE 1：上拉使能；0：上拉关闭。（读/写）

RTCIO_XTAL_DAC_XTAL_32K 32K XTAL 偏置电流 DAC 值。（读/写）

RTCIO_XTAL_XPD_XTAL_32K 给 32 kHz 晶振通电。（读/写）

RTCIO_XTAL_X32N_MUX_SEL 0: 连接 X32N 与数字 IO_MUX；1：连接 RTC 模块。（读/写）

RTCIO_XTAL_X32P_MUX_SEL 0：连接 X32P pad 与数字 IO_MUX；1：连接 RTC 模块。（读/写）

RTCIO_XTAL_X32N_FUN_SEL 选择 pad 的 RTC 功能，0：选择 Function 0。（读/写）

RTCIO_XTAL_X32N_SLP_SEL pad 的睡眠模式选择信号。置 1 则 pad 进入睡眠模式。（读/写）

RTCIO_XTAL_X32N_SLP_IE 睡眠模式下 pad 的输入使能。1：使能；0：关闭。（读/写）

RTCIO_XTAL_X32N_SLP_OE pad 的输出使能。1：使能；0：关闭。（读/写）

RTCIO_XTAL_X32N_FUN_IE pad 的输入使能。1：使能；0：关闭。（读/写）

RTCIO_XTAL_X32P_FUN_SEL 选择 pad 的 RTC 功能，0：选择 Function 0；1：选择 Function 1。（读/写）

RTCIO_XTAL_X32P_SLP_SEL 睡眠模式选择，置 1 则 pad 进入睡眠模式。（读/写）

RTCIO_XTAL_X32P_SLP_IE 睡眠模式下 pad 的输入使能。1：使能；0：关闭。（读/写）

RTCIO_XTAL_X32P_SLP_OE 睡眠模式下 pad 的输出使能。1：使能；0：关闭。（读/写）

RTCIO_XTAL_X32P_FUN_IE pad 的输入使能。1：使能；0：关闭。（读/写）

RTCIO_XTAL_DRES_XTAL_32K 32K XTAL 电阻偏置控制。（读/写）

RTCIO_XTAL_DBIAS_XTAL_32K 32K XTAL 自偏置参考控制。（读/写）

Register 4.52. RTCIO_TOUCH_CFG_REG (0x0090)

RTCIO_TOUCH_XPD_BIAS	RTCIO_TOUCH_DREFH	RTCIO_TOUCH_DREFL	RTCIO_TOUCH_DRANGE	RTCIO_TOUCH_DCUR	(reserved)	
31	30 29	28 27	26 25	24 23	22 0	
0	1 1	0 0	1 1	0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_XPD_BIAS 触摸传感器偏置上电位。1：上电；0：断电。（读/写）

RTCIO_TOUCH_DREFH 触摸传感器波顶电压。（读/写）

RTCIO_TOUCH_DREFL 触摸传感器波底电压。（读/写）

RTCIO_TOUCH_DRANGE 触摸传感器电压变动范围。（读/写）

RTCIO_TOUCH_DCUR 触摸传感器偏置电流。当 BIAS_SLEEP 使能时，可以进行设置。（读/写）

Register 4.53. RTCIO_TOUCH_PADn_REG (n: 0-7) (94+4*n)

RTCIO_TOUCH_PADn_HOLD	RTCIO_TOUCH_PADn_DRV	RTCIO_TOUCH_PADn_RDE	RTCIO_TOUCH_PADn_RUE	(reserved)	RTCIO_TOUCH_PADn_DAC	RTCIO_TOUCH_PADn_START	RTCIO_TOUCH_PADn_TIE_OPT	RTCIO_TOUCH_PADn_XPD	RTCIO_TOUCH_PADn_MUX_SEL	RTCIO_TOUCH_PADn_FUN_SEL	RTCIO_TOUCH_PADn_SLP_SEL	RTCIO_TOUCH_PADn_SLP_IE	RTCIO_TOUCH_PADn_SLP_OE	RTCIO_TOUCH_PADn_FUN_IE	RTCIO_TOUCH_PADn_TO_GPIO	(reserved)	
31	30 29	28	27	26	25 23	22	21	20	19	18 17	16	15	14	13	12	11 0	
0	1 0	1	0	0	1 0 0	0	0	0	0	0 0	0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_PADn_HOLD 写 1 保留 pad 的输出值。（读/写）

RTCIO_TOUCH_PADn_DRV 选择 pad 的驱动强度。值越大，强度越大。具体驱动强度请参考《ESP32 技术规格书》中 > 附录 A.1 管脚清单说明 > 第 8 条。（读/写）

RTCIO_TOUCH_PADn_RDE 1：下拉使能；0：下拉关闭。（读/写）

RTCIO_TOUCH_PADn_RUE 1：上拉使能；0：上拉关闭。（读/写）

RTCIO_TOUCH_PADn_DAC 触摸传感器斜率控制。每个触摸 pad 为 3-bit，默认为 b’100。（读/写）

RTCIO_TOUCH_PADn_START 写 1 启动触摸传感器。（读/写）

RTCIO_TOUCH_PADn_TIE_OPT 默认触摸传感器初始电压位。

0：0 V

1：VDD_RTC 的电压

（读/写）

RTCIO_TOUCH_PADn_XPD 写 1 给触摸传感器上电。（读/写）

RTCIO_TOUCH_PADn_MUX_SEL 选择由 RTC IO_MUX 或者 IO_MUX 控制 RTC pad 的 IE/OE/RUE/RDE 状态。

1：选择 RTC IO_MUX

0：选择 IO_MUX

（读/写）

RTCIO_TOUCH_PADn_FUN_SEL 选择 RTC pad 的功能。

0：RTC Function 0。

1：保留。

2：保留。

3：RTC Function 1。

（读/写）

RTCIO_TOUCH_PADn_SLP_SEL pad 的睡眠模式选择信号。置 1 则 pad 进入睡眠模式。（读/写）

RTCIO_TOUCH_PADn_SLP_IE 睡眠模式下 (SLP_SEL = 1) pad 的输入使能。

1：使能

0：关闭

(读/写)

见下页

Register 4.53. RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7) (94+4**n*)

接上页

RTCIO_TOUCH_PAD*n*_SLP_OE 睡眠模式下 (SLP_SEL = 1) pad 的输出使能。

1：使能

0：关闭

(读/写)

RTCIO_TOUCH_PAD*n*_FUN_IE 正常工作模式下 (SLP_SEL = 0) pad 的输入使能。

1：使能

0：关闭

(读/写)

RTCIO_TOUCH_PAD*n*_TO_GPIO 选择 IO_MUX 的输入。

1：从模拟电路输入

0：从数字电路输入

（读/写）

Register 4.54. RTCIO_TOUCH_PAD*m*_REG (*m* = 8, 9) (0xB4, 0xB8)

(reserved)	RTCIO_TOUCH_PAD*m*_DAC	RTCIO_TOUCH_PAD*m*_START	RTCIO_TOUCH_PAD*m*_TIE_OPT	RTCIO_TOUCH_PAD*m*_XPD	RTCIO_TOUCH_PAD*m*_TO_GPIO	(reserved)	
31 26	25 23	22	21	20	19	18 0	
0 0 0 0 0 0	1 0 0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTCIO_TOUCH_PAD*m*_DAC 触摸传感器斜率控制。每个触摸 pad 为 3-bit，默认为 b’100。（读/写）

RTCIO_TOUCH_PAD*m*_START 写 1 启动触摸传感器。（读/写）

RTCIO_TOUCH_PAD*m*_TIE_OPT 默认触摸传感器初始电压位。

0：0 V

1：VDD_RTC 的电压

（读/写）

RTCIO_TOUCH_PAD*m*_XPD 写 1 给触摸传感器上电。（读/写）

RTCIO_TOUCH_PAD*m*_TO_GPIO 选择 IO_MUX 的输入。

1：从模拟电路输入

0：从数字电路输入

（读/写）

Register 4.55. RTCIO_EXT_WAKEUP0_REG (0x00BC)

RTCIO_EXT_WAKEUP0_SEL	(reserved)	
31 27	26 0	
0	0 0 0 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

RTCIO_EXT_WAKEUP0_SEL GPIO[0-17] 可以用于将芯片从睡眠模式中唤醒。此寄存器选择 pad 源将芯片从 Deep/Light-sleep 模式中唤醒。0：选择 GPIO0；1：选择 GPIO2，以此类推。（读/写）

Register 4.56. RTCIO_XTL_EXT_CTR_REG (0x00C0)

RTCIO_XTL_EXT_CTR_SEL	(reserved)	
31 27	26 0	
0	0 0 0 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

RTCIO_XTL_EXT_CTR_SEL 选择睡眠模式下外部晶振断电使能源。0：选择 GPIO0；1：选择 GPIO2，以此类推。被选择管脚的值异或 RTC_CNTL_XTL_EXT_CTR_LV 上的逻辑值是晶振断电使能信号。（读/写）

Register 4.57. RTCIO_SAR_I2C_IO_REG (0x00C4)

RTCIO_SAR_I2C_SDA_SEL	RTCIO_SAR_I2C_SCL_SEL	(reserved)	
31 30	29 28	27 0	
0	0	0 0 0 0 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

RTCIO_SAR_I2C_SDA_SEL 选择另一个 pad 作为 RTC I2C SDA 信号。0：选择 TOUCH_PAD[1]；1：选择 TOUCH_PAD[3]。默认值为 0。（读/写）

RTCIO_SAR_I2C_SCL_SEL 选择另一个 pad 作为 RTC I2C SCL signal。0：选择 TOUCH_PAD[0]；1：选择 TOUCH_PAD[2]。默认值为 0。（读/写）

5 DPort 寄存器

5.1 概述

ESP32 集成了丰富的外设，通过 DPort 寄存器控制时钟门控，功耗管理，外设以及系统核心模块的配置，可以使得系统在保持最佳性能的同时将功耗降到最低。系统通过 DPort 寄存器中的配置寄存器对各个模块进行配置。

5.2 主要特性

DPort 寄存器包含了多个对应外设和模块的寄存器：

- 系统和存储器
- 复位和时钟
- 中断矩阵
- DMA
- MPU/MMU
- APP_CPU 控制器
- 外设时钟门控和复位

5.3 功能描述

5.3.1 系统和存储器寄存器

系统和存储器寄存器（见表 5.4）用于系统和存储器，例如 cache 配置和存储器映射。寄存器描述详见章节系统和存储器。

5.3.2 复位和时钟寄存器

复位和时钟寄存器（见表 5.4）用于复位和时钟。寄存器描述详见章节复位和时钟。

5.3.3 中断矩阵寄存器

中断矩阵寄存器（见表 5.4）用于通过中断矩阵配置和映射中断。寄存器描述详见章节中断矩阵 *(INTERRUPT)*。

5.3.4 DMA 寄存器

DMA 寄存器（见表 5.4）用于 SPI DMA 配置。寄存器描述详见章节 *DMA 控制器 (DMA)*。

5.3.5 MPU/MMU 寄存器

MPU/MMU 寄存器（见表 5.4）用于 MPU/MMU 配置与操作控制。寄存器描述详见章节 存储器管理和保护单元 *(MMU, MPU)*。

5.3.6 APP_CPU 控制器寄存器

APP_CPU 控制器寄存器（见表 5.4）可用于 APP_CPU 的基本配置，例如暂停任务的执行，以及设置从 ROM code 启动后的跳转地址。寄存器描述详见章节 5.5。需要注意的是，复位寄存器无法通过硬件清除。

5.3.7 外设时钟门控和复位

以下 DPort 寄存器用于控制外设时钟门控和复位，相应位分别为不同外设的门控使能和复位使能。寄存器描述详见章节 5.5。

- DPORT_PERI_CLK_EN_REG
- DPORT_PERI_RST_EN_REG
- DPORT_PERIP_CLK_EN_REG
- DPORT_PERIP_RST_EN_REG
- DPORT_WIFI_CLK_EN_REG
- DPORT_WIFI_RST_EN_REG

需要注意的是：

- 外设时钟门控和复位寄存器均为高电平有效。
- 复位寄存器无法通过硬件清除，因此软件将外设复位后需要清除复位寄存器。
- ESP32 具有低功耗特性，因此有些外设时钟默认为关闭状态。在启用这些外设之前，必须将外设的时钟打开，即把相应的 CLK_EN 位置为 1，并且解除外设的复位状态，即把相应的 RST_EN 位置为 0。

5.4 寄存器列表

名称	描述	地址	访问
系统和存储器寄存器			
DPORT_PRO_BOOT_REMAP_CTRL_REG	PRO_CPU 存储器重映射模式	0x3FF00000	读/写
DPORT_APP_BOOT_REMAP_CTRL_REG	APP_CPU 存储器重映射模式	0x3FF00004	读/写
DPORT_CACHE_MUX_MODE_REG	两个 cache 同时占用内存的模式	0x3FF0007C	读/写
复位和时钟寄存器			
DPORT_CPU_PER_CONF_REG	选择 CPU 时钟	0x3FF0003C	读/写
中断矩阵寄存器			
DPORT_CPU_INTR_FROM_CPU_0_REG	两个 CPU 的 Interrupt 0	0x3FF000DC	读/写
DPORT_CPU_INTR_FROM_CPU_1_REG	两个 CPU 的 Interrupt 1	0x3FF000E0	读/写
DPORT_CPU_INTR_FROM_CPU_2_REG	两个 CPU 的 Interrupt 2	0x3FF000E4	读/写
DPORT_CPU_INTR_FROM_CPU_3_REG	两个 CPU 的 Interrupt 3	0x3FF000E8	读/写
DPORT_PRO_INTR_STATUS_REG_0_REG	PRO_CPU 中断状态 0	0x3FF000EC	只读
DPORT_PRO_INTR_STATUS_REG_1_REG	PRO_CPU 中断状态 1	0x3FF000F0	只读
DPORT_PRO_INTR_STATUS_REG_2_REG	PRO_CPU 中断状态 2	0x3FF000F4	只读
DPORT_APP_INTR_STATUS_REG_0_REG	APP_CPU 中断状态 0	0x3FF000F8	只读
DPORT_APP_INTR_STATUS_REG_1_REG	APP_CPU 中断状态 1	0x3FF000FC	只读
DPORT_APP_INTR_STATUS_REG_2_REG	APP_CPU 中断状态 2	0x3FF00100	只读
DPORT_PRO_MAC_INTR_MAP_REG	中断映射	0x3FF00104	读/写
DPORT_PRO_MAC_NMI_MAP_REG	中断映射	0x3FF00108	读/写
DPORT_PRO_BB_INT_MAP_REG	中断映射	0x3FF0010C	读/写
DPORT_PRO_BT_MAC_INT_MAP_REG	中断映射	0x3FF00110	读/写
DPORT_PRO_BT_BB_INT_MAP_REG	中断映射	0x3FF00114	读/写
DPORT_PRO_BT_BB_NMI_MAP_REG	中断映射	0x3FF00118	读/写
DPORT_PRO_RWBT_IRQ_MAP_REG	中断映射	0x3FF0011C	读/写
DPORT_PRO_RWBLE_IRQ_MAP_REG	中断映射	0x3FF00120	读/写
DPORT_PRO_RWBT_NMI_MAP_REG	中断映射	0x3FF00124	读/写
DPORT_PRO_RWBLE_NMI_MAP_REG	中断映射	0x3FF00128	读/写
DPORT_PRO_SLC0_INTR_MAP_REG	中断映射	0x3FF0012C	读/写
DPORT_PRO_SLC1_INTR_MAP_REG	中断映射	0x3FF00130	读/写
DPORT_PRO_UHCI0_INTR_MAP_REG	中断映射	0x3FF00134	读/写
DPORT_PRO_UHCI1_INTR_MAP_REG	中断映射	0x3FF00138	读/写
DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG	中断映射	0x3FF0013C	读/写
DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG	中断映射	0x3FF00140	读/写
DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG	中断映射	0x3FF00144	读/写
DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG	中断映射	0x3FF00148	读/写
DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG	中断映射	0x3FF0014C	读/写
DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG	中断映射	0x3FF00150	读/写
DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG	中断映射	0x3FF00154	读/写
DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG	中断映射	0x3FF00158	读/写
DPORT_PRO_GPIO_INTERRUPT_MAP_REG	中断映射	0x3FF0015C	读/写
DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG	中断映射	0x3FF00160	读/写

名称	描述	地址	访问
DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG	中断映射	0x3FF00164	读/写
DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG	中断映射	0x3FF00168	读/写
DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG	中断映射	0x3FF0016C	读/写
DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG	中断映射	0x3FF00170	读/写
DPORT_PRO_SPI_INTR_0_MAP_REG	中断映射	0x3FF00174	读/写
DPORT_PRO_SPI_INTR_1_MAP_REG	中断映射	0x3FF00178	读/写
DPORT_PRO_SPI_INTR_2_MAP_REG	中断映射	0x3FF0017C	读/写
DPORT_PRO_SPI_INTR_3_MAP_REG	中断映射	0x3FF00180	读/写
DPORT_PRO_I2S0_INT_MAP_REG	中断映射	0x3FF00184	读/写
DPORT_PRO_I2S1_INT_MAP_REG	中断映射	0x3FF00188	读/写
DPORT_PRO_UART_INTR_MAP_REG	中断映射	0x3FF0018C	读/写
DPORT_PRO_UART1_INTR_MAP_REG	中断映射	0x3FF00190	读/写
DPORT_PRO_UART2_INTR_MAP_REG	中断映射	0x3FF00194	读/写
DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG	中断映射	0x3FF00198	读/写
DPORT_PRO_EMAC_INT_MAP_REG	中断映射	0x3FF0019C	读/写
DPORT_PRO_PWM0_INTR_MAP_REG	中断映射	0x3FF001A0	读/写
DPORT_PRO_PWM1_INTR_MAP_REG	中断映射	0x3FF001A4	读/写
DPORT_PRO_LEDC_INT_MAP_REG	中断映射	0x3FF001B0	读/写
DPORT_PRO_EFUSE_INT_MAP_REG	中断映射	0x3FF001B4	读/写
DPORT_PRO_TWAI_INT_MAP_REG	中断映射	0x3FF001B8	读/写
DPORT_PRO_RTC_CORE_INTR_MAP_REG	中断映射	0x3FF001BC	读/写
DPORT_PRO_RMT_INTR_MAP_REG	中断映射	0x3FF001C0	读/写
DPORT_PRO_PCNT_INTR_MAP_REG	中断映射	0x3FF001C4	读/写
DPORT_PRO_I2C_EXT0_INTR_MAP_REG	中断映射	0x3FF001C8	读/写
DPORT_PRO_I2C_EXT1_INTR_MAP_REG	中断映射	0x3FF001CC	读/写
DPORT_PRO_RSA_INTR_MAP_REG	中断映射	0x3FF001D0	读/写
DPORT_PRO_SPI1_DMA_INT_MAP_REG	中断映射	0x3FF001D4	读/写
DPORT_PRO_SPI2_DMA_INT_MAP_REG	中断映射	0x3FF001D8	读/写
DPORT_PRO_SPI3_DMA_INT_MAP_REG	中断映射	0x3FF001DC	读/写
DPORT_PRO_WDG_INT_MAP_REG	中断映射	0x3FF001E0	读/写
DPORT_PRO_TIMER_INT1_MAP_REG	中断映射	0x3FF001E4	读/写
DPORT_PRO_TIMER_INT2_MAP_REG	中断映射	0x3FF001E8	读/写
DPORT_PRO_TG_T0_EDGE_INT_MAP_REG	中断映射	0x3FF001EC	读/写
DPORT_PRO_TG_T1_EDGE_INT_MAP_REG	中断映射	0x3FF001F0	读/写
DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG	中断映射	0x3FF001F4	读/写
DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG	中断映射	0x3FF001F8	读/写
DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG	中断映射	0x3FF001FC	读/写
DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG	中断映射	0x3FF00200	读/写
DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG	中断映射	0x3FF00204	读/写
DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG	中断映射	0x3FF00208	读/写
DPORT_PRO_MMU_IA_INT_MAP_REG	中断映射	0x3FF0020C	读/写
DPORT_PRO_MPU_IA_INT_MAP_REG	中断映射	0x3FF00210	读/写
DPORT_PRO_CACHE_IA_INT_MAP_REG	中断映射	0x3FF00214	读/写

名称	描述	地址	访问
DPORT_APP_MAC_INTR_MAP_REG	中断映射	0x3FF00218	读/写
DPORT_APP_MAC_NMI_MAP_REG	中断映射	0x3FF0021C	读/写
DPORT_APP_BB_INT_MAP_REG	中断映射	0x3FF00220	读/写
DPORT_APP_BT_MAC_INT_MAP_REG	中断映射	0x3FF00224	读/写
DPORT_APP_BT_BB_INT_MAP_REG	中断映射	0x3FF00228	读/写
DPORT_APP_BT_BB_NMI_MAP_REG	中断映射	0x3FF0022C	读/写
DPORT_APP_RWBT_IRQ_MAP_REG	中断映射	0x3FF00230	读/写
DPORT_APP_RWBLE_IRQ_MAP_REG	中断映射	0x3FF00234	读/写
DPORT_APP_RWBT_NMI_MAP_REG	中断映射	0x3FF00238	读/写
DPORT_APP_RWBLE_NMI_MAP_REG	中断映射	0x3FF0023C	读/写
DPORT_APP_SLC0_INTR_MAP_REG	中断映射	0x3FF00240	读/写
DPORT_APP_SLC1_INTR_MAP_REG	中断映射	0x3FF00244	读/写
DPORT_APP_UHCI0_INTR_MAP_REG	中断映射	0x3FF00248	读/写
DPORT_APP_UHCI1_INTR_MAP_REG	中断映射	0x3FF0024C	读/写
DPORT_APP_TG_T0_LEVEL_INT_MAP_REG	中断映射	0x3FF00250	读/写
DPORT_APP_TG_T1_LEVEL_INT_MAP_REG	中断映射	0x3FF00254	读/写
DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG	中断映射	0x3FF00258	读/写
DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG	中断映射	0x3FF0025C	读/写
DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG	中断映射	0x3FF00260	读/写
DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG	中断映射	0x3FF00264	读/写
DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG	中断映射	0x3FF00268	读/写
DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG	中断映射	0x3FF0026C	读/写
DPORT_APP_GPIO_INTERRUPT_MAP_REG	中断映射	0x3FF00270	读/写
DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG	中断映射	0x3FF00274	读/写
DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG	中断映射	0x3FF00278	读/写
DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG	中断映射	0x3FF0027C	读/写
DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG	中断映射	0x3FF00280	读/写
DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG	中断映射	0x3FF00284	读/写
DPORT_APP_SPI_INTR_0_MAP_REG	中断映射	0x3FF00288	读/写
DPORT_APP_SPI_INTR_1_MAP_REG	中断映射	0x3FF0028C	读/写
DPORT_APP_SPI_INTR_2_MAP_REG	中断映射	0x3FF00290	读/写
DPORT_APP_SPI_INTR_3_MAP_REG	中断映射	0x3FF00294	读/写
DPORT_APP_I2S0_INT_MAP_REG	中断映射	0x3FF00298	读/写
DPORT_APP_I2S1_INT_MAP_REG	中断映射	0x3FF0029C	读/写
DPORT_APP_UART_INTR_MAP_REG	中断映射	0x3FF002A0	读/写
DPORT_APP_UART1_INTR_MAP_REG	中断映射	0x3FF002A4	读/写
DPORT_APP_UART2_INTR_MAP_REG	中断映射	0x3FF002A8	读/写
DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG	中断映射	0x3FF002AC	读/写
DPORT_APP_EMAC_INT_MAP_REG	中断映射	0x3FF002B0	读/写
DPORT_APP_PWM0_INTR_MAP_REG	中断映射	0x3FF002B4	读/写
DPORT_APP_PWM1_INTR_MAP_REG	中断映射	0x3FF002B8	读/写
DPORT_APP_LEDC_INT_MAP_REG	中断映射	0x3FF002C4	读/写
DPORT_APP_EFUSE_INT_MAP_REG	中断映射	0x3FF002C8	读/写

名称	描述	地址	访问
DPORT_APP_TWAI_INT_MAP_REG	中断映射	0x3FF002CC	读/写
DPORT_APP_RTC_CORE_INTR_MAP_REG	中断映射	0x3FF002D0	读/写
DPORT_APP_RMT_INTR_MAP_REG	中断映射	0x3FF002D4	读/写
DPORT_APP_PCNT_INTR_MAP_REG	中断映射	0x3FF002D8	读/写
DPORT_APP_I2C_EXT0_INTR_MAP_REG	中断映射	0x3FF002DC	读/写
DPORT_APP_I2C_EXT1_INTR_MAP_REG	中断映射	0x3FF002E0	读/写
DPORT_APP_RSA_INTR_MAP_REG	中断映射	0x3FF002E4	读/写
DPORT_APP_SPI1_DMA_INT_MAP_REG	中断映射	0x3FF002E8	读/写
DPORT_APP_SPI2_DMA_INT_MAP_REG	中断映射	0x3FF002EC	读/写
DPORT_APP_SPI3_DMA_INT_MAP_REG	中断映射	0x3FF002F0	读/写
DPORT_APP_WDG_INT_MAP_REG	中断映射	0x3FF002F4	读/写
DPORT_APP_TIMER_INT1_MAP_REG	中断映射	0x3FF002F8	读/写
DPORT_APP_TIMER_INT2_MAP_REG	中断映射	0x3FF002FC	读/写
DPORT_APP_TG_T0_EDGE_INT_MAP_REG	中断映射	0x3FF00300	读/写
DPORT_APP_TG_T1_EDGE_INT_MAP_REG	中断映射	0x3FF00304	读/写
DPORT_APP_TG_WDT_EDGE_INT_MAP_REG	中断映射	0x3FF00308	读/写
DPORT_APP_TG_LACT_EDGE_INT_MAP_REG	中断映射	0x3FF0030C	读/写
DPORT_APP_TG1_T0_EDGE_INT_MAP_REG	中断映射	0x3FF00310	读/写
DPORT_APP_TG1_T1_EDGE_INT_MAP_REG	中断映射	0x3FF00314	读/写
DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG	中断映射	0x3FF00318	读/写
DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG	中断映射	0x3FF0031C	读/写
DPORT_APP_MMU_IA_INT_MAP_REG	中断映射	0x3FF00320	读/写
DPORT_APP_MPU_IA_INT_MAP_REG	中断映射	0x3FF00324	读/写
DPORT_APP_CACHE_IA_INT_MAP_REG	中断映射	0x3FF00328	读/写
DMA 寄存器			
DPORT_SPI_DMA_CHAN_SEL_REG	选择 SPI1, SPI2, SPI3 的 DMA 信道	0x3FF005A8	读/写
MPU/MMU 寄存器			
DPORT_PRO_CACHE_CTRL_REG	External SRAM 的虚拟地址模式	0x3FF00040	读/写
DPORT_PRO_CACHE_CTRL1_REG	配置 PRO cache MMU	0x3FF00044	读/写
DPORT_APP_CACHE_CTRL_REG	External SRAM 的虚拟地址模式	0x3FF00058	读/写
DPORT_APP_CACHE_CTRL1_REG	配置 APP cache MMU	0x3FF0005C	读/写
DPORT_IMMU_PAGE_MODE_REG	Internal SRAM 0 的 MMU 页大小	0x3FF00080	读/写
DPORT_DMMU_PAGE_MODE_REG	Internal SRAM 2 的 MMU 页大小	0x3FF00084	读/写
DPORT_AHB_MPU_TABLE_0_REG	配置 DMA 的 MPU	0x3FF000B4	读/写
DPORT_AHB_MPU_TABLE_1_REG	配置 DMA 的 MPU	0x3FF000B8	读/写
DPORT_AHBLITE_MPU_TABLE_UART_REG	外设 MPU	0x3FF0032C	读/写
DPORT_AHBLITE_MPU_TABLE_SPI1_REG	外设 MPU	0x3FF00330	读/写
DPORT_AHBLITE_MPU_TABLE_SPI0_REG	外设 MPU	0x3FF00334	读/写

名称	描述	地址	访问
DPORT_AHBLITE_MPU_TABLE_GPIO_REG	外设 MPU	0x3FF00338	读/写
DPORT_AHBLITE_MPU_TABLE_RTC_REG	外设 MPU	0x3FF00348	读/写
DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG	外设 MPU	0x3FF0034C	读/写
DPORT_AHBLITE_MPU_TABLE_HINF_REG	外设 MPU	0x3FF00354	读/写
DPORT_AHBLITE_MPU_TABLE_UHCI1_REG	外设 MPU	0x3FF00358	读/写
DPORT_AHBLITE_MPU_TABLE_I2S0_REG	外设 MPU	0x3FF00364	读/写
DPORT_AHBLITE_MPU_TABLE_UART1_REG	外设 MPU	0x3FF00368	读/写
DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG	外设 MPU	0x3FF00374	读/写
DPORT_AHBLITE_MPU_TABLE_UHCI0_REG	外设 MPU	0x3FF00378	读/写
DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG	外设 MPU	0x3FF0037C	读/写
DPORT_AHBLITE_MPU_TABLE_RMT_REG	外设 MPU	0x3FF00380	读/写
DPORT_AHBLITE_MPU_TABLE_PCNT_REG	外设 MPU	0x3FF00384	读/写
DPORT_AHBLITE_MPU_TABLE_SLC_REG	外设 MPU	0x3FF00388	读/写
DPORT_AHBLITE_MPU_TABLE_LEDC_REG	外设 MPU	0x3FF0038C	读/写
DPORT_AHBLITE_MPU_TABLE_EFUSE_REG	外设 MPU	0x3FF00390	读/写
DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG	外设 MPU	0x3FF00394	读/写
DPORT_AHBLITE_MPU_TABLE_PWM0_REG	外设 MPU	0x3FF0039C	读/写
DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG	外设 MPU	0x3FF003A0	读/写
DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG	外设 MPU	0x3FF003A4	读/写
DPORT_AHBLITE_MPU_TABLE_SPI2_REG	外设 MPU	0x3FF003A8	读/写
DPORT_AHBLITE_MPU_TABLE_SPI3_REG	外设 MPU	0x3FF003AC	读/写
DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG	外设 MPU	0x3FF003B0	读/写
DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG	外设 MPU	0x3FF003B4	读/写
DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG	外设 MPU	0x3FF003B8	读/写
DPORT_AHBLITE_MPU_TABLE_EMAC_REG	外设 MPU	0x3FF003BC	读/写
DPORT_AHBLITE_MPU_TABLE_PWM1_REG	外设 MPU	0x3FF003C4	读/写
DPORT_AHBLITE_MPU_TABLE_I2S1_REG	外设 MPU	0x3FF003C8	读/写
DPORT_AHBLITE_MPU_TABLE_UART2_REG	外设 MPU	0x3FF003CC	读/写
DPORT_AHBLITE_MPU_TABLE_PWR_REG	外设 MPU	0x3FF003E4	读/写
DPORT_IMMU_TABLE0_REG	配置 Internal SRAM 0 的 MMU	0x3FF00504	读/写
DPORT_IMMU_TABLE1_REG	配置 Internal SRAM 0 的 MMU	0x3FF00508	读/写
DPORT_IMMU_TABLE2_REG	配置 Internal SRAM 0 的 MMU	0x3FF0050C	读/写
DPORT_IMMU_TABLE3_REG	配置 Internal SRAM 0 的 MMU	0x3FF00510	读/写
DPORT_IMMU_TABLE4_REG	配置 Internal SRAM 0 的 MMU	0x3FF00514	读/写
DPORT_IMMU_TABLE5_REG	配置 Internal SRAM 0 的 MMU	0x3FF00518	读/写

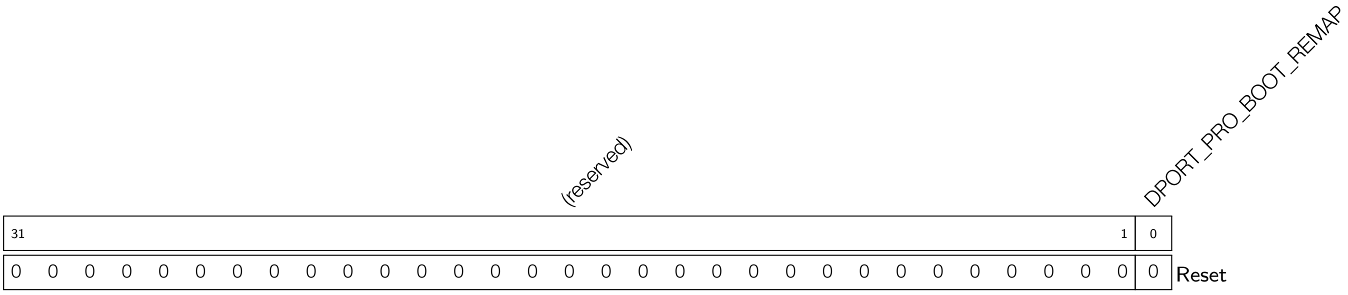
名称	描述	地址	访问
DPORT_IMMU_TABLE6_REG	配置 Internal SRAM 0 的 MMU	0x3FF0051C	读/写
DPORT_IMMU_TABLE7_REG	配置 Internal SRAM 0 的 MMU	0x3FF00520	读/写
DPORT_IMMU_TABLE8_REG	配置 Internal SRAM 0 的 MMU	0x3FF00524	读/写
DPORT_IMMU_TABLE9_REG	配置 Internal SRAM 0 的 MMU	0x3FF00528	读/写
DPORT_IMMU_TABLE10_REG	配置 Internal SRAM 0 的 MMU	0x3FF0052C	读/写
DPORT_IMMU_TABLE11_REG	配置 Internal SRAM 0 的 MMU	0x3FF00530	读/写
DPORT_IMMU_TABLE12_REG	配置 Internal SRAM 0 的 MMU	0x3FF00534	读/写
DPORT_IMMU_TABLE13_REG	配置 Internal SRAM 0 的 MMU	0x3FF00538	读/写
DPORT_IMMU_TABLE14_REG	配置 Internal SRAM 0 的 MMU	0x3FF0053C	读/写
DPORT_IMMU_TABLE15_REG	配置 Internal SRAM 0 的 MMU	0x3FF00540	读/写
DPORT_DMMU_TABLE0_REG	配置 Internal SRAM 2 的 MMU	0x3FF00544	读/写
DPORT_DMMU_TABLE1_REG	配置 Internal SRAM 2 的 MMU	0x3FF00548	读/写
DPORT_DMMU_TABLE2_REG	配置 Internal SRAM 2 的 MMU	0x3FF0054C	读/写
DPORT_DMMU_TABLE3_REG	配置 Internal SRAM 2 的 MMU	0x3FF00550	读/写
DPORT_DMMU_TABLE4_REG	配置 Internal SRAM 2 的 MMU	0x3FF00554	读/写
DPORT_DMMU_TABLE5_REG	配置 Internal SRAM 2 的 MMU	0x3FF00558	读/写
DPORT_DMMU_TABLE6_REG	配置 Internal SRAM 2 的 MMU	0x3FF0055C	读/写
DPORT_DMMU_TABLE7_REG	配置 Internal SRAM 2 的 MMU	0x3FF00560	读/写
DPORT_DMMU_TABLE8_REG	配置 Internal SRAM 2 的 MMU	0x3FF00564	读/写
DPORT_DMMU_TABLE9_REG	配置 Internal SRAM 2 的 MMU	0x3FF00568	读/写
DPORT_DMMU_TABLE10_REG	配置 Internal SRAM 2 的 MMU	0x3FF0056C	读/写
DPORT_DMMU_TABLE11_REG	配置 Internal SRAM 2 的 MMU	0x3FF00570	读/写

名称	描述	地址	访问
DPORT_DMMU_TABLE12_REG	配置 Internal SRAM 2 的 MMU	0x3FF00574	读/写
DPORT_DMMU_TABLE13_REG	配置 Internal SRAM 2 的 MMU	0x3FF00578	读/写
DPORT_DMMU_TABLE14_REG	配置 Internal SRAM 2 的 MMU	0x3FF0057C	读/写
DPORT_DMMU_TABLE15_REG	配置 Internal SRAM 2 的 MMU	0x3FF00580	读/写
APP_CPU 控制器寄存器			
DPORT_APPCPU_CTRL_REG_A_REG	APP_CPU 复位	0x3FF0002C	读/写
DPORT_APPCPU_CTRL_REG_B_REG	APP_CPU 时钟门控	0x3FF00030	读/写
DPORT_APPCPU_CTRL_REG_C_REG	APP_CPU 暂停 (stall)	0x3FF00034	读/写
DPORT_APPCPU_CTRL_REG_D_REG	APP_CPU 启动地址	0x3FF00038	读/写
外设时钟门控和复位寄存器			
DPORT_PERI_CLK_EN_REG	外设时钟门控	0x3FF0001C	读/写
DPORT_PERI_RST_EN_REG	外设复位	0x3FF00020	读/写
DPORT_PERIP_CLK_EN_REG	外设时钟门控	0x3FF000C0	读/写
DPORT_PERIP_RST_EN_REG	外设复位	0x3FF000C4	读/写
DPORT_WIFI_CLK_EN_REG	Wi-Fi 时钟门控	0x3FF000CC	读/写
DPORT_WIFI_RST_EN_REG	Wi-Fi 复位	0x3FF000D0	读/写

5.5 寄存器

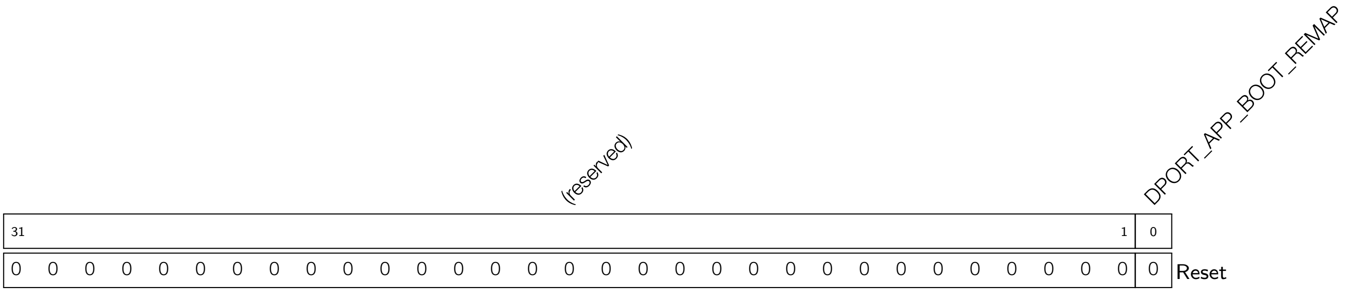
本小节括号中的地址均为相对于 DPORT 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 5.4 寄存器列表。

Register 5.1. DPORT_PRO_BOOT_REMAP_CTRL_REG (0x000)



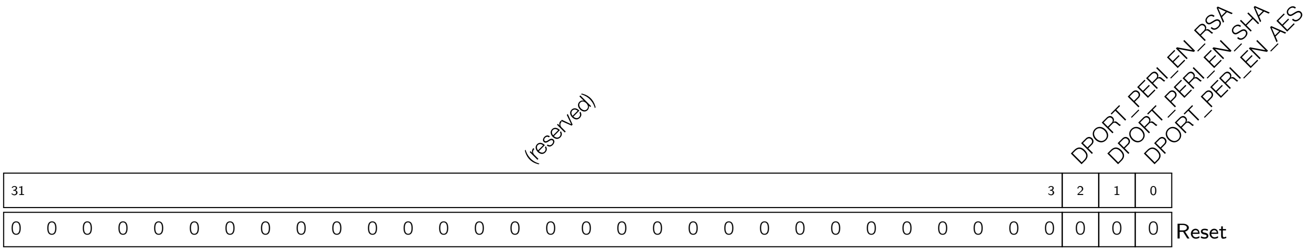
DPORT_PRO_BOOT_REMAP PRO_CPU 的存储器重映射模式。（读/写）

Register 5.2. DPORT_APP_BOOT_REMAP_CTRL_REG (0x004)



DPORT_APP_BOOT_REMAP APP_CPU 的存储器重映射模式。（读/写）

Register 5.3. DPORT_PERI_CLK_EN_REG (0x01C)



DPORT_PERI_EN_RSA 置位使能 RSA 模块的时钟。清零关闭 RSA 模块的时钟。（读/写）

DPORT_PERI_EN_SHA 置位使能 SHA 模块的时钟。清零关闭 SHA 模块的时钟。（读/写）

DPORT_PERI_EN_AES 置位使能 AES 模块的时钟。清零关闭 AES 模块的时钟。（读/写）

Register 5.4. DPORT_PERI_RST_EN_REG (0x020)

(reserved)	DPORT_PERI_RST_RSA	DPORT_PERI_RST_SHA	DPORT_PERI_RST_AES	
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	Reset

DPORT_PERI_RST_RSA 置位复位 RSA 模块。清零关闭 RSA 模块的时钟。（读/写）

DPORT_PERI_RST_SHA 置位复位 SHA 模块的时钟。清零释放 SHA 模块。（读/写）

DPORT_PERI_RST_AES 置位复位 AES 模块。清零释放 AES 模块。（读/写）

Register 5.5. DPORT_APPCPU_CTRL_REG_A_REG (0x02C)

(reserved)	DPORT_APPCPU_RESETTING	
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

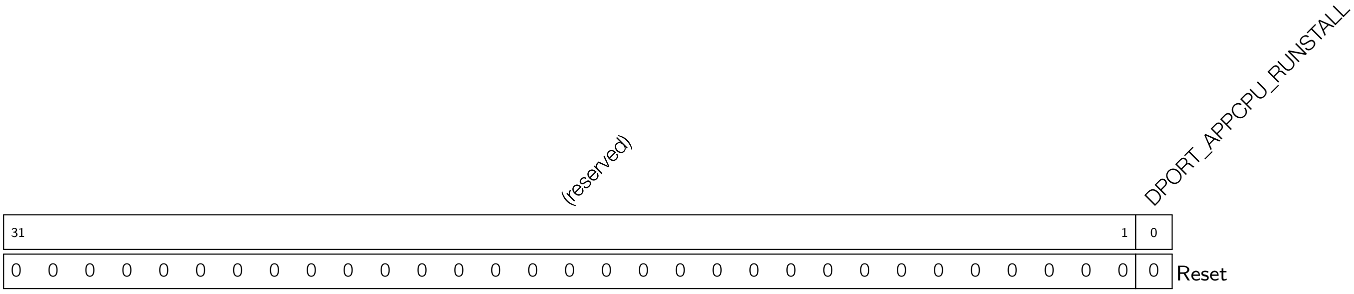
DPORT_APPCPU_RESETTING 置位复位 APP_CPU。清零释放 APP_CPU。（读/写）

Register 5.6. DPORT_APPCPU_CTRL_REG_B_REG (0x030)

(reserved)	DPORT_APPCPU_CLKGATE_EN	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

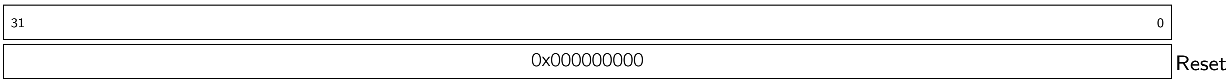
DPORT_APPCPU_CLKGATE_EN 置位使能 APP_CPU 的时钟。清零关闭 APP_CPU 的时钟。（读/写）

Register 5.7. DPORT_APPCPU_CTRL_REG_C_REG (0x034)



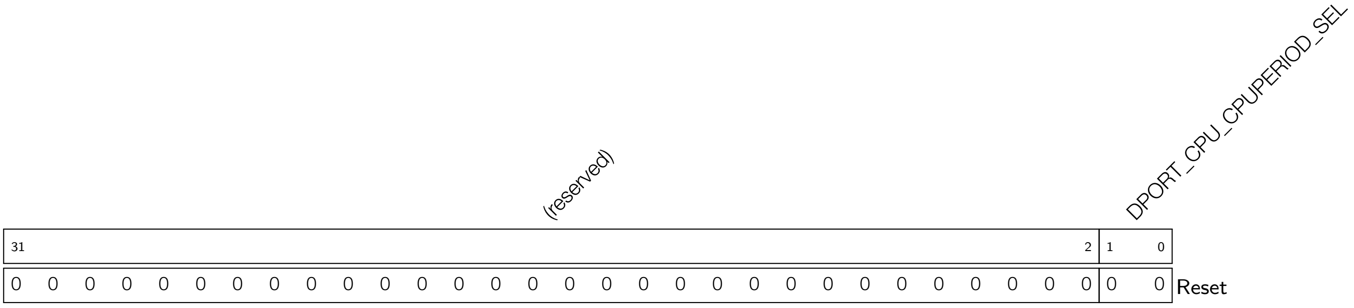
DPORT_APPCPU_RUNSTALL 置位使能 APP_CPU 的暂停 (stall) 状态。清零释放 APP_CPU 的暂停 (stall) 状态。（读/写）

Register 5.8. DPORT_APPCPU_CTRL_REG_D_REG (0x038)



DPORT_APPCPU_CTRL_REG_D_REG APP_CPU 从 ROM code 启动之后，会跳转到该寄存器中的地址。（读/写）

Register 5.9. DPORT_CPU_PER_CONF_REG (0x03C)



DPORT_CPU_CPUPERIOD_SEL 选择 CPU 时钟。详情请参考表 3-3。（读/写）

Register 5.10. DPORT_PRO_CACHE_CTRL_REG (0x040)

(reserved)	DPORT_PRO_DRAM_HL	(reserved)	DPORT_PRO_DRAM_SPLIT	DPORT_PRO_SINGLE_IRAM_ENA	(reserved)	DPORT_PRO_CACHE_FLUSH_DONE	DPORT_PRO_CACHE_FLUSH_ENA	DPORT_PRO_CACHE_ENABLE	(reserved)	
31 17	16	15 12	11	10	9 6	5	4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0 0 0	0	0	0 0 0 0	0	1	0	0 0 0	Reset

DPORT_PRO_DRAM_HL External SRAM 的虚拟地址模式。(读/写)

DPORT_PRO_DRAM_SPLIT External SRAM 的虚拟地址模式。(读/写)

DPORT_PRO_SINGLE_IRAM_ENA PRO_CPU 访问 external flash 的特殊模式。(读/写)

DPORT_PRO_CACHE_FLUSH_DONE 清除 PRO_CPU cache 完成标志。(只读)

DPORT_PRO_CACHE_FLUSH_ENA 清除 PRO_CPU cache。(读/写)

DPORT_PRO_CACHE_ENABLE 使能 PRO_CPU cache。(读/写)

Register 5.11. DPORT_PRO_CACHE_CTRL1_REG (0x044)

(reserved)	DPORT_PRO_CACHE_MMU_IA_CLR	DPORT_PRO_CMMU_PD	(reserved)	DPORT_PRO_CACHE_MASK_OPSDRAM	DPORT_PRO_CACHE_MASK_DROM0	DPORT_PRO_CACHE_MASK_DRAM1	DPORT_PRO_CACHE_MASK_IROM0	DPORT_PRO_CACHE_MASK_IRAM1	DPORT_PRO_CACHE_MASK_IRAM0	
31 14	13	12	11 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0	1	1	1	1	1	1	Reset

DPORT_PRO_CACHE_MMU_IA_CLR 清除 PRO cache MMU 错误标志。（读/写）

DPORT_PRO_CMMU_PD 关闭 PRO cache MMU。（读/写）

DPORT_PRO_CACHE_MASK_OPSDRAM 关闭 APP_CPU DRAM1 总线访问 PRO cache。

1：关闭

0：开启

（读/写）

DPORT_PRO_CACHE_MASK_DROM0 关闭 PRO_CPU DROM0 总线访问 PRO cache。

1：关闭

0：开启

（读/写）

DPORT_PRO_CACHE_MASK_DRAM1 关闭 PRO_CPU DRAM1 总线访问 PRO cache。

1：关闭

0：开启

（读/写）

DPORT_PRO_CACHE_MASK_IROM0 关闭 PRO_CPU IROM0 总线访问 PRO cache。

1：关闭

0：开启

（读/写）

DPORT_PRO_CACHE_MASK_IRAM1 关闭 PRO_CPU IRAM1 总线访问 PRO cache。

1：关闭

0：开启

（读/写）

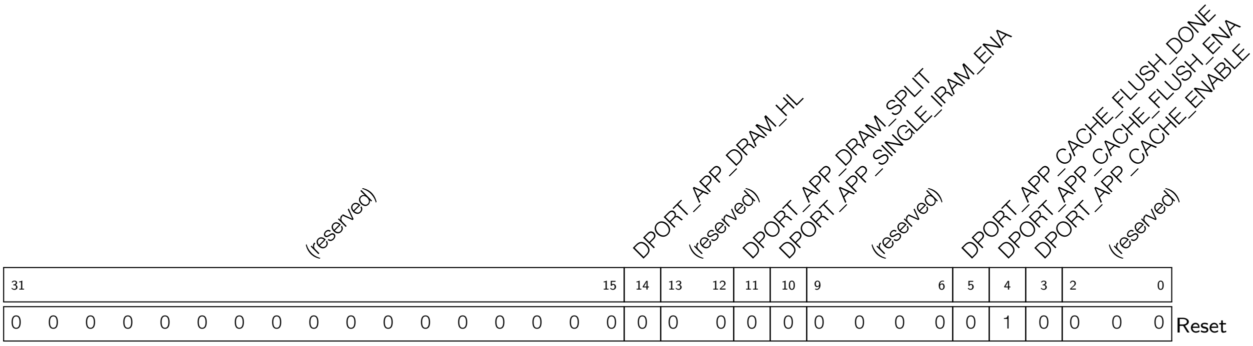
DPORT_PRO_CACHE_MASK_IRAM0 关闭 PRO_CPU IRAM0 总线访问 PRO cache。

1：关闭

0：开启

（读/写）

Register 5.12. DPORT_APP_CACHE_CTRL_REG (0x058)



DPORT_APP_DRAM_HL External SRAM 的虚拟地址模式。（读/写）

DPORT_APP_DRAM_SPLIT External SRAM 的虚拟地址模式。（读/写）

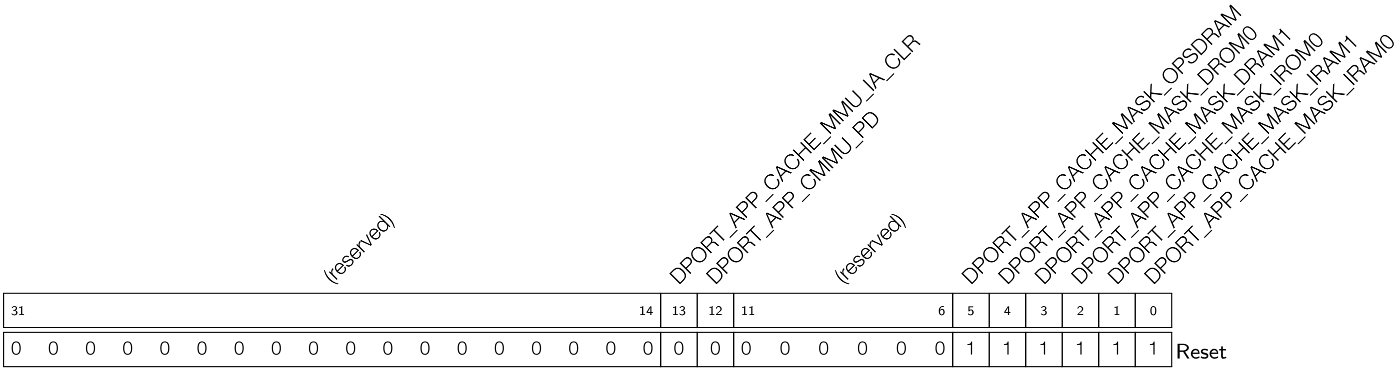
DPORT_APP_SINGLE_IRAM_ENA APP_CPU 访问 external flash 的特殊模式。（读/写）

DPORT_APP_CACHE_FLUSH_DONE 清除 APP_CPU cache 完成标志。（只读）

DPORT_APP_CACHE_FLUSH_ENA 清除 APP_CPU cache。（读/写）

DPORT_APP_CACHE_ENABLE 使能 APP_CPU cache。（读/写）

Register 5.13. DPORT_APP_CACHE_CTRL1_REG (0x05C)



(reserved)	DPORT_APP_CACHE_MMU_IA_CLR	DPORT_APP_CMMU_PD	(reserved)	DPORT_APP_CACHE_MASK_OPSDRAM	DPORT_APP_CACHE_MASK_DROM0	DPORT_APP_CACHE_MASK_DRAM1	DPORT_APP_CACHE_MASK_IROM0	DPORT_APP_CACHE_MASK_IRAM1	DPORT_APP_CACHE_MASK_IRAM0	
31 … 14	13	12	11 … 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0	1	1	1	1	1	1	Reset

DPORT_APP_CACHE_MMU_IA_CLR 清除 APP cache MMU 错误标志。（读/写）

DPORT_APP_CMMU_PD 关闭 APP cache MMU。（读/写）

DPORT_APP_CACHE_MASK_OPSDRAM 关闭 PRO_CPU DRAM1 总线访问 APP cache。

1：关闭
0：开启
（读/写）

DPORT_APP_CACHE_MASK_DROM0 关闭 APP_CPU DROM0 总线访问 APP cache。

1：关闭
0：开启
（读/写）

DPORT_APP_CACHE_MASK_DRAM1 关闭 APP_CPU DRAM1 总线访问 APP cache。

1：关闭
0：开启
（读/写）

DPORT_APP_CACHE_MASK_IROM0 关闭 APP_CPU IROM0 总线访问 APP cache。

1：关闭
0：开启
（读/写）

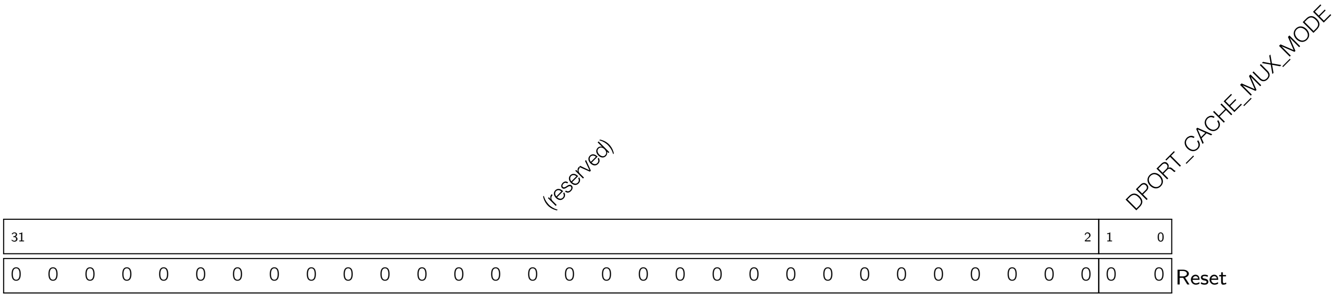
DPORT_APP_CACHE_MASK_IRAM1 关闭 APP_CPU IRAM1 总线访问 APP cache。

1：关闭
0：开启
（读/写）

DPORT_APP_CACHE_MASK_IRAM0 关闭 APP_CPU IRAM0 总线访问 APP cache。

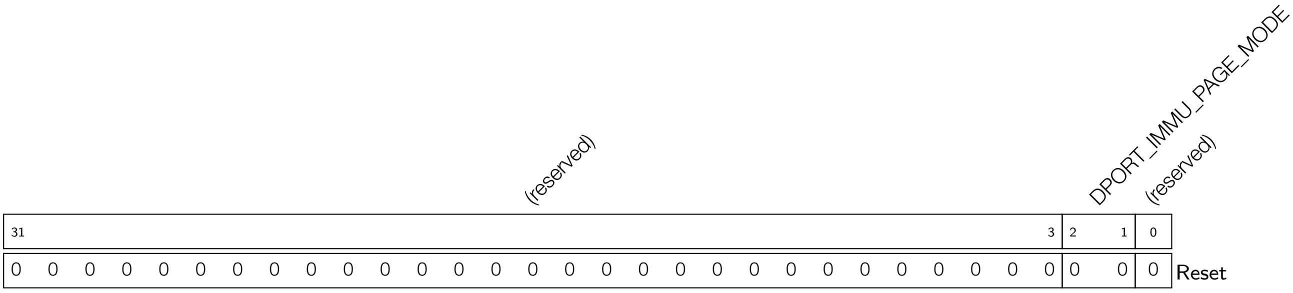
1：关闭
0：开启
（读/写）

Register 5.14. DPORT_CACHE_MUX_MODE_REG (0x07C)



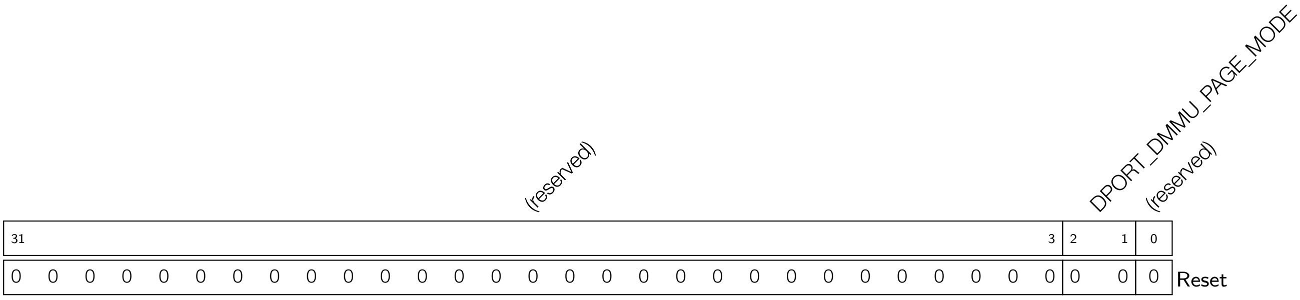
DPORT_CACHE_MUX_MODE 两个 cache 共用内存的模式。（读/写）

Register 5.15. DPORT_IMMU_PAGE_MODE_REG (0x080)



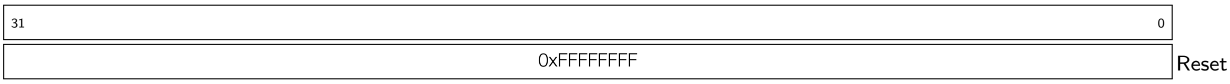
DPORT_IMMU_PAGE_MODE Internal SRAM 0 的 MMU 页大小。（读/写）

Register 5.16. DPORT_DMMU_PAGE_MODE_REG (0x084)



DPORT_DMMU_PAGE_MODE Internal SRAM 2 的 MMU 页大小。（读/写）

Register 5.17. DPORT_AHB_MPU_TABLE_0_REG (0x0B4)



DPORT_AHB_MPU_TABLE_0_REG 配置 DMA 的 MPU。（读/写）

Register 5.18. DPORT_AHB_MPU_TABLE_1_REG (0x0B8)

(reserved)	DPORT_AHB_ACCESS_GRANT_1	
31 9	8 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x1FF	Reset

DPORT_AHB_ACCESS_GRANT_1 配置 DMA 的 MPU。（读/写）

Register 5.19. DPORT_PERIP_CLK_EN_REG (0x0C0)

(reserved)	(reserved)	(reserved)	DPORT_UART_MEM_CLK_EN	DPORT_UART2_CLK_EN	DPORT_SPI_DMA_CLK_EN	DPORT_I2S1_CLK_EN	DPORT_PWM1_CLK_EN	DPORT_TWAI_CLK_EN	DPORT_I2C_EXT1_CLK_EN	DPORT_PWM0_CLK_EN	DPORT_SPI3_CLK_EN	DPORT_TIMERGROUP1_CLK_EN	DPORT_EFUSE_CLK_EN	DPORT_TIMERGROUP_CLK_EN	DPORT_UHCI1_CLK_EN	DPORT_LEDC_CLK_EN	DPORT_PCNT_CLK_EN	DPORT_RMT_CLK_EN	DPORT_UHCI0_CLK_EN	DPORT_I2C_EXT0_CLK_EN	DPORT_SPI2_CLK_EN	DPORT_UART1_CLK_EN	DPORT_I2S0_CLK_EN	(reserved)	DPORT_UART_CLK_EN	DPORT_SPI01_CLK_EN	(reserved)	
31 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	
11111	0	0	1	1	1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	0	1	1	1	1	Reset

置位以下字段使能相应外设的时钟。清零关闭相应外设的时钟。

DPORT_UART_MEM_CLK_EN UART0 ~ 2 共用的存储器。只要有一个 UART 在工作，UART 存储器就不能处于门控状态。（读/写）

DPORT_UART2_CLK_EN UART2 模块。（读/写）

DPORT_SPI_DMA_CLK_EN SPI_DMA 模块。（读/写）

DPORT_I2S1_CLK_EN I2S1 模块。（读/写）

DPORT_PWM1_CLK_EN PWM1 模块。（读/写）

DPORT_TWAI_CLK_EN TWAI 模块。（读/写）

DPORT_I2C_EXT1_CLK_EN I2C1 模块。（读/写）

DPORT_PWM0_CLK_EN PWM0 模块。（读/写）

DPORT_SPI3_CLK_EN SPI3 模块。（读/写）

DPORT_TIMERGROUP1_CLK_EN TIMG1 模块。（读/写）

DPORT_EFUSE_CLK_EN eFuse 模块。（读/写）

DPORT_TIMERGROUP_CLK_EN TIMG0 模块。（读/写）

DPORT_UHCI1_CLK_EN UDMA1 模块。（读/写）

DPORT_LEDC_CLK_EN LEDC 模块。（读/写）

DPORT_PCNT_CLK_EN PCNT 模块。（读/写）

DPORT_RMT_CLK_EN RMT 模块。（读/写）

DPORT_UHCI0_CLK_EN UDMA0 模块。（读/写）

DPORT_I2C_EXT0_CLK_EN I2C0 模块。（读/写）

DPORT_SPI2_CLK_EN SPI2 模块。（读/写）

接下页

Register 5.19. DPORT_PERIP_CLK_EN_REG (0x0C0)

接上页

DPORT_UART1_CLK_EN UART1 模块。（读/写）

DPORT_I2S0_CLK_EN I2S0 模块。（读/写）

DPORT_UART_CLK_EN UART0 模块。（读/写）

DPORT_SPI01_CLK_EN SPI0 和 SPI1 模块。（读/写）

Register 5.20. DPORT_PERIP_RST_EN_REG (0x0C4)

(reserved)	(reserved)	(reserved)	DPORT_UART_MEM_RST	DPORT_UART2_RST	DPORT_SPI_DMA_RST	DPORT_I2S1_RST	DPORT_PWM1_RST	DPORT_TWAI_RST	DPORT_I2C_EXT1_RST	DPORT_PWM0_RST	DPORT_SPI3_RST	DPORT_TIMERGROUP1_RST	DPORT_EFUSE_RST	DPORT_TIMERGROUP_RST	DPORT_UHCI1_RST	DPORT_LEDC_RST	DPORT_PCNT_RST	DPORT_RMT_RST	DPORT_UHCI0_RST	DPORT_I2C_EXT0_RST	DPORT_SPI2_RST	DPORT_UART1_RST	DPORT_I2S0_RST	(reserved)	DPORT_UART_RST	DPORT_SPI01_RST	(reserved)	
31 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	
00000	0	0	0	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

置位每个字段复位相应外设。清零释放相应外设。外设列表请查看寄存器描述 5.19。

Register 5.21. DPORT_WIFI_CLK_EN_REG (0x0CC)

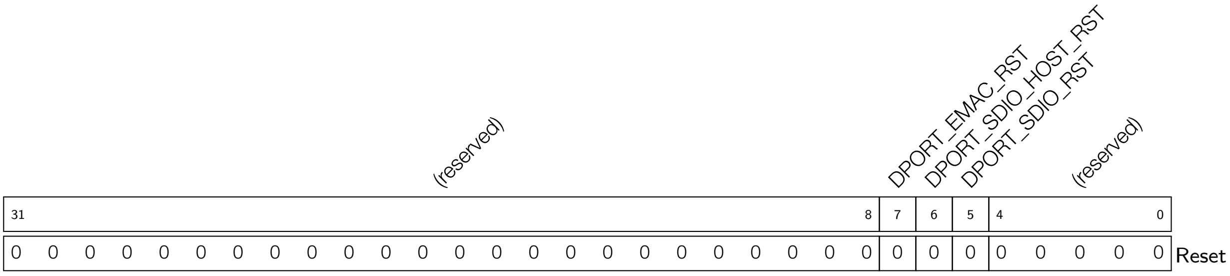
(reserved)	DPORT_WIFI_CLK_EMAC_EN	DPORT_WIFI_CLK_SDIO_HOST_EN	(reserved)	DPORT_WIFI_CLK_SDIOSLAVE_EN	(reserved)	
31 15	14	13	12 5	4	3 0	
1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1	1	1	0 0 0 0 0 0 0 1	1	0 0 0 0	Reset

DPORT_WIFI_CLK_EMAC_EN 置位使能 Ethernet MAC 模块的时钟。清零关闭 Ethernet MAC 模块的时钟。（读/写）

DPORT_WIFI_CLK_SDIO_HOST_EN 置位使能 SD/MMC 模块的时钟。清零关闭 SD/MMC 模块的时钟。（读/写）

DPORT_WIFI_CLK_SDIOSLAVE_EN 置位使能 SDIO 模块的时钟。清零关闭 SDIO 模块的时钟。（读/写）

Register 5.22. DPORT_WIFI_RST_EN_REG (0x0D0)

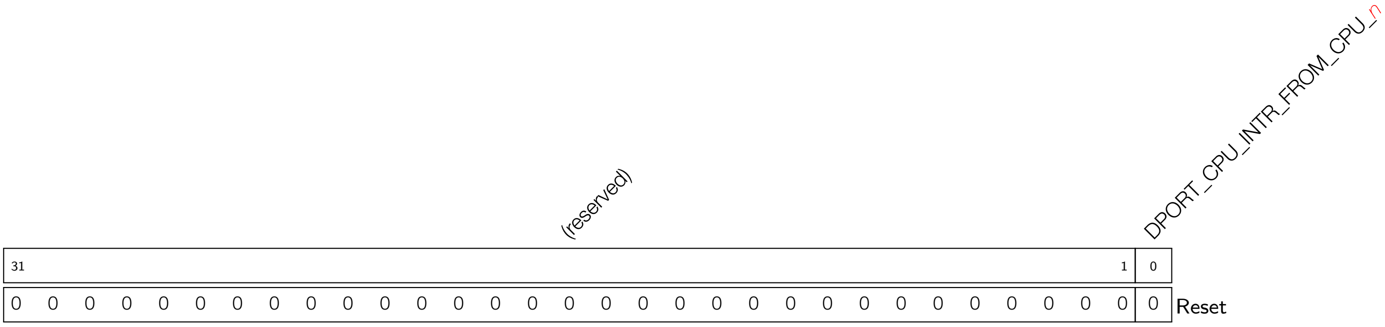


DPORT_EMAC_RST 置位复位 Ethernet MAC 模块。清零释放 Ethernet MAC 模块。(读/写)

DPORT_SDIO_HOST_RST 置位复位 SD/MMC 模块。清零释放 SD/MMC 模块。(读/写)

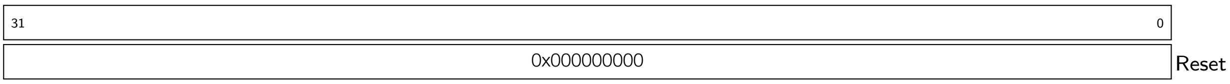
DPORT_SDIO_RST 置位复位 SDIO 模块。清零释放 SDIO 模块。(读/写)

Register 5.23. DPORT_CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3) (0xDC+4**n*)



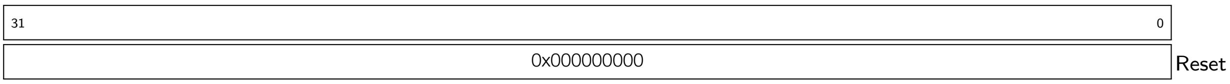
DPORT_CPU_INTR_FROM_CPU_*n* 此寄存器置 1 触发 CPU 中断。(读/写)

Register 5.24. DPORT_PRO_INTR_STATUS_REG_*n*_REG (*n*: 0-2) (0xEC+4**n*)



DPORT_PRO_INTR_STATUS_REG_*n*_REG PRO_CPU 中断状态。(只读)

Register 5.25. DPORT_APP_INTR_STATUS_REG_*n*_REG (*n*: 0-2) (0xF8+4**n*)



DPORT_APP_INTR_STATUS_REG_*n*_REG APP_CPU 中断状态。(只读)

Register 5.26. DPORT_PRO_MAC_INTR_MAP_REG (0x104)

Register 5.27. DPORT_PRO_MAC_NMI_MAP_REG (0x108)

Register 5.28. DPORT_PRO_BB_INT_MAP_REG (0x10C)

Register 5.29. DPORT_PRO_BT_MAC_INT_MAP_REG (0x110)

Register 5.30. DPORT_PRO_BT_BB_INT_MAP_REG (0x114)

Register 5.31. DPORT_PRO_BT_BB_NMI_MAP_REG (0x118)

Register 5.32. DPORT_PRO_RWBT_IRQ_MAP_REG (0x11C)

Register 5.33. DPORT_PRO_RWBLE_IRQ_MAP_REG (0x120)

Register 5.34. DPORT_PRO_RWBT_NMI_MAP_REG (0x124)

Register 5.35. DPORT_PRO_RWBLE_NMI_MAP_REG (0x128)

Register 5.36. DPORT_PRO_SLC0_INTR_MAP_REG (0x12C)

Register 5.37. DPORT_PRO_SLC1_INTR_MAP_REG (0x130)

Register 5.38. DPORT_PRO_UHCI0_INTR_MAP_REG (0x134)

Register 5.39. DPORT_PRO_UHCI1_INTR_MAP_REG (0x138)

Register 5.40. DPORT_PRO_TG_T0_LEVEL_INT_MAP_REG (0x13C)

Register 5.41. DPORT_PRO_TG_T1_LEVEL_INT_MAP_REG (0x140)

Register 5.42. DPORT_PRO_TG_WDT_LEVEL_INT_MAP_REG (0x144)

Register 5.43. DPORT_PRO_TG_LACT_LEVEL_INT_MAP_REG (0x148)

Register 5.44. DPORT_PRO_TG1_T0_LEVEL_INT_MAP_REG (0x14C)

Register 5.45. DPORT_PRO_TG1_T1_LEVEL_INT_MAP_REG (0x150)

Register 5.46. DPORT_PRO_TG1_WDT_LEVEL_INT_MAP_REG (0x154)

Register 5.47. DPORT_PRO_TG1_LACT_LEVEL_INT_MAP_REG (0x158)

Register 5.48. DPORT_PRO_GPIO_INTERRUPT_MAP_REG (0x15C)

Register 5.49. DPORT_PRO_GPIO_INTERRUPT_NMI_MAP_REG (0x160)

Register 5.50. DPORT_PRO_CPU_INTR_FROM_CPU_0_MAP_REG (0x164)

Register 5.51. DPORT_PRO_CPU_INTR_FROM_CPU_1_MAP_REG (0x168)

Register 5.52. DPORT_PRO_CPU_INTR_FROM_CPU_2_MAP_REG (0x16C)

Register 5.53. DPORT_PRO_CPU_INTR_FROM_CPU_3_MAP_REG (0x170)

Register 5.54. DPORT_PRO_SPI_INTR_0_MAP_REG (0x174)

Register 5.55. DPORT_PRO_SPI_INTR_1_MAP_REG (0x178)

Register 5.56. DPORT_PRO_SPI_INTR_2_MAP_REG (0x17C)

Register 5.57. DPORT_PRO_SPI_INTR_3_MAP_REG (0x180)

Register 5.58. DPORT_PRO_I2S0_INT_MAP_REG (0x184)

Register 5.59. DPORT_PRO_I2S1_INT_MAP_REG (0x188)

Register 5.60. DPORT_PRO_UART_INTR_MAP_REG (0x18C)

Register 5.61. DPORT_PRO_UART1_INTR_MAP_REG (0x190)

Register 5.62. DPORT_PRO_UART2_INTR_MAP_REG (0x194)

Register 5.63. DPORT_PRO_SDIO_HOST_INTERRUPT_MAP_REG (0x198)

Register 5.64. DPORT_PRO_EMAC_INT_MAP_REG (0x19C)

Register 5.65. DPORT_PRO_PWM0_INTR_MAP_REG (0x1A0)

Register 5.66. DPORT_PRO_PWM1_INTR_MAP_REG (0x1A4)

Register 5.67. DPORT_PRO_LEDC_INT_MAP_REG (0x1B0)

Register 5.68. DPORT_PRO_EFUSE_INT_MAP_REG (0x1B4)

Register 5.69. DPORT_PRO_TWAI_INT_MAP_REG (0x1B8)

Register 5.70. DPORT_PRO_RTC_CORE_INTR_MAP_REG (0x1BC)

Register 5.71. DPORT_PRO_RMT_INTR_MAP_REG (0x1C0)

Register 5.72. DPORT_PRO_PCNT_INTR_MAP_REG (0x1C4)

Register 5.73. DPORT_PRO_I2C_EXT0_INTR_MAP_REG (0x1C8)

Register 5.74. DPORT_PRO_I2C_EXT1_INTR_MAP_REG (0x1CC)

Register 5.75. DPORT_PRO_RSA_INTR_MAP_REG (0x1D0)

Register 5.76. DPORT_PRO_SPI1_DMA_INT_MAP_REG (0x1D4)

Register 5.77. DPORT_PRO_SPI2_DMA_INT_MAP_REG (0x1D8)

Register 5.78. DPORT_PRO_SPI3_DMA_INT_MAP_REG (0x1DC)

Register 5.79. DPORT_PRO_WDG_INT_MAP_REG (0x1E0)

Register 5.80. DPORT_PRO_TIMER_INT1_MAP_REG (0x1E4)

Register 5.81. DPORT_PRO_TIMER_INT2_MAP_REG (0x1E8)

Register 5.82. DPORT_PRO_TG_T0_EDGE_INT_MAP_REG (0x1EC)

Register 5.83. DPORT_PRO_TG_T1_EDGE_INT_MAP_REG (0x1F0)

Register 5.84. DPORT_PRO_TG_WDT_EDGE_INT_MAP_REG (0x1F4)

Register 5.85. DPORT_PRO_TG_LACT_EDGE_INT_MAP_REG (0x1F8)

Register 5.86. DPORT_PRO_TG1_T0_EDGE_INT_MAP_REG (0x1FC)

Register 5.87. DPORT_PRO_TG1_T1_EDGE_INT_MAP_REG (0x200)

Register 5.88. DPORT_PRO_TG1_WDT_EDGE_INT_MAP_REG (0x204)

Register 5.89. DPORT_PRO_TG1_LACT_EDGE_INT_MAP_REG (0x208)

Register 5.90. DPORT_PRO_MMU_IA_INT_MAP_REG (0x20C)

Register 5.91. DPORT_PRO_MPU_IA_INT_MAP_REG (0x210)

Register 5.92. DPORT_PRO_CACHE_IA_INT_MAP_REG (0x214)

(reserved)	DPORT_PRO_*_MAP	
31 5	4 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	10000	Reset

DPORT_PRO_*_MAP 中断对应关系配置寄存器。（读/写）

Register 5.93. DPORT_APP_MAC_INTR_MAP_REG (0x218)

Register 5.94. DPORT_APP_MAC_NMI_MAP_REG (0x21C)

Register 5.95. DPORT_APP_BB_INT_MAP_REG (0x220)

Register 5.96. DPORT_APP_BT_MAC_INT_MAP_REG (0x224)

Register 5.97. DPORT_APP_BT_BB_INT_MAP_REG (0x228)

Register 5.98. DPORT_APP_BT_BB_NMI_MAP_REG (0x22C)

Register 5.99. DPORT_APP_RWBT_IRQ_MAP_REG (0x230)

Register 5.100. DPORT_APP_RWBLE_IRQ_MAP_REG (0x234)

Register 5.101. DPORT_APP_RWBT_NMI_MAP_REG (0x238)

Register 5.102. DPORT_APP_RWBLE_NMI_MAP_REG (0x23C)

Register 5.103. DPORT_APP_SLC0_INTR_MAP_REG (0x240)

Register 5.104. DPORT_APP_SLC1_INTR_MAP_REG (0x244)

Register 5.105. DPORT_APP_UHCI0_INTR_MAP_REG (0x248)

Register 5.106. DPORT_APP_UHCI1_INTR_MAP_REG (0x24C)

Register 5.107. DPORT_APP_TG_T0_LEVEL_INT_MAP_REG (0x250)

Register 5.108. DPORT_APP_TG_T1_LEVEL_INT_MAP_REG (0x254)

Register 5.109. DPORT_APP_TG_WDT_LEVEL_INT_MAP_REG (0x258)

Register 5.110. DPORT_APP_TG_LACT_LEVEL_INT_MAP_REG (0x25C)

Register 5.111. DPORT_APP_TG1_T0_LEVEL_INT_MAP_REG (0x260)

Register 5.112. DPORT_APP_TG1_T1_LEVEL_INT_MAP_REG (0x264)

Register 5.113. DPORT_APP_TG1_WDT_LEVEL_INT_MAP_REG (0x268)

Register 5.114. DPORT_APP_TG1_LACT_LEVEL_INT_MAP_REG (0x26C)

Register 5.115. DPORT_APP_GPIO_INTERRUPT_MAP_REG (0x270)

Register 5.116. DPORT_APP_GPIO_INTERRUPT_NMI_MAP_REG (0x274)

Register 5.117. DPORT_APP_CPU_INTR_FROM_CPU_0_MAP_REG (0x278)

Register 5.118. DPORT_APP_CPU_INTR_FROM_CPU_1_MAP_REG (0x27C)

Register 5.119. DPORT_APP_CPU_INTR_FROM_CPU_2_MAP_REG (0x280)

Register 5.120. DPORT_APP_CPU_INTR_FROM_CPU_3_MAP_REG (0x284)

Register 5.121. DPORT_APP_SPI_INTR_0_MAP_REG (0x288)

Register 5.122. DPORT_APP_SPI_INTR_1_MAP_REG (0x28C)

Register 5.123. DPORT_APP_SPI_INTR_2_MAP_REG (0x290)

Register 5.124. DPORT_APP_SPI_INTR_3_MAP_REG (0x294)

Register 5.125. DPORT_APP_I2S0_INT_MAP_REG (0x298)

Register 5.126. DPORT_APP_I2S1_INT_MAP_REG (0x29C)

Register 5.127. DPORT_APP_UART_INTR_MAP_REG (0x2A0)

Register 5.128. DPORT_APP_UART1_INTR_MAP_REG (0x2A4)

Register 5.129. DPORT_APP_UART2_INTR_MAP_REG (0x2A8)

Register 5.130. DPORT_APP_SDIO_HOST_INTERRUPT_MAP_REG (0x2AC)

Register 5.131. DPORT_APP_EMAC_INT_MAP_REG (0x2B0)

Register 5.132. DPORT_APP_PWM0_INTR_MAP_REG (0x2B4)

Register 5.133. DPORT_APP_PWM1_INTR_MAP_REG (0x2B8)

Register 5.134. DPORT_APP_LEDC_INT_MAP_REG (0x2C4)

Register 5.135. DPORT_APP_EFUSE_INT_MAP_REG (0x2C8)

Register 5.136. DPORT_APP_TWAI_INT_MAP_REG (0x2CC)

Register 5.137. DPORT_APP_RTC_CORE_INTR_MAP_REG (0x2D0)

Register 5.138. DPORT_APP_RMT_INTR_MAP_REG (0x2D4)

Register 5.139. DPORT_APP_PCNT_INTR_MAP_REG (0x2D8)

Register 5.140. DPORT_APP_I2C_EXT0_INTR_MAP_REG (0x2DC)

Register 5.141. DPORT_APP_I2C_EXT1_INTR_MAP_REG (0x2E0)

Register 5.142. DPORT_APP_RSA_INTR_MAP_REG (0x2E4)

Register 5.143. DPORT_APP_SPI1_DMA_INT_MAP_REG (0x2E8)

Register 5.144. DPORT_APP_SPI2_DMA_INT_MAP_REG (0x2EC)

Register 5.145. DPORT_APP_SPI3_DMA_INT_MAP_REG (0x2F0)

Register 5.146. DPORT_APP_WDG_INT_MAP_REG (0x2F4)

Register 5.147. DPORT_APP_TIMER_INT1_MAP_REG (0x2F8)

Register 5.148. DPORT_APP_TIMER_INT2_MAP_REG (0x2FC)

Register 5.149. DPORT_APP_TG_T0_EDGE_INT_MAP_REG (0x300)

Register 5.150. DPORT_APP_TG_T1_EDGE_INT_MAP_REG (0x304)

Register 5.151. DPORT_APP_TG_WDT_EDGE_INT_MAP_REG (0x308)

Register 5.152. DPORT_APP_TG_LACT_EDGE_INT_MAP_REG (0x30C)

Register 5.153. DPORT_APP_TG1_T0_EDGE_INT_MAP_REG (0x310)

Register 5.154. DPORT_APP_TG1_T1_EDGE_INT_MAP_REG (0x314)

Register 5.155. DPORT_APP_TG1_WDT_EDGE_INT_MAP_REG (0x318)

Register 5.156. DPORT_APP_TG1_LACT_EDGE_INT_MAP_REG (0x31C)

Register 5.157. DPORT_APP_MMU_IA_INT_MAP_REG (0x320)

Register 5.158. DPORT_APP_MPU_IA_INT_MAP_REG (0x324)

Register 5.159. DPORT_APP_CACHE_IA_INT_MAP_REG (0x328)

(reserved)	DPORT_APP_*_MAP	
31 5	4 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	10000	Reset

DPORT_APP_*_MAP 中断对应关系配置寄存器。（读/写）

Register 5.160. DPORT_AHBLITE_MPU_TABLE_UART_REG (0x32C)

Register 5.161. DPORT_AHBLITE_MPU_TABLE_SPI1_REG (0x330)

Register 5.162. DPORT_AHBLITE_MPU_TABLE_SPI0_REG (0x334)

Register 5.163. DPORT_AHBLITE_MPU_TABLE_GPIO_REG (0x338)

Register 5.164. DPORT_AHBLITE_MPU_TABLE_RTC_REG (0x348)

Register 5.165. DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG (0x34C)

Register 5.166. DPORT_AHBLITE_MPU_TABLE_HINF_REG (0x354)

Register 5.167. DPORT_AHBLITE_MPU_TABLE_UHCI1_REG (0x358)

Register 5.168. DPORT_AHBLITE_MPU_TABLE_I2S0_REG (0x364)

Register 5.169. DPORT_AHBLITE_MPU_TABLE_UART1_REG (0x368)

Register 5.170. DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG (0x374)

Register 5.171. DPORT_AHBLITE_MPU_TABLE_UHCI0_REG (0x378)

Register 5.172. DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG (0x37C)

Register 5.173. DPORT_AHBLITE_MPU_TABLE_RMT_REG (0x380)

Register 5.174. DPORT_AHBLITE_MPU_TABLE_PCNT_REG (0x384)

Register 5.175. DPORT_AHBLITE_MPU_TABLE_SLC_REG (0x388)

Register 5.176. DPORT_AHBLITE_MPU_TABLE_LEDC_REG (0x38C)

Register 5.177. DPORT_AHBLITE_MPU_TABLE_EFUSE_REG (0x390)

Register 5.178. DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG (0x394)

Register 5.179. DPORT_AHBLITE_MPU_TABLE_PWM0_REG (0x39C)

Register 5.180. DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG (0x3A0)

Register 5.181. DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG (0x3A4)

Register 5.182. DPORT_AHBLITE_MPU_TABLE_SPI2_REG (0x3A8)

Register 5.183. DPORT_AHBLITE_MPU_TABLE_SPI3_REG (0x3AC)

Register 5.184. DPORT_AHBLITE_MPU_TABLE_SYSCON_REG (0x3B0)

Register 5.185. DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG (0x3B4)

Register 5.186. DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG (0x3B8)

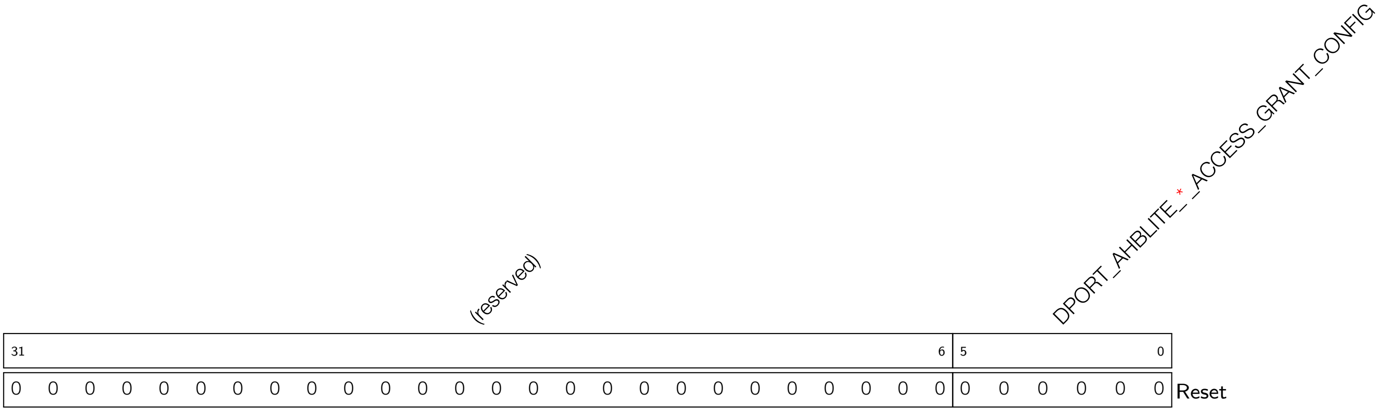
Register 5.187. DPORT_AHBLITE_MPU_TABLE_EMAC_REG (0x3BC)

Register 5.188. DPORT_AHBLITE_MPU_TABLE_PWM1_REG (0x3C4)

Register 5.189. DPORT_AHBLITE_MPU_TABLE_I2S1_REG (0x3C8)

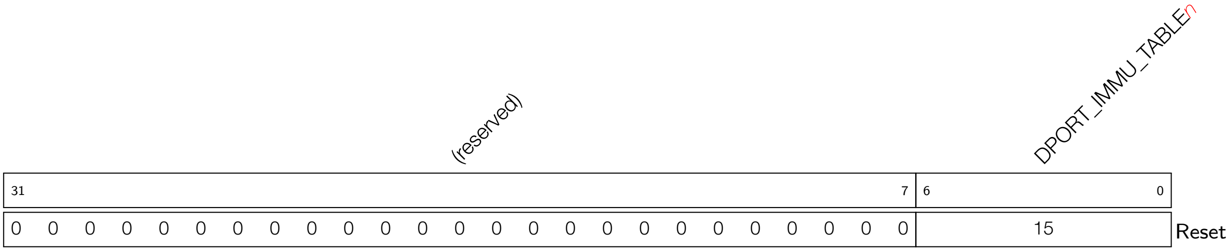
Register 5.190. DPORT_AHBLITE_MPU_TABLE_UART2_REG (0x3CC)

Register 5.191. DPORT_AHBLITE_MPU_TABLE_PWR_REG (0x3E4)



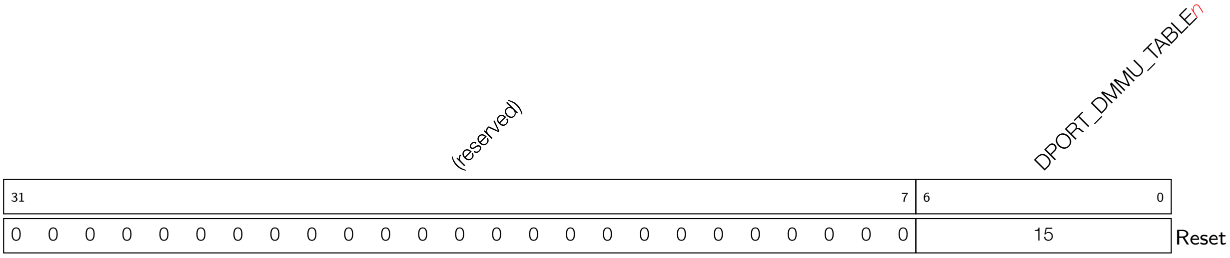
DPORT_AHBLITE_*_ACCESS_GRANT_CONFIG 配置外设 MPU。(读/写)

Register 5.192. DPORT_IMMU_TABLE*n*_REG (*n*: 0-15) (0x504+4**n*)



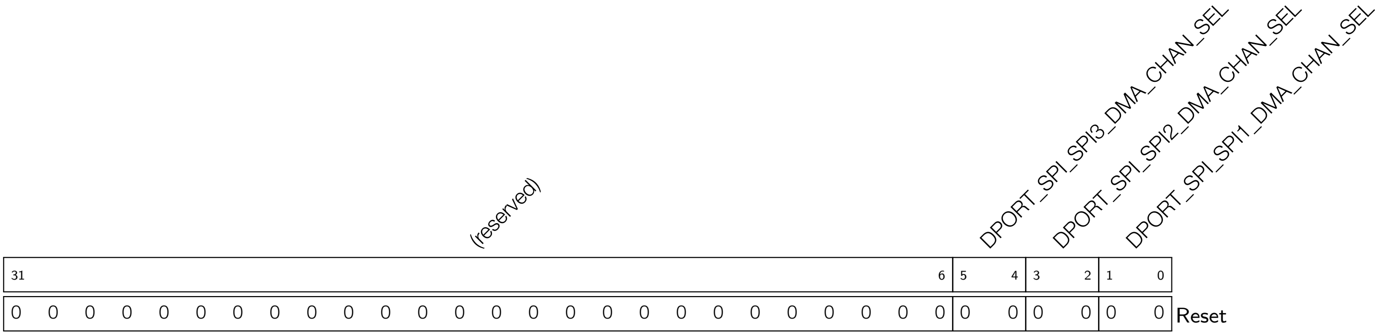
DPORT_IMMU_TABLE*n* 配置 Internal SRAM 的 MMU。*n* 为 0 ~ 9 时，reset 值为 0。*n* 为 10 ~ 15 时，reset 值分别为 10, 11, 12, 13, 14, 15。(读/写)

Register 5.193. DPORT_DMMU_TABLE*n*_REG (*n*: 0-15) (0x544+4**n*)



DPORT_DMMU_TABLE*n* 配置 Internal SRAM 的 MMU。*n* 为 0 ~ 15 时，reset 值分别为 0 ~ 15。(读/写)

Register 5.194. DPORT_SPI_DMA_CHAN_SEL_REG (0x5A8)



DPORT_SPI_SPI3_DMA_CHAN_SEL 选择 SPI3 的 DMA 信道。（读/写）

DPORT_SPI_SPI2_DMA_CHAN_SEL 选择 SPI2 的 DMA 信道。（读/写）

DPORT_SPI_SPI1_DMA_CHAN_SEL 选择 SPI1 的 DMA 信道。（读/写）

6 DMA 控制器 (DMA)

6.1 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 中有 13 个外设都具有 DMA 功能，这 13 个外设是：UART0、UART1、UART2、SPI1、SPI2、SPI3、I2S0、I2S1、SDIO slave、SD/MMC host、EMAC、BT 和 Wi-Fi。

6.2 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
- 支持 4-beat burst 传输
- 328 KB DMA 地址空间
- 通过 DMA 实现高速数据传输

6.3 功能描述

ESP32 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 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。

6.3.1 DMA 引擎的架构

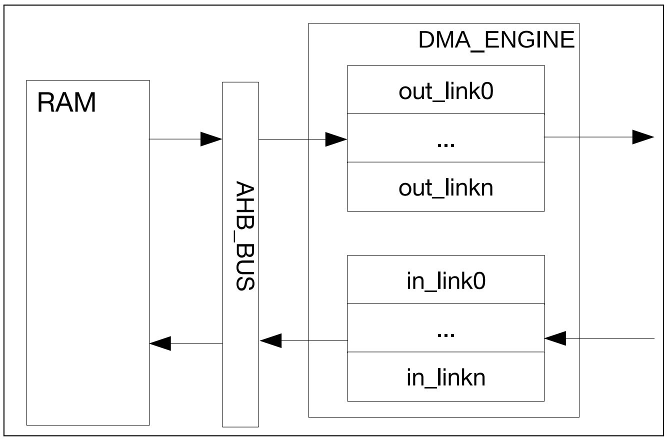


图 6-1. DMA 引擎的架构

DMA 引擎通过 AHB_BUS 将数据存入内部 RAM 或者将数据从 RAM 取出。图 6-1 为 DMA 引擎基本架构图。其中 RAM 为 ESP32 的内部 SRAM，SRAM 的具体使用范围详见章节系统和存储器。软件可以通过挂载链表的方

式来使用 DMA 引擎。DMA_ENGING 根据 out_link 中的内容将相应 RAM 中的数据发送出去，也可根据 in_link 中的内容将接收的数据存入指定 RAM 地址空间。

6.3.2 链表

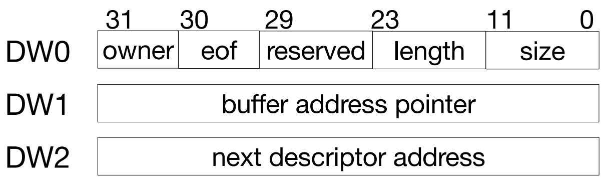


图 6-2. 链表结构图

out_link 与 in_link 结构相同，图 6-2 所示为链表的结构图，一个链表由 3 个字组成。每一字段的意义如下：

- owner (DW0) [31]：表示当前链表对应的 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。
- eof (DW0) [30]：表示结束标志。
  1’b0：当前链表不是最后一个链表；
  1’b1：当前链表为数据包的最后一个链表。
- reserved (DW0) [29:24]：reserved。
  软件不能写 1。
- length (DW0) [23:12]：表示当前链表对应的 buffer 中的有效字节数。从 buffer 中读取数据时表示能够读取的字节数；向 buffer 中存储数据时表示已存数据的字节数。
- size (DW0) [11:0]：表示当前链表对应的 buffer 的大小。
  注意：大小必须字对齐。
- buffer address pointer (DW1)：buffer 地址指针。
  注意：地址必须字对齐。
- next descriptor address (DW2)：下一个链表的地址指针。当前链表为最后一个链表时 (eof=1)，该值为 0。

用 DMA 接收数据时，如果一帧数据长度小于给定的 buffer 长度，那么 DMA 不会接着使用这个 buffer 剩余空间。这使得 DMA_ENGING 可以用于传输任意字节数的数据。

6.4 UART DMA (UDMA) 控制器

ESP32 中有 3 个 UART 接口，它们共用 2 个 UDMA 控制器。UHCI_UART*x*_CE（*x* 为 0、1、2）寄存器用于选择哪一个 UART 使用当前的 UDMA。

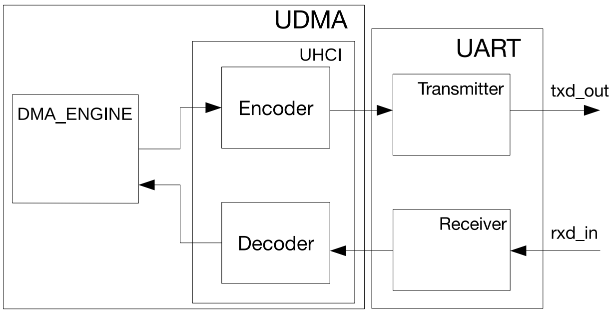


图 6-3. UDMA 模式数据传输

图 6-3 为 UDMA 方式数据传输图。在接收数据前，软件将接收链表准备好。UHCI_INLINK_ADDR 用于指向第一个 in_link 链表。寄存器必须配置接收链表的低 20 位地址。置位 UHCI_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 DMA 引擎的控制下存入接收链表指定的 RAM 空间。

在发送数据前，软件需要将发送链表和发送数据准备好，UHCI_OUTLINK_ADDR 用于指向第一个 out_link 链表。寄存器必须配置发送链表的低 20 位地址。置位 UHCI_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。

UART DMA 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 UHCI_SEPER_CHAR 进行配置，默认值为 0xC0。数据中与分隔符一样的数据可以用 UHCI_ESC_SEQ0_CHAR0（默认为 0xDB）和 UHCI_ESC_SEQ0_CHAR1（默认为 0xDD）进行替换。当数据全部发送完成后，会产生 UHCI_OUT_TOTAL_EOF_INT 中断。当数据接收完成后，会产生 UHCI_IN_SUC_EOF_INT 中断。

说明：

需要注意的是，接收链表描述符的参数 buffer address pointer 需要按字对齐，并且最后一个描述符的参数 size 需要至少比接收的数据长度大 4 字节。

6.5 SPI DMA 控制器

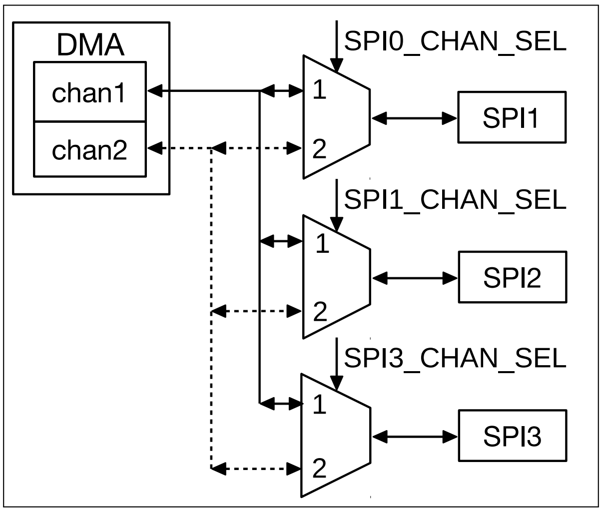


图 6-4. SPI DMA

ESP32 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 6-4 所示，共有两个 DMA 通道可供 SPI1、SPI2 和 SPI3 控制器选择，每个 DMA 通道可供一个 SPI 控制器使用，即每次可以有两个 SPI 控制器同时使用 DMA。

ESP32 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/发送的数据长度应当与 4 字节对齐，以确保正确收发数据。SPI DMA 支持连续收发数据。

配置 DPORT_SPI_DMA_CHAN_SEL_REG 寄存器的 SPI1_DMA_CHAN_SEL[1:0]、SPI2_DMA_CHAN_SEL[1:0] 和 SPI3_DMA_CHAN_SEL[1:0] 三个域以使能 SPI DMA 接口。每个 SPI 控制器对应一个域，每个域有 2 比特，对应取值为 0、1 和 2，而不可以取 3 。

以 SPI1 为例，
若 SPI1_DMA_CHAN_SEL[1:0] = 0，那么 SPI1 不使用 DMA 通道；
若 SPI1_DMA_CHAN_SEL[1:0] = 1，那么 SPI1 使能 DMA 通道 1；
若 SPI1_DMA_CHAN_SEL[1:0] = 2，那么 SPI1 使能 DMA 通道 2。

寄存器 SPI_DMA_OUT_LINK_REG 的 SPI_OUTLINK_START 比特和寄存器 SPI_DMA_IN_LINK_REG 的 SPI_INLINK_START 比特用于使能 DMA 引擎，这两个比特由硬件清零。当 SPI_OUTLINK_START 比特被置为 1 时，DMA 引擎开始处理发送链表，并准备发送数据；当 SPI_INLINK_START 比特被置为 1 时，DMA 引擎开始处理接收链表，并准备接收数据。

SPI_DMA 接口的软件配置流程如下：

1. 首先复位 DMA 状态机和 FIFO 指针；
2. 配置 DMA 相关寄存器；
3. 配置 SPI 接口相关寄存器；
4. 使能一次 DMA 操作。

6.6 I2S DMA 控制器

ESP32 有两个 I2S 接口，即 I2S0 和 I2S1。I2S0 和 I2S1 各有一个 DMA 通道。寄存器 I2S_FIFO_CONF_REG 的 REG_I2S_DSCR_EN 比特用于使能 I2S 的 DMA 操作。ESP32 I2S DMA 使用链表接收/发送数据，发送数据支持 burst 操作，一次接收/发送的数据长度为 1 个字（4 个字节）。寄存器 I2S_RXEOF_NUM_REG 的 REG_I2S_RX_EOF_NUM[31:0] 比特用于配置 DMA 一次接收数据长度，单位为字。

寄存器 I2S_OUT_LINK_REG 的 I2S_OUTLINK_START 比特和寄存器 I2S_IN_LINK_REG 的 I2S_INLINK_START 比特用于使能 DMA 引擎, 这两个比特由硬件清零。当 I2S_OUTLINK_START 比特被置为 1 时，DMA 引擎开始处理发送链表，并准备发送数据，当 I2S_INLINK_START 比特被置为 1 时，DMA 引擎开始处理接收链表，并准备接收数据。

I2S DMA 接口的软件配置流程如下：

1. 首先配置 I2S 接口相关寄存器；
2. 复位 DMA 状态机和 FIFO 指针；
3. 配置 DMA 相关寄存器；
4. 在 I2S 主机模式下，设置 I2S_TX_START 比特或者 I2S_RX_START 比特，发起一次 I2S 操作；
   在 I2S 从机模式下，设置 I2S_TX_START 比特或者 I2S_RX_START 比特后等待主机发起数据传输的请求。

I2S DMA 中断说明详见章节 I2S，DMA 中断 。

7 SPI 控制器 (SPI)

7.1 概述

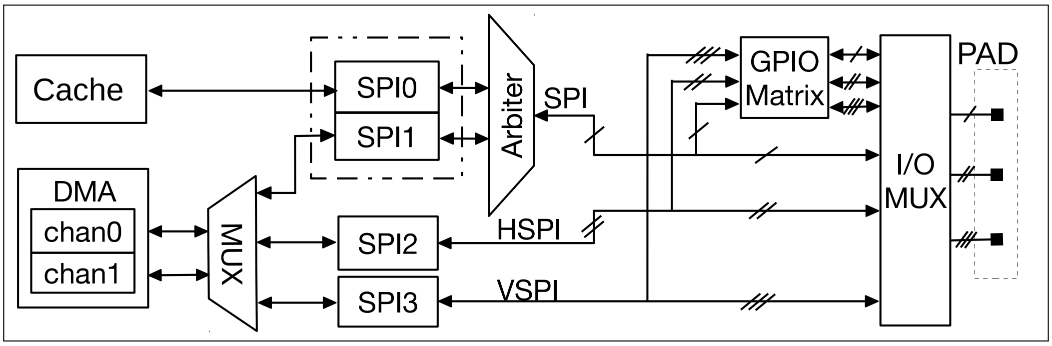


图 7-1. SPI 系统框图

如图 7-1 所示，ESP32 共有 4 个 SPI 控制器 SPI0、SPI1、SPI2、SPI3，用于连接支持 SPI 协议的设备。SPI0 控制器作为 cache 访问外部存储单元接口使用，SPI1 作为主机使用，SPI2 和 SPI3 控制器既可作为主机使用又可作为从机使用。作主机使用时，每个 SPI 控制器可以使用多个片选信号 (CS0 ~ CS2) 来连接多个 SPI 从机设备。SPI1 ~ SPI3 控制器共享两个 DMA 通道。

SPI0 和 SPI1 控制器通过一个仲裁器共用一组信号总线，这组带前缀 SPI 的信号总线由 D、Q、CS0 ~ CS2、CLK、WP 和 HD 信号组成，如表 7-1 所示。相应地，控制器 SPI2 和 SPI3 分别使用带前缀 HSPI 和 VSPI 的信号总线。这些信号总线包含的输入输出信号线可以经过 GPIO 交换矩阵和 IO_MUX 模块实现与芯片管脚的映射（详见章节 IO_MUX）。

SPI 控制器在 GP-SPI 模式下，支持标准的四线全双工/半双工通信（MOSI、MISO、CS、CLK）和三线半双工通信（DATA、CS、CLK）。SPI 控制器在 QSPI 模式下使用信号总线 D、Q、CS0 ~ CS2、CLK、WP 和 HD 作为 4-bit 并行 SPI 总线来访问外部 flash 或 SRAM。不同模式下管脚功能信号与总线信号的对应关系如表 7-1 所示。

表 7-1. 管脚功能信号与总线信号映射关系

GP-SPI 四线 全双工/半双工信号总线	GP-SPI 三线 半双工信号总线	QSPI 信号总线	引脚功能信号		
			SPI 信号总线	HSPI 信号总线	VSPI 信号总线
MOSI	DATA	D	SPID	HSPID	VSPID
MISO	-	Q	SPIQ	HSPIQ	VSPIQ
CS	CS	CS	SPICS0	HSPICS0	VSPICS0
CLK	CLK	CLK	SPICLK	HSPICLK	VSPICLK
-	-	WP	SPIWP	HSPIWP	VSPIWP
-	-	HD	SPIHD	HSPIHD	VSPIHD

7.2 SPI 特征

GP-SPI（通用 SPI）接口

- 数据传输长度以 byte 为单位可配置
- 支持四线全双工/半双工通信和三线半双工通信

- 主机模式和从机模式
- 时钟极性 (CPOL) 和时钟相位 (CPHA) 可配置
- 时钟可配置

并行 QSPI 接口

- 支持诸如 flash 等特殊从机设备的通信格式
- 可配置的通信格式
- 支持 6 种读 flash 操作
- 支持访问 flash 和 SRAM 自动切换
- 支持自动等待 flash 空闲

SPI DMA 接口

- 支持使用链表收/发数据

SPI 中断接口

- SPI 中断
- SPI DMA 中断

7.3 GP-SPI 接口

ESP32 SPI 支持四线全双工/半双工通信和三线半双工通信。四线全双工/半双工通信电气连接如图 7-2 所示。

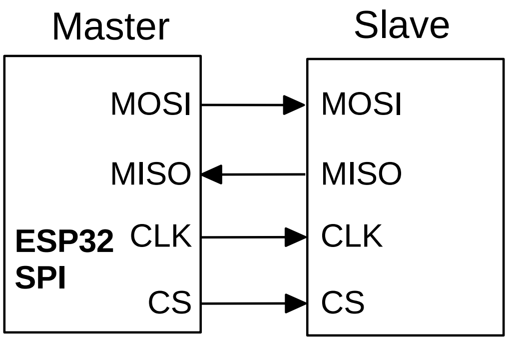


图 7-2. SPI 四线全双工/半双工通信

ESP32 SPI1 ~ SPI3 可以作为 SPI 主机与其他从机通信，SPI2 和 SPI3 也可以作为从机。每个 ESP32 SPI 主机默认最多可以接 3 个从机。在非 DMA 模式下，一次最多可以接收/发送 64 byte 的数据，收发数据长度以字节为单位。

7.3.1 GP-SPI 四线全双工模式

当配置成四线全双工模式时，ESP32 SPI 既可以作为主机也可以作为从机使用。使用时，软件需要配置接收和发送数据的长度：在主机模式下，通过寄存器 SPI_MISO_DLEN_REG 和 SPI_MOSI_DLEN_REG 配置；在从机模式下，通过寄存器 SPI_SLV_RDBUF_DLEN_REG 和 SPI_SLV_WRBUF_DLEN_REG 配置。还需要配置寄存器 SPI_USER_REG 的 SPI_DOUTDIN 位和 SPI_USR_MOSI 位来使能四线全双工模式。最后需要配置寄存器 SPI_CMD_REG 的 SPI_USR 位来启动一次数据传输。

7.3.2 GP-SPI 四线半双工模式

当配置成四线半双工模式时，ESP32 SPI 既可以作为主机也可以作为从机使用。此时 ESP32 SPI 具有灵活的通信格式：命令 + 地址 + 等待 + 接收和/或发送数据。具体为：

1. 命令：长度 0 ~ 16 bit；主机输出从机输入。
2. 地址：长度 0 ~ 32 bit/64 bit；主机输出从机输入。
3. 等待：长度 0 ~ 256 个 SPI 时钟。
4. 接收和/或发送数据：长度 0 ~ 512 bit (64 byte)；主机输出从机输入或主机输入从机输出。

其中 GP-SPI 主机模式的地址状态最大长度为 32 bit，QSPI 主机模式的地址状态最大长度为 64 bit。命令状态、地址状态、等待状态和收/发数据状态分别对应 SPI_USER_REG 寄存器的 SPI_USR_COMMAND、SPI_USR_ADDR、SPI_USR_DUMMY 和 SPI_USR_MISO/SPI_USR_MOSI 等域。仅当这些比特位被置 1 时，一次 SPI 操作才会经历该状态，详见章节寄存器说明。作为主机使用时，软件可以根据需要通过配置寄存器来选择是否包含命令、地址、等待和接收或发送数据等状态。

作为从机使用时，传输时必须包含命令、地址和接收或发送数据状态，并且从机命令固定为表 7-2。在一次收/发过程中，片选信号 CS 必须保持低电平。如果在发送过程中 CS 被拉高，从机内部状态将会复位。

表 7-2. 从机命令描述

命令	描述
0x1	从机接收，将主机发送数据通过 MOSI 写入从机状态寄存器
0x2	从机接收，将主机发送数据通过 MOSI 写入从机数据缓存
0x3	从机发送，将从机缓存中的数据通过 MISO 发送到主机
0x4	从机发送，将从机状态寄存器中的数据通过 MISO 发送到主机
0x6	先将 MOSI 上的主机数据写入数据缓存，然后再将从机数据缓存中的数据发送至 MISO

主机可以写 SPI 从机的状态寄存器 SPI_SLV_WR_STATUS_REG，并可以通过 SPI_SLAVE1_REG 寄存器中的 SPI_SLV_STATUS_READBACK 位，决定读 SPI_SLV_WR_STATUS_REG 寄存器还是 SPI_RD_STATUS_REG 寄存器的数据。SPI 主机可以通过读写从机状态寄存器，达到与从机保持沟通的目的，以此实现较复杂的通信。

软件通过主机模式的 SPI_MISO_DLEN_REG 和 SPI_MOSI_DLEN_REG 寄存器、从机模式的 SPI_SLV_RDBUF_DLEN_REG、SPI_SLV_WRBUF_DLEN_REG 来配置 SPI 接收或发送数据的长度。通过配置寄存器 SPI_USER_REG 的 SPI_USR_MOSI 位或 SPI_USR_MISO 位来控制 SPI 接收或发送数据。最后配置寄存器 SPI_CMD_REG 的 SPI_USR 位来启动一次数据传输。

7.3.3 GP-SPI 三线半双工模式

ESP32 SPI 的三线半双工和四线半双工的区别在于接收和发送数据使用同一根信号线，且必须包含命令、地址和接收或发送数据状态。软件需要通过配置 SPI_USER_REG 寄存器的 SPI_SIO 位来使能三线半双工模式。

说明：

- 在半双工模式下，命令、地址和接收或发送数据的顺序不可变；
- 在半双工模式下，“命令 + 地址 + 收收据 + 发数据” 和 “收数据 + 发数据” 这两种通信格式不支持 DMA；

- ESP32 SPI 作为从机时，主机 CS 需要提前至少一个 SPI 时钟长度有效；读/写结束后，CS 需要至少延迟一个 SPI 时钟长度失效。

7.3.4 GP-SPI 数据缓存

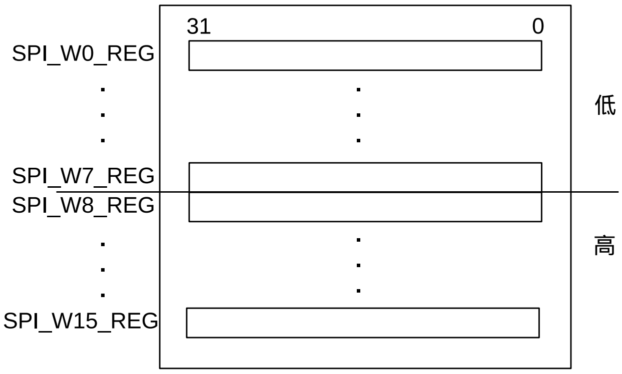


图 7-3. SPI 数据缓存

ESP32 SPI 共有大小为 16 × 32 bit 的数据缓存，用于存储发送/接收的数据。如图 7-3 所示。接收数据时，数据默认从 SPI_W0_REG 的低字节部分开始填充，一直到 SPI_W15_REG 结束。如果数据长度大于 64 byte，多出部分从 SPI_W0_REG 开始继续填充。

数据缓存的 SPI_W0_REG ~ SPI_W7_REG 和 SPI_W8_REG ~ SPI_W15_REG 分别对应低和高两个部分，他们可以分开使用。由寄存器 SPI_USER_REG 的 SPI_USR_MOSI_HIGHPART 和 SPI_USR_MISO_HIGHPART 两个比特控制。例如，当 SPI 作为主机，当 SPI_USR_MOSI_HIGHPART = 1 时，SPI_W8_REG ~ SPI_W15_REG 作为发送数据缓存使用，当 SPI_USR_MISO_HIGHPART = 1 时，SPI_W8_REG-SPI_W15_REG 作为接收数据缓存使用。而当 SPI 作为从机时，若 SPI_USR_MOSI_HIGHPART = 1，则 SPI_W8_REG ~ SPI_W15_REG 作为接收数据缓存使用，若 SPI_USR_MISO_HIGHPART = 1，则 SPI_W8_REG-SPI_W15_REG 作为发送数据缓存使用。

7.4 GP-SPI 时钟控制

ESP32 GP-SPI 主机输出时钟频率最高为 $f_{\text{apb}}/2$，从机输入时钟最高为 $f_{\text{apb}}/8$。主机可以通过分频得到其他时钟频率。

$$f_{\text{spi}} = \frac{f_{\text{apb}}}{\text{(SPI\_CLKCNT\_N+1)(SPI\_CLKDIV\_PRE+1)}}$$

其中 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 为寄存器 SPI_CLOCK_REG 的两个位（详见 7.7 寄存器描述），SPI_CLKCNT_H = $\lfloor \frac{\text{SPI\_CLKCNT\_N+1}}{2} - 1 \rfloor$，SPI_CLKCNT_N=SPI_CLKCNT_L。当寄存器 SPI_CLOCK_REG 的
SPI_CLK_EQU_SYSCLK 位置 1，其他位置为 0 时，SPI 输出时钟为 f_{apb}；除此之外，SPI_CLK_EQU_SYSCLK 位均需置为 0。在从机模式下，SPI_CLKCNT_N、SPI_CLKCNT_L、SPI_CLKCNT_H 和 SPI_CLKDIV_PRE 均需要置为 0。

7.4.1 GP-SPI 时钟极性和时钟相位

ESP32 SPI 的时钟极性和相位，由寄存器 SPI_PIN_REG 的 SPI_CK_IDLE_EDGE 位、寄存器 SPI_USER_REG 的 SPI_CK_OUT_EDGE 位与 SPI_CK_I_EDGE 位，以及 SPI_CTRL2_REG 寄存器的 SPI_MISO_DELAY_MODE[1:0] 位、SPI_MISO_DELAY_NUM[2:0] 位、SPI_MOSI_DELAY_MODE[1:0] 位与 SPI_MOSI_DELAY_MUM[2:0] 位控制。表 7-3 和表 7-4 分别对应为 ESP32 SPI 主机和从机时钟极性和相位控制

及其对应寄存器值。表 7-4 中 mode0 和 mode2 分为 DMA 模式和非 DMA 模式下的寄存器配置。需要注意的是 DMA 模式下 mode0 和 mode2 从机的数据会提前输出。

表 7-3. 主机模式时钟极性和相位控制寄存器值

寄存器	mode0	mode1	mode2	mode3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0
SPI_MISO_DELAY_MODE	2(0)	1(0)	1(0)	2(0)
SPI_MISO_DELAY_NUM	0	0	0	0
SPI_MOSI_DELAY_MODE	0	0	0	0
SPI_MOSI_DELAY_NUM	0	0	0	0

表 7-4. 从机模式时钟极性和相位控制寄存器值

寄存器	mode0		mode1	mode2		mode3
	非 DMA	DMA		非 DMA	DMA	
SPI_CK_IDLE_EDGE	1	0	1	0	1	0
SPI_CK_I_EDGE	0	1	1	1	0	0
SPI_MISO_DELAY_MODE	0	0	2	0	0	1
SPI_MISO_DELAY_NUM	0	2	0	0	2	0
SPI_MOSI_DELAY_MODE	2	0	0	1	0	0
SPI_MOSI_DELAY_NUM	2	3	0	2	3	0

1. mode0 表示 CPOL=0, CPHA=0，SPI 空闲时，时钟的输出为低电平，数据在 SPI 下降沿变化，在上升沿采样；
2. mode1 表示 CPOL=0, CPHA=1，SPI 空闲时，时钟的输出为低电平，数据在 SPI 上升沿变化，在下降沿采样；
3. mode2 表示 CPOL=1, CPHA=0，SPI 空闲时，时钟的输出为高电平，数据在 SPI 上升沿变化，在下降沿采样；
4. mode3 表示 CPOL=1, CPHA=1，SPI 空闲时，时钟的输出为高电平，数据在 SPI 下降沿变化，在上升沿采样。

7.4.2 GP-SPI 时序

ESP32 GP-SPI 接口的数据信号既可以通过 IO_MUX 映射到管脚，又可以通过 IO_MUX 和 GPIO 交换矩阵映射到管脚。当信号通过交换矩阵时，输入信号会被延迟两个 clk_{apb} 时钟周期，输出信号不会被延迟。

GP-SPI 作为主机，并且信号不经过 GPIO 交换矩阵进入到 SPI 控制器时，若 GP-SPI 输出时钟频率为 $clk_{\text{apb}}/2$，则在配置时钟极性时，寄存器 SPI_MISO_DELAY_MODE 需要置为 0。若 GP-SPI 输出时钟频率不大于 $clk_{\text{apb}}/4$，则在配置时钟极性时，SPI_MISO_DELAY_MODE 可以置为表 7-3 中的对应数值。

GP-SPI 作为主机时，并且信号经过 GPIO 交换矩阵进入到 SPI 控制器：

1. 如果 GP-SPI 输出时钟频率为 $clk_{\text{apb}}/2$，则在配置时钟极性时，寄存器 SPI_MISO_DELAY_MODE 需要置为 0，同时需要使能等待状态 (SPI_USR_DUMMY = 1)，等待长度为 1 个 clk_{spi} 时钟周期 (SPI_USR_DUMMY_CYCLELEN = 0)；

2. 如果 GP-SPI 输出时钟频率为 $clk_{\text{apb}}/4$，在配置时钟极性时寄存器 SPI_MISO_DELAY_MODE 需要置为 0；
3. 如果 GP-SPI 输出时钟频率不大于 $clk_{\text{apb}}/8$，则在配置时钟极性时，SPI_MISO_DELAY_MODE 可以置为表 7-3 中的对应数值。

GP-SPI 作为从机时，要求时钟信号和数据信号选择相同的方式进入 SPI 控制器，即时钟信号和数据信号都不经过或都经过 GPIO 交换矩阵进入 SPI 控制器。这样才能防止在信号到达 SPI 硬件之前，延迟的时间不同。

以 mode0 为例，图 7-4 中 t_{spi}、t_{pre} 和 t_{v} 分别为 SPI 时钟周期、数据输出提前时间和数据输出延后时间，同时假设 SPI 从机主时钟周期为 t_{apb}。在非 DMA 模式下，从机数据会延后输出，延后时间 t_{v}：

- 如果 CLK 不经过交换矩阵输入，那么 $t_{\text{v}} < 3.5 * t_{\text{apb}}$；
- 如果 CLK 经过交换矩阵输入，那么 $t_{\text{v}} < 5.5 * t_{\text{apb}}$；

在 DMA 模式下，mode1 和 mode3，从机数据会延后输出，延后时间 t_{v} 和非 DMA 模式相同。而在 mode0 和 mode2，从机数据会提前输出，提前时间 t_{pre}：

- 如果 clk 不经过交换矩阵输入，那么 $t_{\text{pre}} < (t_{\text{spi}}/2 - 5.5 * t_{\text{apb}})$；
- 如果 clk 经过交换矩阵输入，那么 $t_{\text{pre}} < (t_{\text{spi}}/2 - 7.5 * t_{\text{apb}})$；

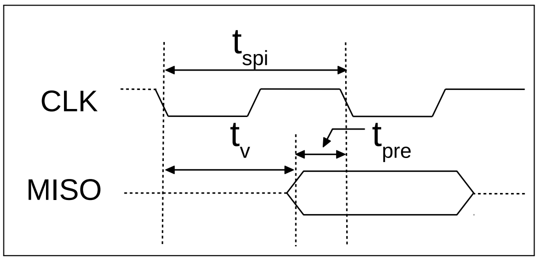


图 7-4. GP-SPI 从机数据输出

因此，如果信号不经过交换矩阵输入从机，那么从机时钟最高为 $f_{\text{apb}}/8$，如果信号经过交换矩阵输入从机，那么从机时钟最高为 $f_{\text{apb}}/12$ 。同时需要注意的是，当 mode0 和 mode2 时，$(t_{\text{spi}}/2 - t_{\text{pre}})$ 也是从机输出数据的保持时间。

7.5 并行 QSPI 接口

ESP32 SPI 控制器对 SPI 接口存储器（如 flash，SRAM）做了特殊的支持。SPI 管脚与存储器的硬件连接如图 7-5 所示。

SPI1、SPI2 和 SPI3 控制器也可以作为 QSPI 接口的主机和外部存储器连接。SPI 存储器接口的最高输出时钟频率为 f_{apb}，时钟配置和 GP-SPI 接口主机时钟配置相同。

7.5.1 并行 QSPI 接口通信格式

ESP32 QSPI 为了支持与特殊从机模式之间的通信，单独设计与之相对应的通信协议。ESP32 QSPI 主机通信格式与 GP-SPI 四线半双工模式相同。不同的是在地址状态和数据状态，软件可以通过配置寄存器使能 2 线或者 4 线模式传输数据，如图 7-6 所示，为地址和数据均为 4 线传输的通信模式。

ESP32 QSPI 接口支持一线模式、两线模式和四线模式的 flash 读操作。与 GP-SPI 相同，作为主机时，QSPI 模式可以根据需要对命令、地址、等待和接收或发送数据等状态进行增减。

注意：在 GPI-SPI 全双工模式下等待状态不可用。

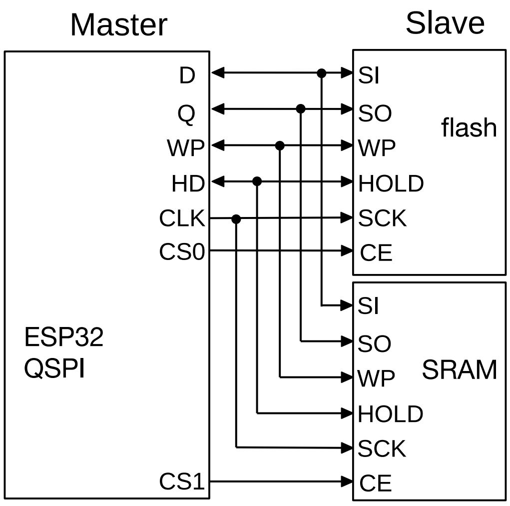


图 7-5. 并行 QSPI 接口

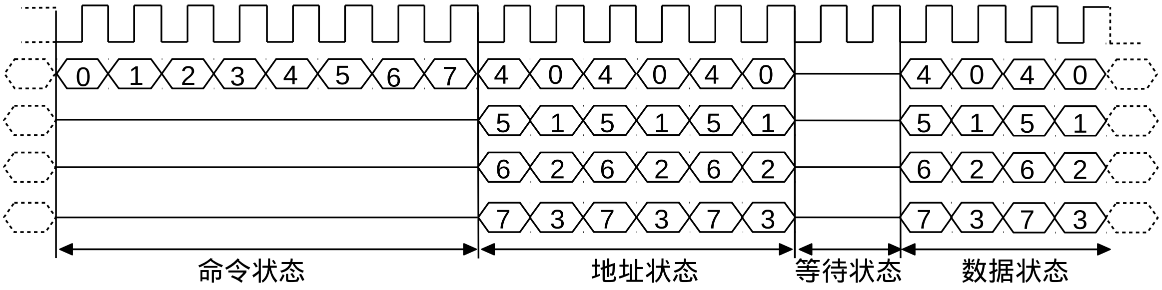


图 7-6. 并行 QSPI 接口的通信模式

7.6 GP-SPI 中断硬件

ESP32 SPI 中断分为两类，一类为 SPI 接口中断，另一类为 SPI DMA 接口中断。

ESP32 SPI 将发送和/或接收两种操作结束时的中断统一成一个，即认为同是控制器一次操作的结束，而不加以区分。ESP32 SPI 作为从机时，根据操作的不同，从机会产生读/写状态寄存器和读/写缓存数据中断。

7.6.1 SPI 中断

通过将寄存器 SPI_SLAVE_REG 的 SPI_*_INTEN 位置 1，可以使能 SPI 接口中断。当中断发生时，对应的 SPI_*_DONE 寄存器的中断标志也会被置 1。中断标志寄存器可写，若要清除中断，只需将对应 bit 置 0。

- SPI_TRANS_DONE_INT：SPI 操作结束被触发。
- SPI_SLV_WR_STA_INT：SPI 从机写状态结束时被触发。
- SPI_SLV_RD_STA_INT：SPI 从机读状态结束时被触发。
- SPI_SLV_WR_BUF_INT：SPI 从机写缓存结束时被触发。
- SPI_SLV_RD_BUD_INT：SPI 从机读缓存结束时被触发。

7.6.2 DMA 中断

- SPI_OUT_TOTAL_EOF_INT：所有链表发送完时被触发。
- SPI_OUT_EOF_INT：一个链表发送完时被触发。

- SPI_OUT_DONE_INT：最后一个链表长度为 0 时被触发。
- SPI_IN_SUC_EOF_INT：所有链表被接受时被触发。
- SPI_IN_ERR_EOF_INT：接收链表出现错误时被触发。
- SPI_IN_DONE_INT：接收的最后一个链表长度为 0 时被触发。
- SPI_INLINK_DSCR_ERROR_INT：接收链表清单错误时被触发。
- SPI_OUTLINK_DSCR_ERROR_INT：要发送的链表无效时被触发。
- SPI_INLINK_DSCR_EMPTY_INT：无有效链表时被触发。

7.7 寄存器列表

名称	描述	SPI0	SPI1	SPI2	SPI3	访问
控制和配置寄存器						
SPI_CTRL_REG	Bit 顺序和 QIO/DIO/QOUT/DOUT 模式设置	3FF43008	3FF42008	3FF64008	3FF65008	读/写
SPI_CTRL2_REG	时序配置	3FF43014	3FF42014	3FF64014	3FF65014	读/写
SPI_CLOCK_REG	时钟配置	3FF43018	3FF42018	3FF64018	3FF65018	读/写
SPI_PIN_REG	极性和 CS 配置	3FF43034	3FF42034	3FF64034	3FF65034	读/写
从机模式配置寄存器						
SPI_SLAVE_REG	从机模式配置与中断状态	3FF43038	3FF42038	3FF64038	3FF65038	读/写
SPI_SLAVE1_REG	从机数据 bit 长度	3FF4303C	3FF4203C	3FF6403C	3FF6503C	读/写
SPI_SLAVE2_REG	等待周期长度配置	3FF43040	3FF42040	3FF64040	3FF65040	读/写
SPI_SLV_WR_STATUS_REG	从机状态/主机低位地址	3FF43030	3FF42030	3FF64030	3FF65030	读/写
SPI_SLV_WRBUF_DLEN_REG	写缓存操作长度	3FF43048	3FF42048	3FF64048	3FF65048	读/写
SPI_SLV_RDBUF_DLEN_REG	读缓存操作长度	3FF4304C	3FF4204C	3FF6404C	3FF6504C	读/写
SPI_SLV_RD_BIT_REG	读数据操作长度	3FF43064	3FF42064	3FF64064	3FF65064	读/写
用户自定义命令模式寄存器						
SPI_CMD_REG	开始用户自定义命令	3FF43000	3FF42000	3FF64000	3FF65000	读/写
SPI_ADDR_REG	地址数据	3FF43004	3FF42004	3FF64004	3FF65004	读/写
SPI_USER_REG	用户自定义命令配置	3FF4301C	3FF4201C	3FF6401C	3FF6501C	读/写
SPI_USER1_REG	地址和等待周期配置	3FF43020	3FF42020	3FF64020	3FF65020	读/写
SPI_USER2_REG	命令长度和值配置	3FF43024	3FF42024	3FF64024	3FF65024	读/写
SPI_MOSI_DLEN_REG	MOSI 长度	3FF43028	3FF42028	3FF64028	3FF65028	读/写
SPI_W0_REG	SPI 数据寄存器 0	3FF43080	3FF42080	3FF64080	3FF65080	读/写
SPI_W1_REG	SPI 数据寄存器 1	3FF43084	3FF42084	3FF64084	3FF65084	读/写
SPI_W2_REG	SPI 数据寄存器 2	3FF43088	3FF42088	3FF64088	3FF65088	读/写
SPI_W3_REG	SPI 数据寄存器 3	3FF4308C	3FF4208C	3FF6408C	3FF6508C	读/写
SPI_W4_REG	SPI 数据寄存器 4	3FF43090	3FF42090	3FF64090	3FF65090	读/写
SPI_W5_REG	SPI 数据寄存器 5	3FF43094	3FF42094	3FF64094	3FF65094	读/写
SPI_W6_REG	SPI 数据寄存器 6	3FF43098	3FF42098	3FF64098	3FF65098	读/写

名称	描述	SPI0	SPI1	SPI2	SPI3	访问
SPI_W7_REG	SPI 数据寄存器 7	3FF4309C	3FF4209C	3FF6409C	3FF6509C	读/写
SPI_W8_REG	SPI 数据寄存器 8	3FF430A0	3FF420A0	3FF640A0	3FF650A0	读/写
SPI_W9_REG	SPI 数据寄存器 9	3FF430A4	3FF420A4	3FF640A4	3FF650A4	读/写
SPI_W10_REG	SPI 数据寄存器 10	3FF430A8	3FF420A8	3FF640A8	3FF650A8	读/写
SPI_W11_REG	SPI 数据寄存器 11	3FF430AC	3FF420AC	3FF640AC	3FF650AC	读/写
SPI_W12_REG	SPI 数据寄存器 12	3FF430B0	3FF420B0	3FF640B0	3FF650B0	读/写
SPI_W13_REG	SPI 数据寄存器 13	3FF430B4	3FF420B4	3FF640B4	3FF650B4	读/写
SPI_W14_REG	SPI 数据寄存器 14	3FF430B8	3FF420B8	3FF640B8	3FF650B8	读/写
SPI_W15_REG	SPI 数据寄存器 15	3FF430BC	3FF420BC	3FF640BC	3FF650BC	读/写
DMA 配置寄存器						
SPI_DMA_CONF_REG	DMA 配置寄存器	3FF43100	3FF42100	3FF64100	3FF65100	读/写
SPI_DMA_OUT_LINK_REG	DMA 发送链表地址与配置	3FF43104	3FF42104	3FF64104	3FF65104	读/写
SPI_DMA_IN_LINK_REG	DMA 接收链表地址与配置	3FF43108	3FF42108	3FF64108	3FF65108	读/写
SPI_DMA_STATUS_REG	DMA 状态	3FF4310C	3FF4210C	3FF6410C	3FF6510C	只读
SPI_IN_ERR_EOF_DES_ADDR_REG	出现错误的描述符地址	3FF43120	3FF42120	3FF64120	3FF65120	只读
SPI_IN_SUC_EOF_DES_ADDR_REG	接收描述符的当前地址	3FF43124	3FF42124	3FF64124	3FF65124	只读
SPI_INLINK_DSCR_REG	当前描述符指针	3FF43128	3FF42128	3FF64128	3FF65128	只读
SPI_INLINK_DSCR_BF0_REG	下一个描述符数据指针	3FF4312C	3FF4212C	3FF6412C	3FF6512C	只读
SPI_INLINK_DSCR_BF1_REG	当前描述符数据指针	3FF43130	3FF42130	3FF64130	3FF65130	只读
SPI_OUT_EOF_BFR_DES_ADDR_REG	带有 EOF 的对应缓存地址	3FF43134	3FF42134	3FF64134	3FF65134	只读
SPI_OUT_EOF_DES_ADDR_REG	带有 EOF 的描述符地址	3FF43138	3FF42138	3FF64138	3FF65138	只读
SPI_OUTLINK_DSCR_REG	当前描述符指针	3FF4313C	3FF4213C	3FF6413C	3FF6513C	只读
SPI_OUTLINK_DSCR_BF0_REG	下一个描述符数据指针	3FF43140	3FF42140	3FF64140	3FF65140	只读
SPI_OUTLINK_DSCR_BF1_REG	当前描述符数据指针	3FF43144	3FF42144	3FF64144	3FF65144	只读
SPI_DMA_RSTATUS_REG	DMA 内存读取状态	3FF43148	3FF42148	3FF64148	3FF65148	只读
SPI_DMA_TSTATUS_REG	DMA 内存写状态	3FF4314C	3FF4214C	3FF6414C	3FF6514C	只读
DMA 中断寄存器						
SPI_DMA_INT_RAW_REG	原始中断状态	3FF43114	3FF42114	3FF64114	3FF65114	只读
SPI_DMA_INT_ST_REG	屏蔽中断状态	3FF43118	3FF42118	3FF64118	3FF65118	只读
SPI_DMA_INT_ENA_REG	中断使能位	3FF43110	3FF42110	3FF64110	3FF65110	读/写
SPI_DMA_INT_CLR_REG	中断清除位	3FF4311C	3FF4211C	3FF6411C	3FF6511C	读/写

7.8 寄存器

本小节括号中的地址均为相对于 SPI0/SPI1/SPI2/SPI3 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 7.7 寄存器列表。

Register 7.1. SPI_CMD_REG (0x0)

(reserved)	SPI_USR	(reserved)	
31 19	18	17 0	
0 0 0 0 0 0 0 0 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

SPI_USR 此位置 1 时, 触发一次 SPI 操作，操作结束后此位被自动清零。(读/写)

Register 7.2. SPI_ADDR_REG (0x4)

31 0	
0x000000000	Reset

SPI_ADDR_REG 用于存储主机在半双工模式或 QSPI 模式的发送地址。如果地址长度大于 32 bit，则此寄存器包含高地址位，SPI_SLV_WR_STATUS_REG 包含低 32 bit。此寄存器仅当 SPI_USR_ADDR 被置 1 时有效。(读/写)

Register 7.3. SPI_CTRL_REG (0x8)

(reserved)	SPI_WR_BIT_ORDER	SPI_RD_BIT_ORDER	SPI_FREAD_QIO	SPI_FREAD_DIO	(reserved)	SPI_WP	SPI_FREAD_QUAD	(reserved)	SPI_FREAD_DUAL	SPI_FASTRD_MODE	(reserved)	
31 27	26	25	24	23	22	21	20	19 15	14	13	12 0	
0 0 0 0 0	0	0	0	0	0	1	0	0 0 0 0 0	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_WR_BIT_ORDER 用于配置发送信号的比特顺序，只对发送的命令、地址和数据有效。1：先发送低有效位；0：先发送高有效位。（读/写）

SPI_RD_BIT_ORDER 用于配置接收信号的比特顺序，只对接收的数据有效。1：先接收低有效位；0：先接收高有效位。（读/写）

SPI_FREAD_QIO 用于使能 4 线模式发送地址和读取数据。仅在 QSPI 模式使用。（读/写）

SPI_FREAD_DIO 用于使能 2 线模式发送地址和读取数据。仅在 QSPI 模式使用。（读/写）

SPI_WP 用于配置空闲时的 WP 信号输出电平。1：输出高电平；2：输出低电平。仅在 QSPI 模式使用。（读/写）

SPI_FREAD_QUAD 用于使能 4 线模式读取数据。仅在 QSPI 模式使用。（读/写）

SPI_FREAD_DUAL 用于使能 2 线模式读取数据。仅在 QSPI 模式使用。（读/写）

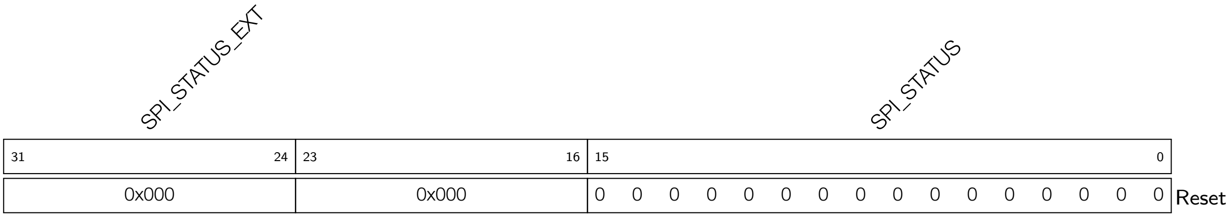
SPI_FASTRD_MODE 保留。

Register 7.4. SPI_CTRL1_REG (0xC)

SPI_CS_HOLD_DELAY	(reserved)	
31 28	27 0	
0x05	0 0 0 0 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

SPI_CS_HOLD_DELAY 保留。

Register 7.5. SPI_RD_STATUS_REG (0x10)



SPI_STATUS_EXT 保留。

SPI_STATUS 保留。

Register 7.6. SPI_CTRL2_REG (0x14)

SPI_CS_DELAY_NUM	SPI_CS_DELAY_MODE	SPI_MOSI_DELAY_NUM	SPI_MOSI_DELAY_MODE	SPI_MISO_DELAY_NUM	SPI_MISO_DELAY_MODE	reserved	reserved	SPI_HOLD_TIME	SPI_SETUP_TIME	
31 28	27 26	25 23	22 21	20 18	17 16	15 12	11 8	7 4	3 0	
0x00	0x0	0x0	0x0	0x0	0x0	0x00	0x00	0x01	0x01	Reset

SPI_CS_DELAY_NUM 保留。

SPI_CS_DELAY_MODE 保留。

SPI_MOSI_DELAY_NUM 用于配置 MOSI 信号基于系统时钟的延迟数。(读/写)

SPI_MOSI_DELAY_MODE 用于配置 MOSI 信号基于 SPI 时钟的延迟的方式。(读/写)
MOSI 信号在延迟 SPI_MOSI_DELAY_NUM 个系统时钟后，再进行如下延迟:
0：无延迟。
1：若 SPI_CK_OUT_EDGE 或 SPI_CK_I_EDGE 被置 1，MOSI 被延迟半个周期，否则被延迟一个周期。
2：若 SPI_CK_OUT_EDGE 或 SPI_CK_I_EDGE 被置 1，MOSI 被延迟一个周期，否则被延迟半个周期。
3：延迟一个周期。

SPI_MISO_DELAY_NUM 用于配置 MISO 信号基于系统时钟的延迟数。(读/写)

SPI_MISO_DELAY_MODE 用于配置 MISO 信号基于 SPI 时钟的延迟的方式。(读/写)
MISO 信号在延迟 SPI_MISO_DELAY_NUM 个系统时钟后，再进行如下延迟:
0：无延迟。
1：若 SPI_CK_OUT_EDGE 或 SPI_CK_I_EDGE 被置 1，MISO 被延迟半个周期，否则被延迟一个周期。
2：若 SPI_CK_OUT_EDGE 或 SPI_CK_I_EDGE 被置 1，MISO 被延迟一个周期，否则被延迟半个周期。
3：延迟一个周期。

SPI_HOLD_TIME CS 信号被延迟的 SPI 时钟周期数。仅当 SPI_CS_HOLD 置 1 时有效。(读/写)

SPI_SETUP_TIME 用于调整 CS 信号有效沿到第一个 SPI 时钟沿的时间。仅在 SPI_CS_SETUP 置 1 时有效。只在半双工模式和 QSPI 模式下有效。(读/写)

Register 7.7. SPI_CLOCK_REG (0x18)

SPI_CLK_EQU_SYSCLK	SPI_CLKDIV_PRE	SPI_CLKCNT_N	SPI_CLKCNT_H	SPI_CLKCNT_L	
31	30 18	17 12	11 6	5 0	
1	0 0 0 0 0 0 0 0 0 0 0 0 0	0x03	0x01	0x03	Reset

SPI_CLK_EQU_SYSCLK 主机模式下，1：SPI 输出时钟等于系统时钟；0：SPI 输出时钟由系统时钟分频而来。从机模式下，此寄存器值需要置为 0。（读/写）

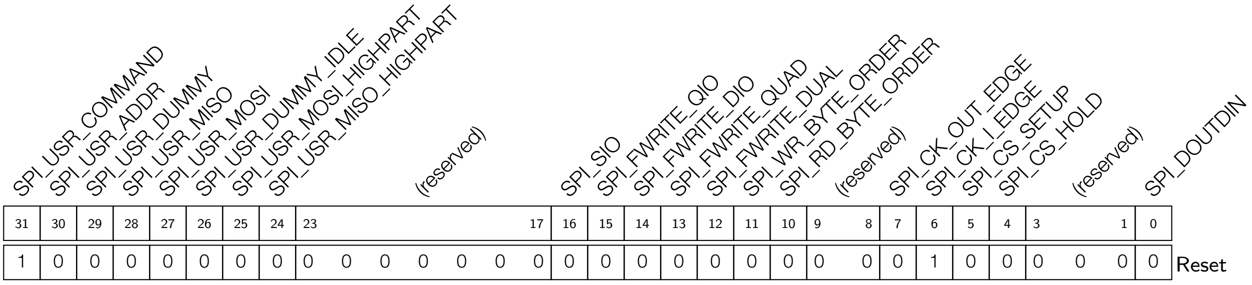
SPI_CLKDIV_PRE 主机模式下，用于配置 SPI 输出时钟的预分频数。仅在 SPI_CLK_EQU_SYSCLK 为 0 时有效。从机模式下，此寄存器值需要置为 0。（读/写）

SPI_CLKCNT_N 主机模式下，用于配置 SPI 输出时钟的分频数。仅在 SPI_CLK_EQU_SYSCLK 为 0 时有效。从机模式下，此寄存器值需要置为 0。（读/写）

SPI_CLKCNT_H 主机模式下，SPI_CLKCNT_H = $\lfloor \frac{\text{SPI\_CLKCNT\_N+1}}{2} - 1 \rfloor$。仅在 SPI_CLK_EQU_SYSCLK 为 0 时有效。从机模式下，此寄存器值需要置为 0。（读/写）

SPI_CLKCNT_L 主机模式下，此寄存器的值等于 SPI_CLKCNT_N。仅在 SPI_CLK_EQU_SYSCLK 为 0 时有效。从机模式下，此寄存器值为 0。（读/写）

Register 7.8. SPI_USER_REG (0x1C)



SPI_USR_COMMAND 在 SPI 半双工模式和 QSPI 模式，用于使能 SPI 操作的命令状态。(读/写)

SPI_USR_ADDR 在 SPI 半双工模式和 QSPI 模式，用于使能 SPI 操作的地址状态。(读/写)

SPI_USR_DUMMY 在 SPI 半双工模式和 QSPI 模式，用于使能 SPI 操作的等待状态。(读/写)

SPI_USR_MISO 在 SPI 半双工模式和 QSPI 模式，用于使能 SPI 操作的数据读取状态。(读/写)

SPI_USR_MOSI 在 SPI 半双工模式和 QSPI 模式，用于使能 SPI 操作的写数据状态。(读/写)

SPI_USR_DUMMY_IDLE 在 SPI 半双工模式和 QSPI 模式，在等待状态无时钟输出。(读/写)

SPI_USR_MOSI_HIGHPART 主机输出/从机输入数据存储于 SPI_W8_REG ~ SPI_W15_REG。(读/写)

SPI_USR_MISO_HIGHPART 主机输入/从机输出数据存储于 SPI_W8_REG ~ SPI_W15_REG。(读/写)

SPI_SIO 使能三线半双工通信。(读/写)

SPI_FWRITE_QIO 保留。

SPI_FWRITE_DIO 保留。

SPI_FWRITE_QUAD 保留。

SPI_FWRITE_DUAL 保留。

SPI_WR_BYTE_ORDER 用于配置发送信号的字节顺序，只对发送的命令，地址和数据有效。1：大端字节序；0：小端字节序。(读/写)

SPI_RD_BYTE_ORDER 用于配置接收信号的字节顺序，只对接收的数据有效。1：大端字节序；0：小端字节序。(读/写)

SPI_CK_OUT_EDGE 此位与 SPI_MOSI_DELAY_MODE 共同设置 MOSI 信号的延迟模式。只对主机模式有效。(读/写)

SPI_CK_I_EDGE 此位与主机模式下的 SPI_CK_OUT_EDGE 相同。与 SPI_MISO_DELAY_MODE 共同使用。只对从机模式有效。(读/写)

SPI_CS_SETUP 在 CS 有效沿和第一个时钟沿之间加入延迟，以 SPI 时钟周期为单位。此位置 1 时，半双工模式和 QSPI 模式下，延迟时间为 (SPI_SETUP_TIME + 1.5)；在全双工模式下，mode0 和 mode2 的延迟时间为 1.5，mode1 和 mode3 的延迟时间为 1。(读/写)

SPI_CS_HOLD 在数据传输结束和 CS 无效沿之间加入一段延迟，延迟时间为 SPI_HOLD_TIME。(读/写)

SPI_DOUTDIN 使能全双工模式。(读/写)

Register 7.9. SPI_USER1_REG (0x20)

SPI_USR_ADDR_BITLEN	(reserved)	SPI_USR_DUMMY_CYCLELEN	
31 26	25 8	7 0	
23	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	7	Reset

SPI_USR_ADDR_BITLEN 在 SPI 半双工模式和 QSPI 模式，表示 SPI 发送的地址位宽减 1，单位为 bit。仅当 SPI_USR_ADDR 被置 1 时有效。（只读）

SPI_USR_DUMMY_CYCLELEN 在 SPI 半双工模式和 QSPI 模式，表示 SPI 等待状态持续的 spi_clk 时钟周期数减 1 。仅当 SPI_USR_DUMMY 被置 1 时有效。（读/写）

Register 7.10. SPI_USER2_REG (0x24)

SPI_USR_COMMAND_BITLEN	(reserved)	SPI_USR_COMMAND_VALUE	
31 28	27 16	15 0	
7	0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_USR_COMMAND_BITLEN 在 SPI 半双工模式和 QSPI 模式，表示 SPI 发送的命令长度减 1，单位为 bit。仅当 SPI_USR_COMMAND 被置 1 时有效。（读/写）

SPI_USR_COMMAND_VALUE 在 SPI 半双工模式和 QSPI 模式，表示 SPI 要发送的命令值。仅当 SPI_USR_COMMAND 被置 1 时有效。（读/写）

Register 7.11. SPI_MOSI_DLEN_REG (0x28)

(reserved)	SPI_USR_MOSI_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_USR_MOSI_DBITLEN 用于配置传输数据长度，主机输出/从机输入的数据长度减 1，单位为 bit。仅当 SPI_USR_MOSI 被置 1 时有效。主机模式有效。（读/写）

Register 7.12. SPI_MISO_DLEN_REG (0x2C)

(reserved)	SPI_USR_MISO_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_USR_MISO_DBITLEN 用于配置传输数据长度，主机输入/从机输出读取的数据长度减 1，单位为 bit。仅当 SPI_USR_MISO 被置 1 时有效。主机模式有效。（读/写）

Register 7.13. SPI_SLV_WR_STATUS_REG (0x30)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_SLV_WR_STATUS_REG 从机模式下，表示主机写入从机的状态寄存器。主机模式下，如果地址长度大于 32 bit，此寄存器为低 32 bit。（读/写）

Register 7.14. SPI_PIN_REG (0x34)

(reserved)	SPI_CS_KEEP_ACTIVE	SPI_CK_IDLE_EDGE	(reserved)	SPI_MASTER_CK_SEL	(reserved)	SPI_MASTER_CS_POL	SPI_CK_DIS	(reserved)	SPI_CS2_DIS	SPI_CS1_DIS	SPI_CS0_DIS	
31	30	29	28 14	13 11	10 9	8 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	1	1	0	Reset

SPI_CS_KEEP_ACTIVE 此位置 1 时，CS 信号将保持有效状态。只用于主机模式。（读/写）

SPI_CK_IDLE_EDGE 用于配置 SPI 输出时钟在空闲状态的电平。只用于主机模式。（读/写）
1：空闲状态时, spi_clk 信号线保持高电平；
0：空闲状态时, spi_clk 信号线保持低电平。

SPI_MASTER_CK_SEL 保留。

SPI_MASTER_CS_POL 保留。

SPI_CK_DIS 保留。

SPI_CS2_DIS 使能 SPI CS2 信号。1：关闭 CS2；0：开启 CS2。（读/写）

SPI_CS1_DIS 使能 SPI CS1 信号。1：关闭 CS1；0：开启 CS1。（读/写）

SPI_CS0_DIS 使能 SPI CS0 信号。1：关闭 CS0；0：开启 CS0。（读/写）

Register 7.15. SPI_SLAVE_REG (0x38)

SPI_SYNC_RESET	SPI_SLAVE_MODE	SPI_SLV_WR_RD_BUF_EN	SPI_SLV_WR_RD_STA_EN	SPI_SLV_CMD_DEFINE	SPI_TRANS_CNT	SPI_SLV_LAST_STATE	SPI_SLV_LAST_COMMAND	(reserved)	SPI_CS_I_MODE	SPI_TRANS_INTEN	SPI_SLV_WR_STA_INTEN	SPI_SLV_RD_STA_INTEN	SPI_SLV_WR_BUF_INTEN	SPI_SLV_RD_BUF_INTEN	SPI_TRANS_DONE	SPI_SLV_WR_STA_DONE	SPI_SLV_RD_STA_DONE	SPI_SLV_WR_BUF_DONE	SPI_SLV_RD_BUF_DONE	
31	30	29	28	27	26 23	22 20	19 17	16 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

SPI_SYNC_RESET 该位用于复位 SPI 时钟线，CS 线和数据线的锁存值。（读/写）

SPI_SLAVE_MODE 该位用于设置 SPI 设备的模式。（读/写）

1：从机模式；

0：主机模式。

SPI_SLV_WR_RD_BUF_EN 将此位置 1，使能从机模式下的读写数据操作命令。只用于从机半双工模式。（读/写）

SPI_SLV_WR_RD_STA_EN 将此位置 1，使能从机模式下的读写状态操作命令。只用于从机半双工模式。（读/写）

SPI_SLV_CMD_DEFINE 保留。

SPI_TRANS_CNT 主机和从机模式下的操作计数器。（只读）

SPI_SLV_LAST_STATE 从机模式下，此位包含了 SPI 状态机的状态。（只读）

SPI_SLV_LAST_COMMAND 保留。

SPI_CS_I_MODE 保留。

SPI_TRANS_INTEN SPI_TRANS_DONE_INT 的中断使能位。（读/写）

SPI_SLV_WR_STA_INTEN SPI_SLV_WR_STA_INT 的中断使能位。（读/写）

SPI_SLV_RD_STA_INTEN SPI_SLV_RD_STA_INT 的中断使能位。（读/写）

SPI_SLV_WR_BUF_INTEN SPI_SLV_WR_BUF_INT 的中断使能位。（读/写）

SPI_SLV_RD_BUF_INTEN SPI_SLV_RD_BUF_INT 的中断使能位。（读/写）

SPI_TRANS_DONE SPI_TRANS_DONE_INT 的中断状态位。此位由硬件置 1，需要软件清零。（读/写）

SPI_SLV_WR_STA_DONE SPI_SLV_WR_STA_INT 的中断状态位。此标志由硬件置 1，需要软件清零。只用于从机半双工模式。（读/写）

SPI_SLV_RD_STA_DONE SPI_SLV_RD_STA_INT 的中断状态位。此标志由硬件置 1，需要软件清零。只用于从机半双工模式。（读/写）

SPI_SLV_WR_BUF_DONE SPI_SLV_WR_BUF_INT 的中断状态位。此标志由硬件置 1，需要软件清零。只用于从机半双工模式。（读/写）

SPI_SLV_RD_BUF_DONE SPI_SLV_RD_BUF_INT 的中断状态位。此标志由硬件置 1，需要软件清零。只用于从机半双工模式。（读/写）

Register 7.16. SPI_SLAVE1_REG (0x3C)

SPI_SLV_STATUS_BITLEN	SPI_SLV_STATUS_FAST_EN	SPI_SLV_STATUS_READBACK	(reserved)	SPI_SLV_RD_ADDR_BITLEN	SPI_SLV_WR_ADDR_BITLEN	SPI_SLV_WRSTA_DUMMY_EN	SPI_SLV_RDSTA_DUMMY_EN	SPI_SLV_WRBUF_DUMMY_EN	SPI_SLV_RDBUF_DUMMY_EN	
31 27	26	25	24 16	15 10	9 4	3	2	1	0	
0 0 0 0 0	0	1	0 0 0 0 0 0 0 0 0	0x00	0x00	0	0	0	0	Reset

SPI_SLV_STATUS_BITLEN 此位用于配置主机写状态寄存器的长度。仅在从机半双工模式下有效。(读/写)

SPI_SLV_STATUS_FAST_EN 保留。

SPI_SLV_STATUS_READBACK 保留。

SPI_SLV_RD_ADDR_BITLEN 表示读取从机数据的地址长度减 1，单位为 bit。仅在从机半双工模式下有效。(读/写)

SPI_SLV_WR_ADDR_BITLEN 表示写从机数据的地址长度减 1，单位为 bit。仅在从机半双工模式下有效。(读/写)

SPI_SLV_WRSTA_DUMMY_EN 使能写状态寄存器的等待状态。仅在从机半双工模式下有效。(读/写)

SPI_SLV_RDSTA_DUMMY_EN 使能读状态寄存器的等待状态。仅在从机半双工模式下有效。(读/写)

SPI_SLV_WRBUF_DUMMY_EN 使能写数据操作的等待状态。仅在从机半双工模式下有效。(读/写)

SPI_SLV_RDBUF_DUMMY_EN 使能读数据操作的等待状态。仅在从机半双工模式下有效。(读/写)

Register 7.17. SPI_SLAVE2_REG (0x40)

SPI_SLV_WRBUF_DUMMY_CYCLELEN	SPI_SLV_RDBUF_DUMMY_CYCLELEN	SPI_SLV_WRSTA_DUMMY_CYCLELEN	SPI_SLV_RDSTA_DUMMY_CYCLELEN	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0x000	0x000	0x000	Reset

SPI_SLV_WRBUF_DUMMY_CYCLELEN 写数据操作的等待状态所持续的 SPI 时钟周期数减 1。此位仅当 SPI_SLV_WRBUF_DUMMY_EN 被置 1 时有效。仅在从机半双工模式下有效。(读/写)

SPI_SLV_RDBUF_DUMMY_CYCLELEN 读数据操作的等待状态所持续的 SPI 时钟周期数减 1。此位仅当 SPI_SLV_RDBUF_DUMMY_EN 被置 1 时有效。仅在从机半双工模式下有效。(读/写)

SPI_SLV_WRSTA_DUMMY_CYCLELEN 写状态寄存器操作的等待状态所持续的 SPI 时钟周期数减 1。此位仅当 SPI_SLV_WRSTA_DUMMY_EN 被置 1 时有效。仅在从机半双工模式下有效。(读/写)

SPI_SLV_RDSTA_DUMMY_CYCLELEN 读状态寄存器操作的等待状态所持续的 SPI 时钟周期数减 1。此位仅当 SPI_SLV_RDSTA_DUMMY_EN 被置 1 时有效。仅在从机半双工模式下有效。(读/写)

Register 7.18. SPI_SLAVE3_REG (0x44)

SPI_SLV_WRSTA_CMD_VALUE	SPI_SLV_RDSTA_CMD_VALUE	SPI_SLV_WRBUF_CMD_VALUE	SPI_SLV_RDBUF_CMD_VALUE	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0	Reset

SPI_SLV_WRSTA_CMD_VALUE 保留。

SPI_SLV_RDSTA_CMD_VALUE 保留。

SPI_SLV_WRBUF_CMD_VALUE 保留。

SPI_SLV_RDBUF_CMD_VALUE 保留。

Register 7.19. SPI_SLV_WRBUF_DLEN_REG (0x48)

(reserved)	SPI_SLV_WRBUF_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_SLV_WRBUF_DBITLEN 表示写入数据长度减 1，单位为 bit。仅在从机半双工模式下有效。

Register 7.20. SPI_SLV_RDBUF_DLEN_REG (0x4C)

(reserved)	SPI_SLV_RDBUF_DBITLEN	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

SPI_SLV_RDBUF_DBITLEN 表示读取数据长度减 1，单位为 bit。仅在从机半双工模式下有效。（读/写）

Register 7.21. SPI_SLV_RD_BIT_REG (0x64)

(reserved)	SPI_SLV_RDATA_BIT	
31 24	23 0	
0 0 0 0 0 0 0 0	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

SPI_SLV_RDATA_BIT 主机读取从机数据长度减 1。仅在从机半双工模式下有效。（读/写）

Register 7.22. SPI_W*n*_REG (*n*: 0-15) (0x80+4**n*)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_W*n*_REG 数据缓存。（读/写）

Register 7.23. SPI_TX_CRC_REG (0xC0)

31 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_TX_CRC_REG 保留。

Register 7.24. SPI_EXT2_REG (0xF8)

(reserved)	SPI_ST	
31 … 3	2 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	Reset

SPI_ST SPI 状态机的当前状态。（只读）

0：空闲状态；

1：准备状态；

2：发送命令状态；

3：发送数据状态；

4：读取数据状态；

5：写数据状态；

6：等待状态；

7：完成状态。

Register 7.25. SPI_DMA_CONF_REG (0x100)

(reserved)	SPI_DMA_CONTINUE	SPI_DMA_TX_STOP	SPI_DMA_RX_STOP	(reserved)	SPI_OUT_DATA_BURST_EN	SPI_INDSCR_BURST_EN	SPI_OUTDSCR_BURST_EN	SPI_OUT_EOF_MODE	(reserved)	SPI_AHBM_RST	SPI_AHBM_FIFO_RST	SPI_OUT_RST	SPI_IN_RST	(reserved)	
31 17	16	15	14	13	12	11	10	9	8 6	5	4	3	2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	1	0 0 0	0	0	0	0	0 0	Reset

SPI_DMA_CONTINUE 使能 SPI DMA 数据连续 TX/RX 模式。(读/写)

SPI_DMA_TX_STOP 在连续 TX/RX 模式下，将此位置 1 停止发送数据。(读/写)

SPI_DMA_RX_STOP 在连续 TX/RX 模式下，将此位置 1 停止接收数据。(读/写)

SPI_OUT_DATA_BURST_EN DMA 使用 burst 模式从内存读取数据。(读/写)

SPI_INDSCR_BURST_EN DMA 使用 burst 模式读接收链表描述符。(读/写)

SPI_OUTDSCR_BURST_EN DMA 使用 burst 模式读发送链表描述符。(读/写)

SPI_OUT_EOF_MODE DMA 控制器生成发送 EOF 标志的模式。(读/写)
1：当 DMA 已从 FIFO 中读出所有数据时，产生 EOF 标志；
0：当 DMA 将所有数据写入 FIFO 时，产生 EOF 标志。

SPI_AHBM_RST 用于复位 SPI DMA AHB 主机。(读/写)

SPI_AHBM_FIFO_RST 用于复位 SPI DMA AHB 主机 FIFO 指针。(读/写)

SPI_OUT_RST 用于复位 DMA 发送状态机和发送数据 FIFO 指针。(读/写)

SPI_IN_RST 用于复位 DMA 接收状态机和接收数据 FIFO 指针。(读/写)

Register 7.26. SPI_DMA_OUT_LINK_REG (0x104)

(reserved)	SPI_OUTLINK_RESTART	SPI_OUTLINK_START	SPI_OUTLINK_STOP	(reserved)	SPI_OUTLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

SPI_OUTLINK_RESTART 将此位置 1 使用新的发送链表描述符。(读/写)

SPI_OUTLINK_START 将此位置 1 开始使用发送链表描述符。(读/写)

SPI_OUTLINK_STOP 将此位置 1 停止使用发送链表描述符。(读/写)

SPI_OUTLINK_ADDR 第一个发送链表描述符地址。(读/写)

Register 7.27. SPI_DMA_IN_LINK_REG (0x108)

(reserved)	SPI_INLINK_RESTART	SPI_INLINK_START	SPI_INLINK_STOP	(reserved)	SPI_INLINK_AUTO_RET	SPI_INLINK_ADDR	
31	30	29	28	27 21	20	19 0	
0	0	0	0	0 0 0 0 0 0 0	0	0x000000	Reset

SPI_INLINK_RESTART 将此位置 1 使用新的接收链表描述符。（读/写）

SPI_INLINK_START 将此位置 1 开始使用接收链表描述符。（读/写）

SPI_INLINK_STOP 将此位置 1 停止使用接收链表描述符。（读/写）

SPI_INLINK_AUTO_RET 此位置 1 时，当数据包无效，接收链表描述符跳到下一个描述符。（读/写）

SPI_INLINK_ADDR 第一个接收链表描述符地址。（读/写）

Register 7.28. SPI_DMA_STATUS_REG (0x10C)

(reserved)	SPI_DMA_TX_EN	SPI_DMA_RX_EN	
31 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

SPI_DMA_TX_EN SPI DMA 发送数据状态位。（只读）

SPI_DMA_RX_EN SPI DMA 接收数据状态位。（只读）

Register 7.29. SPI_DMA_INT_ENA_REG (0x110)

(reserved)	SPI_OUT_TOTAL_EOF_INT_ENA	SPI_OUT_EOF_INT_ENA	SPI_OUT_DONE_INT_ENA	SPI_IN_SUC_EOF_INT_ENA	SPI_IN_ERR_EOF_INT_ENA	SPI_IN_DONE_INT_ENA	SPI_INLINK_DSCR_ERROR_INT_ENA	SPI_OUTLINK_DSCR_ERROR_INT_ENA	SPI_INLINK_DSCR_EMPTY_INT_ENA	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_ENA SPI_OUT_TOTAL_EOF_INT 的中断使能位。（读/写）

SPI_OUT_EOF_INT_ENA SPI_OUT_EOF_INT 的中断使能位。（读/写）

SPI_OUT_DONE_INT_ENA SPI_OUT_DONE_INT 的中断使能位。（读/写）

SPI_IN_SUC_EOF_INT_ENA SPI_IN_SUC_EOF_INT 的中断使能位。（读/写）

SPI_IN_ERR_EOF_INT_ENA SPI_IN_ERR_EOF_INT 的中断使能位。（读/写）

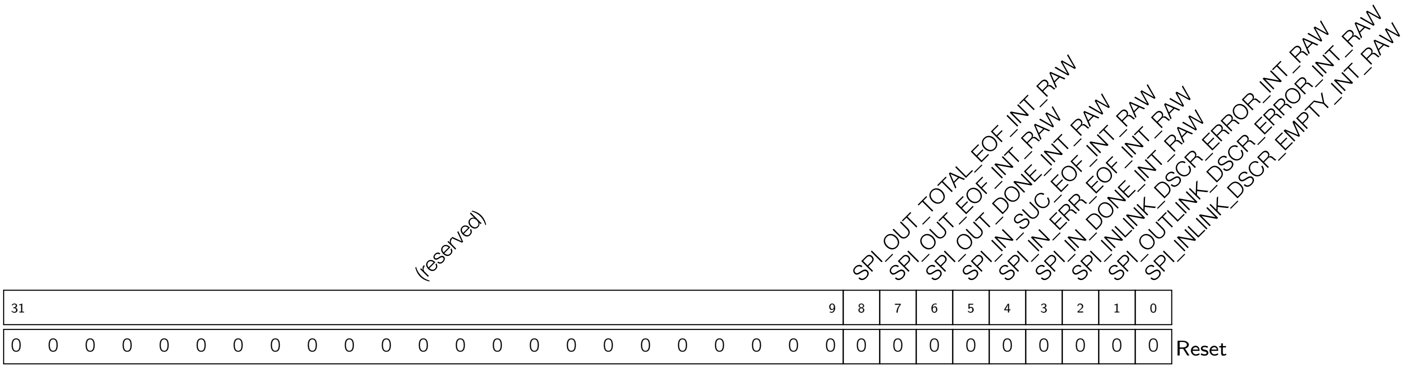
SPI_IN_DONE_INT_ENA SPI_IN_DONE_INT 的中断使能位。（读/写）

SPI_INLINK_DSCR_ERROR_INT_ENA SPI_INLINK_DSCR_ERROR_INT 的中断使能位。（读/写）

SPI_OUTLINK_DSCR_ERROR_INT_ENA SPI_OUTLINK_DSCR_ERROR_INT 的中断使能位。（读/写）

SPI_INLINK_DSCR_EMPTY_INT_ENA SPI_INLINK_DSCR_EMPTY_INT 的中断使能位。（读/写）

Register 7.30. SPI_DMA_INT_RAW_REG (0x114)



SPI_OUT_TOTAL_EOF_INT_RAW SPI_OUT_TOTAL_EOF_INT 的原始中断状态位。（只读）

SPI_OUT_EOF_INT_RAW SPI_OUT_EOF_INT 的原始中断状态位。（只读）

SPI_OUT_DONE_INT_RAW SPI_OUT_DONE_INT 的原始中断状态位。（只读）

SPI_IN_SUC_EOF_INT_RAW SPI_IN_SUC_EOF_INT 的原始中断状态位。（只读）

SPI_IN_ERR_EOF_INT_RAW SPI_IN_ERR_EOF_INT 的原始中断状态位。（只读）

SPI_IN_DONE_INT_RAW SPI_IN_DONE_INT 的原始中断状态位。（只读）

SPI_INLINK_DSCR_ERROR_INT_RAW SPI_INLINK_DSCR_ERROR_INT 的原始中断状态位。（只读）

SPI_OUTLINK_DSCR_ERROR_INT_RAW SPI_OUTLINK_DSCR_ERROR_INT 的原始中断状态位。（只读）

SPI_INLINK_DSCR_EMPTY_INT_RAW SPI_INLINK_DSCR_EMPTY_INT 的原始中断状态位。（只读）

Register 7.31. SPI_DMA_INT_ST_REG (0x118)

(reserved)	SPI_OUT_TOTAL_EOF_INT_ST	SPI_OUT_EOF_INT_ST	SPI_OUT_DONE_INT_ST	SPI_IN_SUC_EOF_INT_ST	SPI_IN_ERR_EOF_INT_ST	SPI_IN_DONE_INT_ST	SPI_INLINK_DSCR_ERROR_INT_ST	SPI_OUTLINK_DSCR_ERROR_INT_ST	SPI_INLINK_DSCR_EMPTY_INT_ST	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_ST SPI_OUT_TOTAL_EOF_INT 的屏蔽中断状态位。(只读)

SPI_OUT_EOF_INT_ST SPI_OUT_EOF_INT 的屏蔽中断状态位。(只读)

SPI_OUT_DONE_INT_ST SPI_OUT_DONE_INT 的屏蔽中断状态位。(只读)

SPI_IN_SUC_EOF_INT_ST SPI_IN_SUC_EOF_INT 的屏蔽中断状态位。(只读)

SPI_IN_ERR_EOF_INT_ST SPI_IN_ERR_EOF_INT 的屏蔽中断状态位。(只读)

SPI_IN_DONE_INT_ST SPI_IN_DONE_INT 的屏蔽中断状态位。(只读)

SPI_INLINK_DSCR_ERROR_INT_ST SPI_INLINK_DSCR_ERROR_INT 的屏蔽中断状态位。(只读)

SPI_OUTLINK_DSCR_ERROR_INT_ST SPI_OUTLINK_DSCR_ERROR_INT 的屏蔽中断状态位。(只读)

SPI_INLINK_DSCR_EMPTY_INT_ST SPI_INLINK_DSCR_EMPTY_INT 的屏蔽中断状态位。(只读)

Register 7.32. SPI_DMA_INT_CLR_REG (0x11C)

(reserved)	SPI_OUT_TOTAL_EOF_INT_CLR	SPI_OUT_EOF_INT_CLR	SPI_OUT_DONE_INT_CLR	SPI_IN_SUC_EOF_INT_CLR	SPI_IN_ERR_EOF_INT_CLR	SPI_IN_DONE_INT_CLR	SPI_INLINK_DSCR_ERROR_INT_CLR	SPI_OUTLINK_DSCR_ERROR_INT_CLR	SPI_INLINK_DSCR_EMPTY_INT_CLR	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_TOTAL_EOF_INT_CLR 将此位置 1 清除 SPI_OUT_TOTAL_EOF_INT 中断。(读/写)

SPI_OUT_EOF_INT_CLR 将此位置 1 清除 SPI_OUT_EOF_INT 中断。(读/写)

SPI_OUT_DONE_INT_CLR 将此位置 1 清除 SPI_OUT_DONE_INT 中断。(读/写)

SPI_IN_SUC_EOF_INT_CLR 将此位置 1 清除 SPI_IN_SUC_EOF_INT 中断。(读/写)

SPI_IN_ERR_EOF_INT_CLR 将此位置 1 清除 SPI_IN_ERR_EOF_INT 中断。(读/写)

SPI_IN_DONE_INT_CLR 将此位置 1 清除 SPI_IN_DONE_INT 中断。(读/写)

SPI_INLINK_DSCR_ERROR_INT_CLR 将此位置 1 清除 SPI_INLINK_DSCR_ERROR_INT 中断。(读/写)

SPI_OUTLINK_DSCR_ERROR_INT_CLR 将此位置 1 清除 SPI_OUTLINK_DSCR_ERROR_INT 中断。(读/写)

SPI_INLINK_DSCR_EMPTY_INT_CLR 将此位置 1 清除 SPI_INLINK_DSCR_EMPTY_INT 中断。(读/写)

Register 7.33. SPI_IN_ERR_EOF_DES_ADDR_REG (0x120)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_IN_ERR_EOF_DES_ADDR_REG 当 SPI DMA 出现接收错误时的接收链表描述符地址。(只读)

Register 7.34. SPI_IN_SUC_EOF_DES_ADDR_REG (0x124)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_IN_SUC_EOF_DES_ADDR_REG 当 SPI DMA 发送成功时的最后一个接收链表描述符地址。(只读)

Register 7.35. SPI_INLINK_DSCR_REG (0x128)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_REG 当前接收链表描述符地址。(只读)

Register 7.36. SPI_INLINK_DSCR_BF0_REG (0x12C)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_BF0_REG 下一个接收链表描述符地址。(只读)

Register 7.37. SPI_INLINK_DSCR_BF1_REG (0x130)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_INLINK_DSCR_BF1_REG 下一个接收链表描述符数据缓存地址。(只读)

Register 7.38. SPI_OUT_EOF_BFR_DES_ADDR_REG (0x134)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_EOF_BFR_DES_ADDR_REG 生成 EOF 的发送链表描述符对应的缓存地址。(只读)

Register 7.39. SPI_OUT_EOF_DES_ADDR_REG (0x138)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUT_EOF_DES_ADDR_REG 当 SPI DMA 发送成功时的最后一个发送链表描述符地址。(只读)

Register 7.40. SPI_OUTLINK_DSCR_REG (0x13C)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

SPI_OUTLINK_DSCR_REG 当前发送链表描述符地址。(只读)

Register 7.41. SPI_OUTLINK_DSCR_BF0_REG (0x140)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUTLINK_DSCR_BF0_REG 下一个发送链表描述符地址。（只读）

Register 7.42. SPI_OUTLINK_DSCR_BF1_REG (0x144)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SPI_OUTLINK_DSCR_BF1_REG 下一个发送链表描述符数据缓存地址。（只读）

Register 7.43. SPI_DMA_RSTATUS_REG (0x148)

TX_FIFO_EMPTY	TX_FIFO_FULL	(reserved)	TX_DES_ADDRESS	
31	30	29 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	Reset

TX_FIFO_EMPTY SPI DMA 发送 FIFO 为空。（只读）

TX_FIFO_FULL SPI DMA 发送 FIFO 为满。（只读）

TX_DES_ADDRESS SPI DMA 发送描述符指针的低有效位。（只读）

Register 7.44. SPI_DMA_TSTATUS_REG (0x14C)

RX_FIFO_EMPTY	RX_FIFO_FULL	(reserved)	RX_DES_ADDRESS	
31	30	29 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	Reset

RX_FIFO_EMPTY SPI DMA 接收 FIFO 为空。（只读）

RX_FIFO_FULL SPI DMA 接收 FIFO 为满。（只读）

RX_DES_ADDRESS SPI DMA 接收描述符指针的低有效位。（只读）

8 SDIO 从机控制器

8.1 概述

ESP32 支持数字输入输出 (SDIO) 设备接口，符合 SDIO 全速卡 V2.0 规范。主控器可以通过 SDIO 总线协议访问 ESP32。

主机 (Host) 可以直接访问 SDIO 接口寄存器，或通过使用 DMA 引擎访问设备上的共享存储器，在保证性能的同时减少了处理性能的浪费。

8.2 主要特性

- 符合 SDIO 全速卡 V2.0 规范
- 支持 SDIO SPI，1-bit 和 4-bit 传输模式
- 0 ~ 50 MHz 时钟范围
- 可配置的采样时钟沿或驱动时钟沿
- 为信息交互设定的特定寄存器
- 支持自动填充 SDIO 总线上的发送数据，同样支持自动丢弃 SDIO 总线上的填充数据
- 高达 512 字节的块大小
- Host 与 Slave 间有中断向量可以相互中断对方
- 用于数据传输的 DMA

8.3 功能描述

8.3.1 SDIO Slave 功能块图

SDIO Slave 的功能块图如图 8-1 所示。

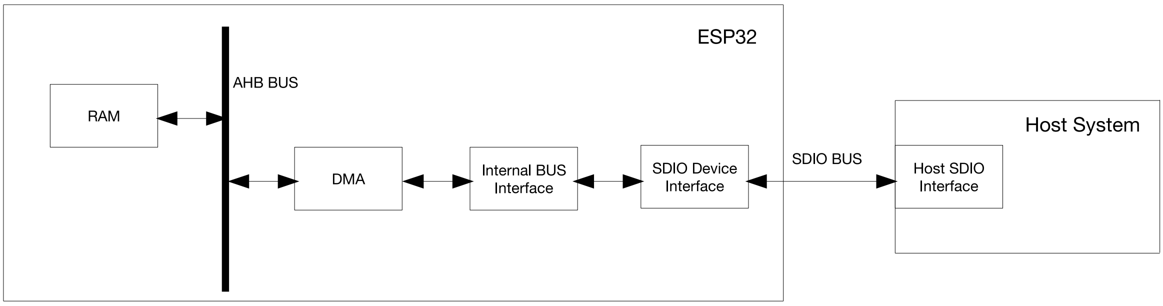


图 8-1. SDIO Slave 功能块图

主机系统 (Host System) 代表任一符合 SDIO V2.0 规范的 Host 设备。Host 通过标准 SDIO 总线与作为 SDIO Slave 的 ESP32 互动。SDIO 设备接口模块通过直接提供 SDIO 接口寄存器并使能 DMA 操作来与 Host 通信，实现高级性能总线 (AHB) 上的高速数据传输，并且不需要 CPU 的参与。

8.3.2 SDIO 总线上的数据发送和接收

Host 与 Slave 间通过 SDIO bus I/O Function1 进行数据传输。当 Host 按照 SDIO 协议使能 Slave 的 I/O Function1 后，即可以进行数据的传输。

数据的传输以包为单位，每次的传输都是指的一个数据包。为提高传输效率，推荐使用单次块传输（具体见 SDIO 协议）进行包的传输。为了实现单次块传输，Host 和 Slave 都需要将 SDIO 总线上发送的数据填充为整个块。Slave 在发包时会自动填充数据，在收包时自动丢弃填充数据。

Slave 自动填充和丢弃数据是通过 SDIO 总线上的数据地址来判断，当数据地址大于等于 0x1F800 后进行数据填充或丢，所以传输的起始数据地址为 0x1F800 –Packet_length（Packet_length 的单位是字节）。在 SDIO 总线上的数据流如图 8-2 所示：

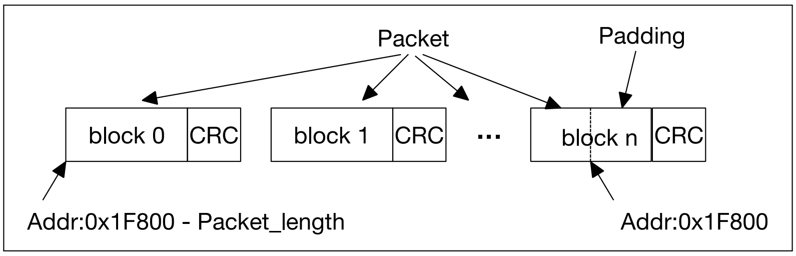


图 8-2. SDIO 总线上数据传输

传输使用的是 IO_RW_EXTENDED (CMD53) 命令，其组成如图 8-3，各部分具体含义请查看 SDIO 协议。

S	D	Command Index 110101b	R/W flag	Function Number 001b	Block Mode 1b	OP Code 1b	Register Address 0X1F800-Packet_length	Byte/Block Count Roundup (Packet_length/Block_size)	CRC7	E
1	1	6	1	3	1	1	17	9	7	1

图 8-3. CMD53 内容

8.3.3 寄存器访问

为了方便 Host 与 Slave 之间的信息交互，Host 可以通过 SDIO bus I/O Function1 访问 Slave 中的特定寄存器。这些寄存器在 SLC0HOST_TOKEN_RDATA 到 SSLC0HOST_INT_ST_REG 的连续地址段内。Host 访问时只需将 CMD52 或 CMD53 中的寄存器地址设置为对应寄存器地址的低 10 位。Host 可以使用 CMD53 同时访问多个寄存器，提高了数据传输的速率。

SLCHOST_CONF_W0_REG 到 SLCHOST_CONF_W15_REG 共有 52 个字节的区域，Host 和 Slave 可以任意访问和修改，方便了 Host 与 Slave 之间的信息交互。

8.3.4 DMA

SDIO Slave 有一个专门的 DMA 用于从 RAM 获取或存储传输数据。如图 8-1 所示，DMA 通过 AHB 访问 RAM。DMA 使用链表结构来访问 RAM。每个链表由 3 个字 (word) 组成，具体结构如图 8-4 所示。

1	1	6	12	12
Owner	Eof	Reserved	Length	Size
Buffer Address Pointer				
Next Descriptor Address				

图 8-4. SDIO Slave DMA 链表结构

- Owner：当前链表对应 buffer 允许的操作者。

  0：允许的操作者为 CPU

  1：允许的操作者为 DMA

- Eof：结束标志，表明当前链表是数据包的最后一个链表。
- Length：Buffer 中的有效字节数，即从 buffer 中能够被读取的字节数。
- Size：Buffer 允许使用的最大字节数。
- Buffer Address Pointer：Buffer 地址指针。
- Next Descriptor Address：下一个链表的地址指针。当当前链表已经是最后一个链表时，Eof 位的值应为 1，该值应为 0。

Slave 链表串如图 8-5 所示：

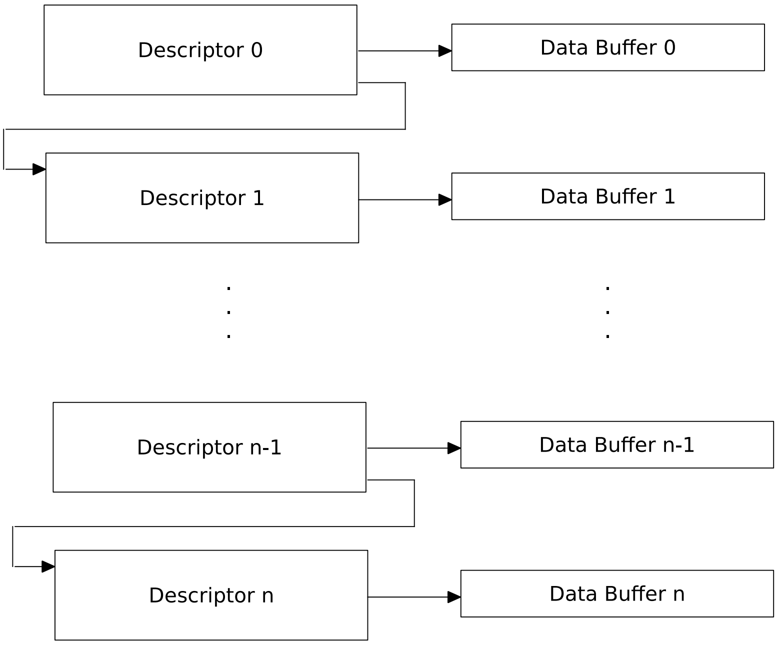


图 8-5. 链表串

8.3.5 包的发送和接收流程

Host 与 Slave 间的包传输需要两者按照特定的流程配合完成。

8.3.5.1 Slave 向 Host 发送包

包的发送是由 Slave 发起，通过中断来通知 Host（中断实现方式参看 SDIO 协议）。Host 从 Slave 读取相关信息后发起 SDIO 总线传输。整个流程如图 8-6 所示：

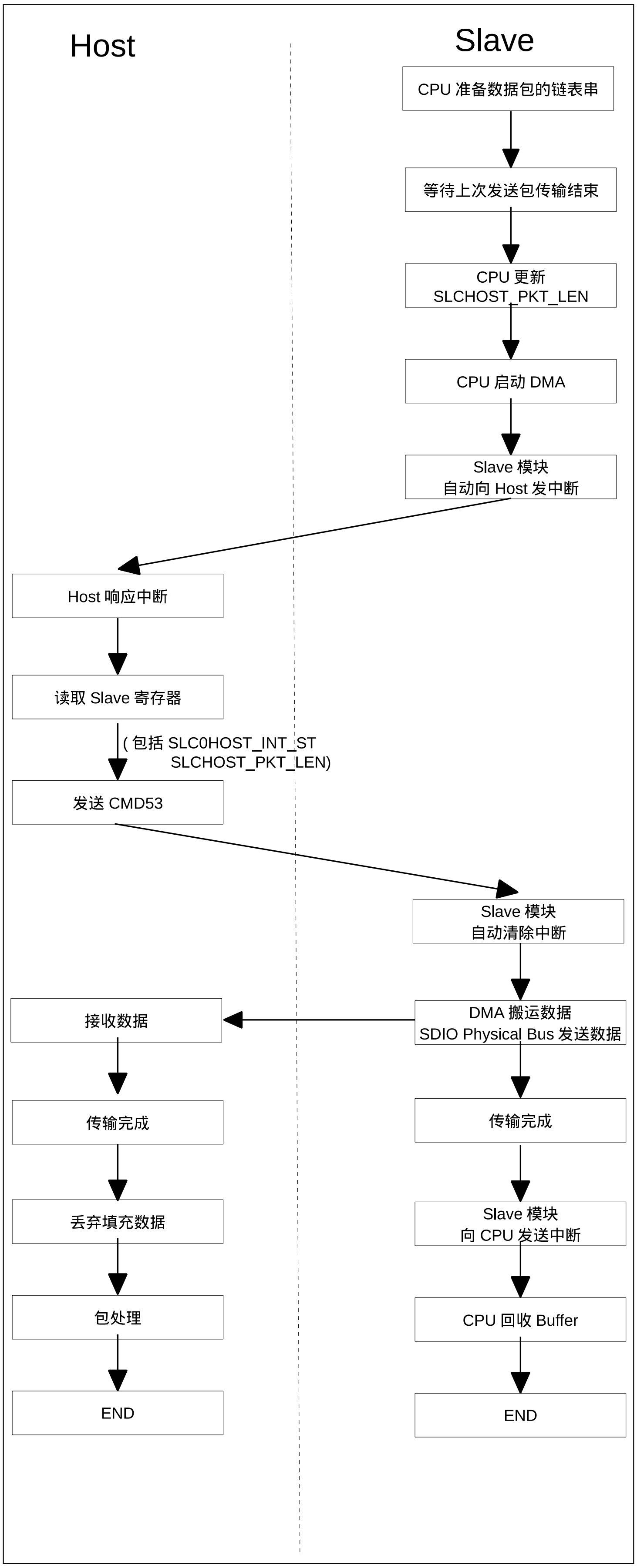


图 8-6. Slave 向 Host 发送包的流程

Host 从 Slave 读取相关信息是通过访问 SLC0HOST_INT 和 SLCHOST_PKT_LEN 这两个寄存器实现的。

- SLC0HOST_INT：中断状态寄存器，其中 bit SLC0_RX_NEW_PACKET_INT_ST 为 1 表明中断的原因是 Slave 发包。
- SLCHOST_PKT_LEN：Slave 发送包长度累加寄存器，Host 用本次读取值减去上次读取值就可以得到本次发送包的长度。

CPU 启动 DMA 需要先将链表串第一个链表的地址低 20 位写到寄存器 SLC0RX_LINK 的 SLC0_RXLINK_ADDR 中，再配置 SLC0RX_LINK 的 SLC0_RXLINK_START 来启动 DMA。之后 DMA 会自动完成数据的传输。

发送完成后 DMA 会向 CPU 发送中断，这时 CPU 可以回收 buffer。

8.3.5.2 Slave 从 Host 接收包

包的接收是由 Host 发起，Slave 通过 DMA 接收数据并存储到 RAM 中，传输完成后通过中断通知 CPU 进行数据处理。整个流程如图 8-7 所示：

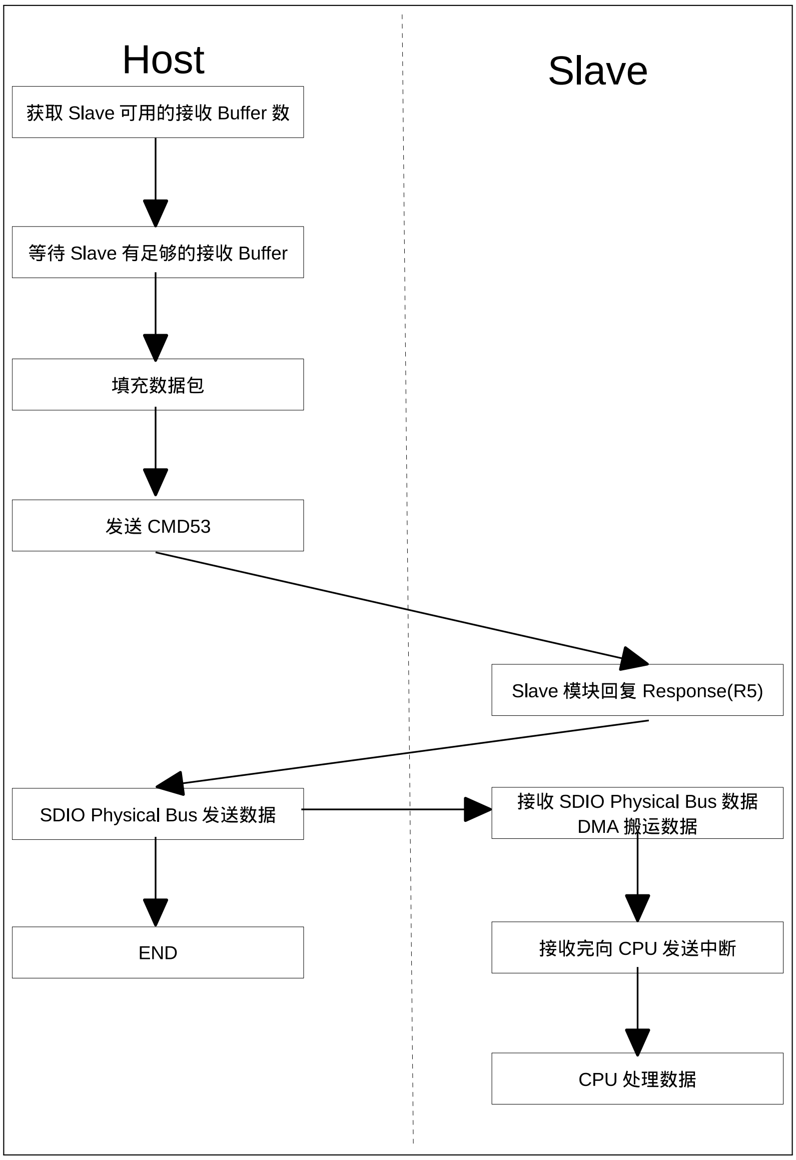


图 8-7. Slave 从 Host 接收包的流程

Host 通过访问寄存器 SLC0HOST_TOKEN_RDATA 获取 Slave 侧可用于接收包的 buffer 数。Slave 侧的 CPU 要在准备好用于接收包的 DMA 链表后更新这个值。

SLC0HOST_TOKEN_RDATA 中 HOSTREG_SLC0_TOKEN1 存放可用 buffer 的累计值。

Host 通过 HOSTREG_SLC0_TOKEN1 减去自己已用掉的 buffer 数就得到了当前可用的 buffer 数。

当 buffer 不够用时，Host 需要通过不停地访问这个寄存器直到 buffer 够用为止。

为了保证有充足的 buffer 来接收包，Slave 的 CPU 必须不停地在接收链表上挂载 buffer。挂载的流程如图 8-8 所示。

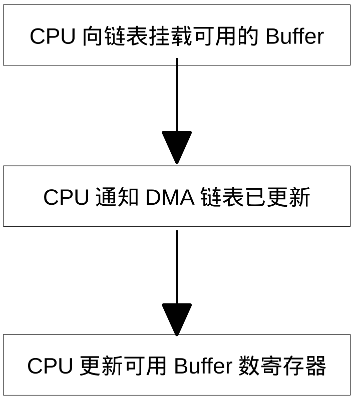


图 8-8. Slave CPU 挂载 buffer 的流程

CPU 首先需要将新的 buffer 组成链表串并且挂载在 DMA 正在使用的链表串结尾。

然后 CPU 需要通知 DMA 链表串已更新，可以通过置位 SLC0TX_LINK 寄存器的 SLC0_TXLINK_RESTART 为 1 来实现。注意，CPU 在首次启动 DMA 用于收包时需置位置 SLC0TX_LINK 寄存器的 SLC0_TXLINK_START。

最后，CPU 通过配置 SLC0TOKEN1 寄存器更新可用的 buffer。

8.3.6 SDIO 总线时序

Host 与 Slave 间是通过 PCB 走线连接，所以延迟大。为了保证总线上时序正确，Slave 支持调整 SDIO 总线输入的采样时钟沿和输出的驱动时钟沿。

当从 Host 来的数据变化的时刻靠近时钟的上升沿时，Slave 会选择时钟的下降沿进行采样。采样时序图如图 8-9 所示：

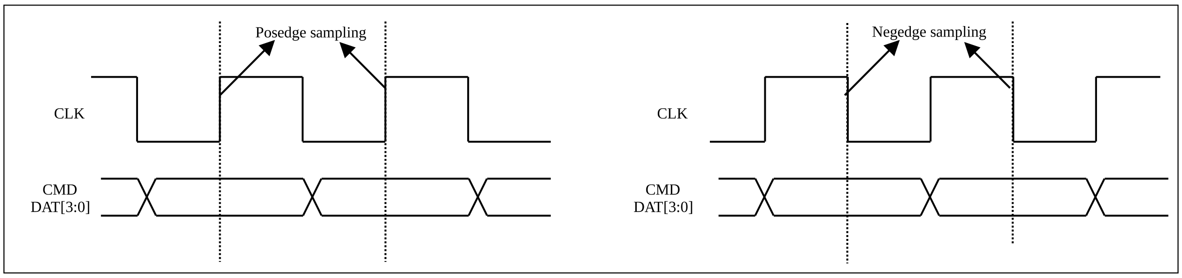


图 8-9. 采样时序图

Slave 管脚采样沿默认由 MTDO strapping 值决定，但通过配置 SLCHOST_CONF_REG 寄存器可以强制决定模

式（优先级从高到低）：(1) 置位 SLCHOST_FRC_POS_SAMP，对应管脚强制上升沿采样；(2) 置位 SLCHOST_FRC_NEG_SAMP，对应管脚强制下降沿采样。

SLCHOST_FRC_POS_SAMP 和 SLCHOST_FRC_NEG_SAMP 的位宽均为 5 bit，5 个 bit 分别对应 CMD 线和 4 个 DATA 线 (0-3)。置位这两个域使得 Slave 对相应的线在时钟上升沿或下降沿上进行采样。

Slave 还可以选择时钟的上升沿还是下降沿对数据输出线进行驱动，以适应不同的延迟。时序图如图 8-10 所示：

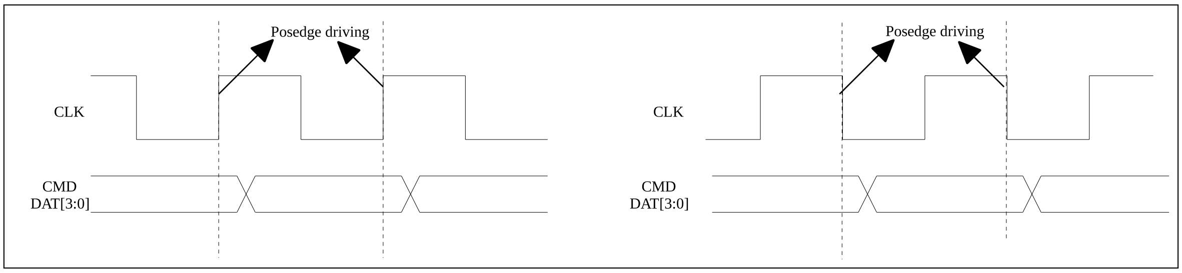


图 8-10. 输出时序图

Slave 管脚输出沿默认由 GPIO5 strapping 值决定，但通过配置以下寄存器可以强制决定模式（优先级从高到低）：(1) 置位 SLCHOST_CONF_REG 中的 SLCHOST_FRC_SDIO11，对应管脚强制下降沿输出；(2) 置位 SLCHOST_CONF_REG 中的 SLCHOST_FRC_SDIO22，对应管脚强制上升沿输出；(3) 置位 HINF_CFG_DATA1_REG 中的 HINF_HIGHSPEED_ENABLE 和 SLCHOST_CONF_REG 中的 SLCHOST_HSPEED_CON_EN 后，主机通过配置 CCCR 中的 EHS (Enable High-Speed) 为 1 强制上升沿输出。

SLCHOST_FRC_SDIO11 和 SLCHOST_FRC_SDIO22 的位宽均为 5 bit，5 个 bit 分别对应 CMD 线和 4 个 DATA 线 (0-3)。置位这两个域使得 Slave 在时钟下降沿或上升沿上对相应的线进行驱动。

关于优先级的说明：无论输入还是输出，strapping 管脚的配置也被包含在优先级范围内（优先级最低）。每一级有效的前提是比它高的优先级都不生效。例如 MTDO strapping 的配置决定对应管脚的采样沿，生效的前提是 SCLHOST_FRC_POS_SAMP 没有置 1，SCLHOST_FRC_NEG_SAMP 也没有置 1。

8.3.7 中断

Host 和 Slave 间可以通过配置中断向量灵活地中断对方。Host 和 Slave 各有 8 个中断向量可用于中断对方。在配置中断向量寄存器后就会向对方发送中断（配置相应的中断使能寄存器）。中断向量寄存器具有自清功能，所以配置一次会产生一次中断，不需要其他操作。

8.3.7.1 Host 侧中断

- *SLC0HOST_SLC0_RX_NEW_PACKET_INT* Slave 发包中断
- *SLC0HOST_SLC0_TX_OVF_INT* Slave 接收 buffer 溢出中断
- *SLC0HOST_SLC0_RX_UDF_INT* Slave 发送 buffer 下溢中断
- *SLC0HOST_SLC0_TOHOST_BITn_INT* (*n*: 0 ~ 7) Slave 中断 Host

8.3.7.2 Slave 侧中断

- *SLC0INT_SLC0_RX_DSCR_ERR_INT* Slave 发送描述符错误中断

- *SLC0INT_SLC0_TX_DSCR_ERR_INT* Slave 接收描述符错误中断
- *SLC0INT_SLC0_RX_EOF_INT* Slave 发送操作完成中断
- *SLC0INT_SLC0_RX_DONE_INT* 单个 buffer 由 Slave 发送完成的中断
- *SLC0INT_SLC0_TX_SUC_EOF_INT* Slave 接收操作完成中断
- *SLC0INT_SLC0_TX_DONE_INT* A 单个 buffer 在 Slave 接收操作时填满了的中断
- *SLC0INT_SLC0_TX_OVF_INT* Slave 接收 buffer 溢出中断
- *SLC0INT_SLC0_RX_UDF_INT* Slave 发送 buffer 下溢中断
- *SLC0INT_SLC0_TX_START_INT* Slave 接收操作开始中断
- *SLC0INT_SLC0_RX_START_INT* Slave 发送操作开始中断
- *SLC0INT_SLC_FRHOST_BITn_INT* (*n*: 0 ~ 7) Host 中断 Slave

8.4 寄存器列表

名称	描述	地址	访问
SDIO DMA (SLC) 配置寄存器			
SLCCONF0_REG	SLCCONF0_SLC 配置	0x3FF58000	读/写
SLC0RX_LINK_REG	发送链表配置	0x3FF5803C	读/写
SLC0TX_LINK_REG	接收链表配置	0x3FF58040	读/写
SLCINTVEC_TOHOST_REG	Slave 中断 Host 的中断向量	0x3FF5804C	只写
SLC0TOKEN1_REG	接收 buffer 数	0x3FF58054	只写
SLCCONF1_REG	控制寄存器	0x3FF58060	读/写
SLC_RX_DSCR_CONF_REG	DMA 传输配置	0x3FF58098	读/写
SLC0_LEN_CONF_REG	传输数据包的长度控制	0x3FF580E4	读/写
SLC0_LENGTH_REG	传输数据包的长度	0x3FF580E8	读/写
中断寄存器			
SLC0INT_RAW_REG	原始中断状态	0x3FF58004	只读
SLC0INT_ST_REG	中断状态	0x3FF58008	只读
SLC0INT_ENA_REG	中断使能	0x3FF5800C	读/写
SLC0INT_CLR_REG	中断清除	0x3FF58010	只写

名称	描述	地址	访问
SDIO SLC Host 寄存器			
SLC0HOST_TOKEN_RDATA	Slave 接收 buffer 的累计值	0x3FF55044	只读
SLCHOST_PKT_LEN_REG	传输数据包的长度	0x3FF55060	只读
SLCHOST_CONF_W0_REG	Host 和 Slave 通信寄存器 0	0x3FF5506C	读/写
SLCHOST_CONF_W1_REG	Host 和 Slave 通信寄存器 1	0x3FF55070	读/写
SLCHOST_CONF_W2_REG	Host 和 Slave 通信寄存器 2	0x3FF55074	读/写
SLCHOST_CONF_W3_REG	Host 和 Slave 通信寄存器 3	0x3FF55078	读/写
SLCHOST_CONF_W4_REG	Host 和 Slave 通信寄存器 4	0x3FF5507C	读/写
SLCHOST_CONF_W6_REG	Host 和 Slave 通信寄存器 6	0x3FF55088	读/写
SLCHOST_CONF_W8_REG	Host 和 Slave 通信寄存器 8	0x3FF5509C	读/写

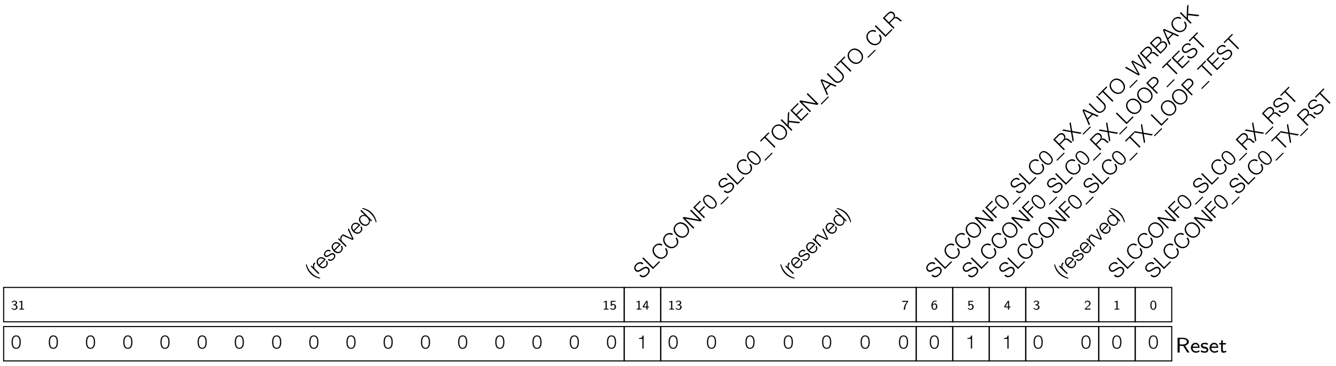
SLCHOST_CONF_W9_REG	Host 和 Slave 通信寄存器 9	0x3FF550A0	读/写
SLCHOST_CONF_W10_REG	Host 和 Slave 通信寄存器 10	0x3FF550A4	读/写
SLCHOST_CONF_W11_REG	Host 和 Slave 通信寄存器 11	0x3FF550A8	读/写
SLCHOST_CONF_W12_REG	Host 和 Slave 通信寄存器 12	0x3FF550AC	读/写
SLCHOST_CONF_W13_REG	Host 和 Slave 通信寄存器 13	0x3FF550B0	读/写
SLCHOST_CONF_W14_REG	Host 和 Slave 通信寄存器 14	0x3FF550B4	读/写
SLCHOST_CONF_W15_REG	Host 和 Slave 通信寄存器 15	0x3FF550B8	读/写
SLCHOST_CONF_REG	沿配置	0x3FF551F0	读/写
中断寄存器			
SLC0HOST_INT_RAW_REG	原始中断	0x3FF55000	只读
SLC0HOST_INT_ST_REG	屏蔽中断状态	0x3FF55058	只读
SLC0HOST_INT_CLR_REG	中断清除	0x3FF550D4	只写
SLC0HOST_FUNC1_INT_ENA_REG	中断使能	0x3FF550DC	读/写
SLCHOST_CONF_W7_REG	Host 中断 Slave 的中断向量	0x3FF5508C	只写

名称	描述	地址	访问
SDIO HINF 寄存器			
HINF_CFG_DATA1_REG	SDIO 规范配置	0x3FF4B004	读/写

8.5 SLC 寄存器

本小节括号中的地址均为相对于 SDIO 从机基地址 (0x3FF5_8000) 的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 8.4 寄存器列表。

Register 8.1. SLCCONF0_REG (0x0)



SLCCONF0_SLC0_TOKEN_AUTO_CLR 初始化为 0。请勿修改。(读/写)

SLCCONF0_SLC0_RX_AUTO_WRBACK 支持在发送数据时在发送 buffer 链表上改写 owner bit。(读/写)

SLCCONF0_SLC0_RX_LOOP_TEST Slave buffer 发送包结束后循环。置位后,硬件不会主动更改链表中 owner bit。(读/写)

SLCCONF0_SLC0_TX_LOOP_TEST Slave buffer 接收包结束后循环。置位后,硬件不会主动更改链表中 owner bit。(读/写)

SLCCONF0_SLC0_RX_RST 置为 1 复位发送 FSM。(读/写)

SLCCONF0_SLC0_TX_RST 置为 1 复位接收 FSM。(读/写)

Register 8.2. SLC0INT_RAW_REG (0x4)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_RAW	SLC0INT_SLC0_TX_DSCR_ERR_INT_RAW	(reserved)	SLC0INT_SLC0_RX_EOF_INT_RAW	SLC0INT_SLC0_RX_DONE_INT_RAW	SLC0INT_SLC0_TX_SUC_EOF_INT_RAW	SLC0INT_SLC0_TX_DONE_INT_RAW	(reserved)	SLC0INT_SLC0_TX_OVF_INT_RAW	SLC0INT_SLC0_RX_UDF_INT_RAW	SLC0INT_SLC0_TX_START_INT_RAW	SLC0INT_SLC0_RX_START_INT_RAW	SLC0INT_SLC_FRHOST_BIT7_INT_RAW	SLC0INT_SLC_FRHOST_BIT6_INT_RAW	SLC0INT_SLC_FRHOST_BIT5_INT_RAW	SLC0INT_SLC_FRHOST_BIT4_INT_RAW	SLC0INT_SLC_FRHOST_BIT3_INT_RAW	SLC0INT_SLC_FRHOST_BIT2_INT_RAW	SLC0INT_SLC_FRHOST_BIT1_INT_RAW	SLC0INT_SLC_FRHOST_BIT0_INT_RAW	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 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

SLC0INT_SLC0_RX_DSCR_ERR_INT_RAW Slave 发送描述符错误的原始中断位。（只读）

SLC0INT_SLC0_TX_DSCR_ERR_INT_RAW Slave 接收描述符错误的原始中断位。（只读）

SLC0INT_SLC0_RX_EOF_INT_RAW Slave 发送操作结束的中断标志位。（只读）

SLC0INT_SLC0_RX_DONE_INT_RAW 单个 buffer 由 Slave 发送完成的原始中断位。（只读）

SLC0INT_SLC0_TX_SUC_EOF_INT_RAW Slave 接收操作完成的原始中断位。（只读）

SLC0INT_SLC0_TX_DONE_INT_RAW 单个 buffer 在 Slave 接收操作时填满了的原始中断位。（只读）

SLC0INT_SLC0_TX_OVF_INT_RAW Slave 接收 buffer 溢出的原始中断位。（只读）

SLC0INT_SLC0_RX_UDF_INT_RAW Slave 发送 buffer 下溢的原始中断位。（只读）

SLC0INT_SLC0_TX_START_INT_RAW Slave 接收开始中断的的原始中断位。（只读）

SLC0INT_SLC0_RX_START_INT_RAW Slave 发送开始中断的的原始中断位。（只读）

SLC0INT_SLC_FRHOST_BIT7_INT_RAW Host 中断 Slave 的中断标志位 7。（只读）

SLC0INT_SLC_FRHOST_BIT6_INT_RAW Host 中断 Slave 的中断标志位 6。（只读）

SLC0INT_SLC_FRHOST_BIT5_INT_RAW Host 中断 Slave 的中断标志位 5。（只读）

SLC0INT_SLC_FRHOST_BIT4_INT_RAW Host 中断 Slave 的中断标志位 4。（只读）

SLC0INT_SLC_FRHOST_BIT3_INT_RAW Host 中断 Slave 的中断标志位 3。（只读）

SLC0INT_SLC_FRHOST_BIT2_INT_RAW Host 中断 Slave 的中断标志位 2。（只读）

SLC0INT_SLC_FRHOST_BIT1_INT_RAW Host 中断 Slave 的中断标志位 1。（只读）

SLC0INT_SLC_FRHOST_BIT0_INT_RAW Host 中断 Slave 的中断标志位 0。（只读）

Register 8.3. SLC0INT_ST_REG (0x8)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_ST	SLC0INT_SLC0_TX_DSCR_ERR_INT_ST	(reserved)	SLC0INT_SLC0_RX_EOF_INT_ST	SLC0INT_SLC0_RX_DONE_INT_ST	SLC0INT_SLC0_TX_SUC_EOF_INT_ST	SLC0INT_SLC0_TX_DONE_INT_ST	(reserved)	SLC0INT_SLC0_TX_OVF_INT_ST	SLC0INT_SLC0_RX_UDF_INT_ST	SLC0INT_SLC0_TX_START_INT_ST	SLC0INT_SLC0_RX_START_INT_ST	SLC0INT_SLC_FRHOST_BIT7_INT_ST	SLC0INT_SLC_FRHOST_BIT6_INT_ST	SLC0INT_SLC_FRHOST_BIT5_INT_ST	SLC0INT_SLC_FRHOST_BIT4_INT_ST	SLC0INT_SLC_FRHOST_BIT3_INT_ST	SLC0INT_SLC_FRHOST_BIT2_INT_ST	SLC0INT_SLC_FRHOST_BIT1_INT_ST	SLC0INT_SLC_FRHOST_BIT0_INT_ST	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 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

SLC0INT_SLC0_RX_DSCR_ERR_INT_ST Slave 发送描述符错误的中断状态位。(只读)

SLC0INT_SLC0_TX_DSCR_ERR_INT_ST Slave 接收描述符错误的中断状态位。(只读)

SLC0INT_SLC0_RX_EOF_INT_ST Slave 发送操作结束的中断状态位。(只读)

SLC0INT_SLC0_RX_DONE_INT_ST 单个 buffer 由 Slave 发送完成的中断状态位。(只读)

SLC0INT_SLC0_TX_SUC_EOF_INT_ST Slave 接收操作完成的中断状态位。(只读)

SLC0INT_SLC0_TX_DONE_INT_ST 单个 buffer 在 Slave 接收操作时填满了的中断状态位。(只读)

SLC0INT_SLC0_TX_OVF_INT_ST Slave 接收溢出中断的中断状态位。(只读)

SLC0INT_SLC0_RX_UDF_INT_ST Slave 发送 buffer 下溢的中断状态位。(只读)

SLC0INT_SLC0_TX_START_INT_ST Slave 接收中断开始的中断状态位。(只读)

SLC0INT_SLC0_RX_START_INT_ST Slave 发送中断开始的中断状态位。(只读)

SLC0INT_SLC_FRHOST_BIT7_INT_ST Host 中断 Slave 的中断状态位 7。(只读)

SLC0INT_SLC_FRHOST_BIT6_INT_ST Host 中断 Slave 的中断状态位 6。(只读)

SLC0INT_SLC_FRHOST_BIT5_INT_ST Host 中断 Slave 的中断状态位 5。(只读)

SLC0INT_SLC_FRHOST_BIT4_INT_ST Host 中断 Slave 的中断状态位 4。(只读)

SLC0INT_SLC_FRHOST_BIT3_INT_ST Host 中断 Slave 的中断状态位 3。(只读)

SLC0INT_SLC_FRHOST_BIT2_INT_ST Host 中断 Slave 的中断状态位 2。(只读)

SLC0INT_SLC_FRHOST_BIT1_INT_ST Host 中断 Slave 的中断状态位 1。(只读)

SLC0INT_SLC_FRHOST_BIT0_INT_ST Host 中断 Slave 的中断状态位 0。(只读)

Register 8.4. SLC0INT_ENA_REG (0xC)

(reserved)	(reserved)	SLC0INT_SLC0_RX_DSCR_ERR_INT_ENA	SLC0INT_SLC0_TX_DSCR_ERR_INT_ENA	(reserved)	SLC0INT_SLC0_RX_EOF_INT_ENA	SLC0INT_SLC0_RX_DONE_INT_ENA	SLC0INT_SLC0_TX_SUC_EOF_INT_ENA	SLC0INT_SLC0_TX_DONE_INT_ENA	(reserved)	SLC0INT_SLC0_TX_OVF_INT_ENA	SLC0INT_SLC0_RX_UDF_INT_ENA	SLC0INT_SLC0_TX_START_INT_ENA	SLC0INT_SLC0_RX_START_INT_ENA	SLC0INT_SLC_FRHOST_BIT7_INT_ENA	SLC0INT_SLC_FRHOST_BIT6_INT_ENA	SLC0INT_SLC_FRHOST_BIT5_INT_ENA	SLC0INT_SLC_FRHOST_BIT4_INT_ENA	SLC0INT_SLC_FRHOST_BIT3_INT_ENA	SLC0INT_SLC_FRHOST_BIT2_INT_ENA	SLC0INT_SLC_FRHOST_BIT1_INT_ENA	SLC0INT_SLC_FRHOST_BIT0_INT_ENA	
31 27	26 21	20	19	18	17	16	15	14	13 12	11	10	9	8	7	6	5	4	3	2	1	0	
0x00	0 0 0 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

SLC0INT_SLC0_RX_DSCR_ERR_INT_ENA Slave 发送链表描述符错误的中断使能位。（读/写）

SLC0INT_SLC0_TX_DSCR_ERR_INT_ENA Slave 接收链表描述符错误的中断使能位。（读/写）

SLC0INT_SLC0_RX_EOF_INT_ENA Slave 发送操作结束的中断使能位。（读/写）

SLC0INT_SLC0_RX_DONE_INT_ENA Slave 发送模式下单个 buffer 发送完成的中断使能位。（读/写）

SLC0INT_SLC0_TX_SUC_EOF_INT_ENA Slave 接收操作完成的中断使能位。（读/写）

SLC0INT_SLC0_TX_DONE_INT_ENA Slave 接收模式下单个 buffer 填满了的中断使能位。（只读）

SLC0INT_SLC0_TX_OVF_INT_ENA Slave 接收 buffer 溢出的中断状态位。（读/写）

SLC0INT_SLC0_RX_UDF_INT_ENA Slave 发送 buffer 下溢的中断状态位。（读/写）

SLC0INT_SLC0_TX_START_INT_ENA Slave 接收操作开始的中断使能位。（读/写）

SLC0INT_SLC0_RX_START_INT_ENA Slave 发送操作开始的中断使能位。（读/写）

SLC0INT_SLC_FRHOST_BIT7_INT_ENA Host 中断 Slave 的中断使能位 7。（读/写）

SLC0INT_SLC_FRHOST_BIT6_INT_ENA Host 中断 Slave 的中断使能位 6。（读/写）

SLC0INT_SLC_FRHOST_BIT5_INT_ENA Host 中断 Slave 的中断使能位 5。（读/写）

SLC0INT_SLC_FRHOST_BIT4_INT_ENA Host 中断 Slave 的中断使能位 4。（读/写）

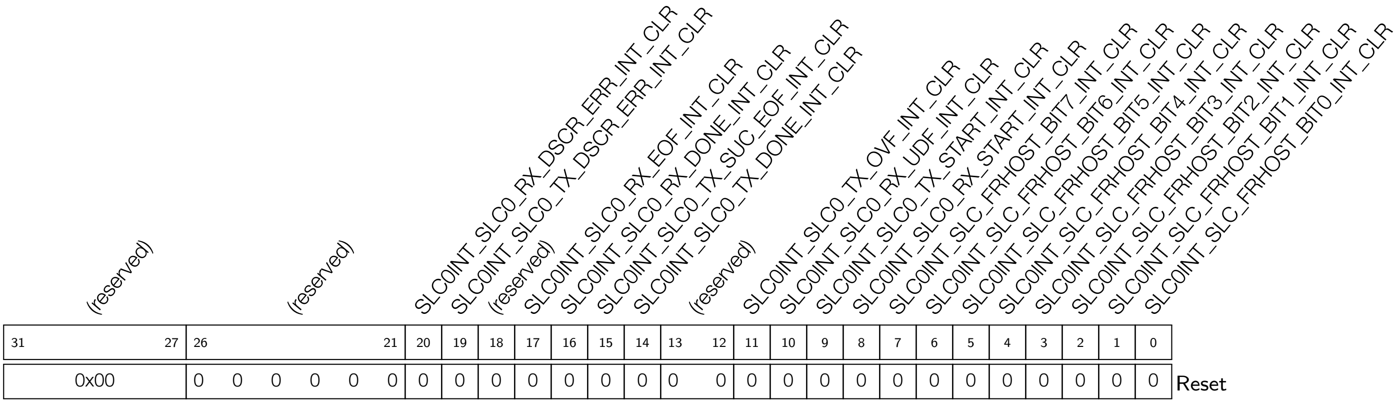
SLC0INT_SLC_FRHOST_BIT3_INT_ENA Host 中断 Slave 的中断使能位 3。（读/写）

SLC0INT_SLC_FRHOST_BIT2_INT_ENA Host 中断 Slave 的中断使能位 2。（读/写）

SLC0INT_SLC_FRHOST_BIT1_INT_ENA Host 中断 Slave 的中断使能位 1。（读/写）

SLC0INT_SLC_FRHOST_BIT0_INT_ENA Host 中断 Slave 的中断使能位 0。（读/写）

Register 8.5. SLC0INT_CLR_REG (0x10)



SLC0INT_SLC0_RX_DSCR_ERR_INT_CLR 将此位置为 1，清除发送链表描述符错误的中断。（只写）

SLC0INT_SLC0_TX_DSCR_ERR_INT_CLR 将此位置为 1，清除接收链表描述符错误的中断。（只写）

SLC0INT_SLC0_RX_EOF_INT_CLR 将此位置为 1，清除 Slave 发送操作结束的中断。（只写）

SLC0INT_SLC0_RX_DONE_INT_CLR 将此位置为 1，清除 Slave 发送模式下单个 buffer 发送完成的中断。（只写）

SLC0INT_SLC0_TX_SUC_EOF_INT_CLR 将此位置为 1，清除 Slave 接收操作完成的中断。（只写）

SLC0INT_SLC0_TX_DONE_INT_CLR 将此位置为 1，清除 Slave 接收模式下单个 buffer 填满了的中断。（只写）

SLC0INT_SLC0_TX_OVF_INT_CLR 将此位置为 1，清除 Slave 接收溢出中断。（只写）

SLC0INT_SLC0_RX_UDF_INT_CLR 将此位置为 1，清除 Slave 发送溢出中断。（只写）

SLC0INT_SLC0_TX_START_INT_CLR 将此位置为 1，清除 Slave 接收操作初始化中断。（只写）

SLC0INT_SLC0_RX_START_INT_CLR 将此位置为 1，清除 Slave 发送操作初始化中断。（只写）

SLC0INT_SLC_FRHOST_BIT7_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT7_INT中断。（只写）

SLC0INT_SLC_FRHOST_BIT6_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT6_INT 中断。（只写）

SLC0INT_SLC_FRHOST_BIT5_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT5_INT 中断。（只写）

SLC0INT_SLC_FRHOST_BIT4_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT4_INT 中断。（只写）

SLC0INT_SLC_FRHOST_BIT3_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT3_INT 中断。（只写）

SLC0INT_SLC_FRHOST_BIT2_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT2_INT 中断。（只写）

寄存器描述下一页继续。

Register 8.5. SLC0INT_CLR_REG (0x10)

继上一页寄存器描述。

SLC0INT_SLC_FRHOST_BIT1_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT1_INT 中断。（只写）

SLC0INT_SLC_FRHOST_BIT0_INT_CLR 置位清除 SLC0INT_SLC_FRHOST_BIT0_INT 中断。（只写）

Register 8.6. SLC0RX_LINK_REG (0x3C)

(reserved)	SLC0RX_SLC0_RXLINK_RESTART	SLC0RX_SLC0_RXLINK_START	SLC0RX_SLC0_RXLINK_STOP	(reserved)	SLC0RX_SLC0_RXLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

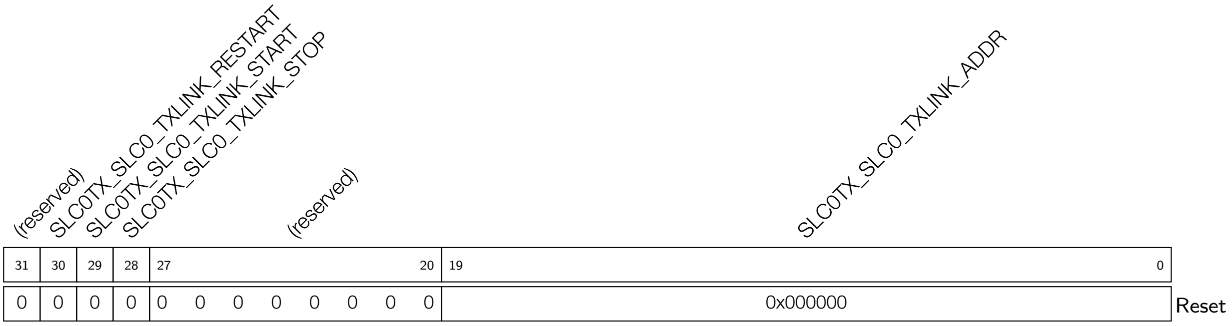
SLC0RX_SLC0_RXLINK_RESTART 将此位置为 1，重启并继续链表操作来发送包。（读/写）

SLC0RX_SLC0_RXLINK_START 将此位置为 1，启动链表操作来发送包。发送的起始地址由 SLC0_RXLINK_ADDR 给出。（读/写）

SLC0RX_SLC0_RXLINK_STOP 将此位置为 1，停止链表操作。（读/写）

SLC0RX_SLC0_RXLINK_ADDR Slave 发包链表的起始地址的低 20 位。（读/写）

Register 8.7. SLC0TX_LINK_REG (0x40)



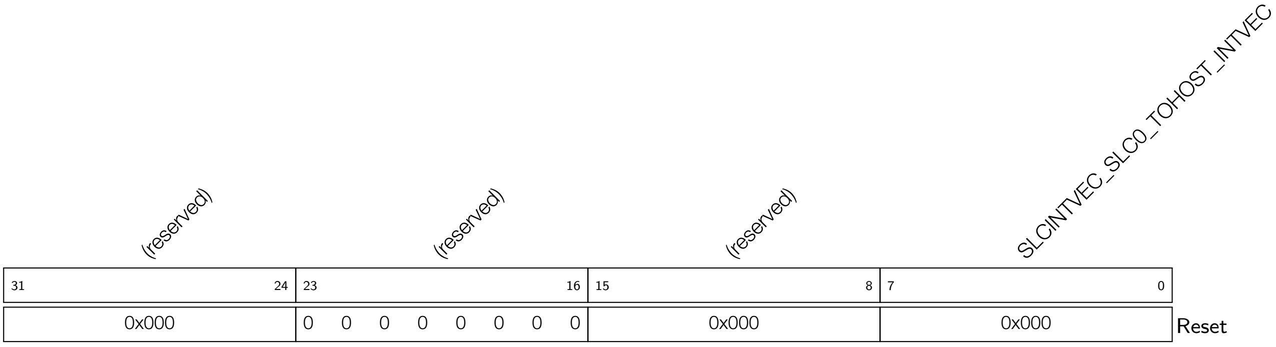
SLC0TX_SLC0_TXLINK_RESTART 将此位置为 1，重启并继续链表操作来接收包。（读/写）

SLC0TX_SLC0_TXLINK_START 将此位置为 1，启动链表操作来接收包。接收的起始地址由 SLC0_TXLINK_ADDR 给出。（读/写）

SLC0TX_SLC0_TXLINK_STOP 将此位置为 1，停止链表操作。（读/写）

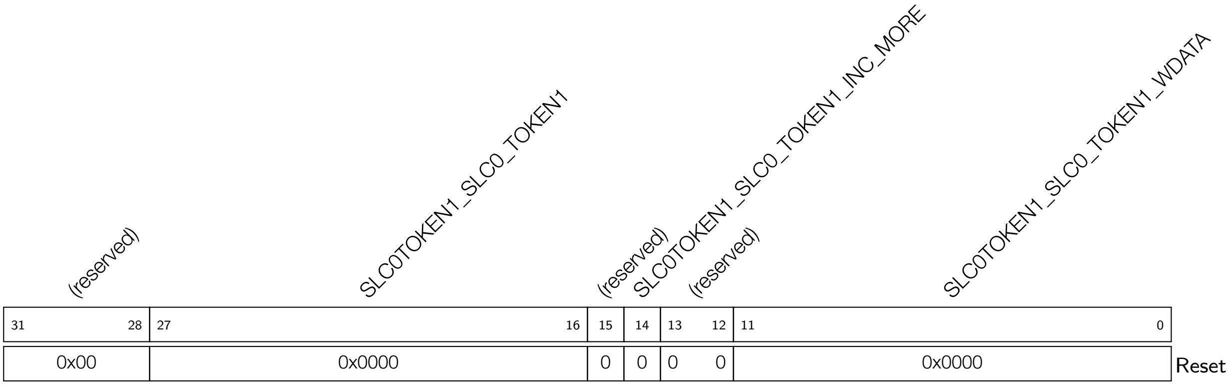
SLC0TX_SLC0_TXLINK_ADDR Slave 收包链表的起始地址的低 20 位。（读/写）

Register 8.8. SLCINTVEC_TOHOST_REG (0x4C)



SLCINTVEC_SLC0_TOHOST_INTVEC Slave 中断 Host 的中断向量。（只写）

Register 8.9. SLC0TOKEN1_REG (0x54)

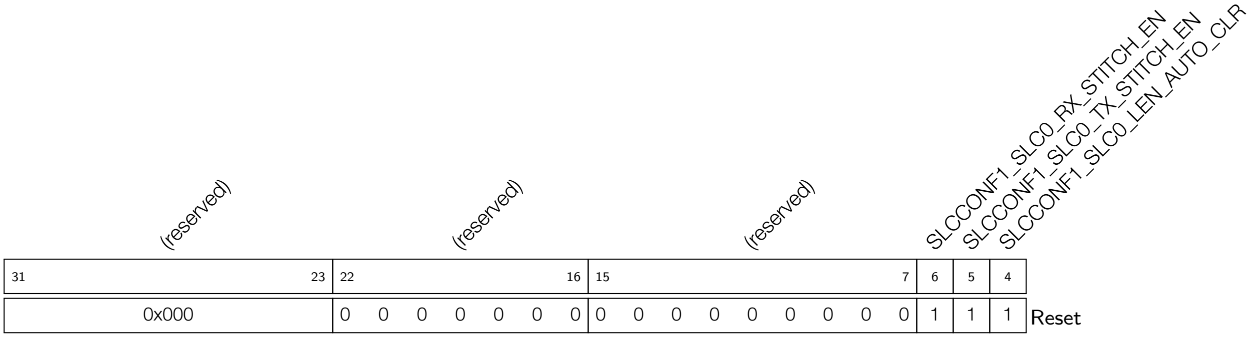


SLC0TOKEN1_SLC0_TOKEN1 收包 buffer 的累计数量。（只读）

SLC0TOKEN1_SLC0_TOKEN1_INC_MORE 累计可用的收包 buffer 的指示信号。（只写）

SLC0TOKEN1_SLC0_TOKEN1_WDATA 可用的收包 buffer 数量。（只写）

Register 8.10. SLCCONF1_REG (0x60)



SLCCONF1_SLC0_RX_STITCH_EN 初始化为 0。请勿修改。（读/写）

SLCCONF1_SLC0_TX_STITCH_EN 初始化为 0。请勿修改。（读/写）

SLCCONF1_SLC0_LEN_AUTO_CLR 初始化为 0。请勿修改。（读/写）

Register 8.11. SLC_RX_DSCR_CONF_REG (0x98)

(reserved)	SLC_SLC0_TOKEN_NO_REPLACE	
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

SLC_SLC0_TOKEN_NO_REPLACE 初始化为 1。请勿修改。(读 / 写)

Register 8.12. SLC0_LEN_CONF_REG (0xE4)

(reserved)	(reserved)	SLC0_LEN_INC_MORE	(reserved)	SLC0_LEN_WDATA	
31 29	28 23	22	21 20	19 0	
0x0	0 0 0 0 0 0	0	0 0	0x000000	Reset

SLC0_LEN_INC_MORE 累计发送的包长的指示信号。(只写)

SLC0_LEN_WDATA 发送的包长。(只写)

Register 8.13. SLC0_LENGTH_REG (0xE8)

(reserved)	SLC0_LEN	
31 20	19 0	
0x0000	0x000000	Reset

SLC0_LEN Slave 发包的累计长度。(只读)

8.6 SLC Host 寄存器

本小节括号中的地址均为相对于 SDIO 从机基地址 (0x3FF5_5000) 的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 8.4 寄存器列表。

Register 8.14. SLC0HOST_TOKEN_RDATA (0x44)

(reserved)	HOSTREG_SLC0_TOKEN1	(reserved)	
31 28	27 16	15 0	
0x000	0x000	0x000	Reset

HOSTREG_SLC0_TOKEN1 Slave 侧可用于接收 Host 数据的 buffer 数的累加值。（只读）

Register 8.15. SLC0HOST_INT_RAW_REG (0x50)

(reserved)	(reserved)	SLC0HOST_SLC0_RX_NEW_PACKET_INT_RAW	(reserved)	SLC0HOST_SLC0_TX_OVF_INT_RAW	SLC0HOST_SLC0_RX_UDF_INT_RAW	(reserved)	SLC0HOST_SLC0_TOHOST_BIT7_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT6_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT5_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT4_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT3_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT2_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT1_INT_RAW	SLC0HOST_SLC0_TOHOST_BIT0_INT_RAW	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	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

SLC0HOST_SLC0_RX_NEW_PACKET_INT_RAW SLC0HOST_SLC0_RX_NEW_PACKET_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TX_OVF_INT_RAW SLC0HOST_SLC0_TX_OVF_INT 的原始中断位。（只读）

SLC0HOST_SLC0_RX_UDF_INT_RAW SLC0HOST_SLC0_RX_UDF_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT7_INT_RAW SLC0HOST_SLC0_TOHOST_BIT7_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT6_INT_RAW SLC0HOST_SLC0_TOHOST_BIT6_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT5_INT_RAW SLC0HOST_SLC0_TOHOST_BIT5_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT4_INT_RAW SLC0HOST_SLC0_TOHOST_BIT4_INT的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT3_INT_RAW SLC0HOST_SLC0_TOHOST_BIT3_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT2_INT_RAW SLC0HOST_SLC0_TOHOST_BIT2_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT1_INT_RAW SLC0HOST_SLC0_TOHOST_BIT1_INT 的原始中断位。（只读）

SLC0HOST_SLC0_TOHOST_BIT0_INT_RAW SLC0HOST_SLC0_TOHOST_BIT0_INT 的原始中断位。（只读）

Register 8.16. SLC0HOST_INT_ST_REG (0x58)

(reserved)	(reserved)	SLC0HOST_SLC0_RX_NEW_PACKET_INT_ST	(reserved)	SLC0HOST_SLC0_TX_OVF_INT_ST	SLC0HOST_SLC0_RX_UDF_INT_ST	(reserved)	SLC0HOST_SLC0_TOHOST_BIT7_INT_ST	SLC0HOST_SLC0_TOHOST_BIT6_INT_ST	SLC0HOST_SLC0_TOHOST_BIT5_INT_ST	SLC0HOST_SLC0_TOHOST_BIT4_INT_ST	SLC0HOST_SLC0_TOHOST_BIT3_INT_ST	SLC0HOST_SLC0_TOHOST_BIT2_INT_ST	SLC0HOST_SLC0_TOHOST_BIT1_INT_ST	SLC0HOST_SLC0_TOHOST_BIT0_INT_ST	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	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

SLC0HOST_SLC0_RX_NEW_PACKET_INT_ST SLC0HOST_SLC0_RX_NEW_PACKET_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TX_OVF_INT_ST SLC0HOST_SLC0_TX_OVF_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_RX_UDF_INT_ST SLC0HOST_SLC0_RX_UDF_INT中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT7_INT_ST SLC0HOST_SLC0_TOHOST_BIT7_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT6_INT_ST SLC0HOST_SLC0_TOHOST_BIT6_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT5_INT_ST SLC0HOST_SLC0_TOHOST_BIT5_INT中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT4_INT_ST SLC0HOST_SLC0_TOHOST_BIT4_INT 中断的屏蔽中断状态位。(只读)

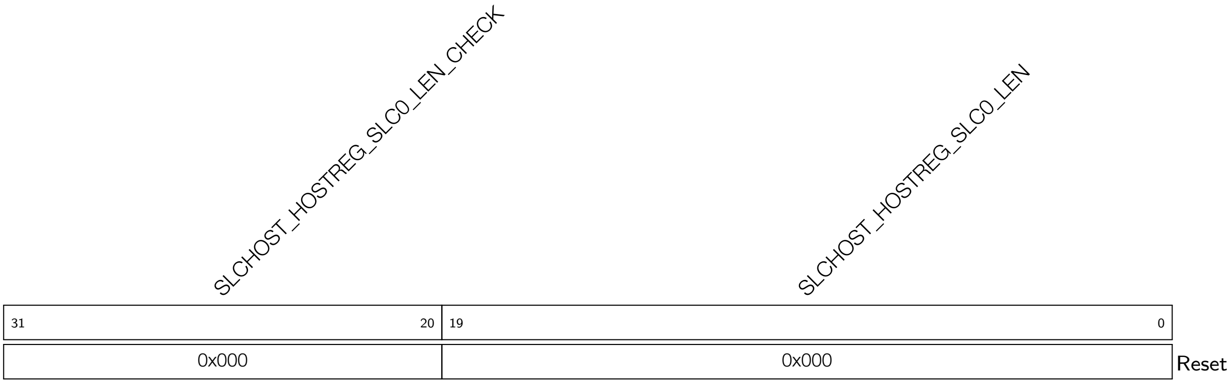
SLC0HOST_SLC0_TOHOST_BIT3_INT_ST SLC0HOST_SLC0_TOHOST_BIT3_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT2_INT_ST SLC0HOST_SLC0_TOHOST_BIT2_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT1_INT_ST SLC0HOST_SLC0_TOHOST_BIT1_INT 中断的屏蔽中断状态位。(只读)

SLC0HOST_SLC0_TOHOST_BIT0_INT_ST SLC0HOST_SLC0_TOHOST_BIT0_INT 中断的屏蔽中断状态位。(只读)

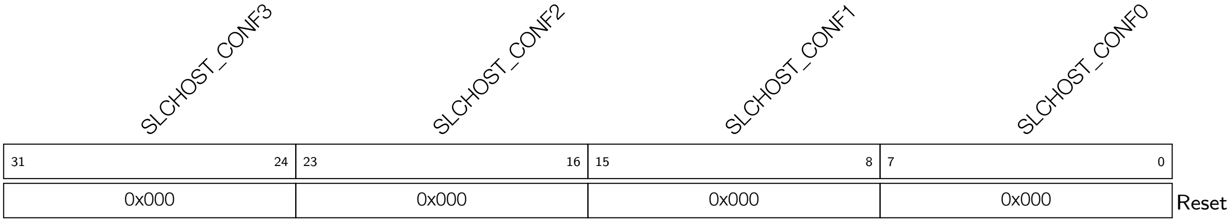
Register 8.17. SLCHOST_PKT_LEN_REG (0x60)



SLCHOST_HOSTREG_SLC0_LEN_CHECK 值为 HOSTREG_SLC0_LEN[9:0] 加上 HOSTREG_SLC0_LEN[19:10]。（只读）

SLCHOST_HOSTREG_SLC0_LEN Slave 发送的包长的累计值。只在 Host 读取时才会更新值。

Register 8.18. SLCHOST_CONF_W0_REG (0x6C)



SLCHOST_CONF3 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。（读/写）

SLCHOST_CONF2 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。（读/写）

SLCHOST_CONF1 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。（读/写）

SLCHOST_CONF0 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。（读/写）

Register 8.19. SLCHOST_CONF_W1_REG (0x70)

SLCHOST_CONF7	SLCHOST_CONF6	SLCHOST_CONF5	SLCHOST_CONF4	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF7 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF6 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF5 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF4 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.20. SLCHOST_CONF_W2_REG (0x74)

SLCHOST_CONF11	SLCHOST_CONF10	SLCHOST_CONF9	SLCHOST_CONF8	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF11 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF10 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF9 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF8 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

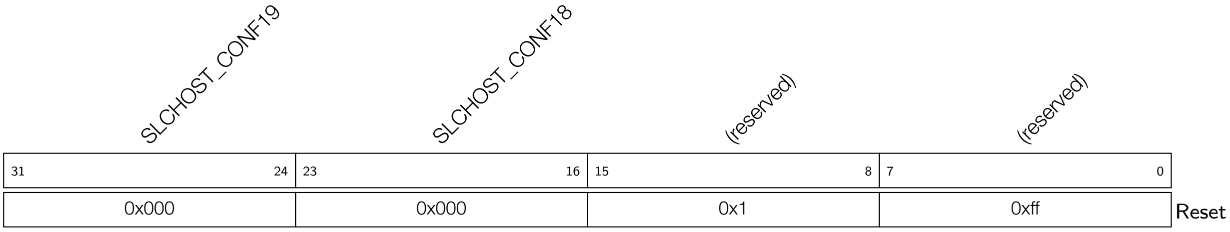
Register 8.21. SLCHOST_CONF_W3_REG (0x78)

SLCHOST_CONF15	SLCHOST_CONF14	(reserved)	(reserved)	
31 24	23 16	15 8	7 0	
0x000	0x000	0x0	0xc0	Reset

SLCHOST_CONF15 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

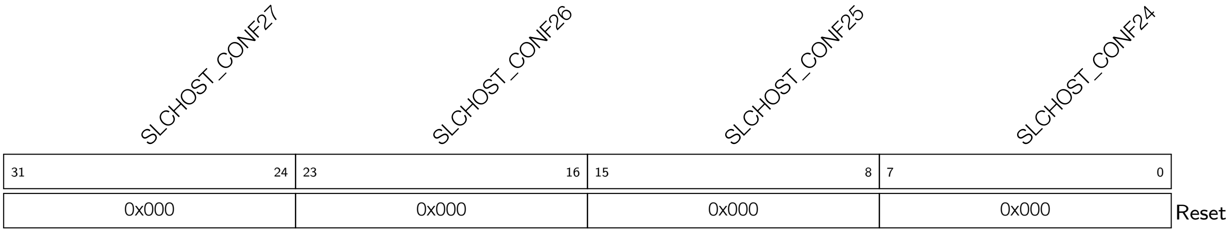
SLCHOST_CONF14 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.22. SLCHOST_CONF_W4_REG (0x7C)



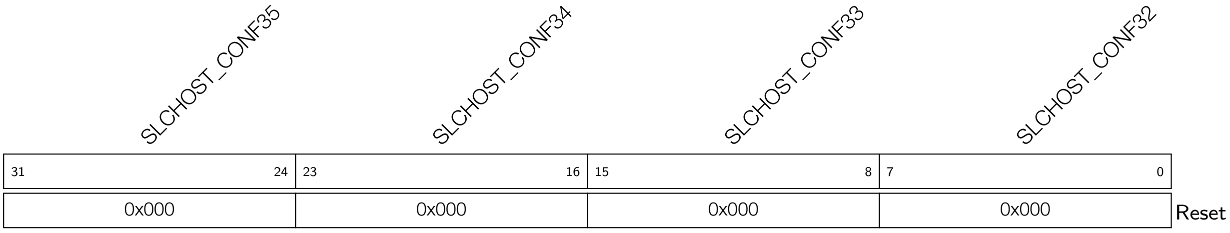
SLCHOST_CONF19	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF18	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.23. SLCHOST_CONF_W6_REG (0x88)



SLCHOST_CONF27	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF26	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF25	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF24	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.24. SLCHOST_CONF_W8_REG (0x9C)



SLCHOST_CONF35	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF34	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF33	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF32	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.25. SLCHOST_CONF_W9_REG (0xA0)

SLCHOST_CONF39	SLCHOST_CONF38	SLCHOST_CONF37	SLCHOST_CONF36	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF39	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF38	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF37	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF36	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.26. SLCHOST_CONF_W10_REG (0xA4)

SLCHOST_CONF43	SLCHOST_CONF42	SLCHOST_CONF41	SLCHOST_CONF40	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

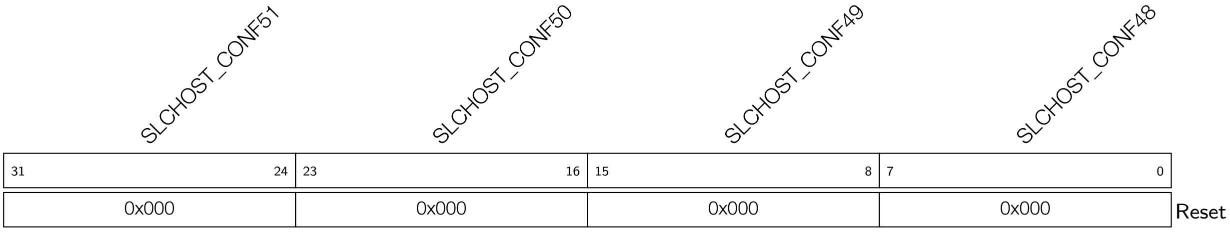
SLCHOST_CONF43	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF42	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF41	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF40	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.27. SLCHOST_CONF_W11_REG (0xA8)

SLCHOST_CONF47	SLCHOST_CONF46	SLCHOST_CONF45	SLCHOST_CONF44	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF47	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF46	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF45	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF44	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.28. SLCHOST_CONF_W12_REG (0xAC)



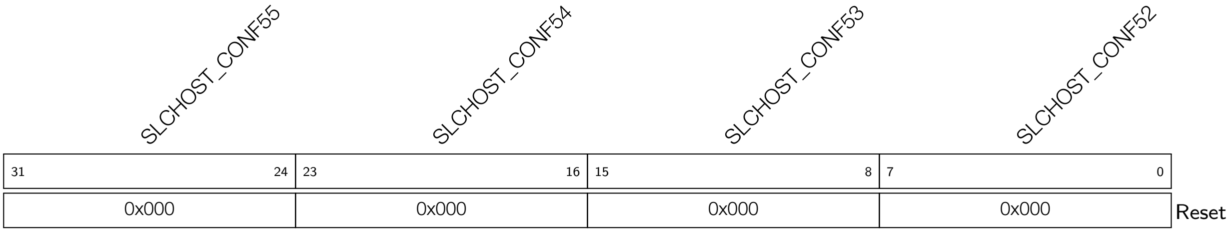
SLCHOST_CONF51 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF50 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF49 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF48 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.29. SLCHOST_CONF_W13_REG (0xB0)



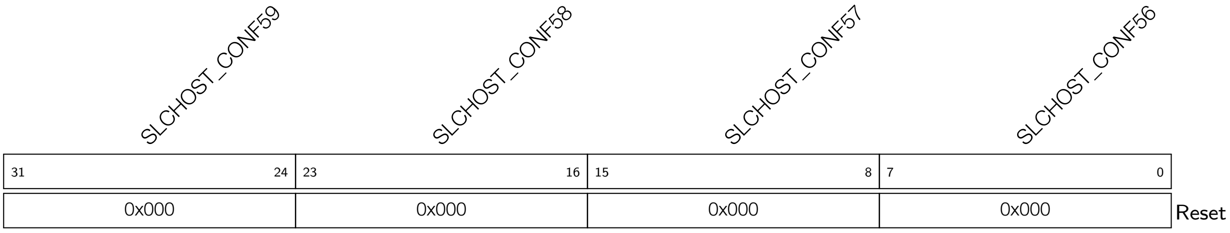
SLCHOST_CONF55 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF54 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF53 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF52 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.30. SLCHOST_CONF_W14_REG (0xB4)



SLCHOST_CONF59 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF58 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF57 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

SLCHOST_CONF56 Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.31. SLCHOST_CONF_W15_REG (0xB8)

SLCHOST_CONF63	SLCHOST_CONF62	SLCHOST_CONF61	SLCHOST_CONF60	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

SLCHOST_CONF63	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF62	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF61	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)
SLCHOST_CONF60	Host 与 Slave 的信息交互寄存器。Host 与 Slave 都可读写。(读/写)

Register 8.32. SLC0HOST_INT_CLR_REG (0xD4)

(reserved)	(reserved)	SLC0HOST_SLC0_RX_NEW_PACKET_INT_CLR	(reserved)	SLC0HOST_SLC0_TX_OVF_INT_CLR	SLC0HOST_SLC0_RX_UDF_INT_CLR	(reserved)	SLC0HOST_SLC0_TOHOST_BIT7_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT6_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT5_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT4_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT3_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT2_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT1_INT_CLR	SLC0HOST_SLC0_TOHOST_BIT0_INT_CLR	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	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

SLC0HOST_SLC0_RX_NEW_PACKET_INT_CLR 置位清除 SLC0HOST_SLC0_RX_NEW_PACKET_INT 中断。（只写）

SLC0HOST_SLC0_TX_OVF_INT_CLR 置位清除 SLC0HOST_SLC0_TX_OVF_INT 中断。（只写）

SLC0HOST_SLC0_RX_UDF_INT_CLR 置位清除 SLC0HOST_SLC0_RX_UDF_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT7_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT7_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT6_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT6_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT5_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT5_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT4_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT4_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT3_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT3_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT2_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT2_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT1_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT1_INT 中断。（只写）

SLC0HOST_SLC0_TOHOST_BIT0_INT_CLR 置位清除 SLC0HOST_SLC0_TOHOST_BIT0_INT 中断。（只写）

Register 8.33. SLC0HOST_FUNC1_INT_ENA_REG (0xDC)

(reserved)	(reserved)	SLC0HOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA	(reserved)	SLC0HOST_FN1_SLC0_TX_OVF_INT_ENA	SLC0HOST_FN1_SLC0_RX_UDF_INT_ENA	(reserved)	SLC0HOST_FN1_SLC0_TOHOST_BIT7_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT6_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT5_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT4_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT3_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT2_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT1_INT_ENA	SLC0HOST_FN1_SLC0_TOHOST_BIT0_INT_ENA	
31 26	25 24	23	22 18	17	16	15 8	7	6	5	4	3	2	1	0	
0x00	0 0	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

SLC0HOST_FN1_SLC0_RX_NEW_PACKET_INT_ENA SLC0HOST_FN1_SLC0_RX_NEW_PACKET_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TX_OVF_INT_ENA SLC0HOST_FN1_SLC0_TX_OVF_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_RX_UDF_INT_ENA SLC0HOST_FN1_SLC0_RX_UDF_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT7_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT7_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT6_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT6_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT5_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT5_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT4_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT4_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT3_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT3_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT2_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT2_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT1_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT1_INT 中断的中断使能位。(读/写)

SLC0HOST_FN1_SLC0_TOHOST_BIT0_INT_ENA SLC0HOST_FN1_SLC0_TOHOST_BIT0_INT 中断的中断使能位。(读/写)

Register 8.34. SLCHOST_CONF_W7_REG (0x8C)

SLCHOST_CONF31	(reserved)	SLCHOST_CONF29	(reserved)	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0x000	0 0 0 0 0 0 0 0	0x000	Reset

SLCHOST_CONF31 Host 用于中断 Slave 的中断向量。(只写)

SLCHOST_CONF29 Host 用于中断 Slave 的中断向量。(只写)

Register 8.35. SLCHOST_CONF_REG (0x1F0)

(reserved)	SLCHOST_HSPEED_CON_EN	(reserved)	SLCHOST_FRC_POS_SAMP	SLCHOST_FRC_NEG_SAMP	SLCHOST_FRC_SDIO20	SLCHOST_FRC_SDIO11	
31 28	27	26 20	19 15	14 10	9 5	4 0	
0 0 0 0	0	0 0 0 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

SLCHOST_HSPEED_CON_EN 置位此位后，且 HINF_HIGHSPEED_ENABLE 置 1 时，Slave 会在主机配置 CCCR 中的 EHS 为 1 后强制上升沿输出。(读/写)

SLCHOST_FRC_POS_SAMP 置位在时钟上升沿采样信号。(读/写)

SLCHOST_FRC_NEG_SAMP 置位在时钟下降沿采样信号。(读/写)

SLCHOST_FRC_SDIO20 置位在时钟上升沿输出信号。(读/写)

SLCHOST_FRC_SDIO11 置位在时钟下降沿输出信号。(读/写)

8.7 HINF 寄存器

本小节括号中的地址均为相对于 SDIO 从机基地址 (0x3FF4_B000) 的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 8.4 寄存器列表。

Register 8.36. HINF_CFG_DATA1_REG (0x4)

(reserved)	HINF_HIGHSPEED_ENABLE	HINF_SDIO_IOREADY1	
31 3	2	1	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

HINF_HIGHSPEED_ENABLE 初始化为 1。请勿修改。（读 / 写）

HINF_SDIO_IOREADY1 初始化为 1。请勿修改。（读 / 写）

9 SD/MMC 主机控制器

9.1 概述

ESP32 存储卡控制器提供了一个访问安全数字输入输出卡 (SDIO)、MMC 卡和 CE-ATA 设备的硬件接口，用于连接高级外围设备总线 (APB) 和外部存储设备。ESP32 支持两个外部卡（卡 0 和卡 1）。

9.2 主要特性

ESP32 存储卡控制器具有以下特性：

- 支持两个外部卡
- 支持 SD 存储卡 3.0 和 3.01 标准
- 支持 MMC 版本 4.41、4.5、4.51
- 支持 CE-ATA 版本 1.1
- 支持 1-bit、4-bit 和 8-bit（仅卡 0 支持）模式

SD/MMC 外设连接的拓扑结构如图 9-1 所示。存储卡控制器支持两组外设工作，但不支持同时工作。

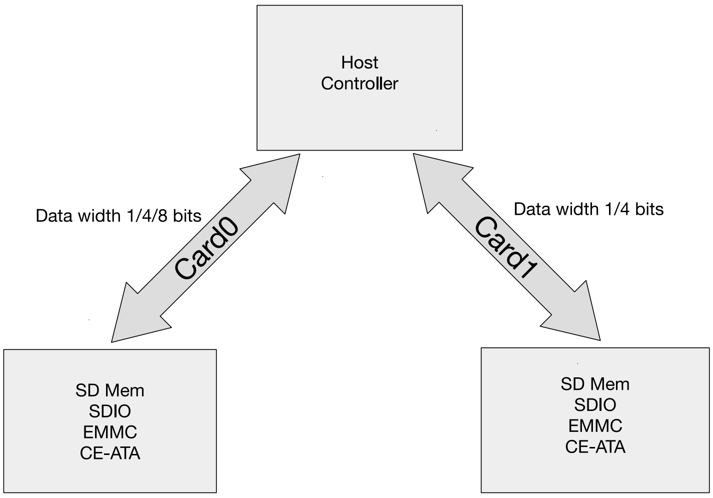


图 9-1. SD/MMC 外设连接的拓扑结构

9.3 SD/MMC 外部接口信号

SD/MMC 的外部接口信号主要为 clk、cmd、data 信号等，还包括卡中断、卡检测和写保护信号等。各个信号的方向如图 9-2 所示。每个管脚的方向和描述如表 9-1 所示。

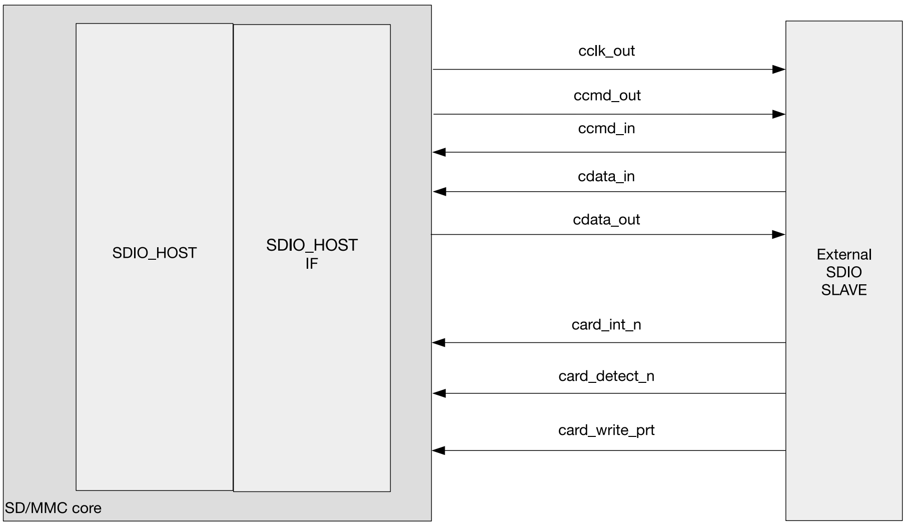


图 9-2. SD/MMC 外部接口信号

表 9-1. SD/MMC 管脚描述

管脚	方向	描述
cclk_out	输出	主机输出给从机的时钟线
ccmd	双向	指令 / 响应双向传输线
cdata	双向	数据读写双向传输线
card_detect_n	输入	探测接口上是否有卡存在的输入线
card_write_prt	输入	卡写保护输入线

9.4 功能描述

9.4.1 SD/MMC 架构

SD/MMC 的结构主要分为总线接口单元 (BIU) 和卡接口单元 (CIU) 两部分，如图 9-3 所示。其中：

BIU 模块：提供寄存器访问的 APB 接口、FIFO 方式读写数据，和数据读写操作的 DMA 访问。

CIU 模块：控制外部存储卡的接口协议，还提供时钟控制。

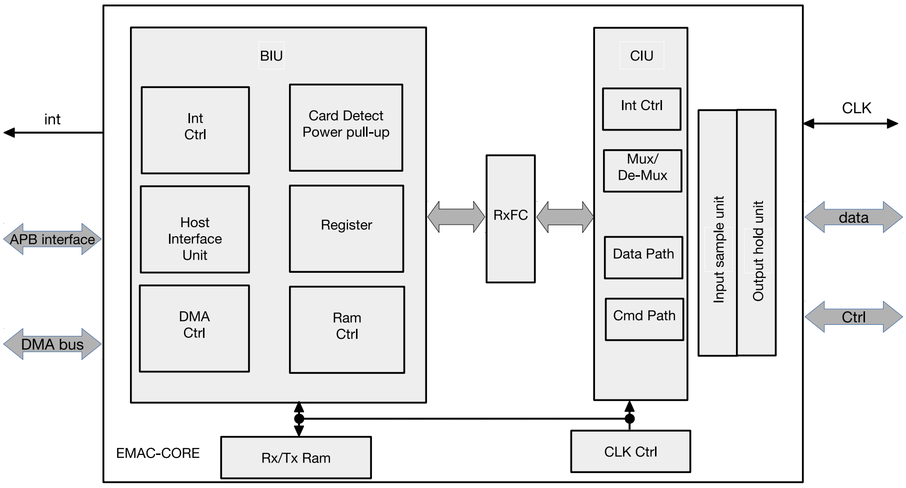


图 9-3. SD/MMC 基本架构

9.4.1.1 BIU 模块

该模块通过主机接口单元 (HIU) APB 总线的方式访问寄存器和数据 FIFO。此外，它通过 DMA 接口提供独立的数据访问。图 9-3 中展示了 BIU 结构框图。

BIU 提供以下功能模块：

- 主机接口
- DMA 接口
- 中断控制
- 寄存器访问
- FIFO 访问
- 上电/上拉控制和卡检测

9.4.1.2 CIU 模块

该模块实现控制卡的接口协议。在 CIU 中，命令通路 (Cmd Path) 控制单元和数据通路 (Data Path) 控制单元将控制器连接到 SD/MMC/CE-ATA 卡的命令和数据端口。CIU 还提供时钟控制逻辑。图 9-3 展示了 CIU 结构框图。

CIU 包含以下主要功能模块：

- 命令通路
- 数据通路
- SDIO 中断控制
- 时钟控制
- Mux/Demux 单元

9.4.2 命令通路

该命令通路具有以下功能：

- 设置时钟参数
- 设置卡命令参数
- 向卡总线发送命令（ccmd_out 线）
- 接收卡总线响应（ccmd_in 线）
- 向 BIU 发送响应
- 在命令线上发送 P-bit 位

命令通路状态机如图 9-4 所示。

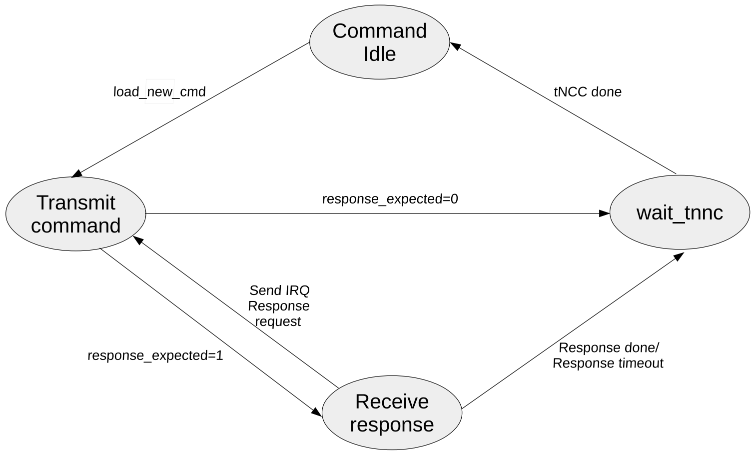


图 9-4. 命令通路状态机

9.4.3 数据通路

数据通路模块在写入数据发送时弹出 FIFO 中的数据并在 cdata_out 上发送数据；或者在读取数据时接收 cdata_in 上的数据并将其导入 FIFO。只有数据发送命令不运行时，数据通路才会加载新的数据参数，包括 expected data、读/写数据发送、流/块发送、块大小、字节计数、卡类型、超时寄存器等。

如果在命令寄存器中设置了 data_expected 位，则新命令是数据传输命令，数据通路将开始执行以下操作：

- 若读/写位为 1 时，发送数据
- 若读/写位为 0 时，接收数据

9.4.3.1 数据发送

数据发送状态机如图 9-5 所示。模块在接收到数据写入命令的响应之后的两个时钟周期开始发送数据；即使命令通路检测到响应错误或循环冗余检查 (CRC) 错误，也会出现这种情况。如果由于响应超时而没有从卡接收到响应，则不发送数据。根据命令寄存器中 transfer_mode 位的值，数据发送状态机将数据以流或块的形式放在卡数据总线上。

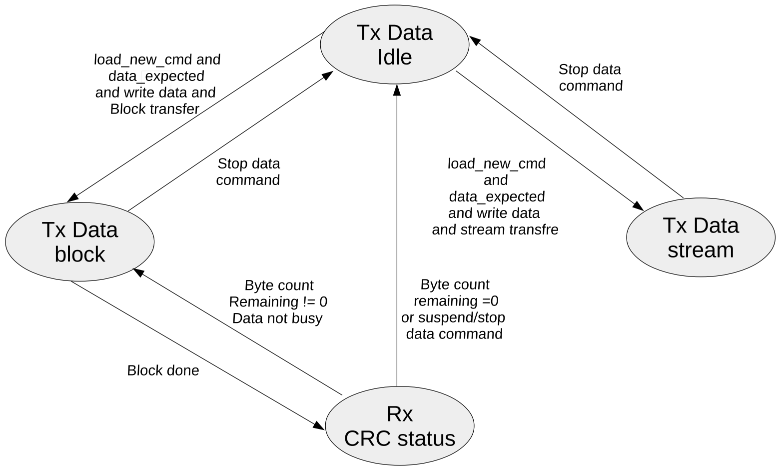


图 9-5. 数据传输状态机

9.4.3.2 数据接收

数据接收状态机如图 9-6 所示。模块在数据读取命令完成后的两个时钟周期开始接收数据；即使命令通路检测到响应错误或响应 CRC 错误，也会如此。如果由于响应超时而未从卡接收到响应，那么 BIU 接收不到数据传输完成的信号。如果 CIU 发送的命令是卡的非法操作，那么卡无法读取数据，BIU 也不会接收到数据传输完成的信号。

若在数据超时前未接收到数据，则数据通路向 BIU 发出数据超时信号并结束数据传输。根据命令寄存器中的 transfer_mode 位的值，数据接收状态机以流或块的形式从卡数据总线获取数据。

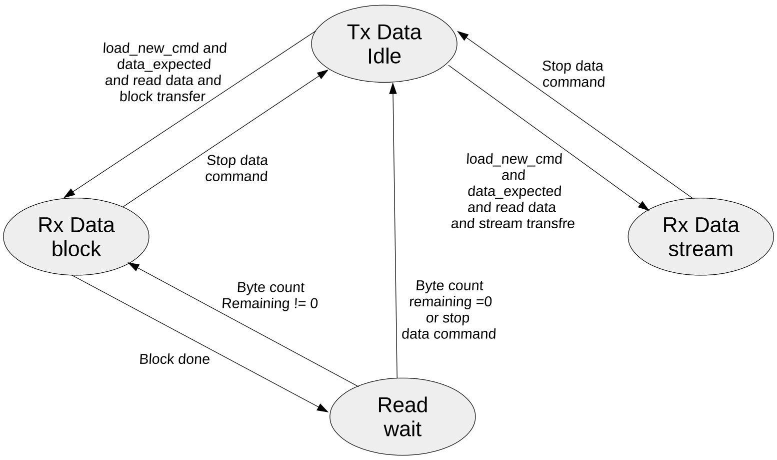


图 9-6. 数据接收状态机

9.5 CIU 操作的软件限制

- 一次只能选择一个卡进行命令或数据传输。例如，当卡传送数据时，不应将新命令发送到另一张卡。但是新命令可以发送到同一张卡，用于读取状态或停止数据传输。
- 一次只能发出一个数据传输命令。
- 在开放式卡写操作期间，如果卡时钟因为 FIFO 为空而停止，那么软件必须首先将数据填充到 FIFO 中并启动卡时钟，然后才可以向卡发出一个停止/中止命令。
- 在 SDIO/COMBO 卡传输期间，如果卡功能暂停，并且软件要恢复所暂停的传输，则必须首先重置 FIFO 并启动恢复命令，这和启动一个新的数据传输命令相似。
- 在进行卡传输时发出卡复位命令（CMD0、CMD15 或者 CMD52_reset），软件必须在命令寄存器上设置 stop_abort_cmd 位，保证 CIU 可以在发出卡复位命令后停止数据传输。
- 当在 RINTSTS 寄存器中设置数据结束位错误时，CIU 不能保证 SDIO 中断。所以软件应忽略 SDIO 中断，并向卡发出停止/中止命令，使得卡停止进行读取数据传输。
- 若在一个读卡过程中因为 FIFO 已满而停止卡时钟，软件应该至少读取两个 FIFO 地址来重启卡时钟。
- 在一次命令/数据传输中只能选取一个 CE-ATA 设备。例如，当一个 CE-ATA 设备传输数据，其它 CE-ATA 设备不可传输新命令。
- 使能 CE-ATA 设备的中断（nIEN = 0），若运行时有正在进行的 RW_BLK 命令，则不应将新的 RW_BLK 命令发送到同一设备（在此数据库中使用的 RW_BLK 命令是由 CE-ATA 规格书定义的 RW_MULTIPLE_BLOCK MMC 命令）。只有在等待 CCS 时可以发送 CCSD。

- 对同一设备来说，若中断在 CE-ATA 设备上未使能 (nIEN = 1)，则需要一个新命令来读取状态信息。
- CE-ATA 设备不支持开放式传输。
- CE-ATA 传输不支持 send_auto_stop 信号（软件不应设置 send_auto_stop 位）。

当命令起始位被置上去之后，在命令被发送出去之前以下寄存器的值不能改变：

- CMD — 命令
- CMDARG — 命令参数
- BYTCNT — 字节计数
- BLKSIZ — 块大小
- CLKDIV — 时钟分频器
- CKLENA — 时钟使能
- CLKSRC — 时钟源
- TMOUT — 超时
- CTYPE — 卡类型

9.6 收发数据 RAM

数据 RAM 子模块是一个收发数据缓冲区，分为接收和发送两个单元。收发时可以通过 CPU 的 APB 接口和 DMA 两种方式来进行读写操作，DMA 方式在章节 9.8 中有详细介绍。

9.6.1 发送 RAM 模块

写数据有两种方式：DMA 方式和 CPU 读写。

如果使能 SDIO 发送，那么可以通过 APB 总线接口或 DMA 方式将数据写入到发送的 RAM 里面。其中，APB 的方式为 CPU 直接将数据写入寄存器 EMAC_FIFO。

9.6.2 接收 RAM 模块

读数据有两种方式：DMA 方式和 CPU 读写。

当数据通路子单元接收到数据时，该子数据会写入接收的 RAM 里面。在读取端可以通过 APB 总线或 DMA 的方式读出放入 RAM 中的数据。其中，APB 的方式为 CPU 直接读取寄存器 EMAC_FIFO 中的值。

9.7 链表环结构

每一组链表由两部分组成：链表本身和数据 buffer。每一个链表指向一个唯一的数据 buffer 和下一个链表。链表环结构如图 9-7 所示。

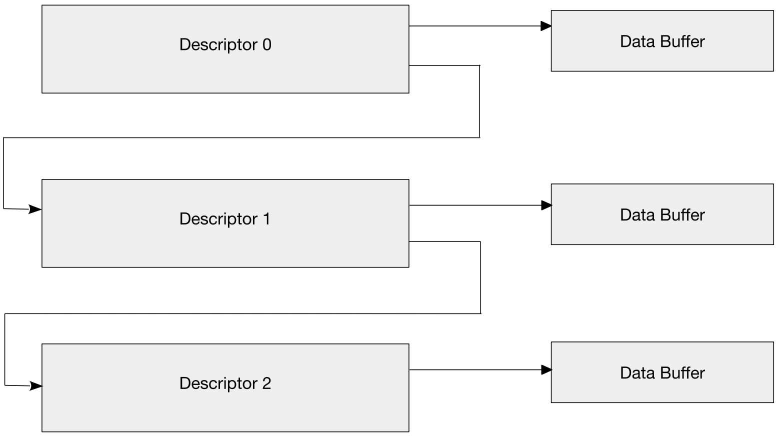


图 9-7. 链表环结构

9.8 链表结构

每个链表的结构如下，每个链表由 4 个 word 组成，如图 9-8 所示。表 9-2、表 9-3、表 9-4、表 9-5 为链表描述。

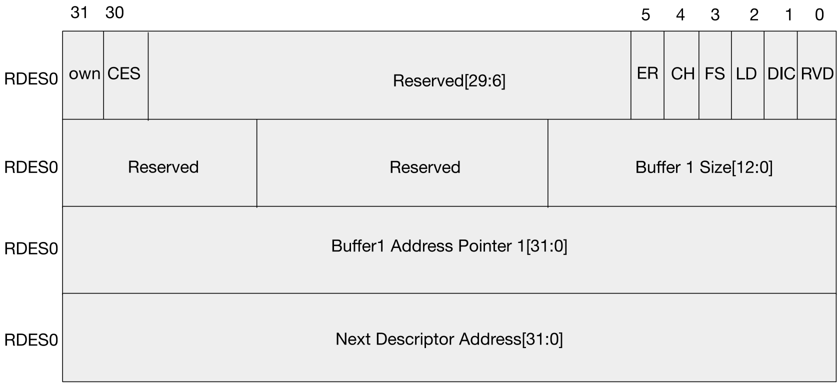


图 9-8. 链表结构

DES0 单元包含了控制和状态信息。

表 9-2. DES0 链表描述

位	名称	描述
31	OWN	当设置时，该位表明链表归 DMAC 所有。当该位被重置，该位表明链表归主机所有。DMAC 在完成数据传输后清除该位。

位	名称	描述
30	CES (Card Error Summary)	这些错误位表明了卡读/写的状态。 这些位也存在于 RINTSTS 之中，CES 是下列位的或运算： • EBE：结束位错误 • RTO：响应超时 • RCRC：响应 CRC • SBE：起始位错误 • DRTO：读取数据超时 • DCRC：用于接收的数据循环冗余校验 • RE：响应错误
29:6	Reserved	Reserved
5	ER (End of Ring)	置位时，该位表示链表已经到达最后一个链表。DMAC 返回到链表的基地址，创建一个链表环。
4	CH (Second Address Chained)	置位时，该位表示链表的第二个地址是下一个链表地址。置位该位时，BS2 (DES1[25:13]) 应该都归零。
3	FD (First Descriptor)	置位时，该位表示该链表包含了数据的第一缓冲区。若第一个缓冲区的大小为 0，则下一个链表包含数据的开始。
2	LD (Last Descriptor)	该位与 DMA 传输的最后一个数据块相关。置位时，该位表示通过该链表指向的缓冲区是数据的最后一个缓冲区。在该链表完成之后，剩余字节计数为 0。换句话说，带有被置位的 LD 位的链表完成之后，剩余字节计数应为 0。
1	DIC (Disable Interrupt on Completion)	置位时，为了保留在该链表指向的缓冲区中结束的数据，该位将阻止 DMAC 状态寄存器 (IDSTS) 上 TI/RI 位的设置。
0	Reserved	Reserved

DES1 元素包含了缓冲区大小。

表 9-3. DES1

位	名称	描述
31:26	Reserved	Reserved
25:13	Reserved	Reserved
12:0	BS1（Buffer 1 大小）	这些位表示数据缓冲区字节大小。缓冲区大小必须是 4 的倍数。如果缓冲区大小不是 4 的倍数，其结果暂未定义。该区不应为 0。

DES2 元素包含了指向数据缓冲区的地址指针。

表 9-4. DES2

位	名称	描述
31:0	Buffer Address Pointer 1 （数据缓冲区地址指针）	这些位表示数据缓冲区的物理地址。

如果当前的链表不是链表结构中的最后一个链表，那么 DES3 元素会包含指向下一个链表的地址指针。

表 9-5. DES3

位	名称	描述
31:0	Next Descriptor Address （下一个链表地址）	如果置位第二个地址链字节 (DES0[4])，则该位会包含此地址包含指向下一个链表存在的物理内存的指针。如果这不是最后一个链表，则下一个链表地址指针必须满足 DES3[1:0] = 0。

9.9 初始化

9.9.1 DMAC 初始化

DMAC 初始化在以下情况出现：

1. 向 DMAC Bus Mode Register (BMOD_REG) 来设置主机总线访问参数。
2. 向 DMAC Interrupt Enable Register (IDINTEN) 写入数据来屏蔽不必要的中断类型。
3. 软件驱动器创建传输链表列表或者接收链表列表。然后写入 DMAC 链表列表基址寄存器 (BDADDR)，为 DMAC 提供列表的起始地址。
4. DMAC 引擎尝试从链表列表获取链表。

9.9.2 DMAC 数据发送初始化

使用 DMAC 发送应遵循以下配置：

1. 软件设置用于发送的元素 (DES0-DES3)，设置 OWN 位 (DES0[31])，并准备了数据缓冲区。
2. 软件在 BIU 中的 CMD 寄存器中配置写数据命令。
3. 软件还将设置所需的发送阈值大小（即 FIFOTH 寄存器中的 TX_WMARK 字段）。
4. DMAC 引擎读取链表并检查 OWN 位。如果 OWN 位未置位，则表明软件拥有链表。在这种情况下，DMAC 挂起，并在 IDSTS 寄存器中产生 Descriptor Unable 中断。此时，主机需要通过将任何值写入 PLDMND_REG 来释放 DMAC。
5. 然后软件等待 Command Done 位，如果没有错误从 BIU 报出，则表明可以完成发送。
6. 然后 DMAC 引擎等待来自于 BIU 的 DMA 接口请求 (dw_dma_req)。该请求将基于配置的发送阈值生成。对于使用突发而不能访问的最后一个字节，则在 AHB 主接口上执行单次发送。
7. DMAC 从软件内存的数据缓冲区读取发送数据，并通过 FIFO 将其传入卡中。
8. 当数据跨越多个链表时，DMAC 将获取下一个链表，并继续使用下一个链表对其进行操作。最后一位链表位表明数据是否跨越多个链表。

9. 当数据发送完成时，通过设置发送中断来更新 IDSTS 寄存器中的状态信息。另外，OWN 位由 DMAC 通过对 DES0 执行写操作来清零。

9.9.3 DMAC 数据接收初始化

使用 DMAC 接收应遵循以下配置：

1. 软件为接收数据建立元素 (DES0 - DES3)，设置 OWN 位 (DES0[31])。
2. 软件通过 BIU 配置命令寄存器中为读数据命令。
3. 软件设置所需的接收阈值（FIFOTH 寄存器中的 RX_WMARK 字段）。
4. DMAC 引擎读取链表并检查 OWN 位。如果 OWN 位未设置，则表明软件拥有链表。在这种情况下，DMAC 挂起，并在 IDSTS 寄存器中产生 Descriptor Unable 中断。此时，软件需要通过将任何值写入 PLDMND_REG 来释放 DMAC。
5. 然后软件等待 Command Done 位，如果没有错误从 BIU 报出，则表明可以完成接收。
6. DMAC 引擎现在将等待来自于 BIU 的 DMA 接口请求 (dw_dma_req)。该请求将基于配置的发送阈值生成。对于使用 burst 而不能访问的最后一个字节，则在 AHB 上执行单次传输。
7. DMAC 通过 FIFO 读取数据，并传输至软件内存。
8. 当数据跨越多个链表时，DMAC 将获取下一个链表，并继续使用下一个链表对其进行操作。最后一位链表位表明数据是否跨越多个链表。
9. 当数据传输完成时，通过设置发送中断来更新 IDSTS 寄存器中的状态信息。另外，OWN 位由 DMAC 通过对 DES0 执行写操作来清零。

9.10 SD/MMC 时序

图 9-9 为高速 (HS) 模式下 SD/MMC 接口的时序图，表 9-6 为时序要求。这些时序要求对保证数据通信的可靠性和同步性至关重要。

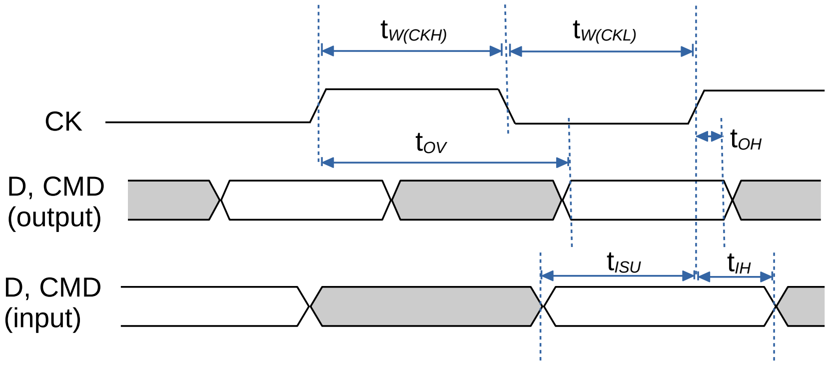


图 9-9. HS 模式下 SD/MMC 时序图

表 9-6. 高速模式下 SD/MMC 接口时序要求

符号	参数	条件	最小值	典型值	最大值	单位
$t_{W(CKL)}$	时钟低电平时间	f_{PP} = 80 MHz	11.5	12.5	—	ns
$t_{W(CKH)}$	时钟高电平时间	f_{PP} = 80 MHz	11.5	12.5	—	ns
t_{ISU}	输入建立时间	f_{PP} = 80 MHz	3.4	—	—	ns
t_{IH}	输入保持时间	f_{PP} = 80 MHz	1.1	—	—	ns
t_{OV}	输出有效时间	f_{PP} = 80 MHz	—	—	5.9	ns
t_{OH}	输出保持时间	f_{PP} = 80 MHz	0.5	—	—	ns

说明：

- f_{PP} - 数据传输的时钟频率。
- $t_{W(CKL)}/t_{W(CKH)}$ - 时钟低/高电平时间：表示在时钟频率 (f_{PP}) 为 80 MHz 时，时钟信号 (CK) 应保持在低电平或高电平状态的时间。
- t_{ISU} - 输入建立时间：指的是 CMD 和 D 数据线上的输入信号达到稳定所需的时间。
- t_{IH} - 输入保持时间：指的是 CMD 和 D 数据线上的输入信号需要保持稳定的时间。
- t_{OV} - 输出有效时间：指的是 CMD 和 D 数据线上的输出信号达到稳定、可以使用所需要的时间。
- t_{OH} - 输出保持时间：指的是 CMD 和 D 数据线上的输出信号保持有效的时间。
- CMD 和 D 输入/输出的时序是相对于时钟信号 CK 来测量的。

9.11 时钟相位选择

如果输入信号或输出信号的采样有时序问题，用户可以参照下图改变时钟相位。

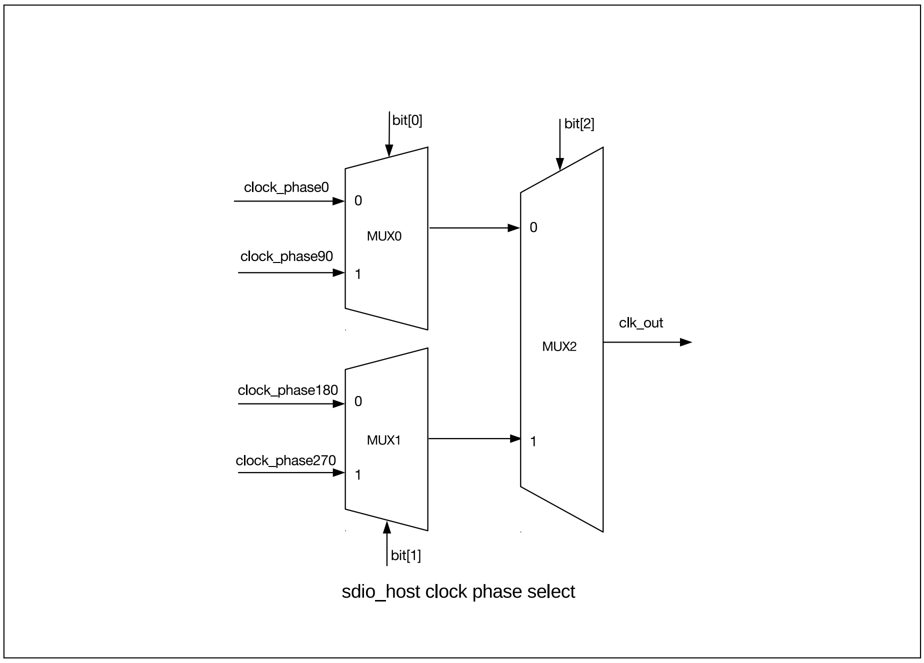


图 9-10. 时钟相位选择

有关时钟相位选择寄存器 CLK_EDGE_SEL 的说明，请见寄存器章节。

9.12 中断

中断可以通过各种情况产生。IDSTS 寄存器包含了可能导致中断的所有位。IDINTEN 寄存器包含了一个适用于所有能导致中断情况的使能位。

IDSTS 寄存器中有两组中断汇总—正常中断汇总 bit8 NIS 和异常中断汇总 bit9 AIS。将 1 写入相应的位，可以清除中断。当某组中的所有使能中断都被清除，相应的汇总位将被清零。当两组的汇总位都被清零，中断信号 dmac_intr_o 将取消。

中断不排序，如果中断在驱动程序响应之前发生，则不会产生其他中断。例如，接收中断 IDSTS [1] 表示一个或多个数据已传输到主机缓冲区。

多个事件同时只会产生一个中断。驱动程序必须扫描 IDSTS 寄存器来查找导致中断的原因。

9.13 寄存器列表

本小节的所有地址均为相对于 SD/MMC 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。

名称	描述	地址	访问
CTRL_REG	控制寄存器	0x0000	R/W
CLKDIV_REG	时钟分频配置寄存器	0x0008	R/W
CLKSRC_REG	时钟源选择寄存器	0x000C	R/W
CLKENA_REG	时钟使能寄存器	0x0010	R/W
TMOUT_REG	读取数据和响应超时配置寄存器	0x0014	R/W
CTYPE_REG	卡总线宽度配置寄存器	0x0018	R/W
BLKSIZ_REG	卡数据块大小配置寄存器	0x001C	R/W
BYTCNT_REG	数据传输长度配置寄存器	0x0020	R/W
INTMASK_REG	SDIO 中断屏蔽寄存器	0x0024	R/W
CMDARG_REG	命令参数数据寄存器	0x0028	R/W
CMD_REG	命令和启动配置寄存器	0x002C	R/W
RESP0_REG	响应数据寄存器	0x0030	RO
RESP1_REG	长响应数据寄存器	0x0034	RO
RESP2_REG	长响应数据寄存器	0x0038	RO
RESP3_REG	长响应数据寄存器	0x003C	RO
MINTSTS_REG	屏蔽中断状态寄存器	0x0040	RO
RINTSTS_REG	原始中断状态寄存器	0x0044	R/W
STATUS_REG	SD/MMC 状态寄存器	0x0048	RO
FIFOTH_REG	FIFO 配置寄存器	0x004C	R/W
CDETECT_REG	卡检测寄存器	0x0050	RO
WRTPRT_REG	卡写保护 (WP) 状态寄存器	0x0054	RO
TCBCNT_REG	传输字节计数寄存器	0x005C	RO
TBBCNT_REG	传输字节计数寄存器	0x0060	RO
DEBNCE_REG	配置去抖动过滤时间	0x0064	R/W
USRID_REG	用户 ID（暂存器）寄存器	0x0068	R/W
RST_N_REG	卡复位寄存器	0x0078	R/W
BMOD_REG	burst 模式传输配置寄存器	0x0080	R/W
PLDMND_REG	轮询需求配置寄存器	0x0084	WO

名称	描述	地址	访问
DBADDR_REG	描述符基地址寄存器	0x0088	R/W
IDSTS_REG	IDMAC 状态寄存器	0x008C	R/W
IDINTEN_REG	IDMAC 中断使能寄存器	0x0090	R/W
DSCADDR_REG	Host 描述符地址指针	0x0094	RO
BUFADDR_REG	Host buffer 地址指针寄存器	0x0098	RO
CLK_EDGE_SEL	时钟相位选择寄存器	0x0800	R/W

9.14 寄存器

SD/MMC 寄存器通过 CPU 的 APB 总线访问。

本小节的所有地址均为相对于 SD/MMC 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。

Register 9.1. CTRL_REG (0x0000)

	(reserved)	(reserved)	(reserved)	CEATA_DEVICE_INTERRUPT_STATUS	SEND_AUTO_STOP_CCSD	SEND_CCSD	ABORT_READ_DATA	SEND_IRQ_RESPONSE	READ_WAIT	(reserved)	INT_ENABLE	(reserved)	DMA_RESET	FIFO_RESET	CONTROLLER_RESET
	31 25	24	23 12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0x00	1	0x00	0	0	0	0	0	0	0	0	0	0	0	0

CEATA_DEVICE_INTERRUPT_STATUS 软件应在 CE-ATA 设备上电复位或其他复位后写此位。CE-ATA 设备的中断通常被禁能 (nIEN = 1)。如果主机使能中断，则软件应将此位置为 1。（读/写）

SEND_AUTO_STOP_CCSD 始终将 send_auto_stop_ccsd 和 send_ccsd 一起置位，不能分开单独置位。置位时，SD/MMC 自动发送内部生成的 STOP 指令 (CMD12) 给 CE-ATA 设备。发送 STOP 指令后，RINTSTS 里的 Auto Command Done (ACD) 位被置为 1，如果 ACD 中断没有屏蔽，则用于主机的中断将会生成。发送 Command Completion Signal Disable (CCSD) 后，SD/MMC 自动清零 send_auto_stop_ccsd 位。（读 / 写）

SEND_CCSD 置位时，SD/MMC 发送 CCSD 给 CE-ATA 设备。只有在当前指令等待 CCS（即 RW_BLK）并且 CE-ATA 设备的中断使能时软件才会将此位置位。一旦 CCSD 模式发送给设备，SD/MMC 就会自动清零 send_ccsd 位。如果 ACD 中断没有屏蔽，则寄存器 RINTSTS 里的 Command Done (CD) 位会被置为 1，且主机的中断将会生成。说明：一旦 SEND_CCSD 被置位，需要两个卡时钟周期来驱动 CMD 线上的 CCSD。因此，即使设备已经发送 CCS 信号，在边界条件内 CCSD 信号可能会发送给 CE-ATA 设备。（读 / 写）

ABORT_READ_DATA 在读操作期间，suspend 指令发送后，软件会轮询卡并找出 suspend 事件是从什么时候开始的。一旦 suspend 事件开始，软件会复位数据状态机，此时状态机正在等待下一个数据块。当数据状态机复位为空闲状态时，此位自动清零。（读 / 写）

SEND_IRQ_RESPONSE 响应发送后此位自动清零。为了等待 MMC 卡发送的中断，主机发送 CMD40，然后等待 MMC 卡的中断响应。同时，如果主机想要 SD/MMC 退出等待中断的状态，则会将此位置 1，此时 SD/MMC 指令状态机发送 CMD40 响应并返回空闲状态。（读 / 写）

READ_WAIT 发送读操作等待给 SDIO 卡。（读 / 写

INT_ENABLE 全局中断使能/禁能位。0：禁能；1：使能。（读 / 写）

DMA_RESET 要复位 DMA 接口，软件应将此位置为 1。两个 AHB 时钟后此位自动清零。（读 / 写）

寄存器描述下一页继续。

Register 9.1. CTRL_REG (0x0000)

继上一页寄存器描述。

FIFO_RESET 要复位 FIFO，软件应将此位置为 1。复位操作结束后自动清零。说明：FIFO_RESET 清零后，再过两个系统时钟周期和同步延迟（两个卡时钟周期），FIFO 指针将会退出复位。(读/写)

CONTROLLER_RESET 要复位控制器，软件应将此位置为 1。两个 AHB 时钟周期和两个 cclk_in 时钟周期后此位自动清零。(读 / 写)

Register 9.2. CLKDIV_REG (0x0008)

CLK_DIVIDER3	CLK_DIVIDER2	CLK_DIVIDER1	CLK_DIVIDER0	
31 24	23 16	15 8	7 0	
0x000	0x000	0x000	0x000	Reset

CLK_DIVIDER3 时钟 divider-3 的值。时钟分频系数为 2*n，n=0 旁路分频器（分频系数为 1）。例如，值为 1 代表分频系数为 2*1 = 2，值为 0xFF 代表分频系数为 2*255 = 510，以此类推。在 MMC-Ver3.3-only 模式下，由于只支持一个时钟分频器，所以这些比特不执行。(读 / 写)

CLK_DIVIDER2 时钟 divider-2 的值。其他同上。(读 / 写)

CLK_DIVIDER1 时钟 divider-1 的值。其他同上。(读 / 写)

CLK_DIVIDER0 时钟 divider-0 的值。其他同上。(读 / 写)

Register 9.3. CLKSRC_REG (0x000C)

(reserved)	CLKSRC_REG	
31 4	3 0	
0x000000	0x0	Reset

CLKSRC_REG 时钟分频源可以支持 2 个 SD 卡。每个卡分配有两个比特。例如，bit[1:0] 分配给卡 0，bit[3:2] 分配给卡 1。卡 0 根据比特值将时钟分频器 [0:3] 的输出信号传输给 cclk_out[1:0] 管脚。

00：时钟分频器 0；

01：时钟分频器 1；

10：时钟分频器 2；

11：时钟分频器 3。

在 MMC-Ver3.3-only 模式下，只支持一个时钟分频器。cclk_out 时钟来自时钟分频器 0，并且此寄存器不执行。(读 / 写)

Register 9.4. CLKENA_REG (0x0010)

(reserved)	CCLK_ENABEL	
31 2	1 0	
0x00000	0x00000	Reset

CCLK_ENABEL 时钟使能控制可支持 2 个 SD 卡时钟和一个 MMC 卡时钟。

0：时钟禁能；

1：时钟使能。

在 MMC-Ver3.3-only 模式下，由于只有一个 cclk_out，因此只使用 cclk_enable[0]。（读 / 写）

Register 9.5. TMOUT_REG (0x0014)

DATA_TIMEOUT	RESPONSE_TIMEOUT	
31 8	7 0	
0x0FFFFFF	0x040	Reset

DATA_TIMEOUT 此位用于设置卡数据读取超时的值，还可以用来设置主机超时的定时器的值。只有当卡时钟停止后超时计数器才开始启动。此位的值以卡输出时钟数来表示，即被选取卡的 cclk_out。

说明：如果超时值是 100 ms 左右，则应该使用软件定时器。这种情况下，读数据超时中断应该被禁能。（读 / 写）

RESPONSE_TIMEOUT 此位用于设置响应超时的值，以卡输出时钟数来表示，即 cclk_out。（读 / 写）

Register 9.6. CTYPE_REG (0x0018)

(reserved)	CARD_WIDTH8	(reserved)	CARD_WIDTH4	
31 18	17 16	15 2	1 0	
0x00000	0x00000	0x00000	0x00000	Reset

CARD_WIDTH8 每个卡一个比特，表明卡是否处于 8-bit 模式。

0：非 8-bit 模式；

1：8-bit 模式。

Bit[17:16] 分别对应卡 [1:0]。（读 / 写）

CARD_WIDTH4 每个卡一个比特，表明卡处于 1-bit 模式还是 4-bit 模式。

0：1-bit 模式；

1：4-bit 模式。

Bit[1:0] 分别对应卡 [1:0]。只有 NUM_CARDS*2 个比特被执行。（读 / 写）

Register 9.7. BLKSIZ_REG (0x001C)

(reserved)	BLOCK_SIZE	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00200	Reset

BLOCK_SIZE 模块大小。（读 / 写）

Register 9.8. BYTCNT_REG (0x0020)

31 0	
0x000000200	Reset

BYTCNT_REG 表明传输的字节数。对于模块的传输，值应为模块大小的整数倍数。对于未定义字节长度的数据传输，字节计数应该设置为 0。当字节计数为 0 时，主机应当明确发送停止 / 终止指令来停止数据传输。（读 / 写）

Register 9.9. INTMASK_REG (0x0024)

(reserved)		SDIO_INT_MASK		INT_MASK		
31	18	17	16	15	0	
0x00000		0x00000		0x00000		Reset

SDIO_INT_MASK SDIO 中断的屏蔽位，每个卡一个比特。Bit[17:16] 分别对应卡 [1:0]。当被屏蔽时，SDIO 中断的检测被禁能。“0” 屏蔽中断，“1” 使能中断。在 MMC-Ver3.3-only 模式下，这些比特始终为 0。(读 / 写)

INT_MASK 这些比特用于屏蔽不想要的中断。“0” 屏蔽中断，“1” 使能中断。(读 / 写)

Bit 15 (EBE): 结束位错误，读 / 写 (无 CRC)；

Bit 14 (ACD): 自动指令结束；

Bit 13 (SBE/BCI): 启动 Bit Error/Busy Clear 中断；

Bit 12 (HLE): 硬件锁定写入错误

Bit 11 (FRUN): FIFO underrun/overrun 错误；

Bit 10 (HTO): 主机填充数据超时 / Volt_switch_int；

Bit 9 (DRTO): 数据读取超时；

Bit 8 (RTO): 响应超时；

Bit 7 (DCRC): 数据 CRC 错误；

Bit 6 (RCRC): 响应 CRC 错误；

Bit 5 (RXDR): 接收 FIFO 数据请求；

Bit 4 (TXDR): 发送 FIFO 数据请求；

Bit 3 (DTO): 数据传输结束；

Bit 2 (CD): 指令执行完毕；

Bit 1 (RE): 响应错误；

Bit 0 (CD): 卡检测。

Register 9.10. CMDARG_REG (0x0028)

31	0	
0x000000000		Reset

CMDARG_REG 传递给卡的命令参数。(读 / 写)

Register 9.11. CMD_REG (0x002C)

START_CMD	(reserved)	USE_HOLE	(reserved)	(reserved)	(reserved)	(reserved)	(reserved)	CCS_EXPECTED	READ_CEATA_DEVICE	UPDATE_CLOCK_REGISTERS_ONLY	CARD_NUMBER	SEND_INITIALIZATION	STOP_ABORT_CMD	WAIT_PRVDATA_COMPLETE	SEND_AUTO_STOP	TRANSFER_MODE	READ/WRITE	DATA_EXPECTED	CHECK_RESPONSE_CRC	RESPONSE_LENGTH	RESPONSE_EXPECT	CMD_INDEX	
31	30	29	28	27	26	25	24	23	22	21	20 16	15	14	13	12	11	10	9	8	7	6	5 0	
0	0	1	0	0	0	0	0	0	0	0	0x00	0	0	0	0	0	0	0	0	0	0	0x00	Reset

START_CMD 开始发送命令。一旦 CIU 响应命令，此位自动清零。当此位置为 1 时，主机不应尝试写任何命令寄存器。如果尝试写寄存器，原始中断寄存器的硬件锁定错误将会被置为 1。一旦命令被发送并且接收到 SD_MMC_CEATA 卡的响应，则原始中断寄存器的 Command Done 比特将会被置为 1。（读 / 写）

USE_HOLE Use Hold 寄存器。（读 / 写）

0：发送给卡的 CMD 和 DATA 旁路 Hold 寄存器；

1：发送给卡的 CMD 和 DATA 经过 Hold 寄存器。

CCS_EXPECTED 预期命令完成信号 (CCS) 的配置。（读 / 写）

0：CE-ATA 设备的中断不使能（ATA 控制寄存器中 nIEN = 1）；或者指令不等待设备的 CCS 信号。

1：CE-ATA 设备的中断使能 (nIEN = 0)，并且 RW_BLK 指令等待 CE-ATA 设备的 CCS 信号。

如果指令等待 CE-ATA 设备的 CCS 信号，软件应该将此位置为 1。SD/MMC 置位 RINTSTS 寄存器里的 Data Transfer Over (DTO) 位并且如果 DTO 中断没有被屏蔽，会产生对主机的中断。

READ_CEATA_DEVICE 读操作标志位。（读 / 写）

0：主机不进行对于 CE-ATA 设备的读操作 (RW_REG 或 RW_BLK)；

1：主机进行对于 CE-ATA 设备的读操作 (RW_REG 或 RW_BLK)。

软件应将此位置为 1 来表明 CE-ATA 设备正在被访问用于读传输。此位用于在执行 CE-ATA 读传输时禁能读数据超时指示。I/O 传输延迟的最大值至少为 10 秒。SD/MMC 在等待来自 CE-ATA 设备的数据时不应指示读取数据超时。（读 / 写）

Register 9.12. CMD_REG (continued) (0x002C)

START_CMD	(reserved)	USE_HOLE	(reserved)	(reserved)	(reserved)	(reserved)	(reserved)	CCS_EXPECTED	READ_CEATA_DEVICE	UPDATE_CLOCK_REGISTERS_ONLY	CARD_NUMBER	SEND_INITIALIZATION	STOP_ABORT_CMD	WAIT_PRVDATA_COMPLETE	SEND_AUTO_STOP	TRANSFER_MODE	READ/WRITE	DATA_EXPECTED	CHECK_RESPONSE_CRC	RESPONSE_LENGTH	RESPONSE_EXPECT	CMD_INDEX	
31	30	29	28	27	26	25	24	23	22	21	20 16	15	14	13	12	11	10	9	8	7	6	5 0	
0	0	1	0	0	0	0	0	0	0	0	0x00	0	0	0	0	0	0	0	0	0	0	0x00	Reset

UPDATE_CLOCK_REGISTERS_ONLY (R/W)

0：正常指令序列；

1：不发送指令，仅更新时钟寄存器的值到卡时钟域内。以下寄存器的值被传输到卡时钟域内：CLKDIV、CLRSRC 和 CLKENA（分频、时钟源和时钟使能）。这是为了改变时钟频率或停止时钟，而不必发送命令给卡。

在正常指令序列下，当 update_clock_registers_only = 0，以下控制寄存器从 BIU 传输到 CIU：CMD、CMDARG、TMOUT、CTYPE、BLKSIZ 和 BYTCNT。CIU 为新的指令序列使用新的寄存器的值。当此位置为 1 时，由于没有指令被发送给 SD_MMC_CEATA 卡，所以没有 Command Done 中断。（读 / 写）

CARD_NUMBER 使用中的卡号，表示正在访问的卡的物理插槽编号。在 MMC-Ver3.3-only 模式下，最多支持 2 张卡。在仅 SD 模式下，支持 2 张卡。（读 / 写）

SEND_INITIALIZATION（读 / 写）

0：在发送指令前不发送初始化序列（80 个周期）；

1：在发送指令前发送初始化序列。

上电后，发送任何命令到卡之前，必须向卡发送 80 个时钟进行初始化。在向卡发送第一个命令时应该将此位置为 1，以便控制器在向卡发送命令之前初始化时钟。

STOP_ABORT_CMD（读 / 写）

0：停止或中止命令，停止命令和中止命令都不会停止当前的数据传输。如果中止发送到当前选择的功能号或不在数据传输模式，则该位应设置为 0；

1：停止或中止命令，用于停止当前的数据传输。当开放式预定义数据传输正在进行时，并且主机发出停止或中止命令以停止数据传输时，应将此位置为 1，以使 CIU 的命令/数据状态机可以正确返回到空闲状态。

Register 9.13. CMD_REG (continued) (0x002C)

START_CMD	(reserved)	USE_HOLE	(reserved)	(reserved)	(reserved)	(reserved)	(reserved)	CCS_EXPECTED	READ_CEATA_DEVICE	UPDATE_CLOCK_REGISTERS_ONLY	CARD_NUMBER	SEND_INITIALIZATION	STOP_ABORT_CMD	WAIT_PRVDATA_COMPLETE	SEND_AUTO_STOP	TRANSFER_MODE	READ/WRITE	DATA_EXPECTED	CHECK_RESPONSE_CRC	RESPONSE_LENGTH	RESPONSE_EXPECT	CMD_INDEX	
31	30	29	28	27	26	25	24	23	22	21	20 16	15	14	13	12	11	10	9	8	7	6	5 0	
0	0	1	0	0	0	0	0	0	0	0	0x00	0	0	0	0	0	0	0	0	0	0	0x00	Reset

WAIT_PRVDATA_COMPLETE （读 / 写）

0：即使以前的数据传输尚未完成，也立即发送命令；

1：等待前面的数据传输完成后再发送指令。

wait_prvdata_complete = 0 选项通常用来在数据传输时询问卡的状态或停止当前的数据传输。card_number 应该与上一个指令相同。

SEND_AUTO_STOP （读 / 写）

0：在数据传输结束时不发送停止命令；

1：在数据传送结束时发送停止命令。

TRANSFER_MODE （读 / 写）

0：模块数据传输指令；

1: 流数据传输指令。如果不等待数据则为无关项。

READ/WRITE （读 / 写）

0：读卡；

1：写卡。

如果不等待数据则为无关项。

DATA_EXPECTED （读 / 写）

0：不等待数据传输；

1：等待数据传输。

CHECK_RESPONSE_CRC （读 / 写）

0：不检查；

1：检查响应 CRC。

有些指令响应不会返回有效的 CRC 位。软件应禁能对于这些指令的 CRC 检查以便禁能控制器进行 CRC 检查。

RESPONSE_LENGTH （读 / 写）

0：等待卡的短响应；

1：等待卡的长响应。

RESPONSE_EXPECT （读 / 写）

0：不等待卡的响应；

1：等待卡的响应。

CMD_INDEX 指令指数。（读 / 写）

Register 9.14. RESP0_REG (0x0030)

31	0	
0x000000000		Reset

RESP0_REG 响应的 bit[31:0]。（只读）

Register 9.15. RESP1_REG (0x0034)

31	0	
0x000000000		Reset

RESP1_REG 长响应的 bit[63:32]。（只读）

Register 9.16. RESP2_REG (0x0038)

31	0	
0x000000000		Reset

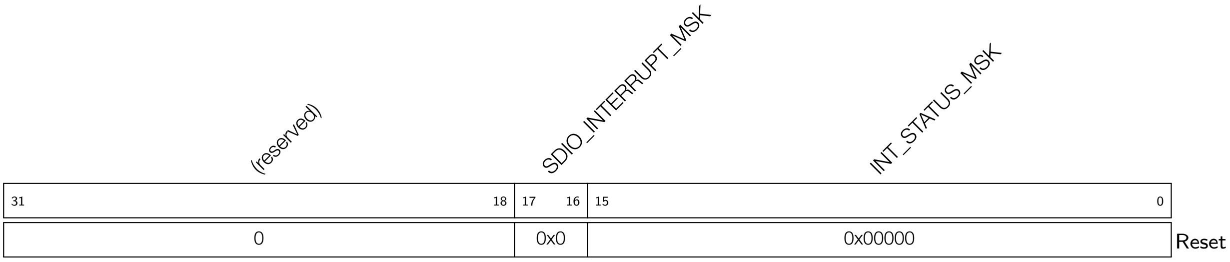
RESP2_REG 长响应的 bit[95:64]。（只读）

Register 9.17. RESP3_REG (0x003C)

31	0	
0x000000000		Reset

RESP3_REG 长响应的 bit[127:96]。（只读）

Register 9.18. MINTSTS_REG (0x0040)



SDIO_INTERRUPT_MSK SDIO 中断的屏蔽位，每个卡一个比特。Bit[17:16] 分别对应卡 1 和卡 0。只有对应的 sdio_int_mask 位被置为 1 时，SDIO 中断才会使能（置位屏蔽位使能中断）。(只读)

INT_STATUS_MSK 只有当中断屏蔽寄存器中的对应位被置为 1 时，中断才会使能。(只读)

Bit 15 (EBE): 结束位错误，读 / 写（无 CRC）;

Bit 14 (ACD): 自动指令结束;

Bit 13 (SBE/BCI): 启动 Bit Error/Busy Clear 中断;

Bit 12 (HLE): 硬件锁定写入错误

Bit 11 (FRUN): FIFO underrun/overrun 错误;

Bit 10 (HTO): 主机填充数据超时;

Bit 9 (DRTO): 数据读取超时;

Bit 8 (RTO): 响应超时;

Bit 7 (DCRC): 数据 CRC 错误;

Bit 6 (RCRC): 响应 CRC 错误;

Bit 5 (RXDR): 接收 FIFO 数据请求;

Bit 4 (TXDR): 发送 FIFO 数据请求;

Bit 3 (DTO): 数据传输结束;

Bit 2 (CD): 指令执行完毕;

Bit 1 (RE): 响应错误;

Bit 0 (CD): 卡检测。

Register 9.19. RINTSTS_REG (0x0044)

(reserved)	SDIO_INTERRUPT_RAW	INT_STATUS_RAW
31 16	17 16	15 0
0x00000	0x0	0x00000

Reset

SDIO_INTERRUPT_RAW 来自 SDIO 卡的中断，一个卡一个中断。Bit[17:16] 分别对应卡 1 和卡 0。置位某比特就把相应的中断位清零，写 0 无效。(读 / 写)

0：没有来自卡的 SDIO 中断；

1：有来自卡的 SDIO 中断。

在 MMC-Ver3.3-only 模式下，这些比特始终为 0。这些中断位是原始的中断，而不管中断屏蔽状态。(读 / 写)

INT_STATUS_RAW 置位某比特就把相应的中断位清零，写 0 无效。无论中断屏蔽状态如何，这些中断位都会被记录。(R/W)

Bit 15 (EBE): 结束位错误，读 / 写（无 CRC）；

Bit 14 (ACD): 自动指令结束；

Bit 13 (SBE/BCI): 启动 Bit Error/Busy Clear 中断；

Bit 12 (HLE): 硬件锁定写入错误

Bit 11 (FRUN): FIFO underrun/overrun 错误；

Bit 10 (HTO): 主机填充数据超时；

Bit 9 (DRTO): 数据读取超时；

Bit 8 (RTO): 响应超时；

Bit 7 (DCRC): 数据 CRC 错误；

Bit 6 (RCRC): 响应 CRC 错误；

Bit 5 (RXDR): 接收 FIFO 数据请求；

Bit 4 (TXDR): 发送 FIFO 数据请求；

Bit 3 (DTO): 数据传输结束；

Bit 2 (CD): 指令执行完毕；

Bit 1 (RE): 响应错误；

Bit 0 (CD): 卡检测。

Register 9.20. STATUS_REG (0x0048)

(reserved)	(reserved)	FIFO_COUNT	RESPONSE_INDEX	DATA_STATE_MC_BUSY	DATA_BUSY	DATA_3_STATUS	COMMAND_FSM_STATES	FIFO_FULL	FIFO_EMPTY	FIFO_TX_WATERMARK	FIFO_RX_WATERMARK	
31	30	29 17	16 11	10	9	8	7 4	3	2	1	0	
0	0	0x000	0x00	1	1	1	0x01	0	1	1	0	Reset

FIFO_COUNT FIFO 计数位，FIFO 中被填充的地址的数量。（只读）

RESPONSE_INDEX 前一个响应的指数，包括内核发送的任何自动停止的响应。（只读）

DATA_STATE_MC_BUSY 数据发送或接收状态机忙。（只读）

DATA_BUSY 原始选择的 card_data[0] 的取反版。（只读）

0：卡数据不忙；

1：卡数据忙。

DATA_3_STATUS 原始选择的 card_data[3]，检查卡是否存在。（只读）

0：卡不存在；

1：卡存在。

COMMAND_FSM_STATES 指令 FSM 状态。（只读）

0: 空闲；

1: 发送初始序列；

2: 发送指令开始位；

3: 发送指令发送位；

4: 发送指令指数和参数；

5: 发送指令 CRC7；

6: 发送指令结束位；

7: 接收响应开始位；

8: 接收响应 IRQ 响应；

9: 接收响应发送位；

10: 接收响应指令指数；

11: 接收响应数据；

12: 接收响应 CRC7；

13: 接收响应结束位；

14: 指令路径等待 NCC；

15: 等待，指令-响应回转。

FIFO_FULL FIFO 为满的状态。（只读）

FIFO_EMPTY FIFO 为空的状态。（只读）

FIFO_TX_WATERMARK FIFO 达到发送阈值，不是数据传输的必要条件。（只读）

FIFO_RX_WATERMARK FIFO 达到接收阈值，不是数据传输的必要条件。（只读）

Register 9.21. FIFOTH_REG (0x004C)

(reserved)	DMA_MULTIPLE_TRANSACTION_SIZE	(reserved)	RX_WMARK	(reserved)	TX_WMARK	
31	30 28	27	26 16	15 12	11 0	
0	0x0	0	x x x x x x x x x x x	0 0 0 0	0x0000	Reset

DMA_MULTIPLE_TRANSACTION_SIZE 多次传输的突发大小，配置的值应与 DMA 控制器多个传输大小 SRC/DEST_MSIZE 相同。000: 1 字节传输；001: 4 字节传输；010: 8 字节传输；011: 16 字节传输；100: 32 字节传输；101: 64 字节传输；110: 128 字节传输；111: 256 字节传输。（读 / 写）

RX_WMARK 当接收数据给卡时 FIFO 的阈值。当 FIFO 数据计数大于该数值 (FIFO_RX_WATERMARK) 时，DMA/FIFO 请求被提出。在数据包结束期间，无论阈值大小如何，都会生成请求，以完成剩余的数据传输。在非 DMA 模式下，当接收 FIFO 阈值 (RXDR) 中断使能时，则会产生中断，而不是 DMA 请求。如果阈值大于任何剩余数据，则不产生中断。当主机看见数据发送结束中断时，主机应该读取剩下的数据。在 DMA 模式下，在数据包结束时，即使剩余的字节数少于阈值，DMA 请求也会在数据传输结束中断设置之前进行单次传输以清除所有剩余的字节。（读 / 写）

TX_WMARK 当发送数据给卡时 FIFO 的阈值。当 FIFO 数据计数小于等于该数值 (FIFO_TX_WATERMARK) 时，DMA/FIFO 请求被提出。如果使能中断，则中断发生。在数据包结束期间，无论阈值大小如何，都会生成请求。在非 DMA 模式下，当发送 FIFO 阈值 (TXDR) 中断使能时，则会产生中断，而不是 DMA 请求。在数据包结束期间，在最后一个中断时，主机负责仅用所需的剩余字节填充 FIFO（不是在 FIFO 满之前或在 CIU 完成数据传输之后，因为 FIFO 可能不为空）。在 DMA 模式下，在数据包结束时，如果最后一次传输比突发传输小，DMA 控制器将执行单个周期，直到传输所需的字节。（读 / 写）

Register 9.22. CDETECT_REG (0x0050)

(reserved)	CARD_DETECT_N	
31 2	1 0	
0x0	0x0	Reset

CARD_DETECT_N card_detect_n 输入端口（每个卡一个比特）的值。0 代表卡存在。只有 NUM_CARDS 的相应位被执行。（只读）

Register 9.23. WRTPRT_REG (0x0054)

(reserved)	WRITE_PROTECT	
31 2	1 0	
0x0	0x0	Reset

WRITE_PROTECT card_write_prt 输入端口（每个卡一个比特）的值。1 表示写保护。只有 NUM_CARDS 的相应位被执行。（只读）

Register 9.24. TCBCNT_REG (0x005C)

31 0	
0x000000000	Reset

TCBCNT_REG CIU 传输给卡的字节数。（只读）

Register 9.25. TBBCNT_REG (0x0060)

31 0	
0x000000000	Reset

TBBCNT_REG 主机/DMA 和 BIU FIFO 之间传输的字节数。（只读）

Register 9.26. DEBNCE_REG (0x0064)

(reserved)	DEBOUNCE_COUNT	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0000000	Reset

DEBOUNCE_COUNT 抖动消除滤波器逻辑使用的主机时钟 (clk) 数。典型的去抖动时间为 5 ~ 25 ms，防止插卡或移除卡的时候的不稳定性。（读 / 写）

Register 9.27. USRID_REG (0x0068)

31 0	
0x000000000	Reset

USRID_REG 用户识别寄存器，其值由用户设置。此寄存器也可以作为用户的暂存器寄存器使用。（读／写）

Register 9.28. RST_N_REG (0x0078)

(reserved)	RST_CARD_RESET	
31 2	1 0	
0	0x1	Reset

RST_CARD_RESET 硬件复位。1：工作模式；0：复位。这些比特让卡进入前空闲状态，这将要求它们被重新初始化。CARD_RESET[0] 应被置为 1’b0 来复位卡 0。CARD_RESET[1] 应被置为 1’b0 来复位卡 1。被执行的比特数量受限于 NUM_CARDS 的值。（读／写）

Register 9.29. BMOD_REG (0x0080)

(reserved)	BMOD_PBL	BMOD_DE	(reserved)	BMOD_FB	BMOD_SWR	
31 11	10 8	7	6 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0	0x00	0	0	Reset

BMOD_PBL 可编程突发长度。这些位指示在一个 IDMAC 传输中要执行的最大节拍数。IDMAC 每次在主机总线上开始突发传输时将总是尝试在 PBL 中指定的突发值。允许的值为 1、4、8、16、32、64、128 和 256。该值是 FIFOTH 寄存器的 MSIZE 的镜像。要修改此值，将所需的值写入 FIFOTH 寄存器。这是一个编码值，如下所示：000: 1 字节传输；001: 4 字节传输；010: 8 字节传输；011: 16 字节传输；100: 32 字节传输；101: 64 字节传输；110: 128 字节传输；111: 256 字节传输。

PBL 是只读值，并且仅适用于数据访问，不适用于链表访问。（读／写）

BMOD_DE IDMAC 使能位。置位后 IDMAC 使能。（读／写）

BMOD_FB 固定突发。控制 AHB 主接口是否执行固定突发传输。置位时，AHB 将在正常突发传输开始期间仅使用 SINGLE、INCR4、INCR8 或 INCR16。当复位时，AHB 将使用 SINGLE 和 INCR 突发传输操作。（读／写）

BMOD_SWR 软件复位。当置位时，DMA 控制器复位所有内部寄存器。一个时钟周期后自动清零。（读／写）

Register 9.30. PLDMND_REG (0x0080)

31	0	
0x000000000		Reset

PLDMND_REG 轮询需求。如果没有设置链表的 OWN 位，则 FSM 进入挂起状态。主机需要对这个寄存器写入任意值，以使 IDMAC FSM 恢复正常链表读取操作。这是一个只写寄存器，PD 位是只写位。（只写）

Register 9.31. DBADDR_REG (0x0088)

31	0	
0x000000000		Reset

DBADDR_REG 链表列表的开始。包含第一个链表的基址。最低效位 (LSB bit) [1:0] 被忽略，并由 IDMAC 内部全部取为零，因此这些 LSB 位可被视为只读。（读 / 写）

Register 9.32. IDSTS_REG (0x008C)

(reserved)	IDSTS_FSM	IDSTS_FBE_CODE	IDSTS_AIS	IDSTS_NIS	(reserved)	IDSTS_CES	IDSTS_DU	(reserved)	IDSTS_FBE	IDSTS_RI	IDSTS_TI	
31 17	16 13	12 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	0x00	0x0	0	0	0 0	0	0	0	0	0	0	Reset

IDSTS_FSM DMAC FSM 当前状态。（只读）

0: DMA_IDLE; 1: DMA_SUSPEND; 2: DESC_RD; 3: DESC_CHK; 4: DMA_RD_REQ_WAIT
5: DMA_WR_REQ_WAIT; 6: DMA_RD; 7: DMA_WR; 8: DESC_CLOSE

IDSTS_FBE_CODE 致命总线错误代码。表明导致总线错误的错误类型。仅当设置致命总线错误位 IDSTS[2] 被置位时有效。此字段不生成中断。（只读）

3b001：传输期间接收到主机中止；
3b010：接收期间接收到主机中止；
其他：保留。

IDSTS_AIS 异常中断汇总。以下各项的逻辑或：IDSTS[2]：致命总线中断，IDSTS[4]：DU 位中断。只有未屏蔽的位影响该位。这是一个粘滞位，必须在每次清零引起 AIS 置 1 的相应位时清零。写 1 清零该位。（读 / 写）

IDSTS_NIS 正常中断汇总。以下各项的逻辑或：IDSTS[0]：发送中断，IDSTS[1]：接收中断。只有未屏蔽的位影响该位。这是一个粘滞位，必须在每次清零引起 NIS 置 1 的相应位时清零。写 1 清零该位。（读 / 写）

IDSTS_CES 卡错误汇总。指示发送/接收卡的传输状态，也出现在 RINTSTS 中。表示以下位的逻辑或：EBE：结束位错误，RTO：响应超时/引导确认超时，RCRC：响应 CRC，SBE：启动位错误，DRTO：数据读取超时/ BDS 超时，DCRC：用于接收的数据 CRC，RE：响应错误。

写 1 清零该位。IDMAC 的中止条件取决于此 CES 位的配置。如果 CES 位被使能，则 IDMAC 在响应错误时中止。（读 / 写）

IDSTS_DU 链表不可用中断。当链表由于 OWN 位 = 0 (DES0 [31] = 0) 而不可用时，该位置 1。写 1 清零该位。（读 / 写）

IDSTS_FBE 致命总线错误中断。表示发生总线错误 (IDSTS[12:10])。当该位置 1 时，DMA 禁止所有总线访问。写 1 清零该位。（读 / 写）

IDSTS_RI 接收中断。表示链表的数据接收完成。写 1 清零该位。（读 / 写）

IDSTS_TI 发送中断。表示链表的数据发送完成。写 1 清零该位。（读 / 写）

Register 9.33. IDINTEN_REG (0x0090)

(reserved)	IDINTEN_AI	IDINTEN_NI	(reserved)	IDINTEN_CES	IDINTEN_DU	(reserved)	IDINTEN_FBE	IDINTEN_RI	IDINTEN_TI	
31 10	9	8	7 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0	0	0	0	0	0	0	Reset

IDINTEN_AI 异常中断汇总使能位。(读 / 写)
置 1 时，使能异常中断。该位使能以下位：IDINTEN [2]：致命总线错误中断；IDINTEN [4]：DU 中断。

IDINTEN_NI 中断汇总使能位。(读 / 写)
置 1 时，使能正常中断。重置时，禁能正常中断。该位使能以下位：
IDINTEN[0]：发送中断；
IDINTEN[1]：接收中断。

IDINTEN_CES 卡错误汇总中断使能位。置 1 时使能卡中断汇总。(读 / 写)

IDINTEN_DU 链表不可用中断。当与异常中断汇总使能位一起设置时，将使能 DU 中断。(读 / 写)

IDINTEN_FBE 致命总线错误使能位。当与异常中断汇总使能位一起设置时，将使能致命总线错误中断。复位时，致命总线错误使能中断被禁能。(读 / 写)

IDINTEN_RI 接收中断使能位。当与正常中断汇总使能位一起设置时，将使能接收中断。复位时，接收中断被禁能。(读 / 写)

IDINTEN_TI 发送中断使能位。当与正常中断汇总使能位一起设置时，将使能发送中断。复位时，发送中断被禁能。(读/写)

Register 9.34. DSCADDR_REG (0x0094)

31	0	
0x000000000		Reset

DSCADDR_REG 主机链表地址指针，在操作期间由 IDMAC 更新，并在复位时清零。该寄存器指向由 IDMAC 读取的当前链表的起始地址。(只读)

Register 9.35. BUFADDR_REG (0x0098)

31	0	
0x000000000		Reset

BUFADDR_REG 主机缓冲区地址指针，在操作期间由 IDMAC 更新，并在复位时清零。该寄存器指向由 IDMAC 访问的当前数据缓冲区地址。(只读)

Register 9.36. CLK_EDGE_SEL (0x0800)

(reserved)	CCLKIN_EDGE_N	CCLKIN_EDGE_L	CCLKIN_EDGE_H	CCLKIN_EDGE_SLF_SEL	CCLKIN_EDGE_SAM_SEL	CCLKIN_EDGE_DRV_SEL	
31 21	20 17	16 13	12 9	8 6	5 3	2 0	
0x000	0x1	0x0	0x1	0x0	0x0	0x0	Reset

CCLKIN_EDGE_N 值与 CCLKIN_EDGE_L 相同。（读/写）

CCLKIN_EDGE_L 分频时钟的低电平，值应比 CCLKIN_EDGE_H 大。（读/写）

CCLKIN_EDGE_H 分频时钟的高电平，值应比 CCLKIN_EDGE_L 小。（读/写）

CCLKIN_EDGE_SLF_SEL 用于选择内部时钟信号的相位，90 度相位、180 度相位或 270 度相位。（读/写）

CCLKIN_EDGE_SAM_SEL 用于选择输入时钟信号的相位，90 度相位、180 度相位或 270 度相位。（读/写）

CCLKIN_EDGE_DRV_SEL 用于选择输出时钟信号的相位，90 度相位、180 度相位或 270 度相位。（读/写）

10 以太网 (MAC)

10.1 概述

Ethernet 主要特性

借助外部以太网物理层 (Ethernet PHY)，ESP32 可以通过以太网介质访问控制 (Ethernet MAC) 按照 IEEE 802.3 标准发送和接收数据。详见图 10-1。以太网是当前应用最普遍的局域网 (LAN) 和广域网 (WAN) 进行数据传播的网络协议。

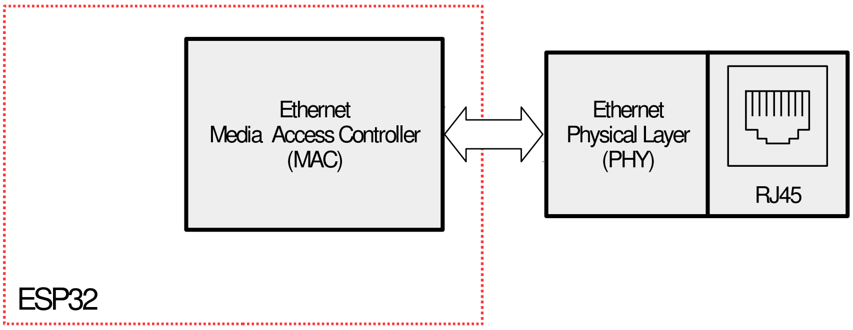


图 10-1. Ethernet MAC 功能概述

ESP32 以太网 MAC 符合以下标准：

- IEEE 802.3-2002，用于以太网 MAC。
- 符合 IEEE 802.3 规范工业标准接口：介质独立接口 (MII) 和简化介质独立接口 (RMII)。

MAC 层特性

- 支持外部 PHY 接口实现 10/100 Mbit/s 数据传输速率
- 可通过符合 IEEE802.3 的 MII 接口和 RMII 接口与外部快速以太网 PHY 进行通信
- 支持全双工和半双工模式
  - 支持适用于半双工模式的 CSMA/CD 协议
  - 支持适用于全双工模式的 IEEE 802.3x 流量控制
  - 全双工模式时可以将接收的暂停控制帧转发到用户应用程序
  - 半双工模式时提供背压流量控制
  - 全双工操作中如果流量控制输入信号消失，将自动发送暂停时间为零的暂停帧
- 报头和帧起始数据 (SFD) 在发送路径中插入、在接收路径中删除
- 可逐帧控制 CRC 和 pad 自动生成
- 如果数据为达到最小帧长度，则自动添加 pad
- 可编程帧长度，支持高达 16 KB 的巨型帧
- 可编程帧间隔 (IFG)（40-96 位时间，以 8 为步长）

- 支持多种灵活的地址过滤模式:
  - 高达 8 个 48 位完美地址过滤器，对每个字节进行掩码操作
  - 高达 8 个 48 位 SA 地址比较检查，对每个字节进行掩码操作
  - 可传送所有多播地址帧
  - 支持混合模式，因此可传送所有帧，无需为网络监视进行过滤
  - 传送所有传入数据包时（每次过滤时）均附有一份状态报告
- 为发送和接收数据包分别返回 32 位状态
- 为应用程序提供单独的发送、接收和控制接口
- 使用 MDIO 接口配置和管理 PHY 设备
- 在接收功能中支持对接收到的由以太网帧封装的 IPv4 和 TCP 数据包进行校验和卸载
- 在接收功能中支持检查 IPv4 头校验和以及在 IPv4/IPv6 数据包中封装的 TCP、UDP 或 ICMP 校验和
- 两组 FIFO：一个具有可编程阈值功能的 2 KB 发送 FIFO 和一个具有可配置阈值（默认为 64 个字节）功能的 2 KB 接收 FIFO
- 接收 FIFO 进行多帧存储时，在 EOF 传输后，通过向接收 FIFO 插入接收状态矢量，从而使得接收 FIFO 无需存储这些帧的接收状态
- 在存储转发模式下，可以在接收时过滤所有的错误帧，但不将这些错误帧转发给应用程序
- 可以转发过小的好帧
- 为接收 FIFO 中由于溢出丢失或损坏的帧生成脉冲，借此支持数据统计
- 向 MAC 内核发送数据时支持存储转发机制
- 发送时处理冲突帧的自动重新发送（符合一定条件，详见 10.2.1.2）
- 丢弃延迟冲突、过度冲突、过度延迟和下溢条件下的帧
- 通过软件控制刷新 Tx FIFO
- 计算 IPv4 头校验和与 TCP、UDP 或 ICMP 校验和并将其插入在存储转发模式下发送的帧中

Ethernet 结构框图

Ethernet 结构框图如图 10-2 所示。

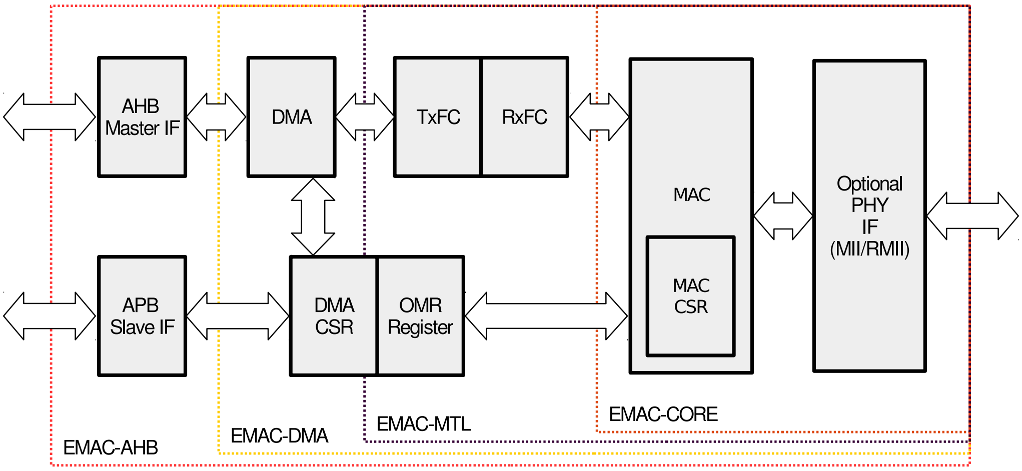


图 10-2. Ethernet 功能框图

Ethernet MAC 层主要包括 EMAC_CORE，EMAC_MTL (MAC Transition Layer)，EMAC_DMA (Direct Memory Access) 三层以及 MAC 层的配置寄存器模块，它们分别有 Rx 和 Tx 两个方向，在芯片内部通过 AHB 和 APB 总线和系统相连接，在芯片外部通过 MII 和 RMII 接口和外部的 PHY 进行通信，最终实现以太网的功能。

10.2 EMAC_CORE

MAC 支持许多兼容 PHY 芯片的接口。复位后 PHY 接口只能选一次。MAC 使用 MAC 发送接口 (MTI)，MAC 接收接口 (MRI) 和 MAC 控制接口 (MCI) 与应用侧（DMA 侧）进行通信。

10.2.1 传输操作

当 MTL 应用程序推送数据并且 SOF（帧起始）信号拉高时，启动发送操作。当检测到 SOF 信号时，MAC 接收数据并开始发送到 RMII 或 MII。应用程序启动传输之后将帧数据发送到 RMII 或 MII 所需的时间取决于延迟因素，如 IFG 延迟，发送报头或 SFD（起始帧分隔符）的时间以及半双工的任何回退延迟模式。在此之前，通过拉低数据就绪信号，MAC 不接收从 MTL 接收到的数据。

在 EOF（帧结束）发送到 MAC 之后，MAC 完成正常传输，并将传输状态 (Transmit Status) 提供给 MTL。如果在传输过程中（在半双工模式下）发生正常的冲突，则 MAC 使 MTL 的发送状态有效。然后它接受并删除所有剩下的数据，直到接收到下一个 SOF。在传输状态中检测到 MAC 发出的重试请求时，MTL 模块应该从 SOF 开始重传相同的帧。

如果 MTL 不能在传输期间连续提供数据，则 MAC 会发出下溢状态。在从 MTL 正常传输帧的过程中，如果 MAC 接收到一个 SOF 而没有得到前一帧的 EOF，则忽略该 SOF，并将该新帧视为前一帧的延续。

10.2.1.1 发送流量控制

在全双工模式下，当发送流量控制使能位（Flow Control Register 中的 TFE/Transmit Flow Control 位）置 1 时，MAC 将生成暂停帧并根据需要发送暂停帧。暂停帧与计算出的 CRC 附加在一起并发送。可以通过两种方式启动暂停帧的生成。

当应用程序将 Flow Control Register 中的 FCB (Flow Control Busy) 位置 1 或接收 FIFO 已满时，将发送暂停帧。

- 如果应用程序已通过将 Flow Control Register 寄存器中的 FCB 位置 1 来请求流量控制，则 MAC 将生成并发送单个暂停帧。生成的帧中的暂停时间值为 Flow Control Register 中编程的暂停时间值。要在先前发送的暂停帧中指定的时间之前延长或结束暂停时间，应用程序必须以适当的值编程暂停时间值（Flow Control Register 中的 PT/Pause Time），然后请求另一次暂停帧发送。
- 如果接收 FIFO 填满时应用程序已请求流量控制，则 MAC 将生成并发送暂停帧。生成的帧中的暂停时间值为 Flow Control Register 中编程的暂停时间值。如果在此暂停时间结束前，接收 FIFO 在可配置的时隙数（Flow Control Register 中的 PLT (Pause Low Threshold) 位）期间保持填满状态，将发送第二个暂停帧。只要接收 FIFO 保持填满状态，该过程将一直重复下去。如果在采样时间之前不再满足此条件，MAC 将发送一个暂停时间为零的暂停帧，向远程端表明接收缓冲区已准备好接收新数据帧。

10.2.1.2 冲突期间的重新发送

在半双工模式下，向 MAC 传输帧时，可能在 MAC 线接口上发生冲突事件。MAC 甚至会在接收到帧结束之前就给出状态来指示重试。然后将使能重新发送并再次将帧从 FIFO 中弹出。当超过 96 个字节弹向 MAC 内核后，FIFO 控制器将释放该空间，使 DMA 可推入更多数据。这意味着超过阈值后或 MAC 内核指示延迟冲突事件时，无法重新发送。

由于碰撞，Tx FIFO 下溢，载波丢失，jabber 超时，无载波，过度延迟或迟冲突，MAC 发送器都可能会中止帧的传输。当帧传输由于冲突而中止时，MAC 请求重发帧。

10.2.2 接收操作

当 MAC 在 RMII 或 MII 上检测到 SFD 时启动接收操作。在继续处理帧之前，MAC 剥去前导码和 SFD。检查报头字段的过滤和用于验证帧的 CRC 的 FCS 字段。接收的帧存储在缓冲器中，直到执行地址过滤。如果帧未通过地址过滤，则帧丢弃在 MAC 中。

MAC 接收的帧将推入 Rx FIFO。此 FIFO 的状态一旦超过配置的接收阈值（Operation Mode 寄存器中的 Receive Threshold Control/RTC 位），并通知给 DMA，这样 DMA 可向 AHB 接口发起预配置的突发传输。

在默认直通模式下，当 FIFO 接收到 64 个字节（通过 Operation Mode 寄存器中的 Receive Threshold Control/RTC 位配置）或完整的数据包时，数据将弹出，其可用性将通知给 DMA。DMA 向 AHB 接口发起传输后，数据传输将从 FIFO 持续进行，直到传输完整个数据包。完成帧 EOF 的传输后，状态字将弹出并发送到 DMA 控制器。

在 Rx FIFO 存储转发模式（通过 Operation Mode 寄存器中的 Receive Store and Forward/RSF 位配置），这样只会读出有效帧并将其转发到应用程序。在直通模式下，某些错误帧不会被丢弃，因为在帧结束时才接收到错误状态，而此时帧数据已从 FIFO 读出。

10.2.2.1 接收协议

接收模块接收到包之后，去除接收的帧的报头和 SFD。检测到 SFD 后，MAC 开始向接收 FIFO 发送以太网帧数据，从 SFD 后面的第一个字节（目标地址）开始发送。

如果接收的帧长度/类型字段小于 0x600 并且为 MAC 使能了自动去除 CRC/pad 选项，则 MAC 将向接收 FIFO 发送帧数据（数据量不超过长度/类型字段中指定的数量），然后开始丢弃字节（包括 FCS 字段）。如果长度/类型字段大于或等于 0x600，则不管配置的自动 CRC 去除选项的值如何，MAC 都会向 Rx FIFO 发送所有接收到的以太网帧数据。默认情况下，使能 MAC 看门狗定时器，即，超过 2048 个字节（DA+SA+LT+ 数据 +pad+FCS）帧会被切断。可通过对 MAC 配置寄存器中的看门狗禁止 (WD/Watchdog Disable) 位编程来禁止此功能。但是，即使禁止看门狗定时器，仍将切断大于 16 KB 的帧并给出看门狗超时状态。

10.2.2.2 接收帧控制器

如果复位 MAC 帧过滤寄存器中的 RA (Receive All) 位，则 MAC 将根据目标/源地址执行帧过滤。如果应用程序决定不接收任何不良帧，如矮帧、CRC 错误帧等，则仍需要执行另一等级的过滤。检测到过滤失败时，帧将被丢弃且不会传输到应用程序。当过滤参数动态改变时，如果 (DA-SA) 过滤失败，则剩余的帧将被丢弃并且接收状态字将立即更新（零帧长度位、CRC 错误位和矮帧错误位将置 1），指示过滤失败。

10.2.2.3 接收流量控制

MAC 将检测接收暂停帧并暂停帧发送，暂停时间为接收的暂停帧内指定的延迟（仅限全双工模式）。可通过 Flow Control Register 中的 (Receive Flow Control Enable/RFCE) 位使能或禁止暂停帧检测功能。使能接收流量控制后，将开始监视接收帧的目标地址是否与控制帧的多播地址 (0x0180 C200 0001) 匹配。如果检测到匹配（接收的帧的目标地址与保留的控制帧的目标地址匹配），MAC 将根据 Frame Filter Register 中的 (Pass Control Frames/PCF) 位来决定是否将接收的控制帧传输应用程序。

MAC 还将对接收控制帧的类型、操作码和暂停定时器字段进行解码。如果状态的字节计数指示 64 个字节，并且不存在任何 CRC 错误，则 MAC 发送器将暂停任何数据帧的发送，暂停时间为解码的暂停时间值乘以时隙（对于 10/100 Mbit/s 模式，均为 64 字节）。同时，如果检测到另一个零暂停时间值的暂停帧，MAC 将复位暂停时间并管理新的暂停请求。

如果接收的控制帧与类型字段 (0x8808)、操作码 (0x00001) 以及字节长度（64 字节）均不匹配，或存在 CRC 错误，则 MAC 不会暂停。

如果暂停帧具有多播目标地址，MAC 将根据地址匹配来过滤帧。

对于具有单播目标地址的暂停帧，MAC 将根据 DA 是否与 MAC 地址 0 寄存器的内容匹配以及 Flow Control Register 中的 UPFD (Unicast Pause Frame Detect) 位是否置 1（检测具有单播目标地址的暂停帧）来进行过滤。PCF 寄存器位（Frame Filter Register 中的 Pass Control Frames 位 [7:6]）可对控制帧的过滤以及地址过滤进行控制。

10.2.2.4 接收多帧的操作处理

由于接收数据后状态立即可用，因此只要 FIFO 未满，就可以向其中存储帧。

10.2.2.5 错误处理

如果在从 MAC 接收 EOF 数据之前 Rx FIFO 已满，则将产生上溢并丢弃整个帧。由于上溢，状态位 (RDESO[11]) 将指示这是一个部分帧。如果使用 Operation Mode Register 中的 FTF (Flush Transmit FIFO) 和 FUGF (Forward Undersized Good Frames) 位使能相应功能，Rx FIFO 可过滤错误帧和过小帧。如果将接收 FIFO 配置为在存储转发模式下工作，则可过滤并丢弃所有错误帧。

在直通模式下，如果在从 Rx FIFO 读取帧的 SOF 时，该帧的状态和长度可用，则可丢弃整个错误帧。DMA 可通过使能接收帧清空位来清空正在从 FIFO 读取的错误帧。然后到应用的数据传输将停止，其余帧将从内部读取并丢弃。如果 FIFO 可用，则可以启动下一帧传输。

10.2.2.6 接收状态字

以太网帧接收结束时，MAC 向应用（DMA 侧）输出接收状态。接收状态的详细说明与 RDES0 中的位 [31:0] 相同。

10.3 MAC 中断控制器

MAC 内核可通过各种事件生成不同中断。

Interrupt Status 寄存器描述了可导致 MAC 内核生成中断的事件。可通过将中断屏蔽寄存器中相应的屏蔽位置 1 来阻止各事件生成中断。

中断寄存器位仅指示各种中断事件。必须读取相应的状态寄存器和其它寄存器才能清除中断。例如，如果中断寄存器的位 3 设置为高电平，则指示在掉电模式下接收到魔术数据包或网络唤醒帧。必须读取 PMT Control and Status 寄存器才能清除此中断事件。

10.4 MAC 地址的过滤

地址过滤将检查所有接收的帧的目标地址和源地址并相应报告地址过滤状态。地址检查基于应用程序选择的不同参数（帧过滤寄存器）。还可以识别过滤的帧：多播帧或广播帧。地址过滤使用的物理 (MAC) 地址进行地址检查。

10.4.1 单播目标地址过滤

MAC 支持多达 8 个用于单播完美过滤的 MAC 地址。如果选择完美过滤（复位帧过滤寄存器中的 bit[1]），MAC 会将接收的单播地址的所有 48 位与编程的 MAC 地址进行比较来确定是否匹配。默认情况下，始终使能 EMACADDR0，其它地址 EMACADDR0 ~ EMACADDR7 则通过单独的使能位进行选择。将其它地址 (EMACADDR0 ~ EMACADDR7) 的各个字节与接收的相应 DA 字节进行比较时，可以将寄存器中相应的屏蔽字节控制位置 1 来屏蔽该字节。这有助于 DA 的组地址过滤。

10.4.2 多播目标地址过滤

可通过将帧过滤寄存器中的 PAM (Pass All Multicast）位置 1，将 MAC 编程为通过所有多播帧。如果 PAM（Pass All Multicast）位复位，MAC 将根据帧过滤寄存器中的 bit[2]（必须保持复位值）执行对多播地址的过滤。

在完美过滤模式下，将多播地址与编程的 MAC 目标地址寄存器 EMACADDR0 ~ EMACADDR7 进行比较。组地址过滤也受到支持。

10.4.3 广播地址过滤

在默认模式下，MAC 不过滤任何广播帧。但是，如果将帧过滤寄存器中的 DBF (Disable Broadcast Frames) 位置 1 来将 MAC 编程为拒绝所有广播帧，则会丢弃任何广播帧。

10.4.4 单播源地址过滤

MAC 还可以根据接收的帧的源地址字段来执行完美过滤。默认情况下，AFM 将 SA 字段与 SA 寄存器中编程的值进行过滤。可通过将相应寄存器中的 bit[30] 置 1，来将 MAC 地址寄存器 [1:3] 配置为包含 SA 而不是 DA 进行比较。带 SA 的组地址过滤也受到支持。如果帧过滤寄存器中的 SAF (Source Address Filter Enable) 位置 1，则 MAC 将丢弃未通过 SA 过滤的帧。否则，SA 过滤的结果将通过接收状态字中的状态位给出（详见表 10-9）。

SAF (Source Address Filter Enable) 位置 1 时，对 SA 过滤和 DA 过滤的结果进行与运算，以决定是否需要转发帧。这意味着任何一个过滤未通过都将丢弃帧。两个过滤必须都通过，才能将帧转发到应用。

10.4.5 反向过滤操作

对于目标地址和源地址过滤，可在最终输出时选择相反的过滤匹配结果。这分别由帧过滤寄存器中的 DAIF 和 SAIF 位控制。DAIF 位同时适用于单播和多播 DA 帧。在反向过滤操作中，当 DAIF 位置 1 时，将反转单播/多播目标地址过滤的结果。类似地，当 SAIF 位置 1 时，将反转单播 SA 过滤的结果。

下面两个表按照接收帧的类型汇总了目标地址和源地址过滤。

表 10-1. 目标地址过滤

帧类型	PM	PF	DAIF	PAM	DB	DA 过滤结果
广播	1	X	X	X	X	通过
	0	X	X	X	0	通过
	0	X	X	X	1	不通过
单播	1	X	X	X	X	通过所有帧
	0	X	0	X	X	完美/组过滤匹配时通过
	0	X	1	X	X	完美/组过滤匹配时不通过
	0	1	0	X	X	完美/组过滤匹配时通过
	0	1	1	X	X	完美/组过滤匹配时不通过
多播	1	X	X	X	X	通过所有帧
	X	X	X	1	X	通过所有帧
	0	X	0	0	X	完美/组过滤匹配时通过，并在 PCF = 0x 时丢弃暂停控制帧
	0	1	0	0	X	完美/组过滤匹配时通过，并在 PCF = 0x 时丢弃暂停控制帧
	0	X	1	0	X	完美/组过滤匹配时不通过，并在 PCF = 0x 时丢弃暂停控制帧
	0	1	1	0	X	完美/组过滤匹配时不通过，并在 PCF = 0x 时丢弃暂停控制帧

表 10-1 中，MAC 帧过滤寄存器中的过滤参数如下:

帧类型说明		**参数设置**	
PM:	Pass All Multicast/ 通过所有组播	1:	置 1
PF:	Perfect Filter/ 完美过滤	0:	清零
DAIF:	Destination Address Inverse Filtering/目标地址反过滤		
PAM:	Pass All Multicast/通过所有组播		
DB:	Disable Broadcast Frames/关闭广播帧		

表 10-2. 源地址过滤

帧类型	PM	SAIF	SAF	SA 过滤操作
单播	1	X	X	通过所有帧
	0	0	0	完美 /组过滤匹配时通过，但不丢弃未通过的帧
	0	1	0	完美 /组过滤匹配时不通过，但不丢弃未通过的帧
	0	0	1	完美 /组过滤器匹配时通过并将不通过的帧丢弃
	0	1	1	完美 /组过滤器匹配时不通过并将不通过的帧丢弃

表 10-2 中，MAC 帧过滤寄存器的过滤参数如下：

帧类型说明		**参数设置**	
PM:	Pass All Multicast/通过所有组播	1:	置 1
SAF:	Source Address Filtering/源地址过滤	0:	清零
SAIF:	Source Address Inverse Filtering/源地址反向过滤	X:	无关

10.4.6 好的发送帧与接收帧

如果帧成功发送，则将发送的帧视为"好帧"。换句话说，如果帧发送过程没有因为以下错误而中止，则认为发送的帧是好帧：

- Jabber 超时
- 无载波 /载波丢失
- 延迟冲突
- 帧下溢
- 过度延迟
- 过度冲突

如果不存在以下错误，则认为接收帧是"好帧"：

- CRC 错误
- 矮帧（短于 64 字节）
- 对齐错误（仅限 10/100 Mbit/s）
- 长度错误（仅限非类型帧）
- 超出包最大大小范围（仅限非类型帧，超过最大大小）
- MII_RXER 输入错误

最大帧大小取决于帧类型，如下：

- 无标记帧的最大大小 = 1518
- VLAN 帧的最大大小 = 1522

10.5 EMAC_MTL（MAC 传输层）

MAC 传输层提供 FIFO 存储器来缓冲和调节应用系统存储器和 MAC 之间的帧。它还可以在应用时钟域和 MAC 时钟域之间传输数据。MTL 层具有两条数据路径，即发送路径和接收路径。双向数据路径位宽为 32，采用简单的 FIFO 协议操作。

10.6 PHY 接口

DMA 和主机驱动程序通过以下两个数据结构进行通信：

- 控制和状态寄存器 (CSR)
- 描述符列表和数据缓存

详见章节寄存器列表和链表描述符。

10.6.1 MII（介质独立接口）

介质独立接口 (MII) 定义了 10 Mbit/s 和 100 Mbit/s 的数据传输速率下 MAC 子层与 PHY 之间的互连。

10.6.1.1 MII 与 PHY 间的接口信号

MII 接口信号如图 10-3 所示。

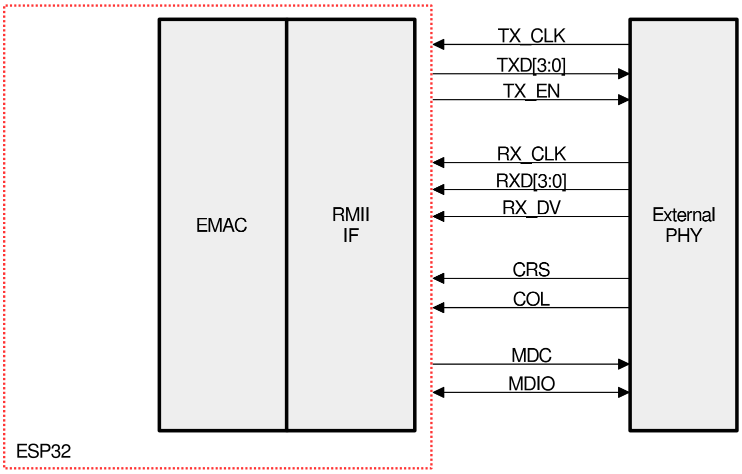


图 10-3. MII 接口

MII 接口信号说明：

- MII_TX_CLK：Tx 时钟信号。该信号提供进行 Tx 数据传输时的参考时序。频率分为两种：速率为 10 Mbit/s 时为 2.5 MHz；速率为 100 Mbit/s 时为 25 MHz。
- MII_TXD[3:0]：发送数据信号。该信号是 4 个一组的数据信号，由 MAC 子层同步驱动，在 MII_TX_EN 信号有效时才为有效信号（有效数据）。MII_TXD[0] 为最低有效位，MII_TXD[3] 为最高有效位。MII_TX_EN 信号为低时，发送数据不会对 PHY 产生任何影响。
- MII_TX_EN：发送数据使能信号。该信号表示 MAC 当前正针对 MII 发送半字节 (4 bits)。该信号必须与报头的前半字节进行同步 (MII_TX_CLK)，并在所有待发送的半字节均发送到 MII 时必须保持同步。
- MII_RX_CLK：RX 时钟信号。该信号提供进行 RX 数据传输时的参考时序。频率也分为两种：速率为 10 Mbit/s 时为 2.5 MHz；速率为 100 Mbit/s 时为 25 MHz。
- MII_RXD[3:0]：接收数据信号。该信号是 4 个一组的数据信号，由 PHY 同步驱动，在 MII_RX_DV 信号有效时才为有效信号（有效数据）。MII_RXD[0] 为最低有效位，MII_RXD[3] 为最高有效位。当 MII_RX_DV 禁止、MII_RX_ER 使能时，特定的 MII_RXD[3:0] 值用于代表来自 PHY 的特定信息。
- MII_RX_DV：接收数据有效信号。该信号表示 PHY 当前正针对 MII 接收已恢复并解码的半字节。该信号必须与恢复帧的头半字节进行同步于 MII_RX_CLK，并且一直保持同步到恢复帧的最后半字节。该信号必须在最后半字节随后的第一个时钟周期之前禁止。为了正确地接收帧，MII_RX_DV 信号必须在时间范围上涵盖要接收的帧，其开始时间不得迟于 SFD 字段出现的时间。
- MII_CRS：载波侦听信号。当发送或接收介质处于非空闲状态时，由 PHY 使能该信号。发送和接收介质均处于空闲状态时，由 PHY 禁止该信号。PHY 必须确保 MII_CS 信号在冲突条件下保持有效状态。该信号无需与 Tx 和 Rx 时钟保持同步。在全双工模式下，该信号没意义。
- MII_COL：冲突检测信号。检测到介质上存在冲突后，PHY 必须立即使能冲突检测信号，并且只要存在冲突条件，冲突检测信号必须保持有效状态。该信号无需与 Tx 和 Rx 时钟保持同步。在全双工模式下，该信号没意义。

- MII_RX_ER：接收错误信号。该信号必须保持一个或多个周期 (MII_RX_CLK)，从而向 MAC 子层指示在帧的某处检测到错误。
- MDIO 和 MDC：管理数据输入输出模块和管理数据时钟。这两个信号构成了符合 IEEE 802.3 标准的以太网串行总线，用于将控制和数据信息传输到 PHY。详见 Station Management Agent (SMA) Interface。

10.6.1.2 MII 时钟

在 MII 时钟模式下，MII 与 PHY 的接口有 Tx 和 Rx 两个方向的时钟，MII_TX_CLK 用于同步 Tx 的数据，MII_RX_CLK 用于同步 Rx 的数据。其中 MII_RX_CLK 时钟由 PHY 提供，MII_TX_CLK 由芯片内部的 PLL 提供或是芯片外部的晶振提供。图 10-4 的配置具体参考寄存器列表中时钟相关的寄存器。

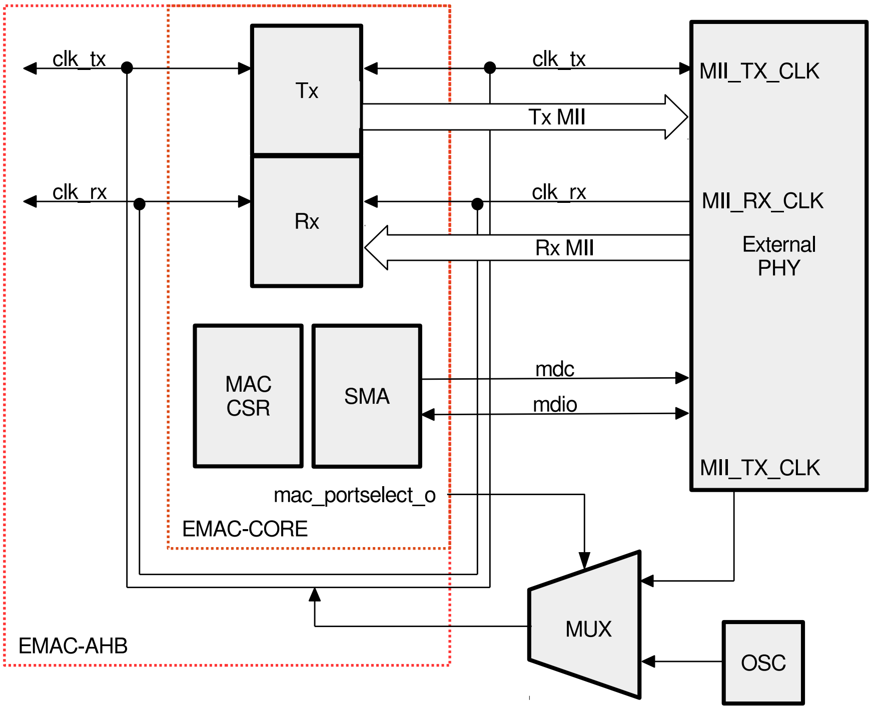


图 10-4. MII 时钟

10.6.2 RMII（精简介质独立接口）

RMII 接口信号如图 10-5 所示。

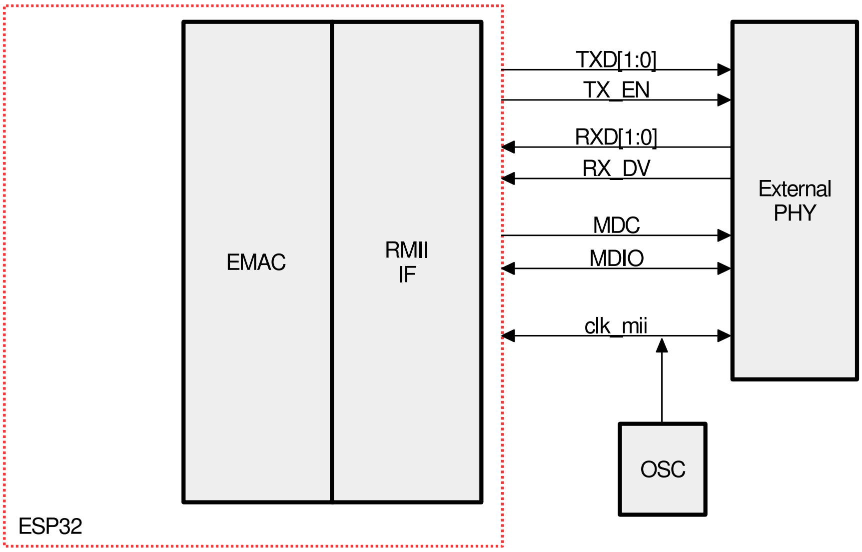


图 10-5. RMII 接口

10.6.2.1 RMII 接口信号描述

精简介质独立接口 (RMII) 规范降低了 10 Mbit/s 或 100 Mbit/s 下微控制器以太网外设与外部 PHY 间的引脚数。根据 IEEE 802.3u 标准，MII 包括 16 个包含数据和控制信号的引脚。RMII 规范将引脚数减少为 7 个（引脚数减少 62.5%）。

RMII 具有以下特性：

- 支持 10 Mbit/s 和 100 Mbit/s 的运行速率
- 参考时钟频率必须是 50 MHz
- 必须提供给 MAC 和外部以太网 PHY 相同的参考时钟。PHY 提供了独立的 2 位宽的发送和接收数据的路径。

10.6.2.2 RMII 时钟

RMII 时钟如图 10-6 所示。

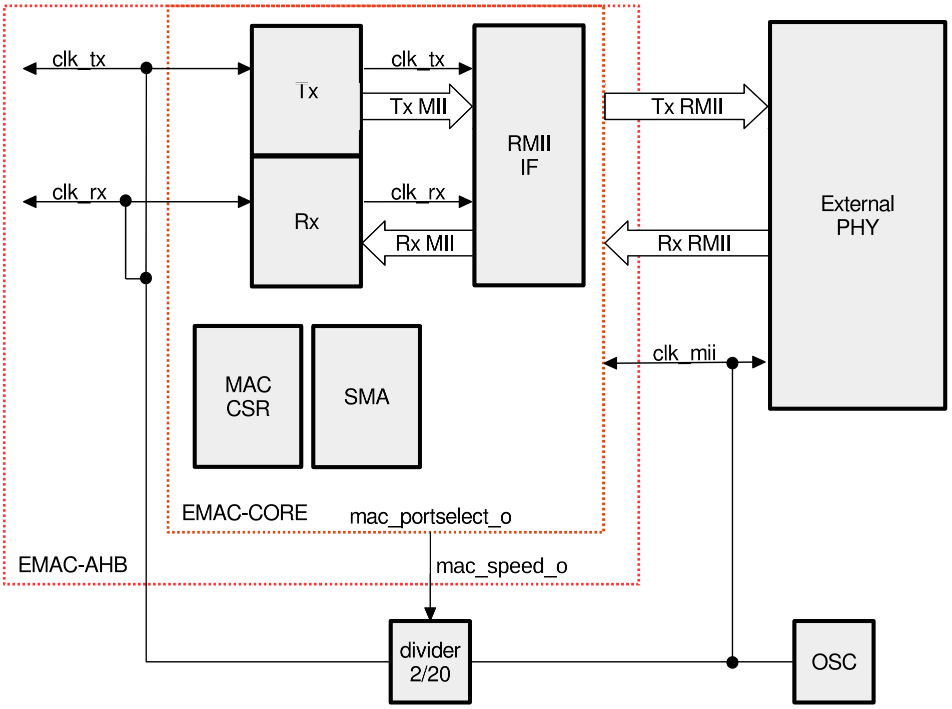


图 10-6. RMII 时钟

10.6.3 Station Management Agent (SMA) 接口

如图 10-4 所示，MAC 通过 MDC 和 MDIO 信号将控制和数据信息传输到 PHY。最大时钟频率为 2.5 MHz。时钟由应用时钟通过时钟分频器分频产生。在通过 MDIO 的写/读操作期间，PHY 发送寄存器数据。该信号与 MDC 时钟同步驱动。

EMII 地址寄存器和 EMII 数据寄存器详见寄存器列表。

10.6.4 RMII 接口时序要求

本章节详细说明了 RMII 接口的时序要求。

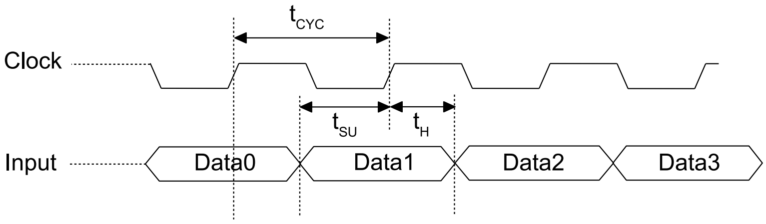


图 10-7. 接收数据时序图

表 10-3. 接收数据时序要求

时序参数	描述	最小值	典型值	最大值	单位
t_{CYC}	时钟周期 (Clock cycle)	20	20	20	ns
t_{SU}	建立时间 (Setup time)	4	–	–	ns
t_H	保持时间 (Hold time)	1	–	–	ns
t_{ID}	输入延迟 (Input delay)	3	5	8	ns

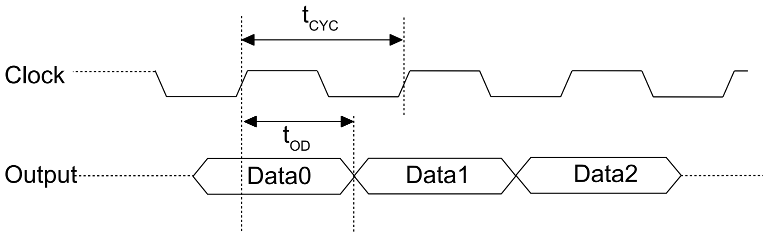


图 10-8. 发送数据时序图

表 10-4. 发送数据时序要求

时序参数	描述	最小值	典型值	最大值	单位
t_{CYC}	时钟周期 (Clock cycle)	20	20	20	ns
t_{SU}	建立时间 (Setup time)	4	–	–	ns
t_{H}	保持时间 (Hold time)	1	–	–	ns
t_{OD}	输出延迟 (Output delay)	6	9	12	ns

10.7 以太网 DMA 特性

DMA 具有独立的发送和接收引擎以及控制和状态寄存器地址。传输引擎将数据从系统内存传输到设备端口 (MTL)，而接收引擎将数据从设备端口传输到系统内存。控制器使用描述符，以最少的主机 CPU 干预来有效地将数据从源移动到目的地。DMA 用于以数据包为主的数据传输，如以太网中的帧。控制器可以配置为在正常情况下发中断给主 CPU，例如完成帧发送或接收，或发生错误时。

10.8 链表描述符

本节介绍链表和描述符结构。每个链表由 8 个 word 组成。

10.8.1 发送描述符

发送链表结构如图 10-9 所示。表 10-5 至表 10-8 为链表描述符说明。

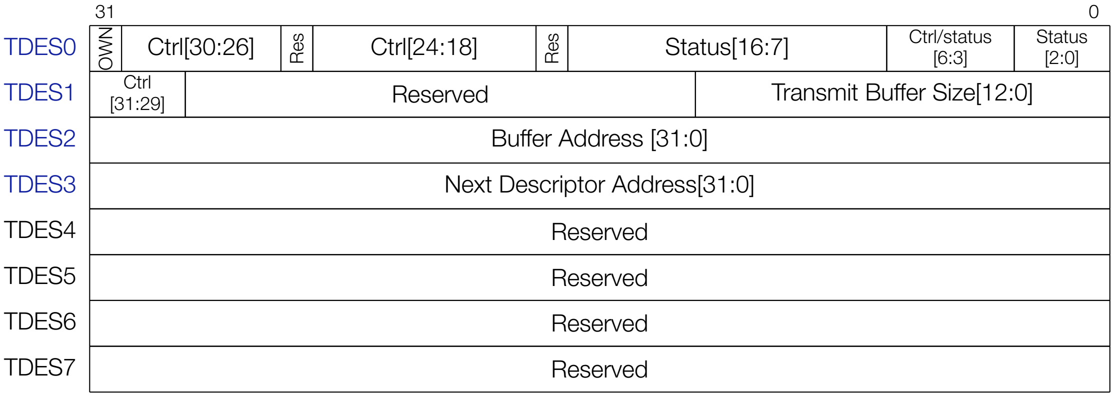


图 10-9. 发送描述符

表 10-5. 发送描述符 0 (TDES0)

位	命名	描述
[31]	OWN: Own Bit	该位置 1 时，表示描述符由 DMA 拥有。复位时，表示描述符由主机拥有。当 DMA 完成帧传输或分配在描述符中的缓存为空时，DMA 将清除该位。在设置属于同一帧的所有后续描述符之后，应设置帧的第一个描述符的控制位。这避免了在获取描述符和驱动程序设置控制位之间的可能发生的竞争。
[30]	IC: Interrupt on Completion	置 1 后，该位在发送当前帧之后设置发送中断 (Operation Mode Register[0])。该位仅在位 TDES0 [29] 置 1 时有效。
[29]	LS: Last Segment	置 1 时，该位表示缓存区中包含该帧的最后一段。当该位置 1 时，TDES1 中的 TBS1 或 TBS2 字段的值不为零。
[28]	FS: First Segment	置 1 时，该位表示缓存区中包含帧的第一段。
[27]	DC: Disable CRC	该位置 1 时，MAC 不会在发送帧的末尾附加循环冗余校验 (CRC)。这仅在位 TDES0 [28] 置 1 时有效。
[26]	DP: Disable Pad	置 1 时，MAC 不会自动在短于 64 字节的帧后面自动添加 padding。当该位复位时，DMA 自动将 padding 和 CRC 添加到短于 64 个字节的帧中，并且无论 DC (TDES0 [27]) 位的状态如何，都会添加 CRC 字段。只有当位 TDES0 [28] 置 1 时才有效。
[25]	Reserved	保留
[24]	CRCR: CRC Replacement Control	置 1 时，MAC 用重新计算的 CRC 字节替换发送的数据包的最后四个字节。主机应确保 CRC 字节在将要发送的帧的缓存中。当控制位 (TDES0 [28]) 置 1 时，该位有效。另外，只有当位 TDES0 [27] 置 1 时才进行 CRC 替换。
[23:22]	CIC: Checksum Insertion Control	这些位控制校验和的计算和插入。以下列表描述了位编码： • 2'b00：关闭校验和插入。 • 2'b01：仅使能 IP 报头校验和计算和插入。 • 2'b10：使能 IP 报头校验和以及有效负载校验的计算和插入，伪报头校验不在硬件中计算。 • 2'b11：使能 IP 报头校验和以及有效负载校验的计算和插入，但是伪报头校验在硬件中计算。 当控制位 TDES0 [28] 置 1 时，此字段有效。
[21]	TER: Transmit End of Ring	置 1 时，该位表示描述符列表已达到最后一个链表描述符。DMA 返回到链表的基地址，创建一个描述符环。
[20]	TCH: Second Address Chained	置 1 时，该位指示描述符中的第二个地址是下一个描述符地址。当 TDES0 [20] 置 1 时，TBS2 (TDES1 [28:16]) 是一个"无关"值。TDES0 [21] 优先于 TDES0 [20]。此位必须置 1。

位	命名	描述
[19:18]	VLIC: VLAN Insertion Control	置 1 时，这些位请求 MAC 在传输帧之前打上 VLAN 标记或取消标记。如果帧被打上 VLAN 标记，则 MAC 会自动重新计算并替换 CRC 字节。以下为这些位的说明： • 2'b00：不加上 VLAN 标记。 • 2'b01：在传输之前去掉 VLAN 标记，只用于 VLAN 帧。 • 2'b10：插入 VLAN 标记，标记的值在 VLAN 标记插入和替换寄存器中配置。 • 2'b11：用 VLAN 标记插入和替换寄存器中的标记值替换帧当中的 VLAN 标记。该选项只能用于 VLAN 帧。
[17]	Reserved	保留
[16]	IHE: IP Header Error	置 1 时，该位指示 MAC 发送器在 IP 报头中检测到错误。发送器检查 IPv4 数据包中的报头长度与从应用程序接收到的报头字节的数量，如果不匹配，则指示错误状态。对于 IPv6 帧，如果报头长度不是 40 个字节，则指示报头错误。此外，IPv4 或 IPv6 帧的“以太网长度/类型”字段值必须与接收到的 IP 报头版本匹配。对于 IPv4 帧，如果“报头长度”字段的值小于 0x5，则指示错误状态。
[15]	ES: Error Summary	此位代表以下 bit 的逻辑或: • TDES0[14]: Jabber 超时 • TDES0[13]: 帧刷新 • TDES0[11]: 载波丢失 • TDES0[10]: 无载波 • TDES0[9]: 延迟冲突 • TDES0[8]: 过度冲突 • TDES0[2]: 过度延迟 • TDES0[1]: 下溢错误 • TDES0[16]: IP 报头错误 • TDES0[12]: IP 有效负载错误
[14]	JT: Jabber Timeout	置 1 时，该位表示 MAC 发送器发生了 jabber 超时。只有在 EMACCONFIG_REG 的 EMACJABBER 位（关闭 Jabber）未置 1 时，该位置 1。
[13]	FF: Frame Flushed	置 1 时，该位表示 DMA 或 MTL 已经按照 CPU 给出的刷新命令刷新帧。
[12]	IPE: IP Payload Error	置 1 时，该位表示 MAC 发送器在 TCP，UDP 或 ICMP IP 数据报有效载荷中检测到错误。 发送器检查 IPv4 或 IPv6 报头中收到的有效负载长度与从应用程序接收到的 TCP，UDP 或 ICMP 数据包字节的实际数量，并在发生不匹配时指示错误状态。
[11]	LOC: Loss of Carrier	置 1 时，该位指示在帧传输期间发生载波丢失（即，在帧传输期间，一个或多个传输时钟周期内，MII_CRS 信号无效）。当 MAC 处于半双工模式时，只对传输中没有冲突的帧有效。
[10]	NC: No Carrier	置 1 时，该位指示在传输过程中来自 PHY 的载波侦听信号未被拉低。

位	命名	描述
[9]	LC: Late Collision	置 1 时，该位指示由于冲突窗口（在 MII 模式下，包括报头在内共 64 字节时间，和 512 字节时间，包括报头和载波扩展）之后发生冲突而发生的帧传输被中止。如果下溢错误位置 1，则该位无效。
[8]	EC: Excessive Collision	置 1 时，该位表示在尝试传送当前帧发生 16 次连续冲突之后传输被中止。如果 EMACCONFIG_REG 的 EMACRETRY 位置 1，则该位在第一次冲突后置 1，并且帧传输被中止。
[7]	VF: VLAN Frame	置 1 时，表示传输的帧是 VLAN 类型的帧。
[6:3]	Ctrl/status	这些状态位指示帧发送之前发生的冲突次数。当过度冲突位 (TDES0 [8]) 置 1 时，此计数无效。CPU 只在半双工模式下更新这个状态字段。
[2]	ED: Excessive Deferral	置 1 时，如果 MAC 配置寄存器 EMACCONFIG_REG 的 EMACDEFERRALCHECK（延迟检测）位置为高，则该位表示传输已经结束，原因是在使能巨帧的情况下，过度延迟超过 24,288 比特时间。
[1]	UF: Underflow Error	置 1 时，该位表示由于数据从主机存储器延迟到达，MAC 中止了帧发送。下溢错误表示 DMA 在传输帧时遇到空的传输缓存。传输过程进入暂停状态并将传输下溢寄存器（状态寄存器 bit[5]）和传输中断寄存器（状态寄存器 bit[0]）置 1。
[0]	DB: Deferred Bit	置 1 时，该位表示 MAC 由于载波的存在而传输延迟。该位仅在半双工模式下有效。

表 10-6. 发送描述符 1 (TDES1)

位	名称	描述
[31:29]	SAIC: SA Insertion Control	这些位请求 MAC 使用 GMACADDR0HIGH_REG，GMACADDR0LOW_REG，GMACADDR1HIGH_REG，GMACADDR1HIGH_REG 寄存器中给定的值添加或替换以太网帧中的源地址字段。如果“源地址”字段在帧中被修改，则 MAC 会自动重新计算并替换 CRC 字节。位 31 指定用于插入或替换源地址的 MAC 地址寄存器值（1 或 0）。以下列表描述了位 [30:29] 的值： • 2’b00：不加入源地址。 • 2’b01：插入源地址。为确保传输的可靠性，应用程序必须提供没有源地址的帧。 • 2’b10：替换源地址。为确保传输的可靠性，应用程序必须提供带有源地址的帧。 • 2’b11：保留 当控制位 TDES0 [28] 置 1 时，这些位有效。
[28:16]	Reserved	保留
[15:13]	Reserved	保留
[12:0]	TBS1: Transmit Buffer Size	链表数据缓存的大小，以字节为单位。如果该字段为 0，则 DMA 将忽略此缓存，并使用下一个描述符。

表 10-7. 发送描述符 2 (TDES2)

位	名称	描述
[31:0]	Buffer Address Pointer	缓存的物理地址。

表 10-8. 发送描述符 3 (TDES3)

位	名称	描述
[31:0]	Next Descriptor Address	该地址包含下一个描述符所在物理内存的指针。

10.8.2 接收描述符

接收链表结构如图 10-10 所示。表 10-9 至表 10-13 为链表描述。

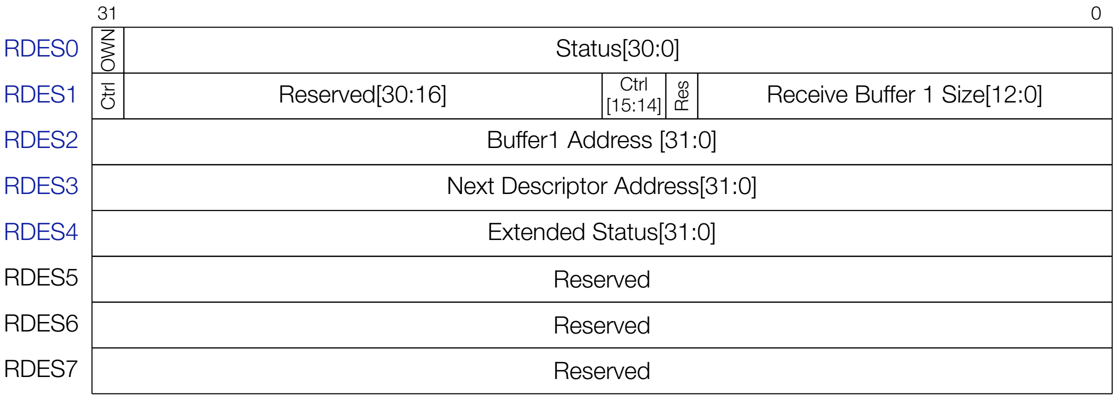


图 10-10. 接收链表结构

表 10-9. 接收描述符 0 (RDES0)

位	名称	描述
[31]	OWN: Own Bit	置 1 时，该位表示描述符由 DMA 所有。当该位复位时，该位表示描述符由主机拥有。DMA 在完成帧接收或与此链表有关的缓存已满时清除该位。
[30]	AFM: Destination Address Filter Fail	置 1 时，该位表示 MAC 中 DA 滤波器失败的帧。
[29:16]	FL: Frame Length	这些位表示发送到 CPU 内存的接收帧的字节长度。当 RDES0 [8] 被置 1 且描述符错误位 RDES0 [14] 或溢出错误位被复位时，该字段有效。当使能 IP 校验和计算（类型 1）并且所接收的帧不是 MAC 控制帧时，帧长度还包括附加到以太网帧的两个字节。

位	名称	描述
[15]	ES: Error Summary	该字段表示以下位的逻辑或： • RDES0[1]：CRC 错误 • RDES0[3]：接受错误 • RDES0[4]：看门狗超时 • RDES0[6]：延迟冲突 • RDES0[7]：巨帧 • RDES4[4:3]：IP 报头或负载错误 • RDES0[11]：上溢错误 • RDES0[14]：描述符错误 只有 RDES0 [8] 置 1 时，该字段有效。
[14]	DE: Descriptor Error	置 1 时，该位表示由帧大小超过当前描述符缓存的帧被截断，并且 DMA 不拥有下一个描述符。该帧被截断。只有当位 RDES0 [8] 置 1 时，该字段才有效。
[13]	SAF: Source Address Filter Fail	置 1 时，该位表示帧的 SA 字段未通过 MAC 中的 SA 过滤。
[12]	LE: Length Error	置 1 时，该位表示接收到的帧的实际长度和长度/类型字段不匹配。该位仅在帧类型 (RDES0 [5]) 位复位时有效。
[11]	OE: Overflow Error	置 1 时，该位表示由于 MTL 中的缓存溢出而导致接收到的帧被损坏。
[10]	VLAN: VLAN Tag	置 1 时，该位表示该描述符所指向的帧是由 MAC 标记的 VLAN 帧。VLAN 标记取决于根据 VLAN 标记寄存器设置检查接收到的帧的 VLAN 字段。
[9]	FS: First Descriptor	置 1 时，该位表示该描述符包含帧的第一个缓存区。如果第一个缓存区的大小是 0，则该帧从第二个缓存开始。如果第二个缓存的大小也是 0，则下一个描述符包含该帧的帧头。
[8]	LS: Last Descriptor	置 1 时，该位表示该描述符指向的缓存是该帧的最后一个缓存区。
[7]	IP Checksum Error (Type1), or Giant Frame	当选择 IP 校验和引擎（类型 1）时，该位置 1 表示以下情况之一： • MAC 内核计算的 16 位 IPv4 报头校验和与收到的校验和字节不匹配。 • 非 IPv4 帧绕过报头校验和检查。 若以上两种情况都不符合，则该位置 1 表示巨帧状态。对于普通帧，大于 1,518 字节的帧为巨帧（若该普通帧为 VLAN 帧，则大于 1,522 字节的帧为巨帧；当 GMAC_CONFIGREG bit[27] 为 1 时，大于 2,000 字节的帧为巨帧）。当使能巨帧处理时，大于 9,018 字节的帧为巨帧（若该巨帧为 VLAN 帧，则大于 9,022 字节的帧为巨帧）。
[6]	LC: Late Collision	置 1 时，该位表示在半双工模式下接收帧时发生延迟冲突。
[5]	FT: Frame Type	置 1 时，该位表示接收帧是以太网类型的帧（LT 字段大于或等于 1,536 字节）。当该位复位时，表示接收到的帧是 IEEE 802.3 帧。此位对于小于 14 个字节的矮帧无效。
[4]	RWT: Receive Watchdog Timeout	置 1 表示接收看门狗定时器在收到当前帧时已经超时，当前帧在看门狗超时后被截断。
[3]	RE: Receive Error	置 1 时，该位表示在接收帧期间若 MII_RXDV 被置 1，则发出 MII_RXER。

反馈文档意见

位	名称	描述
[2]	DE: Dribble Bit Error	置 1 时，该位表示接收到的帧具有非整数倍的字节（奇数个半字节）。该位仅在 MII 模式下有效。
[1]	CE: CRC Error	置 1 时，该位表示在接收到的帧上发生循环冗余校验 (CRC) 错误。只有位 RDES0 [8] 置 1 时，该字段才有效。
[0]	Extended Status Available/ Rx MAC Address	当 IP 校验和卸载（类型 2）存在时，该位置 1 时表明扩展状态在描述符字 4 (RDES4) 中可用。只有位 RDES0 [8] 置 1 时才有效。位 30 置 1 时该位无效。 当 IP 校验和卸载（类型 2）存在时，即使在 IP 校验和卸载引擎绕过接收帧的处理，该位也被置 1。绕过可能是因为非 IP 帧或非 TCP/UDP/ICMP 有效载荷的 IP 帧。 当 IPC 全部卸载未被选择时，该位表示 Rx MAC 地址状态。置 1 时，该位表示 Rx MAC 地址寄存器值（1 至 15）与帧的 DA 字段相匹配。复位时，该位表示 Rx MAC 地址寄存器 0 的值与 DA 字段匹配。

表 10-10. 接收描述符 1 (RDES1)

位	名称	描述
[31]	Ctrl	置 1 时，该位阻止帧的状态寄存器的 RI 位 (CSR5 [6]) 被置 1，使得收到的帧在当前描述符所指的缓存内结束。从而禁止由于该帧的 RI 向主机的中断触发。
[30:29]	Reserved	保留
[28:16]	Reserved	保留
[15]	RER: Receive End of Ring	置 1 时，该位表示描述符列表已达到其最后一个描述符。DMA 返回到列表的基地址，创建一个描述符环。
[14]	RCH: Second Address Chained	置 1 时，该位表示描述符中的第二个地址是下一个描述符地址，此位必须置 1。当该位置 1 时，RBS2 (RDES1 [28:16]) 是一个 "无关" 值。RDES1 [15] 优先级高于 RDES1 [14]。
[13]	Reserved	保留
[12:0]	RBS1: Receive Buffer 1 Size	表示第一个数据缓存的大小（字节）。即使 RDES2（buffer1 地址指针）的值未对齐，缓存大小也必须是 4 的倍数。当缓存大小不是 4 的倍数时，结果是不确定的。如果该字段为 0，则 DMA 将忽略此缓存，并根据 RCH 的值 (bit[14]) 使用缓存 2 或下一个描述符。

表 10-11. 接收描述符 2 (RDES2)

位	名称	描述
[31:0]	Buffer Address Pointer	这些位表示缓存的物理地址。

表 10-12. 接收描述符 3 (RDES3)

位	名称	描述
[31:0]	Next Descriptor Address	该地址包含指向下一个描述符所在物理内存的指针。

表 10-13. 接收描述符 4 (RDES4)

位	名称	描述
[31:28]	Reserved	保留
[27:26]	Reserved	保留
[25]	Reserved	保留
[24]	Reserved	保留
[23:21]	Reserved	保留
[20:18]	Reserved	保留
[17]	Reserved	保留
[16]	Reserved	保留
[15]	Reserved	保留
[14]	Reserved	保留
[13]	Reserved	保留
[12]	Reserved	保留
[11:8]	Message Type	这些位被编码成所接收消息的类型。 • 3’b0000：保留 • 3’b0001：SYNC (所有时钟类型) • 3’b0010：Follow_Up (所有时钟类型) • 3’b0011：Delay_Req (所有时钟类型) • 3’b0100：Delay_Resp (所有时钟类型) • 3’b0101：Pdelay_Req (点对点透明时钟) • 3’b0110：Pdelay_Resp (点对点透明时钟) • 3’b0111：Pdelay_Resp_Follow_Up (点对点透明时钟) • 3’b1000：Announce • 3’b1001：Management • 3’b1010：Signaling • 3’b1011-3’b1110：保留 • 3’b1111：保留
[7]	IPv6 Packet Received	置 1 时，该位表示接收到的数据包是一个 IPv6 数据包。该位仅在寄存器（MAC 配置寄存器）的 bit[10] (IPC) 置 1 时更新。
[6]	IPv4 Packet Received	置 1 时，表示接收到的数据包是 IPv4 数据包。该位仅在寄存器（MAC 配置寄存器）的 bit[10] (IPC) 置 1 时更新。
[5]	IP Checksum Bypassed	置 1 时，该位表示校验和卸载引擎被旁路。
[4]	IP Payload Error	置 1 时，该位表示 MAC 内核计算的 16 位 IP 有效负载校验和（即 TCP，UDP 或 ICMP 校验和）与接收的段中对应的校验和字段不匹配。当 TCP，UDP 或 ICMP 段的长度与 IP Header 字段中的负载长度值不匹配时，该位置 1。当 bit[7] 或 bit[6] 置 1 时，该位有效。
[3]	IP Header Error	置 1 时，该位表示由 MAC 内核计算的 16 位 IPv4 报头校验和与接收的校验和字节不匹配，或 IP 数据报版本与以太网类型值不一致。当 bit[7] 或 bit[6] 被置 1 时，该位有效。

位	名称	描述
[2:0]	IP Payload Type	这些位表示封装在由接收校验和卸载引擎 (COE) 处理的 IP 数据报中的有效载荷的类型。如果 COE 由于 IP 报头错误或分段 IP 而不处理 IP 数据报的有效载荷，COE 也将这些位置为 2’b00。 • 3’b000：未知或未处理 IP 负载 • 3’b001：UDP • 3’b010：TCP • 3’b011：ICMP • 3’b1xx：保留 当位 7 或位 6 置 1 时，该位有效。

10.9 寄存器列表

特定寄存器的特定字段或位具有不同的访问属性。以下列表为寄存器描述中使用的属性缩写。

- Read Only (RO) 只读
- Write Only (WO) 只写
- Read and Write (R/W) 读/写
- Read, Write, and Self Clear (R/W/SC) 读/写/自动清除
- Read, Self Set, and Write Clear (R/SS/WC) 读/自动设置/写清除
- Read, Write Set, and Self Clear (R/WS/SC) 读/写设置/自动清除
- Read, Self Set, and Self Clear or Write Clear (R/SS/SC/WC) 读/自动设置/自动清除/写清除
- Read Only and Write Trigger (RO/WT) 只读/写触发
- Read, Self Set, and Read Clear (R/SS/RC) 读/自动设置/读清除
- Read, Write, and Self Update (R/W/SU) 读/写/自动更新
- Latched-low (LL) 低锁存
- Latched-high (LH) 高锁存

名称	描述	地址	访问
DMA 配置和控制寄存器			
DMABUSMODE_REG	配置总线模式	0x3FF69000	R/WS/SC
DMATXPOLLDEMAND_REG	数据传输指令	0x3FF69004	RO/WT
DMARXPOLLDEMAND_REG	数据接收指令	0x3FF69008	RO/WT
DMARXBASEADDR_REG	第一个接收描述符的基地址	0x3FF6900C	R/W
DMATXBASEADDR_REG	第一个传输描述符的基地址	0x3FF69010	R/W
DMASTATUS_REG	中断，错误和其他事件的基地址	0x3FF69014	R/SS/WC
DMAOPERATION_MODE_REG	收发操作模式和命令寄存器	0x3FF69018	R/SS/WC
DMAIN_EN_REG	中断关闭/使能	0x3FF6901C	R/W
DMAMISSEDFR_REG	丢失帧和缓存溢出计数器	0x3FF69020	R/W
DMARINTWDTIMER_REG	接收看门狗计数器	0x3FF69024	R/W
DMATXCURRDESC_REG	指向当前传输描述符的指针	0x3FF69048	RO

名称	描述	地址	访问
DMARXCURRDESC_REG	指向当前接收描述符的指针	0x3FF6904C	RO
DMATXCURRADDR_BUF_REG	指向当前传输缓存的指针	0x3FF69050	RO
DMARXCURRADDR_BUF_REG	指向当前接受缓存的指针	0x3FF69054	RO
MAC 配置和控制寄存器			
EMACCONFIG_REG	MAC 配置	0x3FF6A000	R/W
EMACFF_REG	帧过滤设置	0x3FF6A004	R/W
EMACMIIADDR_REG	PHY 配置访问权限	0x3FF6A010	R/WS/SC
EMACMIIDATA_REG	PHY 数据读写	0x3FF6A014	R/W
EMACFC_REG	帧流控制	0x3FF6A018	R/WS/SC(FCB) R/W(BPA)
EMACDEBUG_REG	状态调试位	0x3FF6A024	RO
PMT_RWUFFR_REG	远程唤醒滤波器	0x3FF6A028	RO
PMT_CSR_REG	功耗管理控制和状态寄存器	0x3FF6A02C	RO
EMACLPI_CSR_REG	LPI 控制和状态寄存器	0x3FF6A030	RO
EMACLPITIMERSCONTROL_REG	LPI 计时器控制器	0x3FF6A034	RO
EMACINTS_REG	中断状态	0x3FF6A038	RO
EMACINTMASK_REG	中断屏蔽	0x3FF6A03C	R/W
EMACADDR0HIGH_REG	第一个 6 字节 MAC 地址的高 16 位	0x3FF6A040	R/W
EMACADDR0LOW_REG	第一个 6 字节 MAC 地址的低 32 位	0x3FF6A044	R/W
EMACADDR1HIGH_REG	第二个 6 字节 MAC 地址的高 16 位	0x3FF6A048	R/W
EMACADDR1LOW_REG	第二个 6 字节 MAC 地址的低 32 位	0x3FF6A04C	R/W
EMACADDR2HIGH_REG	第三个 6 字节 MAC 地址的高 16 位	0x3FF6A050	R/W
EMACADDR2LOW_REG	第三个 6 字节 MAC 地址的低 32 位	0x3FF6A054	R/W
EMACADDR3HIGH_REG	第四个 6 字节 MAC 地址的高 16 位	0x3FF6A058	R/W
EMACADDR3LOW_REG	第四个 6 字节 MAC 地址的低 32 位	0x3FF6A05C	R/W
EMACADDR4HIGH_REG	第五个 6 字节 MAC 地址的高 16 位	0x3FF6A060	R/W
EMACADDR4LOW_REG	第五个 6 字节 MAC 地址的低 32 位	0x3FF6A064	R/W
EMACADDR5HIGH_REG	第六个 6 字节 MAC 地址的高 16 位	0x3FF6A068	R/W
EMACADDR5LOW_REG	第六个 6 字节 MAC 地址的低 32 位	0x3FF6A06C	R/W
EMACADDR6HIGH_REG	第七个 6 字节 MAC 地址的高 16 位	0x3FF6A070	R/W
EMACADDR6LOW_REG	第七个 6 字节 MAC 地址的低 32 位	0x3FF6A074	R/W
EMACADDR7HIGH_REG	第八个 6 字节 MAC 地址的高 16 位	0x3FF6A078	R/W
EMACADDR7LOW_REG	第八个 6 字节 MAC 地址的低 32 位	0x3FF6A07C	R/W
EMACWDOGTO_REG	看门狗超时控制	0x3FF6A0DC	R/W
时钟配置寄存器			
EMAC_EX_CLKOUT_CONF_REG	RMII 时钟分频设置	0x3FF69800	R/W
EMAC_EX_OSCCLK_CONF_REG	RMII 时钟半整数和整数分频设置	0x3FF69804	R/W
EMAC_EX_CLK_CTRL_REG	时钟使能和外部/内部时钟选择	0x3FF69808	R/W
PHY 类型和 SRAM 配置寄存器			
EMAC_EX_PHYINF_CONF_REG	MII/RMII PHY 选择	0x3FF6980C	R/W
EMAC_PD_SEL_REG	使能关闭 Ethernet RAM，数据不丢失	0x3FF69810	R/W

10.10 寄存器

说明：所有 reserved 寄存器的值必须保持复位值。

本小节括号中的地址均为相对于 EMAC 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 4.12.1 *GPIO* 交换矩阵寄存器列表。

Register 10.1. DMABUSMODE_REG (0x0000)

(reserved)	DMAMIXEDBURST	DMAADDRALIBEA	PBLX8_MODE	USE_SEP_PBL	RX_DMA_PBL	FIXED_BURST	PRI_RATIO	PROG_BURST_LEN	ALT_DESC_SIZE	DESC_SKIP_LEN	DMA_ARB_SCH	SW_RST	
31 27	26	25	24	23	22 17	16	15 14	13 8	7	6 2	1	0	
0 0 0 0 0	0	0	0	0	0x01	0	0x0	0x01	0	0x00	0	1	Reset

DMAMIXEDBURST 当此位置为高电平且 FB(FIXED_BURST) 位为低电平时，AHB 主机接口将使用 INCR（未定义 burst）开始所有长度超过 16 的突发模式，而突发长度为 16 或少于 16 的 AHB 主接口适用固定突发传输（INCRx 和 SINGLE）。(读/写)

DMAADDRALIBEA 当此位置高且 FB 位为 1 时，AHB 接口将产生与起始地址 LS 位对齐的所有突发。如果 FB 位等于 0，则第一个突发（访问数据缓冲区的起始地址）未对齐，但随后的突发与地址对齐。(读/写)

PBLX8_MODE 置为高电平时，该位将编程的 PBL(PROG_BURST_LEN) 值（Bit [22:17] 和 Bit [13:8]）乘以 8。因此，DMA 根据 PBL 值以 8，16，32，64，128 和 256 节拍传输数据。(读/写)

USE_SEP_PBL 该位置位时，Rx DMA 使用 Bit [22:17] 中配置的值作为 PBL。Bit [13:8] 中的 PBL 值仅适用于 Tx DMA 操作。当复位为低电平时，Bit [13:8] 中的 PBL 值适用于两个 DMA 引擎。(读/写)

RX_DMA_PBL 该字段表示在一个 Rx DMA 传输中传送的最大节拍数。这是在单次读或写的最大值。每次在主机总线上开始突发传输时，Rx DMA 总是会尝试按照 RPBL 位的值进行传输。RPBL(RX_DMA_PBL) 值可配置为 1，2，4，8，16 和 32。任何其他值都会导致未定义的行为。该字段仅在 USP(USE_SEP_PBL) 置为高时有效。(读/写)

FIXED_BURST 该位决定 AHB 主接口是否执行固定突发传输。置 1 时，在正常突发传输开始时，AHB 接口只使用 SINGLE，INCR4，INCR8 或 INCR16 模式。复位时，AHB 接口使用 SINGLE 和 INCR 突发传输操作。(读/写)

PRI_RATIO 这些位控制 Rx DMA 和 Tx DMA 之间的加权调度中的优先级比率。只有当位 1（DA）复位时，这些位才有效。Rx 与 Tx 优先级比如下：(读/写)

- 2'b00 —1：1
- 2'b01 —2：0
- 2'b10 —3：1
- 2'b11 —4：1

PROG_BURST_LEN 这些位表示在一次 DMA 传输中要传送的最大节拍数。如果要传输的节拍数大于 32，则执行以下步骤：1. 设置 PBLx8 模式；2. 配置 PBL(PROG_BURST_LEN) 值。(读/写)

寄存器描述下一页继续。

Register 10.1. DMABUSMODE_REG (0x0000)

继上一页寄存器描述。

ALT_DESC_SIZE 置 1 时，描述符的大小增加到 32 个字节。（读/写）

DESC_SKIP_LEN 该位指定在两个未链接的描述符之间跳过的字的数量。地址从当前描述符结尾跳转到下一个描述符开始。当 DSL(DESC_SKIP_LEN) 值等于零时，描述符表在环模式下被 DMA 视为连续的。（读/写）

DMA_ARB_SCH 此位指定发送路径和接收路径之间的仲裁方案。1'b0：用 RX：TX 或 TX：RX 进行循环调度，PR（Bit [15:14]）指定优先级，1'b1 固定优先级（Rx 优先于 Tx）。（读/写）

SW_RST 当该位置 1 时，MAC DMA 控制器重置 MAC 的逻辑和所有 MAC 内部寄存器。在所有 ETH_MAC 时钟域中的复位操作完成后，该位自动清零。在对 ETH_MAC 的任何寄存器进行重新编程之前，应该在该位读取一个零值。（读/写设置/自动清除）

Register 10.2. DMATXPOLLDEMAND_REG (0x0004)

31	0	
0x000000000		Reset

TRANS_POLL_DEMAND 当这些位被写时，DMA 将读取寄存器（当前主机发送描述符寄存器）指向的当前描述符。如果该描述符不可用（由主机拥有），则传输返回挂起状态，并且寄存器（状态寄存器）的 Bit[2]（TU）被置 1。如果描述符可用，则传输继续。（只读/写触发）

Register 10.3. DMARXPOLLDEMAND_REG (0x0008)

31	0	
0x000000000		Reset

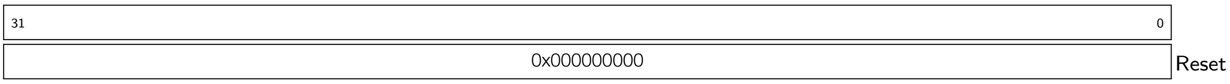
RECV_POLL_DEMAND 当这些位被写入时，DMA 将读取寄存器（当前主机接收描述符寄存器）指向的当前描述符。如果该描述符不可用（由主机拥有），则接收返回到暂停状态，并且状态寄存器的 Bit[7]（RU）被置 1。如果描述符可用，则 Rx DMA 返回到激活状态。（只读/写触发）

Register 10.4. DMARXBASEADDR_REG (0x000C)

31	0	
0x000000000		Reset

START_RECV_LIST 该字段包含接收描述符列表中第一个描述符的基地址。DMA 的 LSB Bit[1:0] 被忽略，并被内部视为全零。因此，这些 LSB 位只读。（读/写）

Register 10.5. DMATXBASEADDR_REG (0x0010)



START_TRANS_LIST 该字段包含发送描述符列表中第一个描述符的基地址。LSB Bit[1:0] 被忽略，在内部被 DMA 视为全零。因此，这些 LSB 位只读。（读/写）

Register 10.6. DMASTATUS_REG (0x0014)

	(reserved)	(reserved)	EMAC_PMT_INT	(reserved)	ERROR_BITS	TRANS_PROC_STATE	RECV_PROC_STATE	NORM_INT_SUMM	ABN_INT_SUMM	EARLY_RECV_INT	FATAL_BUS_ERR_INT	(reserved)	EARLY_TRANS_INT	RECV_WDT_TO	RECV_PROC_STOP	RECV_BUF_UNAVAIL	RECV_INT	TRANS_UNDFLOW	RECV_OVFLOW	TRANS_JABBER_TO	TRANS_BUF_UNAVAIL	TRANS_PROC_STOP	TRANS_INT	
	31 30	29	28	27 26	25 23	22 20	19 17	16	15	14	13	12 11	10	9	8	7	6	5	4	3	2	1	0	
	0 0	0	0	0 0	0x0	0x0	0x0	0	0	0	0	0 0	0	0	0	0	0	0	0	0	0	0	0	Reset

EMAC_PMT_INT 该位表示 ETH_MAC 的 PMT 模块中的中断事件。软件必须读取 MAC 中的 PMT 控制和状态寄存器才能得到确切的中断原因并清除其来源，将该位复位为 1’b0。(只读)

ERROR_BITS 该字段指示导致总线错误的错误类型，例如 AHB 接口上的错误响应。该字段仅在 Bit[13]（FBI）置 1 时有效。该字段不会生成中断。(只读)

- 3’b000：Rx DMA 写数据传输期间出错。
- 3’b011：Tx DMA 读数据传输期间出错。
- 3’b100：读 Rx DMA 描述符期间出错。
- 3’b101：写 Tx DMA 描述符期间出错。
- 3’b110：读取 Rx DMA 描述符期间出错。
- 3’b111：读取 Tx DMA 描述符期间出错。

TRANS_PROC_STATE 该字段表示发送 DMA FSM 状态。该字段不会生成中断。

- 3’b000：停止：复位或停止发出的发送命令。
- 3’b001：运行：获取发送传输描述符。
- 3’b010：保留。
- 3’b011：正在运行：正在等待发送数据包。
- 3’b100：暂停：发送描述符不可用。
- 3’b101：运行：关闭发送描述符。
- 3’b110：保留。
- 3’b111：运行：将发送数据包数据从发送缓存传输到主机内存。

寄存器描述下一页继续。

Register 10.6. DMASTATUS_REG (0x0014)

继上一页寄存器描述。

RECV_PROC_STATE 该字段表示接收 DMA FSM 状态。该字段不会生成中断。（只读）

- 3’b000：停止：复位或停止发出的接收命令。
- 3’b001：运行：获取接收传输描述符。
- 3’b010：保留。
- 3’b011：正在运行：正在等待接收数据包。
- 3’b100：暂停：接收描述符不可用。
- 3’b101：运行：关闭接收描述符。
- 3’b110：保留。
- 3’b111：运行：将接收数据包数据从接收缓存传输到主机内存。

NORM_INT_SUMM 当使能中断使能寄存器中的相应中断时，正常中断集合位的值是以下位的逻辑或：

- Bit[0]：发送中断。
- Bit[2]：发送缓存不可用。
- Bit[6]：接收中断。
- Bit[14]：提前接收中断。

只有未被屏蔽的位会影响正常中断集合位。这是一个粘滞位，必须在每次引起 NIS 置位的相应位清零（通过向该位写入 1）被清除。（读/自动设置/写清除）

ABN_INT_SUMM 当中断使能寄存器中的对应中断使能时，非正常中断集合位的值是以下逻辑或：

- Bit[1]：发送进程已停止。
- Bit[3]：传送 Jabber 超时。
- Bit[4]：接收 FIFO 溢出。
- Bit[5]：传输下溢。
- Bit[7]：接收缓冲器不可用。
- Bit[8]：接收进程已停止。
- Bit[9]：接收看门狗超时。
- Bit[10]：提前发送中断。
- Bit[13]：严重总线错误。

只有未被屏蔽的位会影响非正常中断集合位。这是一个粘滞位，必须在每次引起 AIS 置位的相应位清零时（向该位写入 1）清除。（读/自动设置/写清除）

寄存器描述下一页继续。

Register 10.6. DMASTATUS_REG (0x0014)

继上一页寄存器描述。

EARLY_RECV_INT 该位表示 DMA 填充了数据包的第一个数据缓冲区。当软件向该位写入 1 或该寄存器的 Bit[6]（RI）置 1 时，该位清零。（读/自动设置/写清除）

FATAL_BUS_ERR_INT 该位表示发生 Bit[25:23] 所述的总线错误。当该位置 1 时，相应的 DMA 引擎关闭其所有的总线访问。（读/自动设置/写清除）

EARLY_TRANS_INT 该位表示要发送的帧被完全传送到 MTL 发送 FIFO。（读/自动设置/写清除）

RECV_WDT_TO 置 1 表示接收看门狗定时器在接收到当前帧时已经过期，当前帧在看门狗超时后被截断。（读/自动设置/写清除）

RECV_PROC_STOP 接收进程进入停止状态时，该位置 1。（读/自动设置/写清除）

RECV_BUF_UNAVAIL 该位表示主机拥有接收列表中的下一个描述符，并且 DMA 无法获取该描述符。接收过程被暂停。要恢复处理接收描述符，主机应更改描述符的所有权并发出“接收轮询需求”命令。如果没有收到轮询请求，则接收进程在收到下一个识别的传入帧时恢复。只有当 DMA 拥有前一个接收描述符时，该位才被置 1。（读/自动设置/写清除）

RECV_INT 该位表示帧接收已完成。当接收完成时，RDES1 的 Bit[31] 在最后的描述符中被复位，并且特定的帧状态信息在描述符中被更新。接收处于运行状态。（读/自动设置/写清除）

TRANS_UNDFLOW 该位表示发送缓存在帧传输期间具有下溢。发送暂停，并将下溢错误位 TDES0[1] 置 1。（读/自动设置/写清除）

RECV_OVFLOW 该位表示接收缓存在帧接收过程中有溢出。如果部分帧被传送到应用程序，则溢出状态将在 RDES0 [11] 中置 1。（读/自动设置/写清除）

TRANS_JABBER_TO 该位表示发送 Jabber 定时器超时，当帧大小超过 2048 字节（当 Jumbo 帧被使能时为 10,240 字节）时发生。当发生 Jabber 超时时，传输过程被中止并进入停止状态。这会导致发送 Jabber 超时 TDES0 [14] 标志置 1。（读/自动设置/写清除）

TRANS_BUF_UNAVAIL 该位表示主机在发送列表中拥有下一个描述符，并且 DMA 无法获取它。传输被暂停。Bit[22:20] 表示发送进程状态转换。要恢复处理传输描述符，主机应通过设置 TDES0 [31] 来更改描述符的所有权，然后发出传输轮询需求命令。（读/自动设置/写清除）

TRANS_PROC_STOP 传输停止时该位被置 1。（读/自动设置/写清除）

TRANS_INT 该位表示帧传输已完成。传输完成后，TDES0 的 Bit[31]（OWN）复位，并在描述符中更新特定的帧状态信息。（读/自动设置/写清除）

Register 10.7. DMAOPERATION_MODE_REG (0x0018)

(reserved)	DIS_DROP_TCPIP_ERR_FRAM	RX_STORE_FORWARD	DIS_FLUSH_RECV_FRAMES	(reserved)	(reserved)	TX_STR_FWD	FLUSH_TX_FIFO	(reserved)	TX_THRESH_CTRL	START_STOP_TRANSMISSION_COMMAND	(reserved)	(reserved)	(reserved)	FWD_ERR_FRAME	FWD_UNDER_GF	DROP_GFRM	RX_THRESH_CTRL	OPT_SECOND_FRAME	START_STOP_RX	(reserved)	
31 27	26	25	24	23	22	21	20	19 17	16 14	13	12 11	10 9	8	7	6	5	4 3	2	1	0	
0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

DIS_DROP_TCPIP_ERR_FRAM 置 1 时，ETH_MAC 不会丢弃 checksum 错误的包。置 1 时，并且 FWD_ERR_FRAME 位为 0 时，所有错误的包都会被丢弃。(读/写)

RX_STORE_FORWARD 当此位为 1 时，MTL 层会直接把写到 RX FIFO 的包转发出去。(读/写)

DIS_FLUSH_RECV_FRAMES 当此位为 1 时，RX DAM 不会冲刷掉因为链表或缓存取不可用的包。(读/写)

TX_STR_FWD 当此位为 1 时，并且整个发送的帧已经存储到 TX FIFO 中时，开始传送包。此时，TX_THRESH_CTRL 中的值被忽略。(读/写)

FLUSH_TX_FIFO 置 1 时，TX FIFO 中的所有值被清除为默认值。当清除完成后，此位自动清零。(读/写设置/自动清除)

TX_THRESH_CTRL 当 TX FIFO 中的值大于此既存器中的阈值值，开始发送帧。当整帧的大小小于这个阈值时，此帧也将被发送。仅仅 TX_STR_FWD 寄存器为 0 时，此仅寄存器才有效。阈值大小：3’b000：64；3’b001：128；3’b010：192；3’b011：256；3’b100：40；3’b101：32；3’b110：24；3’b111：1。(读/写)

START_STOP_TRANSMISSION_COMMAND 当 TX FIFO 中的值大于此既存器中的阈值值，开始发送帧。当整帧的大小小于这个阈值时，此帧也将被发送。仅当 TX_STR_FWD 寄存器为 0 时，此寄存器才有效。阈值大小：000：64, 001：128, 010：192, 011：256, 100：40, 101：32, 110：24, 111：16。(读/写)

FWD_ERR_FRAME 当此位为 0 时，RX FIFO 会清除掉所有错误帧 (CRC 错，冲突错，超大帧错，看门狗超时错，溢出错)，否则会转发这些错误帧。(读/写)

FWD_UNDER_GF 当此位置 1 时，RX FIFO 会转发包括 PAD 和 CRC 的较小帧 (长度小于 64 字节的正确帧)，否则不会转发。

DROP_GFRM 当此位置 1 时，ETH_MAC 会丢掉 RX FIFO 中的超大帧。(读/写)

寄存器描述下一页继续。

Register 10.7. DMAOPERATION_MODE_REG (0x0018)

继上一页寄存器描述。

RX_THRESH_CTRL 2’b00：64，2’b01：32，2’b10：96，2’b11：128. 当 RX FIFO 中的值大于这个阈值时，会触发 DMA 开始接受此帧的数据到缓存中。2’b00：64；2’b01：32；2’b10：96；2’b11：128。（读/写）

OPT_SECOND_FRAME 当此位置 1 时，在前一帧的传送状态被硬件获取之前 TX DMA 开始处理下一帧的数据。（读/写）

START_STOP_RX 置 1 时，RX DMA 开始接收数据，当此位为 0 时，RX DMA 接收完当前正在接收的帧之后停止接收新的帧。（读/写）

Register 10.8. DMAIN_EN_REG (0x001C)

(reserved)	DMAIN_NISE	DMAIN_AISE	DMAIN_ERIE	DMAIN_FBEE	(reserved)	DMAIN_ETIE	DMAIN_RWTE	DMAIN_RSE	DMAIN_RBUE	DMAIN_RIE	DMAIN_UIE	DMAIN_OIE	DMAIN_TJTE	DMAIN_TBUE	DMAIN_TSE	DMAIN_TIE	
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	Reset

DMAIN_NISE 该位置 1 时，使能正常中断集合。当该位复位时，关闭正常中断集合。该位使能状态寄存器中的以下中断：状态寄存器 Bit[0]：发送中断。状态寄存器 Bit[2]：发送缓存不可用。状态寄存器 Bit[6]：接收中断。状态寄存器 Bit[14]：早期接收中断。（读/写）

DMAIN_AISE 该位置 1 时，使能非正常中断汇总。该位复位时，关闭非正常中断集合。该位使能状态寄存器中的以下中断：

- 状态寄存器 Bit[1]：传送进程停止。
- 状态寄存器 Bit[3]：传送 Jabber 超时。
- 状态寄存器 Bit[4]：接收溢出。
- 状态寄存器 Bit[5]：发送下溢。
- 状态寄存器 Bit[7]：接收缓存区不可用。
- 状态寄存器 Bit[8]：接收进程停止。
- 状态寄存器 Bit[9]：接收看门狗超时。
- 状态寄存器 Bit[10]：提前发送中断。
- 状态寄存器 Bit[13]：严重的总线错误。（读/写）

DMAIN_ERIE 当该位通过正常中断集合要使能位（Bit[16]）置 1 时，使能提前接收中断。当该位复位时，关闭提前接收中断。（读/写）

DMAIN_FBEE 当该位通过非正常中断集合使能（Bit[15]）置 1 时，使能严重总线错误中断。当该位复位时，关闭致命总线错误使能中断。（读/写）

DMAIN_ETIE 当该位通过非正常中断集合使能位（Bit[15]）置 1 时，使能提前发送中断。当该位复位时，关闭提前发送中断。（读/写）

DMAIN_RWTE 当该位通过非正常中断集合使能位（Bit 15）置 1 时，接收看门狗超时中断使能。当该位复位时，关闭接收看门狗超时中断。（读/写）

DMAIN_RSE 当该位由非正常中断集合使能位（Bit[15]）置 1 时，使能接收停止中断。当该位复位时，关闭接收停止中断。（读/写）

DMAIN_RBUE 当该位通过非正常中断集合使能位（Bit[15]）置 1 时，使能接收缓存不可用中断。当该位复位时，关闭接收缓存区不可用中断。（读/写）

DMAIN_RIE 当该位通过正常中断集合使能位（Bit[16]）置 1 时，使能接收中断。当该位复位时，关闭接收中断。（读/写）

寄存器描述下一页继续。

Register 10.8. DMAIN_EN_REG (0x001C)

继上一页寄存器描述。

DMAIN_UIE 当该位通过非正常中断集合使能位（Bit[15]）置 1 时，使能发送下溢中断。当该位复位时，关闭下溢中断。（读/写）

DMAIN_OIE 当此位通过非正常中断集合使能位（Bit[15]）置 1 时，使能接收溢出中断。当该位复位时，关闭溢出中断。（读/写）

DMAIN_TJTE 当该位通过异常中断集合使能位（Bit[15]）置 1 时，使能发送 Jabber 超时中断。当该位复位时，关闭发送 Jabber 超时中断。（读/写）

DMAIN_TBUE 当该位通过正常中断集合使能位（Bit[16]）置 1 时，使能发送缓存不可用中断。当该位复位时，关闭发送缓存不可用中断。（读/写）

DMAIN_TSE 当此位通过异常中断集合使能位（Bit[15]）置 1 时，使能发送停止中断。当该位复位时，关闭发送停止中断。（读/写）

DMAIN_TIE 当该位通过正常中断集合使能位（Bit[16]）置 1 时，使能发送中断。当该位复位时，关闭发送中断。（读/写）

Register 10.9. DMAMISSEDFR_REG (0x0020)

(reserved)	Overflow_BFOC	Overflow_FC	Overflow_BMFC	Missed_FC	
30 29	28	27 17	16	10 0	
0 0	0x0	0x0	0x0	0x0	Reset

Overflow_BFOC 当寄存器 Overflow_FC (Bits[27:17]) 溢出时，此位置 1。也就是说 RX FIFO 溢出，同时溢出帧计数器达到了最大值。在这种情况下，溢出帧计数器会复位成 0，此位表示发生溢出行为。（读/自动设置/读清除）

Overflow_FC 此位表示溢出过后有多少个帧溢出丢失而没有上传给上层应用程序，MTL 的 FIFO 每溢出一次，这个计数器就会累加 1，每次读取该寄存器，计数器就会清除。（读/自动设置/读清除）

Overflow_BMFC 当寄器 Missed_FC (Bits[15:0]) 溢出时，此位置 1。也就是说此时应用端缓存取不可用，同时丢失帧计数器达到了最大值。在这种情况下，丢失帧计数器会复位成 0，此位表示发生计数器发生溢出行为。（读/自动设置/读清除）

Missed_FC 此位表示接收端缓存区不可用，导致有多少个帧丢失而没有上传给上层应用程序，DMA 每丢弃一帧，这个计数器就会累加 1，每次读取该寄存器，计数器就会清除。（读/自动设置/读清除）

Register 10.10. DMARINTWDTIMER_REG (0x0024)

(reserved)	RIWTC	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x000	Reset

RIWTC 该位表示系统时钟周期数乘以 256。在 Rx DMA 完成传送由于相应描述符 RDES1 [31] 中的设置而未设置 RI 状态位的帧后，看门狗定时器将以编程值触发。看门狗定时器计数停止后，RI 位置 1，定时器停止。当由于接收帧的 RDES1[31] 的 RI 设置，RI (RECV_INT) 位被置为高电平时，看门狗定时器复位。（读/写）

Register 10.11. DMATXCURRDESC_REG (0x0048)

31 0	
0x000000000	Reset

TRANS_DECR_ADDR_PTR 复位时清零。操作期间由 DMA 更新指针。当前发送链表的地址。（只读）

Register 10.12. DMARXCURRDESC_REG (0x004C)

31 0	
0x000000000	Reset

RECV_DECR_ADDR_PTR 复位时清零。操作期间由 DMA 更新指针。当前接收链表的地址。（只读）

Register 10.13. DMATXCURRADDR_BUF_REG (0x0050)

31 0	
0x000000000	Reset

TRANS_BUFF_ADDR_PTR 复位时清零。操作期间由 DMA 更新指针。当前发送链表的地址。（只读）

Register 10.14. DMARXCURRADDR_BUF_REG (0x0054)

31 0	
0x000000000	Reset

RECV_BUFF_ADDR_PTR 复位时清零。操作期间由 DMA 更新指针。当前接收链表的地址。（只读）

Register 10.15. EMACCONFIG_REG (0x1000)

(reserved)	SAIRC	ASS2KP	(reserved)	EMACWATCHDOG	EMACJABBER	(reserved)	EMACJUMBOFRAME	EMACINTERFRAMEGAP	EMACDISABLECRS	EMACMII	EMACFESPEED	EMACRXOWN	EMACLOOPBACK	EMACDUPLEX	EMACRXIPCOFFLOAD	EMACRETRY	(reserved)	EMACPADCRCSTRIP	EMACBACKOFFLIMIT	EMACDEFERRALCHECK	EMACTX	EMACRX	PLTF	
31	30 28	27	26 24	23	22	21	20	19 17	16	15	14	13	12	11	10	9	8	7	6 5	4	3	2	1 0	
0	0x0	0	0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x0	0	0	0	0x0	Reset

SAIRC 该字段控制所有传输帧的源地址插入或替换。Bit[30] 根据 Bit[29:28] 的值指定将哪个 MAC 地址寄存器（0 或 1）用于源地址插入或替换。（读/写）

- 2’b0x：输入信号 mti_sa_ctrl_i 和 ati_sa_ctrl_i 控制 SA 字段的生成。
- 2’b10：如果将 Bit[30] 置 0，则 MAC 将 MAC 地址 0 寄存器的内容插入到所有传输帧的 SA 字段中。如果将 Bit[30] 置 1，则 MAC 将 MAC 地址 1 寄存器的内容插入所有传输帧的 SA 字段。
- 2’b11：如果将 Bit[30] 置 0，则 MAC 替换所有传输帧的 SA 字段中的 MAC 地址 0 寄存器的内容。如果将 Bit[30] 置 1，则 MAC 替换所有传输帧的 SA 字段中的 MAC 地址 1 寄存器的内容。

ASS2KP 置 1 时，MAC 将所有长度不超过 2,000 个字节的帧视为普通数据包。当 Bit[20]（JE）未置 1 时，MAC 将所有接收到的大于 2K 字节的帧视为巨帧。当此位复位且 Bit[20]（JE）未置 1 时，MAC 将所有收到的大小超过 1,518 字节（标记为 1,522 字节）的帧视为巨帧。当 Bit[20] 置 1 时，该位对巨帧状态没有影响。（读/写）

EMACWATCHDOG 当该位置 1 时，MAC 关闭接收器上的看门狗定时器。MAC 可以接收最多 16,383 字节的帧。当该位复位时，MAC 不接收超过 2048 字节（如果 JE 位置高，则为 10,240 字节）的接收帧或看门狗超时寄存器中设置的值。MAC 切断超出看门狗限制范围之后接收到的任何字节数。（读/写）

EMACJABBER 当该位置 1 时，MAC 关闭发送器上的 Jabber 定时器。MAC 可以传输长达 16,383 字节的帧。当此位复位时，如果应用程序在传输过程中发送了超过 2048 字节（如果 JE 设置为高电平，则为 10240 字节）的数据，MAC 将切断发送器。（读/写）

EMACJUMBOFRAME 当该位置 1 时，MAC 允许 9,018 字节的巨帧（或 9,022 字节的 VLAN 标记帧），而不在接收帧状态中报告巨大帧错误。（读/写）

EMACINTERFRAMEGAP 这些位控制传输时帧间的最小 IFG。（读/写）

- 3’b000：96 比特时间。
- 3’b001：88 比特时间。
- 3’b010：80 比特时间。
- 3’b111：40 比特时间。在半双工模式下，最小的 IFG 为 64 比特时间（IFG = 100）。

寄存器描述下一页继续。

Register 10.15. EMACCONFIG_REG (0x1000)

继上一页寄存器描述。

EMACDISABLECRS 当该位置高时，MAC 发送器在半双工模式下的帧发送期间忽略 MII CRS 信号。此请求不会导致在传输过程中由于载波丢失或无载波而产生的错误。当此位为低时，MAC 发送器由于载波侦听而产生这样的错误，甚至可以中止传输。（读/写）

EMACMII 该位选择以太网信号线速度。对于 10 或 100 Mbps 操作，该位置 1。在 10 或 100 Mbps 操作中，此位与 FES(EMACFESPEED) 位一起选择信号线速率。（读/写）

EMACFESPEED 该位选择 MII，RMII 接口速度。0：10 Mbps；1：100 Mbps。（读/写）

EMACRXOWN 该位置 1 时，在半双工模式下当 TX_EN 触发中断时，MAC 关闭接收帧。该位复位时，MAC 将在传输时接收 PHY 发出的所有数据包。MAC 处于全双工模式时，该位不适用。（读/写）

EMACLOOPBACK 该位置 1 时，MAC 使用回调模式 MII。MII 接收时钟输入信号（CLK_RX）是环回正常工作所必需的，因为发送时钟在内部没有环回。（读/写）

EMACDUPLEX 该位置 1 时，MAC 工作在全双工模式下，可以同时发送和接收数据。该位在全双工模式下为 RO，默认值为 1'b1。（读/写）

EMACRXIPCOFFLOAD 当该位置 1 时，MAC 计算所有接收的以太网帧有效载荷的 16 位的补码和，再对相加和取补。它还检查 IPv4 报头校验和（接收到的以太网帧的字节 25/26 或 29/30（VLAN 标记）是否对接收帧是正确的，并在接收状态字中给出状态。MAC 还将 IP 报头数据报有效负载（IPv4 报头后的字节）16 位校验和附加到应用程序的以太网帧当中（当取消选择类型 2 COE 时）。该位复位时，该功能被禁止。（读/写）

EMACRETRY 当该位置 1 时，MAC 只尝试一次传输。当 MII 接口上发生冲突时，MAC 将忽略当前帧传输，并在传输帧状态中报告由于出现过多冲突错误的帧终止。当该位复位时，MAC 将根据 BL 字段（Bit [6:5]）的设置尝试重试。该位仅适用于半双工模式。（读/写）

EMACPADCRCSTRIP 当该位置 1 时，只有在长度字段的值小于 1,536 字节的情况下，MAC 才去除传入帧上的 Pad 或 FCS 字段。所有接收到的长度字段大于或等于 1,536 字节的帧将被传送到应用程序，而不去除 Pad 或 FCS 字段。当该位复位时，MAC 将所有接收的帧传递给主机，而不修改它们。（读/写）

EMACBACKOFFLIMIT 退避极限决定在发生冲突后，在重试时重新安排传输之前，MAC 等待的时隙时间延迟（10/100 Mbps 的 512 比特时间）的随机整数（r）。该位仅适用于半双工模式。00：k = min（n，10）。01：k = min（n，8）。10：k = min（n，4）。11：k = min（n，1），其中 n = 重传尝试次数。随机整数 r 的取值范围为 0 <r <2k。（读/写）

EMACDEFERRALCHECK 延期检查使能。（读/写）

寄存器描述下一页继续。

Register 10.15. EMACCONFIG_REG (0x1000)

继上一页寄存器描述。

EMACTX 当该位置 1 时，MAC 的发送状态机被启用在 MII 上发送。当该位被复位时，MAC 发送状态机在完成当前帧的发送之后被关闭，并且不发送任何其他帧。(读/写)

EMACRX 当该位置 1 时，使能 MAC 的接收器状态机以接收来自 MII 的帧。当该位复位时，MAC 接收状态机在完成当前帧的接收后被关闭，并且不从 MII 接收任何其他帧。(读/写)

PLTF 这些位控制添加到每个发送帧的开始处的前导字节数量。只有在 MAC 出于全双工模式下时，才会减少报头字节数量。(读/写)

- 2'b00：7 字节的报头。
- 2'b01：5 字节的报头。
- 2'b10：3 字节的报头。

Register 10.16. EMACFF_REG (0x1004)

RECEIVE_ALL	(reserved)	SAFE	SAIF	PCF	DBF	PAM	DAIF	(reserved)	PMODE	
31	30 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	0x0	0	0	0	0 0	0	Reset

RECEIVE_ALL 当该位置 1 时，MAC 接收器模块将所有接收到的帧传送给应用程序，而不管它们是否通过地址过滤器。SA 或 DA 过滤的结果在接收状态字的相应位中更新（通过或失败）。当该位复位时，接收器模块仅将那些帧传递给通过 SA 或 DA 地址过滤器的应用程序。（读/写）

SAFE 当该位置 1 时，MAC 将接收帧的 SA 字段与使能的 SA 寄存器中编程的值进行比较。如果比较失败，则 MAC 丢弃帧。当该位复位时，MAC 根据 SA 地址比较将接收到的帧转发到具有 Rx 状态的更新的 SAF 位的应用。（读/写）

SAIF 当该位置 1 时，地址检查模块将在反向滤波模式下进行 SA 地址比较。SA 与 SA 寄存器匹配的帧被标记为未通过 SA 地址过滤。当该位复位时，SA 与 SA 寄存器不匹配的帧被标记为未通过 SA 地址过滤。（读/写）

PCF 这些位控制所有控制帧（包括单播和多播暂停帧）的转发。（读/写）

- 2’b00：MAC 过滤掉所有控制帧。
- 2’b01：MAC 将除暂停帧之外的所有控制帧转发给应用程序，即使这些帧未通过地址过滤。
- 2’b10：即使 MAC 地址过滤器失败，MAC 也会将所有控制帧转发给应用程序。
- 2’b11：MAC 转发通过地址过滤器的控制帧。

暂停帧的处理应符合以下条件：

- 条件 1：MAC 处于全双工模式，通过将流控制寄存器的位 2（RFE）置 1 来使能流控制。
- 条件 2：当流控制寄存器的 Bit 3（UP）置 1 时，接收到的帧的目的地址（DA）与特殊的多播地址或 EMACADDR0 匹配。
- 条件 3：接收帧的 Type 字段为 0x8808，OPCODE 字段为 0x0001。

DBF 此位置 1 后，AFM (Address Filtering Module) 模块会阻止所有传入的广播帧，并且所有其他过滤设置无效。当该位复位时，AFM 模块将传送所有接收到的广播帧。（读/写）

PAM 当该位置 1 时，该位表示所有接收到的具有多播目的地址（目的地址字段中的第一位为“1”）的帧都通过过滤。复位时，多播帧的过滤取决于 HMC 位。（读/写）

DAIF 当该位置 1 时，地址校验模块将对单播和多播帧的 DA 地址比较进行反向过滤。复位时，执行帧的正常过滤。（读/写）

PMODE 当该位置 1 时，地址过滤器模块将传送所有传入帧，而不管目的地址或源地址如何。PR (PRI_RATIO) 置 1 时，总是清除接收状态字的 SA 或 DA 滤波器失败状态位。（读/写）

Register 10.17. EMACMIIADDR_REG (0x1010)

(reserved)	MIIDEV	MIIREG	MIICSRCLK	MIIWRITE	MIIBUSY	
31 16	15 11	10 6	5 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00	0x00	0x00	0	0	Reset

MIIDEV 该字段表示 32 个 PHY 设备中的正被访问的哪一个。（读/写）

MIIREG 该字段在所选 PHY 设备中选择所需的地址寄存器。（读/写）

MIICSRCLK 该字段配置 CSR 时钟频率，支持以下两个取值，其他值保留。

- 4’b0000：APB 时钟频率为 80 MHz。MDC 时钟频率为 APB_CLK/42。
- 4’b0011：APB 时钟频率为 40 MHz。MDC 时钟频率为 APB_CLK/26。（读/写）

MIIWRITE 该字段置 1 时，表示使用 MII_DATA 的写操作。如果未置 1，则表示这是一个读取操作，即将数据放入 MII_DATA。（读/写）

MIIBUSY 该字段与 MIIREG、MII_DATA 配合使用。

在写入 MIIREG 和 MII_DATA 之前，该字段应读取逻辑 0（默认为空闲状态）。

要读写 MIIREG 或 MII_DATA，软件（用户）应将此字段置为 1。

在读或写访问正在进行时，MII_DATA 应保持有效（数据应保持不变），直到该字段被硬件 (MAC) 清零。

请注意在读写 MIIREG 或 MII_DATA 时，ESP32 MAC 不会收到 PHY 的 ACK 信息。（读/写设置/自动清除）

Register 10.18. EMACMIIDATA_REG (0x1014)

(reserved)	MII_DATA	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00000	Reset

MII_DATA 该字段包含在管理读操作之后从 PHY 读取的 16 位数据值，或在管理写操作之前要写入 PHY 的 16 位数据值。（读/写）

Register 10.19. EMACFC_REG (0x1018)

PAUSE_TIME	(reserved)	PLT	UPFD	RFCE	TFCE	FCBBA	
31 16	15 6	5 4	3	2	1	0	
0x00000	0 0 0 0 0 0 0 0 0 0	0x0	0	0	0	0	Reset

PAUSE_TIME 该字段保存在发送控制帧的暂停时间字段中使用的值。如果暂停时间位配置为与 MII 时钟域双重同步，则只有在目标时钟域中至少有四个时钟周期后，才能对此寄存器进行连续写入。(读/写)

PLT 该字段配置暂停帧的暂停定时器自动重传的阈值。阈值应始终小于 Bit [31:16] 中配置的暂停时间。例如，如果 PT = 100H（256 个时隙）并且 PLT = 01，则在发送第一个暂停帧之后的 228 个时隙时间（256-28）自动发送第二个暂停帧。以下表示不同值的阈值：(读/写)

- 2'b00：阈值是暂停时间减去 4 个时隙（PT-4 时隙时间）。
- 2'b01：阈值是暂停时间减去 28 个时隙（PT-28 时隙次数）。
- 2'b10：阈值是暂停时间减去 144 时隙时间（PT-144 时隙时间）。
- 2'b11：阈值是暂停时间减去 256 个时隙（PT-256 个时隙）。时隙时间被定义为在 MII 接口上传输 512 位（64 个字节）所用的时间。

UPFD 当一个暂停帧具有在 IEEE Std 802.3 中指定的唯一的多播地址时，处理暂停帧。当该位置 1 时，MAC 也可以检测站点的单播地址的暂停帧。这个单播地址应该在 EMACADDR0 高寄存器和 EMACADDR0 低寄存器中指定。当该位复位时，MAC 仅检测具有唯一多播地址的暂停帧。

RFCE 当该位置 1 时，MAC 解码接收到的暂停帧，并将发送器关闭指定的（暂停）时间。当该位复位时，关闭暂停帧的解码功能。(读/写)

TFCE 在全双工模式下，当该位置 1 时，MAC 使能流控制操作发送暂停帧。当该位复位时，MAC 中的流控制操作被禁止，MAC 不会传送任何暂停帧。在半双工模式下，当该位置 1 时，MAC 启用背压操作。当该位复位时，关闭背压功能。(读/写)

FCBBA 该位在全双工模式下启动暂停帧，如果 TFCE 位置 1，则在半双工模式下激活背压功能。在全双工模式下，在写入流控制寄存器之前，应将该位读为 1'b0。要启动暂停帧，应用程序必须将此位置为 1'b1。在传输控制帧的过程中，该位继续被设置为表示帧传输正在进行。暂停帧传输完成后，MAC 将该位复位为 1'b0。流控制寄存器不应写入，直到该位被清除。在半双工模式下，当该位置 1（并且 TFCE 置 1）时，背压由 MAC 确定。在背压期间，当 MAC 接收到一个新的帧时，发送器开始发送一个卡纸图案，导致碰撞。当 MAC 被配置为全双工模式时，BPA 被自动关闭。(读/写设置/自动清除）(FCB)/（读/写）(BPA/Backpressure Activate)

Register 10.20. EMACDEBUG_REG (0x1024)

(reserved)	MTLTSFFS	MTLTFNES	(reserved)	MTLTFWCS	MTLTFRCS	MACTP	MACTFCS	MACTPES	(reserved)	MTLRFFLS	(reserved)	MTLRFRCS	MTLRFWCAS	(reserved)	MACRFFCS	MACRPES	
31 26	25	24	23	22	21 20	19	18 17	16	15 10	9 8	7	6 5	4	3	2 1	0	
0 0 0 0 0 0	0	0	0	0	0x0	0	0x0	0	0 0 0 0 0 0	0x0	0	0x0	0	0	0x0	0	Reset

MTLTSFFS 该位为高电平时，该位表示 MTL TxStatus FIFO 已满。因此，MTL 不能接受任何更多的传输帧。（只读）

MTLTFNES 该位为高电平时，表示 MTL Tx FIFO 不为空，剩余一些数据用于传输。（只读）

MTLTFWCS 高电平时，该位表示 MTL Tx FIFO 写控制器激活，并正在将数据传输到 Tx FIFO。（只读）

MTLTFRCS 该字段表示 Tx FIFO 读取控制器的状态：（只读）

- 2’b00：空闲状态。
- 2’b01：READ 状态（将数据传输到 MAC 发射器）。
- 2’b10：等待来自 MAC 发送器的 TxStatus。
- 2’b11：写入接收的 TxStatus 或清空 Tx FIFO。

MACTP 当该位为高电平时，表示 MAC 发送器处于暂停状态（处于全双工模式），因此不传输任何帧。（只读）

MACTFCS 该字段表示 MAC 传输帧控制器模块的状态：（只读）

- 2’b00：空闲状态。
- 2’b01：等待前一帧或 IFG 或退避周期的状态结束。
- 2’b10：生成并发送暂停帧（全双工模式）。
- 2’b11：传输输入帧。

MACTPES 该位为高电平时，表示 MAC MII 传输协议引擎正在主动传输数据，不处于空闲状态。（只读）

MTLRFFLS 该字段表示 Rx FIFO 的填充级别的状态：（只读）

- 2’b00：Rx FIFO 为空。
- 2’b01：Rx FIFO 填充水平低于流量控制关闭阈值。
- 2’b10：Rx FIFO 填充水平高于流量控制激活阈值。
- 2’b11：Rx FIFO 满。

寄存器描述下一页继续。

Register 10.20. EMACDEBUG_REG (0x1024)

继上一页寄存器描述。

MTLRFRCS 该字段给出了 Rx FIFO 读取控制器的状态：（只读）

- 2'b00：空闲状态。
- 2'b01：读取帧数据。
- 2'b10：保留。
- 2'b11：刷新帧数据和状态。

MTLRFWCAS 该位为高电平时，该位表示 MTL Rx FIFO 写控制器激活，并正在将接收到的帧传送到 FIFO。（只读）

MACRFFCS 该位为高电平时，该字段表示 MAC 接收帧控制器模块的小型 FIFO 读写控制器的活动状态。RFCFCSTS[1] 表示 FIFO 读取控制器的状态。RFCFCSTS[0] 表示 FIFO 写入控制器的状态。（只读）

MACRPES 该位为高电平时，该位表示 MAC MII 接收协议引擎正在主动接收数据，不处于空闲状态。（只读）

Register 10.21. PMT_RWUFFR_REG (0x1028)

PMT_RWUFFR																																
31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

WKUPPKTFILTER （读/写）

- RWKPTR 等于 0 时，此寄存器代表过滤器 0 的唤醒帧的字节屏蔽位。bit[31] 必须为 0，bit[30:0] 为字节屏蔽位。如果相应的位置 1，那么对收到的帧的过滤器 0 的 offset+j（j=0–30）进行屏蔽，否则不屏蔽。
- RWKPTR 等于 1 时，此寄存器代表过滤器 1 的唤醒帧的字节屏蔽位。bit[31] 必须为 0，bit[30:0] 为字节屏蔽位。如果相应的位置 1，那么对收到的帧的过滤器 1 的 offset+j（j=0–30）进行屏蔽，否则不屏蔽。
- RWKPTR 等于 2 时，此寄存器代表过滤器 2 的唤醒帧的字节屏蔽位。bit[31] 必须为 0，bit[30:0] 为字节屏蔽位。如果相应的位置 1，那么对收到的帧的过滤器 2 的 offset+j（j=0–30）进行屏蔽，否则不屏蔽。
- RWKPTR 等于 3 时，此寄存器代表过滤器 3 的唤醒帧的字节屏蔽位。bit[31] 必须为 0，bit[30:0] 为字节屏蔽位。如果相应的位置 1，那么对收到的帧的过滤器 3 的 offset+j（j=0–30）进行屏蔽，否则不屏蔽。
- RWKPTR 等于 4 时，bit[3]/bit[11]/bit[19]/bit[27] 代表目的地址类型，1 代表多播包，0 代表单播包。bit[0]/bit[8]/bit[16]/bit[24] 分别代表了过滤器 0/1/2/3 的使能位。
- RWKPTR 等于 5 时，bit [7:0]/[15:8]/[23:16]/[31:24] 分别代表了过滤器 0/1/2/3 的屏蔽字节的起始 offset 位。
- RWKPTR 等于 6 时，bit [15:0] 为过滤器 0 的 CRC 计算值，bit [31:16] 为过滤器 1 的 CRC 计算值，从 offset 字节开始计算被屏蔽的字节的 CRC16。计算公式为

  $G(x) = x^{16} + x^{15} + x^2 + 1$

- RWKPTR 等于 7 时，bit [15:0] 为过滤器 2 的 CRC 计算值，bit [31:16] 为过滤器 3 的 CRC 计算值，从 offset 字节开始计算被屏蔽的字节的 CRC16。计算公式为

  $G(x) = x^{16} + x^{15} + x^2 + 1$

Register 10.22. PMT_CSR_REG (0x102C)

RWKFILTRST	(reserved)	RWKPTR	(reserved)	GLBLUCAST	(reserved)	RWKPRCVD	MGKPRCVD	(reserved)	RWKPKTEN	MGKPKTEN	PWRDWN	
31	30 29	28 24	23 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	00	0 0	0	0	0 0	0	0	0	Reset

RWKFILTRST 置 1 时，把寄存器 RWKPTR 复位为 0。(读/写设置/自动清除)

RWKPTR 最大值为 7，每个值代表寄存器 PMT_RWUFFR 的意义不同，详见 PMT_RWUFFR 说明。(只读)

GLBLUCAST 置 1 使能通过目的地址过滤器的单播包作为唤醒帧。(读/写)

RWKPRCVD 置 1 表明收到了远程唤醒帧的一次事件。(读/自动设置/读清除)

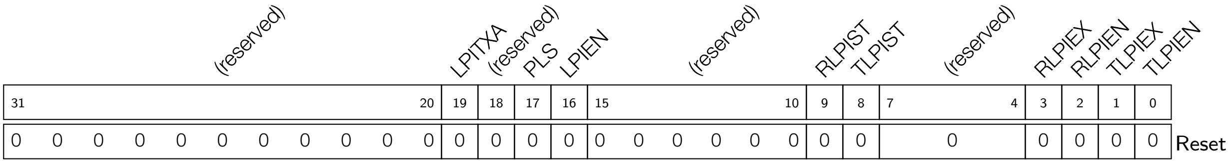
MGKPRCVD 置 1 表明收到了 magic 帧的一次事件。(读/自动设置/读清除)

RWKPKTEN 置 1 表明使能远程唤醒帧的功耗管理事件。(读/写)

MGKPKTEN 置 1 表明使能 magic 帧的功耗管理事件。(读/写)

PWRDWN 置 1 表明 EMAC 接收端放弃所有的接收到的帧，知道受到期望的 magic 帧或者远程唤醒帧。只有 MGKPKTEN, GLBLUCAST 或者 RWKPKTEN 置 1 时才有效。(读/写设置/自动清除)

Register 10.23. EMACLPI_CSR_REG (0x1030)



LPITXA 此位控制 EMAC 进入或退出 LPI 模式，当 LPITXA 和 LPIEN 置 1，并且所有的正常帧和暂停帧全部发送之后，EMAC 进入 LPI 模式，当应用层发送任何帧时退出 LPI 模式。当置 0 时，LPIEN 直接控制 EMAC 进入或退出 LPI 模式。（读/写）

PLS 此位表示 PHY 的链接状态，1 表示链接成功，0 表示链接失败。（读/写）

LPIEN 置 1 表明 EMAC 发送端进入 LPI 状态，置 0 表明 EMAC 退出 LPI 状态回复正常状态。当 LPITXA 位置 1 并且由于 EMAC 收到新的发送帧而使得退出 LPI 状态时，此位被置 0。（读/写/自动清除）

RLPIST 置 1 表明 EMAC 正在接收 LPI 序列。（读/写）

TLPIST 置 1 表明 EMAC 正在发送 LPI 序列。（读/写）

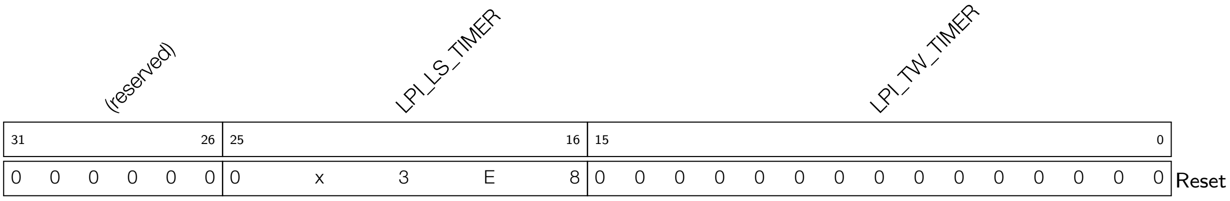
RLPIEX 置 1 表明 EMAC 接收端已经停止接收 LPI 序列，已经退出 LPI 状态，恢复到正常接受状态，该寄存器被读取时清 0。（读/自动设置/读清除）

RLPIEN 置 1 表明 EMAC 接收端受到 LPI 序列并且进入 LPI 状态，该寄存器被读取时清 0。（读/自动设置/读清除）

TLPIEX 置 1 表明 EMAC 发送端在寄存器 LPIEN 被清 0，并且 LPI_TW_TIMER 过期之后已经退出 LPI 状态，该寄存器被读取时清 0。（读/自动设置/读清除）

TLPIEN 置 1 表明 EMAC 的寄存器 LPIEN 置 1 后发送端进入 LPI 状态，该寄存器被读取时清 0。（读/自动设置/读清除）

Register 10.24. EMACLPITIMERSCONTROL_REG (0x1034)



LPI_LS_TIMER 此域代表 LPI 序列发送给物理层之前物理层建立链接的最小时间（毫秒）。根据 IEEE 标准默认值为 1 秒。（读/写）

LPI_TW_TIMER 此域代表 EMAC 在发送正常帧之前停止发送 LPI 帧的最小时间（毫秒）。在这个计时器到期之后，寄存器 TLPIEX 应该置 1。（读/写）

Register 10.25. EMACINTS_REG (0x1038)

(reserved)	LPIINTS	(reserved)	(reserved)	PMTINTS	(reserved)	
31 11	10	9	8 5	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0	0	0 0 0	Reset

LPIINTS 节能以太网功能启用时，该位用于设置 MAC 发射机或接收机进入或退出 LPI 状态。该位在读取寄存器（LPI 控制和状态寄存器）的 Bit[0] 时清零。(只读)

PMTINTS 在掉电模式下收到一个魔术包或远程唤醒帧时，该位置 1（见 PMT 控制和状态寄存器的 Bit[5] 和 Bit[6]）。由于对 PMT 控制和状态寄存器的读取操作，Bit[6:5] 被清零时，该位被清零。只有在核心配置期间选择可选的 PMT 模块时，该位才有效。(只读)

Register 10.26. EMACINTMASK_REG (0x103C)

(reserved)	LPIINTMASK	(reserved)	(reserved)	PMTINTMASK	(reserved)	
31 11	10	9	8 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0 0 0	0	0 0 0	Reset

LPIINTMASK 置 1 时，由于中断状态寄存器中 LPI 中断状态位的设置，该位禁止中断信号的触发。(读/写)

PMTINTMASK 置 1 时，由于中断状态寄存器中 PMT 中断状态位的设置，该位禁止中断信号的置位。(读/写)

Register 10.27. EMACADDR0HIGH_REG (0x1040)

ADDRESS_ENABLE0	(reserved)	MAC_ADDRESS0_HI	
31	30 16	15 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE0 该位始终置 1。(只读)

MAC_ADDRESS0_HI 该字段包含第一个 6 字节 MAC 地址的高 16 位 [47:32]。MAC 使用此字段过滤接收到的帧，并将 MAC 地址插入到传输流控制（暂停）帧中。(读/写)

Register 10.28. EMACADDR0LOW_REG (0x1044)

31 0	
0x0FFFFFFFF	Reset

EMACADDR0LOW_REG 该字段包含第一个 6 字节 MAC 地址的低 32 位。MAC 被用于过滤接收到的帧，并将 MAC 地址插入到传输流控制（暂停）帧中。(读/写)

Register 10.29. EMACADDR1HIGH_REG (0x1048)

ADDRESS_ENABLE1	SOURCE_ADDRESS	MASK_BYTE_CONTROL	(reserved)	MAC_ADDRESS1_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE1 当该位置 1 时，地址过滤器模块使用第二个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。(读/写)

SOURCE_ADDRESS 当该位置 1 时，EMACADDR1 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR1 [47:0] 与接收帧的 DA 字段进行比较。(读/写)

MASK_BYTE_CONTROL 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR1 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR1 寄存器高 [15:8]。
- Bit[28]：EMACADDR1 寄存器高 [7:0]。
- Bit[27]：EMACADDR1 寄存器低 [31:24]。
- Bit[24]：EMACADDR1 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。(读/写)

MAC_ADDRESS1_HI 该字段包含第二个 6 字节 MAC 地址的高 16 位 [47:32]。(读/写)

Register 10.30. EMACADDR1LOW_REG (0x104C)

31 0	
0x0FFFFFFFF	Reset

EMACADDR1LOW_REG 该字段包含第二个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。(读/写)

Register 10.31. EMACADDR2HIGH_REG (0x1050)

ADDRESS_ENABLE2	SOURCE_ADDRESS2	MASK_BYTE_CONTROL2	(reserved)	MAC_ADDRESS2_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE2 当该位置 1 时，地址过滤器模块使用第三个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。（读/写）

SOURCE_ADDRESS2 当该位置 1 时，EMACADDR2 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR2 [47:0] 与接收帧的 DA 字段进行比较。（读/写）

MASK_BYTE_CONTROL2 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR2 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR2 寄存器高 [15:8]。
- Bit[28]：EMACADDR2 寄存器高 [7:0]。
- Bit[27]：EMACADDR2 寄存器低 [31:24]。
- Bit[24]：EMACADDR2 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。（读/写）

MAC_ADDRESS2_HI 该字段包含第三个 6 字节 MAC 地址的高 16 位 [47:32]。（读/写）

Register 10.32. EMACADDR2LOW_REG (0x1054)

31 0	
0x0FFFFFFFF	Reset

EMACADDR2LOW_REG 该字段包含第三个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。（读/写）

Register 10.33. EMACADDR3HIGH_REG (0x1058)

ADDRESS_ENABLE3	SOURCE_ADDRESS3	MASK_BYTE_CONTROL3	(reserved)	MAC_ADDRESS3_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE3 当该位置 1 时，地址过滤器模块使用第四个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。(读/写)

SOURCE_ADDRESS3 当该位置 1 时，EMACADDR3 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR3 [47:0] 与接收帧的 DA 字段进行比较。(读/写)

MASK_BYTE_CONTROL3 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR3 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR3 寄存器高 [15:8]。
- Bit[28]：EMACADDR3 寄存器高 [7:0]。
- Bit[27]：EMACADDR3 寄存器低 [31:24]。
- Bit[24]：EMACADDR3 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。(读/写)

MAC_ADDRESS3_HI 该字段包含第四个 6 字节 MAC 地址的高 16 位 [47:32]。(读/写)

Register 10.34. EMACADDR3LOW_REG (0x1060)

31 0	
0x0FFFFFFFF	Reset

EMACADDR3LOW_REG 该字段包含第四个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。(读/写)

Register 10.35. EMACADDR4HIGH_REG (0x1064)

ADDRESS_ENABLE4	SOURCE_ADDRESS4	MASK_BYTE_CONTROL4	(reserved)	MAC_ADDRESS4_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE4 当该位置 1 时，地址过滤器模块使用第五个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。（读/写）

SOURCE_ADDRESS4 当该位置 1 时，EMACADDR4 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR4 [47:0] 与接收帧的 DA 字段进行比较。（读/写）

MASK_BYTE_CONTROL4 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR4 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR4 寄存器高 [15:8]。
- Bit[28]：EMACADDR4 寄存器高 [7:0]。
- Bit[27]：EMACADDR4 寄存器低 [31:24]。
- Bit[24]：EMACADDR4 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。（读/写）

MAC_ADDRESS4_HI 该字段包含第五个 6 字节 MAC 地址的高 16 位 [47:32]。（读/写）

Register 10.36. EMACADDR4LOW_REG (0x1068)

31 0	
0x0FFFFFFFF	Reset

EMACADDR4LOW_REG 该字段包含第五个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。（读/写）

Register 10.37. EMACADDR5HIGH_REG (0x106C)

ADDRESS_ENABLE5	SOURCE_ADDRESS5	MASK_BYTE_CONTROL5	(reserved)	MAC_ADDRESS5_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE5 当该位置 1 时，地址过滤器模块使用第六个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。(读/写)

SOURCE_ADDRESS5 当该位置 1 时，EMACADDR5 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR5 [47:0] 与接收帧的 DA 字段进行比较。(读/写)

MASK_BYTE_CONTROL5 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR5 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR5 寄存器高 [15:8]。
- Bit[28]：EMACADDR5 寄存器高 [7:0]。
- Bit[27]：EMACADDR5 寄存器低 [31:24]。
- Bit[24]：EMACADDR5 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。(读/写)

MAC_ADDRESS5_HI 该字段包含第六个 6 字节 MAC 地址的高 16 位 [47:32]。(读/写)

Register 10.38. EMACADDR5LOW_REG (0x1070)

31 0	
0x0FFFFFFFF	Reset

EMACADDR5LOW_REG 该字段包含第六个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。(读/写)

Register 10.39. EMACADDR6HIGH_REG (0x1074)

ADDRESS_ENABLE6	SOURCE_ADDRESS6	MASK_BYTE_CONTROL6	(reserved)	MAC_ADDRESS6_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE6 当该位置 1 时，地址过滤器模块使用第七个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。(读/写)

SOURCE_ADDRESS6 当该位置 1 时，EMACADDR6 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR6 [47:0] 与接收帧的 DA 字段进行比较。(读/写)

MASK_BYTE_CONTROL6 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR6 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR6 寄存器高 [15:8]。
- Bit[28]：EMACADDR6 寄存器高 [7:0]。
- Bit[27]：EMACADDR6 寄存器低 [31:24]。
- Bit[24]：EMACADDR6 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。(读/写)

MAC_ADDRESS6_HI 该字段包含第七个 6 字节 MAC 地址的高 16 位 [47:32]。(读/写)

Register 10.40. EMACADDR6LOW_REG (0x1078)

31 0	
0x0FFFFFFFF	Reset

EMACADDR6LOW_REG 该字段包含第七个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。(读/写)

Register 10.41. EMACADDR7HIGH_REG (0x107C)

ADDRESS_ENABLE7	SOURCE_ADDRESS7	MASK_BYTE_CONTROL7	(reserved)	MAC_ADDRESS7_HI	
31	30	29 24	23 16	15 0	
0	0	0x00	0 0 0 0 0 0 0 0	0x0FFFF	Reset

ADDRESS_ENABLE7 当该位置 1 时，地址过滤器模块使用第八个 MAC 地址进行完美过滤。当该位复位时，地址过滤器模块将忽略用于过滤的地址。(读/写)

SOURCE_ADDRESS7 当该位置 1 时，EMACADDR7 [47:0] 与接收帧的 SA 字段进行比较。当该位复位时，使用 EMACADDR7 [47:0] 与接收帧的 DA 字段进行比较。(读/写)

MASK_BYTE_CONTROL7 这些位是屏蔽控制位，用于比较每个 MAC 地址字节。当置高时，MAC 不会将接收到的 DA 或 SA 的相应字节与 EMACADDR7 寄存器的内容进行比较。每一位控制字节的掩码如下：

- Bit[9]：EMACADDR7 寄存器高 [15:8]。
- Bit[28]：EMACADDR7 寄存器高 [7:0]。
- Bit[27]：EMACADDR7 寄存器低 [31:24]。
- Bit[24]：EMACADDR7 寄存器低 [7:0]。

可以通过屏蔽地址的一个或多个字节来过滤一组地址（称为组地址过滤）。(读/写)

MAC_ADDRESS7_HI 该字段包含第八个 6 字节 MAC 地址的高 16 位 [47:32]。(读/写)

Register 10.42. EMACADDR7LOW_REG (0x1070)

31 0	
0x0FFFFFFFF	Reset

EMACADDR7LOW_REG 该字段包含第八个 6 字节 MAC 地址的低 32 位。在初始化之后，该寄存器应该被配置，因为该字段的内容是不确定的。(读/写)

Register 10.43. EMACWDOGTO_REG (0x10DC)

(reserved)	PWDOGEN	(reserved)	WDOGTO	
31 17	16	15 14	13 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0 0	0x0000	Reset

PWDOGEN 当该位置 1 且 EMACCONFIG_REG Bit[23]（WD）复位时，WTO 字段（Bit [13:0]）用作接收帧的看门狗超时。当该位清零时，接收帧的 EMACCONFIG_REG Bit[23]（WD）和 Bit[20]（JE）的设置控制。（读/写）

WDOGTO 当 Bit[16]（PWE）置 1 且 MAC 配置寄存器的 Bit[23]（WD）复位时，该字段用作接收帧的看门狗超时。如果接收到的帧的长度超过了这个字段的值，那么这个帧将被终止并被声明为错误帧。（读/写）

Register 10.44. EMAC_EX_CLKOUT_CONF_REG (0x0800)

(reserved)	EMAC_CLK_OUT_H_DIV_NUM	EMAC_CLK_OUT_DIV_NUM	
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	0x02	0x04	Reset

EMAC_CLK_OUT_H_DIV_NUM 当使用 RMII PHY 时，RMII CLK 使用内部 APLL CLK，进行半整数分频。（读/写）

EMAC_CLK_OUT_DIV_NUM 当使用 RMII PHY 时，RMII CLK 使用内部的 APLL CLK，进行整数分频。（读/写）

Register 10.45. EMAC_EX_OSCCLK_CONF_REG (0x0804)

(reserved)	EMAC_OSC_CLK_SEL	EMAC_OSC_H_DIV_NUM_100M	EMAC_OSC_DIV_NUM_100M	EMAC_OSC_H_DIV_NUM_10M	EMAC_OSC_DIV_NUM_10M	
31 25	24	23 18	17 12	11 6	5 0	
0 0 0 0 0 0 0	0	0	1	9	19	Reset

EMAC_OSC_CLK_SEL 该位决定当使用 RMII PHY 时，以太网是否使用外部 PHY 输出时钟作为 RMII CLK。该位置 1 时，使用外部晶振 CLK；该位置 0 时，使用内部 APLL CLK。(读/写)

EMAC_OSC_H_DIV_NUM_100M RMII/MII 半整数分频寄存器，当 EMAC_EX_CLKOUT_CONF 分频寄存器配置为 100 MHz 时。(读/写)

EMAC_OSC_DIV_NUM_100M RMII/MII 整数分频寄存器，当 EMAC_EX_CLKOUT_CONF 分频寄存器配置为 100 MHz 时。(读/写)

EMAC_OSC_H_DIV_NUM_10M RMII/MII 半整数分频寄存器，当 EMAC_EX_CLKOUT_CONF 分频寄存器配置为 10 MHz 时。(读/写)

EMAC_OSC_DIV_NUM_10M RMII/MII 整数分频寄存器，当 EMAC_EX_CLKOUT_CONF 分频寄存器配置为 10 MHz 时。(读/写)

Register 10.46. EMAC_EX_CLK_CTRL_REG (0x0808)

(reserved)	EMAC_CLK_EN	EMAC_MII_CLK_RX_EN	EMAC_MII_CLK_TX_EN	(reserved)	EMAC_INT_OSC_EN	EMAC_EXT_OSC_EN	
31 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

EMAC_MII_CLK_RX_EN Ethernet RX CLK 使能位。(读/写)

EMAC_MII_CLK_TX_EN Ethernet TX CLK 使能位。(读/写)

EMAC_INT_OSC_EN 在 RMII PHY 模式下使用内部 APLL CLK。(读/写)

EMAC_EXT_OSC_EN 在 RMII PHY 模式下使用外部 XTAL CLK。(读/写)

Register 10.47. EMAC_EX_PHYINF_CONF_REG (0x080c)

(reserved)	EMAC_PHY_INTF_SEL	(reserved)	
31 16	15 13	12 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EMAC_PHY_INTF_SEL 选择的 PHY 接口类型。0x0 表示 PHY 为 MII，0x4 表示 PHY 为 RMII。（读/写）

Register 10.48. EMAC_PD_SEL_REG (0x0810)

(reserved)	EMAC_RAM_PD_EN	
31 2	1 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

EMAC_RAM_PD_EN 以太网 RAM 掉电使能信号，Bit 0 用于 TX SRAM，Bit 1 用于 RX SRAM。置 1 表示掉电。（读/写）

11 I2C 控制器 (I2C)

11.1 概述

I2C (Inter-Integrated Circuit) 总线用于使 ESP32 和多个外部设备进行通信。多个外部设备可以共用一个 I2C 总线。

11.2 主要特性

I2C 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 寻址
- 支持关闭 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能

11.3 I2C 功能描述

11.3.1 I2C 简介

I2C 是一个两线总线，由 SDA 线和 SCL 线构成。这些线设置为漏极开漏输出。因此，I2C 总线上可以挂载多个外设，通常是和一个或多个主机以及一个或多个从机。主机通过总线访问从机。

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，回发送一个停止标志：在 SCL 为高电平时，拉高 SDA 线。

ESP32 I2C 控制器可以处理 I2C 协议，腾出处理器核用于其它任务。

11.3.2 I2C 架构

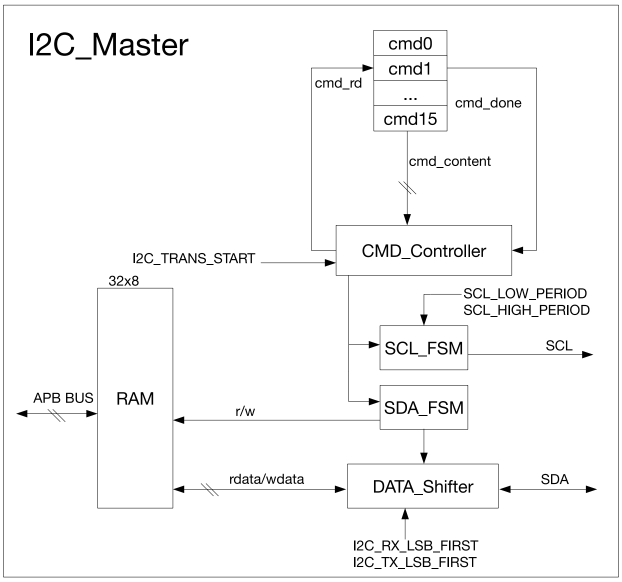


图 11-1. I2C Master 基本架构

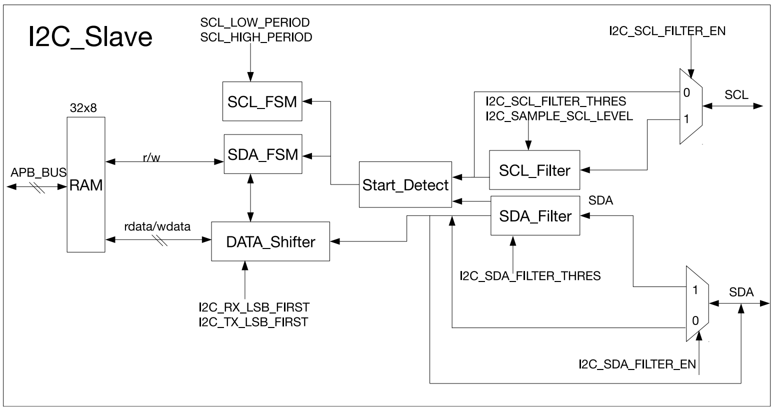


图 11-2. I2C Slave 基本架构

I2C 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 11-1 为 I2C Master 基本架构图，图 11-2 为 I2C Slave 基本架构图。从图中可知，I2C 控制器内部主要有以下几个单元：

- RAM：大小为 32 x 8 bit，直接映射到 CPU 内核的地址上，首地址为 (REG_I2C_BASE+0x100)，I2C 数据的每一个字节占据一个 word 的存储地址（因此，首字节在 +0x100, 第二字节在 +0x104, 第三字节在 +0x108，以此类推）。用户需要置位 I2C_NONFIFO_EN 寄存器。
- 16 个命令寄存器 (cmd0 ~ cmd15) 以及一个 CMD_Controller：用于 I2C Master 控制数据传输过程。I2C 控制器每次执行一个命令。
- SCL_FSM：用于控制 SCL 时钟，I2C_SCL_HIGH_PERIOD_REG 以及 I2C_SCL_LOW_PERIOD_REG 寄存器用于配置 SCL 的频率和占空比。
- SDA_FSM：用于控制 SDA 数据线。
- DATA_Shifter：用于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。

- SCL_Filter 以及 SDA_Filter：用于 I2C_Slave 滤除输入噪声。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。滤波器可以滤除脉宽低于 I2C_SCL_FILTER_THRES 以及 I2C_SDA_FILTER_THRES 的毛刺。

11.3.3 I2C 总线时序

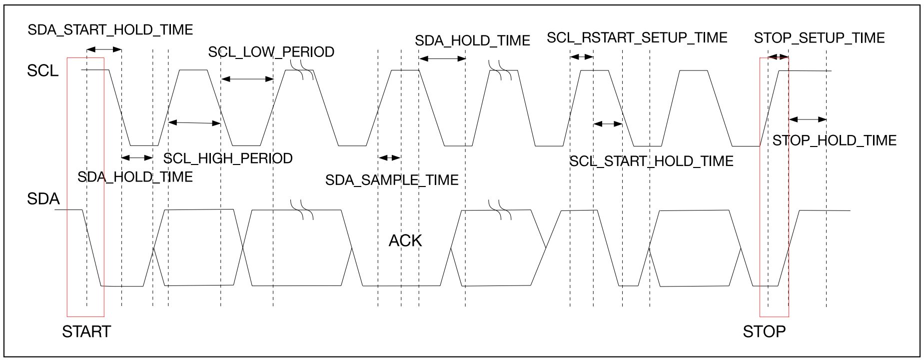


图 11-3. I2C 时序图

图 11-3 为 I2C 的时序图。当 I2C 控制器工作于 Master 模式，SCL 为输出信号。当 I2C 控制器工作于 Slave 模式，SCL 为输入信号。分配给 I2C_SDA_HOLD_REG 和 I2C_SDA_SAMPLE_REG 的值仍适用于 Slave 模式。用户需要根据 I2C Master 的特性合理配置 I2C_SDA_HOLD _TIME 以及 I2C_SDA_SAMPLE_TIME 寄存器，否则可能导致 I2C Slave 不能正确接收数据。当 SCL 配置成直接输出模式时，SCL 的输出高低电平周期数如表 11-1 所示，SCL 的输出频率为 f_{scl}。

表 11-1. SCL 频率配置

I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	SCL_Low_Level_Cycles	SCL_High_Level_Cycles
0	Don’ t care	I2C_SCL_LOW_PERIOD+1	I2C_SCL_HIGH_PERIOD+7
1	[0,2]		I2C_SCL_HIGH_PERIOD+8
	[3,7]		I2C_SCL_HIGH_PERIOD+6+I2C_SCL_FILTER_THRES

$$f_{\mathrm{scl}} = \frac{80\ \mathrm{MHz}}{\mathrm{SCL\_Low\_Level\_Cycles} + \mathrm{SCL\_High\_Level\_Cycles}}$$

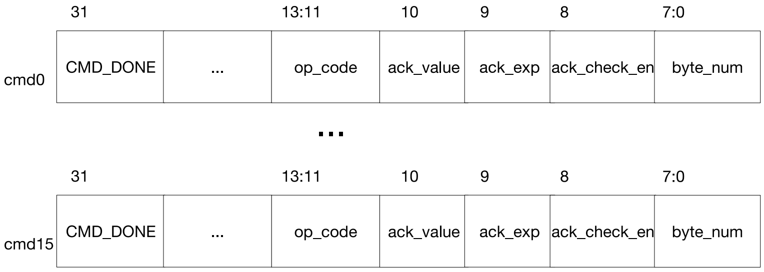
根据 I2C 协议规定，一次数据传输始于 START 位结束于 STOP 位。数据按照每次一个字节进行传输，每个传输的字节后有一个 ACK 位。接收数据方通过回 ACK 的方式告知发送方继续传输数据。接收数据方也可以通过不回 ACK 的方式发送方停止传输数据。

I2C 控制器的 START 位、STOP 位、数据保持时间、数据采样时间均可以通过图 11-3 中所示的寄存器进行配置。

注意：将 SCL 的 pad 配置成开漏方式时，SCL 从低电平转向高电平的时间会变长，这个转变时间由 SCL 线上的上拉电阻以及电容共同决定。开漏模式下，SCL 的输出频率会偏低。

11.3.4 I2C cmd 结构

命令寄存器只在 I2C Master 中有效，其内部结构如图 11-4 所示。



	31		13:11	10	9	8	7:0
cmd0	CMD_DONE	...	op_code	ack_value	ack_exp	ack_check_en	byte_num

...

	31		13:11	10	9	8	7:0
cmd15	CMD_DONE	...	op_code	ack_value	ack_exp	ack_check_en	byte_num

图 11-4. I2C 命令寄存器结构

CMD_DONE：软件可以通过读取每条命令的 CMD_DONE 位来判断一条命令是否执行完毕。

op_code：op_code 用于命令编码，I2C 控制器支持 4 种命令。

- RSTART：op_code 等于 0 时为 RSTART 命令，该命令用于控制 I2C 协议中 START 位以及 RESTART 位的发送。
- WRITE：op_code 等于 1 时为 WRITE 命令，该命令表示当前 Master 将发送数据。
- READ：op_code 等于 2 时为 READ 命令，该命令表示当前 Master 将要接收数据。
- STOP：op_code 等于 3 时为 STOP 命令，该命令用于控制协议中 STOP 位的发送。
- END：op_code 等于 4 时为 END 命令，该命令用于 master 模式下连续发送数据。主要实现方式为关闭 SCL 时钟，当数据准备完毕，继续上次传输。

一次完整的命令序列始于 RSTART 命令，结束于 STOP 命令。

ack_value：当接收数据时，在字节被接收后，该位用于表示接收方将发送一个 ACK 位。

ack_exp：该位用于设置发送方期望的 ACK 值。

ack_check_en：该位用于控制发送方是否对 ACK 位进行检测。1：检测 ACK 值；0：不检测 ACK 值。

byte_num：该寄存器用于说明读写数据的数据长度（单位字节），最大为 255，最小为 1。RSTART、STOP、END 命令中 byte_num 无意义。

11.3.5 I2C 主机写入从机

为了便于描述，下文所有图示中的 I2C Master 和 Slave 都假定为 ESP32 I2C 外设控制器。

图 11-5 为 I2C Master 写 N 个字节数据到 I2C Slave 的配置图。根据 I2C 协议，第一个字节为 I2C Slave 地址，如图中 RAM 所示，第一个数据为（Slave 7-bit 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 框中包含了用于运行的一系列命令。

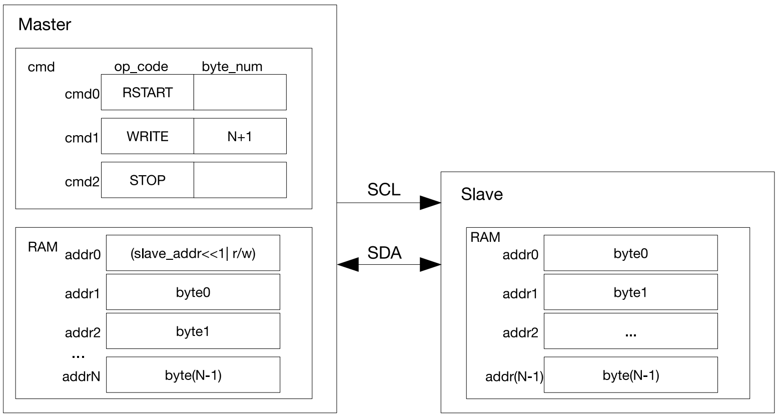


图 11-5. I2C Master 写 7-bit 地址 Slave

要使 I2C Master 开始传输数据，总线不能被占用，也就是说 SCL 线不能被其他 Master 或者 Slave 拉低，此时需要等待 SCL 恢复到高电平，才可以进行数据传输。在 I2C Master 中准备好 cmd 以及数据，置位 I2C_TRANS_START 寄存器即开始一次数据传输。I2C Master 首先根据 RSTART 命令发送一个 START 位，然后根据 WRITE 命令从 RAM 的首地址开始取出 N+1 个字节并发送给从机，第一个字节为地址。当 I2C Master 发送数据量超过 I2C_NONFIFO_TX_THRES 时，会产生 I2C_TX_SEND_EMPTY_INT 中断，软件在检测到该中断后，可以通过读取 I2C Master RXFIFO_ST_REG 寄存器中 TXFIFO_END_ADDR 得到已发送数据在 RAM 中的末地址，从而更新 RAM 中的旧数据。TXFIFO_END_ADDR 寄存器会在每次 I2C_TX_SEND_EMPTY_INT 中断或 I2C_TRANS_COMPLETE_INT 中断产生时更新。

当 I2C Master WRITE 命令中的 ack_check_en 配置为 1 时，I2C 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 不一致时，I2C Master 会产生 I2C_ACK_ERR_INT 中断并停止发送数据。

数据传输过程中，I2C Master 的 SCL 为高电平，SDA 输出值与 SDA 输入值不等时，则 I2C Master 会产生 I2C_ARBITRATION_LOST_INT 中断。当 I2C Master 完成一次数据传输后，会产生 I2C_TRANS_COMPLETE_INT 中断。

I2C Slave 在检测到 I2C Master 发送的 START 位之后，开始接收地址并进行地址匹配，当 I2C Slave 接收的地址与其 I2C_SLAVE_ADDR 寄存器值不匹配时，I2C Slave 停止接收数据。当地址匹配后，I2C Slave 将接下来接收的数据按照顺序存储到 RAM 中。当 I2C Slave 接收的数据超过 I2C_NONFIFO_RX_THRES 时，会产生 I2C_RX_REC_FULL_INT 中断，软件在检测到该中断后，可以通过读取 I2C Slave RXFIFO_ST_REG 寄存器中 RXFIFO_START_ADDR 和 RXFIFO_END_ADDR，得到接收数据在 RAM 中的始末地址，从而取出数据进行处理。RXFIFO_START_ADDR 寄存器在一次传输中只更新一次值，而 RXFIFO_END_ADDR 寄存器在每次 I2C_RX_REC_FULL_INT 中断或 I2C_TRANS_COMPLETE_INT 中断产生时都会更新。

在不使用 END 命令的情况下，I2C Master 一次最多发送 (14*255-1) 个有效数据给 7-bit 地址的 I2C Slave，其 cmd 配置为 1 个 RSTART + 14 个 WRITE + 1 个 STOP。

需要注意总线上的几种特殊情况：

- I2C Master 在发送 STOP 位时，若因为 SDA 被其他设备拉低导致发送不了 STOP 位，则需要复位 I2C Master。
- I2C Master 在发送 START 位时，若因为 SDA 或者 SCL 被其他设备拉低导致发送不了 START 位，则需要复位 I2C Master。建议软件使用超时时间来执行重置。

- 数据传输过程中，SDA 被 I2C Slave 拉为低电平，此时 I2C Master 只需给 Slave 至多九个 SCL 时钟即可释放 SDA 线。

需要注意的是，总线上其他 Master 或者 Slave 的操作可能与 ESP32 I2C 外设有所不同，具体请参考各个 I2C 设备的技术规格书。

ESP32 I2C 控制器默认使用 7-bit 地址，也可以使用 10-bit 地址。在主机中，在发送完第一个地址位之后发送第二个 I2C 地址位即可完成。在从机中，可以通过配置 I2C_ADDR_10BIT_EN 寄存器开启 10-bit 地址。I2C_SLAVE_ADDR 用于配置 I2C Slave 地址。图 11-6 为 I2C Master 写 N 个字节到 10-bit 地址 I2C Slave 的配置图，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。

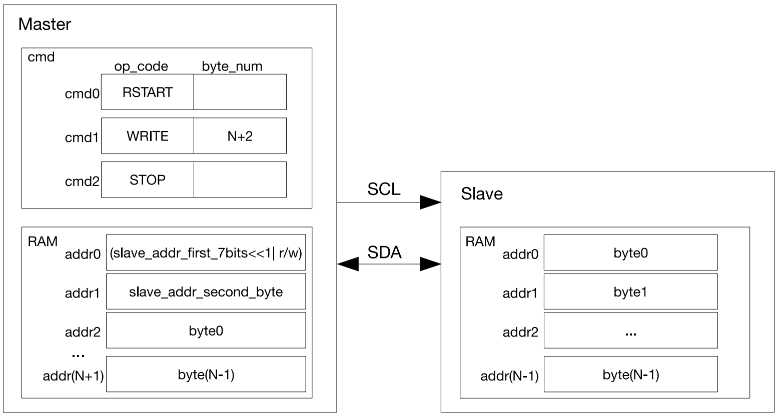


图 11-6. I2C Master 写 10-bit 地址 Slave

在不使用 END 命令的情况下，I2C Master 一次最多发送 (14*255-2) 个有效数据给 10-bit 地址的 I2C Slave。

许多 I2C Slave 具有特殊寄存器。I2C Master 通过向从机发送 1 个寄存器地址可以访问从机中指定的寄存器。ESP32 I2C Slave 控制器具有支持上述操作的硬件。

通过置位 I2C_FIFO_ADDR_CFG_EN 来开启控制 I2C Slave 数据存储位置的功能。如图 11-7 所示，I2C Slave 将接收到的数据 byte0 ~ byte(N-1) 从 Slave RAM 中的 addrM 开始依次存储。在该模式下，I2C Slave 一次传输最多接收 32 字节有效数据，当 I2C Master 需要发送的有效数据超过 32 字节时，需要分成多次发送。

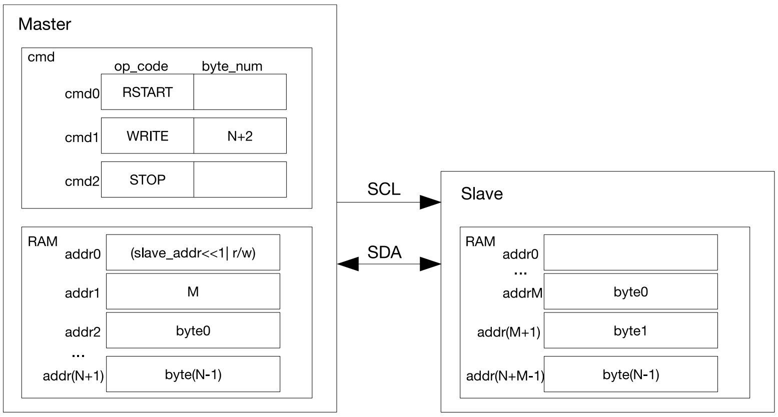


图 11-7. I2C Master 写 7-bit 地址 Slave 的 M 地址 RAM

当一次传输的数据较多，超出 14 个读/写 cmd 支持的最大数据量时，可以通过 END 命令进行分段传输。图

11-8 所示为 I2C Master 分成三段写 Slave。首先配置 I2C Master 的命令如第一段所示，并且在 Master 的 RAM 中准备好数据，置位 I2C_TRANS_START，I2C Master 即开始数据传输。在执行到 END 命令后，I2C Master 会关闭 SCL 时钟，并将 SCL 线拉低来防止其他设备占用 I2C 总线。此时控制器产生 I2C_END_DETECT_INT 中断。

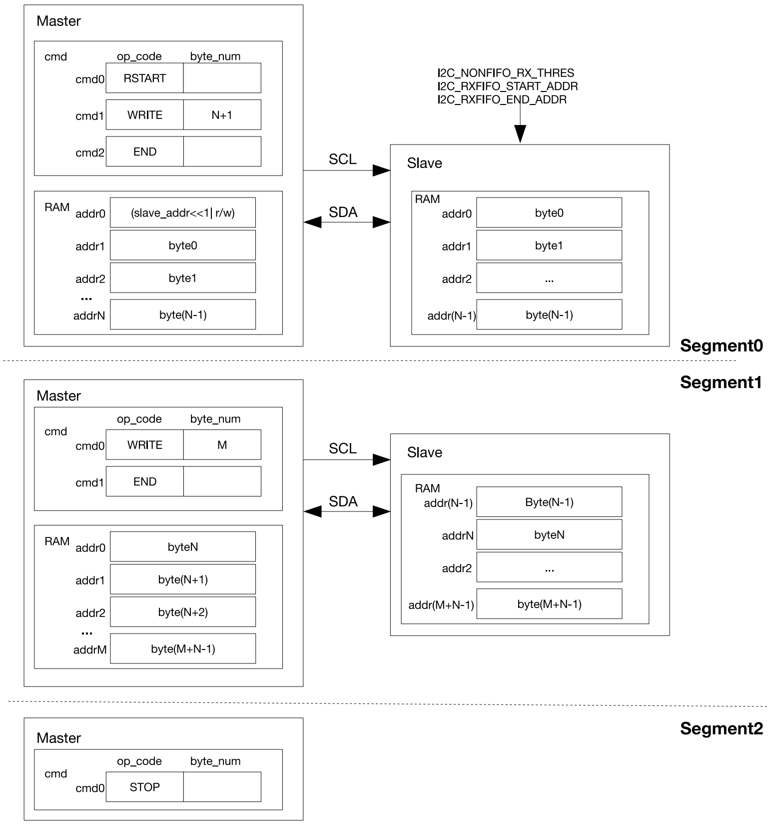


图 11-8. I2C Master 分段写 7-bit 地址 Slave

在检测到 I2C_END_DETECT_INT 中断后，软件可以更新 cmd 以及 RAM 中的内容如第二段所示，并清除 I2C_END_DETECT_INT 中断。置位 I2C_TRANS_START 后，I2C Master 可以继续发送数据。当不需要再传输数据时，在检测到 I2C 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，I2C Master 即产生 STOP 位，从而停止传输。

注意：

- 在两个分段之间，I2C 总线上的其他 Master 设备不会有时间占用总线。只有在发送了 STOP 信号后总线才会被释放。
- 当有超过 3 段时，前一段 END 在 cmd 的位置不能被后一段更新成其他命令。

11.3.6 I2C 主机读取从机

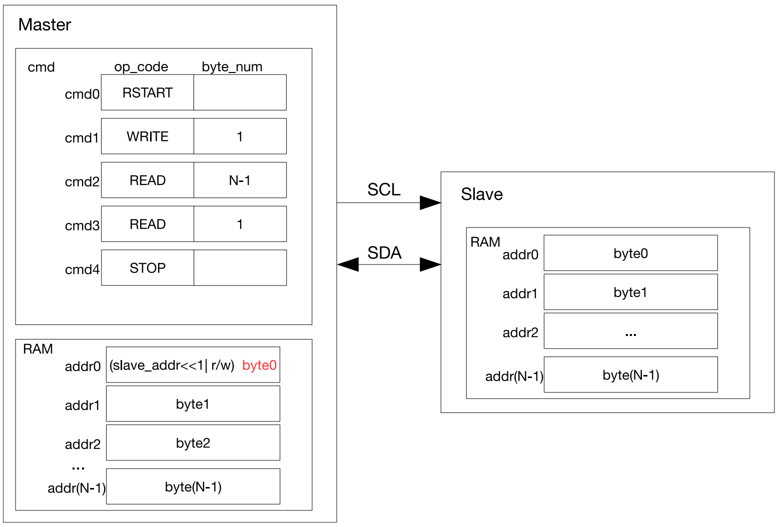


图 11-9. I2C Master 读 7-bit 地址 Slave

图 11-9 为 I2C Master 从 7-bit 地址 I2C Slave 读取 N 个字节数据的配置图。首先 I2C Master 需要将 I2C Slave 的地址发送出去，所以 cmd1 为 WRITE。该命令发送的字节是一个 I2C Slave 地址以及其读写标志位，其中，1 表示这是一个读操作。I2C Slave 在匹配好地址之后即开始发送数据给 I2C Master。I2C Master 根据 READ 命令中的 ack_value 在每个接收的数据之后回复 ACK。图 11-9 中 READ 分成两次，I2C 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不回复 ACK ，实际使用时可以根据需要进行配置。在存储接收的数据时，I2C Master 从 RAM 的首地址开始存储，图中红色 byte0 会覆盖（Slave 地址 +1-bit 读写位）。

在不使用 END 命令的情况下，I2C Master 一次最多从 I2C Slave 读取 (13*255) 个有效数据，其 cmd 配置为 RSTART + 1 个 WRITE + 13 个 READ + 1 个 STOP。

图 11-10 为 I2C Master 从 10-bit 地址 I2C Slave 中读取数据的配置图。首先在 I2C Slave 中置位 I2C_SLAVE_ADDR_10BIT_EN 以及在 I2C Slave 的 RAM 中准备好待发送的数据，然后在 I2C Master 的 RAM 中准备好 I2C Slave 2 个字节的 10-bit 地址。置位 I2C Master 的 I2C_TRANS_START，即可开启一次读传输。

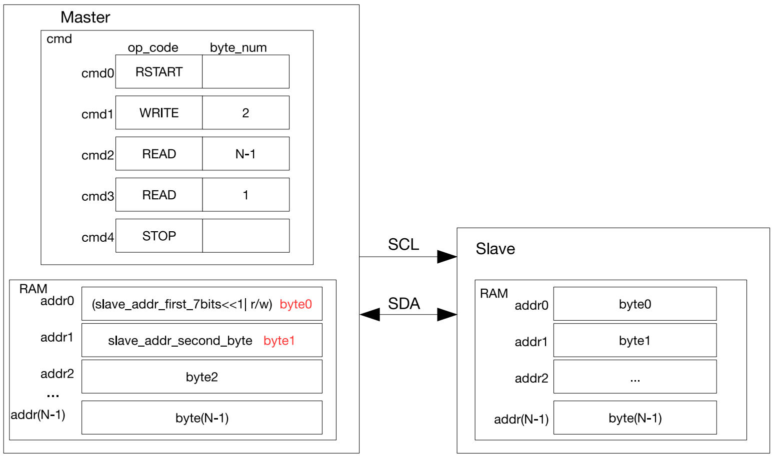


图 11-10. I2C Master 读 10-bit 地址 Slave

图 11-11 为 I2C Master 从 I2C Slave 中指定地址读取数据的配置图。首先在 I2C Slave 中置位 I2C_FIFO_ADDR_CFG_EN 并在其 RAM 中准备好待发送的数据。然后在 I2C Master 中准备好 I2C Slave 的地址以及其指定的寄存器地址 M 。置位 I2C Master 的 I2C_TRANS_START，I2C Slave 会将从 RAM 中 M 地址开始取 N 个数据发送给 I2C Master。

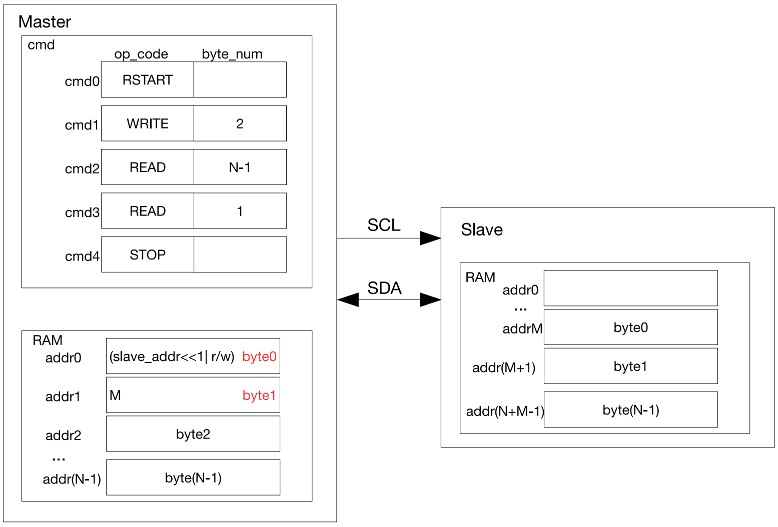


图 11-11. I2C Master 从 7-bit 地址 Slave 的 M 地址读取 N 个数据

图 11-12 为 I2C Master 通过 END 命令分三段从 I2C Slave 读取 N+M 个数据的配置图。首先配置 cmd 如第一段所示并在 Slave 的 RAM 中准备好数据，置位 I2C_TRANS_START，I2C 即开始工作。当执行到 END 命令时，I2C Master 可以更新 cmd 如第二段所示，并且清零其对应的 I2C_END_DETECT_INT 中断。置位 I2C_TRANS_START，I2C Master 继续传输数据。当不再需要传输数据时，在检测到 I2C Master 的 I2C_END_DETECT_INT 中断后，配置 cmd 如第三段所示。置位 I2C_TRANS_START，I2C Master 发送 STOP 位停止传输。

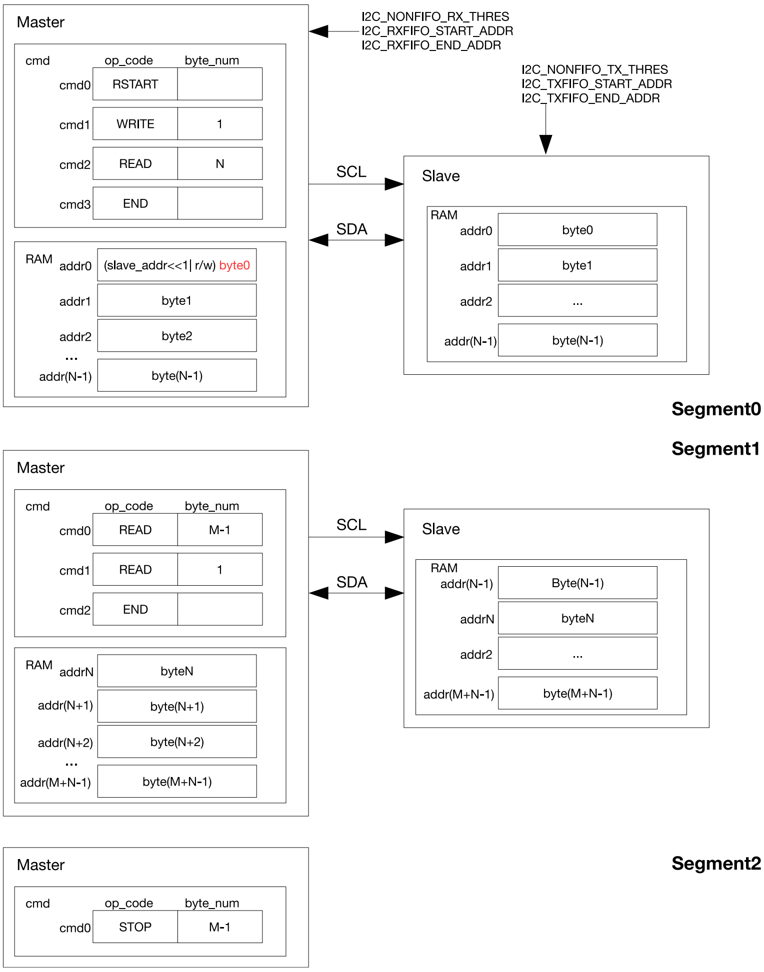


图 11-12. I2C Master 分段读 7-bit 地址 Slave

11.3.7 中断

- I2C_TX_SEND_EMPTY_INT: 每当 I2C Master 或 Slave 发送 nonfifo_tx_thres 个数据，即触发该中断。
- I2C_RX_REC_FULL_INT: 每当 I2C Master 或 Slave 接收 nonfifo_rx_thres 个数据，即触发该中断。
- I2C_ACK_ERR_INT: 当 I2C 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 I2C 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 I2C Master 或 Slave 发送一个 START 位时，即触发该中断。
- I2C_TIME_OUT_INT: 在传输过程中，当 I2C SCL 保持为高或为低电平的时间超过 I2C_TIME_OUT 个时钟后，即触发该中断。
- I2C_TRANS_COMPLETE_INT: 当 I2C Master 或 Slave 检测到 STOP 位时，即触发该中断。
- I2C_MASTER_TRAN_COMP_INT: 当 I2C Master 发送或接收一个字节，即触发该中断。
- I2C_ARBITRATION_LOST_INT: 当 I2C 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_END_DETECT_INT: 当 I2C Master 处理 END 命令时，即触发该中断。

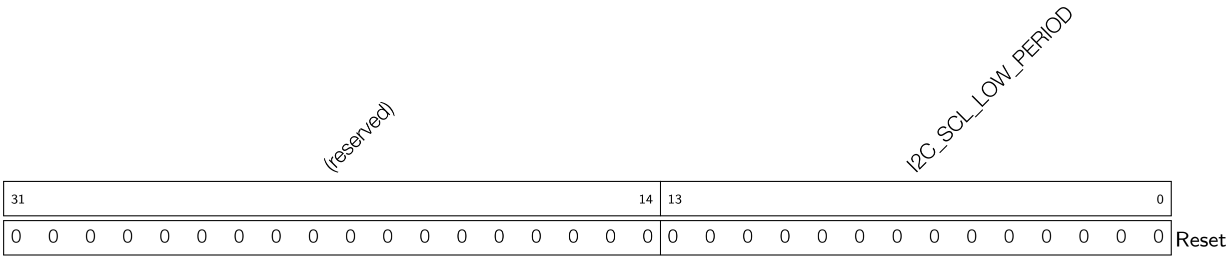
11.4 寄存器列表

名称	描述	I2C0	I2C1	访问
配置寄存器				
I2C_SLAVE_ADDR_REG	配置 I2C slave 地址	0x3FF53010	0x3FF67010	读/写
I2C_RXFIFO_ST_REG	FIFO 状态寄存器	0x3FF53014	0x3FF67014	只读
I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x3FF53018	0x3FF67018	读/写
时序寄存器				
I2C_SDA_HOLD_REG	配置在 SCL 下降沿后保持时间	0x3FF53030	0x3FF67030	读/写
I2C_SDA_SAMPLE_REG	配置在 SCL 上升沿后采样时间	0x3FF53034	0x3FF67034	读/写
I2C_SCL_LOW_PERIOD_REG	配置 SCL 时钟的低电平宽度	0x3FF53000	0x3FF67000	读/写
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平宽度	0x3FF53038	0x3FF67038	读/写
I2C_SCL_START_HOLD_REG	配置 SDA 和 SCL 下降沿之间的延迟，用于启动	0x3FF53040	0x3FF67040	读/写
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟	0x3FF53044	0x3FF67044	读/写
I2C_SCL_STOP_HOLD_REG	配置 SCL 时钟边缘后的延迟，用于停止	0x3FF53048	0x3FF67048	读/写
I2C_SCL_STOP_SETUP_REG	配置 SDA 和 SCL 上升沿之间的延迟，用于停止	0x3FF5304C	0x3FF6704C	读/写
滤波寄存器				
I2C_SCL_FILTER_CFG_REG	SCL 滤波配置寄存器	0x3FF53050	0x3FF67050	读/写
I2C_SDA_FILTER_CFG_REG	SDA 滤波配置寄存器	0x3FF53054	0x3FF67054	读/写
中断寄存器				
I2C_INT_RAW_REG	原始中断状态	0x3FF53020	0x3FF67020	只读
I2C_INT_CLR_REG	中断清除位	0x3FF53024	0x3FF67024	只写
I2C_INT_ENA_REG	中断使能位	0x3FF53028	0x3FF67028	读/写
I2C_INT_STATUS_REG	中断状态位	0x3FF5302C	0x3FF6702C	只读
命令寄存器				
I2C_COMD0_REG	I2C 命令寄存器 0	0x3FF53058	0x3FF67058	读/写
I2C_COMD1_REG	I2C 命令寄存器 1	0x3FF5305C	0x3FF6705C	读/写
I2C_COMD2_REG	I2C 命令寄存器 2	0x3FF53060	0x3FF67060	读/写
I2C_COMD3_REG	I2C 命令寄存器 3	0x3FF53064	0x3FF67064	读/写
I2C_COMD4_REG	I2C 命令寄存器 4	0x3FF53068	0x3FF67068	读/写
I2C_COMD5_REG	I2C 命令寄存器 5	0x3FF5306C	0x3FF6706C	读/写
I2C_COMD6_REG	I2C 命令寄存器 6	0x3FF53070	0x3FF67070	读/写
I2C_COMD7_REG	I2C 命令寄存器 7	0x3FF53074	0x3FF67074	读/写
I2C_COMD8_REG	I2C 命令寄存器 8	0x3FF53078	0x3FF67078	读/写
I2C_COMD9_REG	I2C 命令寄存器 9	0x3FF5307C	0x3FF6707C	读/写
I2C_COMD10_REG	I2C 命令寄存器 10	0x3FF53080	0x3FF67080	读/写
I2C_COMD11_REG	I2C 命令寄存器 11	0x3FF53084	0x3FF67084	读/写
I2C_COMD12_REG	I2C 命令寄存器 12	0x3FF53088	0x3FF67088	读/写
I2C_COMD13_REG	I2C 命令寄存器 13	0x3FF5308C	0x3FF6708C	读/写
I2C_COMD14_REG	I2C 命令寄存器 14	0x3FF53090	0x3FF67090	读/写
I2C_COMD15_REG	I2C 命令寄存器 15	0x3FF53094	0x3FF67094	读/写

11.5 寄存器

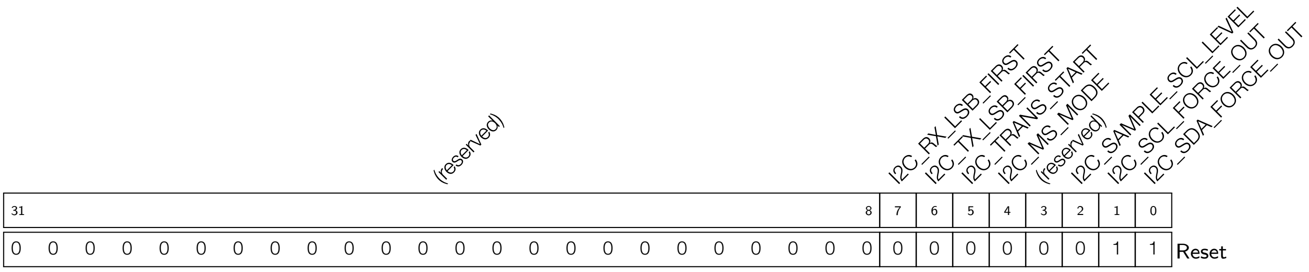
本小节括号中的地址均为相对于 I2C 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 11.4 寄存器列表。

Register 11.1. I2C_SCL_LOW_PERIOD_REG (0x0000)



I2C_SCL_LOW_PERIOD I2C 为 Master 时，该寄存器用于配置 SCL 时钟信号的低电平持续的 APB 时钟周期数。（读/写）

Register 11.2. I2C_CTR_REG (0x0004)



I2C_RX_LSB_FIRST 该位用于控制接收数据的存储模式。（读/写）

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。

I2C_TX_LSB_FIRST 该位用于控制发送数据的发送模式。（读/写）

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。

I2C_TRANS_START 置位该位，开始在 txfifo 中发送数据。（读/写）

I2C_MS_MODE 置位该位，配置该模块为 I2C Master。清除该位，配置该模块为 I2C Slave。（读/写）

I2C_SAMPLE_SCL_LEVEL 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_SCL_FORCE_OUT 0：直接输出；1：漏极开漏输出。（读/写）

I2C_SDA_FORCE_OUT 0：直接输出；1：漏极开漏输出。（读/写）

Register 11.3. I2C_SR_REG (0x0008)

(reserved)	I2C_SCL_STATE_LAST	(reserved)	I2C_SCL_MAIN_STATE_LAST	I2C_TXFIFO_CNT	(reserved)	I2C_RXFIFO_CNT	(reserved)	I2C_BYTE_TRANS	I2C_SLAVE_ADDRESSED	I2C_BUS_BUSY	I2C_ARB_LOST	I2C_TIME_OUT	I2C_SLAVE_RW	I2C_ACK_REC	
31	30 28	27	26 24	23 18	17 14	13 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

I2C_SCL_STATE_LAST 存储产生 SCL 的状态机的值。(只读)
0：空闲；1：开始；2：下降沿；3：低电平；4：上升沿；5：高电平；6：停止

I2C_SCL_MAIN_STATE_LAST 存储 I2C 模块的状态机的值。(只读)
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

I2C_TXFIFO_CNT 该字段存储了 RAM 中接收数据的字节数。(只读)

I2C_RXFIFO_CNT 该字段表示需发送数据的字节数。(只读)

I2C_BYTE_TRANS 当传输了一个字节时，该字段变为 1。(只读)

I2C_SLAVE_ADDRESSED 当配置成 I2C Slave，主机发送的地址为从机地址时，该位翻转为高电平。(只读)

I2C_BUS_BUSY 1：I2C 总线处于传输数据状态；0：I2C 总线处于空闲状态。(只读)

I2C_ARB_LOST 当 I2C 控制器不控制 SCL 线，该寄存器变为 1 。(只读)

I2C_TIME_OUT 当 I2C 控制器接收一个 bit 的时间超时，该字段变为 1。(只读)

I2C_SLAVE_RW 在从机模式中，1：主机在从机中读取数据；0：主机向从机写入数据。(只读)

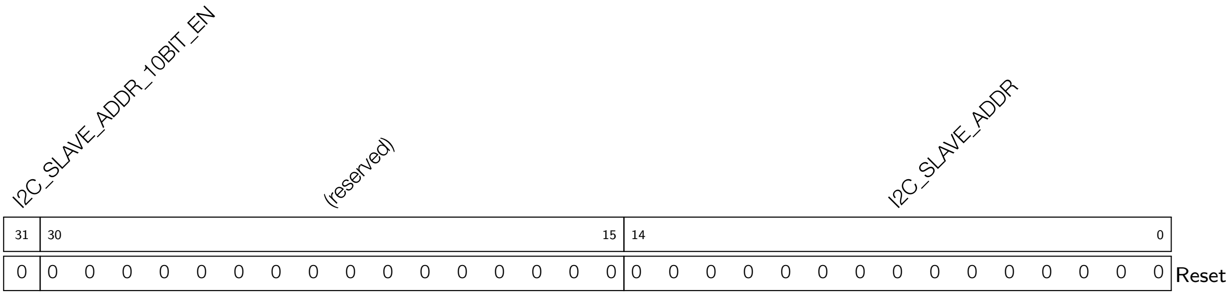
I2C_ACK_REC 该寄存器存储了接收到的 ACK 位的值。(只读)

Register 11.4. I2C_TO_REG (0x000c)

(reserved)	I2C_TIME_OUT_REG	
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	Reset

I2C_TIME_OUT_REG 当接收一个 bit 数据超过该值时，产生超时中断。(读/写)

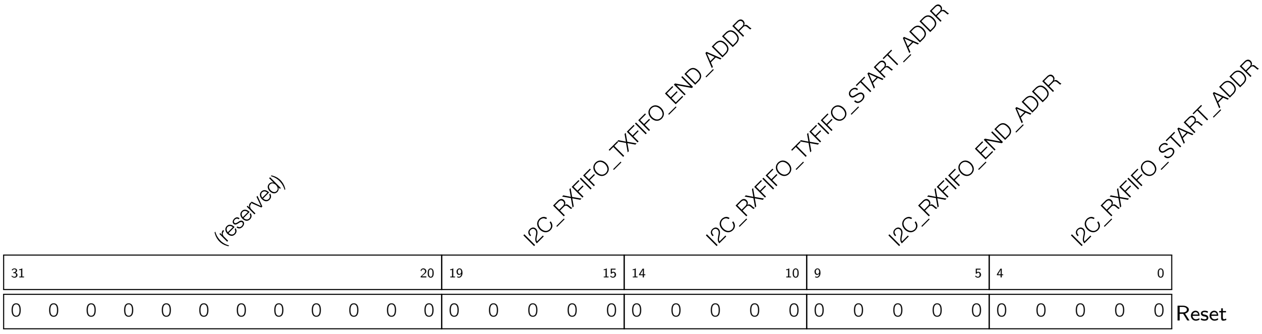
Register 11.5. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR_10BIT_EN I2C 为 Slave 时，该字段用于开启从机的 10-bit 寻址模式。（读/写）

I2C_SLAVE_ADDR 当配置为 I2C Slave 时，该字段用于配置 Slave 地址。（读/写）

Register 11.6. I2C_RXFIFO_ST_REG (0x0014)



I2C_TXFIFO_END_ADDR 如 nonfifo_tx_thres 所述，该位是最后发送数据的偏移地址。该值在每次 I2C_TX_SEND_EMPTY_INT 中断或 I2C_TRANS_COMPLETE_INT 中断产生时都会更新。（只读）

I2C_TXFIFO_START_ADDR 如 nonfifo_tx_thres 所述，该位是最先发送数据的偏移地址。（只读）

I2C_RXFIFO_END_ADDR 如 nonfifo_rx_thres 所述，该位是最先接收数据的偏移地址。该值在每次 I2C_RX_REC_FULL_INT 中断或 I2C_TRANS_COMPLETE_INT 中断产生时都会更新。（只读）

I2C_RXFIFO_START_ADDR 如 nonfifo_rx_thres 所述，该位是最后接收数据的偏移地址。（只读）

Register 11.7. I2C_FIFO_CONF_REG (0x0018)

(reserved)	I2C_NONFIFO_TX_THRES	I2C_NONFIFO_RX_THRES	(reserved)	I2C_FIFO_ADDR_CFG_EN	I2C_NONFIFO_EN	
31 26	25 20	19 14	13 12	11	10	
0 0 0 0 0 0	0x15	0x15	0 0	0	0	Reset

I2C_NONFIFO_TX_THRES 当 I2C 发送的数据多于 nonfifo_tx_thres 存储的字节，它将产生 tx_send_empty_int_raw 溢出中断，并更新发送数据的当前偏移地址。(读/写)

I2C_NONFIFO_RX_THRES 当 I2C 接收的数据多于 nonfifo_tx_thres 存储的字节，它将产生 rx_send_full_int_raw 溢出中断，并更新接收数据的当前偏移地址。(读/写)

I2C_FIFO_ADDR_CFG_EN 当该位为 1 时，可以按寄存器地址访问 Slave。(读/写)

I2C_NONFIFO_EN 置位使能 APB nonfifo 访问方式。(读/写)

Register 11.8. I2C_INT_RAW_REG (0x0020)

(reserved)	I2C_TX_SEND_EMPTY_INT_RAW	I2C_RX_REC_FULL_INT_RAW	I2C_ACK_ERR_INT_RAW	I2C_TRANS_START_INT_RAW	I2C_TIME_OUT_INT_RAW	I2C_TRANS_COMPLETE_INT_RAW	I2C_MASTER_TRAN_COMP_INT_RAW	I2C_ARBITRATION_LOST_INT_RAW	I2C_END_DETECT_INT_RAW	
31 13	12	11	10	9	8	7	6	5	3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

I2C_TX_SEND_EMPTY_INT_RAW I2C_TX_SEND_EMPTY_INT 中断的原始中断状态位。(只读)

I2C_RX_REC_FULL_INT_RAW I2C_RX_REC_FULL_INT 中断的原始中断状态位。(只读)

I2C_ACK_ERR_INT_RAW I2C_ACK_ERR_INT 中断的原始中断状态位。(只读)

I2C_TRANS_START_INT_RAW I2C_TRANS_START_INT 中断的原始中断状态位。(只读)

I2C_TIME_OUT_INT_RAW I2C_TIME_OUT_INT 中断的原始中断状态位。(只读)

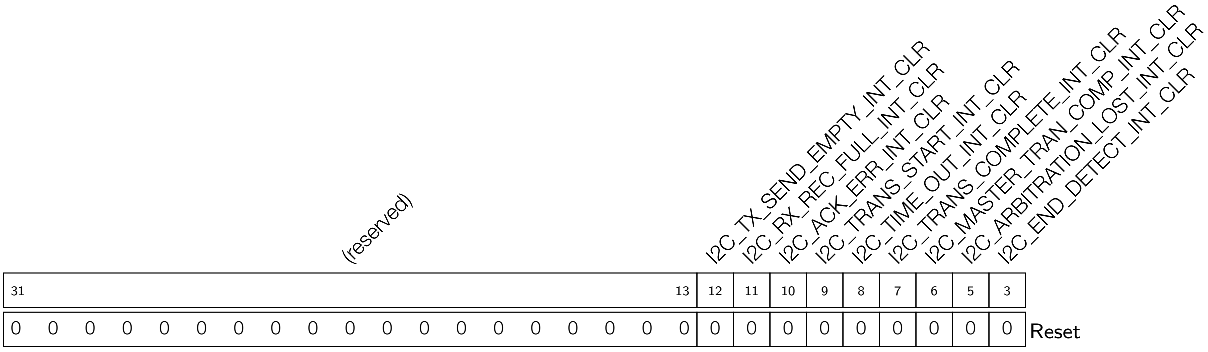
I2C_TRANS_COMPLETE_INT_RAW I2C_TRANS_COMPLETE_INT 中断的原始中断状态位。(只读)

I2C_MASTER_TRAN_COMP_INT_RAW I2C_MASTER_TRAN_COMP_INT 中断的原始中断状态位。(只读)

I2C_ARBITRATION_LOST_INT_RAW I2C_ARBITRATION_LOST_INT 中断的原始中断状态位。(只读)

I2C_END_DETECT_INT_RAW I2C_END_DETECT_INT 中断的原始中断状态位。(只读)

Register 11.9. I2C_INT_CLR_REG (0x0024)



I2C_TX_SEND_EMPTY_INT_CLR 置位该位用以清除 I2C_TX_SEND_EMPTY_INT 中断。（只写）

I2C_RX_REC_FULL_INT_CLR 置位该位用以清除 I2C_RX_REC_FULL_INT 中断。（只写）

I2C_ACK_ERR_INT_CLR 置位该位用以清除 I2C_ACK_ERR_INT 中断。（只写）

I2C_TRANS_START_INT_CLR 置位该位用以清除 I2C_TRANS_START_INT 中断。（只写）

I2C_TIME_OUT_INT_CLR 置位该位用以清除 I2C_TIME_OUT_INT 中断。（只写）

I2C_TRANS_COMPLETE_INT_CLR 置位该位用以清除 I2C_TRANS_COMPLETE_INT 中断。（只写）

I2C_MASTER_TRAN_COMP_INT_CLR 置位该位用以清除 I2C_MASTER_TRAN_COMP_INT 中断。（只写）

I2C_ARBITRATION_LOST_INT_CLR 置位该位用以清除 I2C_ARBITRATION_LOST_INT 中断。（只写）I2C_SLAVE_TRAN_COMP_INT 中断。（只写）

I2C_END_DETECT_INT_CLR 置位该位用以清除 I2C_END_DETECT_INT 中断。（只写）

Register 11.10. I2C_INT_ENA_REG (0x0028)

(reserved)	I2C_TX_SEND_EMPTY_INT_ENA	I2C_RX_REC_FULL_INT_ENA	I2C_ACK_ERR_INT_ENA	I2C_TRANS_START_INT_ENA	I2C_TIME_OUT_INT_ENA	I2C_TRANS_COMPLETE_INT_ENA	I2C_MASTER_TRAN_COMP_INT_ENA	I2C_ARBITRATION_LOST_INT_ENA	I2C_END_DETECT_INT_ENA	
31 13	12	11	10	9	8	7	6	5	3	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

I2C_TX_SEND_EMPTY_INT_ENA 中断使能位，用于 I2C_TX_SEND_EMPTY_INT 中断。（读/写）

I2C_RX_REC_FULL_INT_ENA 中断使能位，用于 I2C_RX_REC_FULL_INT 中断。（读/写）

I2C_ACK_ERR_INT_ENA 中断使能位，用于 I2C_ACK_ERR_INT 中断。（读/写）

I2C_TRANS_START_INT_ENA 中断使能位，用于 I2C_TRANS_START_INT 中断。（读/写）

I2C_TIME_OUT_INT_ENA 中断使能位，用于 I2C_TIME_OUT_INT 中断。（读/写）

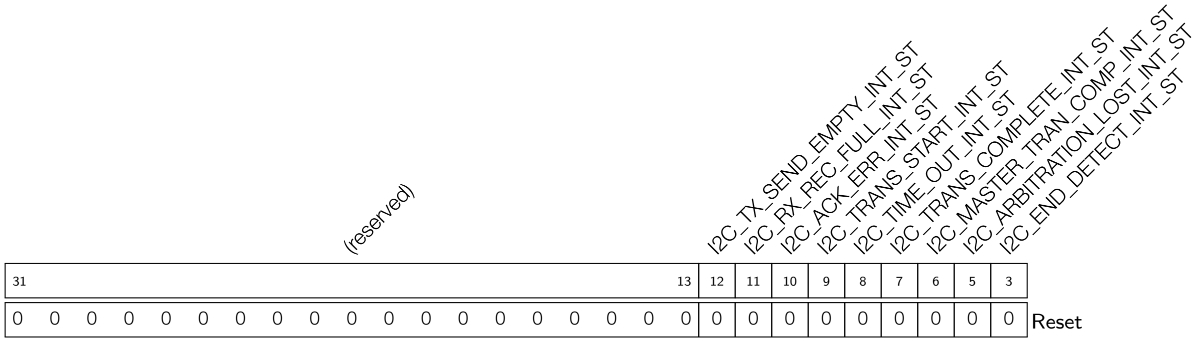
I2C_TRANS_COMPLETE_INT_ENA 中断使能位，用于 I2C_TRANS_COMPLETE_INT 中断。（读/写）

I2C_MASTER_TRAN_COMP_INT_ENA 中断使能位，用于 I2C_MASTER_TRAN_COMP_INT 中断。（读/写）

I2C_ARBITRATION_LOST_INT_ENA 中断使能位，用于 I2C_ARBITRATION_LOST_INT 中断。（读/写）

I2C_END_DETECT_INT_ENA 中断使能位，用于 I2C_END_DETECT_INT 中断。（读/写）

Register 11.11. I2C_INT_STATUS_REG (0x002c)



I2C_TX_SEND_EMPTY_INT_ST 隐蔽中断状态位，用于 I2C_TX_SEND_EMPTY_INT 中断。(只读)

I2C_RX_REC_FULL_INT_ST 隐蔽中断状态位，用于 I2C_RX_REC_FULL_INT中断。(只读)

I2C_ACK_ERR_INT_ST 隐蔽中断状态位，用于 I2C_ACK_ERR_INT 中断。(只读)

I2C_TRANS_START_INT_ST 隐蔽中断状态位，用于 I2C_TRANS_START_INT 中断。(只读)

I2C_TIME_OUT_INT_ST 隐蔽中断状态位，用于 I2C_TIME_OUT_INT 中断。(只读)

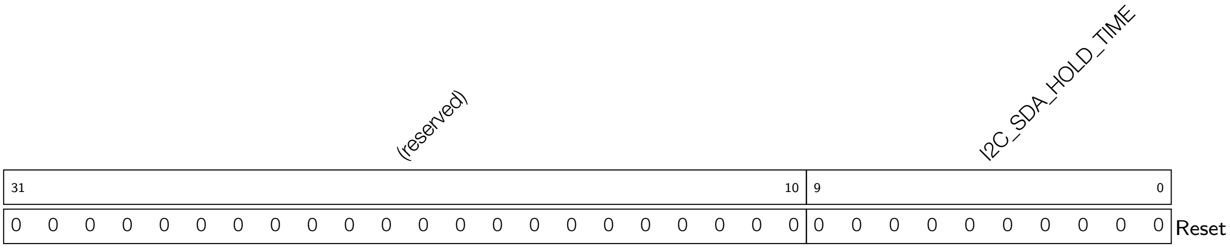
I2C_TRANS_COMPLETE_INT_ST 隐蔽中断状态位，用于 I2C_TRANS_COMPLETE_INT 中断。(只读)

I2C_MASTER_TRAN_COMP_INT_ST 隐蔽中断状态位，用于 I2C_MASTER_TRAN_COMP_INT 中断。(只读)

I2C_ARBITRATION_LOST_INT_ST 隐蔽中断状态位，用于 I2C_ARBITRATION_LOST_INT 中断。(只读)

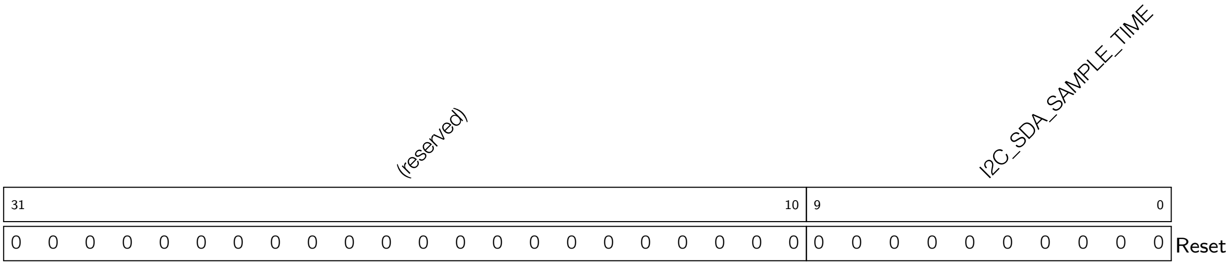
I2C_END_DETECT_INT_ST 隐蔽中断状态位，用于I2C_END_DETECT_INT 中断。(只读)

Register 11.12. I2C_SDA_HOLD_REG (0x0030)



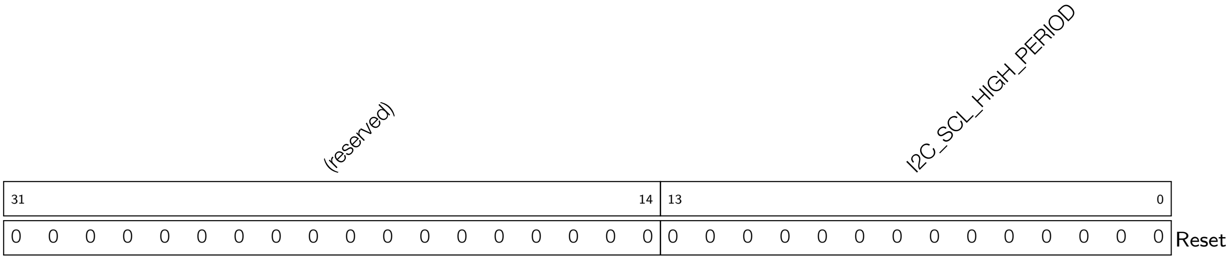
I2C_SDA_HOLD_TIME 用于配置 SCL 下降沿后 SDA 保持不变的 APB 时钟周期数。(读/写)

Register 11.13. I2C_SDA_SAMPLE_REG (0x0034)



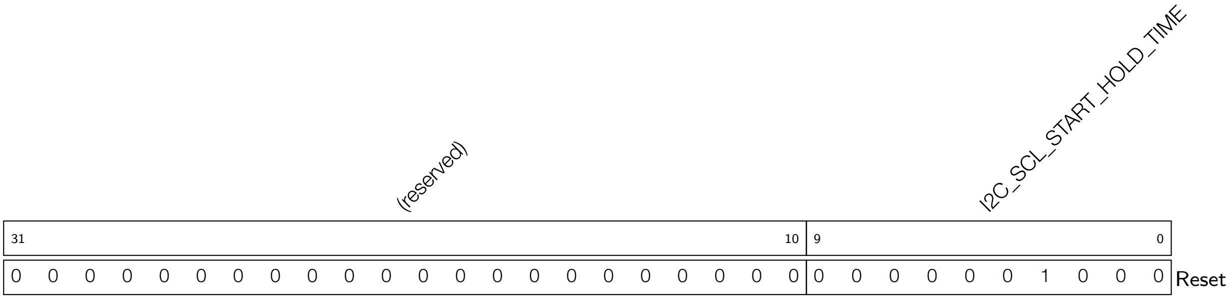
I2C_SDA_SAMPLE_TIME 用于配置 I2C 采样 SDA 的时间，基于 APB 时钟周期数。（读/写）

Register 11.14. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD I2C 为 Master 时，用于配置使 SCL 维持在高电平的 APB 时钟周期数。（读/写）

Register 11.15. I2C_SCL_START_HOLD_REG (0x0040)



I2C_SCL_START_HOLD_TIME 用于配置 SDA 下降沿和 SCL 下降沿之间的时间，用于 START 状态，基于 APB 时钟周期数。（读/写）

Register 11.16. 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 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0	Reset

I2C_SCL_RSTART_SETUP_TIME 用于配置 SCL 的上升沿和 SDA 的下降沿之间的时间，用于 RESTART 状态，基于 APB 时钟周期数。（读/写）

Register 11.17. 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 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_STOP_HOLD_TIME 用于配置 STOP 状态后的延迟，基于 APB 时钟周期数。（读/写）

Register 11.18. I2C_SCL_STOP_SETUP_REG (0x004C)

(reserved)	I2C_SCL_STOP_SETUP_TIME	
31 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	0 0 0 0 0 0 0 0 0 0	Reset

I2C_SCL_STOP_SETUP_TIME 用于配置 SCL 上升沿和 SDA 上升沿之间的时间，基于 APB 时钟周期数。（读/写）

Register 11.19. I2C_SCL_FILTER_CFG_REG (0x0050)

(reserved)	I2C_SCL_FILTER_EN	I2C_SCL_FILTER_THRES	
31 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	Reset

I2C_SCL_FILTER_EN 该位为 SCL 的滤波使能位。（读/写）

I2C_SCL_FILTER_THRES 基于 APB 时钟周期数，当 SCL 输入信号上的脉冲宽度小于该寄存器的值时，I2C 控制器将过滤该脉冲。（读/写）

Register 11.20. I2C_SDA_FILTER_CFG_REG (0x0054)

(reserved)	I2C_SDA_FILTER_EN	I2C_SDA_FILTER_THRES	
31 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	Reset

I2C_SDA_FILTER_EN 该位为 SDA 的滤波使能位。（读/写）

I2C_SDA_FILTER_THRES 基于 APB 时钟周期数，当 SDA 输入信号上的脉冲宽度小于此寄存器的值时，I2C 控制器将过滤该脉冲。（读/写）

Register 11.21. I2C_COMD*n*_REG (*n*: 0-15) (0x58+4**n*)

I2C_COMMAND*n*_DONE	(reserved)	I2C_COMMAND*n*	
31	30 14	13 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2C_COMMAND*n*_DONE 在 I2C 主机模式下完成 command *n* 时，该位翻转为高电平。（读/写）

I2C_COMMAND*n* command *n* 的内容。它包括三个部分：（读/写）

op_code 是命令，0：START；1：WRITE；2：READ；3：STOP；4：END。

Byte_num 代表了需要被发送或接收的字节数。

ack_check_en，ack_exp 和 ack 用于控制 ACK 位。详情请参考 I2C cmd 结构。

12 I2S 控制器 (I2S)

12.1 概述

I2S 总线为多媒体应用，尤其是数字音频应用提供了灵活的数据通信接口。ESP32 内置两个 I2S 接口，即 I2S0 和 I2S1。

I2S 标准总线定义了三种信号：时钟信号 BCK、声道选择信号 WS 和串行数据信号 SD。一个基本的 I2S 数据总线有一个主机和一个从机。主机和从机的角色在通信过程中保持不变。ESP32 的 I2S 模块包含独立的发送和接收声道，能够保证优良的通信性能。

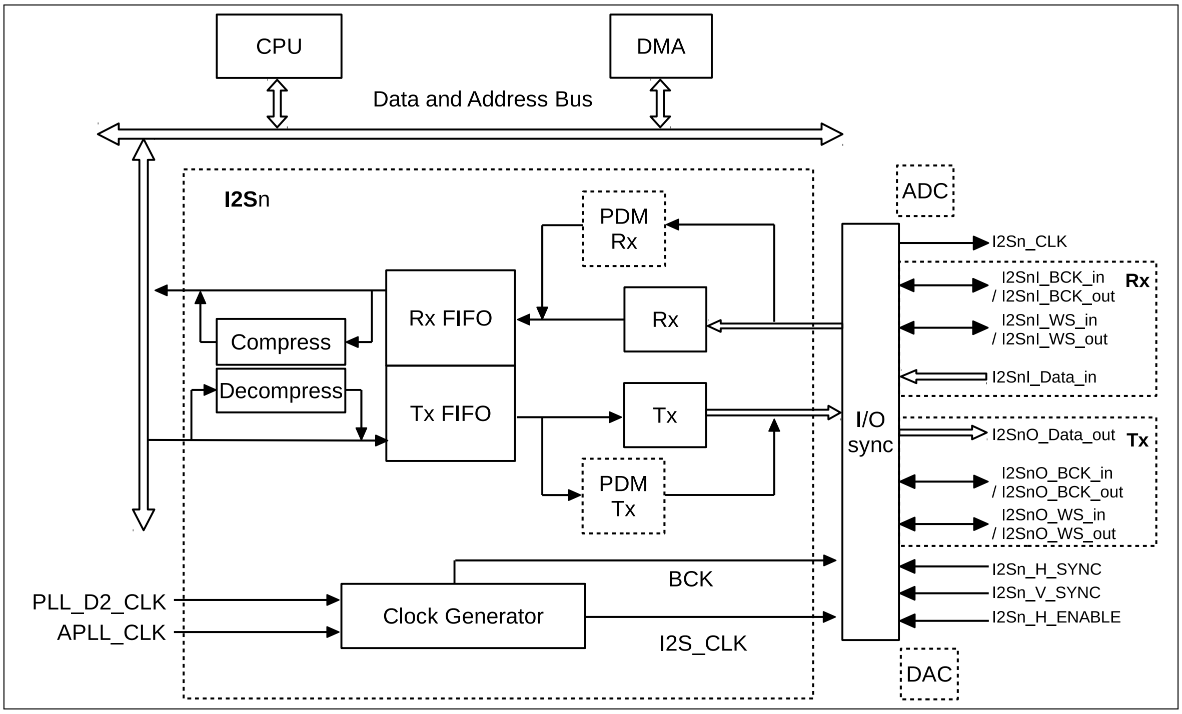


图 12-1. I2S 系统框图

图 12-1 是 ESP32 I2S 模块的结构框图，图中”*n*” 对应为 0 或 1，即 I2S0 或 I2S1。每个 I2S 模块包含一个独立的发送单元 (Tx) 和一个独立的接收单元 (Rx)。发送和接收单元各自有一组三线接口，分别为时钟线 BCK，声道选择线 WS 和串行数据线 SD。其中，发送单元的串行数据线固定为输出，接收单元的串行数据线固定为接收。发送单元和接收单元的时钟线和声道选择线均可配置为主机发送和从机接收。在 LCD 模式下，串行数据线扩展为并行数据总线。I2S 模块发送和接收单元各有一块宽 32 bit、深 64 bit 的 FIFO。此外，只有 I2S0 支持接收/发送 PDM 信号并且支持片上 DAC/ADC 模块。

图 12-1 右侧为 I2S 模块的信号总线。Rx 和 Tx 模块的信号命名规则为: I2S*n*A_B_C。其中”*n*” 为模块名，表示 I2S0 或 I2S1；”A” 表示 I2S 模块的数据总线信号的方向，”I” 表示输入，”O” 表示输出；”B” 表示信号功能；”C” 表示该信号的方向，”in” 表示该信号输入 I2S 模块，”out” 表示该信号自 I2S 模块输出。各信号总线的具体描述见表 12-1。除 I2S*n*_CLK 信号外，其他信号均需要经过 GPIO 交换矩阵和 IO_MUX 映射到芯片的管脚。I2S*n*_CLK 信号需要经过 IO_MUX 映射到芯片管脚。详情请参考章节 IO_MUX 和 GPIO 交换矩阵。

表 12-1. I2S 信号总线描述

信号总线	信号方向	数据信号方向
I2S*n*I_BCK_in	从机模式下，I2S 模块输入信号	表示 I2S 模块接收数据
I2S*n*I_BCK_out	主机模式下，I2S 模块输出信号	表示 I2S 模块接收数据
I2S*n*I_WS_in	从机模式下，I2S 模块输入信号	表示 I2S 模块接收数据
I2S*n*I_WS_out	主机模式下，I2S 模块输出信号	表示 I2S 模块接收数据
I2S*n*I_Data_in[1]	I2S 模块输入信号	I2S 模式下，I2S*n*I_Data_in[15] 为 I2S 的串行数据总线，LCD 模式下，可以根据需要配置数据总线的宽度
I2S*n*O_Data_out[1]	I2S 模块输出信号	I2S 模式下，I2S*n*O_Data_out[23] 为 I2S 的串行数据总线，LCD 模式下，可以根据需要配置数据总线的宽度
I2S*n*O_BCK_in	从机模式下，I2S 模块输入信号	表示 I2S 模块发送数据
I2S*n*O_BCK_out	主机模式下，I2S 模块输出信号	表示 I2S 模块发送数据
I2S*n*O_WS_in	从机模式下，I2S 模块输入信号	表示 I2S 模块发送数据
I2S*n*O_WS_out	主机模式下，I2S 模块输出信号	表示 I2S 模块发送数据
I2S*n*_CLK[2]	I2S 模块输出信号	作为外部芯片的时钟源
I2S*n*_H_SYNC	Camera 模式下，I2S 模块输入信号	来自 Camera 的信号
I2S*n*_V_SYNC		
I2S*n*_H_ENABLE		

说明：

1. 假设 I2S 输入/输出信号位宽为 *N* 比特，那么 I2S 模块输入信号应配置为 I2S*n*I_Data_in[*N*-1:0]，I2S 模块输出信号应配置为 I2S*n*O_Data_out[23:23-*N*+1]。通常对输入信号 *N*=8 或 16，对输出信号 *N*=8，16 或 24（I2S1 不包含 24 bit 模式）。
2. I2S*n*_CLK 仅能映射至 GPIO0、U0RXD (GPIO3) 或 U0TXD (GPIO1) 管脚，选择 CLK_OUT1、CLK_OUT2 或 CLK_OUT3 功能。更多信息见表 4-3：*IO_MUX Pad 列表*。

12.2 主要特性

I2S 模式

- 可配置高精度输出时钟；
- 支持全双工和半双工收发数据；
- 支持多种音频标准；
- 内嵌 A 律压缩/解压缩模块；
- 可配置时钟；
- 支持 PDM 信号输入输出；
- 收发数据模式可配置。

LCD 模式

- 支持外接 LCD；
- 支持外接 Camera；
- 支持多种 LCD 模式；
- 支持连接片上 DAC/ADC 模式。

I2S 中断

- I2S 接口中断；
- I2S DMA 接口中断。

12.3 I2S 模块时钟

I2S*n*_CLK 作为 I2S 模块的主时钟，是由 160 MHz 时钟 PLL_F160M_CLK 或者可配置的模拟 PLL 输出时钟 APLL_CLK 进行分频获得。I2S 模块的串行时钟 BCK 再由 I2S*n*_CLK 分频获得，如图 12-2 所示。寄存器 I2S_CLKM_CONF_REG 中 I2S_CLKA_ENA 比特用于选择 PLL_F160M_CLK 还是 APLL_CLK 作为 I2S*n* 的时钟源，默认使用 PLL_F160M_CLK 作为 I2S*n* 的时钟源。

注意：

- 在使用 PLL_F160M_CLK 作为时钟源时，不建议使用小数分频功能。需要获得更高精度的 I2S*n*_CLK 和 BCK 时，要使用 APLL_CLK 作为时钟源，详情请参考章节复位和时钟。
- 当 ESP32 I2S 作为从机时，主机必须使用 ESP32 的 I2S*n*_CLK 作为主时钟，同时 f_{i2s} >= 8 * f_{BCK}。

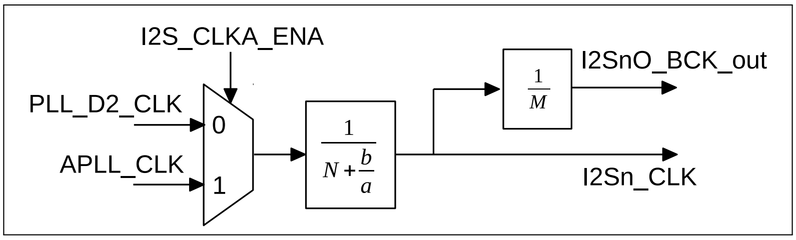


图 12-2. I2S 时钟

I2S*n*_CLK 的频率 f_{i2s} 与分频器时钟源频率 f_{pll} 间的关系如下：

$$f_{\text{i2s}} = \frac{f_{\text{pll}}}{\text{N} + \frac{\text{b}}{\text{a}}}$$

其中，N>=2，N 对应 I2S_CLKM_CONF_REG 寄存器中的 I2S_CLKM_DIV_NUM[7:0] 位，b 为 I2S_CLKM_DIV_B[5:0] 位，a 为 I2S_CLKM_DIV_A[5:0] 位。

在主机模式下，I2S 模块的串行时钟 BCK 由 I2S*n*_CLK 分频获得。即：

$$f_{\text{BCK}} = \frac{f_{\text{i2s}}}{\text{M}}$$

其中，M>=2，在主机发送模式下，M 为寄存器 I2S_SAMPLE_RATE_CONF_REG 的 I2S_TX_BCK_DIV_NUM[5:0] 位，在主机接收模式下，M 为寄存器 I2S_SAMPLE_RATE_CONF_REG 的 I2S_RX_BCK_DIV_NUM[5:0] 位。

12.4 I2S 模式

ESP32 I2S 模块内置数据 A 律压缩/解压缩模块，用于对接收到的音频数据进行 A 律缩/解压缩操作。如果要使用 A 律缩/解压缩模块，需要将 I2S_CONF1_REG 寄存器的 RX_PCM_BYPASS 比特和 TX_PCM_BYPASS 比特清零。

12.4.1 支持的音频标准

I2S 模式下，BCK 为串行时钟；WS 为通道选择信号，用于表示左右声道的切换；SD 为串行数据信号，传输音频数据。WS 信号和 SD 信号在 BCK 的下降沿发生变化，并在 BCK 的上升沿采样 SD 信号。如果将寄存器 I2S_CONF_REG 的 I2S_RX_RIGHT_FIRST 比特和 I2S_TX_RIGHT_FIRST 比特都置 1，表示 I2S 模块首先接收和发送的是右声道数据，否则为首先接收和发送的是左声道数据。

12.4.1.1 Philips 标准

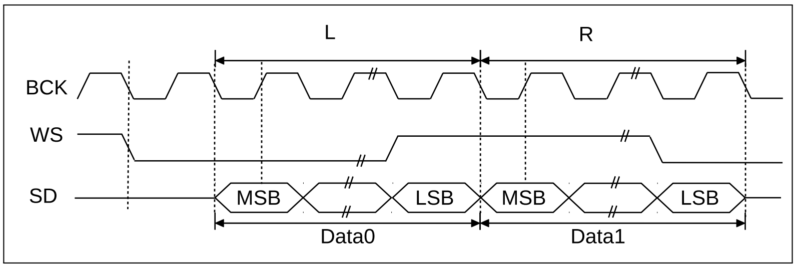


图 12-3. Philips 标准

如图 12-3 所示，在 Philips 标准下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并在当前通道数据发送结束前一个 BCK 时钟周期变化。SD 信号线首先传输音频数据的最高有效位。如果分别将寄存器 I2S_CONF_REG 的 I2S_RX_MSB_SHIFT 比特和 I2S_TX_MSB_SHIFT 比特置 1，I2S 模块接收数据和发送数据将使用 Philips 标准。

12.4.1.2 MSB 对齐标准

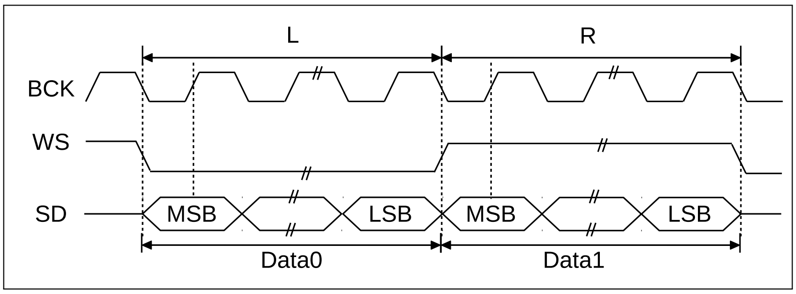


图 12-4. MSB 对齐标准

如图 12-4 所示，MSB 对齐标准下，在 BCK 下降沿，WS 信号和 SD 信号同时变化。WS 持续到当前通道数据发送结束，SD 信号线上首先传输音频数据的最高位。如果寄存器 I2S_CONF_REG 的 I2S_RX_MSB_SHIFT 比特和 I2S_TX_MSB_SHIFT 比特清零，则 I2S 模块接收数据和发送数据将使用 MSB 对齐标准。

12.4.1.3 PCM 标准

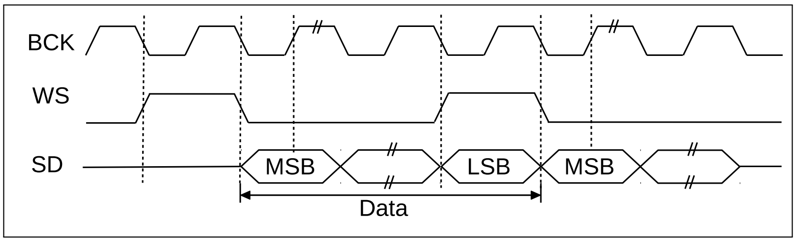


图 12-5. PCM 标准

如图 12-5 所示，在 PCM 标准的短帧同步模式下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并持续 1 个 BCK 时钟周期。SD 信号线上首先传输音频数据的最高位。如果将寄存器 I2S_CONF_REG 的 I2S_RX_SHORT_SYNC 比特和 I2S_TX_SHORT_SYNC 比特置 1，那么 I2S 模块接收数据和发送数据将使用短帧同步模式。

12.4.2 模块复位

寄存器 I2S_CONF_REG 中的低四个比特为 I2S_TX_RESET 比特、I2S_RX_RESET 比特、I2S_TX_FIFO_RESET 比特和 I2S_RX_FIFO_RESET 比特，分别表示对接收模块、发送模块和其对应的 FIFO 进行复位。完成一次复位操作需要先将对应比特置 1，然后清零。

12.4.3 FIFO 操作

I2S 一次 FIFO 操作，写入/读取数据包长度为 32 位。数据包格式由寄存器配置。由图 12-1 可知，待发送数据和接收到的数据，都要先写入 FIFO，然后再读出。读写 FIFO 的方式有两种，一种是使用 CPU 直接进行读写，另一种是使用 DMA 控制器进行读写。

一般情况下 I2S_FIFO_CONF_REG 寄存器的 I2S_RX_FIFO_MOD_FORCE_EN 比特和 I2S_TX_FIFO_MOD_FORCE_EN 比特均需要置为 1。I2S_TX_DATA_NUM[5:0] 比特和 I2S_RX_DATA_NUM[5:0] 比特用于控制 FIFO 缓冲发送数据和接收数据的长度。硬件自动检查 FIFO 缓存的接收数据长度 RX_LEN 和发送数据长度 TX_LEN。

当 RX_LEN > I2S_RX_DATA_NUM[5:0] 时，表示 FIFO 缓存的接收数据已经达到设定阈值，需要及时取走。当 TX_LEN < I2S_TX_DATA_NUM[5:0] 时，表示 FIFO 缓存的发送数据还未达到设定阈值，可以继续向 FIFO 中填充数据。

12.4.4 发送数据

ESP32 I2S 发送数据分为三个阶段：

- 第一阶段从内存中读出数据并写入 FIFO；
- 第二阶段将待发送数据从 FIFO 中读出；
- 第三阶段，在 I2S 模式下，将待发送数据转换为串行数据流输出；在 LCD 模式下，将待发送数据转换为位宽固定的并行数据流输出。

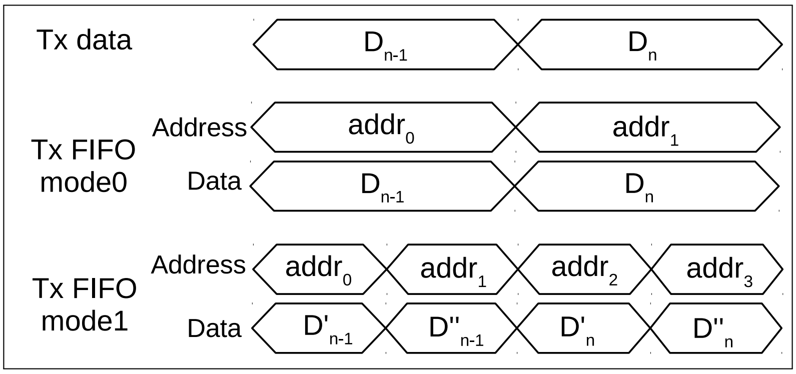


图 12-6. 发送 FIFO 数据模式

表 12-2. 寄存器配置

	I2S_TX_FIFO_MOD[2:0]	描述
Tx FIFO mode0	0	16 位双通道数据
	2	32 位双通道数据
	3	32 位单通道数据
Tx FIFO mode1	1	16 位单通道数据

在第一阶段，待发送数据写入 FIFO 的模式有两种。Tx FIFO mode0 条件下，待发送数据 Tx data 按时间先后被直接写入 FIFO；Tx FIFO mode1 条件下，待发送数据被分成高 16 比特和低 16 比特，重新组合后再写入 FIFO，如图 12-6 所示，对应寄存器如表 12-2 所示。D_n' 由 D_n 的高 16 比特和 16 个 0 组成，D_n'' 由 D_n 的低 16 比特和 16 个 0 组成，即：$D_n' = \{D_n[31:16], 16'h0\}$，$D_n'' = \{D_n[15:0], 16'h0\}$。

在第二阶段，系统会按照相关寄存器配置将待发送数据从 FIFO 读出。从 FIFO 读出数据的模式与 I2S_TX_FIFO_MOD[2.0] 和 I2S_TX_CHAN_MOD[2:0] 的取值有关。I2S_TX_FIFO_MOD[2.0] 决定数据是 16 位还是 32 位；I2S_TX_CHAN_MOD[2:0] 决定待发送数据的格式，如表 12-3 所示。

表 12-3. 发送通道模式

I2S_TX_CHAN_MOD[2:0]	描述
0	双声道模式
1	单声道模式 I2S_TX_MSB_RIGHT=0 时，左声道和右声道数据均为左声道数据。 I2S_TX_MSB_RIGHT=1 时，左声道和右声道数据均为右声道数据。
2	单声道模式 I2S_TX_MSB_RIGHT=0 时，左声道和右声道数据均为右声道数据。 I2S_TX_MSB_RIGHT=1 时，左声道和右声道数据均为左声道数据。
3	单声道模式 I2S_TX_MSB_RIGHT=0 时，左声道数据为常数 REG[31:0]。 I2S_TX_MSB_RIGHT=1 时，右声道数据为常数 REG[31:0]。
4	单声道模式 I2S_TX_MSB_RIGHT=0 时，右声道数据为常数 REG[31:0]。 I2S_TX_MSB_RIGHT=1 时，左声道数据为常数 REG[31:0]。

其中 REG[31:0] 为寄存器 I2S_CONF_SINGLE_DATA_REG[31:0] 的值。

第三阶段输出由 I2S 的模式和寄存器 I2S_SAMPLE_RATE_CONF_REG 的 I2S_TX_BITS_MOD[5:0] 比特决定。

12.4.5 接收数据

ESP32 I2S 接收数据也分为三个阶段：

- 第一阶段，在 I2S 模式下，输入的串行比特流数据会被以声道属性转换成宽度为 64 位的并行数据流；在 LCD 模式下，输入的位宽固定的并行数据流会被扩展成宽度为 64 位的并行数据流；
- 第二阶段将待接收的数据写入 FIFO；
- 第三阶段将待接收的数据从 FIFO 中读出，并写入内存。

在第一阶段，接收模块按 I2S*n*I_WS_out（或 I2S*n*I_WS_in）信号电平的高低，将接收到数据流扩展成高电平对应 32 位和低电平对应 32 位的并行数据流，不足的补 0。寄存器 I2S_CONF_REG 的 I2S_RX_MSB_RIGHT 比特用于确定扩展后数据的排列方式。

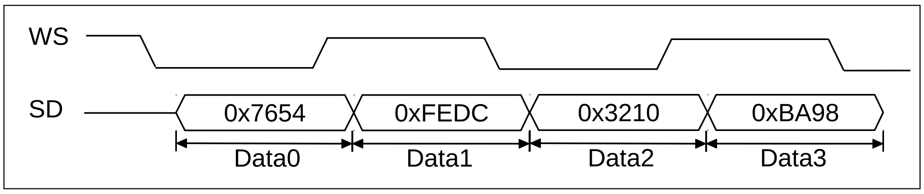


图 12-7. 第一阶段接收数据

例如，如图 12-7 所示，如果串行数据宽度为 16 位，当 I2S_RX_RIGHT_FIRST = 1 时，那么数据 Data0 将会被丢弃，I2S 将从 Data1 开始接收数据；此时若 I2S_RX_MSB_RIGHT = 1，那么第一阶段数据为 $\{0xFEDC0000, 0x32100000\}$，若 I2S_RX_MSB_RIGHT = 0，那么第一阶段数据为 $\{0x32100000, 0xFEDC0000\}$。当 I2S_RX_RIGHT_FIRST = 0 时，I2S 将从 Data0 开始接收数据；如果 I2S_RX_MSB_RIGHT = 1，那么第一阶段数据为 $\{0xFEDC0000, 0x76540000\}$，如果 I2S_RX_MSB_RIGHT = 0，那么第一阶段数据为 $\{0x76540000, 0xFEDC0000\}$。

在第二阶段，将接收模块接收到的数据写入 FIFO。接收数据写入 FIFO 的模式共有四种，对应 I2S_RX_FIFO_MOD[2:0] 比特取值，如表 12-4 和图 12-8 所示。

表 12-4. 接收数据写入 FIFO 模式和对应寄存器配置

I2S_RX_FIFO_MOD[2:0]	数据格式
0	16 位双通道数据
1	16 位单通道数据
2	32 位双通道数据
3	32 位单通道数据

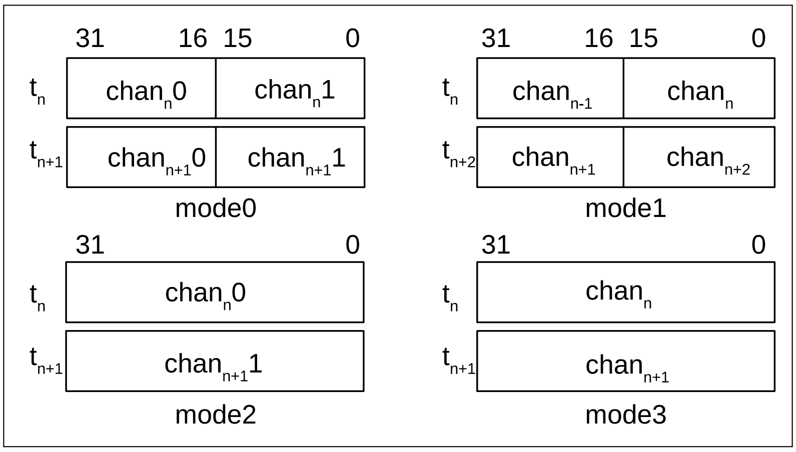


图 12-8. 接收数据写入 FIFO 模式

在第三阶段，CPU 或者 DMA 会将数据从 FIFO 中读出，并直接写到内部存储区域。各模式对应寄存器配置见表 12-5。

表 12-5. 4 种模式对应寄存器配置

I2S_RX_MSB_RIGHT	I2S_RX_CHAN_MOD	mode0	mode1	mode2	mode3
0	0	左声道 + 右声道	-	左声道 + 右声道	-
	1		左声道 + 左声道		左声道 + 左声道
	2		右声道 + 右声道		右声道 + 右声道
	3		-		-
1	0	右声道 + 左声道	-	右声道 + 左声道	-
	1		右声道 + 右声道		右声道 + 右声道
	2		左声道 + 左声道		左声道 + 左声道
	3		-		-

12.4.6 I2S 主机/从机模式

I2S 模块可以配置为主机接收/发送接口，支持半双工模式和全双工模式；也可以配置为从机接收/发送接口，也支持半双工模式和全双工模式。

寄存器 I2S_CONF_REG 的 I2S_RX_SLAVE_MOD 比特和 I2S_TX_SLAVE_MOD 比特分别用于将 I2S 配置为从机接收和从机发送模式。

寄存器 I2S_CONF_REG 中的 I2S_TX_START 位用于启动一次发送操作。当 I2S 为主机发送模式时，若该位置 1，发送模块会一直输出时钟信号和左右声道数据。如果 FIFO 将所有写入的数据发送完毕，并且没有新数据填入，那么数据线上会循环输出最后一帧数据。当该位被清零时，主机停止输出时钟和数据。当 I2S 被配置为从机发送时，若该位被置 1，发送模块会等待主机 BCK 时钟，来启动发送操作。

寄存器 I2S_CONF_REG 中的 I2S_RX_START 位用于启动一次接收操作。当 I2S 为主机接收模式时，若该位置 1，接收模块会一直输出时钟信号，并对输入数据进行采样，直到该位被清零。如果 I2S 配置为从机接收模式时，若该位被置 1，接收模块会等待主机 BCK 时钟，来启动接收操作。

12.4.7 I2S PDM 模式

如图 12-1 所示，ESP32 I2S0 内部集成了 PDM 模块，用于 PCM 编码信号和 PDM 编码信号之间相互转换。

PDM 模块输出时钟信号映射到 I2S0*_WS_out 信号，其配置方式和 I2S 模式下的 BCK 相同，具体请参考章节 12.3。I2S PDM 接收和发送的 PCM 信号数据位宽均为 16 位。

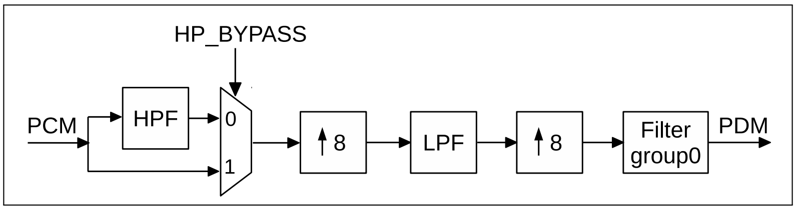


图 12-9. PDM 发送模块

PDM 发送模块用于将 PCM 信号转换为 PDM 信号，如图 12-9 所示。HPF 为高通滤波器，LPF 为低通滤波器。PCM 信号经过上采样和滤波，最终输出 PDM 信号。寄存器 I2S_PDM_CONF_REG 的 I2S_TX_PDM_HP_BYPASS 信号用于选择 PCM 信号是否经过 HPF 模块。Filter group0 模块具有上采样功能。假设 PDM 信号的频率为 f_{pdm}，PCM 信号的频率为 f_{pcm}，那么 f_{pdm} 与 f_{pcm} 之间的关系为：

$$f_{\mathrm{pdm}} = 64 \times f_{\mathrm{pcm}} \times \frac{I2S\_TX\_PDM\_FP}{I2S\_TX\_PDM\_FS}$$

其中 64 为前面两次过采样率的乘积。

如表 12-6 所示，不同 PCM 信号的频率下，可以通过配置寄存器 I2S_PDM_FREQ_CONF_REG 的 I2S_TX_PDM_FP 比特和 I2S_TX_PDM_FS 比特，得到输出频率均为 48×128 KHz 的 PDM 信号。

表 12-6. 过采样率配置

f_{pcm} (KHz)	I2S_TX_PDM_FP	I2S_TX_PDM_FS	f_{pdm} (KHz)
48	960	480	48×128
44.1	960	441	
32	960	320	
24	960	240	
16	960	160	
8	960	80	

寄存器 I2S_PDM_CONF_REG 的 I2S_TX_PDM_SINC_OSR2 比特表示 Filter group0 模块的过采样率。

$$I2S\_TX\_PDM\_SINC\_OSR2 = \left\lfloor \frac{I2S\_TX\_PDM\_FP}{I2S\_TX\_PDM\_FS} \right\rfloor$$

当使用 PDM 发送模块时，需要将寄存器 I2S_PDM_CONF_REG 的 I2S_TX_PDM_EN 比特与 I2S_PCM2PDM_CONV_EN 比特全部置 1，如图 12-10 所示。寄存器 I2S_PDM_CONF_REG 的 I2S_TX_PDM_SIGMADELTA_IN_SHIFT 比特、I2S_TX_PDM_SINC_IN_SHIFT 比特、I2S_TX_PDM_LP_IN_SHIFT 比特和 I2S_TX_PDM_HP_IN_SHIFT 比特用于调整各个滤波模块的输入信号的大小。

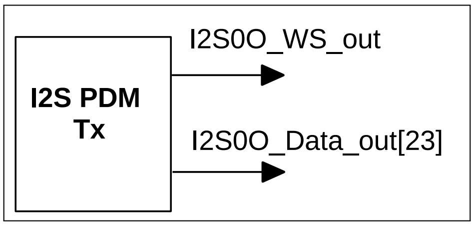


图 12-10. PDM 发送信号

当使用 PDM 接收模块时，需要将寄存器 I2S_PDM_CONF_REG 的 I2S_RX_PDM_EN 比特与 I2S_PDM2PCM_CONV_EN 比特全部置为 1，如图 12-11 所示。PDM 接收模块结构将接收到的 PDM 信号转换为 16 比特的 PCM 信号。图 12-12 中 Filter group1 用于对 PDM 信号进行下采样，寄存器 I2S_PDM_CONF_REG 的 I2S_RX_PDM_SINC_DSR_16_EN 用于调整下采样率。PDM 接收模块的输入 PDM 信号频率需要为 PCM 信号 128 或 64 倍。

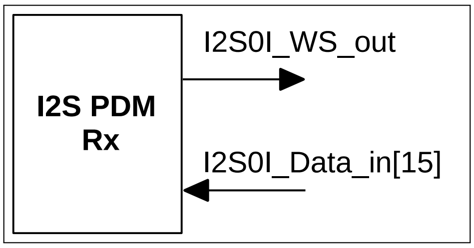


图 12-11. PDM 接收信号

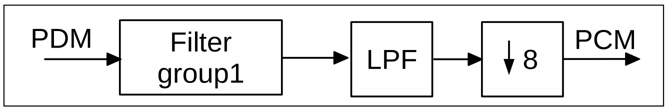


图 12-12. PDM 接收模块

下表 12-7 给出不同 PDM 信号的频率下，PDM 信号转 PCM 信号的 I2S_RX_PDM_SINC_DSR_16_EN 比特配置。

表 12-7. 下采样配置

PDM freq (KHz)	I2S_RX_PDM_SINC_DSR_16_EN	PCM freq (KHz)
$f_{\text{pcm}}\times128$	1	f_{pcm}
$f_{\text{pcm}}\times64$	0	

12.5 Camera-LCD 控制器

ESP32 I2S 的 LCD 模式分为：

- LCD 主机发送模式
- Camera 从机接收模式
- ADC/DAC 模式

LCD 模式的时钟配置与 I2S 模式的时钟配置一致。LCD 模式下，WS 频率为 f_{BCK} 的一半。

ADC/DAC 模式下，要使用 PLL_F160M_CLK 作为时钟源。

12.5.1 LCD 主机发送模式

如图 12-13 所示，在 LCD 主机发送模式下，LCD 的 WR 信号接 I2S 模块的 WS 信号，数据信号线宽度为 24 比特。

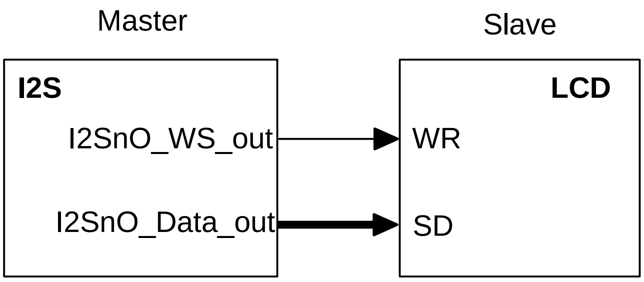


图 12-13. LCD 主机发送模式

首先将寄存器 I2S_CONF2_REG 的 I2S_LCD_EN 比特置为 1，并且将寄存器 I2S_CONF_REG 的 I2S_TX_SLAVE_MOD 比特置 0，使得 I2S 模块工作在 LCD 主机发送模式。同时，根据需要配置寄存器 I2S_CONF_CHAN_REG 的 I2S_TX_CHAN_MOD[2:0] 比特和寄存器 I2S_FIFO_CONF_REG 的 I2S_TX_FIFO_MOD[2:0] 比特，以使用正确的模式发送数据。WS 信号经过 GPIO 交换矩阵输出时需要反相。详情请参考章节 IO_MUX 和 GPIO 交换矩阵。在 LCD 模式主机发送模式下，还需要通过配置寄存器 I2S_CONF2_REG 的 I2S_LCD_TX_SDX2_EN 比特和 I2S_LCD_TX_WRX2_EN 比特，使得发送数据和 WR 信号工作在需要的模式下。

如图 12-14 和图 12-15 所示，数据帧格式 1 下，需要将 I2S_LCD_TX_WRX2_EN 比特置 1，I2S_LCD_TX_SDX2_EN 比特置 0 。数据帧格式 2 下，需要将 I2S_LCD_TX_SDX2_EN 比特和 I2S_LCD_TX_WRX2_EN 比特都置 1。

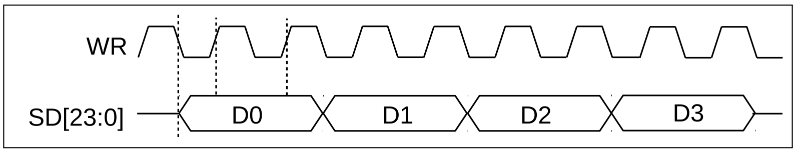


图 12-14. LCD 主机发送数据帧格式 1

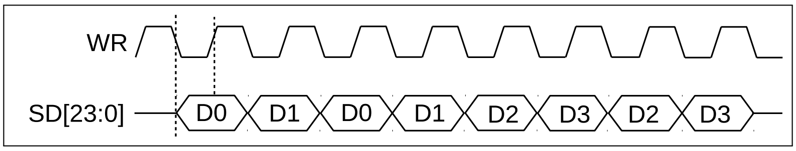


图 12-15. LCD 主机发送数据帧格式 2

12.5.2 Camera 从机接收模式

ESP32 I2S 可以配置成 Camera 从机模式，以此实现与外部 camera 模块的高速数据传输。在此模式下，I2S 模块为从机接收模式，除了 16 路数据信号总线 I2S*n*I_Data_in 外，还有 I2S*n*_H_SYNC、I2S*n*_V_SYNC 和 I2S*n*_H_ENABLE 信号。

Camera 的 PCLK 接 I2S 模块的 I2S*n*I_WS_in 信号，如图 12-16 所示。

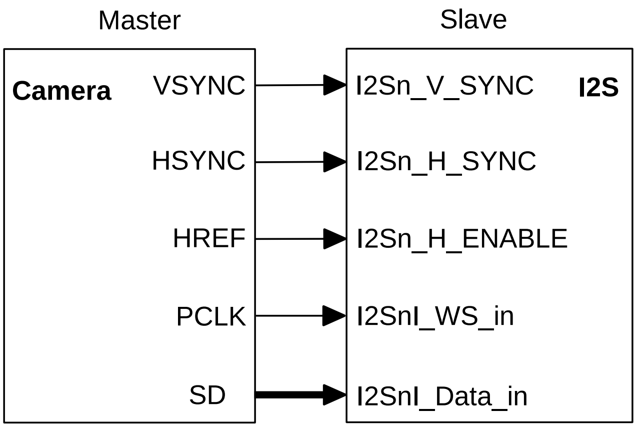


图 12-16. Camera 从机接收模式

当 I2S 为 Camera 从机接收模式时，并且当 I2S*n*_H_SYNC、I2S_V_SYNC 和 I2S_H_REF 均为高电平时，认为主机开始传输数据，即：

$$transmission\_start = (I2Sn\_H\_SYNC == 1)\&\&(I2Sn\_V\_SYNC == 1)\&\&(I2Sn\_H\_ENABLE == 1)$$

即在传输数据过程中，这三个信号需要保持高电平。例如某款 camera，I2S*n*_V_SYNC 信号在传输数据过程中为低电平，那么在输入 I2S 模块时 I2S*n*_V_SYNC 需要取反。ESP32 支持信号经过 GPIO 交换矩阵时取反。详情请参考章节 IO_MUX 和 GPIO 交换矩阵。

为了使 I2S 工作在 Camera 模式，寄存器 I2S_CONF2_REG 的 I2S_LCD_EN 比特和 I2S_CAMERA_EN 比特需要置为 1，并且将寄存器 I2S_CONF_REG 的 I2S_RX_SLAVE_MOD 比特置 1，I2S_RX_MSB_RIGHT 比特和 I2S_RX_RIGHT_FIRST 比特均设置为 0，使得 I2S 模块工作在 LCD 从机接收模式。同时，需要将寄存器 I2S_CONF_CHAN_REG 的 I2S_RX_CHAN_MOD[2:0] 比特和寄存器 I2S_FIFO_CONF_REG 的 I2S_RX_FIFO_MOD[2:0] 比特均配置成 1，以使用正确的模式接收数据。

12.5.3 ADC/DAC 模式

LCD 模式使得片上 ADC 模块接收到的数据可以使用 I2S0 模块搬到内部存储区，也可以使用 I2S0 模块将内部存储区的数据搬到片上 DAC 模块。当 I2S0 模块连接片上 ADC 时，需要将 I2S0 模块配置为主机接收模式。图 12-17 为 I2S0 模块与 ADC 控制器之间的信号连接。

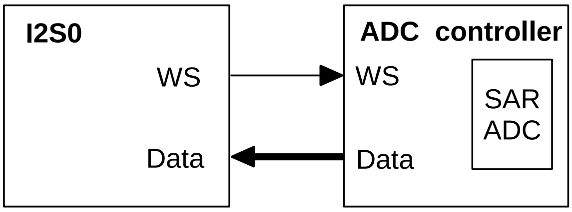


图 12-17. I2S 的 ADC 接口

首先将寄存器 I2S_CONF2_REG 的 I2S_LCD_EN 比特置为 1，并且将寄存器 I2S_CONF_REG 的 I2S_RX_SLAVE_MOD 比特置 0，使得 I2S0 模块工作在 LCD 主机接收模式，配置 I2S0 模块时钟，使得 I2S0 的 WS 信号输出合适的频率。然后将 SYSCON_APB_SARADC_CTRL_REG 寄存器的 SYSCON_SARADC_DATA_TO_I2S 比特置 1 。在配置好 SARADC 相关寄存器后启动 I2S 接收数据。详情请参考章节片上传感器与模拟信号处理。

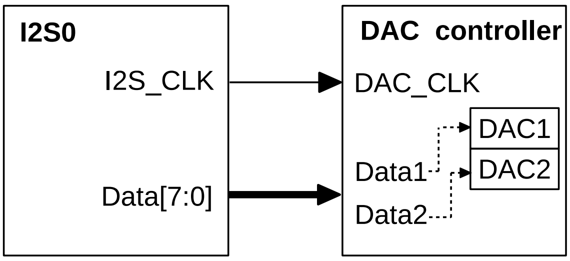


图 12-18. I2S 的 DAC 接口

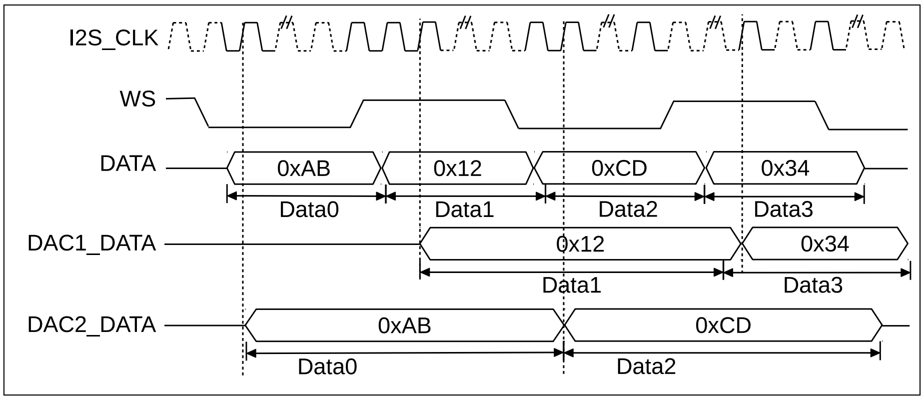


图 12-19. I2S DAC 接口数据输入

当 I2S0 模块接片内 DAC 时，需要将 I2S0 模块配置成主机发送模式。图 12-18 为 I2S0 模块与 DAC 控制器之间的信号连接。DAC 控制模块以 I2S_CLK 为时钟，此时 I2S_CLK 最高为 APB_CLK/2。如图 12-19 所示，当数据总线向 DAC 控制模块输入数据时，DAC 控制器会将右声道数据输入 DAC1 模块，将左声道数据输入 DAC2 模块。使用 I2S DMA 模块时，需要将待发送的 8 位数据左移 8 位填入 DMA 的双字节类型 buffer 中。

使用 I2S0 的 DAC 模式时，首先将寄存器 I2S_CONF2_REG 的 I2S_LCD_EN 比特置为 1，分别将 I2S_RX_SHORT_SYNC、I2S_TX_SHORT_SYNC、I2S_CONF_REG、I2S_RX_MSB_SHIFT 和 I2S_TX_MSB_SHIFT 清零，并且将寄存器 I2S_CONF_REG 的 I2S_TX_SLAVE_MOD 比特置 0，使得 I2S0 模块工作在 LCD 主机发送模式。按照传输 16 位数据的标准，选择合适的发送模式。配置 I2S0 模块时钟，使得 I2S 的 I2S_CLK 和 WS 输出合适的频率。在配置好 DAC 相关寄存器后，启动 I2S0 发送数据。

12.6 I2S 中断

12.6.1 FIFO 中断

- I2S_TX_HUNG_INT：当发送数据超时即触发此中断。
- I2S_RX_HUNG_INT：当接收数据超时即触发此中断。
- I2S_TX_REMPTY_INT：当发送 FIFO 为空时即触发此中断。
- I2S_TX_WFULL_INT：当发送 FIFO 满时即触发此中断。
- I2S_RX_REMPTY_INT：当接收 FIFO 为空时即触发此中断。
- I2S_RX_WFULL_INT：当接收 FIFO 满时即触发此中断。
- I2S_TX_PUT_DATA_INT：当发送 FIFO 将要空时即触发此中断。

- I2S_RX_TAKE_DATA_INT：当接收 FIFO 将要满时即触发此中断。

12.6.2 DMA 中断

- I2S_OUT_TOTAL_EOF_INT：当所有发送链表使用完毕时即触发此中断。
- I2S_IN_DSCR_EMPTY_INT：无有效接收链表可用时即触发此中断。
- I2S_OUT_DSCR_ERR_INT：当遇到无效发送链表描述符时即触发此中断。
- I2S_IN_DSCR_ERR_INT：当遇到无效接收链表描述符时即触发此中断。
- I2S_OUT_EOF_INT：当发送链表完成发送一个数据包时即触发此中断。
- I2S_OUT_DONE_INT：当所有发送缓存数据被读取完毕后即触发此中断。
- I2S_IN_SUC_EOF_INT：当所有数据接收完毕时即触发此中断。
- I2S_IN_DONE_INT：当前接收链表描述符被处理时即触发此中断。

12.7 寄存器列表

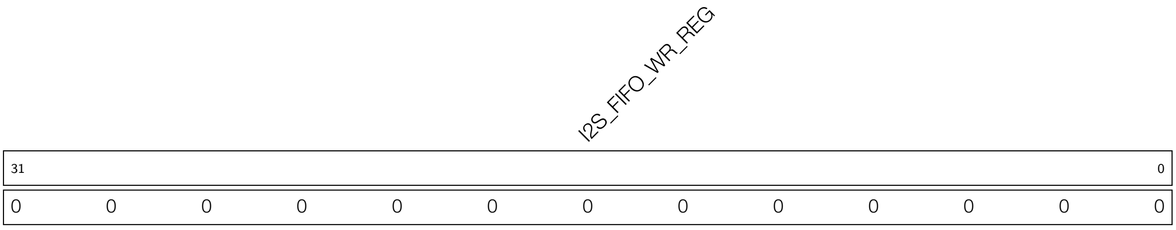
名称	描述	I2S0	I2S1	访问
I²S FIFO 寄存器				
I2S_FIFO_WR_REG	写入 I²S 发送 FIFO 的数据	0x3FF4F000	0x3FF6D000	只写
I2S_FIFO_RD_REG	存储 I²S 接收 FIFO 的数据	0x3FF4F004	0x3FF6D004	只读
配置寄存器				
I2S_CONF_REG	配置与开始/停止位	0x3FF4F008	0x3FF6D008	读/写
I2S_CONF1_REG	PCM 配置寄存器	0x3FF4F0A0	0x3FF6D0A0	读/写
I2S_CONF2_REG	ADC/LCD/Camera 配置寄存器	0x3FF4F0A8	0x3FF6D0A8	读/写
I2S_TIMING_REG	信号延迟和时序参数	0x3FF4F01C	0x3FF6D01C	读/写
I2S_FIFO_CONF_REG	FIFO 配置	0x3FF4F020	0x3FF6D020	读/写
I2S_CONF_SINGLE_DATA_REG	静态通道输出值	0x3FF4F028	0x3FF6D028	读/写
I2S_CONF_CHAN_REG	通道配置	0x3FF4F02C	0x3FF6D02C	读/写
I2S_LC_HUNG_CONF_REG	超时检测配置	0x3FF4F074	0x3FF6D074	读/写
I2S_CLKM_CONF_REG	Bitclock 配置	0x3FF4F0AC	0x3FF6D0AC	读/写
I2S_SAMPLE_RATE_CONF_REG	采样率配置	0x3FF4F0B0	0x3FF6D0B0	读/写
I2S_PD_CONF_REG	Power-down 寄存器	0x3FF4F0A4	0x3FF6D0A4	读/写
I2S_STATE_REG	I²S 状态	0x3FF4F0BC	0x3FF6D0BC	只读
中断寄存器				
DMA 寄存器				
I2S_LC_CONF_REG	DMA 配置寄存器	0x3FF4F060	0x3FF6D060	读/写
I2S_RXEOF_NUM_REG	接收数据计数器	0x3FF4F024	0x3FF6D024	读/写
I2S_OUT_LINK_REG	DMA 发送链表配置和地址	0x3FF4F030	0x3FF6D030	读/写
I2S_IN_LINK_REG	DMA 接收链表配置和地址	0x3FF4F034	0x3FF6D034	读/写
I2S_OUT_EOF_DES_ADDR_REG	生成 EOF 的发送链表描述符地址	0x3FF4F038	0x3FF6D038	只读
I2S_IN_EOF_DES_ADDR_REG	生成 EOF 的接收链表描述符地址	0x3FF4F03C	0x3FF6D03C	只读

I2S_OUT_EOF_BFR_DES_ADDR_REG	生成 EOF 的接收缓存地址	0x3FF4F040	0x3FF6D040	只读
I2S_INLINK_DSCR_REG	当前接收链表描述符的地址	0x3FF4F048	0x3FF6D048	只读
I2S_INLINK_DSCR_BF0_REG	下一个接收链表描述符的地址	0x3FF4F04C	0x3FF6D04C	只读
I2S_INLINK_DSCR_BF1_REG	下一个接收链表数据 buffer 的地址	0x3FF4F050	0x3FF6D050	只读
I2S_OUTLINK_DSCR_REG	当前发送链表描述符的地址	0x3FF4F054	0x3FF6D054	只读
I2S_OUTLINK_DSCR_BF0_REG	下一个发送链表描述符的地址	0x3FF4F058	0x3FF6D058	只读
I2S_OUTLINK_DSCR_BF1_REG	下一个发送链表数据 buffer 的地址	0x3FF4F05C	0x3FF6D05C	只读
I2S_LC_STATE0_REG	DMA 接收状态	0x3FF4F06C	0x3FF6D06C	只读
I2S_LC_STATE1_REG	DMA 发送状态	0x3FF4F070	0x3FF6D070	只读
脉冲密度 (DE) 调制寄存器				
I2S_PDM_CONF_REG	PDM 配置	0x3FF4F0B4	0x3FF6D0B4	读/写
I2S_PDM_FREQ_CONF_REG	PDM 频率	0x3FF4F0B8	0x3FF6D0B8	读/写
I2S_INT_RAW_REG	原始中断状态	0x3FF4F00C	0x3FF6D00C	只读
I2S_INT_ST_REG	屏蔽中断状态	0x3FF4F010	0x3FF6D010	只读
I2S_INT_ENA_REG	中断使能位	0x3FF4F014	0x3FF6D014	读/写
I2S_INT_CLR_REG	中断清除位	0x3FF4F018	0x3FF6D018	只写

12.8 寄存器

本小节括号中的地址均为相对于 I2S 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 12.7 寄存器列表。

Register 12.1. I2S_FIFO_WR_REG (0x0000)



I2S_FIFO_WR_REG 写入 I2S 发送 FIFO 的数据。(只写)

Register 12.2. I2S_FIFO_RD_REG (0x0004)

I2S_FIFO_RD_REG 存储 I2S 接收 FIFO 的数据。(只读)

Register 12.3. I2S_CONF_REG (0x0008)

(reserved)	I2S_SIG_LOOPBACK	I2S_RX_MSB_RIGHT	I2S_TX_MSB_RIGHT	I2S_RX_MONO	I2S_TX_MONO	I2S_RX_SHORT_SYNC	I2S_TX_SHORT_SYNC	I2S_RX_MSB_SHIFT	I2S_TX_MSB_SHIFT	I2S_RX_RIGHT_FIRST	I2S_TX_RIGHT_FIRST	I2S_RX_SLAVE_MOD	I2S_TX_SLAVE_MOD	I2S_RX_START	I2S_TX_START	I2S_RX_FIFO_RESET	I2S_TX_FIFO_RESET	I2S_RX_RESET	I2S_TX_RESET	
31 19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_SIG_LOOPBACK 置 1 时，发送模块和接收模块共享 WS 和 BCK 信号。（读/写）

I2S_RX_MSB_RIGHT 此位置 1 时，将接收到的右声道数据放在 FIFO 的最高有效位。（读/写）

I2S_TX_MSB_RIGHT 此位置 1 时，将要发送的右声道数据放在 FIFO 的最高有效位。（读/写）

I2S_RX_MONO 置位使能 PCM 标准模式下接收模块的单声道模式。（读/写）

I2S_TX_MONO 置位使能 PCM 标准模式下发送模块的单声道模式。（读/写）

I2S_RX_SHORT_SYNC 置位使能 PCM 标准模式下的接收器。（读/写）

I2S_TX_SHORT_SYNC 置位使能 PCM 标准模式下的发送器。（读/写）

I2S_RX_MSB_SHIFT 置位使能 Philips 标准模式下的接收器。（读/写）

I2S_TX_MSB_SHIFT 置位使能 Philips 标准模式下的发送器。（读/写）

I2S_RX_RIGHT_FIRST 将此位置 1，先接收右声道数据。（读/写）

I2S_TX_RIGHT_FIRST 将此位置 1，先发送右声道数据。（读/写）

I2S_RX_SLAVE_MOD 将此位置 1，使能从机接收模式。（读/写）

I2S_TX_SLAVE_MOD 将此位置 1，使能从机发送模式。（读/写）

I2S_RX_START 置位开始接收数据。（读/写）

I2S_TX_START 置位开始发送数据。（读/写）

I2S_RX_FIFO_RESET 置位重置接收 FIFO。（读/写）

I2S_TX_FIFO_RESET 置位重置发送 FIFO。（读/写）

I2S_RX_RESET 置位重置接收器。（读/写）

I2S_TX_RESET 置位重置发送器。（读/写）

Register 12.4. I2S_INT_RAW_REG (0x000c)

(reserved)	I2S_OUT_TOTAL_EOF_INT_RAW	I2S_IN_DSCR_EMPTY_INT_RAW	I2S_OUT_DSCR_ERR_INT_RAW	I2S_IN_DSCR_ERR_INT_RAW	I2S_OUT_EOF_INT_RAW	I2S_OUT_DONE_INT_RAW	(reserved)	I2S_IN_SUC_EOF_INT_RAW	I2S_IN_DONE_INT_RAW	I2S_TX_HUNG_INT_RAW	I2S_RX_HUNG_INT_RAW	I2S_TX_REMPTY_INT_RAW	I2S_TX_WFULL_INT_RAW	I2S_RX_REMPTY_INT_RAW	I2S_RX_WFULL_INT_RAW	I2S_TX_PUT_DATA_INT_RAW	I2S_RX_TAKE_DATA_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	0	0	0	0	Reset

I2S_OUT_TOTAL_EOF_INT_RAW I2S_OUT_TOTAL_EOF_INT 的原始中断状态位。（只读）

I2S_IN_DSCR_EMPTY_INT_RAW I2S_IN_DSCR_EMPTY_INT 的原始中断状态位。（只读）

I2S_OUT_DSCR_ERR_INT_RAW I2S_OUT_DSCR_ERR_INT 的原始中断状态位。（只读）

I2S_IN_DSCR_ERR_INT_RAW I2S_IN_DSCR_ERR_INT 的原始中断状态位。（只读）

I2S_OUT_EOF_INT_RAW I2S_OUT_EOF_INT 的原始中断状态位。（只读）

I2S_OUT_DONE_INT_RAW I2S_OUT_DONE_INT 的原始中断状态位。（只读）

I2S_IN_SUC_EOF_INT_RAW I2S_IN_SUC_EOF_INT 的原始中断状态位。（只读）

I2S_IN_DONE_INT_RAW I2S_IN_DONE_INT 的原始中断状态位。（只读）

I2S_TX_HUNG_INT_RAW I2S_TX_HUNG_INT 的原始中断状态位。（只读）

I2S_RX_HUNG_INT_RAW I2S_RX_HUNG_INT 的原始中断状态位。（只读）

I2S_TX_REMPTY_INT_RAW I2S_TX_REMPTY_INT 的原始中断状态位。（只读）

I2S_TX_WFULL_INT_RAW I2S_TX_WFULL_INT 的原始中断状态位。（只读）

I2S_RX_REMPTY_INT_RAW I2S_RX_REMPTY_INT 的原始中断状态位。（只读）

I2S_RX_WFULL_INT_RAW I2S_RX_WFULL_INT 的原始中断状态位。（只读）

I2S_TX_PUT_DATA_INT_RAW I2S_TX_PUT_DATA_INT 的原始中断状态位。（只读）

I2S_RX_TAKE_DATA_INT_RAW TI2S_RX_TAKE_DATA_INT 的原始中断状态位。（只读）

Register 12.5. I2S_INT_ST_REG (0x0010)

(reserved)	I2S_OUT_TOTAL_EOF_INT_ST	I2S_IN_DSCR_EMPTY_INT_ST	I2S_OUT_DSCR_ERR_INT_ST	I2S_IN_DSCR_ERR_INT_ST	I2S_OUT_EOF_INT_ST	I2S_OUT_DONE_INT_ST	(reserved)	I2S_IN_SUC_EOF_INT_ST	I2S_TX_DONE_INT_ST	I2S_TX_HUNG_INT_ST	I2S_RX_HUNG_INT_ST	I2S_TX_REMPTY_INT_ST	I2S_TX_WFULL_INT_ST	I2S_RX_REMPTY_INT_ST	I2S_RX_WFULL_INT_ST	I2S_TX_PUT_DATA_INT_ST	I2S_RX_TAKE_DATA_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	Reset

I2S_OUT_TOTAL_EOF_INT_ST I2S_OUT_TOTAL_EOF_INT 的屏蔽中断状态位。(只读)

I2S_IN_DSCR_EMPTY_INT_ST I2S_IN_DSCR_EMPTY_INT 的屏蔽中断状态位。(只读)

I2S_OUT_DSCR_ERR_INT_ST I2S_OUT_DSCR_ERR_INT 的屏蔽中断状态位。(只读)

I2S_IN_DSCR_ERR_INT_ST I2S_IN_DSCR_ERR_INT 的屏蔽中断状态位。(只读)

I2S_OUT_EOF_INT_ST I2S_OUT_EOF_INT 的屏蔽中断状态位。(只读)

I2S_OUT_DONE_INT_ST I2S_OUT_DONE_INT 的屏蔽中断状态位。(只读)

I2S_IN_SUC_EOF_INT_ST I2S_IN_SUC_EOF_INT 的屏蔽中断状态位。(只读)

I2S_IN_DONE_INT_ST I2S_IN_DONE_INT 的屏蔽中断状态位。(只读)

I2S_TX_HUNG_INT_ST I2S_TX_HUNG_INT 的屏蔽中断状态位。(只读)

I2S_RX_HUNG_INT_ST I2S_RX_HUNG_INT 的屏蔽中断状态位。(只读)

I2S_TX_REMPTY_INT_ST I2S_TX_REMPTY_INT 的屏蔽中断状态位。(只读)

I2S_TX_WFULL_INT_ST I2S_TX_WFULL_INT 的屏蔽中断状态位。(只读)

I2S_RX_REMPTY_INT_ST I2S_RX_REMPTY_INT 的屏蔽中断状态位。(只读)

I2S_RX_WFULL_INT_ST I2S_RX_WFULL_INT 的屏蔽中断状态位。(只读)

I2S_TX_PUT_DATA_INT_ST I2S_TX_PUT_DATA_INT 的屏蔽中断状态位。(只读)

I2S_RX_TAKE_DATA_INT_ST I2S_RX_TAKE_DATA_INT 的屏蔽中断状态位。(只读)

Register 12.6. I2S_INT_ENA_REG (0x0014)

	(reserved)	I2S_OUT_TOTAL_EOF_INT_ENA	I2S_IN_DSCR_EMPTY_INT_ENA	I2S_OUT_DSCR_ERR_INT_ENA	I2S_IN_DSCR_ERR_INT_ENA	I2S_OUT_EOF_INT_ENA	I2S_OUT_DONE_INT_ENA	(reserved)	I2S_IN_SUC_EOF_INT_ENA	I2S_IN_DONE_INT_ENA	I2S_TX_HUNG_INT_ENA	I2S_RX_HUNG_INT_ENA	I2S_TX_REMPTY_INT_ENA	I2S_TX_WFULL_INT_ENA	I2S_RX_REMPTY_INT_ENA	I2S_RX_WFULL_INT_ENA	I2S_TX_PUT_DATA_INT_ENA	I2S_RX_TAKE_DATA_INT_ENA
	31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

I2S_OUT_TOTAL_EOF_INT_ENA I2S_OUT_TOTAL_EOF_INT 的中断使能位。（读/写）

I2S_IN_DSCR_EMPTY_INT_ENA I2S_IN_DSCR_EMPTY_INT 的中断使能位。（读/写）

I2S_OUT_DSCR_ERR_INT_ENA I2S_OUT_DSCR_ERR_INT 的中断使能位。（读/写）

I2S_IN_DSCR_ERR_INT_ENA I2S_IN_DSCR_ERR_INT 的中断使能位。（读/写）

I2S_OUT_EOF_INT_ENA I2S_OUT_EOF_INT 的中断使能位。（读/写）

I2S_OUT_DONE_INT_ENA I2S_OUT_DONE_INT 的中断使能位。（读/写）

I2S_IN_SUC_EOF_INT_ENA I2S_IN_SUC_EOF_INT 的中断使能位。（读/写）

I2S_IN_DONE_INT_ENA I2S_IN_DONE_INT 的中断使能位。（读/写）

I2S_TX_HUNG_INT_ENA I2S_TX_HUNG_INT 的中断使能位。（读/写）

I2S_RX_HUNG_INT_ENA I2S_RX_HUNG_INT 的中断使能位。（读/写）

I2S_TX_REMPTY_INT_ENA I2S_TX_REMPTY_INT 的中断使能位。（读/写）

I2S_TX_WFULL_INT_ENA I2S_TX_WFULL_INT 的中断使能位。（读/写）

I2S_RX_REMPTY_INT_ENA I2S_RX_REMPTY_INT 的中断使能位。（读/写）

I2S_RX_WFULL_INT_ENA I2S_RX_WFULL_INT 的中断使能位。（读/写）

I2S_TX_PUT_DATA_INT_ENA I2S_TX_PUT_DATA_INT 的中断使能位。（读/写）

I2S_RX_TAKE_DATA_INT_ENA I2S_RX_TAKE_DATA_INT 的中断使能位。（读/写）

Register 12.7. I2S_INT_CLR_REG (0x0018)

(reserved)	I2S_OUT_TOTAL_EOF_INT_CLR	I2S_IN_DSCR_EMPTY_INT_CLR	I2S_OUT_DSCR_ERR_INT_CLR	I2S_IN_DSCR_ERR_INT_CLR	I2S_OUT_EOF_INT_CLR	I2S_OUT_DONE_INT_CLR	(reserved)	I2S_IN_SUC_EOF_INT_CLR	I2S_IN_DONE_INT_CLR	I2S_TX_HUNG_INT_CLR	I2S_RX_HUNG_INT_CLR	I2S_TX_REMPTY_INT_CLR	I2S_TX_WFULL_INT_CLR	I2S_RX_REMPTY_INT_CLR	I2S_RX_WFULL_INT_CLR	I2S_TX_PUT_DATA_INT_CLR	I2S_RX_TAKE_DATA_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	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUT_TOTAL_EOF_INT_CLR I2S_OUT_TOTAL_EOF_INT 的中断清除位。（只写）

I2S_IN_DSCR_EMPTY_INT_CLR I2S_IN_DSCR_EMPTY_INT 的中断清除位。（只写）

I2S_OUT_DSCR_ERR_INT_CLR I2S_OUT_DSCR_ERR_INT 的中断清除位。（只写）

I2S_IN_DSCR_ERR_INT_CLR I2S_IN_DSCR_ERR_INT 的中断清除位。（只写）

I2S_OUT_EOF_INT_CLR I2S_OUT_EOF_INT 的中断清除位。（只写）

I2S_OUT_DONE_INT_CLR I2S_OUT_DONE_INT 的中断清除位。（只写）

I2S_IN_SUC_EOF_INT_CLR I2S_IN_SUC_EOF_INT 的中断清除位。（只写）

I2S_IN_DONE_INT_CLR I2S_IN_DONE_INT 的中断清除位。（只写）

I2S_TX_HUNG_INT_CLR I2S_TX_HUNG_INT 的中断清除位。（只写）

I2S_RX_HUNG_INT_CLR I2S_RX_HUNG_INT 的中断清除位。（只写）

I2S_TX_REMPTY_INT_CLR I2S_TX_REMPTY_INT 的中断清除位。（只写）

I2S_TX_WFULL_INT_CLR I2S_TX_WFULL_INT 的中断清除位。（只写）

I2S_RX_REMPTY_INT_CLR I2S_RX_REMPTY_INT 的中断清除位。（只写）

I2S_RX_WFULL_INT_CLR I2S_RX_WFULL_INT 的中断清除位。（只写）

I2S_TX_PUT_DATA_INT_CLR I2S_TX_PUT_DATA_INT 的中断清除位。（只写）

I2S_RX_TAKE_DATA_INT_CLR I2S_RX_TAKE_DATA_INT 的中断清除位。（只写）

Register 12.8. I2S_TIMING_REG (0x001c)

(reserved)	I2S_TX_BCK_IN_INV	I2S_DATA_ENABLE_DELAY	I2S_RX_DSYNC_SW	I2S_TX_DSYNC_SW	I2S_RX_BCK_OUT_DELAY	I2S_RX_WS_OUT_DELAY	I2S_TX_SD_OUT_DELAY	I2S_TX_WS_OUT_DELAY	I2S_TX_BCK_OUT_DELAY	I2S_RX_SD_IN_DELAY	I2S_RX_WS_IN_DELAY	I2S_RX_BCK_IN_DELAY	I2S_TX_WS_IN_DELAY	I2S_TX_BCK_IN_DELAY	
31 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

I2S_TX_BCK_IN_INV 将此位置 1，当 BCK 信号进入从机发送模块时会被反转。(读/写)

I2S_DATA_ENABLE_DELAY 数据有效标志位的延迟周期数。(读/写)

I2S_RX_DSYNC_SW 将此位置 1，在 double sync 模式下将信号同步于接收模块中。(读/写)

I2S_TX_DSYNC_SW 将此位置 1，在 double sync 模式下将信号同步于发送模块中。(读/写)

I2S_RX_BCK_OUT_DELAY 接收模式下 BCK 输出的延迟周期数。(读/写)

I2S_RX_WS_OUT_DELAY 接收模式下 WS 信号的延迟周期数。(读/写)

I2S_TX_SD_OUT_DELAY 发送模式下 SD 信号的延迟周期数。(读/写)

I2S_TX_WS_OUT_DELAY 发送模式下 WS 信号的延迟周期数。(读/写)

I2S_TX_BCK_OUT_DELAY 发送模式下 BCK 信号的延迟周期数。(读/写)

I2S_RX_SD_IN_DELAY 接收模式下 SD 输入的延迟周期数。(读/写)

I2S_RX_WS_IN_DELAY 接收模式下 WS 输入的延迟周期数。(读/写)

I2S_RX_BCK_IN_DELAY 接收模式下 BCK 输入的延迟周期数。(读/写)

I2S_TX_WS_IN_DELAY 从机发送模式下 WS 信号的延迟周期数。(读/写)

I2S_TX_BCK_IN_DELAY 从机发送模式下 BCK 输入的延迟周期数。(读/写)

Register 12.9. I2S_FIFO_CONF_REG (0x0020)

(reserved)	I2S_RX_FIFO_MOD_FORCE_EN	I2S_TX_FIFO_MOD_FORCE_EN	I2S_RX_FIFO_MOD	I2S_TX_FIFO_MOD	I2S_DSCR_EN	I2S_TX_DATA_NUM	I2S_RX_DATA_NUM	
31 21	20	19	18 16	15 13	12	11 6	5 0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0 0 0	0 0 0	1	32	32	Reset

I2S_RX_FIFO_MOD_FORCE_EN 此位永远置 1。（读/写）

I2S_TX_FIFO_MOD_FORCE_EN 此位永远置 1。（读/写）

I2S_RX_FIFO_MOD 接收 FIFO 模式配置位。（读/写）

I2S_TX_FIFO_MOD 发送 FIFO 模式配置位。（读/写）

I2S_DSCR_EN 置位使能 I2S DMA 模式。（读/写）

I2S_TX_DATA_NUM 发送 FIFO 数据长度的阈值。（读/写）

I2S_RX_DATA_NUM 接收 FIFO 数据长度的阈值。（读/写）

Register 12.10. I2S_RXEOF_NUM_REG (0x0024)

31 0	
64	Reset

I2S_RXEOF_NUM_REG 待接收数据的长度。会触发 I2S_IN_SUC_EOF_INT 中断。（读/写）

Register 12.11. I2S_CONF_SINGLE_DATA_REG (0x0028)

31 0	
0	Reset

I2S_CONF_SINGLE_DATA_REG 左声道或右声道根据 TX_CHAN_MOD 和 I2S_TX_MSB_RIGHT 输出保存在此寄存器的常量值。（读/写）

Register 12.12. I2S_CONF_CHAN_REG (0x002c)

(reserved)	I2S_RX_CHAN_MOD	I2S_TX_CHAN_MOD	
31 5	4 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0	0 0 0	Reset

I2S_RX_CHAN_MOD I2S 接收通道模式配置位，详情请参考章节 12.4.5。(读/写)

I2S_TX_CHAN_MOD I2S 发送通道模式配置位，详情请参考章节 12.4.4。(读/写)

Register 12.13. I2S_OUT_LINK_REG (0x0030)

(reserved)	I2S_OUTLINK_RESTART	I2S_OUTLINK_START	I2S_OUTLINK_STOP	(reserved)	I2S_OUTLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

I2S_OUTLINK_RESTART 置位重启发送链表描述符。(读/写)

I2S_OUTLINK_START 置位开始发送链表描述符。(读/写)

I2S_OUTLINK_STOP 置位停止发送链表描述符。(读/写)

I2S_OUTLINK_ADDR 第一个发送链表描述符的地址。(读/写)

Register 12.14. I2S_IN_LINK_REG (0x0034)

(reserved)	I2S_INLINK_RESTART	I2S_INLINK_START	I2S_INLINK_STOP	(reserved)	I2S_INLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

I2S_INLINK_RESTART 置位重启接收链表描述符。(读/写)

I2S_INLINK_START 置位开始接收链表描述符。(读/写)

I2S_INLINK_STOP 置位停止接收链表描述符。(读/写)

I2S_INLINK_ADDR 第一个接收链表描述符的地址。(读/写)

Register 12.15. I2S_OUT_EOF_DES_ADDR_REG (0x0038)

31 0	
0x000000000	Reset

I2S_OUT_EOF_DES_ADDR_REG 生成 EOF 的发送链表描述符的地址。(只读)

Register 12.16. I2S_IN_EOF_DES_ADDR_REG (0x003c)

31 0	
0x000000000	Reset

I2S_IN_EOF_DES_ADDR_REG 生成 EOF 的接收链表描述符地址。(只读)

Register 12.17. I2S_OUT_EOF_BFR_DES_ADDR_REG (0x0040)

31 0	
0x000000000	Reset

I2S_OUT_EOF_BFR_DES_ADDR_REG 生成 EOF 的发送链表描述符对应的缓存的地址。(只读)

Register 12.18. I2S_INLINK_DSCR_REG (0x0048)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2S_INLINK_DSCR_REG 当前接收链表描述符的地址。(只读)

Register 12.19. I2S_INLINK_DSCR_BF0_REG (0x004c)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2S_INLINK_DSCR_BF0_REG 下一个接收链表描述符的地址。(只读)

Register 12.20. I2S_INLINK_DSCR_BF1_REG (0x0050)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

I2S_INLINK_DSCR_BF1_REG 下一个接收链表数据 buffer 的地址。(只读)

Register 12.21. I2S_OUTLINK_DSCR_REG (0x0054)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUTLINK_DSCR_REG 当前发送链表描述符的地址。(只读)

Register 12.22. I2S_OUTLINK_DSCR_BF0_REG (0x0058)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUTLINK_DSCR_BF0_REG 下一个发送链表描述符的地址。(只读)

Register 12.23. I2S_OUTLINK_DSCR_BF1_REG (0x005c)

31																															0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

I2S_OUTLINK_DSCR_BF1_REG 下一个发送链表数据 buffer 的地址。(只读)

Register 12.24. I2S_LC_CONF_REG (0x0060)

(reserved)	I2S_CHECK_OWNER	I2S_OUT_DATA_BURST_EN	I2S_INDSCR_BURST_EN	I2S_OUTDSCR_BURST_EN	I2S_OUT_EOF_MODE	(reserved)	I2S_OUT_AUTO_WRBACK	I2S_OUT_LOOP_TEST	I2S_IN_LOOP_TEST	I2S_AHBM_RST	I2S_AHBM_FIFO_RST	I2S_OUT_RST	I2S_IN_RST	
31 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	1	0	0	0	0	0	0	0	0	Reset

I2S_CHECK_OWNER 置位，硬件查看 owner bit。(读/写)

I2S_OUT_DATA_BURST_EN 发送数据模式配置位。(读/写)

1：发送数据使用 burst 模式；

0：发送数据使用字节模式。

I2S_INDSCR_BURST_EN DMA 接收链表描述符传输模式配置位。(读/写)

1：使用 burst 模式；

0：使用字节模式。

I2S_OUTDSCR_BURST_EN DMA 发送链表描述符传输模式配置位。(读/写)

1：使用 burst 模式；

0：使用字节读取模式。

I2S_OUT_EOF_MODE 产生 I2S_OUT_EOF_INT 的模式。(读/写)

1：DMA 从 FIFO 中弹出所有数据；

0：AHB 将所有数据推入 FIFO 中。

I2S_OUT_AUTO_WRBACK 将此位置 1，当发送 buffer 中的数据发送完毕时，自动回写发送链表。(读/写)

I2S_OUT_LOOP_TEST 置位循环测试发送链表。(读/写)

I2S_IN_LOOP_TEST 置位循环测试接收链表。(读/写)

I2S_AHBM_RST 置位重置 DMA 的 AHB 接口。(读/写)

I2S_AHBM_FIFO_RST 置位重置 DMA 的 AHB 接口 cmdFIFO。(读/写)

I2S_OUT_RST 置位重置发送 DMA FSM。(读/写)

I2S_IN_RST 置位重置接收 DMA FSM。(读/写)

Register 12.25. I2S_LC_STATE0_REG (0x006c)

31 0	
0x000000000	Reset

I2S_LC_STATE0_REG 接收 DMA 通道状态寄存器。(只读)

Register 12.26. I2S_LC_STATE1_REG (0x0070)

31 … 0	
0x000000000	Reset

I2S_LC_STATE1_REG 发送 DMA 通道状态寄存器。（只读）

Register 12.27. I2S_LC_HUNG_CONF_REG (0x0074)

(reserved)	I2S_LC_FIFO_TIMEOUT_ENA	I2S_LC_FIFO_TIMEOUT_SHIFT	I2S_LC_FIFO_TIMEOUT	
31 … 12	11	10 … 8	7 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0 0 0	0x010	Reset

I2S_LC_FIFO_TIMEOUT_ENA FIFO 超时的使能位。（读/写）

I2S_LC_FIFO_TIMEOUT_SHIFT 用于设置计数器的阈值。当计数器的值 >= $88000/2^{i2s\_lc\_fifo\_timeout\_shift}$ 时，计数器重置。（读/写）

I2S_LC_FIFO_TIMEOUT 当 FIFO hung 计数器等于此位的值时，发送数据超时中断或接收数据超时中断被触发。（读/写）

Register 12.28. I2S_CONF1_REG (0x00a0)

(reserved)	I2S_TX_STOP_EN	I2S_RX_PCM_BYPASS	I2S_RX_PCM_CONF	I2S_TX_PCM_BYPASS	I2S_TX_PCM_CONF	
31 … 9	8	7	6 … 4	3	2 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	0x0	1	0x1	Reset

I2S_TX_STOP_EN 将此位置 1，当发送 FIFO 为空时，发送模块停止输出 BCK 和 WS 信号。（读/写）

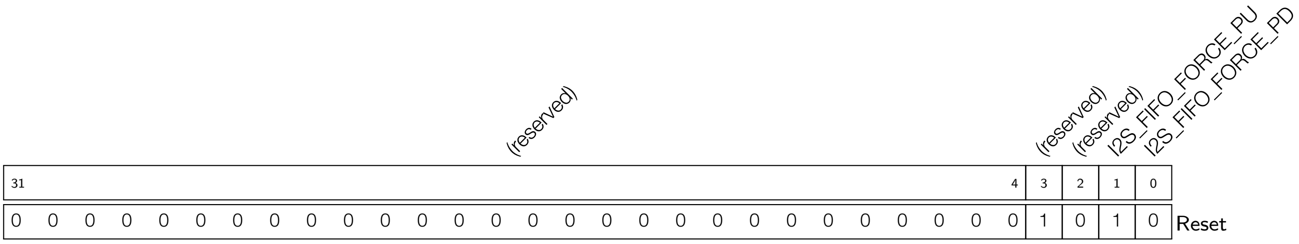
I2S_RX_PCM_BYPASS 置位使接收数据绕过压缩/解压缩模块。（读/写）

I2S_RX_PCM_CONF 压缩/解压缩模块配置位。（读/写）
0：解压缩接收数据；
1：压缩接收数据。

I2S_TX_PCM_BYPASS 置位使发送数据绕过压缩/解压缩模块。（读/写）

I2S_TX_PCM_CONF 压缩/解压缩模块配置位。（读/写）
0：解压缩发送数据；
1：压缩发送数据。

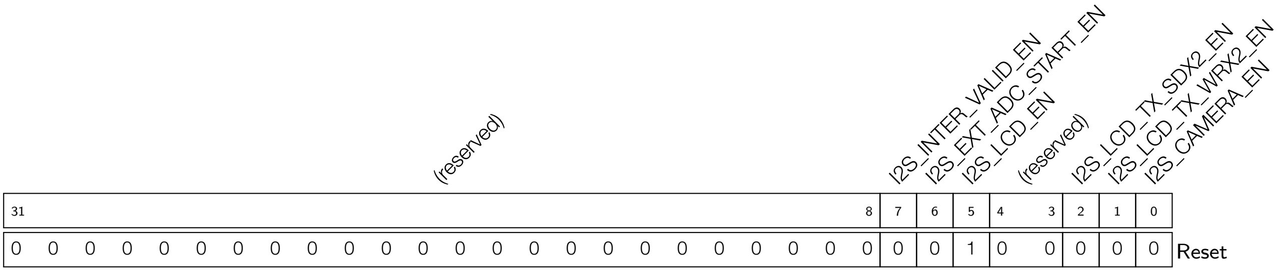
Register 12.29. I2S_PD_CONF_REG (0x00a4)



I2S_FIFO_FORCE_PU 强制开启 FIFO。(读/写)

I2S_FIFO_FORCE_PD 强制关闭 FIFO。(读/写)

Register 12.30. I2S_CONF2_REG (0x00a8)



I2S_INTER_VALID_EN 置位使能 camera 的内部认证。(读/写)

I2S_EXT_ADC_START_EN 置位使能外部 ADC。(读/写)

I2S_LCD_EN 置位使能 LCD 模式。(读/写)

I2S_LCD_TX_SDX2_EN 置位，在 LCD 模式下复制数据对（数据帧，Form 2)。(读/写)

I2S_LCD_TX_WRX2_EN LCD 模式下，一个数据写两次。(读/写)

I2S_CAMERA_EN 置位使能 camera 模式。(读/写)

Register 12.31. I2S_CLKM_CONF_REG (0x00ac)

(reserved)	I2S_CLKA_ENA	(reserved)	I2S_CLKM_DIV_A	I2S_CLKM_DIV_B	I2S_CLKM_DIV_NUM	
31 22	21	20	19 14	13 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0x00	0x00	4	Reset

I2S_CLKA_ENA 置位使能 clk_apll。(读/写)

I2S_CLKM_DIV_A 小数分频器的分母值。(读/写)

I2S_CLKM_DIV_B 小数分频器的分子值。(读/写)

I2S_CLKM_DIV_NUM I2S 时钟分频器的整数值。(读/写)

Register 12.32. I2S_SAMPLE_RATE_CONF_REG (0x00b0)

(reserved)	I2S_RX_BITS_MOD	I2S_TX_BITS_MOD	I2S_RX_BCK_DIV_NUM	I2S_TX_BCK_DIV_NUM	
31 24	23 18	17 12	11 6	5 0	
0 0 0 0 0 0 0 0	16	16	6	6	Reset

I2S_RX_BITS_MOD 置位配置接收通道的比特长度。(读/写)

I2S_TX_BITS_MOD 置位配置发送通道的比特长度。(读/写)

I2S_RX_BCK_DIV_NUM 接收模式下的比特时钟配置位。(读/写)

I2S_TX_BCK_DIV_NUM 发送模式下的比特时钟配置位。(读/写

Register 12.33. I2S_PDM_CONF_REG (0x00b4)

(reserved)	I2S_TX_PDM_HP_BYPASS	I2S_RX_PDM_SINC_DSR_16_EN	I2S_TX_PDM_SIGMADELTA_IN_SHIFT	I2S_TX_PDM_SINC_IN_SHIFT	I2S_TX_PDM_LP_IN_SHIFT	I2S_TX_PDM_HP_IN_SHIFT	(reserved)	I2S_TX_PDM_SINC_OSR2	I2S_PDM2PCM_CONV_EN	I2S_PCM2PDM_CONV_EN	I2S_RX_PDM_EN	I2S_TX_PDM_EN	
31 26	25	24	23 22	21 20	19 18	17 16	15 8	7 4	3	2	1	0	
0 0 0 0 0 0	0	1	0x1	0x1	0x1	0x1	0 0 0 0 0 0 0 0	0x02	1	1	0	0	Reset

I2S_TX_PDM_HP_BYPASS 置位绕过发送 PDM HP 过滤器。(读/写)

I2S_RX_PDM_SINC_DSR_16_EN Filter group1 的 PDM 下采样率。(读/写)

1：下采样率 = 128；

0：下采样率 = 64。

I2S_TX_PDM_SIGMADELTA_IN_SHIFT 调整输入到 Filter 模块的信号大小。(读/写)

0：除以 2；1: 乘以 1；2: 乘以 2；3: 乘以 4。

I2S_TX_PDM_SINC_IN_SHIFT 调整输入到 Filter 模块的信号大小。(读/写)

0：除以 2；1: 乘以 1；2: 乘以 2；3: 乘以 4。

I2S_TX_PDM_LP_IN_SHIFT 调整输入到 Filter 模块的信号大小。(读/写)

0：除以 2；1: 乘以 1；2: 乘以 2；3: 乘以 4。

I2S_TX_PDM_HP_IN_SHIFT 调整输入到 Filter 模块的信号大小。(读/写)

0：除以 2；1: 乘以 1；2: 乘以 2；3: 乘以 4。

I2S_TX_PDM_SINC_OSR2 上采样率 = 64×i2s_tx_pdm_sinc_osr2。(读/写)

I2S_PDM2PCM_CONV_EN 将此位置 1，使能 PDM-PCM 转换器。(读/写)

I2S_PCM2PDM_CONV_EN 将此位置 1，使能 PCM-PDM 转换器。(读/写)

I2S_RX_PDM_EN 将此位置 1，使能接收 PDM 模式。(读/写)

I2S_TX_PDM_EN 将此位置 1，使能发送 PDM 模式。(读/写)

Register 12.34. I2S_PDM_FREQ_CONF_REG (0x00b8)

(reserved)	I2S_TX_PDM_FP	I2S_TX_PDM_FS	
31 20	19 10	9 0	
0 0 0 0 0 0 0 0 0 0 0 0	960	441	Reset

I2S_TX_PDM_FP PCM-PDM 转换器的 PDM 频率参数。(读/写)

I2S_TX_PDM_FS PCM-PDM 转换器的 PCM 频率参数。(读/写)

Register 12.35. I2S_STATE_REG (0x00bc)

(reserved)	I2S_RX_FIFO_RESET_BACK	I2S_TX_FIFO_RESET_BACK	I2S_TX_IDLE	
31 3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0	0	0	1	Reset

I2S_RX_FIFO_RESET_BACK 此位用于确认接收 FIFO 复位是否完成。1：复位未完成；0：复位已完成。（只读）

I2S_TX_FIFO_RESET_BACK 此位用于确认发送 FIFO 复位是否完成。1：复位未完成；0：复位已完成。（只读）

I2S_TX_IDLE 发送设备的状态位。1：发送设备为空闲状态；0：发送设备为工作状态。（只读）

13 UART 控制器 (UART)

13.1 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。ESP32 芯片中有 3 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 RS-485 调制解调器。

3 个 UART 控制器有一组功能相同的寄存器。本文以 UART*n* 指代 3 个 UART 控制器，*n* 为 0、1、2。

13.2 主要特性

- 可编程收发波特率
- 3 个 UART 的发送 FIFO 以及接收 FIFO 共享 1024 × 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 个停止位
- 支持奇偶校验位
- 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

13.3 功能描述

13.3.1 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。ESP32 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。

13.3.2 UART 架构

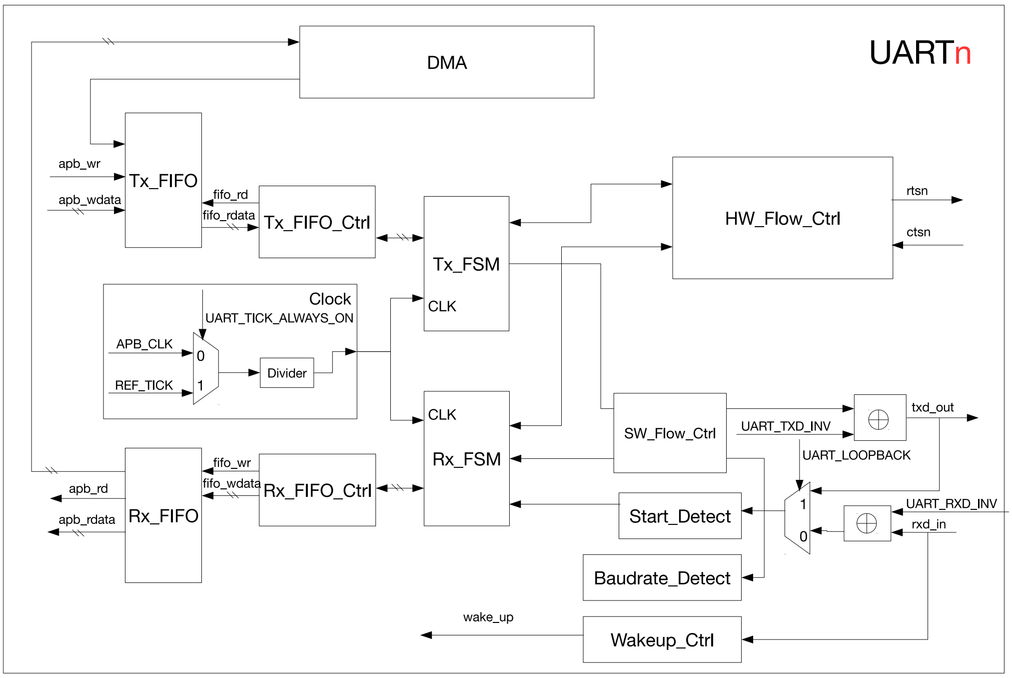


图 13-1. UART 基本架构图

图 13-1 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 （详情请参考章节复位和时钟）。可以通过配置 UART_TICK_REF_ALWAYS_ON 来选择时钟源。时钟中的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 总线写 Tx_FIFO，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。为了提高数据传输效率，可以使用 DMA 方式进行数据发送或接收。

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（详情请参考章节低功耗管理）状态时，Wakeup_Ctrl 开始计算 rxd_in 的脉冲个数，当输入 RxD 沿变化的次数大于等于 (UART_ACTIVE_THRESHOLD+2) 时产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒 UART 控制器。**注意：**只有 UART0 和 UART1 具有 Light-sleep 功能，且 rxd_in 不能通过 GPIO 交换矩阵输入，只能通过 IO_MUX 输入。

13.3.3 UART RAM

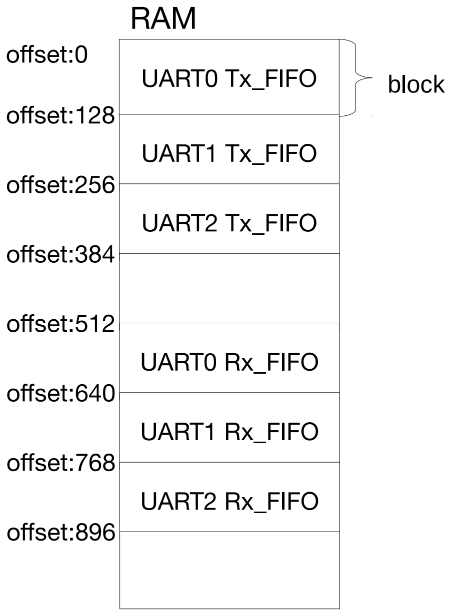


图 13-2. UART 共享 RAM 图

芯片中 3 个 UART 控制器共用 1024x8-bit RAM 空间。如图 13-2 所示，RAM 以 block 为单位进行分配，1 block 为 128×8 bit。图 13-2 所示为默认情况下 3 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，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。

当 3 个 UART 控制器都不工作时，可以通过置位 UART_MEM_PD、UART1_MEM_PD 以及 UART2_MEM_PD 来使 RAM 进入低功耗状态。

UART0 的 Tx_FIFO 和 Rx_FIFO 可以通过置位 UART_TXFIFO_RST 和 UART_RXFIFO_RST 来复位。UART1 的 Tx_FIFO 和 Rx_FIFO 可以通过置位 UART1_TXFIFO_RST 和 UART1_RXFIFO_RST 来复位。

> **说明：**
> UART2 的 Tx_FIFO 和 Rx_FIFO 没有相应的寄存器来复位。另外，UART1 的 UART1_TXFIFO_RST 和 UART1_RXFIFO_RST 可能会影响 UART2 的性能。因此，UART1 的 UART1_TXFIFO_RST 和 UART1_RXFIFO_RST 只能在 UART2 的 Tx_FIFO 和 Rx_FIFO 都没有数据的时候使用。

3 个 UART*n* 可通过寄存器 UART_FIFO_REG 访问对应的 FIFO。

13.3.4 波特率检测

置位 UART_AUTOBAUD_EN 可以开启 UART 波特率自检测功能。图 13-1 中的 Baudrate_Detect 可以滤除信号脉宽小于 UART_GLITCH_FILT 的噪声。

在 UART 双方进行通信之前可以通过发送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。UART_LOWPULSE_MIN_CNT 存储了最小低电平脉冲宽度，UART_HIGHPULSE_MIN_CNT 存储了最小高电平脉冲宽度，软件可以通过读取这两个寄存器获取发送方的波特率。

13.3.5 UART 数据帧

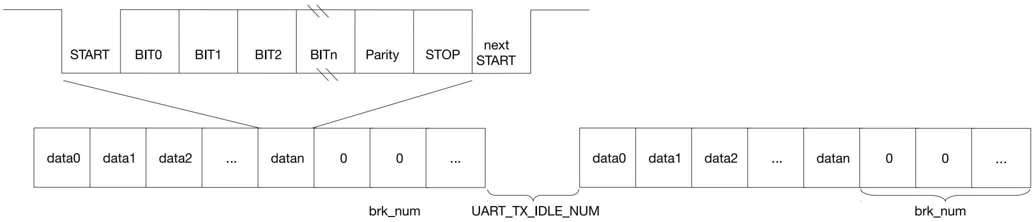


图 13-3. UART 数据帧结构

图 13-3 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 实现 1、1.5、2 位宽（RS485 模式下可通过配置 UART_DL0_EN 和 UART_DL1_EN 增加转换延时）。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，当接收器检测到数据帧格式错误时会产生 UART_FRM_ERR_INT 中断。

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，发送数据完成后，发送端会进入终止状态 (break condition)，发送几个连续的特殊数据帧 NULL，NULL 的数量可由 UART_TX_BRK_NUM 进行配置。发送器发送完所有的 NULL 之后会产生 UART_TX_BRK_DONE_INT 中断。数据帧之间可以通过配置 UART_TX_IDLE_NUM 保持最小间隔时间。当一帧数据之后的空闲时间大于等于 UART_TX_IDLE_NUM 寄存器的配置值时则产生 UART_TX_BRK_IDLE_DONE_INT 中断。

在传输一个 NULL 数据帧所需的时间内，RX 数据线若一直输出低电平，接收端会检测为终止状态，并触发 UART_BRK_DET_INT 中断表示终止状态已结束。

接收端通过 UART_RXFIFO_TOUT_INT 中断检测总线状态。接收端接收到至少一个字节数据后，总线处于空闲状态超过 UART_RX_TOUT_THRHD 位时间时，触发 UART_RXFIFO_TOUT_INT 中断。您可用此中断检测发送端是否已经发送所有数据。

13.3.6 AT_CMD 字符格式

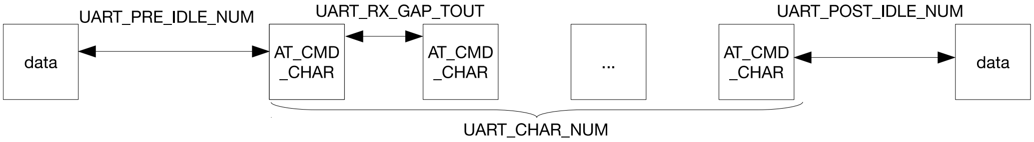


图 13-4. AT_CMD 字符格式

图 13-4 为一种特殊的 AT_CMD 字符格式。当接收器连续收到 UART_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 UART_AT_CMD_CHAR 与上一个非 UART_AT_CMD_CHAR 之间至少保持 UART_PER_IDLE_NUM 个 APB 时钟。
- UART_AT_CMD_CHAR 字符之间必须小于 UART_RX_GAP_TOUT 个 APB 时钟。

- 接收的 UART_AT_CMD_CHAR 字符个数必须大于等于 UART_CHAR_NUM。
- 接收到的最后一个 UART_AT_CMD_CHAR 字符与下一个非 UART_AT_CMD_CHAR 之间至少保持 UART_POST_IDLE_NUM 个 APB 时钟。

13.3.7 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实现数据流控功能。

13.3.7.1 硬件流控

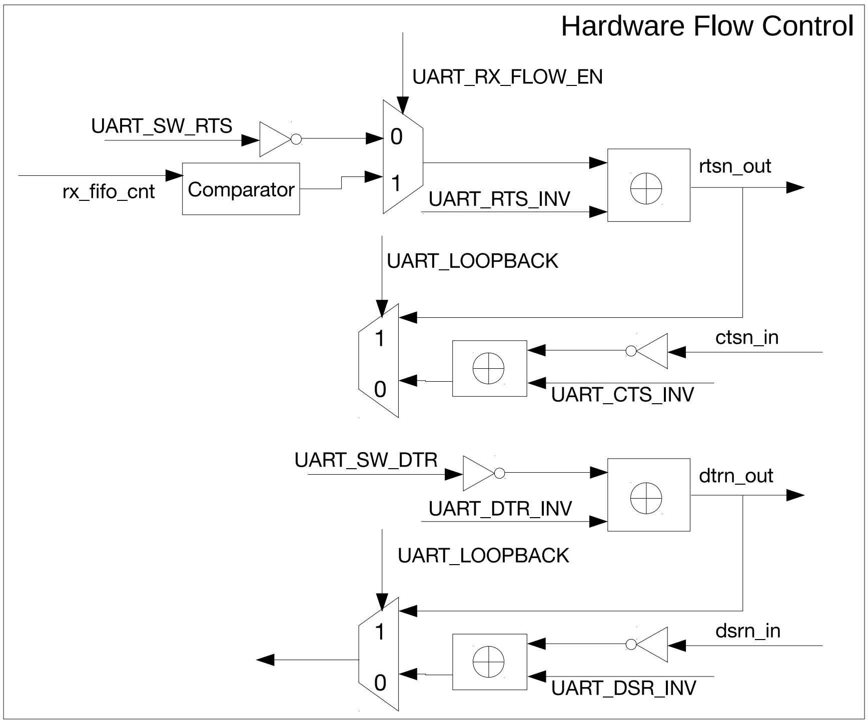


图 13-5. 硬件流控图

图 13-5 为 UART 硬件流控图。当使用硬件流控功能时，输出信号 rtsn_out 为高电平表示请求对方发送数据，rtsn_out 为低电平表示通知对方中止数据发送直到 rtsn_out 恢复高电平。发送器的硬件流控有两种方式。

- UART_RX_FLOW_EN 等于 0：可以通过配置 UART_SW_RTS 改变 rtsn_out 的电平。
- UART_RX_FLOW_EN 等于 1：当 Rx_FIFO 中的数据大于 UART_RX_FLOW_THRHD 时拉低 rtsn_out 的电平。

当 UART 检测到输入信号 ctsn_in 的沿变化时会产生 UART_CTS_CHG_INT 中断并且在发送完当前数据后停止接下来的数据发送。

输出信号 dtrn_out 为高电平表示发送方数据已经准备完毕，UART 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，从而判断当前是否可以接收数据。

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 rxd_in

相连，rtsn_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。

13.3.7.2 软件流控

软件可以通过置位 UART_FORCE_XOFF 来强制停止发送器发送数据，也可以通过置位 UART_FORCE_XON 来强制发送器发送数据。

UART 还可以通过传输特殊字符进行软件流控。置位 UART_SW_FLOW_CON_EN 可以开启软件流控功能。当 UART 接收的数据字节数超过 UART_XOFF 的阈值时，可以通过发送 UART_XOFF_CHAR 来告知对方停止发送数据。

在 UART_SW_FLOW_CON_EN 为 1 时，软件可以在任意时候发送流控字符。置位 UART_SEND_XOFF，发送器会在发送完当前数据之后插入发送一个 UART_XOFF_CHAR；置位 UART_SEND_XON，发送器会在发送完当前数据之后插入发送一个 UART_XON_CHAR。

13.3.8 UDMA

ESP32 芯片内部有两个 UDMA (UART DMA)，更多信息请见章节 DMA 控制器。

13.3.9 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 RS-485 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 RS-485 模式下检测到数据帧错误时触发此中断。
- UART_RS485_PARITY_ERR_INT: 在 RS-485 模式下检测到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: 当接收器检测到起始位时触发此中断。
- UART_SW_XOFF_INT: UART_SW_FLOW_CON_EN 置位时，当接收器接收到 Xon 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 Xoff 字符时触发此中断。
- UART_RXFIFO_TOUT_INT: 当接收器接收一个字节的时间大于 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一个 NULL （即传输一个 NULL 的时间内保持逻辑低电平）时触发此中断。
- UART_CTS_CHG_INT: 当接收器检测到 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 [tx_mem_cnt, txfifo_cnt] 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 [rx_flow_thrhd_h3, rx_flow_thrhd] 所指定的值时触发此中断。

13.3.10 UHCI 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: 当接收链表描述符里有错误时触发此中断。
- UHCI_IN_DSCR_ERR_INT: 当发送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。

13.4 寄存器列表

13.4.1 UART 寄存器列表

名称	描述	UART0	UART1	UART2	访问
配置寄存器					
UART_CONF0_REG	配置寄存器 0	0x3FF40020	0x3FF50020	0x3FF6E020	读/写
UART_CONF1_REG	配置寄存器 1	0x3FF40024	0x3FF50024	0x3FF6E024	读/写
UART_CLKDIV_REG	时钟分频配置	0x3FF40014	0x3FF50014	0x3FF6E014	读/写
UART_FLOW_CONF_REG	软件流控配置	0x3FF40034	0x3FF50034	0x3FF6E034	读/写
UART_SWFC_CONF_REG	软件流控字符配置	0x3FF4003C	0x3FF5003C	0x3FF6E03C	读/写
UART_SLEEP_CONF_REG	睡眠模式配置	0x3FF40038	0x3FF50038	0x3FF6E038	读/写
UART_IDLE_CONF_REG	帧结束空闲配置	0x3FF40040	0x3FF50040	0x3FF6E040	读/写
UART_RS485_CONF_REG	RS485 模式配置	0x3FF40044	0x3FF50044	0x3FF6E044	读/写
状态寄存器					

UART_STATUS_REG	UART 状态寄存器	0x3FF4001C	0x3FF5001C	0x3FF6E01C	只读
UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x3FF4005C	0x3FF5005C	0x3FF6E05C	RO
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x3FF40060	0x3FF50060	0x3FF6E060	RO
波特率自检寄存器					
UART_AUTOBAUD_REG	波特率自检寄存器配置寄存器	0x3FF40018	0x3FF50018	0x3FF6E018	读/写
UART_LOWPULSE_REG	波特率自检最低低电平脉冲持续时间寄存器	0x3FF40028	0x3FF50028	0x3FF6E028	只读
UART_HIGHPULSE_REG	波特率自检最短高电平脉冲持续时间寄存器	0x3FF4002C	0x3FF5002C	0x3FF6E02C	只读
UART_POSPULSE_REG	波特率自检高电平脉冲寄存器	0x3FF40068	0x3FF50068	0x3FF6E068	只读
UART_NEGPULSE_REG	波特率自检低电平脉冲寄存器	0x3FF4006C	0x3FF5006C	0x3FF6E06C	只读
UART_RXD_CNT_REG	波特率自检沿变化计数寄存器	0x3FF40030	0x3FF50030	0x3FF6E030	只读
AT 转义序列检测寄存器					
UART_AT_CMD_PRECNT_REG	字符序列发送前的时序配置	0x3FF40048	0x3FF50048	0x3FF6E048	读/写
UART_AT_CMD_POSTCNT_REG	字符序列发送后的时序配置	0x3FF4004C	0x3FF5004C	0x3FF6E04C	读/写
UART_AT_CMD_GAPTOUT_REG	超时配置寄存器	0x3FF40050	0x3FF50050	0x3FF6E050	读/写
UART_AT_CMD_CHAR_REG	AT 转义序列检测配置	0x3FF40054	0x3FF50054	0x3FF6E054	读/写
FIFO 配置寄存器					
UART_FIFO_REG	FIFO 数据寄存器	0x3FF40000	0x3FF50000	0x3FF6E000	读/写
UART_MEM_CONF_REG	UART 阈值和分配配置	0x3FF40058	0x3FF50058	0x3FF6E058	读/写
UART_MEM_CNT_STATUS_REG	接受和发送存储器配置	0x3FF40064	0x3FF50064	0x3FF6E064	只读
中断寄存器					
UART_INT_RAW_REG	原始中断状态	0x3FF40004	0x3FF50004	0x3FF6E004	只读
UART_INT_ST_REG	屏蔽中断状态	0x3FF40008	0x3FF50008	0x3FF6E008	只读
UART_INT_ENA_REG	中断使能位	0x3FF4000C	0x3FF5000C	0x3FF6E00C	读/写
UART_INT_CLR_REG	中断清除位	0x3FF40010	0x3FF50010	0x3FF6E010	只写

13.4.2 UHCI 寄存器列表

名称	描述	UDMA0	UDMA1	访问
配置寄存器				
UHCI_CONF0_REG	UART 和数据帧分离配置	0x3FF54000	0x3FF4C000	读/写
UHCI_CONF1_REG	UHCI 配置寄存器	0x3FF5402C	0x3FF4C02C	读/写
UHCI_ESCAPE_CONF_REG	转义字符配置	0x3FF54064	0x3FF4C064	读/写
UHCI_HUNG_CONF_REG	超时配置寄存器	0x3FF54068	0x3FF4C068	读/写
UHCI_ESC_CONF0_REG	转义序列配置寄存器 0	0x3FF540B0	0x3FF4C0B0	读/写
UHCI_ESC_CONF1_REG	转义序列配置寄存器 1	0x3FF540B4	0x3FF4C0B4	读/写

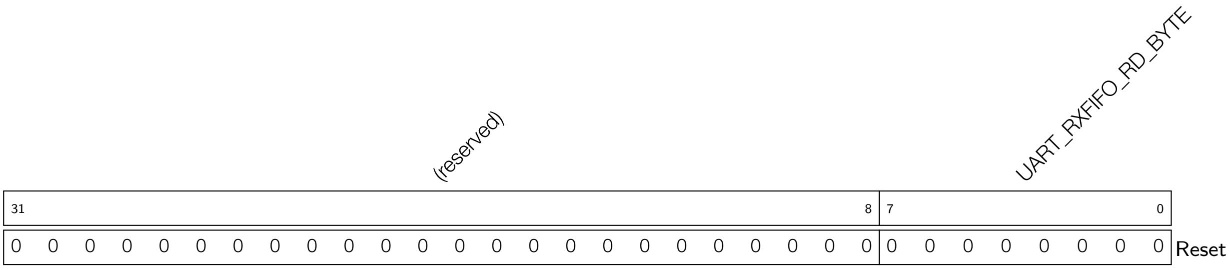
*UHCI_ESC_CONF2_REG	转义序列配置寄存器 2	0x3FF540B8	0x3FF4C0B8	读/写
UHCI_ESC_CONF3_REG	转义序列配置寄存器 3	0x3FF540BC	0x3FF4C0BC	读/写
DMA 配置寄存器				
UHCI_DMA_OUT_LINK_REG	链表描述符地址与控制	0x3FF54024	0x3FF4C024	读/写
UHCI_DMA_IN_LINK_REG	链表描述符地址与控制	0x3FF54028	0x3FF4C028	读/写
UHCI_DMA_OUT_PUSH_REG	FIFO 入栈数据寄存器	0x3FF54018	0x3FF4C018	读/写
UHCI_DMA_IN_POP_REG	FIFO 出栈数据寄存器	0x3FF54020	0x3FF4C020	只读
DMA 状态寄存器				
UHCI_DMA_OUT_STATUS_REG	DMA FIFO 状态	0x3FF54014	0x3FF4C014	只读
UHCI_DMA_OUT_EOF_DES_ADDR_REG	EOF 有效时发送链表描述符的地址	0x3FF54038	0x3FF4C038	只读
UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG	EOF 报错时发送链表描述符的地址	0x3FF54044	0x3FF4C044	只读
UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG	EOF 有效时接收链表描述符的地址	0x3FF5403C	0x3FF4C03C	只读
UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG	EOF 报错时接收链表描述符的地址	0x3FF54040	0x3FF4C040	只读
UHCI_DMA_IN_DSCR_REG	当前接收链表描述符的第一个字	0x3FF5404C	0x3FF4C04C	只读
UHCI_DMA_IN_DSCR_BF0_REG	当前接收链表描述符的第二个字	0x3FF54050	0x3FF4C050	只读
UHCI_DMA_IN_DSCR_BF1_REG	当前接收链表描述符的第三个字	0x3FF54054	0x3FF4C054	只读
UHCI_DMA_OUT_DSCR_REG	当前发送链表描述符的第一个字	0x3FF54058	0x3FF4C058	只读
UHCI_DMA_OUT_DSCR_BF0_REG	当前发送链表描述符的第二个字	0x3FF5405C	0x3FF4C05C	只读
UHCI_DMA_OUT_DSCR_BF1_REG	当前发送链表描述符的第三个字	0x3FF54060	0x3FF4C060	只读
中断寄存器				
UHCI_INT_RAW_REG	原始中断状态	0x3FF54004	0x3FF4C004	只读
UHCI_INT_ST_REG	屏蔽中断状态	0x3FF54008	0x3FF4C008	只读
UHCI_INT_ENA_REG	中断使能位	0x3FF5400C	0x3FF4C00C	读/写
UHCI_INT_CLR_REG	中断清除位	0x3FF54010	0x3FF4C010	只写

13.5 寄存器

13.5.1 UART 寄存器

本小节括号中的地址均为相对于 UART 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 13.4.1 *UART 寄存器列表*。

Register 13.1. UART_FIFO_REG (0x0)



UART_RXFIFO_RD_BYTE 访问 FIFO。（读/写）

Register 13.2. UART_INT_RAW_REG (0x4)

(reserved)	UART_AT_CMD_CHAR_DET_INT_RAW	UART_RS485_CLASH_INT_RAW	UART_RS485_FRM_ERR_INT_RAW	UART_RS485_PARITY_ERR_INT_RAW	UART_TX_DONE_INT_RAW	UART_TX_BRK_IDLE_DONE_INT_RAW	UART_TX_BRK_DONE_INT_RAW	UART_GLITCH_DET_INT_RAW	UART_SW_XOFF_INT_RAW	UART_SW_XON_INT_RAW	UART_RXFIFO_TOUT_INT_RAW	UART_BRK_DET_INT_RAW	UART_CTS_CHG_INT_RAW	UART_DSR_CHG_INT_RAW	UART_RXFIFO_OVF_INT_RAW	UART_FRM_ERR_INT_RAW	UART_PARITY_ERR_INT_RAW	UART_TXFIFO_EMPTY_INT_RAW	UART_RXFIFO_FULL_INT_RAW	
31 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_AT_CMD_CHAR_DET_INT_RAW UART_AT_CMD_CHAR_DET_INT 中断的原始中断状态位。（只读）

UART_RS485_CLASH_INT_RAW UART_RS485_CLASH_INT 中断的原始中断状态位。（只读）

UART_RS485_FRM_ERR_INT_RAW UART_RS485_FRM_ERR_INT 中断的原始中断状态位。（只读）

UART_RS485_PARITY_ERR_INT_RAW UART_RS485_PARITY_ERR_INT 中断的原始中断状态位。（只读）

UART_TX_DONE_INT_RAW UART_TX_DONE_INT 中断的原始中断状态位。（只读）

UART_TX_BRK_IDLE_DONE_INT_RAW UART_TX_BRK_IDLE_DONE_INT 中断的原始中断状态位。（只读）

UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 中断的原始中断状态位。（只读）

UART_GLITCH_DET_INT_RAW UART_GLITCH_DET_INT 中断的原始中断状态位。（只读）

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT 中断的原始中断状态位。（只读）

UART_SW_XON_INT_RAW UART_SW_XON_INT 中断的原始中断状态位。（只读）

UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 中断的原始中断状态位。（只读）

UART_BRK_DET_INT_RAW UART_BRK_DET_INT 中断的原始中断状态位。（只读）

UART_CTS_CHG_INT_RAW UART_CTS_CHG_INT 中断的原始中断状态位。（只读）

UART_DSR_CHG_INT_RAW UART_DSR_CHG_INT 中断的原始中断状态位。（只读）

UART_RXFIFO_OVF_INT_RAW UART_RXFIFO_OVF_INT 中断的原始中断状态位。（只读）

UART_FRM_ERR_INT_RAW UART_FRM_ERR_INT 中断的原始中断状态位。（只读）

UART_PARITY_ERR_INT_RAW UART_PARITY_ERR_INT 中断的原始中断状态位。（只读）

UART_TXFIFO_EMPTY_INT_RAW UART_TXFIFO_EMPTY_INT 中断的原始中断状态位。（只读）

UART_RXFIFO_FULL_INT_RAW UART_RXFIFO_FULL_INT 中断的原始中断状态位。（只读）

Register 13.3. UART_INT_ST_REG (0x8)

(reserved)	UART_AT_CMD_CHAR_DET_INT_ST	UART_RS485_CLASH_INT_ST	UART_RS485_FRM_ERR_INT_ST	UART_RS485_PARITY_ERR_INT_ST	UART_TX_DONE_INT_ST	UART_TX_BRK_IDLE_DONE_INT_ST	UART_TX_BRK_DONE_INT_ST	UART_GLITCH_DET_INT_ST	UART_SW_XOFF_INT_ST	UART_SW_XON_INT_ST	UART_RXFIFO_TOUT_INT_ST	UART_BRK_DET_INT_ST	UART_CTS_CHG_INT_ST	UART_DSR_CHG_INT_ST	UART_RXFIFO_OVF_INT_ST	UART_FRM_ERR_INT_ST	UART_PARITY_ERR_INT_ST	UART_TXFIFO_EMPTY_INT_ST	UART_RXFIFO_FULL_INT_ST	
31 … 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_AT_CMD_CHAR_DET_INT_ST UART_AT_CMD_CHAR_DET_INT 中断的隐蔽中断状态位。(只读)

UART_RS485_CLASH_INT_ST UART_RS485_CLASH_INT 中断的隐蔽中断状态位。(只读)

UART_RS485_FRM_ERR_INT_ST UART_RS485_FRM_ERR_INT 中断的隐蔽中断状态位。(只读)

UART_RS485_PARITY_ERR_INT_ST UART_RS485_PARITY_ERR_INT 中断的隐蔽中断状态位。(只读)

UART_TX_DONE_INT_ST UART_TX_DONE_INT 中断的隐蔽中断状态位。(只读)

UART_TX_BRK_IDLE_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT 中断的隐蔽中断状态位。(只读)

UART_TX_BRK_DONE_INT_ST UART_TX_BRK_DONE_INT 中断的隐蔽中断状态位。(只读)

UART_GLITCH_DET_INT_ST UART_GLITCH_DET_INT 中断的隐蔽中断状态位。(只读)

UART_SW_XOFF_INT_ST UART_SW_XOFF_INT 中断的隐蔽中断状态位。(只读)

UART_SW_XON_INT_ST UART_SW_XON_INT 中断的隐蔽中断状态位。(只读)

UART_RXFIFO_TOUT_INT_ST UART_RXFIFO_TOUT_INT 中断的隐蔽中断状态位。(只读)

UART_BRK_DET_INT_ST UART_BRK_DET_INT 中断的隐蔽中断状态位。(只读)

UART_CTS_CHG_INT_ST UART_CTS_CHG_INT 中断的隐蔽中断状态位。(只读)

UART_DSR_CHG_INT_ST UART_DSR_CHG_INT 中断的隐蔽中断状态位。(只读)

UART_RXFIFO_OVF_INT_ST UART_RXFIFO_OVF_INT 中断的隐蔽中断状态位。(只读)

UART_FRM_ERR_INT_ST UART_FRM_ERR_INT 中断的隐蔽中断状态位。(只读)

UART_PARITY_ERR_INT_ST UART_PARITY_ERR_INT 中断的隐蔽中断状态位。(只读)

UART_TXFIFO_EMPTY_INT_ST UART_TXFIFO_EMPTY_INT 中断的隐蔽中断状态位。(只读)

UART_RXFIFO_FULL_INT_ST UART_RXFIFO_FULL_INT 中断的隐蔽中断状态位。(只读)

Register 13.4. UART_INT_ENA_REG (0xC)

(reserved)	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_TX_BRK_DONE_INT_ENA	UART_GLITCH_DET_INT_ENA	UART_SW_XOFF_INT_ENA	UART_SW_XON_INT_ENA	UART_RXFIFO_TOUT_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 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_AT_CMD_CHAR_DET_INT_ENA UART_AT_CMD_CHAR_DET_INT 中断的使能位。(读/写)

UART_RS485_CLASH_INT_ENA UART_RS485_CLASH_INT 中断的使能位。(读/写)

UART_RS485_FRM_ERR_INT_ENA UART_RS485_FRM_ERR_INT 中断的使能位。(读/写)

UART_RS485_PARITY_ERR_INT_ENA UART_RS485_PARITY_ERR_INT 中断的使能位。(读/写)

UART_TX_DONE_INT_ENA T UART_TX_DONE_INT 中断的使能位。(读/写)

UART_TX_BRK_IDLE_DONE_INT_ENA UART_TX_BRK_IDLE_DONE_INT 中断的使能位。(读/写)

UART_TX_BRK_DONE_INT_ENA UART_TX_BRK_DONE_INT 中断的使能位。(读/写)

UART_GLITCH_DET_INT_ENA UART_GLITCH_DET_INT 中断的使能位。(读/写)

UART_SW_XOFF_INT_ENA UART_SW_XOFF_INT 中断的使能位。(读/写)

UART_SW_XON_INT_ENA UART_SW_XON_INT 中断的使能位。(读/写)

UART_RXFIFO_TOUT_INT_ENA UART_RXFIFO_TOUT_INT 中断的使能位。(读/写)

UART_BRK_DET_INT_ENA UART_BRK_DET_INT 中断的使能位。(读/写)

UART_CTS_CHG_INT_ENA UART_CTS_CHG_INT 中断的使能位。(读/写)

UART_DSR_CHG_INT_ENA UART_DSR_CHG_INT 中断的使能位。(读/写)

UART_RXFIFO_OVF_INT_ENA UART_RXFIFO_OVF_INT 中断的使能位。(读/写)

UART_FRM_ERR_INT_ENA UART_FRM_ERR_INT 中断的使能位。(读/写)

UART_PARITY_ERR_INT_ENA UART_PARITY_ERR_INT 中断的使能位。(读/写)

UART_TXFIFO_EMPTY_INT_ENA UART_TXFIFO_EMPTY_INT 中断的使能位。(读/写)

UART_RXFIFO_FULL_INT_ENA UART_RXFIFO_FULL_INT 中断的使能位。(读/写)

Register 13.5. UART_INT_CLR_REG (0x10)

(reserved)	UART_AT_CMD_CHAR_DET_INT_CLR	UART_RS485_CLASH_INT_CLR	UART_RS485_FRM_ERR_INT_CLR	UART_RS485_PARITY_ERR_INT_CLR	UART_TX_DONE_INT_CLR	UART_TX_BRK_IDLE_DONE_INT_CLR	UART_TX_BRK_DONE_INT_CLR	UART_GLITCH_DET_INT_CLR	UART_SW_XOFF_INT_CLR	UART_SW_XON_INT_CLR	UART_RXFIFO_TOUT_INT_CLR	UART_BRK_DET_INT_CLR	UART_CTS_CHG_INT_CLR	UART_DSR_CHG_INT_CLR	UART_RXFIFO_OVF_INT_CLR	UART_FRM_ERR_INT_CLR	UART_PARITY_ERR_INT_CLR	UART_TXFIFO_EMPTY_INT_CLR	UART_RXFIFO_FULL_INT_CLR	
31 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_AT_CMD_CHAR_DET_INT_CLR 置位清除 UART_AT_CMD_CHAR_DET_INT 中断。(只写)

UART_RS485_CLASH_INT_CLR 置位清除 UART_RS485_CLASH_INT 中断。(只写)

UART_RS485_FRM_ERR_INT_CLR 置位清除 UART_RS485_FRM_ERR_INT 中断。(只写)

UART_RS485_PARITY_ERR_INT_CLR 置位清除 UART_RS485_PARITY_ERR_INT 中断。(只写)

UART_TX_DONE_INT_CLR 置位清除 UART_TX_DONE_INT 中断。(只写)

UART_TX_BRK_IDLE_DONE_INT_CLR 置位清除 UART_TX_BRK_IDLE_DONE_INT 中断。(只写)

UART_TX_BRK_DONE_INT_CLR 置位清除 UART_TX_BRK_DONE_INT 中断。(只写)

UART_GLITCH_DET_INT_CLR 置位清除 UART_GLITCH_DET_INT 中断。(只写)

UART_SW_XOFF_INT_CLR 置位清除 UART_SW_XOFF_INT 中断。(只写)

UART_SW_XON_INT_CLR 置位清除 UART_SW_XON_INT 中断。(只写)

UART_RXFIFO_TOUT_INT_CLR 置位清除 UART_RXFIFO_TOUT_INT 中断。该寄存器只能在 rxfifo_cnt 和 rx_mem_cnt 都为 0 的情况下才可以置位。(只写)

UART_BRK_DET_INT_CLR 置位清除 UART_BRK_DET_INT 中断。(只写)

UART_CTS_CHG_INT_CLR 置位清除 UART_CTS_CHG_INT 中断。(只写)

UART_DSR_CHG_INT_CLR 置位清除 UART_DSR_CHG_INT 中断。(只写)

UART_RXFIFO_OVF_INT_CLR 置位清除 UART_RXFIFO_OVF_INT 中断。(只写)

UART_FRM_ERR_INT_CLR 置位清除 UART_FRM_ERR_INT 中断。(只写)

UART_PARITY_ERR_INT_CLR 置位清除 UART_PARITY_ERR_INT 中断。(只写)

UART_TXFIFO_EMPTY_INT_CLR 置位清除 UART_TXFIFO_EMPTY_INT 中断。(只写)

UART_RXFIFO_FULL_INT_CLR 置位清除 UART_RXFIFO_FULL_INT 中断。该寄存器只有在 Rx_FIFO 中数据个数小于 UART_RXFIFO_FULL_THRHD 的情况下置位才能生效。(只写)

Register 13.6. UART_CLKDIV_REG (0x14)

(reserved)	UART_CLKDIV_FRAG	UART_CLKDIV	
31 24	23 20	19 0	
0 0 0 0 0 0 0 0	0x00	0x0002B6	Reset

UART_CLKDIV_FRAG 分频系数的小数部分。（读/写）

UART_CLKDIV 分频系数的整数部分。（读/写）

Register 13.7. UART_AUTOBAUD_REG (0x18)

(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	0x010	0 0 0 0 0 0 0	0	Reset

UART_GLITCH_FILT 滤波门限值，当输入脉冲宽度小于此寄存器的值时，脉冲被忽略。此寄存器用于自动波特率检测的过程中。（读/写）

UART_AUTOBAUD_EN 自动波特率检测的使能位。（读/写）

Register 13.8. UART_STATUS_REG (0x1C)

UART_TXD	UART_RTSN	UART_DTRN	(reserved)	UART_ST_UTX_OUT	UART_TXFIFO_CNT	UART_RXD	UART_CTSN	UART_DSRN	(reserved)	UART_ST_URX_OUT	UART_RXFIFO_CNT	
31	30	29	28	27 24	23 16	15	14	13	12	11 8	7 0	
0x000	0	0	0	0 0 0 0	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_TXD 此位表明内部 UART RxD 信号的电平。(只读)

UART_RTSN 此位对应内部 UART CTS 信号的电平。(只读)

UART_DTRN 此位对应内部 UAR DSR 信号的电平。(只读)

UART_ST_UTX_OUT 此寄存器存储发送器有限状态机的状态。0: TX_IDLE;1: TX_STRT;2: TX_DAT0;3: TX_DAT1;4: TX_DAT2;5: TX_DAT3;6: TX_DAT4;7: TX_DAT5;8: TX_DAT6;9: TX_DAT7;10: TX_PRTY;11: TX_STP1;12: TX_STP2;13: TX_DL0;14: TX_DL1。(只读)

UART_TXFIFO_CNT (tx_mem_cnt, txfifo_cnt) 存储发送 FIFO 中的有效数据字节数。tx_mem_cnt 存储 3 个最高位;txfifo_cnt 存储 8 个最低位。(只读)

UART_RXD 此位对应内部 UART RxD 信号的电平。(只读)

UART_CTSN 此位对应内部 UART CTS 信号的电平。(只读)

UART_DSRN 此位对应内部 UAR DSR 信号的电平。(只读)

UART_ST_URX_OUT 此寄存器存储接收器有限状态机的状态。0: RX_IDLE;1: RX_STRT;2: RX_DAT0;3: RX_DAT1;4: RX_DAT2;5: RX_DAT3;6: RX_DAT4;7: RX_DAT5;8: RX_DAT6;9: RX_DAT7;10: RX_PRTY;11: RX_STP1;12: RX_STP2;13: RX_DL1。(只读)

UART_RXFIFO_CNT (rx_mem_cnt, rxfifo_cnt) 存储接收 FIFO 中的有效数据字节数。rx_mem_cnt 存储 3 个最高位;rxfifo_cnt 存储 8 个最低位。(只读)

Register 13.9. UART_CONF0_REG (0x20)

(reserved)	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_TX_FLOW_EN	UART_LOOPBACK	UART_IRDA_RX_INV	UART_IRDA_TX_INV	UART_IRDA_WCTL	UART_IRDA_TX_EN	UART_IRDA_DPLX	UART_TXD_BRK	UART_SW_DTR	UART_SW_RTS	UART_STOP_BIT_NUM	UART_BIT_NUM	UART_PARITY_EN	UART_PARITY	
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	
0 0 0 0	1	0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	0	0	Reset

UART_TICK_REF_ALWAYS_ON 此寄存器用于选择时钟。1：APB 时钟；0: REF_TICK。(读/写)

UART_DTR_INV 置位反转 UART DTR 信号的电平。(读/写)

UART_RTS_INV 置位反转 UART RTS 信号的电平。(读/写)

UART_TXD_INV 置位反转 UART TxD 信号的电平。(读/写)

UART_DSR_INV 置位反转 UART DSR 信号的电平。(读/写)

UART_CTS_INV 置位反转 UART CTS 信号的电平。(读/写)

UART_RXD_INV 置位反转 UART Rxd 信号的电平。(读/写)

UART_TXFIFO_RST 置位，复位 UART 发送 FIFO。注意，UART2 没有该复位寄存器，且 UART1 的 UART1_TXFIFO_RST 和 UART1_RXFIFO_RST 会影响 UART2 的工作。因此，只有在 UART2 的 Tx_FIFO 和 Rx_FIFO 中没有数据时，才可以置位这两个寄存器。(读/写)

UART_RXFIFO_RST 置位，复位 UART 接收 FIFO。注意，UART2 没有该复位寄存器，且 UART1 的 UART1_TXFIFO_RST 和 UART1_RXFIFO_RST 会影响 UART2 的工作。因此，只有在 UART2 的 Tx_FIFO 和 Rx_FIFO 中没有数据时，才可以置位这两个寄存器。(读/写)

UART_IRDA_EN 置位使能 IrDA 协议。(读/写)

UART_TX_FLOW_EN 置位使能发送器的流控功能。(读/写)

UART_LOOPBACK 置位使能 UART 回环测试功能。(读/写)

UART_IRDA_RX_INV 置位反转 IrDA 接收器的电平。(读/写)

UART_IRDA_TX_INV 置位反转 IrDA 发送器的电平。(读/写)

UART_IRDA_WCTL 1：IrDA 发送器的第 11 位与第 10 位相同。0: 设置 IrDA 发送器的第 11 位为 0。(读/写)

UART_IRDA_TX_EN IrDA 发送器的启动使能位。(读/写)

UART_IRDA_DPLX 置位使能 IrDA 回环模式。(读/写)

UART_TXD_BRK 置位使发送器在数据发送完成后发送 NULL。(读/写)

UART_SW_DTR 配置用于软件流控的软件 DTR 信号。(读/写)

UART_SW_RTS 用于 UART_RX_FLOW_EN 为 0 时的硬件流控。置位拉低 RTS (rtsn_out) 信号，复位拉高信号。(读/写)

寄存器描述下一页继续。

Register 13.9. UART_CONF0_REG (0x20)

继上一页寄存器描述。

UART_STOP_BIT_NUM 用于设置停止位的长度。

0: 无效值，没有作用

1: 1 位

2: 1.5 位

3: 2 位

（读/写）

UART_BIT_NUM 用于设置数据的长度；0: 5 bit；1: 6 bit；2: 7 bit；3: 8 bit。（读/写）

UART_PARITY_EN 置位使能 UART 奇偶校验。（读/写）

UART_PARITY 配置奇偶校验方式。0: 偶校验；1: 奇校验。（读/写）

Register 13.10. UART_CONF1_REG (0x24)

UART_RX_TOUT_EN	UART_RX_TOUT_THRHD	UART_RX_FLOW_EN	UART_RX_FLOW_THRHD	(reserved)	UART_TXFIFO_EMPTY_THRHD	(reserved)	UART_RXFIFO_FULL_THRHD	
31	30 24	23	22 16	15	14 8	7	6 0	
0	0 0 0 0 0 0 0	0	0x00	0	0x60	0	0x60	Reset

UART_RX_TOUT_EN 置位使能 UART 接收器的超时功能。（读/写）

UART_RX_TOUT_THRHD 配置 UART 接收器等待接收的超时时间。在使用 APB_CLK 时钟源时，该寄存器以波特率周期的 8 倍为计时单位；在使用 REF_TICK 时钟源时，该寄存器的计时单位为 UART 波特率周期 * 8 * (REF_TICK 频率)/(APB_CLK 频率)。（读/写）

UART_RX_FLOW_EN 置位使能 UART 接收器的流控功能。1: 配置 sw_rts 信号选择软件流控；0: 关闭软件流控。（读/写）

UART_RX_FLOW_THRHD 当 UART_RX_FLOW_EN 为 1、接收 FIFO 超过阈值时，接收器产生 rtsn_out 信号告诉发送器停止发送数据。阈值为 (rx_flow_thrhd_h3, rx_flow_thrhd)。（读/写）

UART_TXFIFO_EMPTY_THRHD 当发送 FIFO 的数据量少于阈值时，会产生 TXFIFO_EMPTY_INT_RAW 中断。阈值为 (tx_mem_empty_thrhd, txfifo_empty_thrhd)。（读/写）

UART_RXFIFO_FULL_THRHD 当接收器接收到比阈值多的数据时，接收器产生 RXFIFO_FULL_INT_RAW 中断。阈值为 (rx_flow_thrhd_h3, rxfifo_full_thrhd)。（读/写）

Register 13.11. UART_LOWPULSE_REG (0x28)

(reserved)	UART_LOWPULSE_MIN_CNT	
31 20	19 0	
0 0 0 0 0 0 0 0 0 0 0 0	0x0FFFFF	Reset

UART_LOWPULSE_MIN_CNT 此寄存器存储最小低电平脉冲宽度，用于波特率自检过程。（只读）

Register 13.12. UART_HIGHPULSE_REG (0x2C)

(reserved)	UART_HIGHPULSE_MIN_CNT	
31 20	19 0	
0 0 0 0 0 0 0 0 0 0 0 0	0x0FFFFF	Reset

UART_HIGHPULSE_MIN_CNT 此寄存器存储最小高电平脉冲宽度值，用于波特率自检过程。（只读）

Register 13.13. UART_RXD_CNT_REG (0x30)

(reserved)	UART_RXD_EDGE_CNT	
31 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	0x000	Reset

UART_RXD_EDGE_CNT 此寄存器存储 RxD 沿变化的次数，用于波特率自检过程。（只读）

Register 13.14. UART_FLOW_CONF_REG (0x34)

(reserved)	UART_SEND_XOFF	UART_SEND_XON	UART_FORCE_XOFF	UART_FORCE_XON	UART_XONOFF_DEL	UART_SW_FLOW_CON_EN	
31 6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

UART_SEND_XOFF 硬件自清 0，置位发送 Xoff 字符。（读/写）

UART_SEND_XON 硬件自清 0，置位发送 Xon 字符。（读/写）

UART_FORCE_XOFF 置位设置 CTSn 阻止发送器发送数据。（读/写）

UART_FORCE_XON 置位清除 CTSn 使能发送器继续发送数据。（读/写）

UART_XONOFF_DEL 置位移除接收到的数据中的流控字符。（读/写）

UART_SW_FLOW_CON_EN 置位使能软件流控，与 sw_xon 或 sw_xoff 寄存器一起使用。（读/写）

Register 13.15. UART_SLEEP_CONF_REG (0x38)

(reserved)	UART_ACTIVE_THRESHOLD	
31 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	0x0F0	Reset

UART_ACTIVE_THRESHOLD 当输入 RxD 沿变化的次数大于等于 (UART_ACTIVE_THRESHOLD+2) 时，系统从 Light-sleep 中醒来。（读/写）

Register 13.16. UART_SWFC_CONF_REG (0x3C)

UART_XOFF_CHAR	UART_XON_CHAR	UART_XOFF_THRESHOLD	UART_XON_THRESHOLD	
31 24	23 16	15 8	7 0	
0x013	0x011	0x0E0	0x000	Reset

UART_XOFF_CHAR 存储 Xoff 流控字符。（读/写）

UART_XON_CHAR 存储 Xon 流控字符。（读/写）

UART_XOFF_THRESHOLD 当接收 FIFO 中的数据量多于配置值时，会发送一个 Xoff 字符，需要 UART_SW_FLOW_CON_EN 置为 1。（读/写）

UART_XON_THRESHOLD 当接收 FIFO 中的数据量少于配置值时，会发送一个 Xon 字符，需要 UART_SW_FLOW_CON_EN 置为 1。（读/写）

Register 13.17. UART_IDLE_CONF_REG (0x40)

(reserved)	UART_TX_BRK_NUM	UART_TX_IDLE_NUM	UART_RX_IDLE_THRHD	
31 28	27 20	19 10	9 0	
0 0 0 0	0x00A	0x100	0x100	Reset

UART_TX_BRK_NUM 用于在发送数据结束后配置发送的 0 的数量，当 txd_brk 置为 1 时工作。（读/写）

UART_TX_IDLE_NUM 用于配置数据传输的间隔。（读/写）

UART_RX_IDLE_THRHD 当接收器等待的时间大于配置值时，会产生帧结束信号。（读/写）

Register 13.18. UART_RS485_CONF_REG (0x44)

(reserved)	UART_RS485_TX_DLY_NUM	UART_RS485_RX_DLY_NUM	UART_RS485RXBY_TX_EN	UART_RS485TX_RX_EN	UART_DL1_EN	UART_DL0_EN	UART_RS485_EN	
31 10	9 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_RS485_TX_DLY_NUM 用于延迟发送器内部数据信号。(读/写)

UART_RS485_RX_DLY_NUM 用于延迟接收器内部数据信号。(读/写)

UART_RS485RXBY_TX_EN 1: 当 RS-485 接收器线路繁忙时 RS-485 发送器能够发送数据。0: 当接收器繁忙时 RS-485 发送器不发送数据。(读/写)

UART_RS485TX_RX_EN 置位使能发送器输出信号回环到接收器输入信号。(读/写)

UART_DL1_EN 置位延迟停止位 1 bit。(读/写)

UART_DL0_EN 置位之后，在 DL1 之后延迟 1 bit 停止位。(读/写)

UART_RS485_EN 置位选择 RS-485 模式。(读/写)

Register 13.19. UART_AT_CMD_PRECNT_REG (0x48)

(reserved)	UART_PRE_IDLE_NUM	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0186A00	Reset

UART_PRE_IDLE_NUM 用于配置接收器接收到第一个 at_cmd 前的空闲时间。当空闲时间少于配置值时，接收器不会将接收到的下一个数据当作 at_cmd 字符。(读/写)

Register 13.20. UART_AT_CMD_POSTCNT_REG (0x4c)

(reserved)	UART_POST_IDLE_NUM	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0186A00	Reset

UART_POST_IDLE_NUM 用于配置最后一个 at_cmd 和下一个数据之间的间隔时间。当间隔时间少于配置值时，不会将前一个数据当做 at_cmd 字符。（读/写）

Register 13.21. UART_AT_CMD_GAPTOUT_REG (0x50)

(reserved)	UART_RX_GAP_TOUT	
31 24	23 0	
0 0 0 0 0 0 0 0	0x0001E00	Reset

UART_RX_GAP_TOUT 用于配置 at_cmd 字符之间的间隔时间。当间隔时间大于配置值时，不会将数据当做连续的 at_cmd 字符，且该寄存器的配置值至少要大于二分之一波特率。（读/写）

Register 13.22. UART_AT_CMD_CHAR_REG (0x54)

(reserved)	UART_CHAR_NUM	UART_AT_CMD_CHAR	
31 16	15 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x003	0x02B	Reset

UART_CHAR_NUM 用于配置接收器接收到的连续 at_cmd 字符的数量。（读/写）

UART_AT_CMD_CHAR 用于配置 at_cmd 字符的内容。（读/写）

Register 13.23. UART_MEM_CONF_REG (0x58)

(reserved)	UART_TX_MEM_EMPTY_THRHD	UART_RX_MEM_FULL_THRHD	UART_XOFF_THRESHOLD_H2	UART_XON_THRESHOLD_H2	UART_RX_TOUT_THRHD_H3	UART_RX_FLOW_THRHD_H3	(reserved)	UART_TX_SIZE	UART_RX_SIZE	(reserved)	UART_MEM_PD	
31	30 28	27 25	24 23	22 21	20 18	17 15	14 11	10 7	6 3	2 1	0	
0	0x0	0x0	0x0	0x0	0x0	0x0	0 0 0 0	0x01	0x01	0 0	0	Reset

UART_TX_MEM_EMPTY_THRHD 参考 TXFIFO_EMPTY_THRHD 的描述。（读/写）

UART_RX_MEM_FULL_THRHD 参考 RXFIFO_FULL_THRHD 的描述。（读/写）

UART_XOFF_THRESHOLD_H2 参考 UART_XOFF_THRESHOLD 的描述。（读/写）

UART_XON_THRESHOLD_H2 参考 UART_XON_THRESHOLD 的描述。（读/写）

UART_RX_TOUT_THRHD_H3 参考 RX_TOUT_THRHD 的描述。（读/写）

UART_RX_FLOW_THRHD_H3 参考 RX_FLOW_THRHD 的描述。（读/写）

UART_TX_SIZE 用于配置分配给发送 FIFO 的 RAM 空间，默认为 128 byte。（读/写）

UART_RX_SIZE 用于配置分配给接收 FIFO 的 RAM 空间，默认为 128 byte。（读/写）

UART_MEM_PD 置位关闭 RAM，当 3 个 UART 控制器的 reg_mem_pd 都置为 1 时，RAM 进入低功耗模式。（读/写）

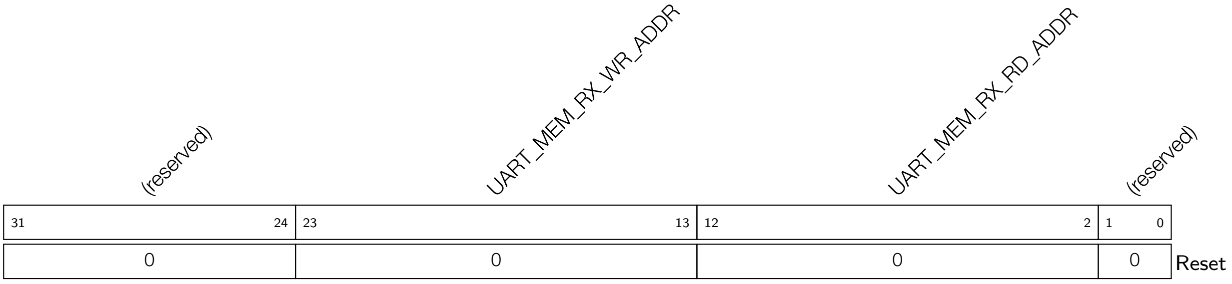
Register 13.24. UART_MEM_TX_STATUS_REG (0x5c)

(reserved)	UART_MEM_TX_WR_ADDR	UART_MEM_TX_RD_ADDR	(reserved)	
31 24	23 13	12 2	1 0	
0	0	0	0	Reset

UART_MEM_TX_WR_ADDR 表示写 TX FIFO 的偏移地址。(RO)

UART_MEM_TX_RD_ADDR 表示读 TX FIFO 的偏移地址。(RO)

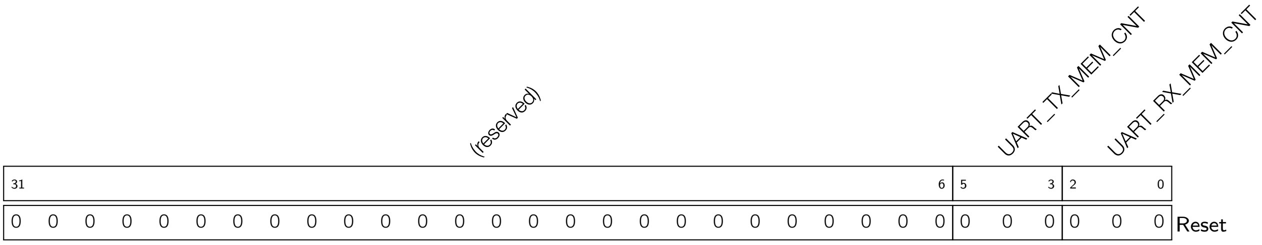
Register 13.25. UART_MEM_RX_STATUS_REG (0x0060)



UART_MEM_RX_RD_ADDR 表示读 RX FIFO 的偏移地址。(RO)

UART_MEM_RX_WR_ADDR 表示写 RX FIFO 的偏移地址。(RO)

Register 13.26. UART_MEM_CNT_STATUS_REG (0x64)



UART_TX_MEM_CNT 参考 TXFIFO_CNT 的描述。（只读）

UART_RX_MEM_CNT 参考 RXFIFO_CNT 的描述。（只读）

Register 13.27. UART_POSPULSE_REG (0x68)

UART_POSEDGE_MIN_CNT 存储 RxD 上升沿的沿变化次数，用于波特率自检的过程。（只读）

Register 13.28. UART_NEGPULSE_REG (0x6c)

(reserved)	UART_NEGEDGE_MIN_CNT	
31 … 20	19 … 0	
0 0 0 0 0 0 0 0 0 0 0 0	0x0FFFFF	Reset

UART_NEGEDGE_MIN_CNT 存储 RxD 下降沿的沿变化次数，用于波特率自检的过程。（只读）

13.5.2 UHCI 寄存器

本小节括号中的地址均为相对于 UDMA 基地址的地址偏移量（相对地址），具体基地址见章节 1 *系统和存储器*中的表 1-6 *外设地址映射*。寄存器绝对地址见章节 13.4.2 *UHCI 寄存器列表*。

Register 13.29. UHCI_CONF0_REG (0x0)

(reserved)	UHCI_ENCODE_CRC_EN	UHCI_LEN_EOF_EN	UHCI_UART_IDLE_EOF_EN	UHCI_CRC_REC_EN	UHCI_HEAD_EN	UHCI_SEPER_EN	(reserved)	UHCI_UART2_CE	UHCI_UART1_CE	UHCI_UART0_CE	(reserved)	
31 … 22	21	20	19	18	17	16	15 … 12	11	10	9	8 … 0	
0 0 0 0 0 0 0 0 0 0	1	1	0	1	1	1	0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0	Reset

UHCI_ENCODE_CRC_EN 保留。请初始化为 0。（读/写）

UHCI_LEN_EOF_EN 保留。请初始化为 0。（读/写）

UHCI_UART_IDLE_EOF_EN 保留。请初始化为 0。（读/写）

UHCI_CRC_REC_EN 保留。请初始化为 0。（读/写）

UHCI_HEAD_EN 保留。请初始化为 0。（读/写）

UHCI_SEPER_EN 置位使用特殊字符来分离数据帧。（读/写）

UHCI_UART2_CE 置位使用 UART2 发送或接收数据。（读/写）

UHCI_UART1_CE 置位使用 UART1 发送或接收数据。（读/写）

UHCI_UART0_CE 置位使用 UART 发送或接收数据。（读/写）

Register 13.30. UHCI_INT_RAW_REG (0x4)

(reserved)	UHCI_OUT_TOTAL_EOF_INT_RAW	UHCI_OUTLINK_EOF_ERR_INT_RAW	UHCI_IN_DSCR_EMPTY_INT_RAW	UHCI_OUT_DSCR_ERR_INT_RAW	UHCI_IN_DSCR_ERR_INT_RAW	UHCI_OUT_EOF_INT_RAW	UHCI_OUT_DONE_INT_RAW	UHCI_IN_ERR_EOF_INT_RAW	UHCI_IN_SUC_EOF_INT_RAW	UHCI_IN_DONE_INT_RAW	UHCI_TX_HUNG_INT_RAW	UHCI_RX_HUNG_INT_RAW	UHCI_TX_START_INT_RAW	UHCI_RX_START_INT_RAW	
31 14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UHCI_OUT_TOTAL_EOF_INT_RAW UHCI_OUT_TOTAL_EOF_INT 中断的原始中断状态位。（只读）

UHCI_OUTLINK_EOF_ERR_INT_RAW UHCI_OUTLINK_EOF_ERR_INT 中断的原始中断状态位。（只读）

UHCI_IN_DSCR_EMPTY_INT_RAW UHCI_IN_DSCR_EMPTY_INT 中断的原始中断状态位。（只读）

UHCI_OUT_DSCR_ERR_INT_RAW UHCI_OUT_DSCR_ERR_INT 中断的原始中断状态位。（只读）

UHCI_IN_DSCR_ERR_INT_RAW UHCI_IN_DSCR_ERR_INT 中断的原始中断状态位。（只读）

UHCI_OUT_EOF_INT_RAW UHCI_OUT_EOF_INT 中断的原始中断状态位。（只读）

UHCI_OUT_DONE_INT_RAW UHCI_OUT_DONE_INT 中断的原始中断状态位。（只读）

UHCI_IN_ERR_EOF_INT_RAW UHCI_IN_ERR_EOF_INT 中断的原始中断状态位。（只读）

UHCI_IN_SUC_EOF_INT_RAW UHCI_IN_SUC_EOF_INT 中断的原始中断状态位。（只读）

UHCI_IN_DONE_INT_RAW UHCI_IN_DONE_INT 中断的原始中断状态位。（只读）

UHCI_TX_HUNG_INT_RAW UHCI_TX_HUNG_INT 中断的原始中断状态位。（只读）

UHCI_RX_HUNG_INT_RAW UHCI_RX_HUNG_INT 中断的原始中断状态位。（只读）

UHCI_TX_START_INT_RAW UHCI_TX_START_INT 中断的原始中断状态位。（只读）

UHCI_RX_START_INT_RAW UHCI_RX_START_INT 中断的原始中断状态位。（只读）

Register 13.31. UHCI_INT_ST_REG (0x8)

(reserved)		UHCI_DMA_INFIFO_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_EMPTY_INT_ST	UHCI_OUT_DSCR_ERR_INT_ST	UHCI_IN_DSCR_ERR_INT_ST	UHCI_OUT_EOF_INT_ST	UHCI_OUT_DONE_INT_ST	UHCI_IN_ERR_EOF_INT_ST	UHCI_IN_SUC_EOF_INT_ST	UHCI_IN_DONE_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	0	0	0	0	0

Reset

UHCI_SEND_A_REG_Q_INT_ST UHCI_SEND_A_REG_Q_INT 中断的隐蔽中断状态位。(只读)

UHCI_SEND_S_REG_Q_INT_ST UHCI_SEND_S_REG_Q_INT 中断的隐蔽中断状态位。(只读)

UHCI_OUT_TOTAL_EOF_INT_ST UHCI_OUT_TOTAL_EOF_INT 中断的隐蔽中断状态位。(只读)

UHCI_OUTLINK_EOF_ERR_INT_ST UHCI_OUTLINK_EOF_ERR_INT 中断的隐蔽中断状态位。(只读)

UHCI_IN_DSCR_EMPTY_INT_ST UHCI_IN_DSCR_EMPTY_INT 中断的隐蔽中断状态位。(只读)

UHCI_OUT_DSCR_ERR_INT_ST UHCI_OUT_DSCR_ERR_INT 中断的隐蔽中断状态位。(只读)

UHCI_IN_DSCR_ERR_INT_ST UHCI_IN_DSCR_ERR_INT 中断的隐蔽中断状态位。(只读)

UHCI_OUT_EOF_INT_ST UHCI_OUT_EOF_INT 中断的隐蔽中断状态位。(只读)

UHCI_OUT_DONE_INT_ST UHCI_OUT_DONE_INT 中断的隐蔽中断状态位。(只读)

UHCI_IN_ERR_EOF_INT_ST UHCI_IN_ERR_EOF_INT 中断的隐蔽中断状态位。(只读)

UHCI_IN_SUC_EOF_INT_ST UHCI_IN_SUC_EOF_INT 中断的隐蔽中断状态位。(只读)

UHCI_IN_DONE_INT_ST UHCI_IN_DONE_INT 中断的隐蔽中断状态位。(只读)

UHCI_TX_HUNG_INT_ST UHCI_TX_HUNG_INT 中断的隐蔽中断状态位。(只读)

UHCI_RX_HUNG_INT_ST UHCI_RX_HUNG_INT 中断的隐蔽中断状态位。(只读)

UHCI_TX_START_INT_ST UHCI_TX_START_INT 中断的隐蔽中断状态位。(只读)

UHCI_RX_START_INT_ST UHCI_RX_START_INT 中断的隐蔽中断状态位。(只读)

Register 13.32. UHCI_INT_ENA_REG (0xC)

(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_IN_DSCR_ERR_INT_ENA	UHCI_OUT_EOF_INT_ENA	UHCI_OUT_DONE_INT_ENA	UHCI_IN_ERR_EOF_INT_ENA	UHCI_IN_SUC_EOF_INT_ENA	UHCI_IN_DONE_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	0	0	0	0	0	0	0	0	0	0	0	0	Reset

UHCI_SEND_A_REG_Q_INT_ENA UHCI_SEND_A_REG_Q_INT 中断的使能位。（读/写）

UHCI_SEND_S_REG_Q_INT_ENA UHCI_SEND_S_REG_Q_INT 中断的使能位。（读/写）

UHCI_OUT_TOTAL_EOF_INT_ENA UHCI_OUT_TOTAL_EOF_INT 中断的使能位。（读/写）

UHCI_OUTLINK_EOF_ERR_INT_ENA UHCI_OUTLINK_EOF_ERR_INT 中断的使能位。（读/写）

UHCI_IN_DSCR_EMPTY_INT_ENA UHCI_IN_DSCR_EMPTY_INT 中断的使能位。（读/写）

UHCI_OUT_DSCR_ERR_INT_ENA UHCI_OUT_DSCR_ERR_INT 中断的使能位。（读/写）

UHCI_IN_DSCR_ERR_INT_ENA UHCI_IN_DSCR_ERR_INT 中断的使能位。（读/写）

UHCI_OUT_EOF_INT_ENA UHCI_OUT_EOF_INT 中断的使能位。（读/写）

UHCI_OUT_DONE_INT_ENA UHCI_OUT_DONE_INT 中断的使能位。（读/写）

UHCI_IN_ERR_EOF_INT_ENA UHCI_IN_ERR_EOF_INT 中断的使能位。（读/写）

UHCI_IN_SUC_EOF_INT_ENA UHCI_IN_SUC_EOF_INT 中断的使能位。（读/写）

UHCI_IN_DONE_INT_ENA UHCI_IN_DONE_INT 中断的使能位。（读/写）

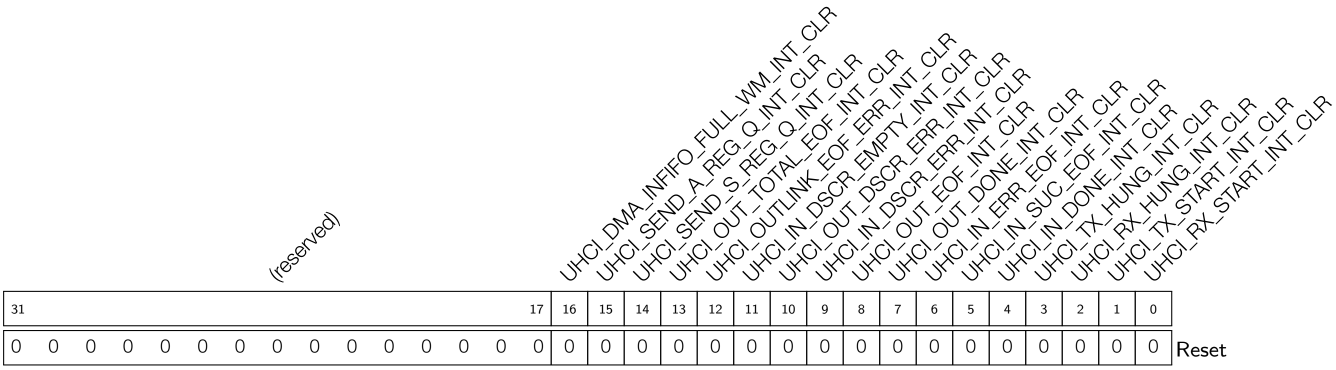
UHCI_TX_HUNG_INT_ENA UHCI_TX_HUNG_INT 中断的使能位。（读/写）

UHCI_RX_HUNG_INT_ENA UHCI_RX_HUNG_INT 中断的使能位。（读/写）

UHCI_TX_START_INT_ENA UHCI_TX_START_INT 中断的使能位。（读/写）

UHCI_RX_START_INT_ENA UHCI_RX_START_INT 中断的使能位。（读/写）

Register 13.33. UHCI_INT_CLR_REG (0x10)



UHCI_SEND_A_REG_Q_INT_CLR 置位清除 UHCI_SEND_A_REG_Q_INT 中断。(只写)

UHCI_SEND_S_REG_Q_INT_CLR 置位清除 UHCI_SEND_S_REG_Q_INT 中断。(只写)

UHCI_OUT_TOTAL_EOF_INT_CLR 置位清除 UHCI_OUT_TOTAL_EOF_INT 中断。(只写)

UHCI_OUTLINK_EOF_ERR_INT_CLR 置位清除 UHCI_OUTLINK_EOF_ERR_INT 中断。(只写)

UHCI_IN_DSCR_EMPTY_INT_CLR 置位清除 UHCI_IN_DSCR_EMPTY_INT 中断。(只写)

UHCI_OUT_DSCR_ERR_INT_CLR 置位清除 UHCI_OUT_DSCR_ERR_INT 中断。(只写)

UHCI_IN_DSCR_ERR_INT_CLR 置位清除 UHCI_IN_DSCR_ERR_INT 中断。(只写)

UHCI_OUT_EOF_INT_CLR 置位清除 UHCI_OUT_EOF_INT 中断。(只写)

UHCI_OUT_DONE_INT_CLR 置位清除 UHCI_OUT_DONE_INT 中断。(只写)

UHCI_IN_ERR_EOF_INT_CLR 置位清除 UHCI_IN_ERR_EOF_INT 中断。(只写)

UHCI_IN_SUC_EOF_INT_CLR 置位清除 UHCI_IN_SUC_EOF_INT 中断。(只写)

UHCI_IN_DONE_INT_CLR 置位清除 UHCI_IN_DONE_INT 中断。(只写)

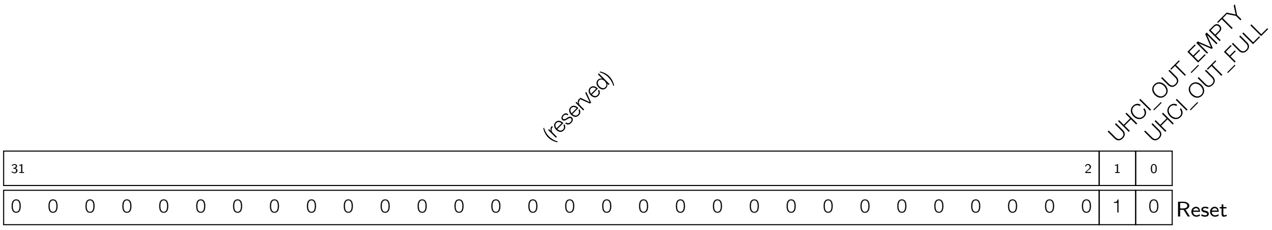
UHCI_TX_HUNG_INT_CLR 置位清除 UHCI_TX_HUNG_INT 中断。(只写)

UHCI_RX_HUNG_INT_CLR 置位清除 UHCI_RX_HUNG_INT 中断。(只写)

UHCI_TX_START_INT_CLR 置位清除 UHCI_TX_START_INT 中断。(只写)

UHCI_RX_START_INT_CLR 置位清除 UHCI_RX_START_INT 中断。(只写)

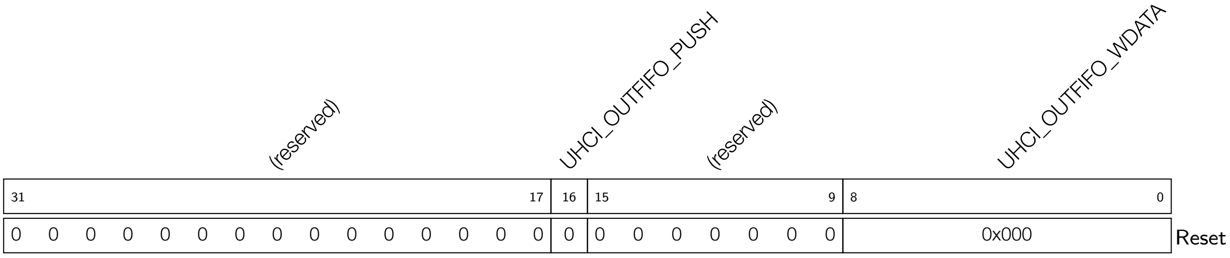
Register 13.34. UHCI_DMA_OUT_STATUS_REG (0x14)



UHCI_OUT_EMPTY 1: DMA 接收链表描述符的 FIFO 为空。(只读)

UHCI_OUT_FULL 1: DMA 发送链表描述符的 FIFO 是满的。(只读)

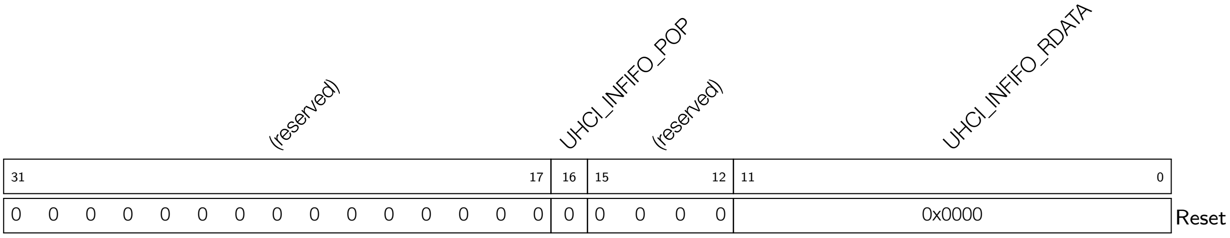
Register 13.35. UHCI_DMA_OUT_PUSH_REG (0x18)



UHCI_OUTFIFO_PUSH 置位将数据推入 DMA FIFO。(读/写)

UHCI_OUTFIFO_WDATA 需要被推入 DMA FIFO 的值。(读/写)

Register 13.36. UHCI_DMA_IN_POP_REG (0x20)



UHCI_INFIFO_POP 置位将数据从 DMA FIFO 中弹出。(读/写)

UHCI_INFIFO_RDATA 存储 DMA FIFO 中弹出的数据。(只读)

Register 13.37. UHCI_DMA_OUT_LINK_REG (0x24)

UHCI_OUTLINK_PARK	UHCI_OUTLINK_RESTART	UHCI_OUTLINK_START	UHCI_OUTLINK_STOP	(reserved)	UHCI_OUTLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

UHCI_OUTLINK_PARK 1：发送链表描述符的 FSM 处于空闲状态；0: 发送链表描述符的 FSM 处于工作状态。（只读）

UHCI_OUTLINK_RESTART 置位将发送链表描述符从上一次停止的地方重新启动。（读/写）

UHCI_OUTLINK_START 置位启动新的发送链表描述符。（读/写）

UHCI_OUTLINK_STOP 置位停止处理发送链表描述符。（读/写）

UHCI_OUTLINK_ADDR 存储第一个发送链表描述符的低 20 位地址。（读/写）

Register 13.38. UHCI_DMA_IN_LINK_REG (0x28)

UHCI_INLINK_PARK	UHCI_INLINK_RESTART	UHCI_INLINK_START	UHCI_INLINK_STOP	(reserved)	UHCI_INLINK_ADDR	
31	30	29	28	27 20	19 0	
0	0	0	0	0 0 0 0 0 0 0 0	0x000000	Reset

UHCI_INLINK_PARK 1: 接收链表描述符的 FSM 处于空闲状态；0: 接收链表描述符的 FSM 处于工作状态。（只读）

UHCI_INLINK_RESTART 置位重启新的接收链表描述符。（读/写）

UHCI_INLINK_START 置位开始处理接收链表描述符。（读/写）

UHCI_INLINK_STOP 置位停止处理接收链表描述符。（读/写）

UHCI_INLINK_ADDR 存储第一个接收链表描述符的低 20 位地址。（读/写）

Register 13.39. UHCI_CONF1_REG (0x2C)

(reserved)	UHCI_TX_ACK_NUM_RE	UHCI_TX_CHECK_SUM_RE	(reserved)	UHCI_CHECK_SEQ_EN	UHCI_CHECK_SUM_EN	
31 6	5	4	3 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	0 0	1	1	Reset

UHCI_TX_ACK_NUM_RE 保留。请初始化为 0。（读/写）

UHCI_TX_CHECK_SUM_RE 保留。请初始化为 0。（读/写）

UHCI_CHECK_SEQ_EN 保留。请初始化为 0。（读/写）

UHCI_CHECK_SUM_EN 保留。请初始化为 0。（读/写）

Register 13.40. UHCI_DMA_OUT_EOF_DES_ADDR_REG (0x38)

31 0	
0x000000000	Reset

UHCI_DMA_OUT_EOF_DES_ADDR_REG 存储当发送链表描述符的 EOF 位为 1 时的地址。（只读）

Register 13.41. UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG (0x3C)

31 0	
0x000000000	Reset

UHCI_DMA_IN_SUC_EOF_DES_ADDR_REG 存储当接收链表描述符的 EOF 位为 1 时的地址。（只读）

Register 13.42. UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG (0x40)

31 0	
0x000000000	Reset

UHCI_DMA_IN_ERR_EOF_DES_ADDR_REG 存储当接收链表描述符中有错误时的地址。（只读）

Register 13.43. UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG (0x44)

31	0	
0x000000000		Reset

UHCI_DMA_OUT_EOF_BFR_DES_ADDR_REG 存储当发送链表描述符中有错误时的地址。(只读)

Register 13.44. UHCI_DMA_IN_DSCR_REG (0x4C)

31	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		Reset

UHCI_DMA_IN_DSCR_REG 当前接收链表描述符的地址。(只读)

Register 13.45. UHCI_DMA_IN_DSCR_BF0_REG (0x50)

31	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		Reset

UHCI_DMA_IN_DSCR_BF0_REG 当前接收链表描述符的前面第 1 个的地址。(只读)

Register 13.46. UHCI_DMA_IN_DSCR_BF1_REG (0x54)

31	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		Reset

UHCI_DMA_IN_DSCR_BF1_REG 当前接收链表描述符的前面第 2 个的地址。(只读)

Register 13.47. UHCI_DMA_OUT_DSCR_REG (0x58)

31	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		Reset

UHCI_DMA_OUT_DSCR_REG 当前发送链表描述符的地址。(只读)

Register 13.48. UHCI_DMA_OUT_DSCR_BF0_REG (0x5C)

31	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		Reset

UHCI_DMA_OUT_DSCR_BF0_REG 当前发送链表描述符的前面第 1 个的地址。(只读)

反馈文档意见

Register 13.49. UHCI_DMA_OUT_DSCR_BF1_REG (0x60)

31	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

UHCI_DMA_OUT_DSCR_BF1_REG 当前发送链表描述符的前面第 2 个的地址。(只读)

Register 13.50. UHCI_ESCAPE_CONF_REG (0x64)

(reserved)		UHCI_RX_13_ESC_EN	UHCI_RX_11_ESC_EN	UHCI_RX_DB_ESC_EN	UHCI_RX_C0_ESC_EN	UHCI_TX_13_ESC_EN	UHCI_TX_11_ESC_EN	UHCI_TX_DB_ESC_EN	UHCI_TX_C0_ESC_EN	
31	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	1	1	0	0	1	1	Reset

UHCI_RX_13_ESC_EN 置位使能 DMA 发送数据时流控字符 0x13 的替换。(读/写)

UHCI_RX_11_ESC_EN 置位使能 DMA 发送数据时流控字符 0x11 的替换。(读/写)

UHCI_RX_DB_ESC_EN 置位使能 DMA 发送数据时流控字符 0xdb 的替换。(读/写)

UHCI_RX_C0_ESC_EN 置位使能 DMA 发送数据时流控字符 0xc0 的替换。(读/写)

UHCI_TX_13_ESC_EN 置位使能 DMA 接收数据时流控字符 0x13 的解码。(读/写)

UHCI_TX_11_ESC_EN 置位使能 DMA 接收数据时流控字符 0x11 的解码。(读/写)

UHCI_TX_DB_ESC_EN 置位使能 DMA 接收数据时流控字符 0xdb 的解码。(读/写)

UHCI_TX_C0_ESC_EN 置位使能 DMA 接收数据时流控字符 0xc0 的解码。(读/写)

Register 13.51. UHCI_HUNG_CONF_REG (0x68)

(reserved)	UHCI_RXFIFO_TIMEOUT_ENA	UHCI_RXFIFO_TIMEOUT_SHIFT	UHCI_RXFIFO_TIMEOUT	UHCI_TXFIFO_TIMEOUT_ENA	UHCI_TXFIFO_TIMEOUT_SHIFT	UHCI_TXFIFO_TIMEOUT	
31 24	23	22 20	19 12	11	10 8	7 0	
0 0 0 0 0 0 0 0	1	0 0 0	0x010	1	0 0 0	0x010	Reset

UHCI_RXFIFO_TIMEOUT_ENA DMA 发送数据超时的使能位。（读/写）

UHCI_RXFIFO_TIMEOUT_SHIFT 当计数值 >=(17’d8000»reg_rxfifo_timeout_shift) 时计数器清零。（读/写）

UHCI_RXFIFO_TIMEOUT 存储超时值。当 DMA 从 RAM 中读取数据的时间超过配置值时，会产生 UHCI_RX_HUNG_INT 中断。（读/写）

UHCI_TXFIFO_TIMEOUT_ENA 发送 FIFO 接收数据超时的使能位。（读/写）

UHCI_TXFIFO_TIMEOUT_SHIFT 当计数值 >=(17’d8000»reg_txfifo_timeout_shift) 时计数器清零。（读/写）

UHCI_TXFIFO_TIMEOUT 存储超时值。当 DMA 从 RAM 中读取数据的时间超过配置值时，会产生 UHCI_TX_HUNG_INT 中断。（读/写）

Register 13.52. UHCI_ESC_CONFn_REG (n: 0-3) (0xB0+4*n)

(reserved)	UHCI_ESC_SEQ2_CHAR1	UHCI_ESC_SEQ2_CHAR0	UHCI_ESC_SEQ2	
31 24	23 16	15 8	7 0	
0 0 0 0 0 0 0 0	0x0DF	0x0DB	0x013	Reset

UHCI_ESC_SEQ2_CHAR1 存储用于替换数据中 reg_esc_seq2 的第 2 个字符。（读/写）

UHCI_ESC_SEQ2_CHAR0 存储用于替换数据中 reg_esc_seq2 的第 1 个字符。（读/写）

UHCI_ESC_SEQ2 存储 flow_control 字符用以关闭 flow_control。（读/写）

14 LED PWM 控制器 (LEDC)

14.1 概述

LED_PWM 主要用于控制 LED 的亮度和颜色，也可以产生 PWM 信号用于其他用途。LED_PWM 有 16 路通道，即 8 路高速通道和 8 路低速通道。这 16 路通道能够产生独立的数字波形来驱动 RGB LED 设备。高速或低速通道可以由四个高速定时器之一或四个低速定时器之一进行驱动。PWM 控制器还能够自动逐渐增加或减少占空比，在无须处理器干预的情况下实现亮度和颜色渐变。LED_PWM 还支持小数分频。

在本文中，hsch*n* 指高速通道，lsch*n* 指低速通道。高速定时器和低速定时器分别命名为 h_timer*x* 和 l_timer*x*。

14.2 功能描述

14.2.1 架构

图 14-1 为 LED_PWM 基本架构图。从图中可知，LED_PWM 内部有 8 个高速通道以及 8 个低速通道。高速通道有 4 个高速时钟模块，可以从中任选一个 h_timer*x*。低速通道有 4 个低速时钟模块，可以从中任选一个 l_timer*x*。

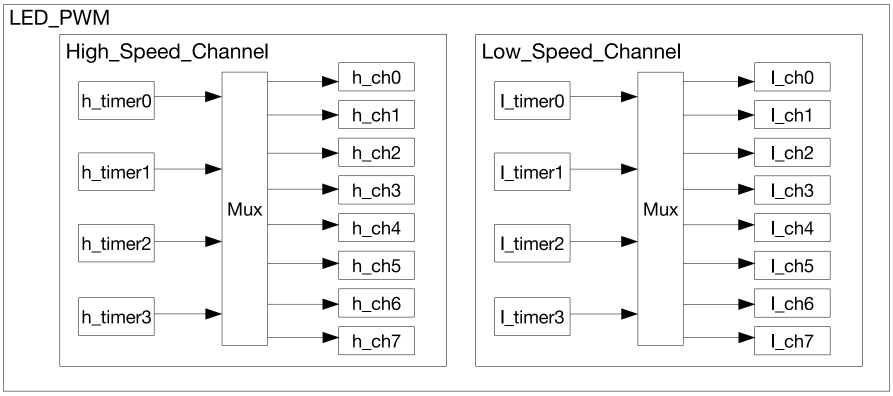


图 14-1. LED_PWM 架构

图 14-2 表示一个 PWM 通道和它选取的分频器；在该情况下，一个高速通道配有一个高速分频器。

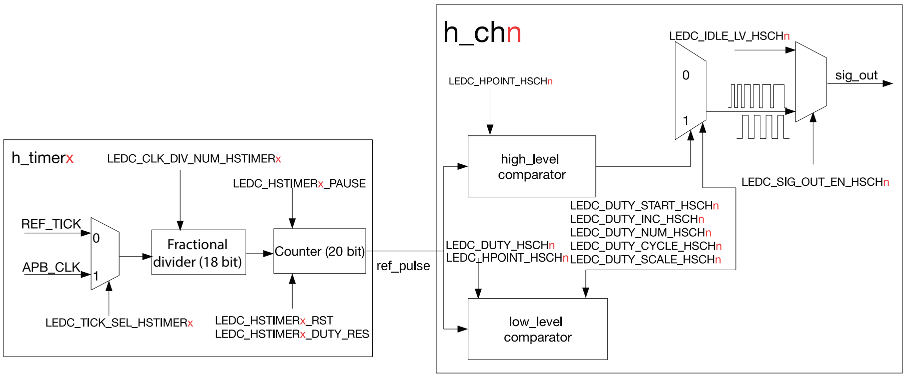


图 14-2. LED_PWM 高速通道框图

14.2.2 分频器

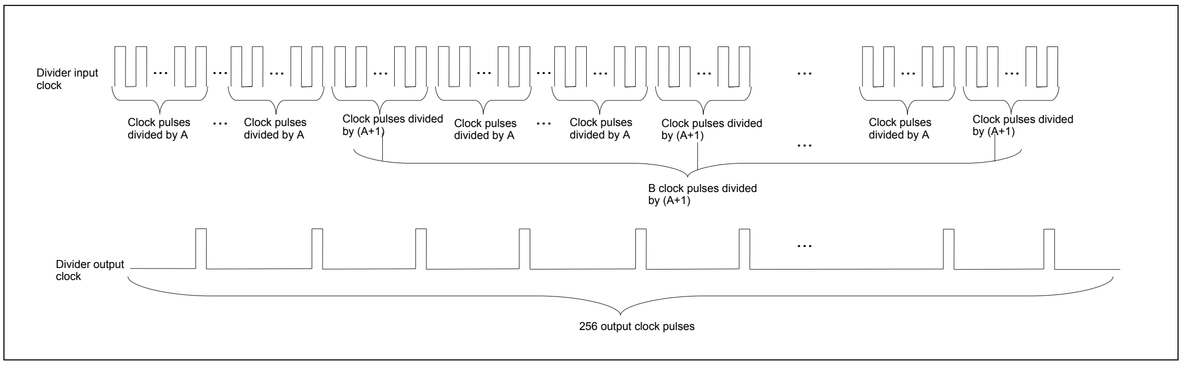


图 14-3. LED_PWM 分频器

LED PWM 高速定时器的时钟 LEDC_CLK*x* 有两个时钟源：REF_TICK 和 APB_CLK（关于时钟源的更多信息请参考章节复位和时钟）。输入时钟首先由分频器进行分频，分频系数为 LEDC_CLK_DIV_NUM_HSTIMER*x*，该系数的固定位宽是 18 位：其中高 10 位为整数部分 A，低 8 位为小数部分 B。分频系数 LEDC_CLK_DIV*x* 的公式为：

$$LEDC\_CLK\_DIVx = A + \frac{B}{256}$$

分频系数的范围为 1 ∼ 1023。

小数部分不为 0 时，分频器的输入输出时钟如图 14-3 所示。256 个输出周期中有 B 个周期以 (A+1) 分频，有 (256-B) 个周期以 A 分频。以 (A+1) 分频的 B 个周期均匀分布在 256 个周期中。

分频器的输出时钟作为计数器的基准时钟，计数器的计数范围由 LEDC_HSTIMER*x*_DUTY_RES 进行配置，每次计数达到最大值 $2^{LEDC\_HSTIMERx\_DUTY\_RES} - 1$ 时，产生溢出中断，并且计数值回归到 0。软件可以复位、暂停以及读取计数器的计数值。

定时器的输出信号由计数器产生，位宽为 20 位。信号的循环周期决定了任何连接到该定时器的 PWM 通道的信号频率。

PWM 生成器输出信号 sig_out*n* 的频率取决于定时器时钟源 LEDC_CLK*x* 的频率、时钟分频系数 LEDC_CLK_DIV*x* 以及占空比精度（计数器位宽）LEDC_HSTIMER*x*_DUTY_RES：

$$f_{\text{sig\_out}n} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV}x \cdot 2^{\text{LEDC\_HSTIMER}x\text{\_DUTY\_RES}}}$$

上述公式变形后，可得到以下公式计算预期的占空比精度：

$$\text{LEDC\_HSTIMER}x\text{\_DUTY\_RES} = \log_2\left(\frac{f_{\text{LEDC\_CLK}x}}{f_{\text{sig\_out}n} \cdot \text{LEDC\_CLK\_DIV}x}\right)$$

表 14-1 列出了常用配置频率及其对应精度。

表 14-1. 常用配置频率及精度

LEDC_CLKx	PWM 频率	最高精度（位）[1]	最低精度（位）[2]
APB_CLK (80 MHz)	1 kHz	16	7
APB_CLK (80 MHz)	5 kHz	13	4
APB_CLK (80 MHz)	10 kHz	12	3
RC_FAST_CLK (8 MHz)	1 kHz	12	3
RC_FAST_CLK (8 MHz)	2 kHz	11	2
REF_TICK (1 MHz)	1 kHz	9	1

[1] 最高精度指时钟分频系数 LEDC_CLK_DIVx 为 1 时的精度，向下取整。如果经公式计算出的最高精度超过了计数器位宽 20 位，则最高精度为 20。

[2] 最低精度指时钟分频系数 LEDC_CLK_DIVx 为 $1023 + \frac{255}{256}$ 时的精度，向上取整。如果经公式计算出的最低精度小于 0，则最低精度为 1。

低速通道的分频器 l_timerx 相对于高速通道的分频器 h_timerx 来说有以下 2 点区别：

1. 高速定位器的时钟源采用了 REF_TICK 或 APB_CLK，低速定位器采用了 REF_TICK 或 SLOW_CLOCK。置位 LEDC_APB_CLK_SEL 寄存器，SLOW_CLOCK 的频率为 80 MHz，否则为 8 MHz。
2. 当软件修改了高速通道计数器的最大值或分频系数的话，输出信号的更新将会在下一次溢出中断之后生效。而低速通道在置位 LEDC_LSTIMERx_PARA_UP 之后，立刻更新计数器的计数范围参数和分频器的分频系数。

14.2.3 通道

每个通道有两个比较器，即图 14-2 中的 high_level_comparator 以及 low_level_comparator。

high_level_comparator 的比较值是 hpoint，由 LEDC_HPOINT_HSCHn 配置。当计数器的值达到 hpoint 时，输出信号翻转为高电平。

low_level_comparator 的比较值是 lpoint，由 LEDC_DUTY_HSCHn，LEDC_DUTY_START_HSCHn，LEDC_DUTY_INC_HSCHn，LEDC_DUTY_NUM_HSCHn 以及 LEDC_DUTY_SCALE_HSCHn 共同决定。当计数器的值等于 lpoint 时，输出信号翻转为低电平。图 14-4 为 LED_PWM 输出信号图。

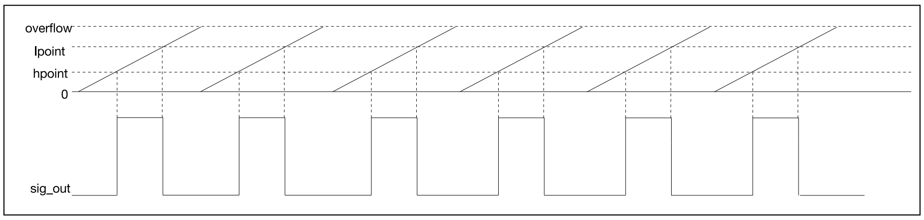


图 14-4. LED_PWM 输出信号图

LEDC_DUTY_HSCHn 是一个具有 4 位小数的浮点寄存器，其中高 20 位是整数部分，低 4 位是小数部分。当低 4 位非 0 时，输出信号的脉冲宽度有 LEDC_DUTY_HSCHn [3：0]/16 的概率多一个计数周期。低 4 位小数有利于提高输出信号占空比的精度。置位 LEDC_DUTY_START_HSCHn，通道更新后的 LEDC_DUTY_HSCHn 寄存器值才会起作用，即通道可以实现一种占空比到另一种占空比的转换。

LEDC_DUTY_INC_HSCHn 决定了占空比的变化方向。渐变占空比的定义如下：
每 LEDC_DUTY_CYCLE_HSCHn 个脉冲，LEDC_DUTY_HSCHn的高 20 位就会递增或递减

LEDC_DUTY_SCALE_HSCH*n*，渐变长度由 LEDC_DUTY_NUM_HSCH*n* 控制，当渐变完成后，会产生渐变完成中断，且之后的输出信号将延续最后一次脉冲。图 14-5 为 LEDC_DUTY_INC_HSCH*n* 为 1 时的渐变图，即每隔 LEDC_DUTY_CYCLE_HSCH*n* 个脉冲，输出信号脉宽递增 LEDC_DUTY_SCALE_HSCH*n*。

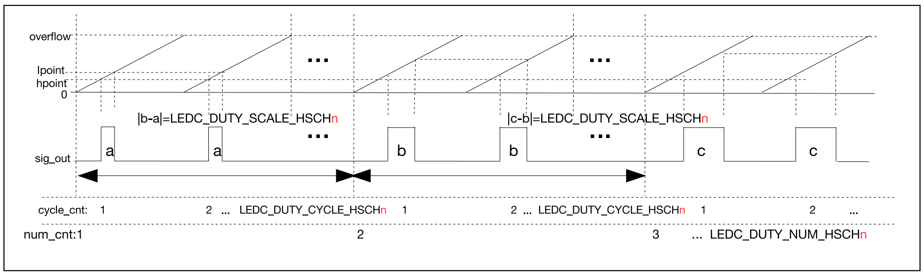


图 14-5. 渐变占空比输出信号图

注意

- 当 LEDC 处于渐变过程中，不支持操作（如暂停）渐变，或者配置以下寄存器：
  - LEDC_HPOINT_H/LSCH*n*
  - LEDC_DUTY_H/LSCH*n*
  - LEDC_DUTY_START_H/LSCH*n*
  - LEDC_DUTY_INC_H/LSCH*n*
  - LEDC_DUTY_NUM_H/LSCH*n*
  - LEDC_DUTY_CYCLE_H/LSCH*n*
  - LEDC_DUTY_SCALE_H/LSCH*n*

  必须在 LEDC_DUTY_CHNG_END_HSCH*n* 或者 LEDC_DUTY_CHNG_END_LSCH*n* 中断之后才能进行第二次渐变配置。

- 当配置 LEDC 为递减渐变且 LEDC_DUTY_HSCH*n* 为 $2^{LEDC\_HSTIMERx\_DUTY\_RES}$ 时，不允许配置 LEDC_DUTY_SCALE_HSCH*n* 为 1；当配置 LEDC 为递减渐变且 LEDC_DUTY_LSCH*n* 为 $2^{LEDC\_LSTIMERx\_DUTY\_RES}$ 时，不允许配置 LEDC_DUTY_SCALE_LSCH*n* 为 1。

14.2.4 中断

- LEDC_DUTY_CHNG_END_LSCH*n*_INT 低速通道上的占空比渐变结束触发中断。
- LEDC_DUTY_CHNG_END_HSCH*n*_INT 高速通道上的占空比渐变结束触发中断。
- LEDC_HS_TIMER*n*_OVF_INT 高速时钟计数器达到最大计数值触发中断。
- LEDC_LS_TIMER*n*_OVF_INT 低速时钟计数器达到最大计数值触发中断。

14.3 寄存器列表

名称	描述	地址	访问
配置寄存器			
LEDC_CONF_REG	LEDC 全局配置寄存器	0x3FF59190	读 / 写

名称	描述	地址	访问
LEDC_HSCH0_CONF0_REG	高速通道 0 的配置寄存器 0	0x3FF59000	读 / 写
LEDC_HSCH1_CONF0_REG	高速通道 1 的配置寄存器 0	0x3FF59014	读 / 写
LEDC_HSCH2_CONF0_REG	高速通道 2 的配置寄存器 0	0x3FF59028	读 / 写
LEDC_HSCH3_CONF0_REG	高速通道 3 的配置寄存器 0	0x3FF5903C	读 / 写
LEDC_HSCH4_CONF0_REG	高速通道 4 的配置寄存器 0	0x3FF59050	读 / 写
LEDC_HSCH5_CONF0_REG	高速通道 5 的配置寄存器 0	0x3FF59064	读 / 写
LEDC_HSCH6_CONF0_REG	高速通道 6 的配置寄存器 0	0x3FF59078	读 / 写
LEDC_HSCH7_CONF0_REG	高速通道 7 的配置寄存器 0	0x3FF5908C	读 / 写
LEDC_HSCH0_CONF1_REG	高速通道 0 的配置寄存器 1	0x3FF5900C	读 / 写
LEDC_HSCH1_CONF1_REG	高速通道 1 的配置寄存器 1	0x3FF59020	读 / 写
LEDC_HSCH2_CONF1_REG	高速通道 2 的配置寄存器 1	0x3FF59034	读 / 写
LEDC_HSCH3_CONF1_REG	高速通道 3 的配置寄存器 1	0x3FF59048	读 / 写
LEDC_HSCH4_CONF1_REG	高速通道 4 的配置寄存器 1	0x3FF5905C	读 / 写
LEDC_HSCH5_CONF1_REG	高速通道 5 的配置寄存器 1	0x3FF59070	读 / 写
LEDC_HSCH6_CONF1_REG	高速通道 6 的配置寄存器 1	0x3FF59084	读 / 写
LEDC_HSCH7_CONF1_REG	高速通道 7 的配置寄存器 1	0x3FF59098	读 / 写
LEDC_LSCH0_CONF0_REG	低速通道 0 的配置寄存器 0	0x3FF590A0	读 / 写
LEDC_LSCH1_CONF0_REG	低速通道 1 的配置寄存器 0	0x3FF590B4	读 / 写
LEDC_LSCH2_CONF0_REG	低速通道 2 的配置寄存器 0	0x3FF590C8	读 / 写
LEDC_LSCH3_CONF0_REG	低速通道 3 的配置寄存器 0	0x3FF590DC	读 / 写
LEDC_LSCH4_CONF0_REG	低速通道 4 的配置寄存器 0	0x3FF590F0	读 / 写
LEDC_LSCH5_CONF0_REG	低速通道 5 的配置寄存器 0	0x3FF59104	读 / 写
LEDC_LSCH6_CONF0_REG	低速通道 6 的配置寄存器 0	0x3FF59118	读 / 写
LEDC_LSCH7_CONF0_REG	低速通道 7 的配置寄存器 0	0x3FF5912C	读 / 写
LEDC_LSCH0_CONF1_REG	低速通道 0 的配置寄存器 1	0x3FF590AC	读 / 写
LEDC_LSCH1_CONF1_REG	低速通道 1 的配置寄存器 1	0x3FF590C0	读 / 写
LEDC_LSCH2_CONF1_REG	低速通道 2 的配置寄存器 1	0x3FF590D4	读 / 写
LEDC_LSCH3_CONF1_REG	低速通道 3 的配置寄存器 1	0x3FF590E8	读 / 写
LEDC_LSCH4_CONF1_REG	低速通道 4 的配置寄存器 1	0x3FF590FC	读 / 写
LEDC_LSCH5_CONF1_REG	低速通道 5 的配置寄存器 1	0x3FF59110	读 / 写
LEDC_LSCH6_CONF1_REG	低速通道 6 的配置寄存器 1	0x3FF59124	读 / 写
LEDC_LSCH7_CONF1_REG	低速通道 7 的配置寄存器 1	0x3FF59138	读 / 写
占空比寄存器			
LEDC_HSCH0_DUTY_REG	高速通道 0 的初始占空比	0x3FF59008	读 / 写
LEDC_HSCH1_DUTY_REG	高速通道 1 的初始占空比	0x3FF5901C	读 / 写
LEDC_HSCH2_DUTY_REG	高速通道 2 的初始占空比	0x3FF59030	读 / 写
LEDC_HSCH3_DUTY_REG	高速通道 3 的初始占空比	0x3FF59044	读 / 写
LEDC_HSCH4_DUTY_REG	高速通道 4 的初始占空比	0x3FF59058	读 / 写
LEDC_HSCH5_DUTY_REG	高速通道 5 的初始占空比	0x3FF5906C	读 / 写
LEDC_HSCH6_DUTY_REG	高速通道 6 的初始占空比	0x3FF59080	读 / 写
LEDC_HSCH7_DUTY_REG	高速通道 7 的初始占空比	0x3FF59094	读 / 写
LEDC_HSCH0_DUTY_R_REG	高速通道 0 的当前占空比	0x3FF59010	只读
LEDC_HSCH1_DUTY_R_REG	高速通道 1 的当前占空比	0x3FF59024	只读

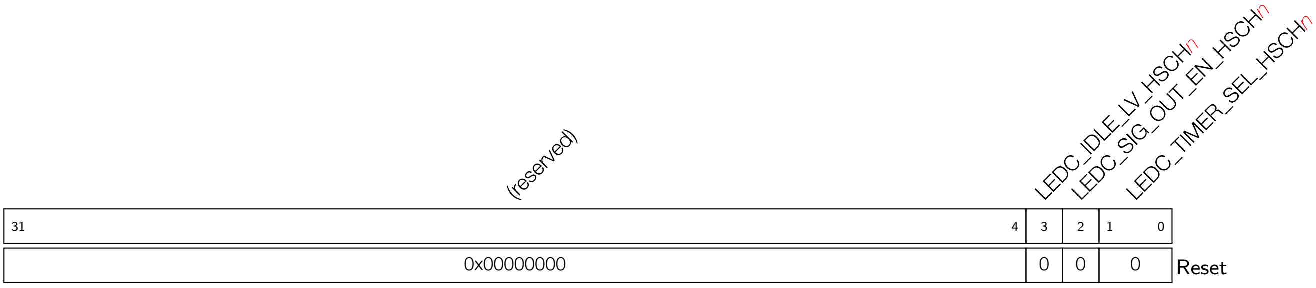
名称	描述	地址	访问
LEDC_HSCH2_DUTY_R_REG	高速通道 2 的当前占空比	0x3FF59038	只读
LEDC_HSCH3_DUTY_R_REG	高速通道 3 的当前占空比	0x3FF5904C	只读
LEDC_HSCH4_DUTY_R_REG	高速通道 4 的当前占空比	0x3FF59060	只读
LEDC_HSCH5_DUTY_R_REG	高速通道 5 的当前占空比	0x3FF59074	只读
LEDC_HSCH6_DUTY_R_REG	高速通道 6 的当前占空比	0x3FF59088	只读
LEDC_HSCH7_DUTY_R_REG	高速通道 7 的当前占空比	0x3FF5909C	只读
LEDC_LSCH0_DUTY_REG	低速通道 0 的初始占空比	0x3FF590A8	读 / 写
LEDC_LSCH1_DUTY_REG	低速通道 1 的初始占空比	0x3FF590BC	读 / 写
LEDC_LSCH2_DUTY_REG	低速通道 2 的初始占空比	0x3FF590D0	读 / 写
LEDC_LSCH3_DUTY_REG	低速通道 3 的初始占空比	0x3FF590E4	读 / 写
LEDC_LSCH4_DUTY_REG	低速通道 4 的初始占空比	0x3FF590F8	读 / 写
LEDC_LSCH5_DUTY_REG	低速通道 5 的初始占空比	0x3FF5910C	读 / 写
LEDC_LSCH6_DUTY_REG	低速通道 6 的初始占空比	0x3FF59120	读 / 写
LEDC_LSCH7_DUTY_REG	低速通道 7 的初始占空比	0x3FF59134	读 / 写
LEDC_LSCH0_DUTY_R_REG	低速通道 0 的当前占空比	0x3FF590B0	只读
LEDC_LSCH1_DUTY_R_REG	低速通道 1 的当前占空比	0x3FF590C4	只读
LEDC_LSCH2_DUTY_R_REG	低速通道 2 的当前占空比	0x3FF590D8	只读
LEDC_LSCH3_DUTY_R_REG	低速通道 3 的当前占空比	0x3FF590EC	只读
LEDC_LSCH4_DUTY_R_REG	低速通道 4 的当前占空比	0x3FF59100	只读
LEDC_LSCH5_DUTY_R_REG	低速通道 5 的当前占空比	0x3FF59114	只读
LEDC_LSCH6_DUTY_R_REG	低速通道 6 的当前占空比	0x3FF59128	只读
LEDC_LSCH7_DUTY_R_REG	低速通道 7 的当前占空比	0x3FF5913C	只读
分频器寄存器			
LEDC_HSTIMER0_CONF_REG	配置高速时钟计数器 0	0x3FF59140	读 / 写
LEDC_HSTIMER1_CONF_REG	配置高速时钟计数器 1	0x3FF59148	读 / 写
LEDC_HSTIMER2_CONF_REG	配置高速时钟计数器 2	0x3FF59150	读 / 写
LEDC_HSTIMER3_CONF_REG	配置高速时钟计数器 3	0x3FF59158	读 / 写
LEDC_HSTIMER0_VALUE_REG	高速时钟计数器 0 的当前计数值	0x3FF59144	只读
LEDC_HSTIMER1_VALUE_REG	高速时钟计数器 1 的当前计数值	0x3FF5914C	只读
LEDC_HSTIMER2_VALUE_REG	高速时钟计数器 2 的当前计数值	0x3FF59154	只读
LEDC_HSTIMER3_VALUE_REG	高速时钟计数器 3 的当前计数值	0x3FF5915C	只读
LEDC_LSTIMER0_CONF_REG	配置低速时钟计数器 0	0x3FF59160	读 / 写
LEDC_LSTIMER1_CONF_REG	配置低速时钟计数器 1	0x3FF59168	读 / 写
LEDC_LSTIMER2_CONF_REG	配置低速时钟计数器 2	0x3FF59170	读 / 写
LEDC_LSTIMER3_CONF_REG	配置低速时钟计数器 3	0x3FF59178	读 / 写
LEDC_LSTIMER0_VALUE_REG	低速时钟计数器 0 的当前计数值	0x3FF59164	只读
LEDC_LSTIMER1_VALUE_REG	低速时钟计数器 1 的当前计数值	0x3FF5916C	只读
LEDC_LSTIMER2_VALUE_REG	低速时钟计数器 2 的当前计数值	0x3FF59174	只读
LEDC_LSTIMER3_VALUE_REG	低速时钟计数器 3 的当前计数值	0x3FF5917C	只读
中断寄存器			
LEDC_INT_RAW_REG	原始中断状态	0x3FF59180	只读
LEDC_INT_ST_REG	屏蔽中断状态	0x3FF59184	只读
LEDC_INT_ENA_REG	中断使能位	0x3FF59188	读 / 写

名称	描述	地址	访问
LEDC_INT_CLR_REG	中断清除位	0x3FF5918C	只写

14.4 寄存器

本小节括号中的地址均为相对于 LED PWM 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 14.3 寄存器列表。

Register 14.1. LEDC_HSCH*n*_CONF0_REG (*n*: 0-7) (0x1C+0x10**n*)



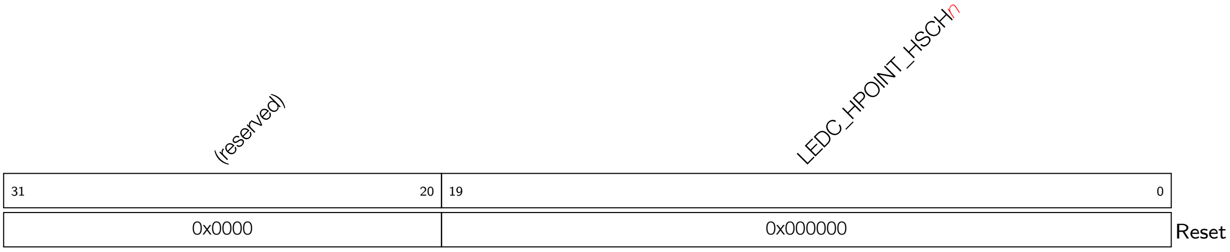
LEDC_IDLE_LV_HSCH*n* 高速通道 *n* 不工作时，用于控制输出值。（读 / 写）

LEDC_SIG_OUT_EN_HSCH*n* 高速通道 *n* 输出使能控制位。（读 / 写）

LEDC_TIMER_SEL_HSCH*n* 用于从 4 种高速时钟计数器中为高速通道 *n* 选择一个时钟计数器。（读 / 写）

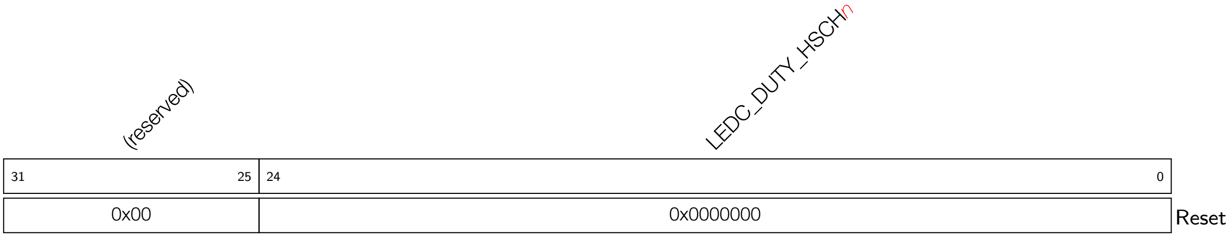
0: 选择 hstimer0；
1: 选择 hstimer1；
2: 选择 hstimer2；
3: 选择 hstimer3。

Register 14.2. LEDC_HSCH*n*_HPOINT_REG (*n*: 0-7) (0x20+0x10**n*)



LEDC_HPOINT_HSCH*n* 当高速通道 *n* 的 htimerx(x=[0,3]) 达到 LEDC_HPOINT_HSCH*n*[19:0]，输出信号翻转为高电平。（读 / 写）

Register 14.3. LEDC_HSCHn_DUTY_REG (n: 0-7) (0x24+0x10*n)



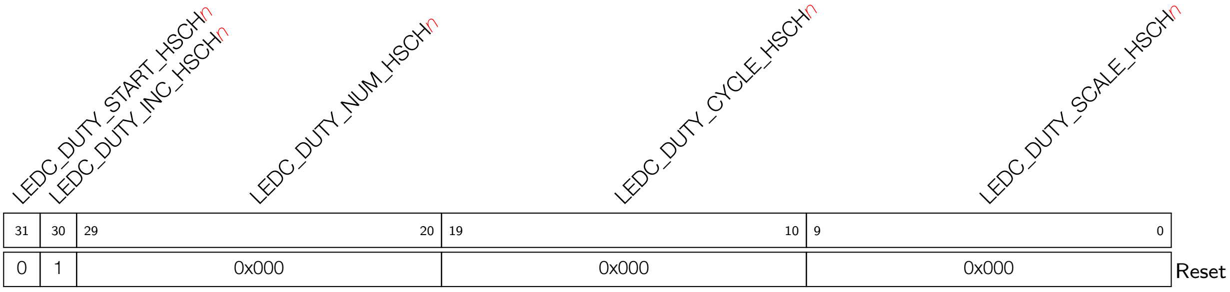
LEDC_DUTY_HSCHn 用于控制输出占空比。当通道 n 的 hstimerx(x=[0,3]) 达到 LEDC_LPOINT_HSCHn 时，输出信号翻转为低电平。

LEDC_LPOINT_HSCHn=LEDC_LPOINT_HSCHn[19:0]+LEDC_DUTY_HSCHn[24:4]) (1)

LEDC_LPOINT_HSCHn=LEDC_LPOINT_HSCHn[19:0]+LEDC_DUTY_HSCHn[24:4] +1) (2)

当寄存器选择 1，2 时，请从功能描述中获取更多信息。

Register 14.4. LEDC_HSCHn_CONF1_REG (n: 0-7) (0x28+0x10*n)



LEDC_DUTY_START_HSCHn 当配置了 LEDC_DUTY_NUM_HSCHn，LEDC_DUTY_CYCLE_HSCHn 以及 LEDC_DUTY_SCALE_HSCHn 时，需要置位 LEDC_DUTY_START_HSCHn，这些寄存器才会生效。硬件自动清零此位。（读 / 写）

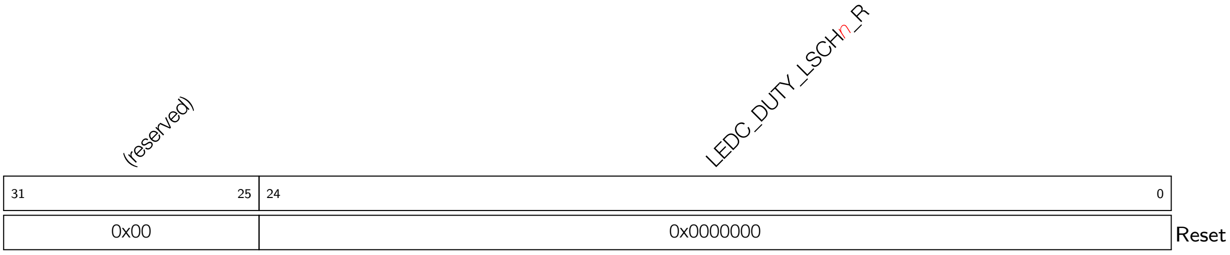
LEDC_DUTY_INC_HSCHn 用于递增或递减高速通道 n 的输出信号的占空比。（读 / 写）

LEDC_DUTY_NUM_HSCHn 用于控制高速通道 n 占空比变化的次数。（读 / 写）

LEDC_DUTY_CYCLE_HSCHn 每 LEDC_DUTY_CYCLE_HSCHn 个时钟周期递增或递减高速通道 n 的占空比。

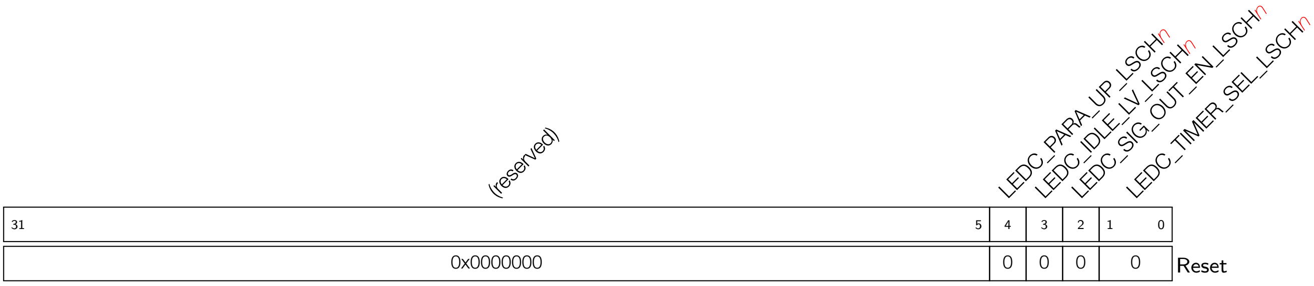
LEDC_DUTY_SCALE_HSCHn 用于递增或递减高速通道 n 的步长。（读 / 写）

Register 14.5. LEDC_HSCHn_DUTY_R_REG (n: 0-7) (0x2C+0x10*n)



LEDC_DUTY_HSCHn_R 当下高速通道 n 的输出信号的占空比。（只读）

Register 14.6. LEDC_LSCHn_CONF0_REG (n: 0-7) (0xBC+0x10*n)



LEDC_PARA_UP_LSCHn 用于更新低速通道 n 的 LEDC_LSCHn_HPOINT 和 LEDC_LSCHn_DUTY 寄存器。（读 / 写）

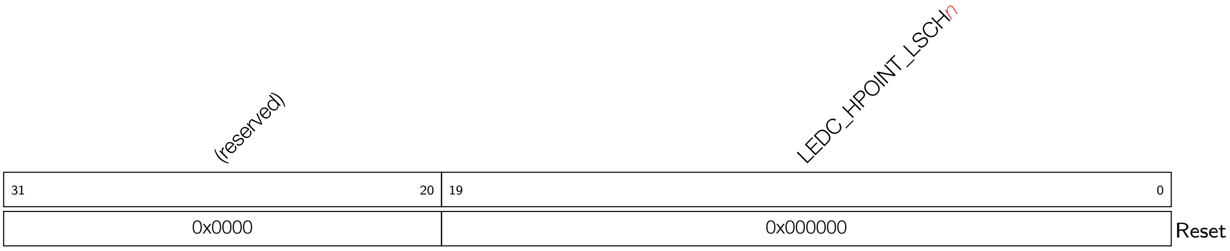
LEDC_IDLE_LV_LSCHn 当低速通道 n 不工作时，用于控制输出值。（读 / 写）

LEDC_SIG_OUT_EN_LSCHn 低速通道 n 的输出控制位。（读 / 写）

LEDC_TIMER_SEL_LSCHn 用于从 4 种低速分频器中为低速通道 n 选择一个时钟计数器。（读 / 写）

0: 选择 lstimer0；
1: 选择 lstimer1；
2: 选择 lstimer2；
3: 选择 lstimer3。

Register 14.7. LEDC_LSCHn_HPOINT_REG (n: 0-7) (0xC0+0x10*n)



LEDC_HPOINT_LSCHn 当低速通道 n 的 lstimerx(x=[0,3]) 达到 LEDC_HPOINT_LSCHn[19:0] 时，输出信号翻转为高电平。（读 / 写）

Register 14.8. LEDC_LSCHn_DUTY_REG (n: 0-7) (0xC4+0x10*n)

(reserved)	LEDC_DUTY_LSCHn	
31 25	24 0	
0x00	0x0000000	Reset

LEDC_DUTY_LSCHn 用于控制输出占空比。当低速通道 n 的 lstimerx(x=[0,3]) 达到 LEDC_LPOINT_LSCHn时，输出信号翻转为低电平。(读 / 写)
LEDC_LPOINT_LSCHn=(LEDC_HPOINT_LSCHn[19:0]+LEDC_DUTY_LSCHn[24:4]) (1)
LEDC_LPOINT_LSCHn=(LEDC_HPOINT_LSCHn[19:0]+LEDC_DUTY_LSCHn[24:4] +1) (2)
当寄存器选择 1，2 时，请从功能描述中获取更多信息。

Register 14.9. LEDC_LSCHn_CONF1_REG (n: 0-7) (0xC8+0x10*n)

LEDC_DUTY_START_LSCHn	LEDC_DUTY_INC_LSCHn	LEDC_DUTY_NUM_LSCHn	LEDC_DUTY_CYCLE_LSCHn	LEDC_DUTY_SCALE_LSCHn	
31	30	29 20	19 10	9 0	
0	1	0x000	0x000	0x000	Reset

LEDC_DUTY_START_LSCHn 当配置了 LEDC_DUTY_NUM_HSCHn、LEDC_DUTY_CYCLE_HSCHn 以及 LEDC_DUTY_SCALE_HSCHn 时，需要置位 LEDC_DUTY_START_HSCHn，这些寄存器才能生效。硬件自动清零此位。(读 / 写)

LEDC_DUTY_INC_LSCHn 用于递增或递减低速通道 n 的输出信号的占空比。(读 / 写)

LEDC_DUTY_NUM_LSCHn 用于控制低速通道 n 的占空比变化的次数。(读 / 写)

LEDC_DUTY_CYCLE_LSCHn 用于每 LEDC_DUTY_CYCLE_LSCHn 个时钟周期递增或递减占空比。(读 / 写)

LEDC_DUTY_SCALE_LSCHn 用递增或递减低速通道 n 的步长。(读 / 写)

Register 14.10. LEDC_LSCHn_DUTY_R_REG (n: 0-7) (0xCC+0x10*n)

(reserved)	LEDC_DUTY_LSCHn_R	
31 25	24 0	
0x00	0x0000000	Reset

LEDC_DUTY_R_LSCHn_R 低速通道 n 的输出信号的当前占空比。（只读）

Register 14.11. LEDC_HSTIMERx_CONF_REG (x: 0-3) (0x140+8*x)

(reserved)	LEDC_TICK_SEL_HSTIMERx	LEDC_HSTIMERx_RST	LEDC_HSTIMERx_PAUSE	LEDC_CLK_DIV_NUM_HSTIMERx	LEDC_HSTIMERx_DUTY_RES	
31 26	25	24	23	22 5	4 0	
0x00	0	1	0	0x00000	0x00	Reset

LEDC_TICK_SEL_HSTIMERx 用于选择 APB_CLK 或 REF_TICK 作为高速时钟计数器。（读 / 写）

1：APB_CLK；

0：REF_TICK。

LEDC_HSTIMERx_RST 用于复位高速时钟计数器 x，复位后计数器为 0。（读 / 写）

LEDC_HSTIMERx_PAUSE 用于暂停高速时钟计数器 x。（读 / 写）

LEDC_CLK_DIV_NUM_HSTIMERx 用于配置高速时钟计数器 x，低 8 位为小数部分。（读 / 写）

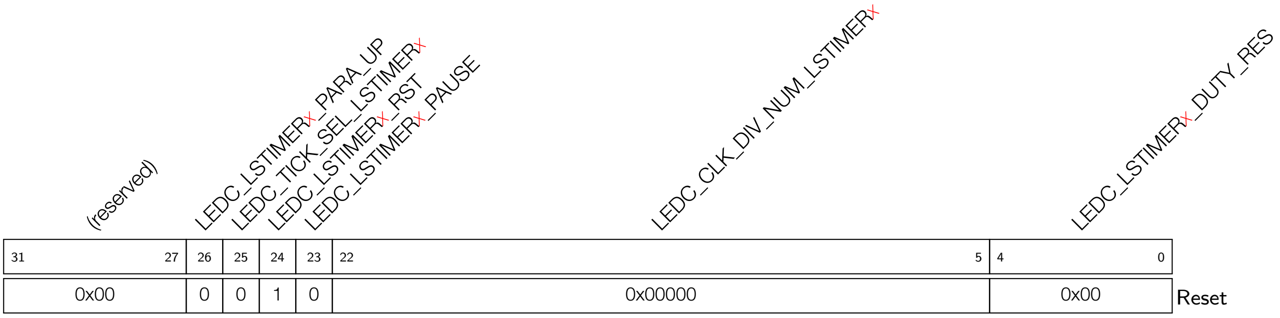
LEDC_HSTIMERx_DUTY_RES 用于控制高速时钟计数器 x 的计数范围。计数范围为 [0,2**LEDC_HSTIMERx_DUTY_RES]，最大位宽为 20 位。（读 / 写）

Register 14.12. LEDC_HSTIMERx_VALUE_REG (x: 0-3) (0x144+8*x)

(reserved)	LEDC_HSTIMERx_CNT	
31 20	19 0	
0x0000	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

LEDC_HSTIMERx_CNT 软件可以读取高速时钟计数器 x 的当前计数值。（只读）

Register 14.13. LEDC_LSTIMERx_CONF_REG (x: 0-3) (0x160+8*x)



LEDC_LSTIMERx_PARA_UP 用于更新 LEDC_CLK_DIV_NUM_LSTIMERx 和 LEDC_LSTIMERx_DUTY_RES。

LEDC_TICK_SEL_LSTIMERx 用于为低速时钟计数器 x 选择 RTC_SLOW_CLK 或 REF_TICK 时钟。（读 / 写）

1：RTC_SLOW_CLK；

0：REF_TICK。

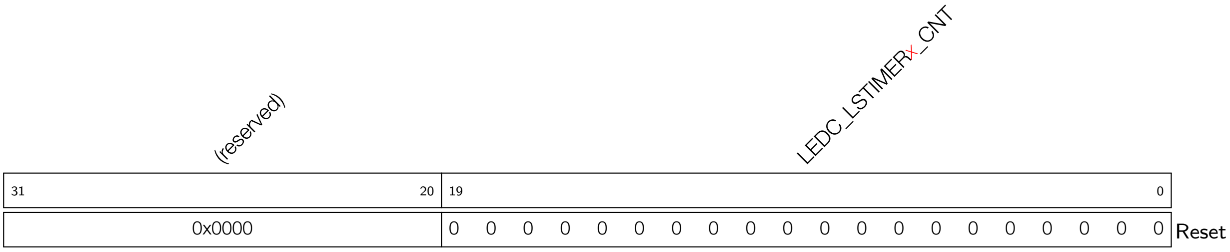
LEDC_LSTIMERx_RST 用于复位低速时钟计数器 x，复位后计数器为 0。（读 / 写）

LEDC_LSTIMERx_PAUSE 用于暂停低速时钟计数器 x。（读 / 写）

LEDC_CLK_DIV_NUM_LSTIMERx 用于配置低速时钟计数器 x，低 8 位为小数部分。（读 / 写）

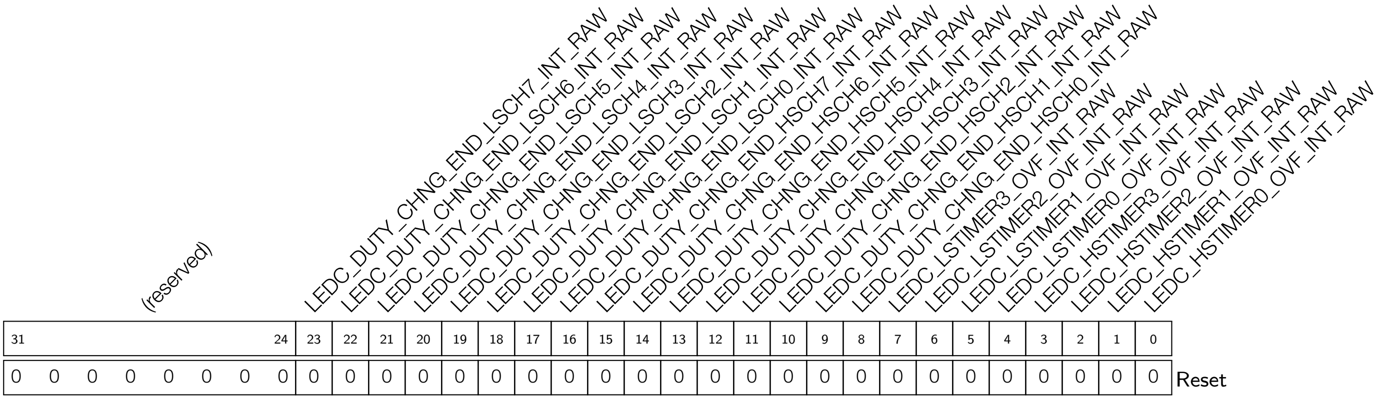
LEDC_LSTIMERx_DUTY_RES 用于控制低速时钟计数器 x 的计数范围。计数范围为 [0,2**LEDC_LSTIMERx_DUTY_RES]，最大位宽为 20 位。（读 / 写）

Register 14.14. LEDC_LSTIMERx_VALUE_REG (x: 0-3) (0x164+8*x)



LEDC_LSTIMERx_CNT 存储低速通道时钟计数器 x 的当前计数值。（只读）

Register 14.15. LEDC_INT_RAW_REG (0x0180)



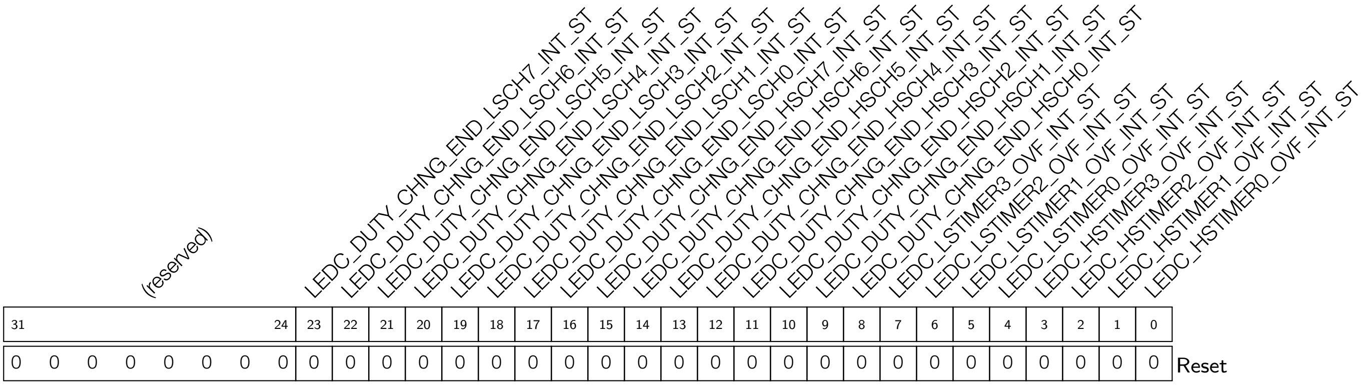
LEDC_DUTY_CHNG_END_LSCH*n*_INT_RAW LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断的原始中断状态位。（只读）

LEDC_DUTY_CHNG_END_HSCH*n*_INT_RAW LEDC_DUTY_CHNG_END_HSCH*n*_INT 中断的原始中断状态位。（只读）

LEDC_LSTIMER*x*_OVF_INT_RAW LEDC_LSTIMER*x*_OVF_INT 中断的原始中断状态位。（只读）

LEDC_HSTIMER*x*_OVF_INT_RAW LEDC_HSTIMER*x*_OVF_INT 中断的原始中断状态位。（只读）

Register 14.16. LEDC_INT_ST_REG (0x0184)



LEDC_DUTY_CHNG_END_LSCH*n*_INT_ST LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断的隐蔽中断状态位。（只读）

LEDC_DUTY_CHNG_END_HSCH*n*_INT_ST LEDC_DUTY_CHNG_END_HSCH*n*_INT 中断的隐蔽中断状态位。（只读）

LEDC_LSTIMER*x*_OVF_INT_ST LEDC_LSTIMER*x*_OVF_INT 中断的隐蔽中断状态位。（只读）

LEDC_HSTIMER*x*_OVF_INT_ST LEDC_HSTIMER*x*_OVF_INT 中断的隐蔽中断状态位。（只读）

Register 14.17. LEDC_INT_ENA_REG (0x0188)

31–24	(reserved)	0 0 0 0 0 0 0 0
23	LEDC_DUTY_CHNG_END_LSCH7_INT_ENA	0
22	LEDC_DUTY_CHNG_END_LSCH6_INT_ENA	0
21	LEDC_DUTY_CHNG_END_LSCH5_INT_ENA	0
20	LEDC_DUTY_CHNG_END_LSCH4_INT_ENA	0
19	LEDC_DUTY_CHNG_END_LSCH3_INT_ENA	0
18	LEDC_DUTY_CHNG_END_LSCH2_INT_ENA	0
17	LEDC_DUTY_CHNG_END_LSCH1_INT_ENA	0
16	LEDC_DUTY_CHNG_END_LSCH0_INT_ENA	0
15	LEDC_DUTY_CHNG_END_HSCH7_INT_ENA	0
14	LEDC_DUTY_CHNG_END_HSCH6_INT_ENA	0
13	LEDC_DUTY_CHNG_END_HSCH5_INT_ENA	0
12	LEDC_DUTY_CHNG_END_HSCH4_INT_ENA	0
11	LEDC_DUTY_CHNG_END_HSCH3_INT_ENA	0
10	LEDC_DUTY_CHNG_END_HSCH2_INT_ENA	0
9	LEDC_DUTY_CHNG_END_HSCH1_INT_ENA	0
8	LEDC_DUTY_CHNG_END_HSCH0_INT_ENA	0
7	LEDC_LSTIMER3_OVF_INT_ENA	0
6	LEDC_LSTIMER2_OVF_INT_ENA	0
5	LEDC_LSTIMER1_OVF_INT_ENA	0
4	LEDC_LSTIMER0_OVF_INT_ENA	0
3	LEDC_HSTIMER3_OVF_INT_ENA	0
2	LEDC_HSTIMER2_OVF_INT_ENA	0
1	LEDC_HSTIMER1_OVF_INT_ENA	0
0	LEDC_HSTIMER0_OVF_INT_ENA	0
		Reset

LEDC_DUTY_CHNG_END_LSCH*n*_INT_ENA LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断的使能位。（读 / 写）

LEDC_DUTY_CHNG_END_HSCH*n*_INT_ENA LEDC_DUTY_CHNG_END_HSCH*n*_INT 中断的使能位。（读 / 写）

LEDC_LSTIMER*x*_OVF_INT_ENA LEDC_LSTIMER*x*_OVF_INT 中断的使能位。（读 / 写）

LEDC_HSTIMER*x*_OVF_INT_ENA LEDC_HSTIMER*x*_OVF_INT 中断的使能位。（读 / 写）

Register 14.18. LEDC_INT_CLR_REG (0x018C)

31–24	(reserved)	0 0 0 0 0 0 0 0
23	LEDC_DUTY_CHNG_END_LSCH7_INT_CLR	0
22	LEDC_DUTY_CHNG_END_LSCH6_INT_CLR	0
21	LEDC_DUTY_CHNG_END_LSCH5_INT_CLR	0
20	LEDC_DUTY_CHNG_END_LSCH4_INT_CLR	0
19	LEDC_DUTY_CHNG_END_LSCH3_INT_CLR	0
18	LEDC_DUTY_CHNG_END_LSCH2_INT_CLR	0
17	LEDC_DUTY_CHNG_END_LSCH1_INT_CLR	0
16	LEDC_DUTY_CHNG_END_LSCH0_INT_CLR	0
15	LEDC_DUTY_CHNG_END_HSCH7_INT_CLR	0
14	LEDC_DUTY_CHNG_END_HSCH6_INT_CLR	0
13	LEDC_DUTY_CHNG_END_HSCH5_INT_CLR	0
12	LEDC_DUTY_CHNG_END_HSCH4_INT_CLR	0
11	LEDC_DUTY_CHNG_END_HSCH3_INT_CLR	0
10	LEDC_DUTY_CHNG_END_HSCH2_INT_CLR	0
9	LEDC_DUTY_CHNG_END_HSCH1_INT_CLR	0
8	LEDC_DUTY_CHNG_END_HSCH0_INT_CLR	0
7	LEDC_LSTIMER3_OVF_INT_CLR	0
6	LEDC_LSTIMER2_OVF_INT_CLR	0
5	LEDC_LSTIMER1_OVF_INT_CLR	0
4	LEDC_LSTIMER0_OVF_INT_CLR	0
3	LEDC_HSTIMER3_OVF_INT_CLR	0
2	LEDC_HSTIMER2_OVF_INT_CLR	0
1	LEDC_HSTIMER1_OVF_INT_CLR	0
0	LEDC_HSTIMER0_OVF_INT_CLR	0
		Reset

LEDC_DUTY_CHNG_END_LSCH*n*_INT_CLR 用于清除 LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断。（只写）

LEDC_DUTY_CHNG_END_HSCH*n*_INT_CLR 用于清除 LEDC_DUTY_CHNG_END_HSCH*n*_INT 中断。（只写）

LEDC_LSTIMER*x*_OVF_INT_CLR 用于清除 LEDC_LSTIMER*x*_OVF_INT 中断。（只写）

LEDC_HSTIMER*x*_OVF_INT_CLR 用于清除 LEDC_HSTIMER*x*_OVF_INT 中断。（只写）

Register 14.19. LEDC_CONF_REG (0x0190)

(reserved)	LEDC_APB_CLK_SEL	
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

LEDC_APB_CLK_SEL 用于设置 RTC_SLOW_CLK 的频率。（读 / 写）

0：8 MHz；

1：80 MHz。

15 红外遥控 (RMT)

15.1 概述

RMT（红外遥控器）是一个红外发送 / 接收控制器，其特殊设计支持生成各类信号。红外遥控发射器从内置的 RAM（随机存取存储器）区中读取连续的脉冲码，并对输出信号进行载波调制。接收器检测输入信号，并进行滤波，然后将信号高低电平以及长度值存入 RAM 中。

RMT 有 8 个通道，编码为 0 ~ 7，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。

15.2 功能描述

15.2.1 RMT 架构

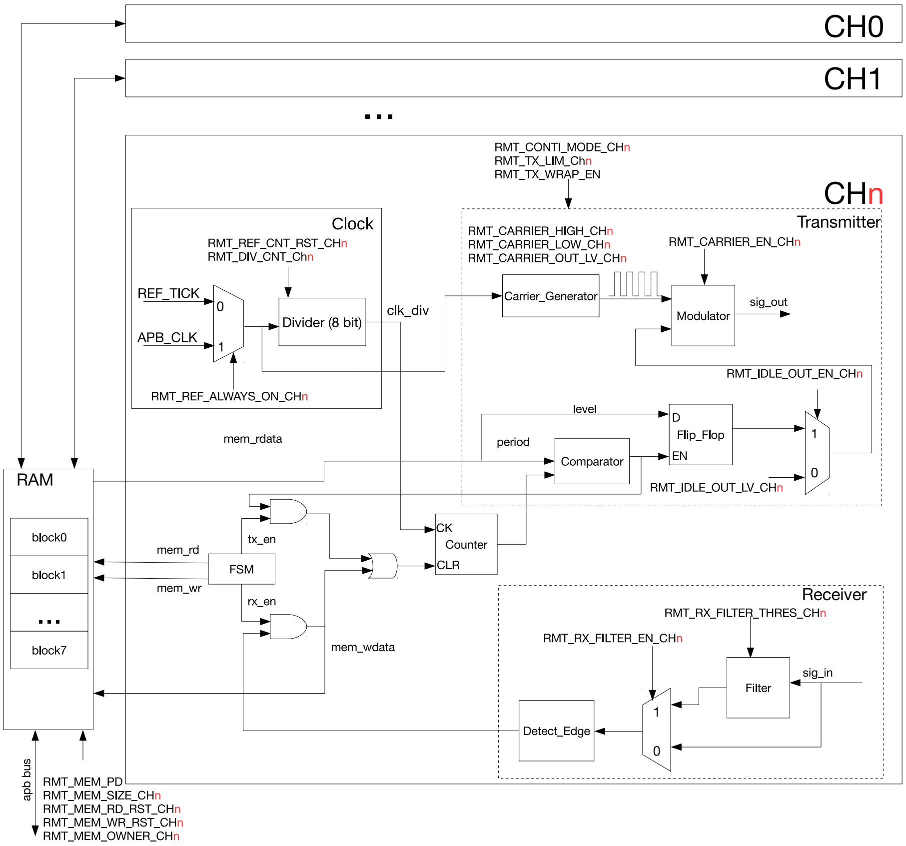


图 15-1. RMT 架构

RMT 有 8 个独立通道，每个通道内部有一个发送器和一个接收器，发送和接收不可同时工作。8 个通道共享一块 512x32 bit 的 RAM。RAM 可由处理器内核通过 APB 总线进行读写。发射器可以对信号进行载波调制。软件可以通过配置 RMT_REF_ALWAYS_ON_CH*n* 选择每个通道的工作时钟：80 MHz APB（外围总线）时钟或者 REF_TICK。

15.2.2 RMT RAM

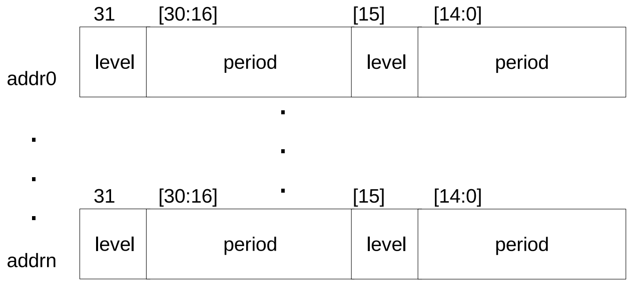


图 15-2. 数据结构

RAM 中数据结构如图 15-2 所示，由低 16 位和高 16 位组成。两个字段中 level 表示电平高低值，period 表示 level 持续的分频时钟（图 15-1 中 clk_div）周期数。period 为 0 表示结束标志，发射器停止发送数据，接收器在检测到接收的信号变成空闲状态后会写此值。

RAM 可以通过 APB 总线进行访问，其起始地址为 0x3FF56800。每个通道可使用的 RAM 范围按照 64x32 bit 分成 8 个 block。默认情况下每个通道可使用的 block 数量为 1（通道 0 使用 block0，通道 1 使用 block1，以此类推）。当通道 *n* 发送的数据量大于可使用的 block 时，除了可以通过配置 RMT_MEM_SIZE_CH*n* 寄存器扩展 block 还可以进行乒乓操作。将 RMT_MEM_SIZE_CH*n* 寄存器配置为 >1, 意味着此通道会占用下一个通道的 block。通道 *n* 使用的 RAM 地址范围为:

$\text{start\_addr\_ch}n = \text{0x3FF56800} + 64 * 4 * n$, and
$\text{end\_addr\_ch}n = \text{0x3FF56800} + (64 * 4 * n + 64 * 4 * \text{RMT\_MEM\_SIZE\_CH}n)\text{mod}(512 * 4) - 4$

为了防止数据覆盖，可以通过配置 RMT_MEM_OWNER_CH*n* 来表示发射器或者接收器对于信道 *n* 的 RAM 使用权。当配置不正确时，则会产生 RMT_CH*n*_ERR 中断。

注意：当开启连续发送模式（即置位 RMT_REG_TX_CONTI_MODE）时，发送器会连续发送通道中的数据（即从第一个数据发送到最后一个数据，再从第一个发送到最后一个，依此类推）。在连续发送模式下，第 N 次和第 N+1 次之间会有一个 clk_div 周期的 IDLE 电平。

15.2.3 时钟

软件可以通过配置 RMT_REF_ALWAYS_ON_CH*n* 选择每个通道的工作时钟：80 MHz APB 时钟或者 REF_TICK，请参考章节复位和时钟。经过 8 位宽分频器分频之后的时钟供载波发生器以及计数器使用。分频器可以通过配置 RMT_REF_CNT_RST_CH*n* 进行复位。其分频系数由 RMT_DIV_CNT_CH*n* 控制。

15.2.4 发射器

当 RMT_TX_START_CH*n* 置为 1 时，通道 *n* 的发射器开始从 RAM 中读取数据。发射器每读取一次 RAM，可以获得 32 位的数据, 低 16 位首先发送，高 16 位其次发送。

当发送的数据量大于可使用的 block 时, 可以进行乒乓操作。在进行乒乓操作前配置 RMT_TX_LIM_CH*n*，当发送的数据量大于等于 RMT_TX_LIM_CH*n* 时，会产生 RMT_CH*n*_TX_THR_EVENT_INT 中断，软件可以在检测到该中断之后更新已发送的数据。当发射器发送的数据量达到 block 的上限时, 发射器会返回到 start_addr_ch*n* 继续循环发送。

只有当数据中 period 等于 0 时，发射器将结束工作并产生 RMT_CH*n*_TX_END_INT 中断，然后返回空闲状态。空闲状态默认发送的电平为结束标志 0 对应的电平，此时也可以通过配置 RMT_IDLE_OUT_EN_CH*n* 和 RMT_IDLE_OUT_LV_CH*n* 来控制发射器的输出电平。

输出信号可以通过配置 RMT_CARRIER_EN_CH*n* 进行载波调制。载波的频率可以通过

RMT_CARRIER_HIGH_CH*n* 和 RMT_CARRIER_LOW_CH*n* 进行设置。

15.2.5 接收器

当 RMT_RX_EN_CH*n* 置为 1 时，计数器开始测量信号的长度并在下一次信号沿变化时将上次的信号高低电平以及长度值存入 RAM。当发射器长时间检测不到信号沿变化即计数器的值大于等于 RMT_IDLE_THRES_CH*n* 时，接收器结束接收过程，产生 RMT_CH*n*_RX_END_INT 中断并返回空闲状态。

输入信号可以通过置位 RMT_RX_FILTER_EN_CH*n* 来进行滤波。滤波器可以滤除信号宽度小于 RMT_RX_FILTER_THRES_CH*n* 个 APB 时钟周期的脉冲。

当 RMT 模块不工作时，可以通过配置 RMT_MEM_PD 寄存器使 RAM 工作于低功耗模式。

15.2.6 中断

- RMT_CH*n*_TX_THR_EVENT_INT：发射器每发送 RMT_CH*n*_TX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*n*_TX_END_INT：当发射器停止发送信号时，即触发此中断。
- RMT_CH*n*_RX_END_INT：当接收器停止接收信号时，即触发此中断。

15.3 寄存器列表

名称	描述	地址	访问
配置寄存器			
RMT_CH0CONF0_REG	通道 0 的配置寄存器 0	0x3FF56020	读/写
RMT_CH0CONF1_REG	通道 0 的配置寄存器 1	0x3FF56024	读/写
RMT_CH1CONF0_REG	通道 1 的配置寄存器 0	0x3FF56028	读/写
RMT_CH1CONF1_REG	通道 1 的配置寄存器 1	0x3FF5602C	读/写
RMT_CH2CONF0_REG	通道 2 的配置寄存器 0	0x3FF56030	读/写
RMT_CH2CONF1_REG	通道 2 的配置寄存器 1	0x3FF56034	读/写
RMT_CH3CONF0_REG	通道 3 的配置寄存器 0	0x3FF56038	读/写
RMT_CH3CONF1_REG	通道 3 的配置寄存器 1	0x3FF5603C	读/写
RMT_CH4CONF0_REG	通道 4 的配置寄存器 0	0x3FF56040	读/写
RMT_CH4CONF1_REG	通道 4 的配置寄存器 1	0x3FF56044	读/写
RMT_CH5CONF0_REG	通道 5 的配置寄存器 0	0x3FF56048	读/写
RMT_CH5CONF1_REG	通道 5 的配置寄存器 1	0x3FF5604C	读/写
RMT_CH6CONF0_REG	通道 6 的配置寄存器 0	0x3FF56050	读/写
RMT_CH6CONF1_REG	通道 6 的配置寄存器 1	0x3FF56054	读/写
RMT_CH7CONF0_REG	通道 7 的配置寄存器 0	0x3FF56058	读/写
RMT_CH7CONF1_REG	通道 7 的配置寄存器 1	0x3FF5605C	读/写
中断寄存器			
RMT_INT_RAW_REG	原始中断状态	0x3FF560A0	只读
RMT_INT_ST_REG	隐蔽中断状态	0x3FF560A4	只读
RMT_INT_ENA_REG	中断使能位	0x3FF560A8	读/写
RMT_INT_CLR_REG	中断清除位	0x3FF560AC	只写
载波占空比寄存器			
RMT_CH0CARRIER_DUTY_REG	通道 0 的占空比配置寄存器	0x3FF560B0	读/写
RMT_CH1CARRIER_DUTY_REG	通道 1 的占空比配置寄存器	0x3FF560B4	读/写

名称	描述	地址	访问
RMT_CH2CARRIER_DUTY_REG	通道 2 的占空比配置寄存器	0x3FF560B8	读/写
RMT_CH3CARRIER_DUTY_REG	通道 3 的占空比配置寄存器	0x3FF560BC	读/写
RMT_CH4CARRIER_DUTY_REG	通道 4 的占空比配置寄存器	0x3FF560C0	读/写
RMT_CH5CARRIER_DUTY_REG	通道 5 的占空比配置寄存器	0x3FF560C4	读/写
RMT_CH6CARRIER_DUTY_REG	通道 6 的占空比配置寄存器	0x3FF560C8	读/写
RMT_CH7CARRIER_DUTY_REG	通道 7 的占空比配置寄存器	0x3FF560CC	读/写
发送事件配置寄存器			
RMT_CH0_TX_LIM_REG	通道 0 的 Tx 事件配置寄存器	0x3FF560D0	读/写
RMT_CH1_TX_LIM_REG	通道 1 的 Tx 事件配置寄存器	0x3FF560D4	读/写
RMT_CH2_TX_LIM_REG	通道 2 的 Tx 事件配置寄存器	0x3FF560D8	读/写
RMT_CH3_TX_LIM_REG	通道 3 的 Tx 事件配置寄存器	0x3FF560DC	读/写
RMT_CH4_TX_LIM_REG	通道 4 的 Tx 事件配置寄存器	0x3FF560E0	读/写
RMT_CH5_TX_LIM_REG	通道 5 的 Tx 事件配置寄存器	0x3FF560E4	读/写
RMT_CH6_TX_LIM_REG	通道 6 的 Tx 事件配置寄存器	0x3FF560E8	读/写
RMT_CH7_TX_LIM_REG	通道 7 的 Tx 事件配置寄存器	0x3FF560EC	读/写
其他寄存器			
RMT_APB_CONF_REG	RMT 通用配置寄存器	0x3FF560F0	读/写

15.4 寄存器

本小节括号中的地址均为相对于 RMT 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 15.3 寄存器列表。

Register 15.1. RMT_CH*n*CONF0_REG (*n*: 0-7) (0x0020+8**n*)

RMT_MEM_PD 用于降低 RMT RAM 的功耗（仅存在于 RMT_CH0CONF0 寄存器）。1：RAM 处于低功耗状态；0：RAM 处于正常工作状态。（读/写）

RMT_CARRIER_OUT_LV_CH*n* 用于配置载波调制方式。0：载波加载在低电平输出信号上；1：载波加载在高电平输出信号上。（读/写）

RMT_CARRIER_EN_CH*n* 通道 *n* 的载波调制使能位。1：使能载波调制；0: 关闭载波调制。（读/写）

RMT_MEM_SIZE_CH*n* 配置通道 *n* 的 block 大小。（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 RMT_IDLE_THRES_CH*n* 时，接收器结束接收过程。（读/写）

RMT_DIV_CNT_CH*n* 用于设置通道 *n* 的分频器的分频系数。（读/写）

Register 15.2. RMT_CH*n*CONF1_REG (*n*: 0-7) (0x0024+8**n*)

(reserved)	RMT_IDLE_OUT_EN_CH*n*	RMT_IDLE_OUT_LV_CH*n*	RMT_REF_ALWAYS_ON_CH*n*	RMT_REF_CNT_RST_CH*n*	RMT_RX_FILTER_THRES_CH*n*	RMT_RX_FILTER_EN_CH*n*	RMT_TX_CONTI_MODE_CH*n*	RMT_MEM_OWNER_CH*n*	(reserved)	RMT_MEM_RD_RST_CH*n*	RMT_MEM_WR_RST_CH*n*	RMT_RX_EN_CH*n*	RMT_TX_START_CH*n*	
31 20	19	18	17	16	15 8	7	6	5	4	3	2	1	0	
0x0000	0	0	0	0	0x00F	0	0	1	0	0	0	0	0	Reset

RMT_IDLE_OUT_EN_CH*n* 通道 *n* 位于空闲状态下的输出使能控制位。(读/写)

RMT_IDLE_OUT_LV_CH*n* 配置通道 *n* 位于空闲状态下的输出信号电平。(读/写)

RMT_REF_ALWAYS_ON_CH*n* 用于选择通道的基础时钟。1：clk_apb；0：clk_ref。(读/写)

RMT_REF_CNT_RST_CH*n* 复位通道 *n* 的时钟分频器。(读/写)

RMT_RX_FILTER_THRES_CH*n* 接收模式下，通道 *n* 忽略那些宽度小于 RMT_RX_FILTER_THRES_CH*n* 个 APB 时钟周期的脉冲。(读/写)

RMT_RX_FILTER_EN_CH*n* 通道 *n* 的接收滤波使能位。(读/写)

RMT_TX_CONTI_MODE_CH*n* 发送结束时，发射器重启发送，不进入空闲状态，导致重复输出信号。(读/写)

RMT_MEM_OWNER_CH*n* 标志通道 *n* 的 RAM 使用权。1: 接收器；0: 发射器。(读/写)

RMT_MEM_RD_RST_CH*n* 用于复位通道 *n* 的发射器的 RAM 读取地址。(读/写)

RMT_MEM_WR_RST_CH*n* 用于复位通道 *n* 的接收器的 RAM 写地址。(读/写)

RMT_RX_EN_CH*n* 用于使能通道 *n* 的接收器。(读/写)

RMT_TX_START_CH*n* 用于使能通道 *n* 的发射器。(读/写)

Register 15.3. RMT_INT_RAW_REG (0x00A0)

RMT_CH7_TX_THR_EVENT_INT_RAW	RMT_CH6_TX_THR_EVENT_INT_RAW	RMT_CH5_TX_THR_EVENT_INT_RAW	RMT_CH4_TX_THR_EVENT_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_CH7_ERR_INT_RAW	RMT_CH7_RX_END_INT_RAW	RMT_CH7_TX_END_INT_RAW	RMT_CH6_ERR_INT_RAW	RMT_CH6_RX_END_INT_RAW	RMT_CH6_TX_END_INT_RAW	RMT_CH5_ERR_INT_RAW	RMT_CH5_RX_END_INT_RAW	RMT_CH5_TX_END_INT_RAW	RMT_CH4_ERR_INT_RAW	RMT_CH4_RX_END_INT_RAW	RMT_CH4_TX_END_INT_RAW	RMT_CH3_ERR_INT_RAW	RMT_CH3_RX_END_INT_RAW	RMT_CH3_TX_END_INT_RAW	RMT_CH2_ERR_INT_RAW	RMT_CH2_RX_END_INT_RAW	RMT_CH2_TX_END_INT_RAW	RMT_CH1_ERR_INT_RAW	RMT_CH1_RX_END_INT_RAW	RMT_CH1_TX_END_INT_RAW	RMT_CH0_ERR_INT_RAW	RMT_CH0_RX_END_INT_RAW	RMT_CH0_TX_END_INT_RAW	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_THR_EVENT_INT_RAW RMT_CH*n*_TX_THR_EVENT_INT 中断的原始中断状态位。（只读）

RMT_CH*n*_ERR_INT_RAW RMT_CH*n*_ERR_INT 中断的原始中断状态位。（只读）

RMT_CH*n*_RX_END_INT_RAW RMT_CH*n*_RX_END_INT 中断的原始中断状态位。（只读）

RMT_CH*n*_TX_END_INT_RAW RMT_CH*n*_TX_END_INT 中断的原始中断状态位。（只读）

Register 15.4. RMT_INT_ST_REG (0x00A4)

RMT_CH7_TX_THR_EVENT_INT_ST	RMT_CH6_TX_THR_EVENT_INT_ST	RMT_CH5_TX_THR_EVENT_INT_ST	RMT_CH4_TX_THR_EVENT_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_CH7_ERR_INT_ST	RMT_CH7_RX_END_INT_ST	RMT_CH7_TX_END_INT_ST	RMT_CH6_ERR_INT_ST	RMT_CH6_RX_END_INT_ST	RMT_CH6_TX_END_INT_ST	RMT_CH5_ERR_INT_ST	RMT_CH5_RX_END_INT_ST	RMT_CH5_TX_END_INT_ST	RMT_CH4_ERR_INT_ST	RMT_CH4_RX_END_INT_ST	RMT_CH4_TX_END_INT_ST	RMT_CH3_ERR_INT_ST	RMT_CH3_RX_END_INT_ST	RMT_CH3_TX_END_INT_ST	RMT_CH2_ERR_INT_ST	RMT_CH2_RX_END_INT_ST	RMT_CH2_TX_END_INT_ST	RMT_CH1_ERR_INT_ST	RMT_CH1_RX_END_INT_ST	RMT_CH1_TX_END_INT_ST	RMT_CH0_ERR_INT_ST	RMT_CH0_RX_END_INT_ST	RMT_CH0_TX_END_INT_ST	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RMT_CH*n*_TX_THR_EVENT_INT_ST RMT_CH*n*_TX_THR_EVENT_INT 中断的隐蔽中断状态位。（只读）

RMT_CH*n*_ERR_INT_ST RMT_CH*n*_ERR_INT 中断的隐蔽中断状态位。（只读）

RMT_CH*n*_RX_END_INT_ST RMT_CH*n*_RX_END_INT 中断的隐蔽中断状态位。（只读）

RMT_CH*n*_TX_END_INT_ST RMT_CH*n*_TX_END_INT 中断的隐蔽中断状态位。（只读）

Register 15.5. RMT_INT_ENA_REG (0x00A8)

31	RMT_CH7_TX_THR_EVENT_INT_ENA	0
30	RMT_CH6_TX_THR_EVENT_INT_ENA	0
29	RMT_CH5_TX_THR_EVENT_INT_ENA	0
28	RMT_CH4_TX_THR_EVENT_INT_ENA	0
27	RMT_CH3_TX_THR_EVENT_INT_ENA	0
26	RMT_CH2_TX_THR_EVENT_INT_ENA	0
25	RMT_CH1_TX_THR_EVENT_INT_ENA	0
24	RMT_CH0_TX_THR_EVENT_INT_ENA	0
23	RMT_CH7_ERR_INT_ENA	0
22	RMT_CH7_RX_END_INT_ENA	0
21	RMT_CH7_TX_END_INT_ENA	0
20	RMT_CH6_ERR_INT_ENA	0
19	RMT_CH6_RX_END_INT_ENA	0
18	RMT_CH6_TX_END_INT_ENA	0
17	RMT_CH5_ERR_INT_ENA	0
16	RMT_CH5_RX_END_INT_ENA	0
15	RMT_CH5_TX_END_INT_ENA	0
14	RMT_CH4_ERR_INT_ENA	0
13	RMT_CH4_RX_END_INT_ENA	0
12	RMT_CH4_TX_END_INT_ENA	0
11	RMT_CH3_ERR_INT_ENA	0
10	RMT_CH3_RX_END_INT_ENA	0
9	RMT_CH3_TX_END_INT_ENA	0
8	RMT_CH2_ERR_INT_ENA	0
7	RMT_CH2_RX_END_INT_ENA	0
6	RMT_CH2_TX_END_INT_ENA	0
5	RMT_CH1_ERR_INT_ENA	0
4	RMT_CH1_RX_END_INT_ENA	0
3	RMT_CH1_TX_END_INT_ENA	0
2	RMT_CH0_ERR_INT_ENA	0
1	RMT_CH0_RX_END_INT_ENA	0
0	RMT_CH0_TX_END_INT_ENA	0
	Reset	

RMT_CH*n*_TX_THR_EVENT_INT_ENA RMT_CH*n*_TX_THR_EVENT_INT 中断的使能位。（读/写）

RMT_CH*n*_ERR_INT_ENA RMT_CH*n*_ERROR_INT 中断的使能位。（读/写）

RMT_CH*n*_RX_END_INT_ENA RMT_CH*n*_RX_END_INT 中断的使能位。（读/写）

RMT_CH*n*_TX_END_INT_ENA RMT_CH*n*_TX_END_INT 中断的使能位。（读/写）

Register 15.6. RMT_INT_CLR_REG (0x00AC)

31	RMT_CH7_TX_THR_EVENT_INT_CLR	0
30	RMT_CH6_TX_THR_EVENT_INT_CLR	0
29	RMT_CH5_TX_THR_EVENT_INT_CLR	0
28	RMT_CH4_TX_THR_EVENT_INT_CLR	0
27	RMT_CH3_TX_THR_EVENT_INT_CLR	0
26	RMT_CH2_TX_THR_EVENT_INT_CLR	0
25	RMT_CH1_TX_THR_EVENT_INT_CLR	0
24	RMT_CH0_TX_THR_EVENT_INT_CLR	0
23	RMT_CH7_ERR_INT_CLR	0
22	RMT_CH7_RX_END_INT_CLR	0
21	RMT_CH7_TX_END_INT_CLR	0
20	RMT_CH6_ERR_INT_CLR	0
19	RMT_CH6_RX_END_INT_CLR	0
18	RMT_CH6_TX_END_INT_CLR	0
17	RMT_CH5_ERR_INT_CLR	0
16	RMT_CH5_RX_END_INT_CLR	0
15	RMT_CH5_TX_END_INT_CLR	0
14	RMT_CH4_ERR_INT_CLR	0
13	RMT_CH4_RX_END_INT_CLR	0
12	RMT_CH4_TX_END_INT_CLR	0
11	RMT_CH3_ERR_INT_CLR	0
10	RMT_CH3_RX_END_INT_CLR	0
9	RMT_CH3_TX_END_INT_CLR	0
8	RMT_CH2_ERR_INT_CLR	0
7	RMT_CH2_RX_END_INT_CLR	0
6	RMT_CH2_TX_END_INT_CLR	0
5	RMT_CH1_ERR_INT_CLR	0
4	RMT_CH1_RX_END_INT_CLR	0
3	RMT_CH1_TX_END_INT_CLR	0
2	RMT_CH0_ERR_INT_CLR	0
1	RMT_CH0_RX_END_INT_CLR	0
0	RMT_CH0_TX_END_INT_CLR	0
	Reset	

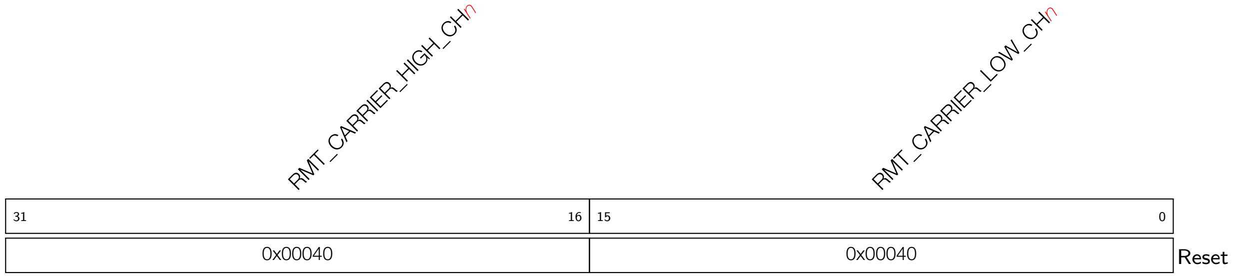
RMT_CH*n*_TX_THR_EVENT_INT_CLR 用于清除 RMT_CH*n*_TX_THR_EVENT_INT 中断。（只写）

RMT_CH*n*_ERR_INT_CLR 用于清除 RMT_CH*n*_ERRINT 中断。（只写）

RMT_CH*n*_RX_END_INT_CLR 用于清除 RMT_CH*n*_RX_END_INT 中断。（只写）

RMT_CH*n*_TX_END_INT_CLR 用于清除 RMT_CH*n*_TX_END_INT 中断。（只写）

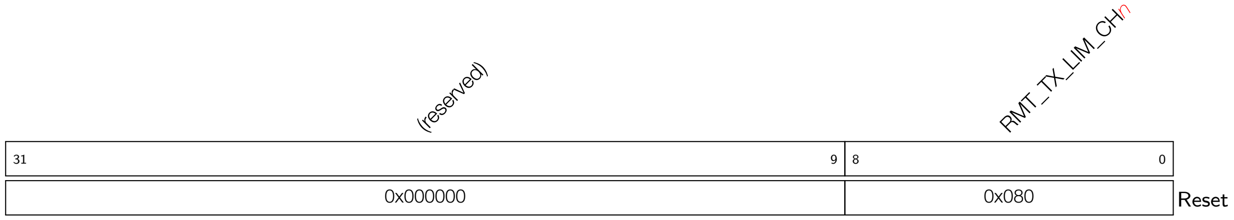
Register 15.7. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-7) (0x00B0+4**n*)



RMT_CARRIER_HIGH_CH*n* 用于配置通道 *n* 的载波的高电平时钟周期。时钟源可以是 REF_TICK 或 APB_CLK。（读/写）

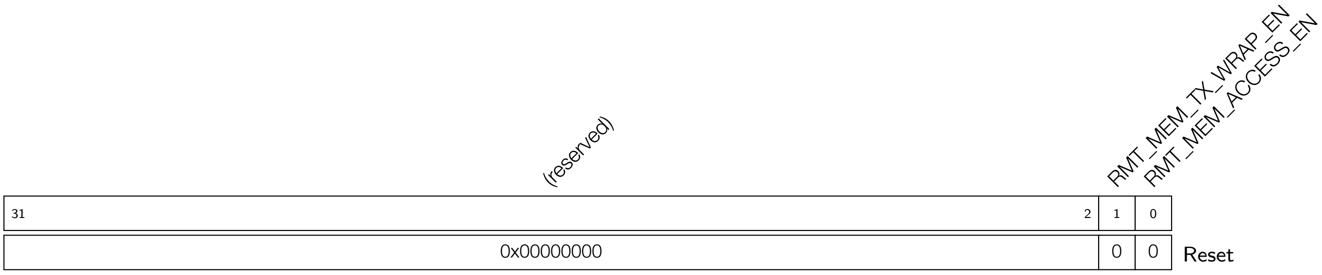
RMT_CARRIER_LOW_CH*n* 用于配置通道 *n* 的载波的低电平时钟周期。时钟源可以是 REF_TICK 或 APB_CLK。（读/写）

Register 15.8. RMT_CH*n*_TX_LIM_REG (*n*: 0-7) (0x00D0+4**n*)



RMT_TX_LIM_CH*n* 当通道发送的数据量大于指定的 block 大小，则产生 TX_THR_EVENT 中断。（读/写）

Register 15.9. RMT_APB_CONF_REG (0x00F0)



RMT_MEM_TX_WRAP_EN 乒乓操作使能位。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_ACCESS_EN 必须置 1，否则无法访问 RMT 存储器。

16 电机控制脉宽调制器 (PWM)

16.1 概述

电机控制脉宽调制器 (MCPWM) 外设用于电机和电源控制。它提供了六个 PWM 输出，可在几种拓扑结构中运行。常见的拓扑结构之一是用一对 PWM 输出来驱动 H 桥以控制电机旋转速度和旋转方向。

MCPWM 的时序和控制资源分为两种主要类型的模块：PWM 定时器和 PWM 操作器。每个 PWM 定时器提供定时参考，可以自由运行，或同步到其他定时器或外部源。每个 PWM 操作器具有用于为一个 PWM 通道生成波形对的所有控制资源。MCPWM 外设还包含专用捕获模块，用于需要精确定时外部事件的系统。

ESP32 有两个 MCPWM 外设，分别是 MCPWM0 和 MCPWM1。它们的控制寄存器分别位于从地址 0x3FF5E000 和 0x3FF6C000 开始的 4 KB 内存中。详见章节 16.4 寄存器列表。

16.2 主要特性

每个 MCPWM 外设都有一个时钟分频器（预分频器），三个 PWM 定时器，三个 PWM 操作器和一个捕获模块。图 16-1 展示了 MCPWM 的模块和接口上的信号。PWM 定时器用于生成定时参考。PWM 操作器将根据定时参考生成所需的波形。通过配置，任一 PWM 操作器可以使用任一 PWM 定时器的定时参考。不同的 PWM 操作器可以使用相同的 PWM 定时器的定时参考来产生 PWM 信号。此外，不同的 PWM 操作器也可以使用不同的 PWM 定时器的值来生成单独的 PWM 信号。不同的 PWM 定时器也可进行同步。

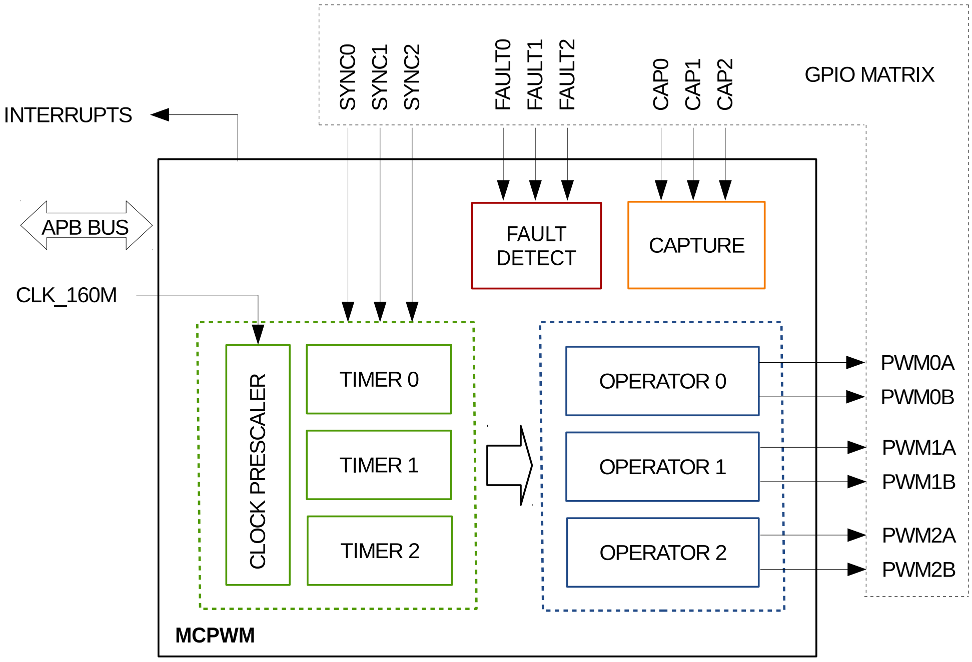


图 16-1. MCPWM 外设概览

以下是图 16-1 中模块的功能概述：

- PWM 定时器 0，1 和 2
    - 每个 PWM 定时器都有一个专用的 8 位时钟预分频器。
    - PWM 定时器中的 16 位计数的工作模式包括：递增计数模式，递减计数模式，递增递减循环计数模

    式。

  - 硬件同步可以触发 PWM 定时器重载, 重载值位于相位寄存器中；同时触发预分频的重启，从而同步定时器的时钟。同步源可以来自任何 GPIO 或任何其他 PWM 定时器的 sync_out。

- PWM 操作器 0，1 和 2
  - 每个 PWM 操作器有两个 PWM 输出（PWMxA 和 PWMxB），可以在对称和非对称配置中独立工作。
  - 软件异步优先控制 PWM 信号。
  - 死区在上升沿和下降沿可配置，并可分别设置。
  - 所有事件都可触发 CPU 中断。
  - 通过高频载波信号调制 PWM 输出，在使用变压器隔离栅极驱动器时非常有用。
  - 周期，时间戳寄存器和其他主要的控制寄存器有影子寄存器，更新方法灵活。
- 故障检测模块
  - 出现故障时，可选择在逐周期模式或一次性模式下处理。
  - 故障条件可强制 PWM 输出高或低电平。
- 捕获模块
  - 旋转电机的速度测量。
  - 位置传感器脉冲之间的间隔时间测量。
  - 脉冲序列信号的周期和占空比测量。
  - 从占空比编码的电流/电压传感器导出的解码电流或电压振幅。
  - 3 个独立的捕获通道，各具备一个 32 位的时间戳寄存器。
  - 输入捕获信号可以预分频，边沿极性可选。
  - 捕获定时器可以与 PWM 定时器或外部信号同步。
  - 3 个捕获通道上都可以产生中断。

16.3 模块

16.3.1 模块概述

以下提供 MCPWM 关键模块的主要配置参数。调整特定参数，例如 PWM 定时器的同步源，请参考第 16.3.2 章。

16.3.1.1 预分频器模块

图 16-2. 预分频器模块

配置参数：

- 根据 CLK_160M 对 PWM 时钟进行分频。

16.3.1.2 定时器模块

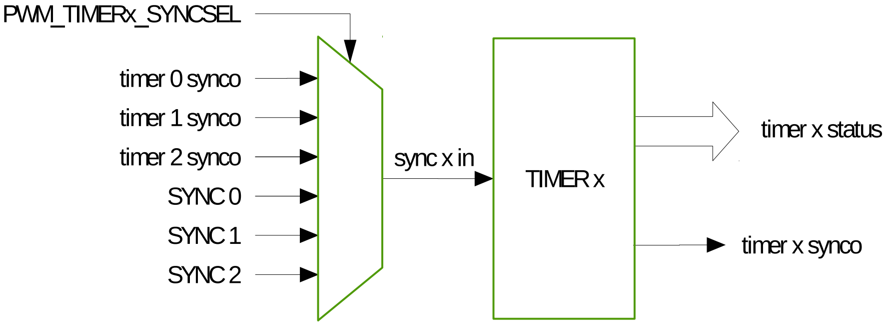


图 16-3. 定时器模块

配置参数：

- 设置 PWM 定时器的频率或周期。
- 配置定时器的工作模式：
  - 递增计数模式：用于非对称 PWM 输出
  - 递减计数模式：用于非对称 PWM 输出
  - 递增递减循环计数模式：用于于对称 PWM 输出
- 配置软件或硬件同步发生时的重载相位，包括值和方向。

- 通过硬件或软件同步使 PWM 定时器彼此同步。
- 设置 PWM 定时器的同步输入源，7 选 1：
  - 3 个 PWM 定时器的同步输出
  - 来自 GPIO 矩阵的 3 个同步信号（SYNC0，SYNC1，SYNC2）
  - 未选择同步输入信号
- 配置 PWM 定时器的同步输出源或输出时刻，4 选 1：
  - 同步输入信号
  - PWM 定时器的值为 0 时
  - PMW 定时器的值等于时钟周期的值
  - 没有生成同步输出
- 配置周期更新方式。

16.3.1.3 操作器模块

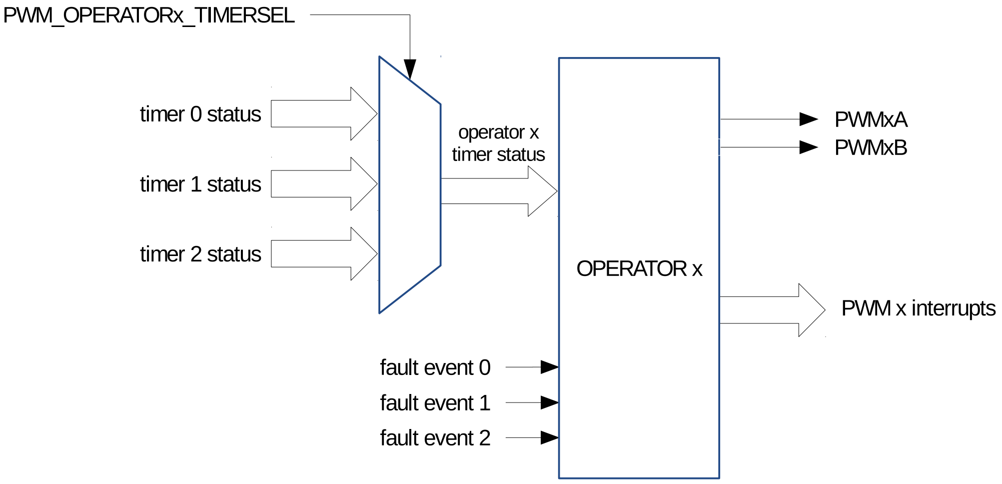


图 16-4. 操作器模块

表 16-1 列举操作器模块的主要配置参数。

表 16-1. 操作器模块的配置参数

子模块	配置参数 / 选项
PWM 生成器	• 设置 PWMxA 和 / 或 PWMxB 输出的 PWM 占空比 • 设置定时事件发生的时间 • 设置发生定时事件时采取的行动： – 改变 PWMxA 和 / 或 PWMxB 输出为高或低 – 将 PWMxA 和 / 或 PWMxB 取反 – 不对输出执行任何操作 • 通过直接 S/W 控制强制 PWM 输出的状态 • 在 PWM 输出的上升和 / 或下降边沿上增加死区 • 配置此子模块的更新方式
死区生成器	• 控制高侧和低侧开关之间的互补死区关系 • 指定上升沿死区 • 指定下降沿死区 • 绕过死区发生器模块，PWM 波形不插入死区 • 可根据 PWMxA 输出进行 PWMxB 相移 • 配置此子模块的更新方式
PWM 载波	• 使能载波，设置载波频率 • 设置载波波形中第一个脉冲的持续时间 • 设置第二个以及之后的脉冲的占空比 • 绕过 PWM 载波模块，PWM 波形无变动
故障处理器	• 配置 PWM 模块是否以及如何响应故障事件信号 • 指定发生故障事件时采取的操作： – 强制 PWMxA 和 / 或 PWMxB 为高电平 – 强制 PWMxA 和 / 或 PWMxB 为低电平 – 配置 PWMxA 和 / 或 PWMxB 忽略任何故障事件 • 配置 PWM 应对故障事件的模式： – 一次性模式 – 逐周期模式 • 生成中断 • 绕过故障处理器模块 • 设置逐周期操作清除的方式 • 当时基计数器采取倒计时和正计时计数时，可采取不同操作

16.3.1.4 故障检测模块

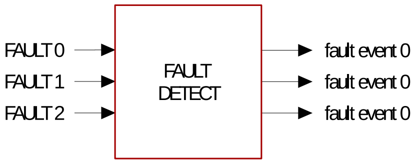


图 16-5. 故障检测模块

配置参数：

- 开启故障事件的生成，并为每个故障信号配置故障事件生成的极性
- 生成故障事件中断

16.3.1.5 捕获模块

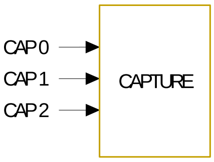


图 16-6. 捕获模块

配置参数：

- 选择捕获模块输入的边沿极性和预分频
- 设置软件触发捕获
- 配置捕获定时器同步触发和同步相位
- 软件同步捕获定时器

16.3.2 PWM 定时器模块

每个 MCPWM 外设都有三个 PWM 定时器模块。它们中的任何一个都可以决定三个 PWM 操作器模块中任意一个的必要事件时序。通过使用 GPIO 矩阵的同步信号，内置同步逻辑允许一个或多个 MCPWM 外设中的多个 PWM 定时器模块作为一个系统协同工作。

16.3.2.1 PWM 定时器模块的配置

用户可配置 PWM 定时器模块的以下功能：

- 通过指定 PWM 定时器频率或周期来控制事件发生的频率。
- 配置特定 PWM 定时器与其他 PWM 定时器或模块同步。

- 使 PWM 定时器与其他 PWM 定时器或模块同相。
- 设置定时器计数模式：递增，递减，或递增递减循环计数模式。
- 使用预分频器更改 PWM 定时器时钟（PT_clk）的速率。每个定时器都有自己的预分频器，通过寄存器 PWM_TIMER0_CFG0_REG 的 PWM_TIMERx_PRESCALE 配置。PWM 定时器根据该寄存器的设置以较慢的速度递增或递减。

16.3.2.2 PWM 定时器工作模式和定时事件生成

PWM 定时器有三种工作模式，由 PWMx 定时器模式寄存器配置：

- 递增计数模式：
  定时器从零增加到周期寄存器中配置的值。一旦到达周期值，PWM 定时器清零，并再次开始递增。PWM 周期等于周期寄存器中的周期值 + 1。
  说明：周期寄存器为 PWM_TIMERx_PERIOD，x = 0, 1, 2，对应 PWM_TIMER0_PERIOD，PWM_TIMER1_PERIOD，PWM_TIMER2_PERIOD。
- 递减计数模式：
  PWM 定时器从周期寄存器中的值开始递减到零。达到零后，将恢复为周期值，再次开始递减。在这种情况下，PWM 周期等于周期寄存器中的周期值 + 1。
- 递增-递减循环模式：
  此模式结合了上述两种模式。PWM 定时器从零开始递增，直到达到周期值，再次递减为零。PWM 定时器按照此模式循环递增递减。PWM 周期为周期寄存器的周期值 × 2 + 1。

图 16-7 至 16-10 显示不同的模式下 PWM 定时器波形，包括同步事件期间的定时器行为。

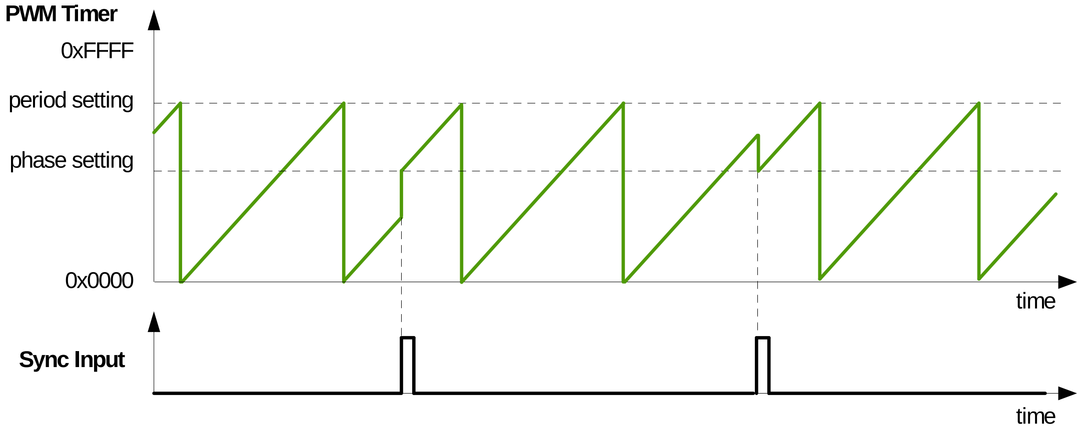


图 16-7. 递增计数模式波形

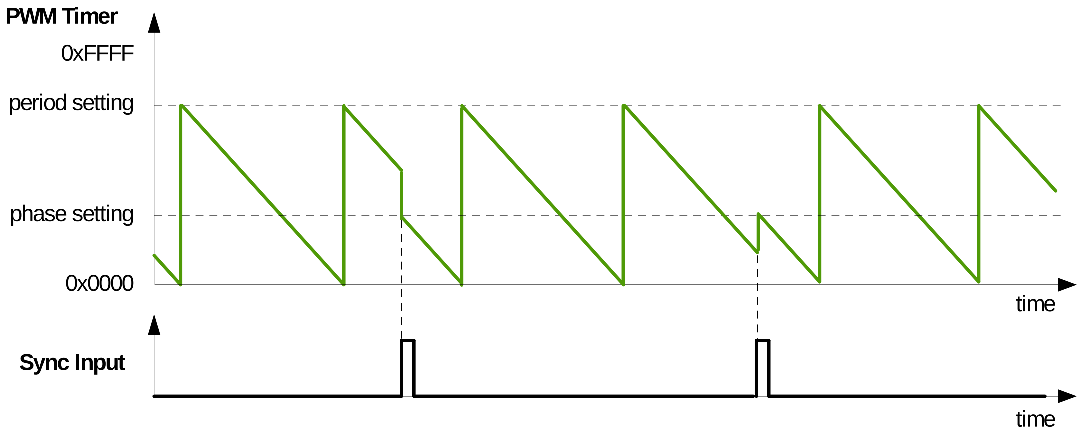


图 16-8. 递减计数模式波形

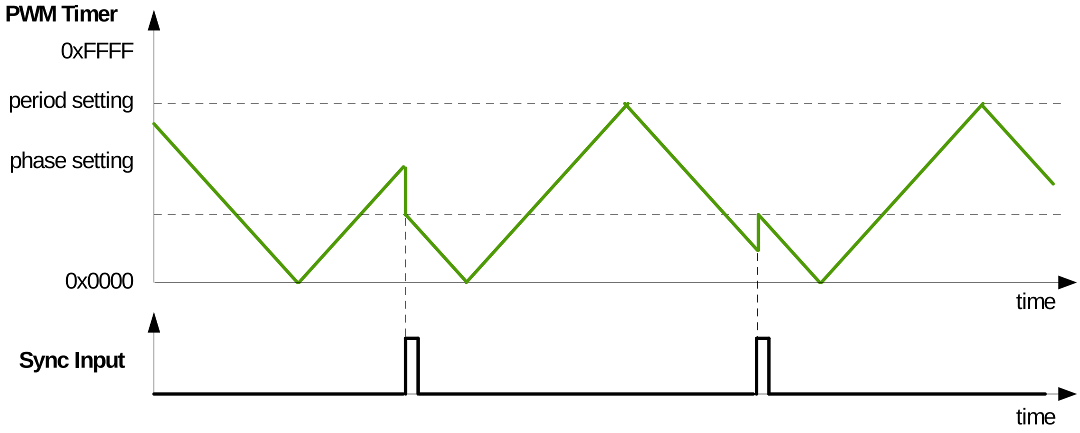


图 16-9. 递增递减循环模式波形，同步事件后递减

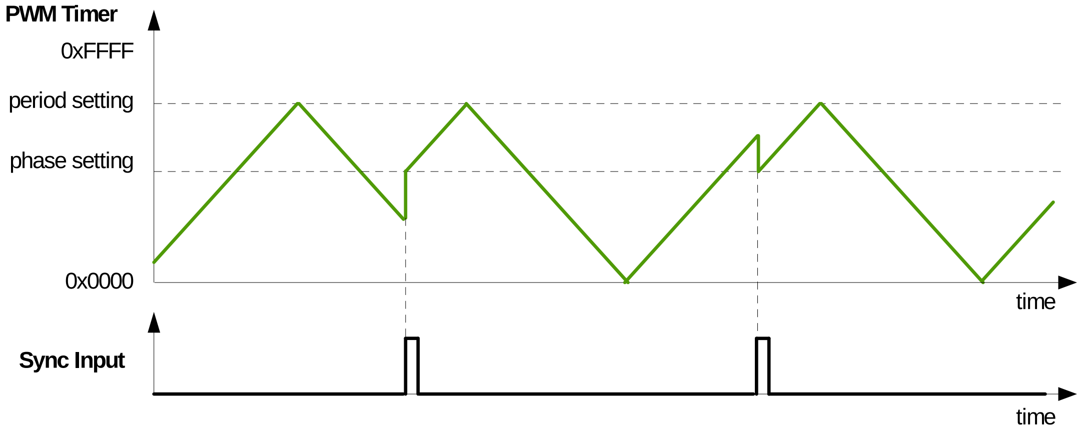


图 16-10. 递增递减循环模式波形，同步事件后递增

当 PWM 定时器运行时，定期自动地生成以下定时事件：

- UTEP
  当 PWM 定时器等于周期寄存器值（PWM_TIMERx_PEROD）且 PWM 定时器递增计数时生成的定时事件。
- UTEZ
  当 PWM 定时器等于零且 PWM 定时器递增计数时生成的定时事件。
- DTEP
  当 PWM 定时器等于周期寄存器值（PWM_TIMERx_PEROD）且 PWM 定时器递减时生成的定时事件。
- DTEZ
  当 PWM 定时器等于零且 PWM 定时器递减时生成的定时事件。

图 16-11 至 16-13 为 U/DTEP 和 U/DTEZ 定时事件的时序波形。

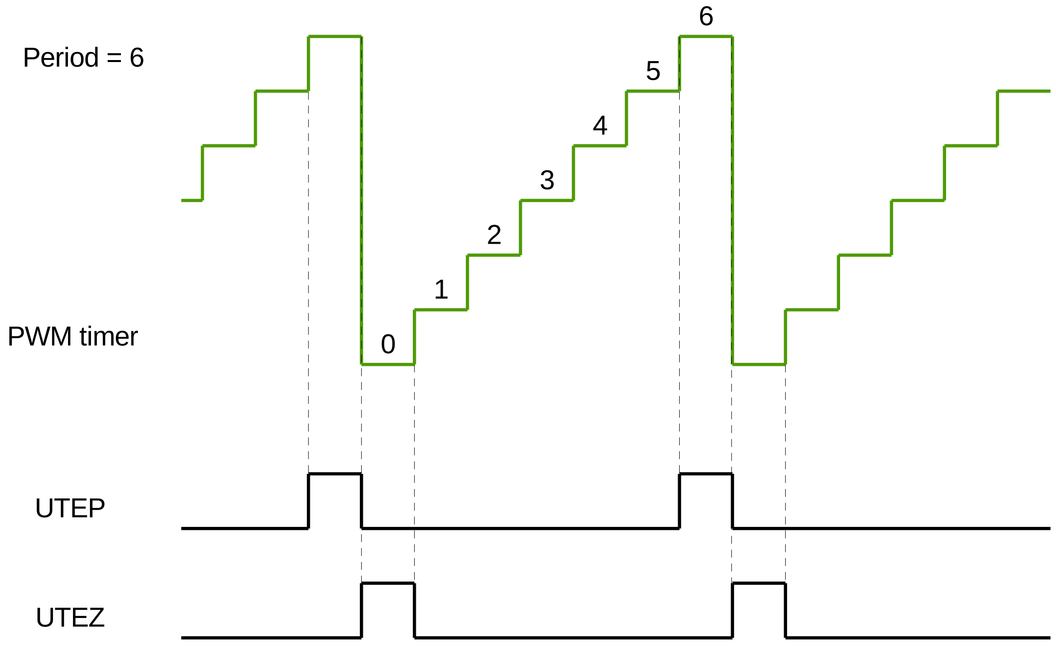


图 16-11. 递增模式中生成的 UTEP 和 UTEZ

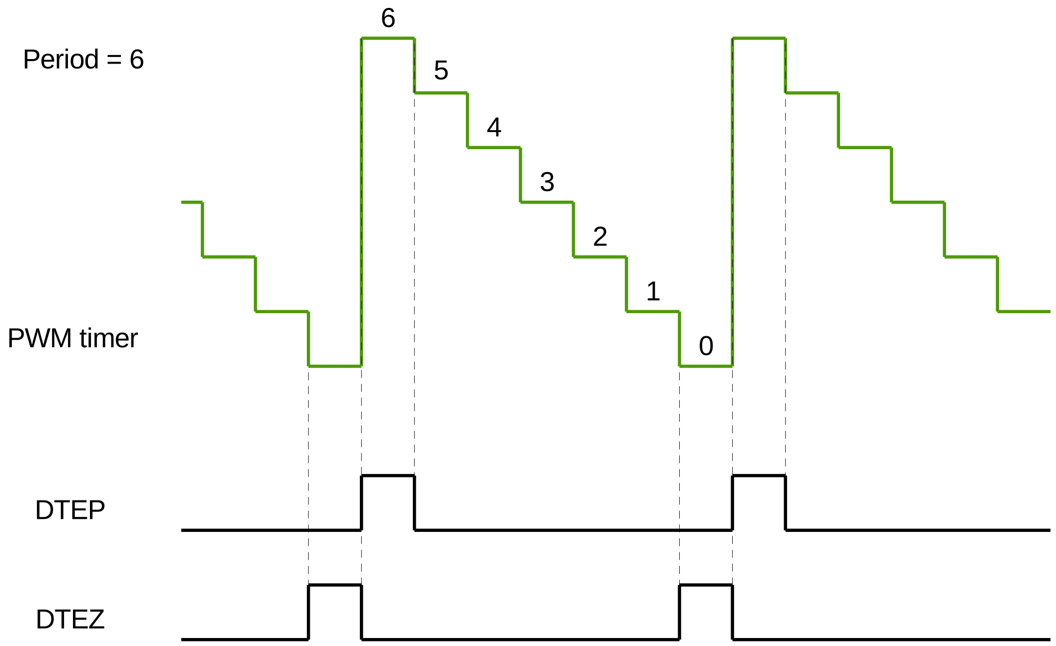


图 16-12. 递减模式中生成的 DTEP 和 DTEZ

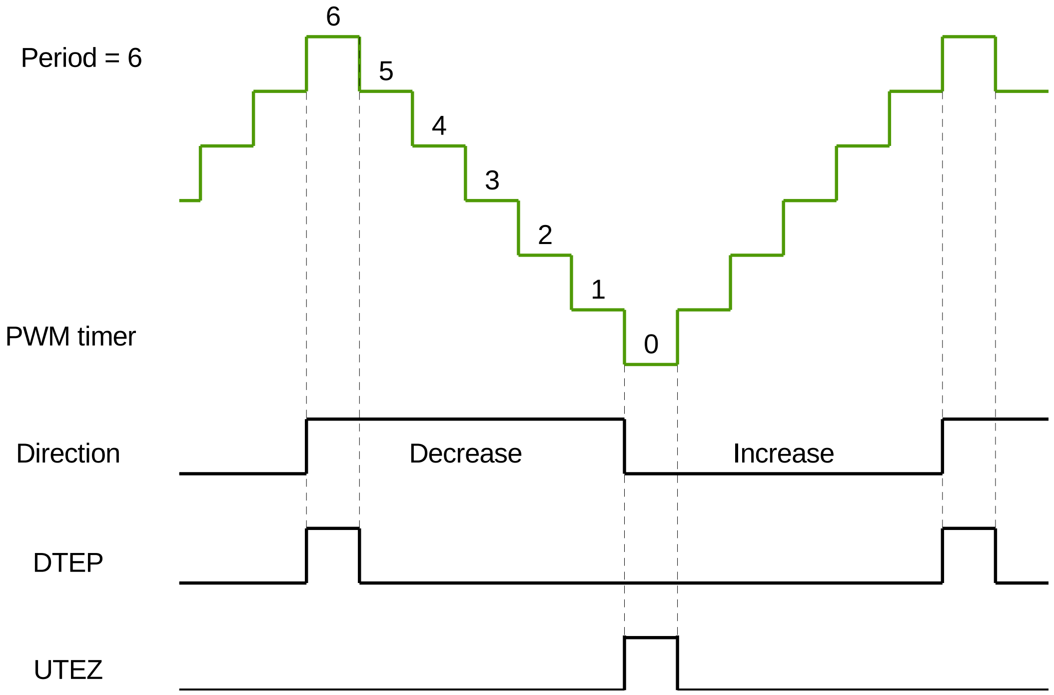


图 16-13. 递增递减模式中生成的 DTEP 和 UTEZ

需注意，在递增递减模式下，当定时器递增计数时，定时器范围为 [0，周期值 - 1]；递减计数时，定时器范围为 [周期值，1]。也就是说，在该模式下，将定时器同步为 0 时，则不支持递减计数，即 MCPWM_TIMER*n*_PHASE_DIRECTION 不能为 1；定时器同步为周期值时，也不支持递增计数，即 MCPWM_TIMER*n*_PHASE_DIRECTION 不能为 0。所以，当定时器同步为 0 时，计数方向只能为递增，即将 MCPWM_TIMER*n*_PHASE_DIRECTION 配置为 0；定时器同步为周期值时，计数方向只能为递减，即将 MCPWM_TIMER*n*_PHASE_DIRECTION 配置为 1。

16.3.2.3 PWM 定时器影子寄存器

PWM 定时器周期寄存器和 PWM 定时器时钟预分频器寄存器具有影子寄存器。影子寄存器的作用是保存某个值，在与硬件同步的特定时刻写入有效寄存器。两种寄存器类型定义如下：

- 有效寄存器
  有效寄存器直接控制硬件执行的所有操作。

- 影子寄存器
  影子寄存器作为要写入有效寄存器的值的临时缓存。在用户配置的某个时间点，影子寄存器中的值被写入有效寄存器。在此之前，影子寄存器的内容对受控硬件没有任何直接影响。这有助于防止寄存器由软件异步修改时可能发生的错误硬件操作。影子寄存器和有效寄存器具有相同的存储器地址。软件总是写入或读取影子寄存器。有效寄存器的更新时间点由其特定的更新方式寄存器决定。更新时间点可以是 PWM 定时器等于零时，PWM 定时器等于周期时，同步时间点或立即。软件可以触发强制全局更新，根据影子寄存器更新模块中的所有有效寄存器。

16.3.2.4 PWM 定时器同步和锁相

PWM 模块采用灵活的同步方法。每个 PWM 定时器都有一个同步输入和一个同步输出。同步输入可以从 GPIO 矩阵的三个同步输出和三个同步信号中选择。同步输出可以使用同步输入信号，或在 PWM 定时器等于周期或 PWM 定时器等于零时产生。因此，PWM 定时器可以通过同步使它们之间的相位锁定。在同步期间，PWM 定时器时钟预分频器将复位其计数器，以同步 PWM 定时器时钟。

16.3.3 PWM 操作器模块

PWM 操作器模块具备以下功能：

- 根据相应 PWM 定时器的定时参考生成 PWM 信号对。
- PWM 信号对的每个信号都可以独立设置特定的死区。
- 可通过配置将载波叠加到 PWM 信号上。
- 故障条件下处理响应。

图 16-14 为 PWM 操作器的框图。

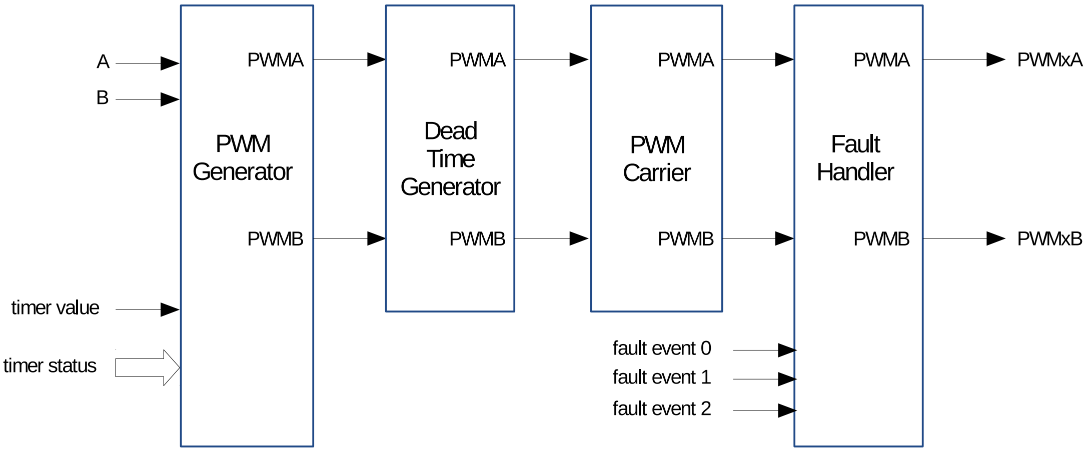


图 16-14. PWM 操作器的子模块

16.3.3.1 PWM 生成器模块

PWM 生成器模块的作用

此模块中生成或导入重要的时序事件，并转化为特定操作，在 PWMxA 和 PWMxB 输出处生成所需的波形。

PWM 生成器模块执行以下操作：

- 基于使用寄存器 A 和 B 配置的时间戳生成定时事件。满足以下条件时发生定时事件：
  - UTEA：PWM 定时器递增计数并且其值等于寄存器 A。
  - UTEB：PWM 定时器递增计数并且其值等于寄存器 B。

  - DTEA：PWM 定时器递减计数并且其值等于寄存器 A。
  - DTEB：PWM 定时器递减计数并且其值等于寄存器 B。
- 基于故障或同步事件生成 U/DT1，U/DT2 定时事件。
- 当这些定时事件同时发生时管理优先级。
- 基于定时事件产生置 1，置 0 和取反操作。
- 根据 PWM 生成器模块的配置来控制 PWM 占空比。
- 使用影子寄存器处理新的时间戳值，以防止 PWM 波形中的脉冲干扰。

PWM 操作器影子寄存器

时间戳寄存器 A 和 B，以及操作配置寄存器 PWM_GEN*x*_A_REG 和 PWM_GEN*x*_B_REG 都有影子寄存器。影子寄存器提供了一种与硬件同步更新寄存器的方法。影子寄存器描述请查看第 16.3.2.3 章。

定时事件

表 16-2 概括了所有定时信号和事件。

表 16-2. PWM 生成器中的所有定时事件

信号	事件描述	PWM 定时器操作
DTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递减计数
DTEZ	PWM 定时器的值等于 0	
DTEA	PWM 定时器的值等于寄存器 A	
DTEB	PWM 定时器的值等于寄存器 B	
DT0 事件	基于故障或同步事件	
DT1 事件	基于故障或同步事件	
UTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递增计数
UTEZ	定时器的值等于 0	
UTEA	PWM 定时器的值等于寄存器 A	
UTEB	PWM 定时器的值等于寄存器 B	
UT0 事件	基于故障或同步事件	
UT1 事件	基于故障或同步事件	
软件强制事件	软件触发的异步事件	-

软件强制事件用于在 PWM*x*A 和 PWM*x*B 输出上施加非连续或连续的强制电平。此更改是异步完成的。软件强制由寄存器 PWM_PWM_GEN*x*_FORCE_REG 控制。

PWM 生成器模块中 T0/T1 的选择和配置独立于故障处理模块中的故障事件的配置。故障事件可以不被配置为在故障处理器模块中引起跳闸动作，但相同的事件可以由 PWM 生成器用于触发 T0/T1 以控制 PWM 波形。

需要注意的是，当 PWM 定时器处于递增递减循环计数模式时，它将在 TEP 事件后递减，在 TEZ 事件后递增。因此，当 PWM 定时器处于此模式时，将出现 DTEP 和 UTEZ，但 UTEP 和 DTEZ 不会出现。

PWM 生成器可以同时处理多个事件。事件优先级由硬件决定，详见表 16-3 和表 16-4。优先级从 1（最高）到

7（最低）排列。需要注意的是，TEP 和 TEZ 事件的优先级取决于 PWM 定时器的计数模式。

如果 A 或 B 的值设置为大于周期，则 U/DTEA 和 U/DTEB 将永远不会发生。

表 16-3. PWM 定时器递增计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	UTEP
3	UT0
4	UT1
5	UTEB
6	UTEA
7（最低）	UTEZ

表 16-4. PWM 定时器递减计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	DTEZ
3	DT0
4	DT1
5	DTEB
6	DTEA
7（最低）	DTEP

说明：

1. UTEP 和 UTEZ 不同时发生。当 PWM 定时器处于递增计数模式，UTEP 将始终比 UTEZ 提前一个周期发生，如图 16-11 所示，因此它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，UTEP 不会发生。
2. DTEP 和 DTEZ 不同时发生。当 PWM 定时器处于递减计数模式时，DTEZ 始终比 DTEP 早一个周期发生，如图 16-12 所示，因此它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，DTEZ 不会发生。

PWM 信号生成

当某个定时事件发生时，PWM 生成器控制输出 PWMxA 和 PWMxB 的电平。定时事件通过 PWM 定时器计数方向（递增或递减）进一步限定。根据定时器计数方向，模块可以对 PWM 定时器递增或递减计数的阶段执行不同的操作。

可以在 PWMxA 和 PWMxB 输出上配置以下操作：

- 置为高电平：
  将 PWMxA 或 PWMxB 的输出设置为高电平。
- 置为低电平：
  通过将 PWMxA 或 PWMxB 的输出设置为低电平来清除 PWMxA 或 PWMxB 的输出。

- 取反：
  将 PWMxA 或 PWMxB 的当前输出电平更改为相反的值。如果它当前被拉高，则拉低，或反之。
- 不进行操作：
  保持 PWMxA 和 PWMxB 输出电平不变。在这种状态下，仍然可以触发中断。

输出上的操作通过寄存器 PWN_GENx_A_REG 和 PWN_GENx_B_REG 配置。每一次输出的操作都独立配置。此外，基于事件在某个输出上灵活地执行不同的操作。表 16-2 中列举的任何事件都可以作用于 PWMxA 或 PWMxB 输出上。关于生成器 0，1 或 2 的寄存器信息，请参考第 16.4 章。

常见配置的波形

图 16-15 为 PWM 定时器在递增递减循环计数时生成的对称 PWM 波形。该模式下的直流 0%-100% 调制可由以下公式获得：

$$Duty = (Period - A) \div Period$$

如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递增，则 PWM 输出被上拉。如果 A 的值在 PWM 定时器递减时等于 PWM 定时器的值，则 PWM 输出被拉低。

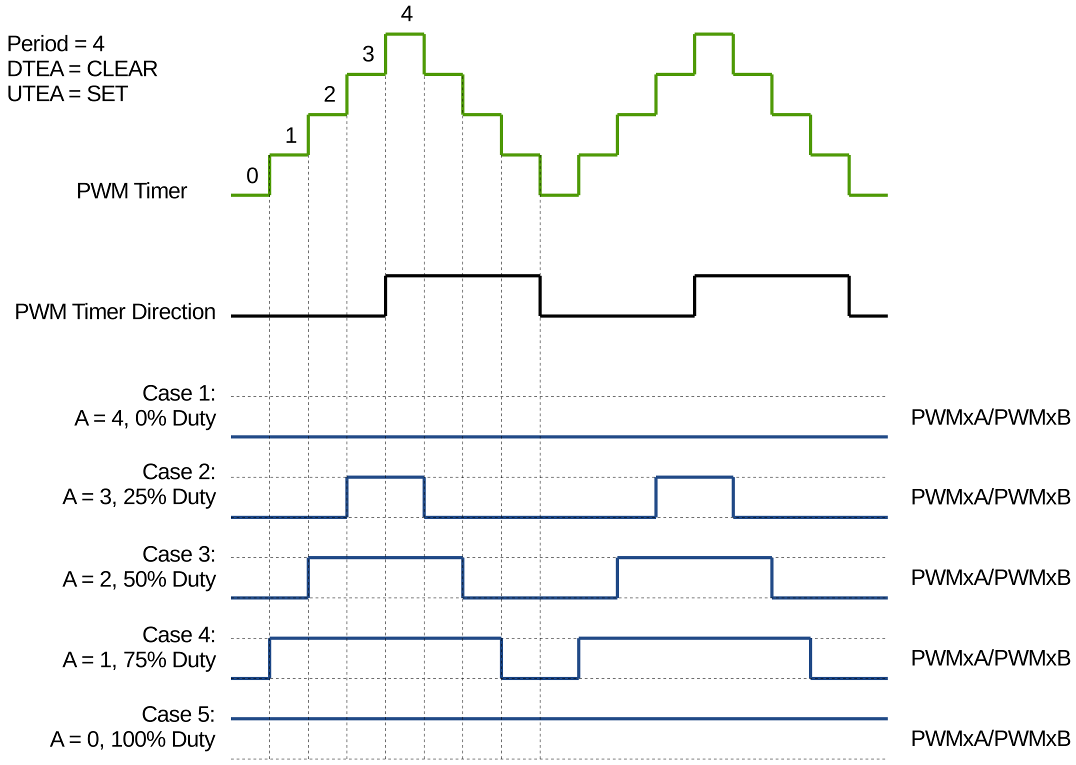


图 16-15. 递增递减模式下的对称波形

图 16-16 至图 16-19 的 PWM 波形描述了常见的 PWM 操作器配置。图中数据说明如下：

- Period，A 和 B 分别表示写入周期寄存器，寄存器 A 和 B 的值。
- PWMxA 和 PWMxB 是 PWM 操作器 x 的输出信号。

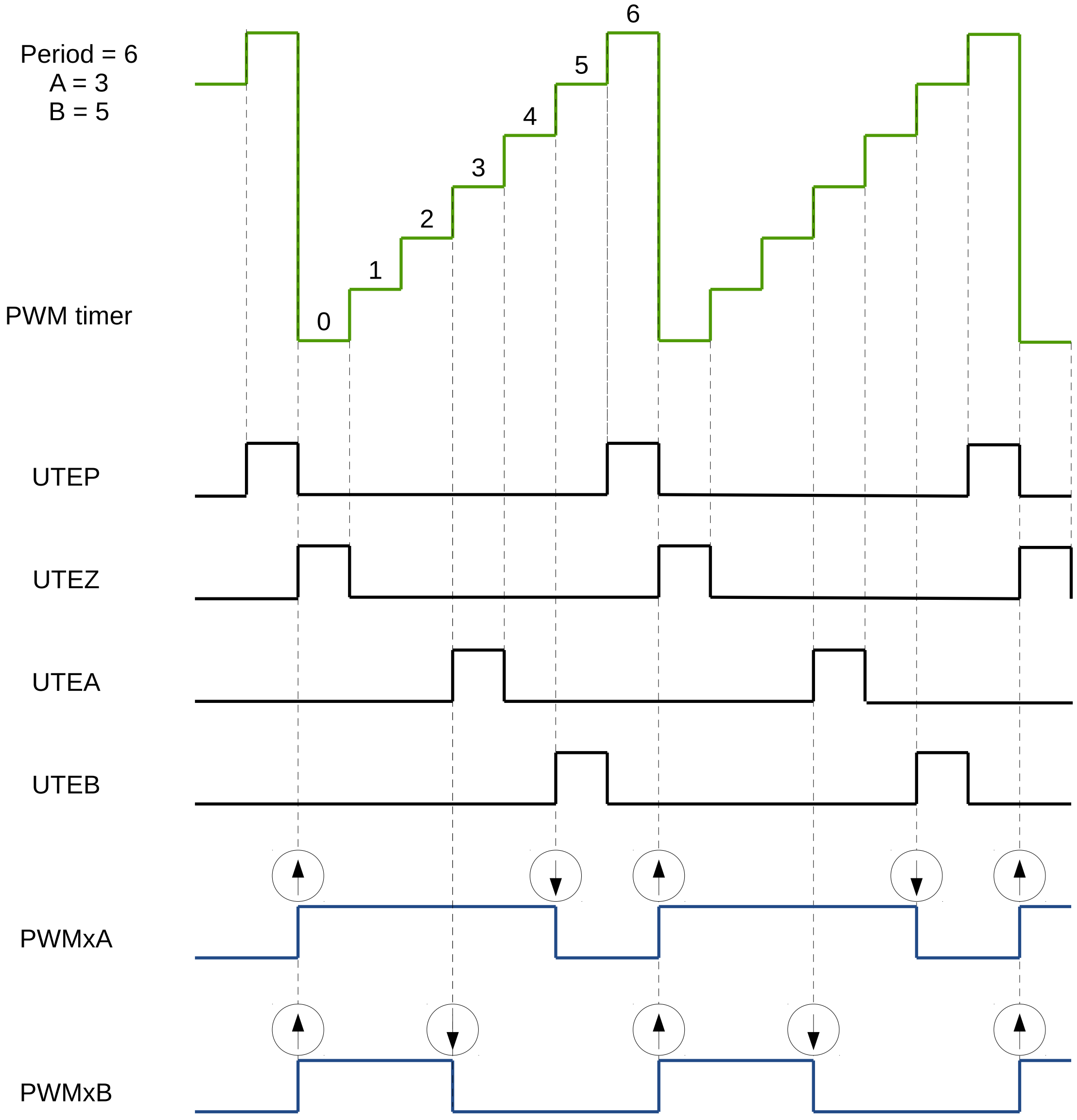


图 16-16. 递增计数模式，单边不对称波形，PWMxA 和 PWMxB 独立调制–高电平

PWMxA 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWMxB 的占空比调制由 A 设置，高电平有效，与 A 成正比。

$$Period = (PWM\_TIMERx\_PERIOD + 1) \times T_{PT\_clk}$$

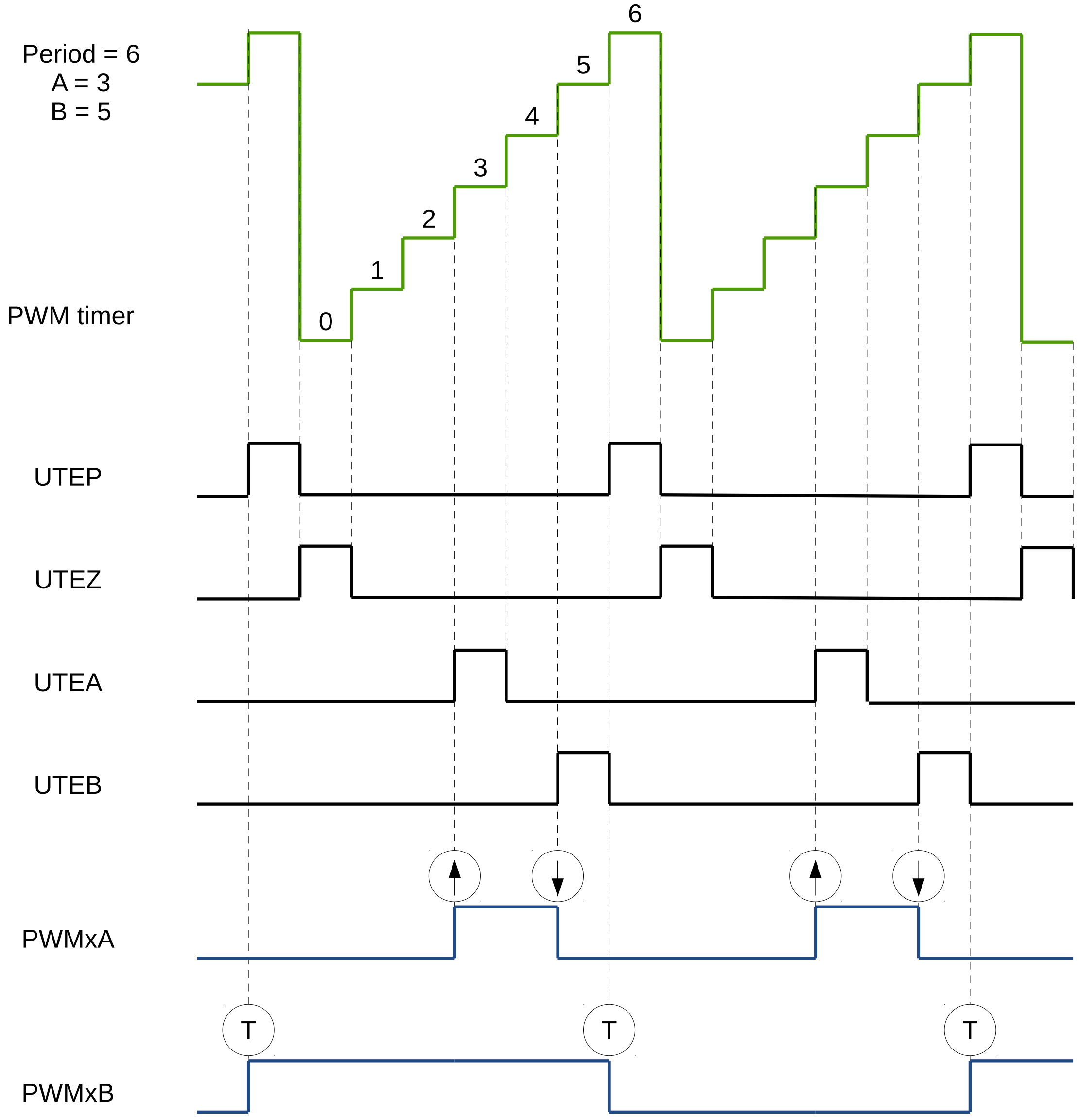


图 16-17. 递增计数模式，脉冲位置不对称波形，PWMxA 独立调制

脉冲可以在 PWM 波形内（零至周期值之间）的任何地方生成。
PWMxA 高电平占空比与 (B –A) 成正比。

$$Period = (PWM\_TIMERx\_PERIOD + 1) \times T_{PT\_clk}$$

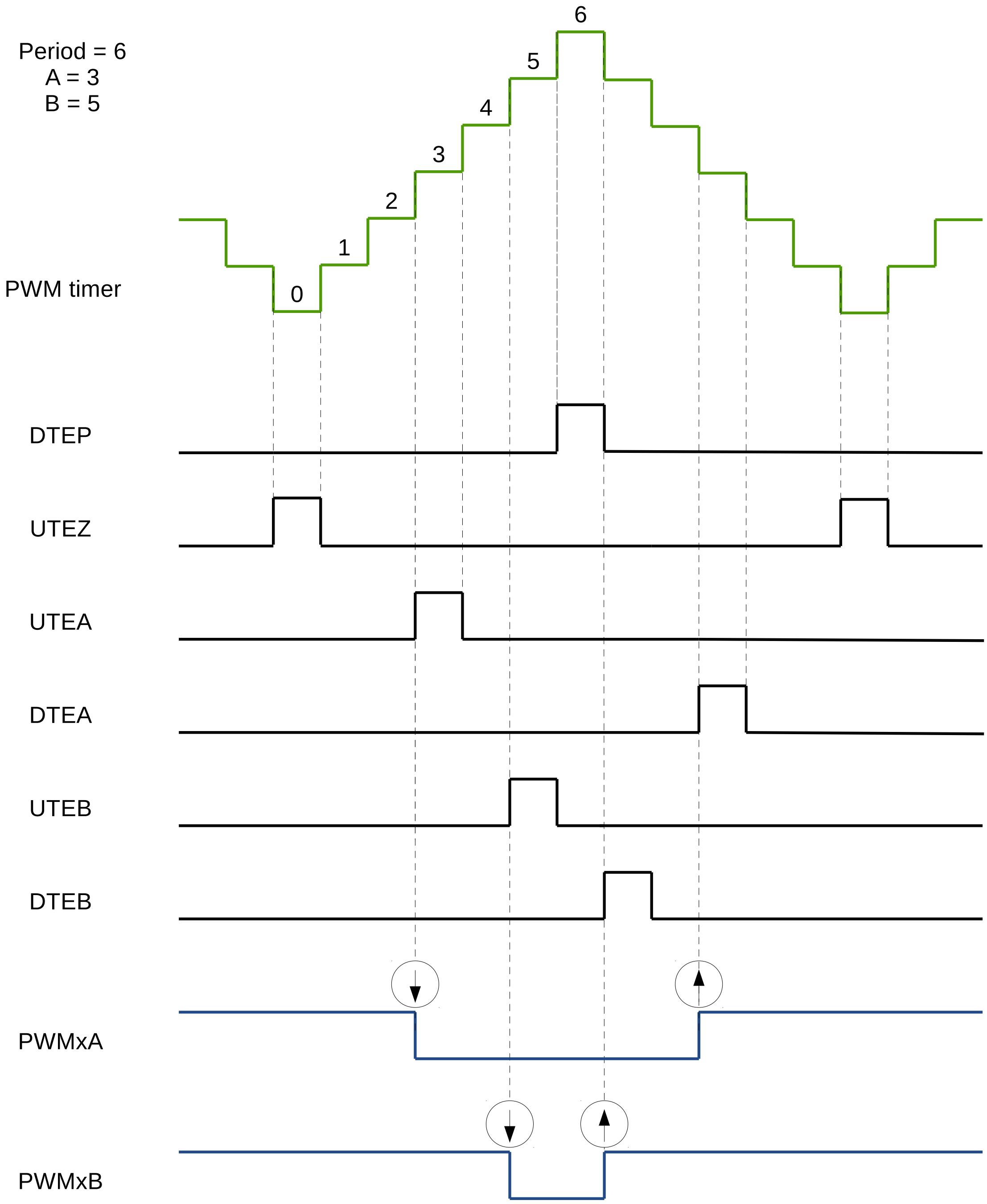


图 16-18. 递增递减循环计数模式，双沿对称波形，在 PWMxA 和 PWMxB 上独立调制–高电平有效

PWMxA 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWMxB 的占空比调制由 B 设置，高电平有效，与 B 成正比。
输出 PWMxA 和 PWMxB 可驱动不同开关。

$$Period = (2 \times PWM\_TIMERx\_PERIOD + 1) \times T_{PT\_clk}$$

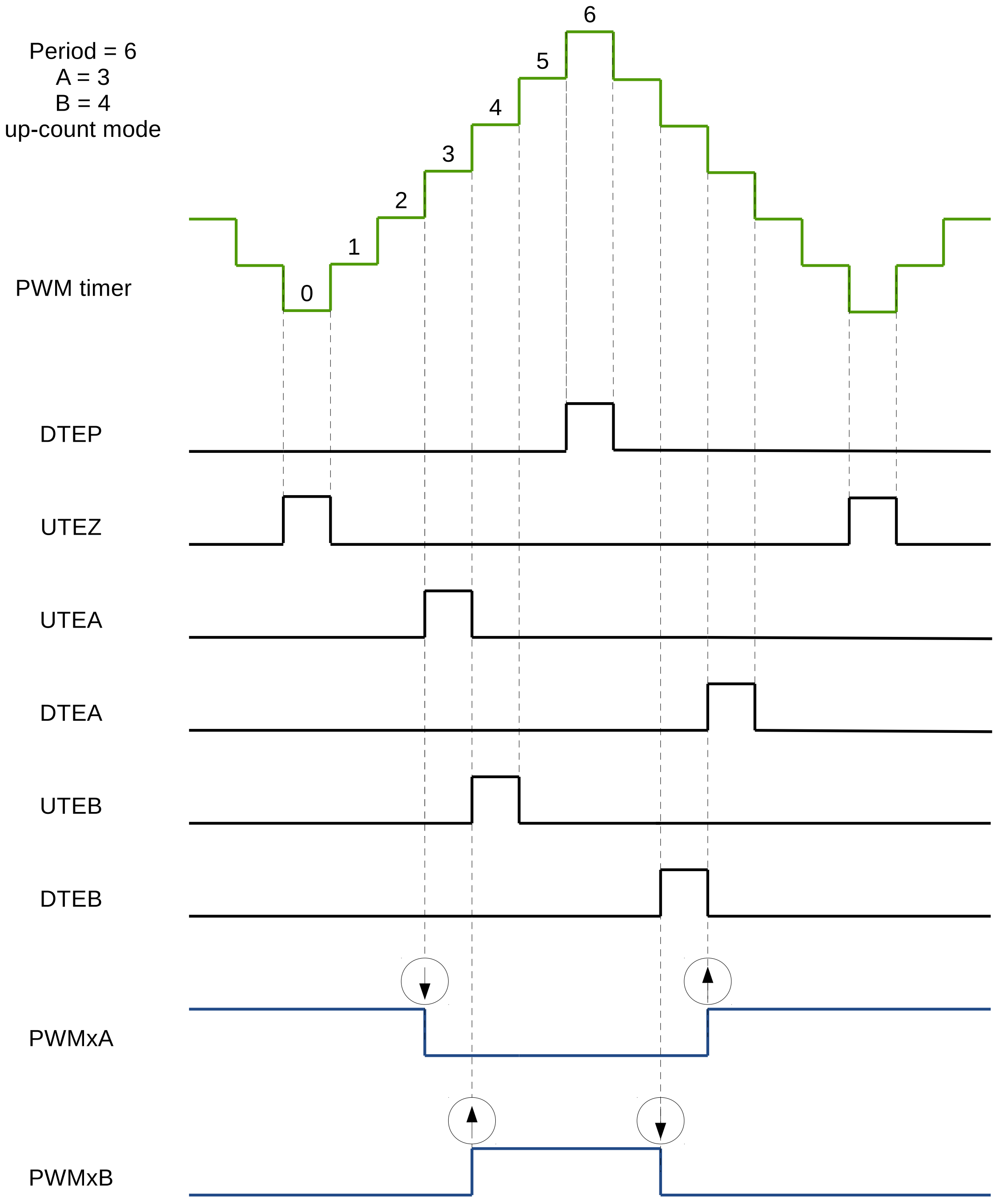


图 16-19. 递增递减循环计数模式，双沿对称波形，在 PWMxA 和 PWMxB 上独立调制–互补

PWMxA 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWMxB 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWMxA/B 输出可驱动上/下（互补）开关。
死区 = B –A，边沿位置完全可由软件配置。必要时，可使用死区生成器模块设置其他边沿延迟方式。

$$Period = (2 \times PWM\_TIMERx\_PERIOD + 1) \times T_{PT\_clk}$$

软件强制事件

在 PWM 生成器内有 2 种软件强制事件：

- 非连续即时 (NCI) 软件强制事件
  当由软件触发时，这些类型的事件在 PWM 输出上立即生效。并且强制是不连续的，这意味着下一个激活的定时事件能够改变 PWM 输出。
- 连续 (CNTU) 软件强制事件
  这一类型事件是连续的。直到通过软件释放，强制 PWM 持续输出。事件触发器可配置。这一类事件可配置为定时或即时发生。

图 16-20 为 NCI 软件强制事件的一种波形。NCI 用于单独强制 PWMxA 输出为低电平，PWMxB 不受强制。

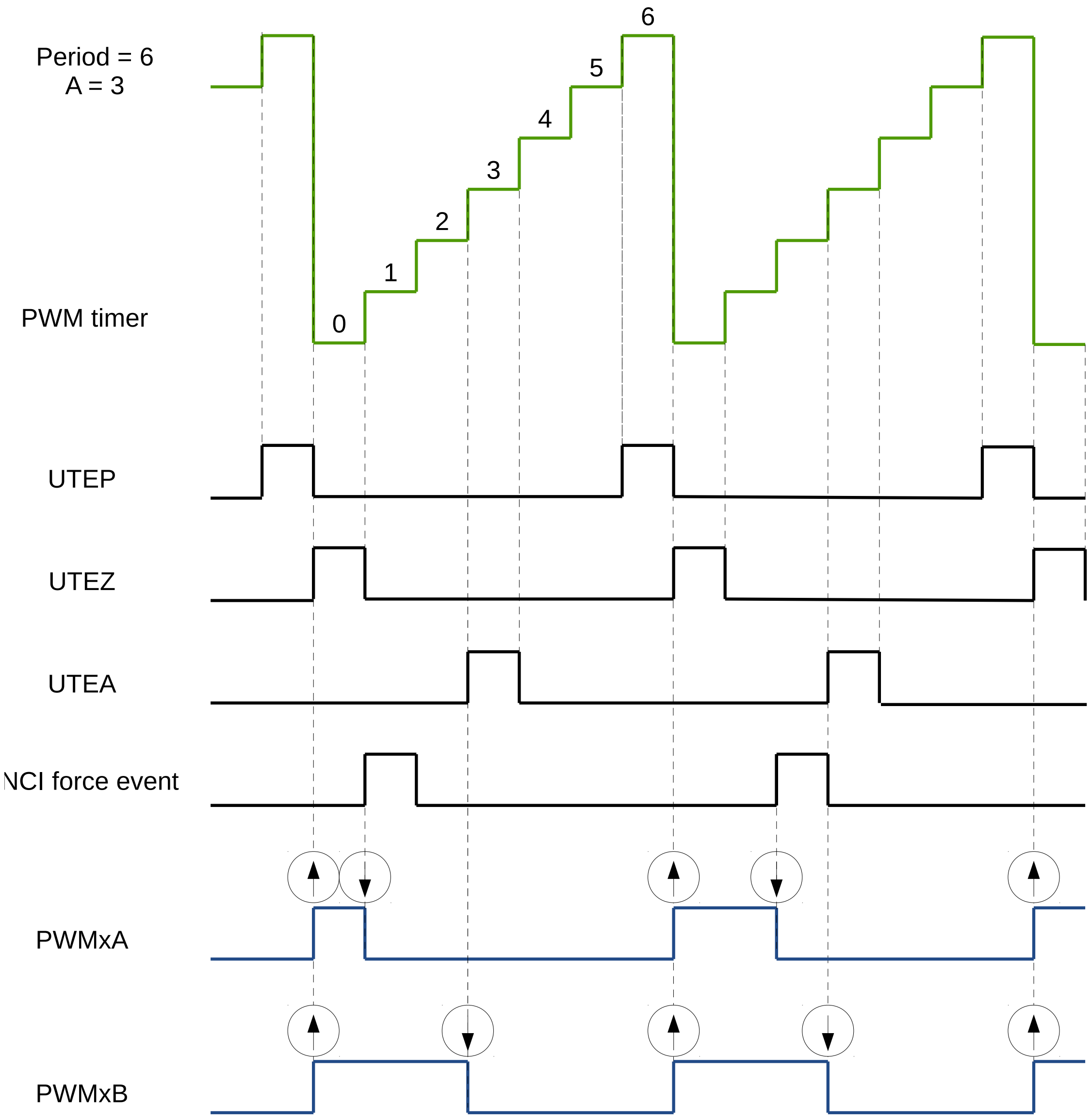


图 16-20. NCI 在 PWMxA 输出上软件强制事件示例

图 16-21 为 CNTU 软件强制事件的波形。UTEZ 事件被选为 CNTU 软件强制事件的触发器。CNTU 用于单独强制 PWMxB 输出为低电平，但 PWMxA 不受强制。

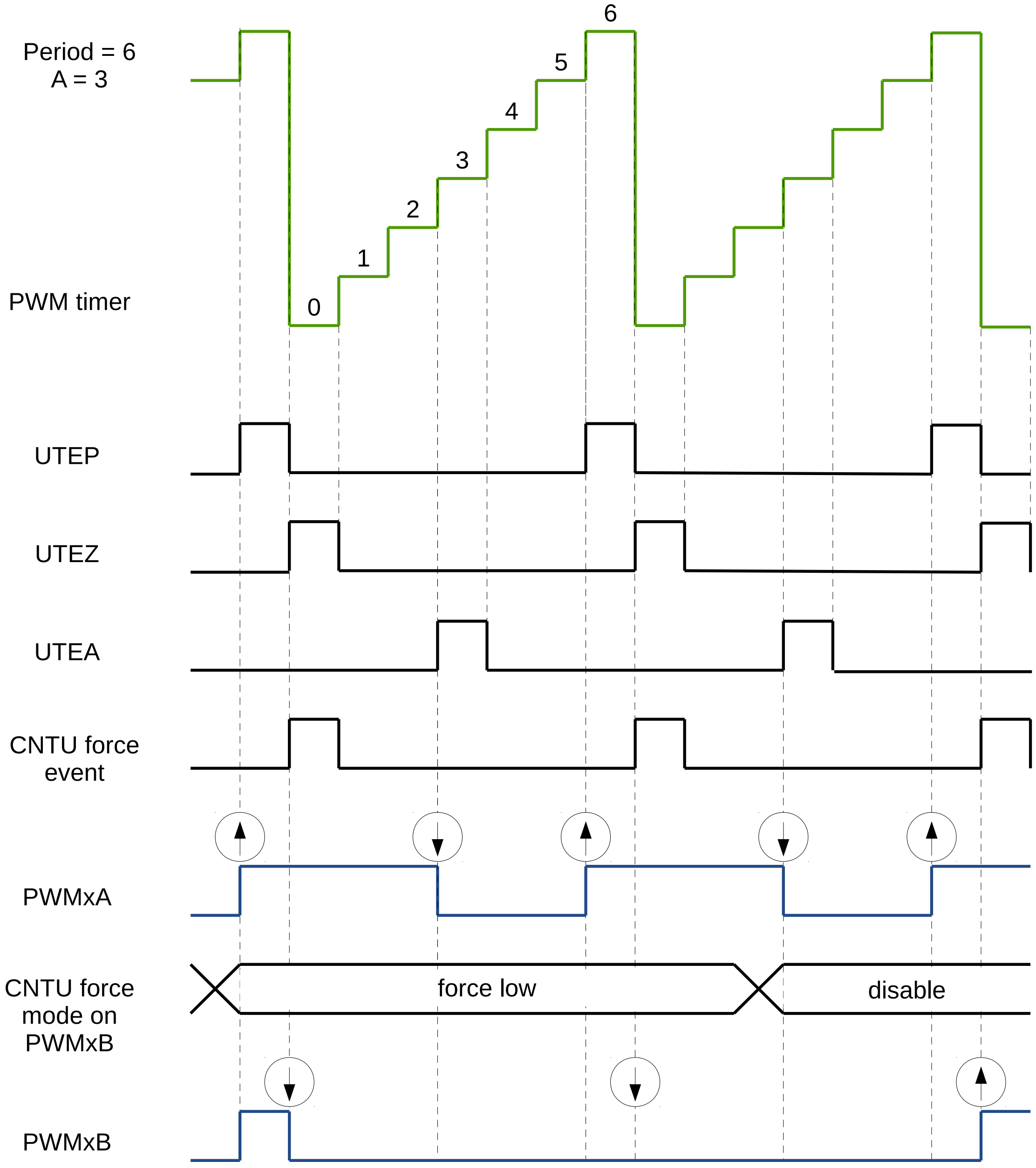


图 16-21. CNTU 在 PWMxB 输出上软件强制事件示例

16.3.3.2 死区生成器模块

死区生成器模块作用

章节 PWM 生成器模块讲述了在 PWMxA 和 PWMxB 上输出特定边沿位置的信号的几个配置选项。通过改变信号之间的边沿位置以及设置信号的占空比，可获得所需的死区。另一种方式是使用专门的死区生成器模块控制死区。

死区生成器模块的主要功能如下：

- 根据单个 PWMxA 输入的死区生成信号对（PWMxA 和 PWMxB）
- 通过在信号边沿增加延迟来生成死区：
  - 上升沿延迟 (RED)
  - 下降沿延迟 (FED)
- 配置信号对：
  - 高电平有效互补 (AHC)
  - 低电平有效互补 (ALC)
  - 高电平有效 (AH)
  - 低电平有效 (AL)
- 如果死区直接在生成器模块中配置，则死区发生器不生效。

死区模块生成器影子寄存器

延迟寄存器 RED 和 FED 的影子寄存器为 PWM_DTx_RED_CFG_REG 和 PWM_DTx_FED_CFG_REG。寄存器描述详见 16.3.2.3。

死区生成器模块的操作要点

图 16-22 描述了创建死区模块的开关拓扑。

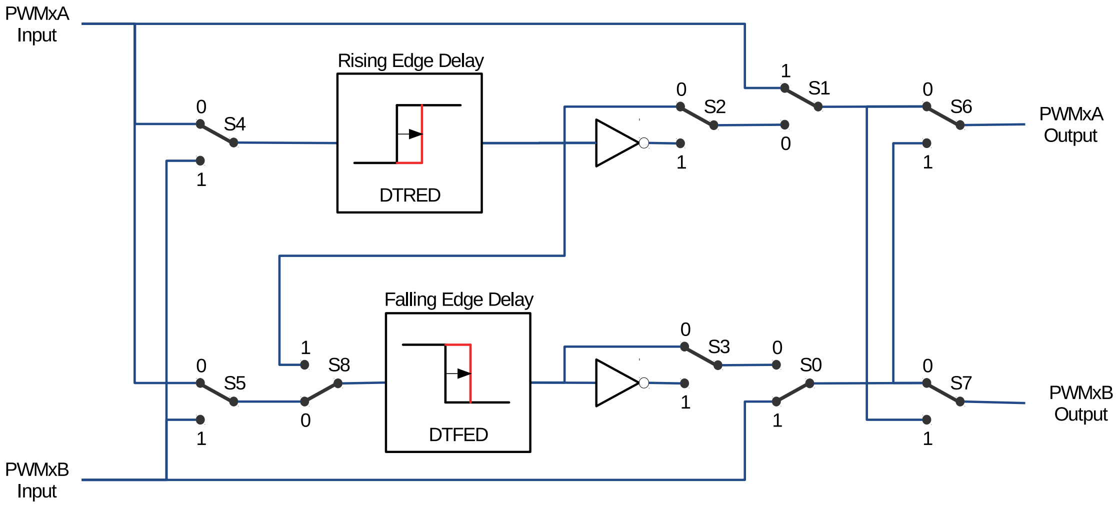


图 16-22. 死区模块的开关拓扑

上图中的 S0 - S8 是由表 16-5 中的 PWM_DTx_CFG_REG 寄存器控制的开关。

表 16-5. 控制死区时间生成器开关的寄存器

开关	寄存器
S0	PWM_DTx_B_OUTBYPASS
S1	PWM_DTx_A_OUTBYPASS
S2	PWM_DTx_RED_OUTINVERT
S3	PWM_DTx_FED_OUTINVERT
S4	PWM_DTx_RED_INSEL
S5	PWM_DTx_FED_INSEL
S6	PWM_DTx_A_OUTSWAP
S7	PWM_DTx_B_OUTSWAP
S8	PWM_DTx_DEB_MODE

支持所有开关组合，但不是所有的开关模式都是典型的使用模式。表 16-6 列举了一些典型的死区配置。在这些配置中，S4 和 S5 的开关位置将 PWMxA 设置为下降沿和上升沿延迟的公共源。表 16-6 中的模式可分为以下几类：

- **模式 1：绕过下降沿 (FED) 和上升沿 (RED) 的延迟**
  在该模式下，死区模块被关闭。PWMxA 和 PWMxB 信号的波形无变化。
- **模式 2-5：经典死区极性设置**
  这些模式为典型极性配置，涵盖工业电源栅极驱动器中的高 / 低电平有效模式。图 16-23 至 16-26为典型波形。
- **模式 6 和 7：绕过下降沿 (FED) 或上升沿 (RED) 的延迟**
  此模式下，绕过上升沿延迟 (RED) 或下降沿延迟 (FED) 。因此，不使用对应延迟。

表 16-6. 死区生成器的典型操作模式

模式	描述	S0	S1	S2	S3
1	PWMxA 和 PWMxB 波形无变化	1	1	X	X
2	高电平有效互补 (AHC)，参见图 16-23	0	0	0	1
3	低电平有效互补 (ALC)，参见图 16-24	0	0	1	0
4	高电平有效 (AH)，参见图 16-25	0	0	0	0
5	低电平有效 (AL)，参见图 16-26	0	0	1	1
6	PWMxA 输出 = PWMxA 输入（无延迟） PWMxB 输出 = PWMxA 输入，下降沿延迟	0	1	0 或 1	0 或 1
7	PWMxA 输出 = PWMxA 输入，上升沿延迟 PWMxB 输出 = PWMxB 输入（无延迟）	1	0	0 或 1	0 或 1
说明：以上所有模式中，S4 - S8 的开关位置都置 0。					

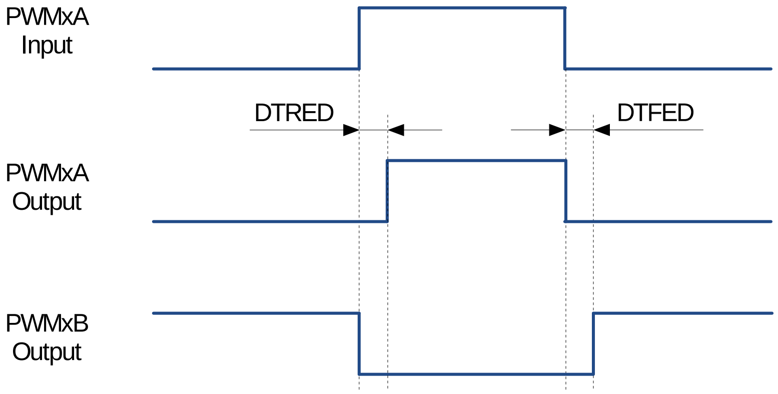


图 16-23. 高电平有效互补 (AHC) 死区波形

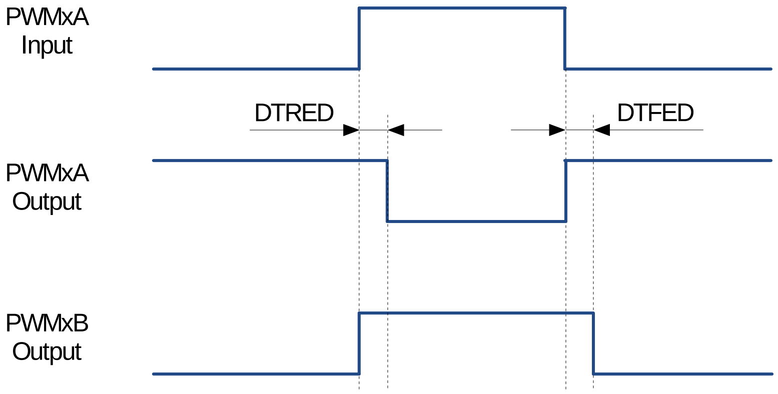


图 16-24. 低电平有效互补 (ALC) 死区波形

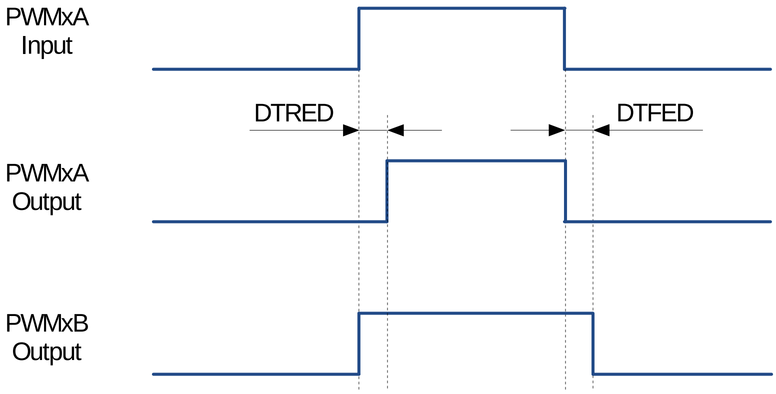


图 16-25. 高电平有效 (AH) 死区波形

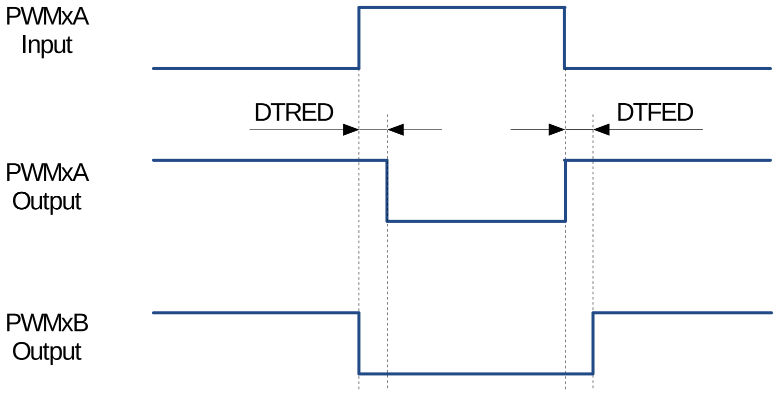


图 16-26. 低电平有效 (AL) 死区波形

上升沿延迟 (RED) 和下降沿延迟 (FED) 可分别设置。延迟的值通过 16 位寄存器 PWM_DTx_RED 和 PWM_DTx_FED 配置。寄存器值表示一个信号边沿可以延迟的 DT_clk 时钟周期值。DT_clk 可通过寄存器 PWM_DTx_CLK_SEL 从 PWM_clk 或 PT_clk 中选择。

通过以下公式计算下降沿延迟 (FED) 和上升沿延迟 (RED) 的值:

$$FED = PWM\_DTx\_FED \times T_{DT\_clk}$$

$$RED = PWM\_DTx\_RED \times T_{DT\_clk}$$

16.3.3.3 PWM 载波模块

将 PWM 输出耦合到电机驱动器可能需要使用变压器隔离。变压器只提供交流信号，而 PWM 信号的占空比可能在 0% 到 100% 之间变化。PWM 载波模块可以通过使用高频载波对其进行调制，将该信号传递给变压器。

功能概述

此模块的以下关键功能可配置:

- 载波频率
- 第一个脉冲的脉宽
- 第二个以及之后的脉冲的占空比
- 开启/关闭载波

操作要点

PWM 载波时钟 (PC_clk) 来自于 PWM_clk。通过寄存器 PWM_CARRIERx_CFG_REG 的 PWM_CARRIERx_PRESCALE 和 PWM_CARRIERx_DUTY 位配置频率和占空比。一次性脉冲的功能在于提供高能量脉冲以接通电源开关。随后的脉冲用于保持上电的状态。一次性脉冲宽度可通过 PWM_CARRIERx_OSHTWTH 位进行配置。通过 PWM_CARRIERx_EN 位来使能 / 禁止载波模块。

载波示例

图 16-27 描述了载波叠加在原始 PWM 脉冲上的示例波形。该图不显示第一个脉冲和占空比控制，相关详细信息将在后两节中介绍。

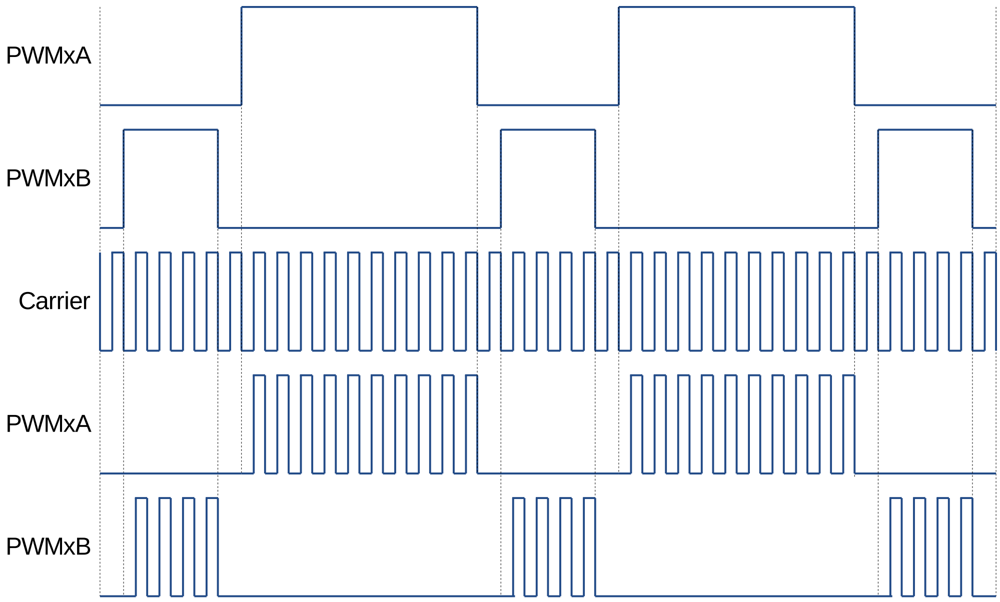


图 16-27. PWM 载波操作的波形示例

第一个脉冲

第一个脉冲的宽度可配置，其值有 16 种可能，可通过下公式计算：

$$T_{1stpulse} = T_{PWM\_clk} \times 8 \times (PWM\_CARRIER\text{x}\_PRESCALE + 1) \times (PWM\_CARRIER\text{x}\_OSHTWTH + 1)$$

其中：

- T_{PMW_clk} 为 PWM 时钟周期 (PWM_clk)
- $(PWM\_CARRIER\text{x}\_OSHTWTH + 1)$ 为一次性脉冲宽度值（取值范围：1-16）
- $(PWM\_CARRIER\text{x}\_PRESCALE + 1)$ PWM 载波时钟 (PC_clk) 预分频值

图 16-28 展示了第一个脉冲和之后持续的脉冲。

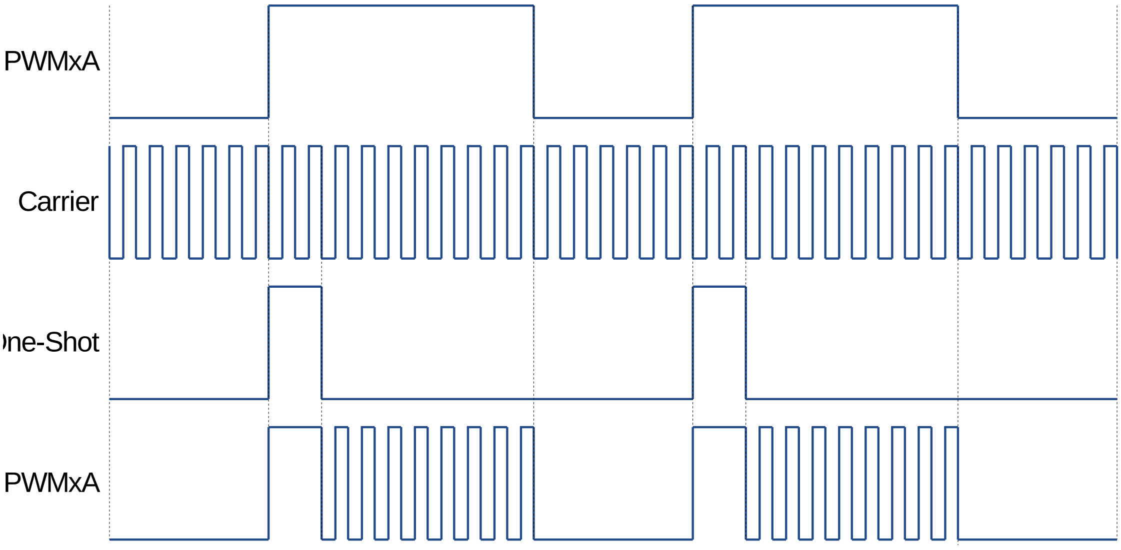


图 16-28. 载波模块的第一个脉冲和之后持续的脉冲示例

占空比控制

在发出第一个一次性脉冲之后，根据载波频率调制剩余的 PWM 信号。用户可配置该信号的占空比。在一定情况下，调整占空比可使信号通过隔离变压器后仍然可以开启或关闭电动机驱动器，改变电机旋转速度和方向。

占空比通过寄存器 PWM_CARRIERx_CFG_REG 的 PWM_CARRIERx_DUTY 第 5 到 7 位设置，其值有 7 种可能性。

占空比的值可通过以下方式计算：

$$Duty = PWM\_CARRIERx\_DUTY \div 8$$

图 16-29 为所有 7 种占空比设置。

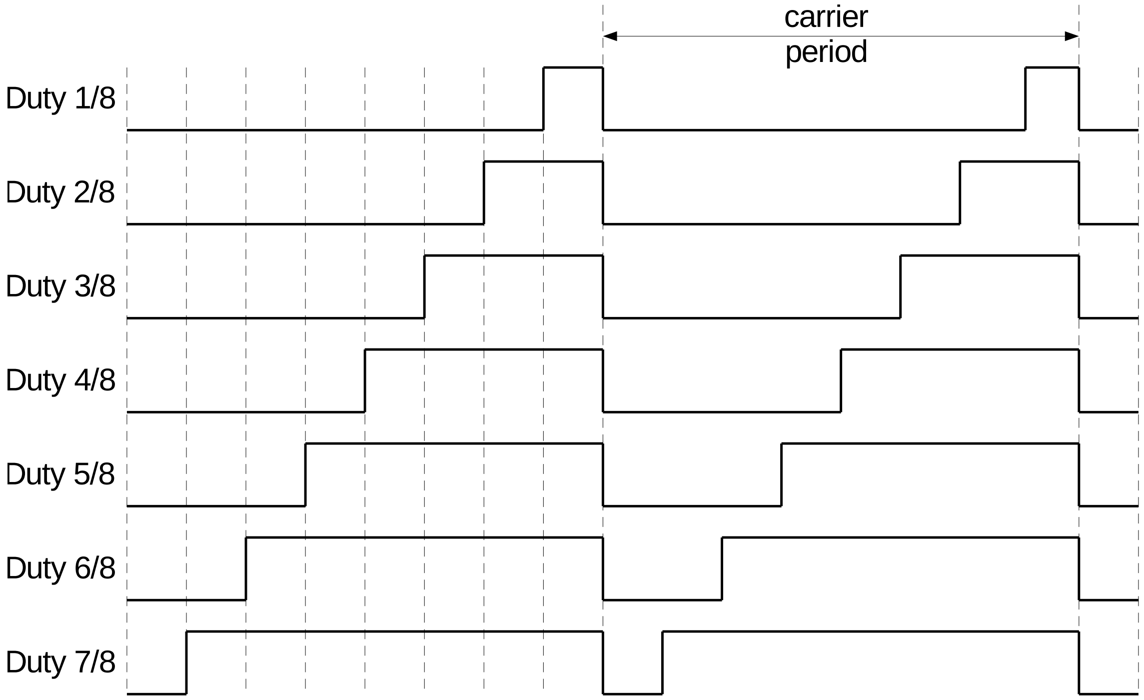


图 16-29. PWM 载波模块中持续脉冲的 7 种占空比设置

16.3.3.4 故障处理器模块

每个 MCPWM 外设都连接来自 GPIO 矩阵的 3 个故障信号（FAULT0，FAULT1 和 FAULT2）。这些信号用于指示外部故障状况，并且可由故障检测模块预处理后生成故障事件。故障事件通过执行用户代码，针对特定故障调整 MCPWM 输出。

故障处理模块功能

故障处理模块的主要功能为：

- 在检测到故障时强制 PWMxA 和 PWMxB 输出信号进入以下状态之一：
  - 高
  - 低
  - 取反
  - 无
- 在检测到过电流过载 / 短路时执行一次性跳闸（OST）。
- 逐周期跳闸（CBC）以提供限流操作。
- 每个故障信号单独分配一次性或逐周期操作。
- 每个故障输入都生成中断。
- 支持软件强制跳闸。
- 根据需要开启或关闭模块功能。

操作与配置要点

本节提供故障处理模块的操作要点和配置选项。

来自管脚的故障信号在 GPIO 矩阵中采样和同步。为了保证故障脉冲采样的成功，每个脉冲持续时间必须至少为 2 个 APB 时钟周期。故障检测模块使用 PWM_clk 对故障信号进行采样，因此来自 GPIO 矩阵的故障脉冲持续时间必须至少为 1 个 PWM_clk 周期。因此，无论 APB 时钟周期和 PWM_clk 周期的大小关系如何，管脚上的故障信号脉冲的宽度必须至少等于两个 APB 时钟周期与一个 PWM_clk 周期的和。

故障处理器模块可以使用故障信号 FAULT0 至 FAULT2 中的高电平或低电平来生成故障事件 fault_event0 至 fault_event2。每个故障事件可以单独配置为进行 CBC 操作，OST 操作或无操作。

- **逐周期 (CBC) 操作：**
  当 CBC 操作被触发，PWM*x*A 和 PWM*x*B 的状态立即根据寄存器 PWM_FH*x*_A_CBC_U/D 和 PWM_FH*x*_B_CBC_U/D 的设置改变。PWM 定时器递增或递减计数时，可指定不同的操作。不同的故障事件可触发不同的逐周期操作中断。通过状态寄存器 PWM_FH*x*_CBC_ON 开启或关闭 CBC 操作。在没有故障事件时，将在指定时间点，即发生 D/UTEP 或 D/UTEZ 事件时清除 PWM*x*A/B 上的 CBC 操作。寄存器 PWM_FH*x*_CBCPULSE 控制决定 PWM*x*A 和 PWM*x*B 恢复正常的事件。因此，在此模式下，CBC 操作在每个 PWM 循环后清除或刷新。

- **一次性（OST）操作：**
  当 OST 操作被触发时，PWM*x*A 和 PWM*x*B 的状态立即根据寄存器 PWM_FH*x*_A_OST_U/D 和 PWM_FH*x*_B_OST_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 OST 操作中断。通过状态寄存器 PWM_FH*x*_OST_ON 开启或关闭 OST 操作。PWM*x*A/B 上的 OST 操作将在没有故障事件时不能自动清除。一次性操作必需人为通过将寄存器 PWM_FH*x*_CLR_OST 的值取反来清除。

16.3.4 捕获模块

16.3.4.1 介绍

捕获模块包含 3 个完整的捕获通道。通道输入信号 CAP0，CAP1 和 CAP2 来自于 GPIO 矩阵。由于 GPIO 矩阵的灵活性，CAP0，CAP1 和 CAP2 可以通过任一管脚输入配置。多个捕获通道可以来自同一管脚输入，而每个通道的预分频可以分别设置。此外，每个捕获通道还可以来自不同的管脚输入。因此，可以通过后台硬件用多种方式处理捕获信号，而不直接由 CPU 处理。

每个模块都有以下独立资源：

- 一个 32 位定时器（计数器），可与 PWM 定时器，另一个模块或软件同步。
- 3 个捕获通道，每个通道配有一个 32 位时间戳和一个捕获预分频器。
- 任何捕获通道的边沿极性（上升/下降沿）可独立选择。
- 输入捕获信号预分频（分频取值范围：1 –256）。
- 三个捕获事件都有中断功能。

16.3.4.2 捕获定时器

捕获定时器是一个 32 位计数器，使能时不断递增计数。输入端的 APB 时钟频率通常为 80 MHz。发生同步事件时，加载计数器，其相位存储在寄存器 PWM_CAP_TIMER_PHASE_REG 中。同步事件可来自 PWM 定时器同步输出，PWM 模块同步输入，或软件。该捕获定时器为所有 3 个捕获通道提供定时参考。

16.3.4.3 捕获通道

必要时，到达捕获通道的捕获信号可先被反相，然后预分频。最后，预处理后的捕获信号的指定边沿将触发捕获事件。在捕获事件发生时，捕获定时器的值存储在时间戳寄存器 PWM_CAP_CHx_REG 中。捕获事件中的不同捕获通道可生成不同的中断。触发捕获事件的边沿储存在寄存器 PWM_CAPx_EDGE 中。捕获事件可由软件强制发生。

16.4 寄存器列表

寄存器列表	描述	PWM0	PWM1	访问
预分频器配置				
PWM_CLK_CFG_REG	配置预分频器	0x3FF5E000	0x3FF6C000	读/写
PWM 定时器 0 配置与状态				
PWM_TIMER0_CFG0_REG	定时器周期与更新方法	0x3FF5E004	0x3FF6C004	读/写
PWM_TIMER0_CFG1_REG	工作模式与开始 / 停止控制	0x3FF5E008	0x3FF6C008	读/写
PWM_TIMER0_SYNC_REG	同步设置	0x3FF5E00C	0x3FF6C00C	读/写
PWM_TIMER0_STATUS_REG	定时器状态	0x3FF5E010	0x3FF6C010	只读
PWM 定时器 1 配置与状态				
PWM_TIMER1_CFG0_REG	定时器更新方式与周期	0x3FF5E014	0x3FF6C014	读/写
PWM_TIMER1_CFG1_REG	工作模式与开始 / 停止控制	0x3FF5E018	0x3FF6C018	读/写
PWM_TIMER1_SYNC_REG	同步设置	0x3FF5E01C	0x3FF6C01C	读/写
PWM_TIMER1_STATUS_REG	定时器状态	0x3FF5E020	0x3FF6C020	只读
PWM 定时器 2 配置与状态				
PWM_TIMER2_CFG0_REG	定时器更新与状态	0x3FF5E024	0x3FF6C024	读/写
PWM_TIMER2_CFG1_REG	工作模式与开始 / 停止控制	0x3FF5E028	0x3FF6C028	读/写
PWM_TIMER2_SYNC_REG	同步设置	0x3FF5E02C	0x3FF6C02C	读/写
PWM_TIMER2_STATUS_REG	定时器状态	0x3FF5E030	0x3FF6C030	只读
PWM 定时器常见配置				
PWM_TIMER_SYNCI_CFG_REG	定时器同步输入选择	0x3FF5E034	0x3FF6C034	读/写
PWM_OPERATOR_TIMERSEL_REG	为 PWM 操作器选择特定的计时器	0x3FF5E038	0x3FF6C038	读/写
PWM 操作器 0 配置与状态				
PWM_GEN0_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x3FF5E03C	0x3FF6C03C	读/写
PWM_GEN0_TSTMP_A_REG	寄存器 A 的影子寄存器	0x3FF5E040	0x3FF6C040	读/写
PWM_GEN0_TSTMP_B_REG	寄存器 B 的影子寄存器	0x3FF5E044	0x3FF6C044	读/写
PWM_GEN0_CFG0_REG	故障时间 T0 和 T1 处理	0x3FF5E048	0x3FF6C048	读/写
PWM_GEN0_FORCE_REG	软件强制 PWM0A 和 PWM0B 输出	0x3FF5E04C	0x3FF6C04C	读/写
PWM_GEN0_A_REG	PWM0A 输出上事件触发的操作	0x3FF5E050	0x3FF6C050	读/写
PWM_GEN0_B_REG	PWM0B 输出上事件触发的操作	0x3FF5E054	0x3FF6C054	读/写
PWM_DT0_CFG_REG	死区与类型的选择与配置	0x3FF5E058	0x3FF6C058	读/写
PWM_DT0_FED_CFG_REG	FED 的影子寄存器	0x3FF5E05C	0x3FF6C05C	读/写
PWM_DT0_RED_CFG_REG	RED 的影子寄存器	0x3FF5E060	0x3FF6C060	读/写
PWM_CARRIER0_CFG_REG	载波使能与配置	0x3FF5E064	0x3FF6C064	读/写
PWM_FH0_CFG0_REG	故障事件中 PWM0A 和 PWM0B 上的操作	0x3FF5E068	0x3FF6C068	读/写

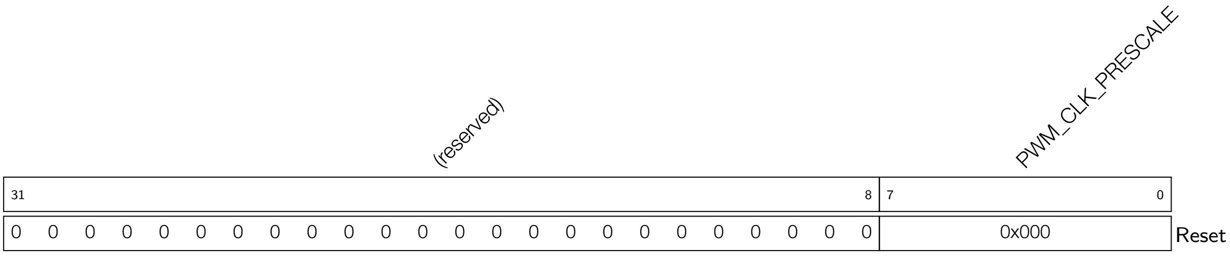
寄存器列表	描述	PWM0	PWM1	访问
PWM_FH0_CFG1_REG	故障处理的软件触发	0x3FF5E06C	0x3FF6C06C	读/写
PWM_FH0_STATUS_REG	故障事件状态	0x3FF5E070	0x3FF6C070	只读
PWM 操作器 1 配置与状态				
PWM_GEN1_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x3FF5E074	0x3FF6C074	读/写
PWM_GEN1_TSTMP_A_REG	寄存器 A 的影子寄存器	0x3FF5E078	0x3FF6C078	读/写
PWM_GEN1_TSTMP_B_REG	寄存器 B 的影子寄存器	0x3FF5E07C	0x3FF6C07C	读/写
PWM_GEN1_CFG0_REG	故障事件 T0 和 T1 处理	0x3FF5E080	0x3FF6C080	读/写
PWM_GEN1_FORCE_REG	软件强制 PWM1A 和 PWM1B 输出	0x3FF5E084	0x3FF6C084	读/写
PWM_GEN1_A_REG	PWM1A 输出上的事件触发的操作	0x3FF5E088	0x3FF6C088	读/写
PWM_GEN1_B_REG	PWM1B 输出上的事件触发的操作	0x3FF5E08C	0x3FF6C08C	读/写
PWM_DT1_CFG_REG	死区类型的选择与配置	0x3FF5E090	0x3FF6C090	读/写
PWM_DT1_FED_CFG_REG	FED 的影子寄存器	0x3FF5E094	0x3FF6C094	读/写
PWM_DT1_RED_CFG_REG	RED 的影子寄存器	0x3FF5E098	0x3FF6C098	读/写
PWM_CARRIER1_CFG_REG	使能与配置载波	0x3FF5E09C	0x3FF6C09C	读/写
PWM_FH1_CFG0_REG	故障事件中 PWM1A 和 PWM1B 输出上的操作	0x3FF5E0A0	0x3FF6C0A0	读/写
PWM_FH1_CFG1_REG	故障处理的软件触发	0x3FF5E0A4	0x3FF6C0A4	读/写
PWM_FH1_STATUS_REG	故障事件状态	0x3FF5E0A8	0x3FF6C0A8	只读
PWM 操作器 2 的配置与状态				
PWM_GEN2_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x3FF5E0AC	0x3FF6C0AC	读/写
PWM_GEN2_TSTMP_A_REG	寄存器 A 的影子寄存器	0x3FF5E0B0	0x3FF6C0B0	读/写
PWM_GEN2_TSTMP_B_REG	寄存器 B 的影子寄存器	0x3FF5E0B4	0x3FF6C0B4	读/写
PWM_GEN2_CFG0_REG	故障事件 T0 和 T1 处理	0x3FF5E080	0x3FF6C080	读/写
PWM_GEN2_FORCE_REG	软件强制 PWM2A 和 PWM2B 输出	0x3FF5E0BC	0x3FF6C0BC	读/写
PWM_GEN2_A_REG	PWM2A 输出上的事件触发的操作	0x3FF5E0C0	0x3FF6C0C0	读/写
PWM_GEN2_B_REG	PWM2B 输出上的事件触发的操作	0x3FF5E0C4	0x3FF6C0C4	读/写
PWM_DT2_CFG_REG	死区类型的选择与配置	0x3FF5E0C8	0x3FF6C0C8	读/写
PWM_DT2_FED_CFG_REG	FED 影子寄存器	0x3FF5E0CC	0x3FF6C0CC	读/写
PWM_DT2_RED_CFG_REG	RED 影子寄存器	0x3FF5E0D0	0x3FF6C0D0	读/写
PWM_CARRIER2_CFG_REG	使能与配置载波	0x3FF5E0D4	0x3FF6C0D4	读/写
PWM_FH2_CFG0_REG	故障事件中 PWM2A 和 PWM2B 输出上的操作	0x3FF5E0D8	0x3FF6C0D8	读/写
PWM_FH2_CFG1_REG	故障处理的软件触发	0x3FF5E0DC	0x3FF6C0DC	读/写
PWM_FH2_STATUS_REG	故障事件状态	0x3FF5E0E0	0x3FF6C0E0	只读
故障检测与配置				
PWM_FAULT_DETECT_REG	故障检测与配置	0x3FF5E0E4	0x3FF6C0E4	读/写
捕获配置与状态				
PWM_CAP_TIMER_CFG_REG	配置捕获定时器	0x3FF5E0E8	0x3FF6C0E8	读/写
PWM_CAP_TIMER_PHASE_REG	捕获定时器同步相位	0x3FF5E0EC	0x3FF6C0EC	读/写

寄存器列表	描述	PWM0	PWM1	访问
PWM_CAP_CH0_CFG_REG	捕获通道 0 的配置与使能	0x3FF5E0F0	0x3FF6C0F0	读/写
PWM_CAP_CH1_CFG_REG	捕获通道 1 的配置与使能	0x3FF5E0F4	0x3FF6C0F4	读/写
PWM_CAP_CH2_CFG_REG	捕获通道 2 的配置与使能	0x3FF5E0F8	0x3FF6C0F8	读/写
PWM_CAP_CH0_REG	捕获通道 0 上一次捕获的值	0x3FF5E0FC	0x3FF6C0FC	只读
PWM_CAP_CH1_REG	捕获通道 1 上一次捕获的值	0x3FF5E100	0x3FF6C100	只读
PWM_CAP_CH2_REG	捕获通道 2 上一次捕获的值	0x3FF5E104	0x3FF6C104	只读
PWM_CAP_STATUS_REG	上一次捕获触发器的边沿	0x3FF5E108	0x3FF6C108	只读
使能有效寄存器的更新				
PWM_UPDATE_CFG_REG	使能更新	0x3FF5E10C	0x3FF6C10C	读/写
管理中断				
INT_ENA_PWM_REG	中断使能位	0x3FF5E110	0x3FF6C110	读/写
INT_RAW_PWM_REG	原始中断状态	0x3FF5E114	0x3FF6C114	只读
INT_ST_PWM_REG	屏蔽中断状态	0x3FF5E118	0x3FF6C118	只读
INT_CLR_PWM_REG	中断清除位	0x3FF5E11C	0x3FF6C11C	WO

16.5 寄存器

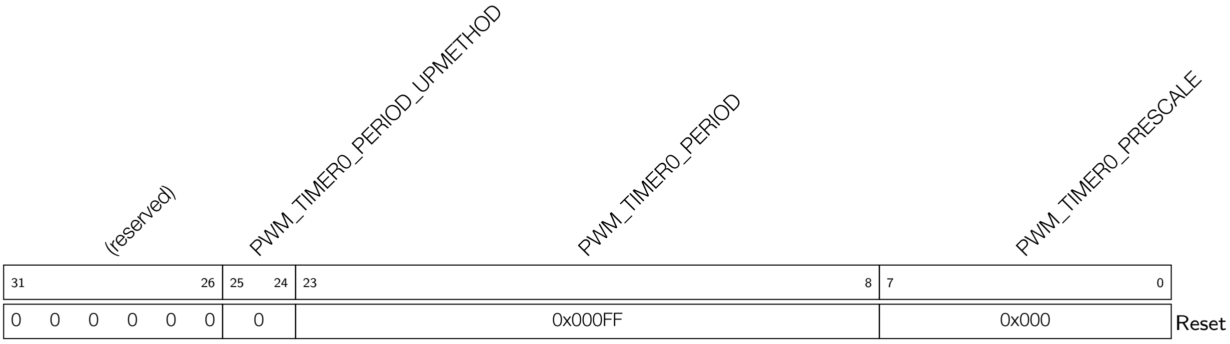
本小节括号中的地址均为相对于 MCPWM 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 16.4 寄存器列表。

Register 16.1. PWM_CLK_CFG_REG (0x0000)



PWM_CLK_PRESCALE PWM_clk 的周期 = 6.25 ns * (PWM_CLK_PRESCALE + 1)。(读 / 写)

Register 16.2. PWM_TIMER0_CFG0_REG (0x0004)



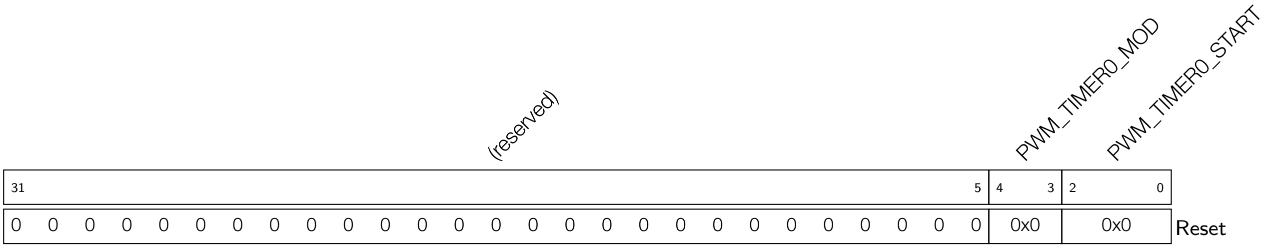
(reserved)	PWM_TIMER0_PERIOD_UPMETHOD	PWM_TIMER0_PERIOD	PWM_TIMER0_PRESCALE	
31 26	25 24	23 8	7 0	
0 0 0 0 0 0	0	0x000FF	0x000	Reset

PWM_TIMER0_PERIOD_UPMETHOD PWM 定时器 0 周期有效寄存器的更新方式。0：立即更新；1：发生 TEZ 事件时更新；2：同步时更新；3：发生 TEZ 事件或同步时更新。本文档中，TEZ 指定时器为 0 时的事件。（读 / 写）

PWM_TIMER0_PERIOD 计时器 0 的影子周期寄存器。（读 / 写）

PWM_TIMER0_PRESCALE PT0_clk 周期 = PWM_clk 周期 * (PWM_TIMER0_PRESCALE + 1)。（读 / 写）

Register 16.3. PWM_TIMER0_CFG1_REG (0x0008)



(reserved)	PWM_TIMER0_MOD	PWM_TIMER0_START	
31 5	4 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	Reset

PWM_TIMER0_MOD PWM 定时器 0 工作模式。0：暂停；1：递增模式；2：递减模式；3：递增递减循环模式。（读 / 写）

PWM_TIMER0_START 控制 PWM 定时器的开启与关闭。0：如果开启，在 TEZ 事件发生时停止；1：如果开启，在 TEP 事件发生时停止；2：开启；3：开启，并在下一个 TEZ 事件发生时停止；4：开启，并在下一个 TEP 事件发生时停止。本文档中，TEP 指定时器为周期值时发生的事件。（读 / 写）

Register 16.4. PWM_TIMER0_SYNC_REG (0x000c)

(reserved)	PWM_TIMER0_PHASE_DIRECTION	PWM_TIMER0_PHASE	PWM_TIMER0_SYNCO_SEL	PWM_TIMER0_SYNC_SW	PWM_TIMER0_SYNCI_EN	
31 21	20	19 4	3 2	1	0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	Reset

PWM_TIMER0_PHASE_DIRECTION 0：递增；1：递减。（读/写）

PWM_TIMER0_PHASE 同步事件中定时器重载的相位。（读 / 写）

PWM_TIMER0_SYNCO_SEL 选择 PWM 定时器 0 的同步输出来源。0：同步；1：TEZ；2：TEP；其他值：同步输出一直输出 0。（读 / 写）

PWM_TIMER0_SYNC_SW 此位取反，触发软件同步。（读 / 写）

PWM_TIMER0_SYNCI_EN 置 1 时，使能在同步输入事件发生时的定时器相位重载。（读 / 写）

Register 16.5. PWM_TIMER0_STATUS_REG (0x0010)

(reserved)	PWM_TIMER0_DIRECTION	PWM_TIMER0_VALUE	
31 17	16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	Reset

PWM_TIMER0_DIRECTION 当前 PWM 定时器 0 的计数器模式。0：递增模式；1：递减模式。（只读）

PWM_TIMER0_VALUE 当前 PWM 定时器 0 计数器的值。（只读）

Register 16.6. PWM_TIMER1_CFG0_REG (0x0014)

(reserved)	PWM_TIMER1_PERIOD_UPMETHOD	PWM_TIMER1_PERIOD	PWM_TIMER1_PRESCALE	
31 26	25 24	23 8	7 0	
0 0 0 0 0 0	0	0x000FF	0x000	Reset

PWM_TIMER1_PERIOD_UPMETHOD PWM 定时器 1 周期有效寄存器的更新方式。0：立即更新；1：发生 TEZ 事件时更新；2：同步时更新；3：发生 TEZ 事件或同步时更新。（读 / 写）

PWM_TIMER1_PERIOD 定时器 1 的影子周期寄存器。（读 / 写）

PWM_TIMER1_PRESCALE PT1_clk 周期 = PWM_clk 周期 * (PWM_TIMER1_PRESCALE + 1)。（读 / 写）

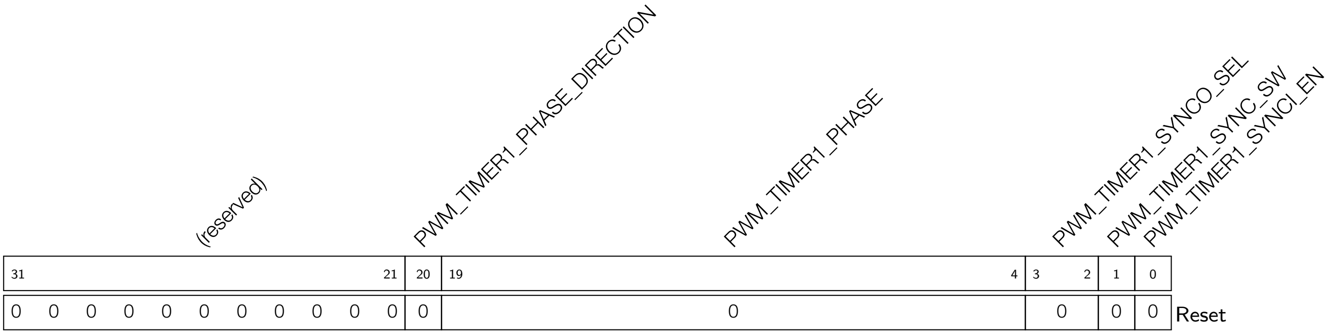
Register 16.7. PWM_TIMER1_CFG1_REG (0x0018)

(reserved)	PWM_TIMER1_MOD	PWM_TIMER1_START	
31 5	4 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	Reset

PWM_TIMER1_MOD PWM 计时器 1 的工作模式。0：暂停；1：递增模式；2：递减模式；3：递增递减循环模式。（读 / 写）

PWM_TIMER1_START PWM 控制定时器 1 的开启与停止。0：如果开启，在发生 TEZ 事件时停止；1：如果开启，在发生 TEP 事件时停止；2：开启；3：开启并在下一次 TEZ 事件中停止；4：开启并在下一次 TEP 事件中停止。（读 / 写）

Register 16.8. PWM_TIMER1_SYNC_REG (0x001c)



PWM_TIMER1_PHASE_DIRECTION 0：递增；1：递减。（读/写）

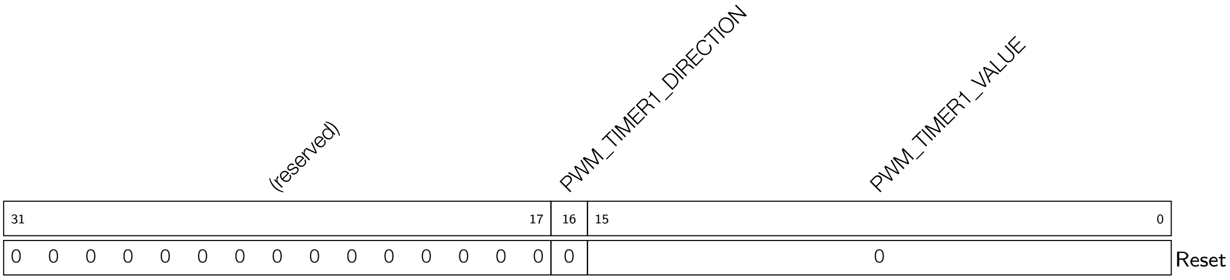
PWM_TIMER1_PHASE 同步时间中计时器重载的相位。（读 / 写）

PWM_TIMER1_SYNCO_SEL 选择 PWM 计时器 1 同步输出来源。0：同步输入；1：TEZ；2：TEP；其他值：同步输出一直输出 0。（读 / 写）

PWM_TIMER1_SYNC_SW 此位取反，触发软件同步事件。（读 / 写）

PWM_TIMER1_SYNCI_EN 置 1 时，使能在同步输入事件时的定时器相位重载。（读 / 写）

Register 16.9. PWM_TIMER1_STATUS_REG (0x0020)



PWM_TIMER1_DIRECTION 当前 PWM 计时器 1 的计数器模式。0：递增；1：递减。（只读）

PWM_TIMER1_VALUE 当前 PWM 计时器 1 的计数器值。（只读）

Register 16.10. PWM_TIMER2_CFG0_REG (0x0024)

(reserved)	PWM_TIMER2_PERIOD_UPMETHOD	PWM_TIMER2_PERIOD	PWM_TIMER2_PRESCALE	
31 26	25 24	23 8	7 0	
0 0 0 0 0 0	0	0x000FF	0x000	Reset

PWM_TIMER2_PERIOD_UPMETHOD PWM 定时器 2 周期有效寄存器的更新方式。0：立即更新；1：发生 TEZ 事件时更新；2：发生同步时间时更新；3：发生 TEZ 或同步事件时更新。（读 / 写）

PWM_TIMER2_PERIOD PWM 定时器 2 的影子周期寄存器。（读 / 写）

PWM_TIMER2_PRESCALE PT2_clk 周期 = PWM_clk 周期 * (PWM_TIMER2_PRESCALE + 1)。（读 / 写）

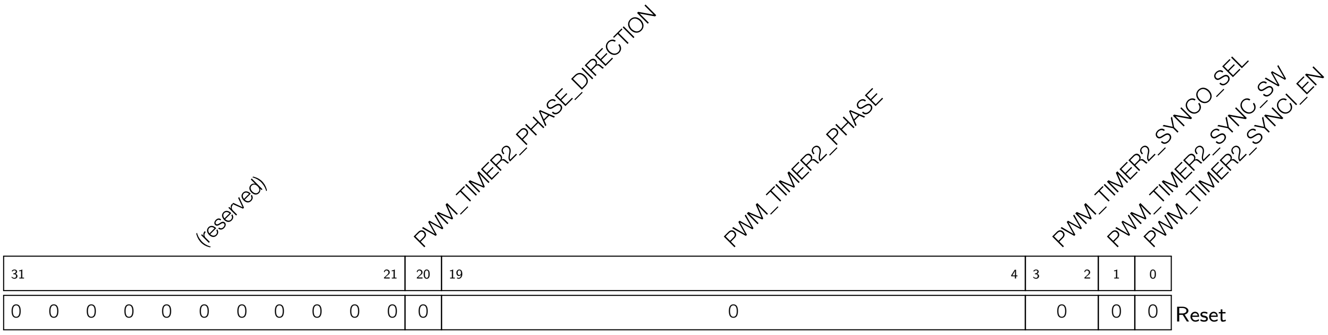
Register 16.11. PWM_TIMER2_CFG1_REG (0x0028)

(reserved)	PWM_TIMER2_MOD	PWM_TIMER2_START	
31 5	4 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	0x0	Reset

PWM_TIMER2_MOD PWM 定时器 2 的工作模式。0：暂停；1：递增模式；2：递减模式；3：递增递减循环模式。（读 / 写）

PWM_TIMER2_START 控制 PWM 定时器 2 的开启与停止。0：如果开启，在发生 TEZ 事件时停止；1：如果开启，在发生 TEP 事件时停止；2：开启；3：开启并在下一个 TEZ 事件时停止；4：开启并在下一个 TEZ 事件时停止。（读 / 写）

Register 16.12. PWM_TIMER2_SYNC_REG (0x002c)



PWM_TIMER2_PHASE_DIRECTION 0：递增；1：递减。(读/写)

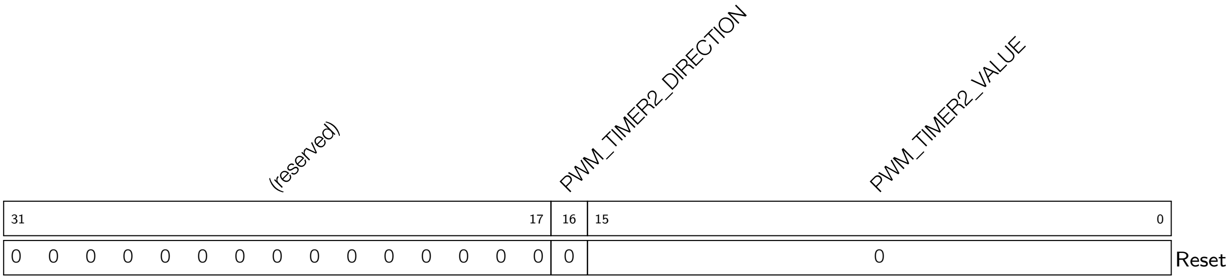
PWM_TIMER2_PHASE 同步事件中定时器重载相位。(读 / 写)

PWM_TIMER2_SYNCO_SEL 选择 PWM 定时器 2 同步输出来源。0：同步输入；1：TEZ；2：TEP；其他值：同步输出一直输出 0。(读 / 写)

PWM_TIMER2_SYNC_SW 此位取反，触发软件同步事件。(读 / 写)

PWM_TIMER2_SYNCI_EN 置 1 时使能在同步输入事件时的定时器相位重载。(读 / 写)

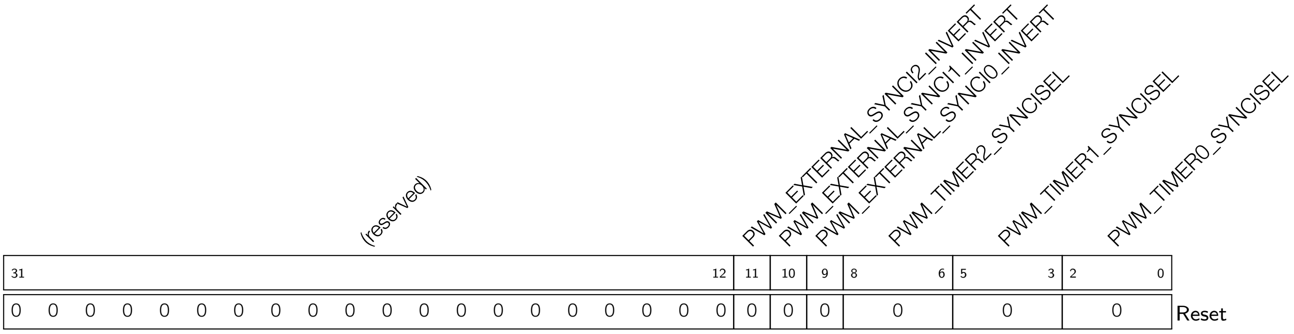
Register 16.13. PWM_TIMER2_STATUS_REG (0x0030)



PWM_TIMER2_DIRECTION 当前 PWM 定时器 2 的计数器模式。0：递增模式；1：递减模式。(只读)

PWM_TIMER2_VALUE 当前 PWM 定时器 2 计数器的值。(只读)

Register 16.14. PWM_TIMER_SYNCI_CFG_REG (0x0034)



PWM_EXTERNAL_SYNCI2_INVERT 将来自 GPIO 矩阵的 SYNC2 反相。（读 / 写）

PWM_EXTERNAL_SYNCI1_INVERT 将来自 GPIO 矩阵的 SYNC1 反相。（读 / 写）

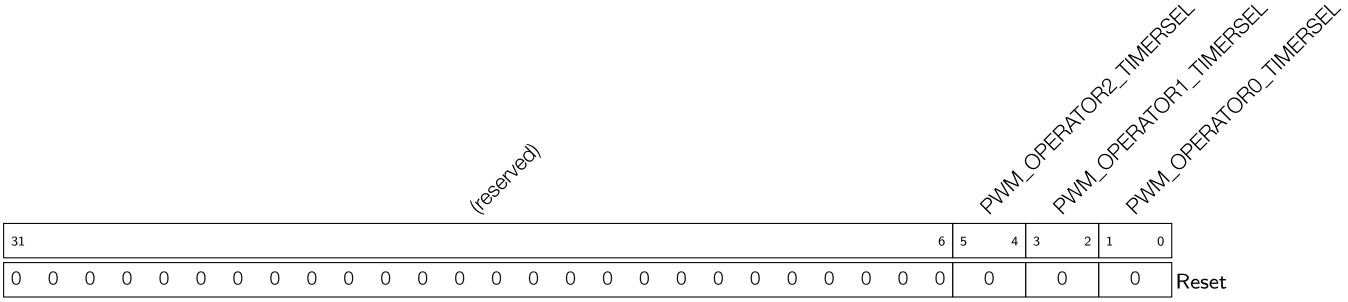
PWM_EXTERNAL_SYNCI0_INVERT 将来自 GPIO 矩阵的 SYNC0 反相。（读 / 写）

PWM_TIMER2_SYNCISEL 选择 PWM 定时器 2 的同步输入来源。1：PWM 定时器 0 同步输出；2：PWM 定时器 1 同步输出；3：PWM 定时器 2 同步输出；4：来自 GPIO 矩阵的 SYNC0；5：来自 GPIO 矩阵的 SYNC1；6：来自 GPIO 矩阵的 SYNC2；其他值：未选择任何同步输入。（读 / 写）

PWM_TIMER1_SYNCISEL 选择 PWM 定时器 1 的同步输入来源。1：PWM 定时器 0 同步输出；2：PWM 定时器 1 同步输出；3：PWM 定时器 2 同步输出；4：来自 GPIO 矩阵的 SYNC0；5：来自 GPIO 矩阵的 SYNC1；6：来自 GPIO 矩阵的 SYNC2；其他值：未选择任何同步输入。（读 / 写）

PWM_TIMER0_SYNCISEL 选择 PWM 定时器 0 的同步输入来源。1：PWM 定时器 0 同步输出；2：PWM 定时器 1 同步输出；3：PWM 定时器 2 同步输出；4：来自 GPIO 矩阵的 SYNC0；5：来自 GPIO 矩阵的 SYNC1；6：来自 GPIO 矩阵的 SYNC2；其他值：未选择任何同步输入。（读 / 写）

Register 16.15. PWM_OPERATOR_TIMERSEL_REG (0x0038)



PWM_OPERATOR2_TIMERSEL 选择 PWM 操作器 2 的定时参考来源。0：定时器 0；1：定时器 1；2：定时器 2。（读 / 写）

PWM_OPERATOR1_TIMERSEL 选择 PWM 操作器 1 的定时参考来源。0：定时器 0；1：定时器 1；2：定时器 2。（读 / 写）

PWM_OPERATOR0_TIMERSEL 选择 PWM 操作器 0 的定时参考来源。0：定时器 0；1：定时器 1；2：定时器 2。（读 / 写）

Register 16.16. PWM_GEN0_STMP_CFG_REG (0x003c)

(reserved)	PWM_GEN0_B_SHDW_FULL	PWM_GEN0_A_SHDW_FULL	PWM_GEN0_B_UPMETHOD	PWM_GEN0_A_UPMETHOD	
31 10	9	8	7 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

PWM_GEN0_B_SHDW_FULL 由硬件置 1 和清零。置 1 时，PWM 生成器 0 时间戳寄存器 B 的影子寄存器被写入，写入的值将传输给有效寄存器 B。清零时，有效寄存器 B 中写入其影子寄存器最新的值。（只读）

PWM_GEN0_A_SHDW_FULL 硬件置 1 或复位。置 1 时，PWM 生成器 0 时间戳寄存器 A 的影子寄存器被写入，写入的值将传输给有效寄存器 A。清零时，有效寄存器 A 中写入其影子寄存器最新的值。（只读）

PWM_GEN0_B_UPMETHOD PWM 生成器 0 时间戳寄存器 B 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

PWM_GEN0_A_UPMETHOD PWM 生成器 0 时间戳寄存器 A 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.17. PWM_GEN0_TSTMP_A_REG (0x0040)

(reserved)	PWM_GEN0_A	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_GEN0_A PWM 生成器 0 时间戳寄存器 A 的影子寄存器。（读 / 写）

Register 16.18. PWM_GEN0_TSTMP_B_REG (0x0044)

(reserved)	PWM_GEN0_B	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_GEN0_B PWM 生成器 0 时间戳寄存器 B 的影子寄存器。（读 / 写）

Register 16.19. PWM_GEN0_CFG0_REG (0x0048)

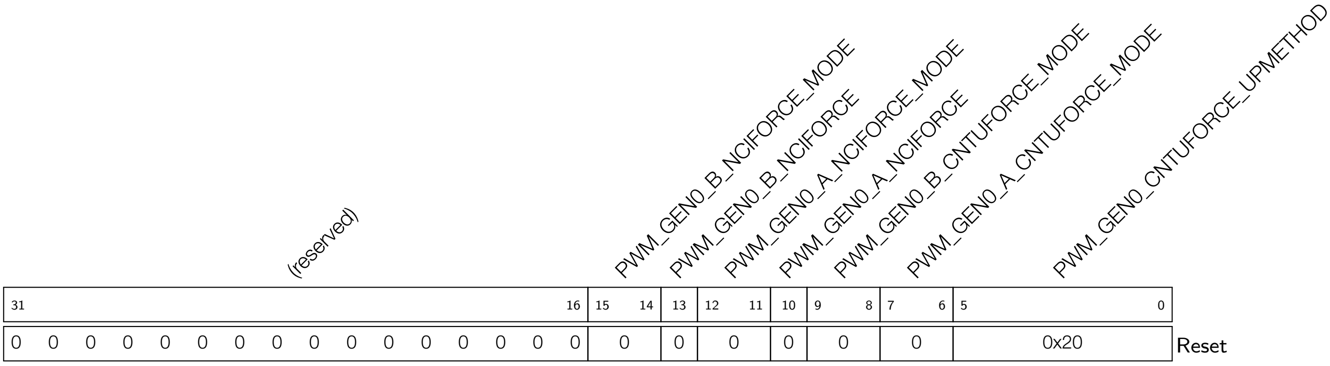
(reserved)	PWM_GEN0_T1_SEL	PWM_GEN0_T0_SEL	PWM_GEN0_CFG_UPMETHOD	
31 10	9 7	6 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	Reset

PWM_GEN0_T1_SEL 选择 PWM 生成器 0 event_t1 的信号源，立即生效。0：fault_event0；1：fault_event1；2：fault_event2；3：sync_taken；4：无。（读 / 写）

PWM_GEN0_T0_SEL 选择 PWM 生成器 0 event_t0 的信号源，立即生效。0：fault_event0；1：fault_event1；2：fault_event2；3：sync_taken；4：无。（读 / 写）

PWM_GEN0_CFG_UPMETHOD PWM 生成器 0 有效配置寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.20. PWM_GEN0_FORCE_REG (0x004c)



PWM_GEN0_B_NCIFORCE_MODE 设置 PWM0B 上的非连续即时软件强制模式。0：关闭；1：低电平；2：高电平；3：关闭。(读 / 写)

PWM_GEN0_B_NCIFORCE 该位的值取反时将触发 PWM0B 上的非连续即时软件强制事件。(读 / 写)

PWM_GEN0_A_NCIFORCE_MODE 设置用于 PWM0A 的非连续即时软件强制模式。0：关闭；1：低电平；2：高电平；3：关闭。(读 / 写)

PWM_GEN0_A_NCIFORCE 该位的值取反时将触发 PWM0A 上的非连续即时软件强制事件。(读 / 写)

PWM_GEN0_B_CNTUFORCE_MODE PWM0B 的连续软件强制模式。0：关闭；1：低电平；2：高电平；3：关闭。(读 / 写)

PWM_GEN0_A_CNTUFORCE_MODE 设置 PWM0A 的连续软件强制模式。0：关闭；1：低电平；2：高电平；3：关闭。(读 / 写)

PWM_GEN0_CNTUFORCE_UPMETHOD 生成器 0 的连续软件强制事件更新方式。所有 bit 为 0 时：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生 TEA 事件时更新；bit3 为 1：发生 TEB 事件时更新；bit4：发生同步事件时更新；bit5 为 1：关闭更新。(本文档中，TEA/B 指定时器值为寄存器 A/B 的值时生成的事件。)(读 / 写)

Register 16.21. PWM_GEN0_A_REG (0x0050)

(reserved)	PWM_GEN0_A_DT1	PWM_GEN0_A_DT0	PWM_GEN0_A_DTEB	PWM_GEN0_A_DTEA	PWM_GEN0_A_DTEP	PWM_GEN0_A_DTEZ	PWM_GEN0_A_UT1	PWM_GEN0_A_UT0	PWM_GEN0_A_UTEB	PWM_GEN0_A_UTEA	PWM_GEN0_A_UTEP	PWM_GEN0_A_UTEZ	
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PWM_GEN0_A_DT1 定时器递减时，event_t1 在 PWM0A 上触发的操作。0：波形无改变；1：拉低；2：拉高；3：取反。（读 / 写）

PWM_GEN0_A_DT0 定时器递减时，event_t0 在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_DTEB 定时器递减时，TEB 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_DTEA 定时器递减时，TEA 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_DTEP 定时器递减时，TEP 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_DTEZ 定时器递减时，TEZ 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_UT1 定时器递增时，event_t1 在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_UT0 定时器递增时，event_t0 在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_UTEB 定时器递增时，TEB 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_UTEA 定时器递增时，TEA 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_UTEP 定时器递增时，TEP 事件在 PWM0A 上触发的操作。（读 / 写）

PWM_GEN0_A_UTEZ 定时器递增时，TEZ 事件在 PWM0A 上触发的操作。（读 / 写）

Register 16.22. PWM_GEN0_B_REG (0x0054)

	(reserved)	PWM_GEN0_B_DT1	PWM_GEN0_B_DT0	PWM_GEN0_B_DTEB	PWM_GEN0_B_DTEA	PWM_GEN0_B_DTEP	PWM_GEN0_B_DTEZ	PWM_GEN0_B_UT1	PWM_GEN0_B_UT0	PWM_GEN0_B_UTEB	PWM_GEN0_B_UTEA	PWM_GEN0_B_UTEP	PWM_GEN0_B_UTEZ
	31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Reset	0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0

PWM_GEN0_B_DT1 定时器递减时，event_t1 在 PWM0B 上触发的操作。0：波形无改变；1：拉低；2：拉高；3：取反。（读 / 写）

PWM_GEN0_B_DT0 定时器递减时，event_t0 事件在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_DTEB 定时器递减时，TEB 事件在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_DTEA 定时器递减时，TEA 事件在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_DTEP 定时器递减时，TEP 事件在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_DTEZ 定时器递减时，TEZ 事件在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_UT1 定时器递增时，event_t1 在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_UT0 定时器递增时，event_t0 在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_UTEB 定时器递增时，TEB 在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_UTEA 定时器递增时，TEA 在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_UTEP 定时器递增时，TEP 在 PWM0B 上触发的操作。（读 / 写）

PWM_GEN0_B_UTEZ 定时器递增时，TEZ 在 PWM0B 上触发的操作。（读 / 写）

Register 16.23. PWM_DT0_CFG_REG (0x0058)

(reserved)	PWM_DT0_CLK_SEL	PWM_DT0_B_OUTBYPASS	PWM_DT0_A_OUTBYPASS	PWM_DT0_FED_OUTINVERT	PWM_DT0_RED_OUTINVERT	PWM_DT0_FED_INSEL	PWM_DT0_RED_INSEL	PWM_DT0_B_OUTSWAP	PWM_DT0_A_OUTSWAP	PWM_DT0_DEB_MODE	PWM_DT0_RED_UPMETHOD	PWM_DT0_FED_UPMETHOD	
31 18	17	16	15	14	13	12	11	10	9	8	7 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	1	0	0	0	0	0	0	0	0	0	Reset

PWM_DT0_CLK_SEL 选择死区时间生成器 0 的时钟。0：PWM_clk；1：PT_clk。（读 / 写）

PWM_DT0_B_OUTBYPASS 表 16-5 中的 S0。（读 / 写）

PWM_DT0_A_OUTBYPASS 表 16-5 中的 S1。（读 / 写）

PWM_DT0_FED_OUTINVERT 表 16-5 中的 S3。（读 / 写）

PWM_DT0_RED_OUTINVERT 表 16-5 中的 S2。（读 / 写）

PWM_DT0_FED_INSEL 表 16-5 中的 S5。（读 / 写）

PWM_DT0_RED_INSEL 表 16-5 中的 S4。（读 / 写）

PWM_DT0_B_OUTSWAP 表 16-5 中的 S7。（读 / 写）

PWM_DT0_A_OUTSWAP S6 表 16-5 中的 S6。（读 / 写）

PWM_DT0_DEB_MODE S8 表 16-5 中的 S8, B 路双边沿模式。0：下降沿延迟 / 下降沿延迟分别在不同的路径中生效；1：下降沿延迟 / 下降沿延迟在路径 B 上生效；PWMxA 正常输出。（读 / 写）

PWM_DT0_RED_UPMETHOD 上升沿延迟有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

PWM_DT0_FED_UPMETHOD 下降沿延迟有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.24. PWM_DT0_FED_CFG_REG (0x005c)

(reserved)	PWM_DT0_FED	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_DT0_FED FED 的影子寄存器。（读 / 写）

Register 16.25. PWM_DT0_RED_CFG_REG (0x0060)

(reserved)	PWM_DT0_RED	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_DT0_RED RED 的影子寄存器。（读 / 写）

Register 16.26. PWM_CARRIER0_CFG_REG (0x0064)

(reserved)	PWM_CARRIER0_IN_INVERT	PWM_CARRIER0_OUT_INVERT	PWM_CARRIER0_OSHWTH	PWM_CARRIER0_DUTY	PWM_CARRIER0_PRESCALE	PWM_CARRIER0_EN	
31 14	13	12	11 8	7 5	4 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

PWM_CARRIER0_IN_INVERT 置 1 时，将此模块的 PWM0A 和 PWM0B 输入反相。（读 / 写）

PWM_CARRIER0_OUT_INVERT 置 1 时，将此模块的 PWM0A 和 PWM0B 输出反相。（读 / 写）

PWM_CARRIER0_OSHWTH 载波第一个脉冲的宽度，单位为载波周期。（读 / 写）

PWM_CARRIER0_DUTY 选择载波占空比。占空比 = PWM_CARRIER0_DUTY / 8。（读 / 写）

PWM_CARRIER0_PRESCALE PWM 载波 0 时钟 (PC_clk) 的预分频值。PC_clk 周期 = PWM_clk 周期 * (PWM_CARRIER0_PRESCALE + 1)。（读 / 写）

PWM_CARRIER0_EN 置 1 时，使能载波 0 的功能。清零时，载波 0 被绕过。（读 / 写）

Register 16.27. PWM_FH0_CFG0_REG (0x0068)

(reserved)	PWM_FH0_B_OST_U	PWM_FH0_B_OST_D	PWM_FH0_B_CBC_U	PWM_FH0_B_CBC_D	PWM_FH0_A_OST_U	PWM_FH0_A_OST_D	PWM_FH0_A_CBC_U	PWM_FH0_A_CBC_D	PWM_FH0_F0_OST	PWM_FH0_F1_OST	PWM_FH0_F2_OST	PWM_FH0_SW_OST	PWM_FH0_F0_CBC	PWM_FH0_F1_CBC	PWM_FH0_F2_CBC	PWM_FH0_SW_CBC	
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PWM_FH0_B_OST_U 定时器递增计数并且发生故障事件时, PWM0B 上的一次性模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_B_OST_D 定时器递减计数并且发生故障事件时, PWM0B 上的一次性模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_B_CBC_U 定时器递增计数并且发生故障事件时, PWM0B 上的逐周期模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_B_CBC_D 定时器递减计数并且发生故障事件时, PWM0B 上的逐周期模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_A_OST_U 定时器递增计数并且发生故障事件时, PWM0A 上的一次性模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_A_OST_D 定时器递减计数并且发生故障事件时, PWM0A 上的一次性模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_A_CBC_U 定时器递增计数并且发生故障事件时, PWM0A 上的逐周期模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_A_CBC_D 定时器递减计数并且发生故障事件时, PWM0A 上的逐周期模式操作。0: 无; 1: 强制拉低; 2: 强制拉高; 3: 取反。(读 / 写)

PWM_FH0_F0_OST 设置 event_f0 是否触发一次性模式操作。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_F1_OST 设置 event_f1 是否触发一次性模式操作。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_F2_OST 设置 event_f2 是否触发一次性模式操作。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_SW_OST 软件强制一次性模式操作的使能寄存器。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_F0_CBC 设置 event_f0 是否触发逐周期模式操作。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_F1_CBC 设置 event_f1 是否触发逐周期模式操作。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_F2_CBC 设置 event_f2 是否触发逐周期模式操作。0: 关闭; 1: 使能。(读 / 写)

PWM_FH0_SW_CBC 使能软件强制逐周期模式操。0: 关闭; 1: 使能。(读 / 写)

Register 16.28. PWM_FH0_CFG1_REG (0x006c)

(reserved)	PWM_FH0_FORCE_OST	PWM_FH0_FORCE_CBC	PWM_FH0_CBCPULSE	PWM_FH0_CLR_OST	
31 5	4	3	2 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

PWM_FH0_FORCE_OST 通过软件将此位的值取反，可触发一次性模式的操作。（读 / 写）

PWM_FH0_FORCE_CBC 通过软件将此位的值取反，可触发逐周期模式的操作。（读 / 写）

PWM_FH0_CBCPULSE 设置逐周期模式操作更新的时间点。bit0 为 1：发生 TEZ 事件时；bit1 为 1：发生 TEP 事件时。（读 / 写）

PWM_FH0_CLR_OST 通过软件将此位的值取反，清除持续一次性模式操作。（读 / 写）

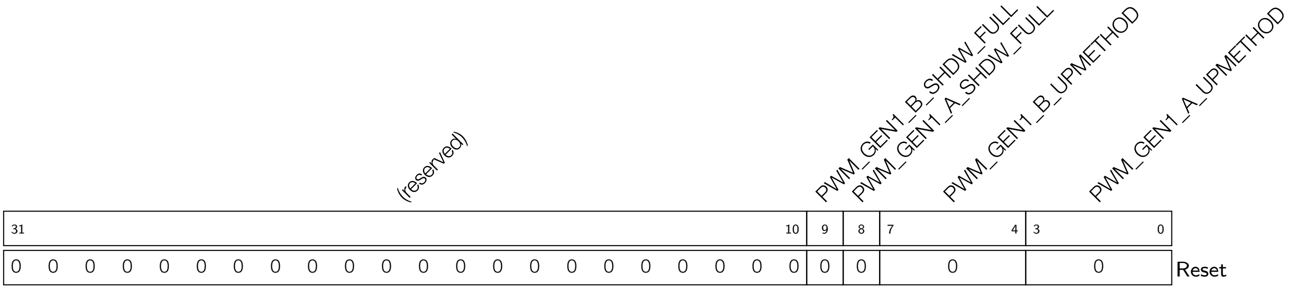
Register 16.29. PWM_FH0_STATUS_REG (0x0070)

(reserved)	PWM_FH0_OST_ON	PWM_FH0_CBC_ON	
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	Reset

PWM_FH0_OST_ON 由硬件置 1 和清零。置 1 时，一次性模式的操作正在进行进行。（只读）

PWM_FH0_CBC_ON 由硬件置 1 和清零。置 1 时，逐周期模式的操作正在进行进行。（只读）

Register 16.30. PWM_GEN1_STMP_CFG_REG (0x0074)



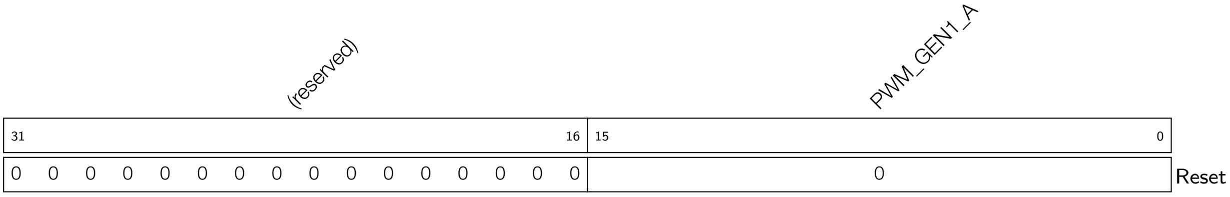
PWM_GEN1_B_SHDW_FULL 由硬件置 1 和清零。置 1 时，PWM 生成器 1 时间戳寄存器 B 的影子寄存器被写入，写入的值将传输给有效寄存器 B。清零时，有效寄存器 B 中写入其影子寄存器最新的值。（只读）

PWM_GEN1_A_SHDW_FULL 由硬件置 1 和清零。置 1 时，PWM 生成器 1 时间戳寄存器 A 的影子寄存器被写入，写入的值将传输给有效寄存器 A。清零时，有效寄存器 A 中写入其影子寄存器最新的值。（只读）

PWM_GEN1_B_UPMETHOD PWM 生成器 1 时间戳寄存器 B 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时更新；bit3 为 1：关闭更新。（读 / 写）

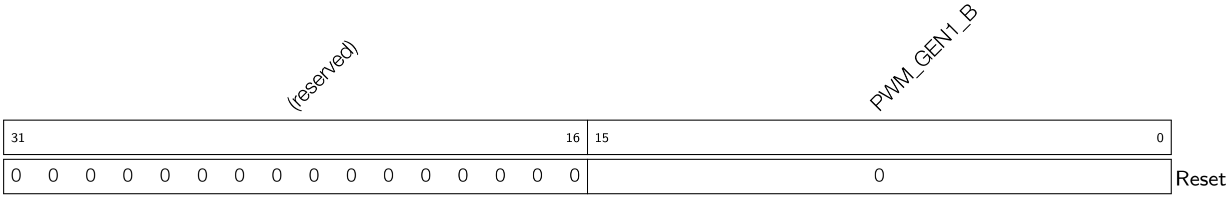
PWM_GEN1_A_UPMETHOD PWM 生成器 1 时间戳寄存器 A 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.31. PWM_GEN1_TSTMP_A_REG (0x0078)



PWM_GEN1_A PWM 生成器 1 时间戳寄存器 A 的影子寄存器。（读 / 写）

Register 16.32. PWM_GEN1_TSTMP_B_REG (0x007c)



PWM_GEN1_B PWM 生成器 1 时间戳寄存器 B 的影子寄存器。（读 / 写）

Register 16.33. PWM_GEN1_CFG0_REG (0x0080)

(reserved)	PWM_GEN1_T1_SEL	PWM_GEN1_T0_SEL	PWM_GEN1_CFG_UPMETHOD	
31 10	9 7	6 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	Reset

PWM_GEN1_T1_SEL 选择 PWM 生成器 1 event_t1 的信号源，立即生效。0：fault_event0；1：fault_event1；2：fault_event2；3：sync_taken；4：无。(读 / 写)

PWM_GEN1_T0_SEL 选择 PWM 生成器 1 event_t0 的信号源，立即生效。0：fault_event0；1：fault_event1；2：fault_event2；3：sync_taken；4：无。(读 / 写)

PWM_GEN1_CFG_UPMETHOD PWM 生成器 1 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。(读 / 写)

Register 16.34. PWM_GEN1_FORCE_REG (0x0084)

(reserved)	PWM_GEN1_B_NCIFORCE_MODE	PWM_GEN1_B_NCIFORCE	PWM_GEN1_A_NCIFORCE_MODE	PWM_GEN1_A_NCIFORCE	PWM_GEN1_B_CNTUFORCE_MODE	PWM_GEN1_A_CNTUFORCE_MODE	PWM_GEN1_CNTUFORCE_UPMETHOD	
31 16	15 14	13	12 11	10	9 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	0x20	Reset

PWM_GEN1_B_NCIFORCE_MODE 用于 PWM1B 的非持续性即时软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN1_B_NCIFORCE 该位的值取反时将触发 PWM1B 上的非连续即时软件强制事件。（读 / 写）

PWM_GEN1_A_NCIFORCE_MODE 用于 PWM1B 的非持续性即时软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN1_A_NCIFORCE 该位的值取反时将触发 PWM1A 上的非连续即时软件强制事件。（读 / 写）

PWM_GEN1_B_CNTUFORCE_MODE 用于 PWM1B 的连续软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN1_A_CNTUFORCE_MODE 用于 PWM1A 的连续软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN1_CNTUFORCE_UPMETHOD PWM 生成器 1 持续软件强制的更新方式。0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生 TEA 事件时更新；bit3 为 1：发生 TEB 事件时更新；bit4 为 1：发生同步时间时更新；bit5 为 1：关闭更新。（本文档中的 TEA/B 表示计时器值等于寄存器 A/B 生成的事件。）（读 / 写）

Register 16.35. PWM_GEN1_A_REG (0x0088)

(reserved)	PWM_GEN1_A_DT1	PWM_GEN1_A_DT0	PWM_GEN1_A_DTEB	PWM_GEN1_A_DTEA	PWM_GEN1_A_DTEP	PWM_GEN1_A_DTEZ	PWM_GEN1_A_UT1	PWM_GEN1_A_UT0	PWM_GEN1_A_UTEB	PWM_GEN1_A_UTEA	PWM_GEN1_A_UTEP	PWM_GEN1_A_UTEZ
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0

Reset

PWM_GEN1_A_DT1 定时器递减计数时，event_t1 在 PWM1A 上触发的操作。0：无；1：拉低；2：拉高；3：取反。（读 / 写）

PWM_GEN1_A_DT0 定时器递减计数时，event_t0 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_DTEB 定时器递减计数时，TEB 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_DTEA 定时器递减计数时，TEA 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_DTEP 定时器递减计数时，TEP 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_DTEZ 定时器递减计数时，TEZ 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_UT1 定时器递增计数时，event_t1 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_UT0 定时器递增计数时，event_t0 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_UTEB 定时器递增计数时，TEB 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_UTEA 定时器递增计数时，TEA 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_UTEP 定时器递增计数时，TEP 在 PWM1A 上触发的操作。（读 / 写）

PWM_GEN1_A_UTEZ 定时器递增计数时，TEZ 在 PWM1A 上触发的操作。（读 / 写）

Register 16.36. PWM_GEN1_B_REG (0x008c)

(reserved)	PWM_GEN1_B_DT1	PWM_GEN1_B_DT0	PWM_GEN1_B_DTEB	PWM_GEN1_B_DTEA	PWM_GEN1_B_DTEP	PWM_GEN1_B_DTEZ	PWM_GEN1_B_UT1	PWM_GEN1_B_UT0	PWM_GEN1_B_UTEB	PWM_GEN1_B_UTEA	PWM_GEN1_B_UTEP	PWM_GEN1_B_UTEZ	
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PWM_GEN1_B_DT1 定时器递减计数时，event_t1 在 PWM1B 上触发的操作。0：无；1：拉低；2：拉高；3：取反。（读 / 写）

PWM_GEN1_B_DT0 定时器递减计数时，event_t0 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_DTEB 定时器递减计数时，TEB 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_DTEA 定时器递减计数时，TEA 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_DTEP 定时器递减计数时，TEP 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_DTEZ 定时器递减计数时，TEZ 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_UT1 定时器递增计数时，event_t1 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_UT0 定时器递增计数时，event_t0 在 PWM1B 上触发的操作。（读 / 写）

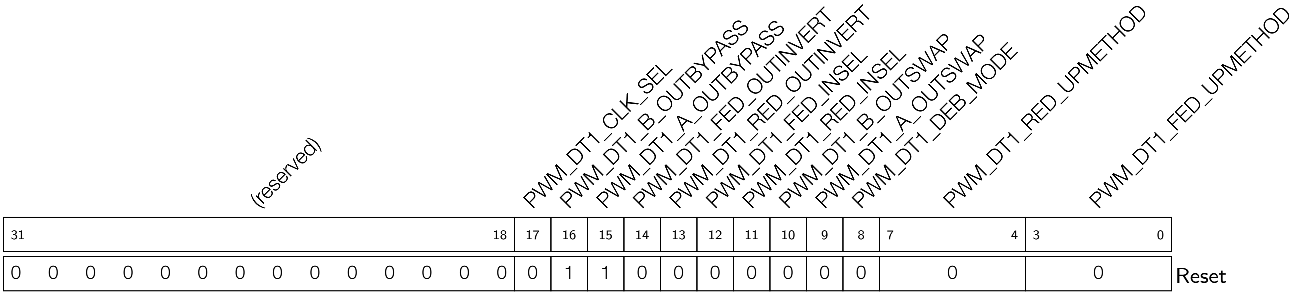
PWM_GEN1_B_UTEB 定时器递增计数时，TEB 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_UTEA 定时器递增计数时，TEA 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_UTEP 定时器递增计数时，TEP 在 PWM1B 上触发的操作。（读 / 写）

PWM_GEN1_B_UTEZ 定时器递增计数时，TEZ 在 PWM1B 上触发的操作。（读 / 写）

Register 16.37. PWM_DT1_CFG_REG (0x0090)



PWM_DT1_CLK_SEL 设置死区时间生成器时钟。0：PWM_clk；1：PT_clk。（读 / 写）

PWM_DT1_B_OUTBYPASS 表 16-5 中的 S0。（读 / 写）

PWM_DT1_A_OUTBYPASS 表 16-5 中的 S1。（读 / 写）

PWM_DT1_FED_OUTINVERT 表 16-5 中的 S3。（读 / 写）

PWM_DT1_RED_OUTINVERT 表 16-5 中的 S2。（读 / 写）

PWM_DT1_FED_INSEL 表 16-5 中的 S5。（读 / 写）

PWM_DT1_RED_INSEL 表 16-5 中的 S4。（读 / 写）

PWM_DT1_B_OUTSWAP 表 16-5 中的 S7。（读 / 写）

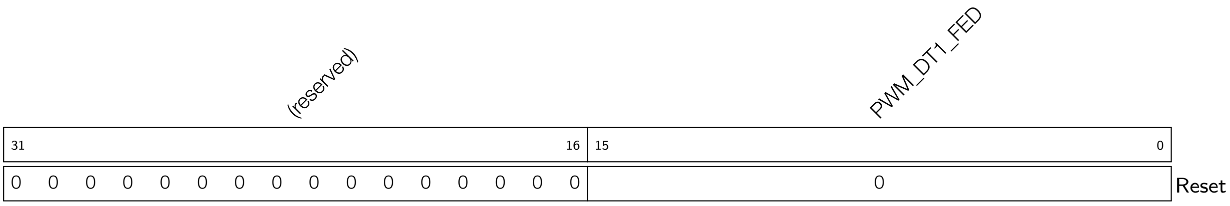
PWM_DT1_A_OUTSWAP 表 16-5 中的 S6。（读 / 写）

PWM_DT1_DEB_MODE 表 16-5 中的 S8, B 路双沿模式。0：下降沿延迟 / 下降沿延迟分别在不同的路径中生效；1：下降沿延迟 / 下降沿延迟在路径 B 上生效；PWMxA 正常输出。（读 / 写）

PWM_DT1_RED_UPMETHOD RED（上升沿延迟）有效寄存器的更新方式。0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步事件时更新；bit3 为 1：关闭更新。（读 / 写）

PWM_DT1_FED_UPMETHOD FED（下降沿延迟）有效寄存器的更新方式。0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步事件时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.38. PWM_DT1_FED_CFG_REG (0x0094)



PWM_DT1_FED FED 影子寄存器。（读 / 写）

Register 16.39. PWM_DT1_RED_CFG_REG (0x0098)

	(reserved)	PWM_DT1_RED	
	31 16	15 0	
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_DT1_RED RED 影子寄存器。(读 / 写)

Register 16.40. PWM_CARRIER1_CFG_REG (0x009c)

	(reserved)	PWM_CARRIER1_IN_INVERT	PWM_CARRIER1_OUT_INVERT	PWM_CARRIER1_OSHWTH	PWM_CARRIER1_DUTY	PWM_CARRIER1_PRESCALE	PWM_CARRIER1_EN	
	31 14	13	12	11 8	7 5	4 1	0	
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

PWM_CARRIER1_IN_INVERT 置 1 时，将此模块的 PWM1A 和 PWM1B 输入反相。(读 / 写)

PWM_CARRIER1_OUT_INVERT 置 1 时，将此模块的 PWM1A 和 PWM1B 输出反相。(读 / 写)

PWM_CARRIER1_OSHWTH 载波第一个脉冲的宽度，单位为载波周期。(读 / 写)

PWM_CARRIER1_DUTY 设置载波占空比。占空比 = PWM_CARRIER1_DUTY/8。(读 / 写)

PWM_CARRIER1_PRESCALE PWM 载波 1 时钟 (PC_clk) 预分频值。PC_clk 周期 = PWM_clk 周期 * (PWM_CARRIER1_PRESCALE + 1)。(读 / 写)

PWM_CARRIER1_EN 置 1 时，使能载波 1 功能。此位清零时，绕过载波 1。(读 / 写)

Register 16.41. PWM_FH1_CFG0_REG (0x00a0)

	(reserved)	PWM_FH1_B_OST_U	PWM_FH1_B_OST_D	PWM_FH1_B_CBC_U	PWM_FH1_B_CBC_D	PWM_FH1_A_OST_U	PWM_FH1_A_OST_D	PWM_FH1_A_CBC_U	PWM_FH1_A_CBC_D	PWM_FH1_F0_OST	PWM_FH1_F1_OST	PWM_FH1_F2_OST	PWM_FH1_SW_OST	PWM_FH1_F0_CBC	PWM_FH1_F1_CBC	PWM_FH1_F2_CBC	PWM_FH1_SW_CBC
	31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7	6	5	4	3	2	1	0
Reset	0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

PWM_FH1_B_OST_U 定时器递增计数并且发生故障事件时，PWM1B 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_B_OST_D 定时器递减计数并且发生故障事件时，PWM1B 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_B_CBC_U 定时器递增计数并且发生故障事件时，PWM1B 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_B_CBC_D 定时器递减计数并且发生故障事件时，PWM1B 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_A_OST_U 定时器递增计数并且发生故障事件时，PWM1A 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_A_OST_D 定时器递减计数并且发生故障事件时，PWM1A 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_A_CBC_U 定时器递增计数并且发生故障事件时，PWM1A 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_A_CBC_D 定时器递减计数并且发生故障事件时，PWM1A 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。(读 / 写)

PWM_FH1_F0_OST 设置 event_f0 触发一次性模式操作。0：关闭；1：使能。(读 / 写)

PWM_FH1_F1_OST 设置 event_f1 触发一次性模式操作。0：关闭；1：使能。(读 / 写)

PWM_FH1_F2_OST 设置 event_f2 触发一次性模式操作。0：关闭；1：使能。(读 / 写)

PWM_FH1_SW_OST 软件强制一次性模式操作的使能寄存器。0：关闭；1：使能。(读 / 写)

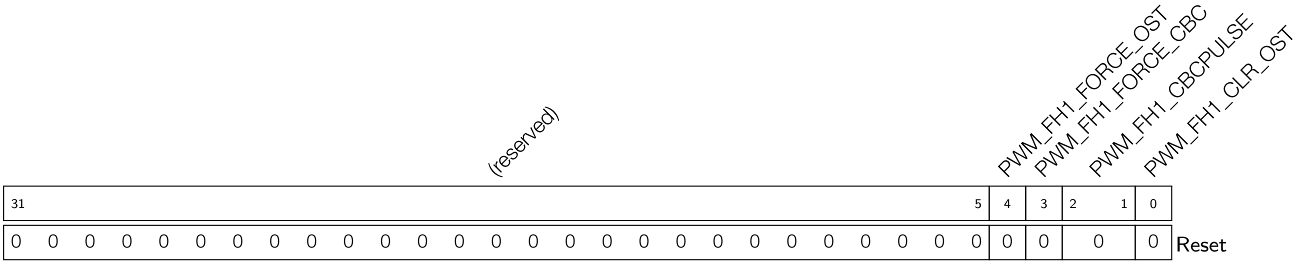
PWM_FH1_F0_CBC 设置 event_f0 触发逐周期模式操作。0：关闭；1：使能。(读 / 写)

PWM_FH1_F1_CBC 设置 event_f1 触发逐周期模式操作。0：关闭；1：使能。(读 / 写)

PWM_FH1_F2_CBC 设置 event_f2 触发逐周期模式操作。0：关闭；1：使能。(读 / 写)

PWM_FH1_SW_CBC 软件强制逐周期模式操作的使能寄存器。0：关闭；1：使能。(读 / 写)

Register 16.42. PWM_FH1_CFG1_REG (0x00a4)



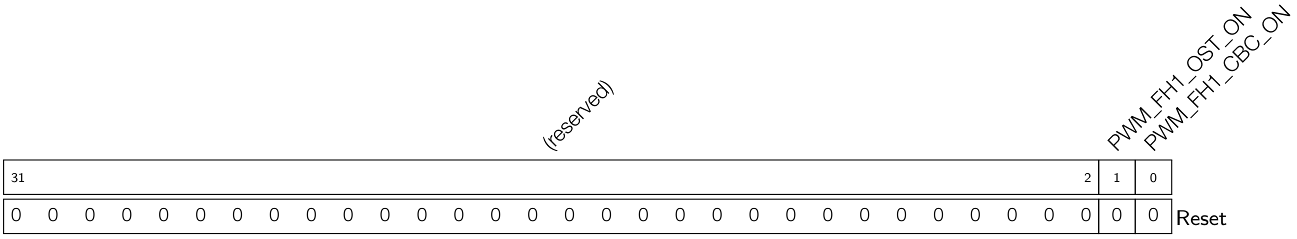
PWM_FH1_FORCE_OST 通过软件将该位取反时，触发一次性模式操作。（读 / 写）

PWM_FH1_FORCE_CBC 通过软件将该位取反时，触发逐周期模式操作。（读 / 写）

PWM_FH1_CBCPULSE 设置逐周期模式操作的更新方式。bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新。（读 / 写）

PWM_FH1_CLR_OST 置 1 清除正在进行的一次性模式操作。（读 / 写）

Register 16.43. PWM_FH1_STATUS_REG (0x00a8)



PWM_FH1_OST_ON 通过硬件将置 1 时或清零。置 1 时，一次性模式操作正在进行。（只读）

PWM_FH1_CBC_ON 通过硬件将置 1 时或清零。置 1 时，逐周期模式操作正在进行。（只读）

Register 16.44. PWM_GEN2_STMP_CFG_REG (0x00ac)

(reserved)	PWM_GEN2_B_SHDW_FULL	PWM_GEN2_A_SHDW_FULL	PWM_GEN2_B_UPMETHOD	PWM_GEN2_A_UPMETHOD	
31 10	9	8	7 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

PWM_GEN2_B_SHDW_FULL 由硬件置 1 和清零。置 1 时，PWM 生成器 2 时间戳寄存器 B 的影子寄存器被写入，写入的值将传输给有效寄存器 B。清零时，有效寄存器 B 中写入其影子寄存器最新的值。（只读）

PWM_GEN2_A_SHDW_FULL 由硬件置 1 和清零。置 1 时，PWM 生成器 2 时间戳寄存器 A 的影子寄存器被写入，写入的值将传输给有效寄存器 A。清零时，有效寄存器 A 中写入其影子寄存器最新的值。（只读）

PWM_GEN2_B_UPMETHOD 生成器 2 时间戳寄存器 B 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步事件时更新；bit3 为 1：关闭更新。（读 / 写）

PWM_GEN2_A_UPMETHOD 生成器 2 时间戳寄存器 A 有效寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步事件时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.45. PWM_GEN2_TSTMP_A_REG (0x00b0)

(reserved)	PWM_GEN2_A	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_GEN2_A PWM 生成器 2 时间戳 A 的影子寄存器。（读 / 写）

Register 16.46. PWM_GEN2_TSTMP_B_REG (0x00b4)

(reserved)	PWM_GEN2_B	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_GEN2_B PWM 生成器 2 时间戳 B 的影子寄存器。（读 / 写）

Register 16.47. PWM_GEN2_CFG0_REG (0x00b8)

(reserved)		PWM_GEN2_T1_SEL		PWM_GEN2_T0_SEL		PWM_GEN2_CFG_UPMETHOD		
31	10	9	7	6	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		Reset

PWM_GEN2_T1_SEL 设置 PWM 操作器 2 event_t1 信号源，立即生效。0：fault_event0；1：fault_event1；2：fault_event2；3：sync_taken, 4：无。（读 / 写）

PWM_GEN2_T0_SEL 设置 PWM 操作器 2 event_t0 信号源，立即生效。0：fault_event0；1：fault_event1；2：fault_event2；3：sync_taken, 4：无。（读 / 写）

PWM_GEN2_CFG_UPMETHOD PWM 生成器 2 有效配置寄存器的更新方式。所有 bit 值为 0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步时间时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.48. PWM_GEN2_FORCE_REG (0x00bc)

(reserved)	PWM_GEN2_B_NCIFORCE_MODE	PWM_GEN2_B_NCIFORCE	PWM_GEN2_A_NCIFORCE_MODE	PWM_GEN2_A_NCIFORCE	PWM_GEN2_B_CNTUFORCE_MODE	PWM_GEN2_A_CNTUFORCE_MODE	PWM_GEN2_CNTUFORCE_UPMETHOD	
31 16	15 14	13	12 11	10	9 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	0x20	Reset

PWM_GEN2_B_NCIFORCE_MODE 用于设置 PWM2B 的非持续性即时软件强制模式。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN2_B_NCIFORCE 该位的值取反时将触发 PWM2B 上的非连续即时软件强制事件。（读 / 写）

PWM_GEN2_A_NCIFORCE_MODE 用于 PWM2A 的非持续性即时软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN2_A_NCIFORCE 该位的值取反时将触发 PWM2A 上的非连续即时软件强制事件。（读 / 写）

PWM_GEN2_B_CNTUFORCE_MODE 用于 PWM2B 的持续性即时软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN2_A_CNTUFORCE_MODE 用于 PWM2A 的持续性即时软件强制事件。0：关闭；1：拉低；2：拉高；3：关闭。（读 / 写）

PWM_GEN2_CNTUFORCE_UPMETHOD PWM 生成器 2 的持续性软件强制事件的更新方式。0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生 TEA 事件时更新；bit3 为 1：发生 TEB 事件时更新；bit4 为 1：发生同步事件时更新；bit5 为 1：关闭更新。（这里以及以下的 TEA/B 表示定时器的值为寄存器 A/B 的值时生成的事件。）（读 / 写）

Register 16.49. PWM_GEN2_A_REG (0x00c0)

(reserved)	PWM_GEN2_A_DT1	PWM_GEN2_A_DT0	PWM_GEN2_A_DTEB	PWM_GEN2_A_DTEA	PWM_GEN2_A_DTEP	PWM_GEN2_A_DTEZ	PWM_GEN2_A_UT1	PWM_GEN2_A_UT0	PWM_GEN2_A_UTEB	PWM_GEN2_A_UTEA	PWM_GEN2_A_UTEP	PWM_GEN2_A_UTEZ	
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PWM_GEN2_A_DT1 定时器递减时，event_t1 在 PWM2A 上触发的操作。0：波形无改变；1：拉低；2：拉高；3：取反。（读 / 写）

PWM_GEN2_A_DT0 定时器递减时，event_t0 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_DTEB 定时器递减时，TEB 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_DTEA 定时器递减时，TEA 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_DTEP 定时器递减时，TEP 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_DTEZ 定时器递减时，TEZ 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_UT1 定时器递增时，event_t1 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_UT0 定时器递增时，event_t0 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_UTEB 定时器递增时，TEB 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_UTEA 定时器递增时，TEA 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_UTEP 定时器递增时，TEP 在 PWM2A 上触发的操作。（读 / 写）

PWM_GEN2_A_UTEZ 定时器递增时，TEZ 在 PWM2A 上触发的操作。（读 / 写）

Register 16.50. PWM_GEN2_B_REG (0x00c4)

(reserved)	PWM_GEN2_B_DT1	PWM_GEN2_B_DT0	PWM_GEN2_B_DTEB	PWM_GEN2_B_DTEA	PWM_GEN2_B_DTEP	PWM_GEN2_B_DTEZ	PWM_GEN2_B_UT1	PWM_GEN2_B_UT0	PWM_GEN2_B_UTEB	PWM_GEN2_B_UTEA	PWM_GEN2_B_UTEP	PWM_GEN2_B_UTEZ	
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7 6	5 4	3 2	1 0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PWM_GEN2_B_DT1 定时器递减时，event_t1 在 PWM2B 上触发的操作。0：波形无改变；1：拉低；2：拉高；3：取反。（读 / 写）

PWM_GEN2_B_DT0 定时器递减时，event_t0 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_DTEB 定时器递减时，TEB 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_DTEA 定时器递减时，TEA 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_DTEP 定时器递减时，TEP 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_DTEZ 定时器递减时，TEZ 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_UT1 定时器递增时，event_t1 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_UT0 定时器递增时，event_t0 在 PWM2B 上触发的操作。（读 / 写）

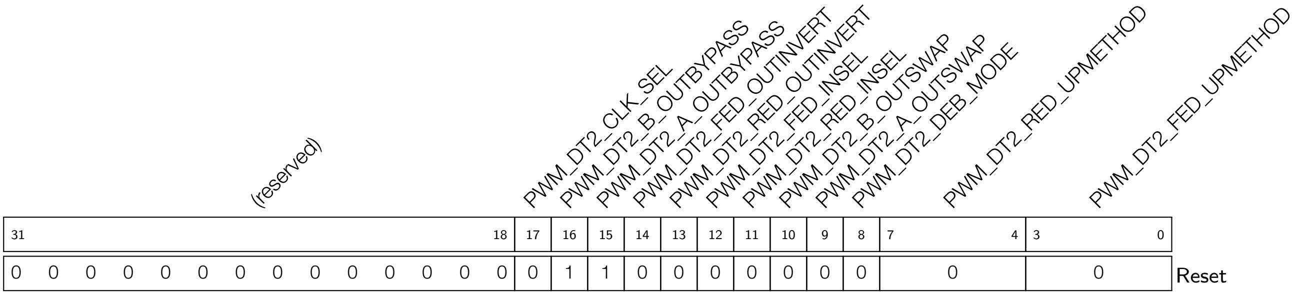
PWM_GEN2_B_UTEB 定时器递增时，TEB 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_UTEA 定时器递增时，TEA 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_UTEP 定时器递增时，TEP 在 PWM2B 上触发的操作。（读 / 写）

PWM_GEN2_B_UTEZ 定时器递增时，TEZ 在 PWM2B 上触发的操作。（读 / 写）

Register 16.51. PWM_DT2_CFG_REG (0x00c8)



PWM_DT2_CLK_SEL 设置死区时间生成器 1 的时钟。0：PWM_clk；1：PT_clk。（读 / 写）

PWM_DT2_B_OUTBYPASS 表 16-5 中的 S0。（读 / 写）

PWM_DT2_A_OUTBYPASS 表 16-5 中的 S1。（读 / 写）

PWM_DT2_FED_OUTINVERT 表 16-5 中的 S3。（读 / 写）

PWM_DT2_RED_OUTINVERT 表 16-5 中的 S2。（读 / 写）

PWM_DT2_FED_INSEL 表 16-5 中的 S5。（读 / 写）

PWM_DT2_RED_INSEL 表 16-5 中的 S4。（读 / 写）

PWM_DT2_B_OUTSWAP 表 16-5 中的 S7。（读 / 写）

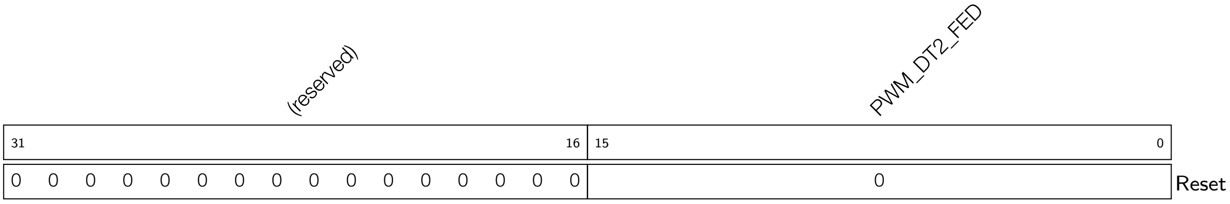
PWM_DT2_A_OUTSWAP 表 16-5 中的 S6。（读 / 写）

PWM_DT2_DEB_MODE 表 16-5 中的 S8, B 路双沿模式。0：下降沿延迟 / 下降沿延迟分别在不同的路径中生效；1：下降沿延迟 / 下降沿延迟在路径 B 上生效；PWMxA 正常输出。（读 / 写）

PWM_DT2_RED_UPMETHOD RED（上升沿延迟）有效寄存器的更新方式。0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步事件时更新；bit3 为 1：关闭更新。（读 / 写）

PWM_DT2_FED_UPMETHOD FED（下降沿延迟）有效寄存器的更新方式。0：立即更新；bit0 为 1：发生 TEZ 事件时更新；bit1 为 1：发生 TEP 事件时更新；bit2 为 1：发生同步事件时更新；bit3 为 1：关闭更新。（读 / 写）

Register 16.52. PWM_DT2_FED_CFG_REG (0x00cc)



PWM_DT2_FED FED 影子寄存器。（读 / 写）

Register 16.53. PWM_DT2_RED_CFG_REG (0x00d0)

(reserved)	PWM_DT2_RED	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PWM_DT2_RED RED 影子寄存器。(读 / 写)

Register 16.54. PWM_CARRIER2_CFG_REG (0x00d4)

(reserved)	PWM_CARRIER2_IN_INVERT	PWM_CARRIER2_OUT_INVERT	PWM_CARRIER2_OSHWTH	PWM_CARRIER2_DUTY	PWM_CARRIER2_PRESCALE	PWM_CARRIER2_EN	
31 14	13	12	11 8	7 5	4 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	Reset

PWM_CARRIER2_IN_INVERT 置 1 时，将此模块的 PWM2A 和 PWM2B 输入反相。(读 / 写)

PWM_CARRIER2_OUT_INVERT 置 1 时，将此模块的 PWM2A 和 PWM2B 输出反相。(读 / 写)

PWM_CARRIER2_OSHWTH 载波第一个脉冲的宽度，单位为载波周期。(读 / 写)

PWM_CARRIER2_DUTY 设置载波占空比。占空比 = PWM_CARRIER2_DUTY / 8。(读 / 写)

PWM_CARRIER2_PRESCALE PWM 载波 2 时钟 (PC_clk) 的预分频值。PC_clk 周期 = PWM_clk 周期 * (PWM_CARRIER0_PRESCALE + 1)。(读 / 写)

PWM_CARRIER2_EN 置 1 时，使能载波 2 功能。清零时，载波 2 被绕过。(读 / 写)

Register 16.55. PWM_FH2_CFG0_REG (0x00d8)

(reserved)	PWM_FH2_B_OST_U	PWM_FH2_B_OST_D	PWM_FH2_B_CBC_U	PWM_FH2_B_CBC_D	PWM_FH2_A_OST_U	PWM_FH2_A_OST_D	PWM_FH2_A_CBC_U	PWM_FH2_A_CBC_D	PWM_FH2_F0_OST	PWM_FH2_F1_OST	PWM_FH2_F2_OST	PWM_FH2_SW_OST	PWM_FH2_F0_CBC	PWM_FH2_F1_CBC	PWM_FH2_F2_CBC	PWM_FH2_SW_CBC	
31 24	23 22	21 20	19 18	17 16	15 14	13 12	11 10	9 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

PWM_FH2_B_OST_U 发生故障事件并且定时器递增时，PWM2B 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_B_OST_D 发生故障事件并且定时器递减时，PWM2B 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_B_CBC_U 发生故障事件并且定时器递增时，PWM2B 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_B_CBC_D 发生故障事件并且定时器递减时，PWM2B 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_A_OST_U 发生故障事件并且定时器递增时，PWM2A 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_A_OST_D 发生故障事件并且定时器递减时，PWM2A 上的一次性模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_A_CBC_U 发生故障事件并且定时器递增时，PWM2A 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_A_CBC_D 发生故障事件并且定时器递减时，PWM2A 上的逐周期模式操作。0：无；1：强制拉低；2：强制拉高；3：取反。（读 / 写）

PWM_FH2_F0_OST 设置 event_f0 触发一次性模式操作。0：关闭；1：使能。（读 / 写）

PWM_FH2_F1_OST 设置 event_f1 触发一次性模式操作。0：关闭；1：使能。（读 / 写）

PWM_FH2_F2_OST 设置 event_f2 触发一次性模式操作。0：关闭；1：使能。（读 / 写）

PWM_FH2_SW_OST 软件强制一次性模式操作的使能寄存器。0：关闭；1：使能。（读 / 写）

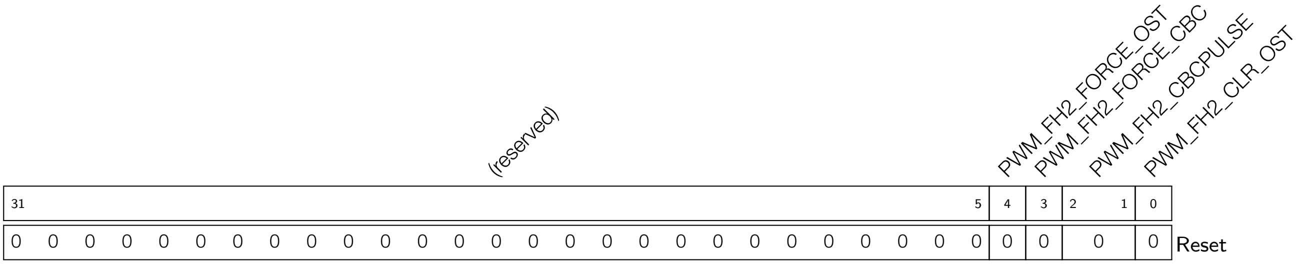
PWM_FH2_F0_CBC 设置 event_f0 触发逐周期模式操作。0：关闭；1：使能。（读 / 写）

PWM_FH2_F1_CBC 设置 event_f1 触发逐周期模式操作。0：关闭；1：使能。（读 / 写）

PWM_FH2_F2_CBC 设置 event_f2 触发逐周期模式操作。0：关闭；1：使能。（读 / 写）

PWM_FH2_SW_CBC 软件强制逐周期模式操作的使能寄存器。0：关闭；1：使能。（读 / 写）

Register 16.56. PWM_FH2_CFG1_REG (0x00dc)



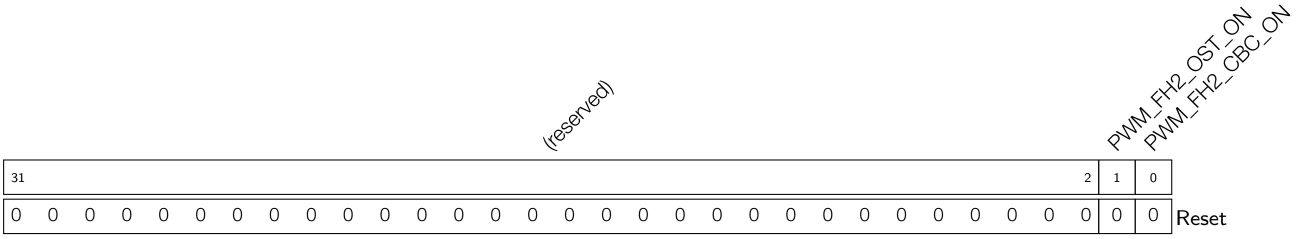
PWM_FH2_FORCE_OST 通过软件取反此位的值触发一次性模式操作。(读 / 写)

PWM_FH2_FORCE_CBC 通过软件取反此位的值触发逐周期模式操作。(读 / 写)

PWM_FH2_CBCPULSE 设置逐周期模式的更新方式。bit0 为 1:立即更新;bit1 为 1:发生 TEP 事件时更新。(读 / 写)

PWM_FH2_CLR_OST 取反此位的值清除正在进行的一次性模式的操作。(读 / 写)

Register 16.57. PWM_FH2_STATUS_REG (0x00e0)



PWM_FH2_OST_ON 由硬件置 1 和清零。置 1 时,一次性模式操作正在进行。(只读)

PWM_FH2_CBC_ON 由硬件置 1 和清零。置 1 时,逐周期模式操作正在进行。(只读)

Register 16.58. PWM_FAULT_DETECT_REG (0x00e4)

(reserved)	PWM_EVENT_F2	PWM_EVENT_F1	PWM_EVENT_F0	PWM_F2_POLE	PWM_F1_POLE	PWM_F0_POLE	PWM_F2_EN	PWM_F1_EN	PWM_F0_EN	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

PWM_EVENT_F2 由硬件置 1 和清零。置 1 时，event_f2 事件持续。（只读）

PWM_EVENT_F1 由硬件置 1 和清零。置 1 时，event_f1 事件持续。（只读）

PWM_EVENT_F0 由硬件置 1 和清零。置 1 时，event_f0 事件持续。（只读）

PWM_F2_POLE 设置来自 GPIO 矩阵的 FAULT2 信号源触发 event_f2 时极性。0：低电平触发；1：高电平触发。（读 / 写）

PWM_F1_POLE 设置来自 GPIO 矩阵的 FAULT2 信号源触发 event_f1 时极性。0：低电平触发；1：高电平触发。（读 / 写）

PWM_F0_POLE 设置来自 GPIO 矩阵的 FAULT2 信号源触发 event_f0 时极性。0：低电平触发；1：高电平触发。（读 / 写）

PWM_F2_EN 置 1 使能 event_f2 的生成。（读 / 写）

PWM_F1_EN 置 1 使能 event_f1 的生成。（读 / 写）

PWM_F0_EN 置 1 使能 event_f0 的生成。（读 / 写）

Register 16.59. PWM_CAP_TIMER_CFG_REG (0x00e8)

(reserved)	PWM_CAP_SYNC_SW	PWM_CAP_SYNCI_SEL	PWM_CAP_SYNCI_EN	PWM_CAP_TIMER_EN	
31 6	5	4 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	Reset

PWM_CAP_SYNC_SW 置 1 时，强制同步捕获定时器，捕获定时器中写入相位寄存器的值。（只写）

PWM_CAP_SYNCI_SEL 选择捕获模块的同步输入。0：无；1：定时器 0 的同步输出；2：定时器 1 的同步输出；3：定时器 2 的同步输出；4：来自 GPIO 矩阵的 SYNC0；5：来自 GPIO 矩阵的 SYNC1；6：来自 GPIO 矩阵的 SYNC2。（读 / 写）

PWM_CAP_SYNCI_EN 置 1 时，使能捕获定时器同步。（读 / 写）

PWM_CAP_TIMER_EN 置 1 时，使能捕获定时器在 APB_clk 下的递增。（读 / 写）

Register 16.60. PWM_CAP_TIMER_PHASE_REG (0x00ec)

31 0	
0	Reset

PWM_CAP_TIMER_PHASE_REG 捕获定时器同步操作的相位值。(读 / 写)

Register 16.61. PWM_CAP_CH0_CFG_REG (0x00f0)

(reserved)	PWM_CAP0_SW	PWM_CAP0_IN_INVERT	PWM_CAP0_PRESCALE	PWM_CAP0_MODE	PWM_CAP0_EN	
31 13	12	11	10 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	Reset

PWM_CAP0_SW 置 1 触发信道 0 上的软件强制捕获。(只写)

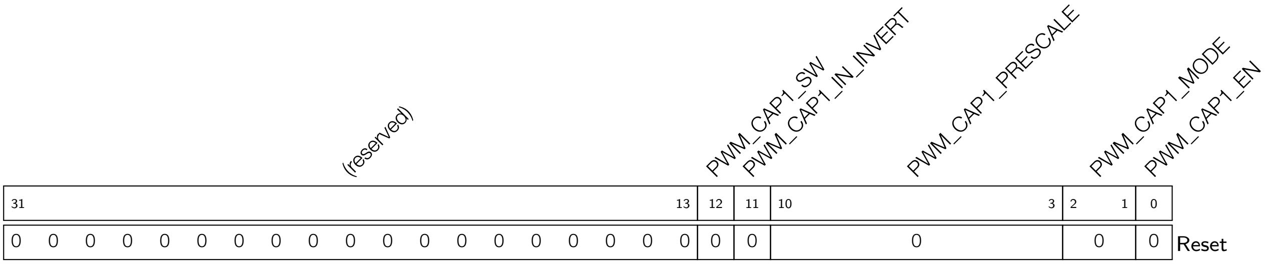
PWM_CAP0_IN_INVERT 置 1 时，来自 GPIO 矩阵的 CAP0 在预分频之前反相。(读 / 写)

PWM_CAP0_PRESCALE CAP0 上升沿的预分频值。预分频值 = PWM_CAP0_PRESCALE + 1。(读 / 写)

PWM_CAP0_MODE 预分频后信道 0 上的捕获边缘。bit0 为 1：使能下降沿捕获；bit1 为 1：使能上升沿捕获。(读 / 写)

PWM_CAP0_EN 置 1 时，使能信道 0 上的捕获。(读 / 写)

Register 16.62. PWM_CAP_CH1_CFG_REG (0x00f4)



PWM_CAP1_SW 置 1 时，触发信道 1 上的软件强制捕获事件。（只写）

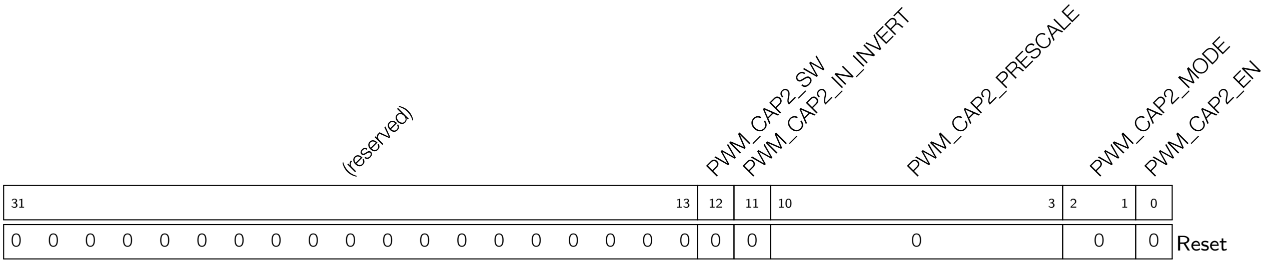
PWM_CAP1_IN_INVERT 置 1 时，来自于 GPIO 矩阵的 CAP1 在预分频前被反相。（读 / 写）

PWM_CAP1_PRESCALE CAP1 上升沿的预分频值。预分频值 = PWM_CAP1_PRESCALE + 1。（读 / 写）

PWM_CAP1_MODE 预分频后信道 1 上的捕获沿。bit0 为 1：使能下降沿捕获，bit1 为 1：使能上升沿捕获。（读 / 写）

PWM_CAP1_EN 置 1 时，使能信道 1 上的捕获事件。（读 / 写）

Register 16.63. PWM_CAP_CH2_CFG_REG (0x00f8)



PWM_CAP2_SW 置 1 触发信道 2 上的软件强制捕获事件。（只写）

PWM_CAP2_IN_INVERT 置 1 时，来自 GPIO 矩阵的 CAP2 在预分频前被反相。（读 / 写）

PWM_CAP2_PRESCALE CAP2 上升沿的预分频值。该预分频值 = PWM_CAP2_PRESCALE + 1。（读 / 写）

PWM_CAP2_MODE 预分频后信道 2 上的捕获沿。bit0：使能下降沿捕获；bit1 为 1：使能上升沿捕获。（读 / 写）

PWM_CAP2_EN 置 1 时，使能信道 2 上的捕获。（读 / 写）

Register 16.64. PWM_CAP_CH0_REG (0x00fc)

31 0	
0	Reset

PWM_CAP_CH0_REG 信道 0 上一次捕获的值。(只读)

Register 16.65. PWM_CAP_CH1_REG (0x0100)

31 0	
0	Reset

PWM_CAP_CH1_REG 信道 1 上一次捕获的值。(只读)

Register 16.66. PWM_CAP_CH2_REG (0x0104)

31 0	
0	Reset

PWM_CAP_CH2_REG 信道 2 上一次捕获的值。(只读)

Register 16.67. PWM_CAP_STATUS_REG (0x0108)

(reserved)	PWM_CAP2_EDGE	PWM_CAP1_EDGE	PWM_CAP0_EDGE	
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	Reset

PWM_CAP2_EDGE 信道 2 上一次捕获触发事件的边沿。0：上升沿；1：下降沿。(只读)

PWM_CAP1_EDGE 信道 1 上一次捕获触发事件的边沿。0：上升沿；1：下降沿。(只读)

PWM_CAP0_EDGE 信道 0 上一次捕获触发事件的边沿。0：上升沿；1：下降沿。(只读)

Register 16.68. PWM_UPDATE_CFG_REG (0x010c)

(reserved)	PWM_OP2_FORCE_UP	PWM_OP2_UP_EN	PWM_OP1_FORCE_UP	PWM_OP1_UP_EN	PWM_OP0_FORCE_UP	PWM_OP0_UP_EN	PWM_GLOBAL_FORCE_UP	PWM_GLOBAL_UP_EN	
31 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	0	1	0	1	0	1	Reset

PWM_OP2_FORCE_UP 通过软件取反此位的值将触发 PWM 操作器 2 有效寄存器强制更新。（读 / 写）

PWM_OP2_UP_EN 此位以及 PWM_GLOBAL_UP_EN 置 1 时，使能 PWM 操作器 2 有效寄存器的更新。（读 / 写）

PWM_OP1_FORCE_UP 通过软件取反此位的值将触发 PWM 操作器 1 有效寄存器强制更新。（读 / 写）

PWM_OP1_UP_EN 此位以及 PWM_GLOBAL_UP_EN 置 1 时，使能 PWM 操作器 1 有效寄存器的更新。（读 / 写）

PWM_OP0_FORCE_UP 通过软件取反此位的值将触发 PWM 操作器 0 有效寄存器强制更新。（读 / 写）

PWM_OP0_UP_EN 此位以及 PWM_GLOBAL_UP_EN 置 1 时，使能 PWM 操作器 0 有效寄存器的更新。（读 / 写）

PWM_GLOBAL_FORCE_UP 通过软件取反此位的值将触发 MCPWM 模块所有有效寄存器的强制更新。（读 / 写）

PWM_GLOBAL_UP_EN MCPWM 模块所有有效寄存器的更新使能位。（读 / 写）

Register 16.69. INT_ENA_PWM_REG (0x0110)

(reserved)		INT_CAP2_INT_ENA	INT_CAP1_INT_ENA	INT_CAP0_INT_ENA	INT_FH2_OST_INT_ENA	INT_FH1_OST_INT_ENA	INT_FH0_OST_INT_ENA	INT_FH2_CBC_INT_ENA	INT_FH1_CBC_INT_ENA	INT_FH0_CBC_INT_ENA	INT_OP2_TEB_INT_ENA	INT_OP1_TEB_INT_ENA	INT_OP0_TEB_INT_ENA	INT_OP2_TEA_INT_ENA	INT_OP1_TEA_INT_ENA	INT_OP0_TEA_INT_ENA	INT_FAULT2_CLR_INT_ENA	INT_FAULT1_CLR_INT_ENA	INT_FAULT0_CLR_INT_ENA	INT_FAULT2_INT_ENA	INT_FAULT1_INT_ENA	INT_FAULT0_INT_ENA	INT_TIMER2_TEP_INT_ENA	INT_TIMER1_TEP_INT_ENA	INT_TIMER0_TEP_INT_ENA	INT_TIMER2_TEZ_INT_ENA	INT_TIMER1_TEZ_INT_ENA	INT_TIMER0_TEZ_INT_ENA	INT_TIMER2_STOP_INT_ENA	INT_TIMER1_STOP_INT_ENA	INT_TIMER0_STOP_INT_ENA	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

INT_CAP2_INT_ENA 该位用于使能由信道 2 上的捕获事件触发的中断。(读 / 写)

INT_CAP1_INT_ENA 该位用于使能由信道 1 上的捕获事件触发的中断。(读 / 写)

INT_CAP0_INT_ENA 该位用于使能由信道 0 上的捕获事件触发的中断。(读 / 写)

INT_FH2_OST_INT_ENA 该位用于使能由 PWM2 上的一次性模式操作触发的中断。(读 / 写)

INT_FH1_OST_INT_ENA 该位用于使能由 PWM1 上的一次性模式操作触发的中断。(读 / 写)

INT_FH0_OST_INT_ENA 该位用于使能由 PWM0 上的一次性模式操作触发的中断。(读 / 写)

INT_FH2_CBC_INT_ENA 该位用于使能由 PWM2 上的逐周期模式操作触发的中断。(读 / 写)

INT_FH1_CBC_INT_ENA 该位用于使能由 PWM1 上的逐周期模式操作触发的中断。(读 / 写)

INT_FH0_CBC_INT_ENA 该位用于使能由 PWM0 上的逐周期模式操作触发的中断。(读 / 写)

INT_OP2_TEB_INT_ENA 该位用于使能由 PWM 操作器 2 TEB 事件触发的中断。(读 / 写)

INT_OP1_TEB_INT_ENA 该位用于使能由 PWM 操作器 2 TEB 事件触发的中断。(读 / 写)

INT_OP0_TEB_INT_ENA 该位用于使能由 PWM 操作器 0 TEB 事件触发的中断。(读 / 写)

INT_OP2_TEA_INT_ENA 该位用于使能由 PWM 操作器 2 TEA 事件触发的中断。(读 / 写)

INT_OP1_TEA_INT_ENA 该位用于使能由 PWM 操作器 1 TEA 事件触发的中断。(读 / 写)

INT_OP0_TEA_INT_ENA 该位用于使能由 PWM 操作器 0 TEA 事件触发的中断。(读 / 写)

INT_FAULT2_CLR_INT_ENA 该位用于使能 event_f2 结束后触发的中断。(读 / 写)

INT_FAULT1_CLR_INT_ENA 该位用于使能 event_f1 结束后触发的中断。(读 / 写)

INT_FAULT0_CLR_INT_ENA 该位用于使能 event_f0 结束后触发的中断。(读 / 写)

INT_FAULT2_INT_ENA 该位用于使能 event_f2 开始时触发的中断。(读 / 写)

INT_FAULT1_INT_ENA 该位用于使能 event_f1 开始时触发的中断。(读 / 写)

INT_FAULT0_INT_ENA 该位用于使能 event_f0 开始时触发的中断。(读 / 写)

INT_TIMER2_TEP_INT_ENA 该位用于使能 PWM 定时器 2 TEP 事件触发的中断。(读 / 写)

INT_TIMER1_TEP_INT_ENA 该位用于使能 PWM 定时器 1 TEP 事件触发的中断。(读 / 写)

INT_TIMER0_TEP_INT_ENA 该位用于使能 PWM 定时器 0 TEP 事件触发的中断。(读 / 写)

INT_TIMER2_TEZ_INT_ENA 该位用于使能 PWM 定时器 2 TEZ 事件触发的中断。(读 / 写)

INT_TIMER1_TEZ_INT_ENA 该位用于使能 PWM 定时器 1 TEZ 事件触发的中断。(读 / 写)

INT_TIMER0_TEZ_INT_ENA 该位用于使能 PWM 定时器 0 TEZ 事件触发的中断。(读 / 写)

INT_TIMER2_STOP_INT_ENA 该位用于使能定时器 2 停止时触发的中断。(读 / 写)

INT_TIMER1_STOP_INT_ENA 该位用于使能定时器 1 停止时触发的中断。(读 / 写)

INT_TIMER0_STOP_INT_ENA 该位用于使能定时器 0 停止时触发的中断。(读 / 写)

Register 16.70. INT_RAW_PWM_REG (0x0114)

	(reserved)	INT_CAP2_INT_RAW	INT_CAP1_INT_RAW	INT_CAP0_INT_RAW	INT_FH2_OST_INT_RAW	INT_FH1_OST_INT_RAW	INT_FH0_OST_INT_RAW	INT_FH2_CBC_INT_RAW	INT_FH1_CBC_INT_RAW	INT_FH0_CBC_INT_RAW	INT_OP2_TEB_INT_RAW	INT_OP1_TEB_INT_RAW	INT_OP0_TEB_INT_RAW	INT_OP2_TEA_INT_RAW	INT_OP1_TEA_INT_RAW	INT_OP0_TEA_INT_RAW	INT_FAULT2_CLR_INT_RAW	INT_FAULT1_CLR_INT_RAW	INT_FAULT0_CLR_INT_RAW	INT_FAULT2_INT_RAW	INT_FAULT1_INT_RAW	INT_FAULT0_INT_RAW	INT_TIMER2_TEP_INT_RAW	INT_TIMER1_TEP_INT_RAW	INT_TIMER0_TEP_INT_RAW	INT_TIMER2_TEZ_INT_RAW	INT_TIMER1_TEZ_INT_RAW	INT_TIMER0_TEZ_INT_RAW	INT_TIMER2_STOP_INT_RAW	INT_TIMER1_STOP_INT_RAW	INT_TIMER0_STOP_INT_RAW
	31 30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reset	0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INT_CAP2_INT_RAW 由信道 2 上捕获事件触发的中断的原始状态位。(只读)

INT_CAP1_INT_RAW 由信道 1 上捕获事件触发的中断的原始状态位。(只读)

INT_CAP0_INT_RAW 由信道 0 上捕获事件触发的中断的原始状态位。(只读)

INT_FH2_OST_INT_RAW 由 PWM2 一次性操作触发的中断的原始状态位。(只读)

INT_FH1_OST_INT_RAW 由 PWM1 一次性操作触发的中断的原始状态位。(只读)

INT_FH0_OST_INT_RAW 由 PWM0 一次性操作触发的中断的原始状态位。(只读)

INT_FH2_CBC_INT_RAW 由 PWM2 逐周期操作触发的中断的原始状态位。(只读)

INT_FH1_CBC_INT_RAW 由 PWM1 逐周期操作触发的中断的原始状态位。(只读)

INT_FH0_CBC_INT_RAW 由 PWM0 逐周期操作触发的中断的原始状态位。(只读)

INT_OP2_TEB_INT_RAW 由 PWM 操作器 2 TEB 事件触发的中断的原始状态位。(只读)

INT_OP1_TEB_INT_RAW 由 PWM 操作器 1 TEB 事件触发的中断的原始状态位。(只读)

INT_OP0_TEB_INT_RAW 由 PWM 操作器 0 TEB 事件触发的中断的原始状态位。(只读)

INT_OP2_TEA_INT_RAW 由 PWM 操作器 2 TEA 事件触发的中断的原始状态位。(只读)

INT_OP1_TEA_INT_RAW 由 PWM 操作器 1 TEA 事件触发的中断的原始状态位。(只读)

INT_OP0_TEA_INT_RAW 由 PWM 操作器 0 TEA 事件触发的中断的原始状态位。(只读)

INT_FAULT2_CLR_INT_RAW event_f2 结束时触发的中断的原始状态位。(只读)

INT_FAULT1_CLR_INT_RAW event_f1 结束时触发的中断的原始状态位。(只读)

INT_FAULT0_CLR_INT_RAW event_f0 结束时触发的中断的原始状态位。(只读)

INT_FAULT2_INT_RAW event_f2 开始后触发的中断的原始状态位。(只读)

INT_FAULT1_INT_RAW event_f1 开始后触发的中断的原始状态位。(只读)

INT_FAULT0_INT_RAW event_f0 开始后触发的中断的原始状态位。(只读)

INT_TIMER2_TEP_INT_RAW PWM 定时器 2 TEP 事件触发的中断的原始状态位。(只读)

INT_TIMER1_TEP_INT_RAW PWM 定时器 1 TEP 事件触发的中断的原始状态位。(只读)

INT_TIMER0_TEP_INT_RAW PWM 定时器 0 TEP 事件触发的中断的原始状态位。(只读)

INT_TIMER2_TEZ_INT_RAW PWM 定时器 2 TEZ 事件触发的中断的原始状态位。(只读)

INT_TIMER1_TEZ_INT_RAW PWM 定时器 1 TEP 事件触发的中断的原始状态位。(只读)

INT_TIMER0_TEZ_INT_RAW PWM 定时器 0 TEP 事件触发的中断的原始状态位。(只读)

INT_TIMER2_STOP_INT_RAW 定时器 2 停止后触发的中断的原始状态位。(只读)

INT_TIMER1_STOP_INT_RAW 定时器 1 停止后触发的中断的原始状态位。(只读)

INT_TIMER0_STOP_INT_RAW 定时器 0 停止后触发的中断的原始状态位。(只读)

Register 16.71. INT_ST_PWM_REG (0x0118)

(reserved)		INT_CAP2_INT_ST	INT_CAP1_INT_ST	INT_CAP0_INT_ST	INT_FH2_OST_INT_ST	INT_FH1_OST_INT_ST	INT_FH0_OST_INT_ST	INT_FH2_CBC_INT_ST	INT_FH1_CBC_INT_ST	INT_FH0_CBC_INT_ST	INT_OP2_TEB_INT_ST	INT_OP1_TEB_INT_ST	INT_OP0_TEB_INT_ST	INT_OP2_TEA_INT_ST	INT_OP1_TEA_INT_ST	INT_OP0_TEA_INT_ST	INT_FAULT2_CLR_INT_ST	INT_FAULT1_CLR_INT_ST	INT_FAULT0_CLR_INT_ST	INT_FAULT2_INT_ST	INT_FAULT1_INT_ST	INT_FAULT0_INT_ST	INT_TIMER2_TEP_INT_ST	INT_TIMER1_TEP_INT_ST	INT_TIMER0_TEP_INT_ST	INT_TIMER2_TEZ_INT_ST	INT_TIMER1_TEZ_INT_ST	INT_TIMER0_TEZ_INT_ST	INT_TIMER2_STOP_INT_ST	INT_TIMER1_STOP_INT_ST	INT_TIMER0_STOP_INT_ST	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

INT_CAP2_INT_ST 信道 2 上捕获事件触发的中断的屏蔽状态位。(只读)

INT_CAP1_INT_ST 信道 1 上捕获事件触发的中断的屏蔽状态位。(只读)

INT_CAP0_INT_ST 信道 0 上捕获事件触发的中断的屏蔽状态位。(只读)

INT_FH2_OST_INT_ST PWM2 一次性操作触发的中断的屏蔽状态位。(只读)

INT_FH1_OST_INT_ST PWM1 一次性操作触发的中断的屏蔽状态位。(只读)

INT_FH0_OST_INT_ST PWM0 一次性操作触发的中断的屏蔽状态位。(只读)

INT_FH2_CBC_INT_ST PWM2 逐周期操作触发的中断的屏蔽状态位。(只读)

INT_FH1_CBC_INT_ST PWM1 逐周期操作触发的中断的屏蔽状态位。(只读)

INT_FH0_CBC_INT_ST PWM0 逐周期操作触发的中断的屏蔽状态位。(只读)

INT_OP2_TEB_INT_ST PWM 操作器 2 TEB 事件触发的中断的屏蔽状态位。(只读)

INT_OP1_TEB_INT_ST PWM 操作器 1 TEB 事件触发的中断的屏蔽状态位。(只读)

INT_OP0_TEB_INT_ST PWM 操作器 0 TEB 事件触发的中断的屏蔽状态位。(只读)

INT_OP2_TEA_INT_ST PWM 操作器 2 TEA 事件触发的中断的屏蔽状态位。(只读)

INT_OP1_TEA_INT_ST PWM 操作器 1 TEA 事件触发的中断的屏蔽状态位。(只读)

INT_OP0_TEA_INT_ST PWM 操作器 0 TEA 事件触发的中断的屏蔽状态位。(只读)

INT_FAULT2_CLR_INT_ST event_f2 结束时触发的中断的屏蔽状态位。(只读)

INT_FAULT1_CLR_INT_ST event_f1 结束时触发的中断的屏蔽状态位。(只读)

INT_FAULT0_CLR_INT_ST event_f0 结束时触发的中断的屏蔽状态位。(只读)

INT_FAULT2_INT_ST event_f2 开始时触发的中断的屏蔽状态位。(只读)

INT_FAULT1_INT_ST event_f1 开始时触发的中断的屏蔽状态位。(只读)

INT_FAULT0_INT_ST event_f0 开始时触发的中断的屏蔽状态位。(只读)

INT_TIMER2_TEP_INT_ST PWM 定时器 2 TEP 事件触发的中断的屏蔽状态位。(只读)

INT_TIMER1_TEP_INT_ST PWM 定时器 1 TEP 事件触发的中断的屏蔽状态位。(只读)

INT_TIMER0_TEP_INT_ST PWM 定时器 0 TEP 事件触发的中断的屏蔽状态位。(只读)

INT_TIMER2_TEZ_INT_ST PWM 定时器 2 TEZ 事件触发的中断的屏蔽状态位。(只读)

INT_TIMER1_TEZ_INT_ST PWM 定时器 1 TEZ 事件触发的中断的屏蔽状态位。(只读)

INT_TIMER0_TEZ_INT_ST PWM 定时器 0 TEZ 事件触发的中断的屏蔽状态位。(只读)

INT_TIMER2_STOP_INT_ST 定时器 2 停止后触发的中断的屏蔽状态位。(只读)

INT_TIMER1_STOP_INT_ST 定时器 1 停止后触发的中断的屏蔽状态位。(只读)

INT_TIMER0_STOP_INT_ST 定时器 0 停止后触发的中断的屏蔽状态位。(只读)

Register 16.72. INT_CLR_PWM_REG (0x011c)

(reserved)		INT_CAP2_INT_CLR	INT_CAP1_INT_CLR	INT_CAP0_INT_CLR	INT_FH2_OST_INT_CLR	INT_FH1_OST_INT_CLR	INT_FH0_OST_INT_CLR	INT_FH2_CBC_INT_CLR	INT_FH1_CBC_INT_CLR	INT_FH0_CBC_INT_CLR	INT_OP2_TEB_INT_CLR	INT_OP1_TEB_INT_CLR	INT_OP0_TEB_INT_CLR	INT_OP2_TEA_INT_CLR	INT_OP1_TEA_INT_CLR	INT_OP0_TEA_INT_CLR	INT_FAULT2_CLR_INT_CLR	INT_FAULT1_CLR_INT_CLR	INT_FAULT0_CLR_INT_CLR	INT_FAULT2_INT_CLR	INT_FAULT1_INT_CLR	INT_FAULT0_INT_CLR	INT_TIMER2_TEP_INT_CLR	INT_TIMER1_TEP_INT_CLR	INT_TIMER0_TEP_INT_CLR	INT_TIMER2_TEZ_INT_CLR	INT_TIMER1_TEZ_INT_CLR	INT_TIMER0_TEZ_INT_CLR	INT_TIMER2_STOP_INT_CLR	INT_TIMER1_STOP_INT_CLR	INT_TIMER0_STOP_INT_CLR	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

INT_CAP2_INT_CLR 置 1 时清除信道 2 上捕获事件触发的中断。(只写)

INT_CAP1_INT_CLR 置 1 时清除信道 1 上捕获事件触发的中断。(只写)

INT_CAP0_INT_CLR 置 1 时清除信道 0 上捕获事件触发的中断。(只写)

INT_FH2_OST_INT_CLR 置 1 时清除 PWM2 一次性操作触发的中断。(只写)

INT_FH1_OST_INT_CLR 置 1 时清除 PWM1 一次性操作触发的中断。(只写)

INT_FH0_OST_INT_CLR 置 1 时清除 PWM0 一次性操作触发的中断。(只写)

INT_FH2_CBC_INT_CLR 置 1 时清除 PWM2 逐周期操作触发的中断。(只写)

INT_FH1_CBC_INT_CLR 置 1 时清除 PWM1 逐周期操作触发的中断。(只写)

INT_FH0_CBC_INT_CLR 置 1 时清除 PWM0 逐周期操作触发的中断。(只写)

INT_OP2_TEB_INT_CLR 置 1 时清除 PWM 操作器 2 TEB 事件触发的中断。(只写)

INT_OP1_TEB_INT_CLR 置 1 时清除 PWM 操作器 1 TEB 事件触发的中断。(只写)

INT_OP0_TEB_INT_CLR 置 1 时清除 PWM 操作器 0 TEB 事件触发的中断。(只写)

INT_OP2_TEA_INT_CLR 置 1 时清除 PWM 操作器 2 TEA 事件触发的中断。(只写)

INT_OP1_TEA_INT_CLR 置 1 时清除 PWM 操作器 1 TEA 事件触发的中断。(只写)

INT_OP0_TEA_INT_CLR 置 1 时清除 PWM 操作器 0 TEA 事件触发的中断。(只写)

INT_FAULT2_CLR_INT_CLR 置 1 时清除 event_f2 结束时触发的中断。(只写)

INT_FAULT1_CLR_INT_CLR 置 1 时清除 event_f1 结束时触发的中断。(只写)

INT_FAULT0_CLR_INT_CLR 置 1 时清除 event_f0 结束时触发的中断。(只写)

INT_FAULT2_INT_CLR 置 1 时清除 event_f2 开始时触发的中断。(只写)

INT_FAULT1_INT_CLR 置 1 时清除 event_f1 开始时触发的中断。(只写)

INT_FAULT0_INT_CLR 置 1 时清除 event_f0 开始时触发的中断。(只写)

INT_TIMER2_TEP_INT_CLR 置 1 时清除 PWM 定时器 2 TEP 事件触发的中断。(只写)

INT_TIMER1_TEP_INT_CLR 置 1 时清除 PWM 定时器 1 TEP 事件触发的中断。(只写)

INT_TIMER0_TEP_INT_CLR 置 1 时清除 PWM 定时器 0 TEP 事件触发的中断。(只写)

INT_TIMER2_TEZ_INT_CLR 置 1 时清除 PWM 定时器 2 TEZ 事件触发的中断。(只写)

INT_TIMER1_TEZ_INT_CLR 置 1 时清除 PWM 定时器 1 TEZ 事件触发的中断。(只写)

INT_TIMER0_TEZ_INT_CLR 置 1 时清除 PWM 定时器 0 TEZ 事件触发的中断。(只写)

INT_TIMER2_STOP_INT_CLR 置 1 时清除定时器 2 停止后触发的中断。(只写)

INT_TIMER1_STOP_INT_CLR 置 1 时清除定时器 1 停止后触发的中断。(只写)

INT_TIMER0_STOP_INT_CLR 置 1 时清除定时器 0 停止后触发的中断。(只写)

17 脉冲计数器 (PCNT)

17.1 概述

脉冲计数器模块用于对输入脉冲的上升沿或下降沿进行计数。每个脉冲计数器单元均有一个带符号的 16-bit 计数寄存器以及两个通道，通过配置可以加减计数器。每个通道均有一个脉冲输入信号以及一个能够用于控制输入信号的控制信号。输入信号可以打开或关闭滤波功能。

脉冲计数器有 8 组单元，各自独立工作，命名为 PULSE_CNT_U*n*。

ESP32 脉冲计数器支持的最大脉冲频率为 40 MHz。

17.2 功能描述

17.2.1 架构图

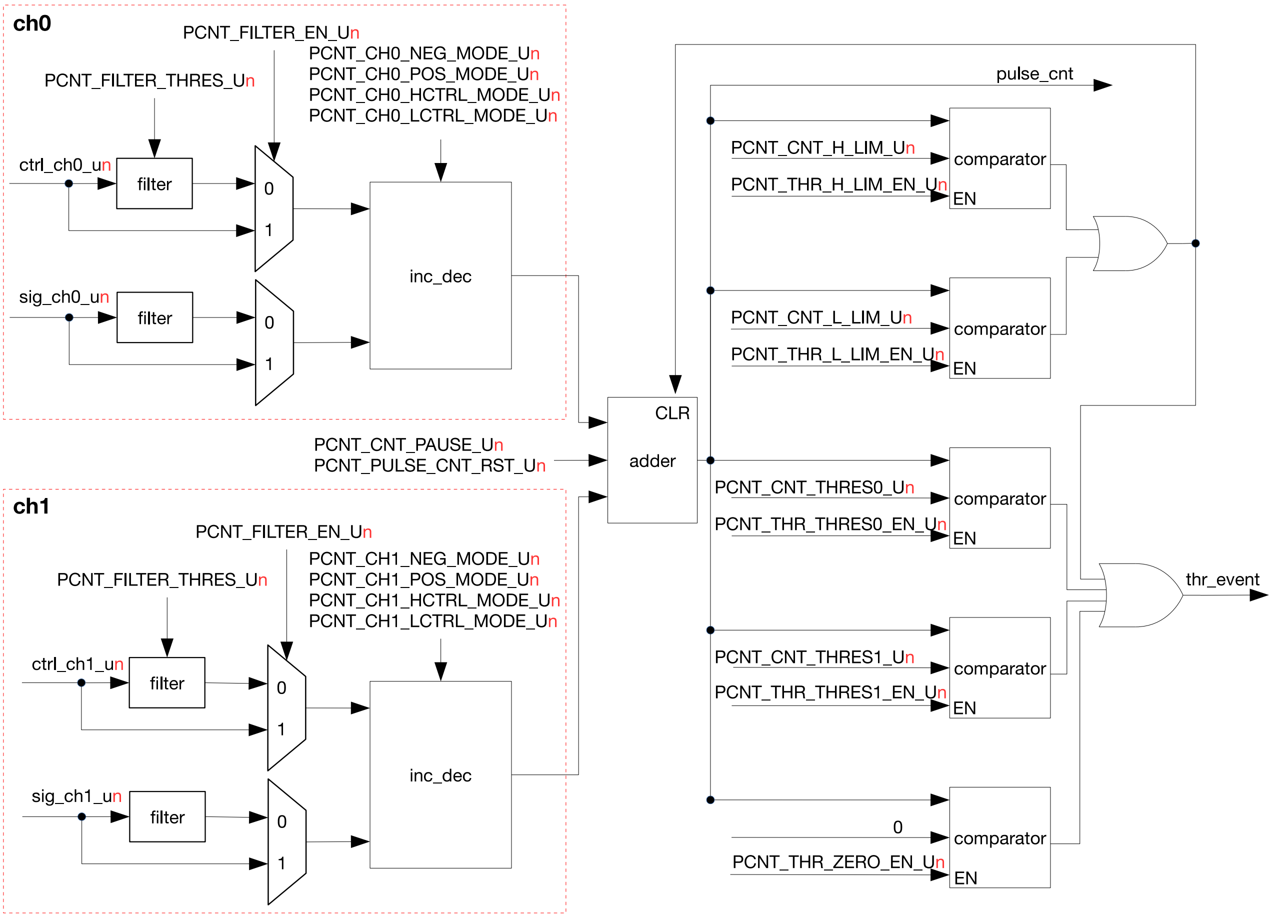


图 17-1. PULSE_CNT 单元基本架构图

图 17-1 为脉冲计数器的基本架构图。每个单元有两个通道：ch0 和 ch1。这两个通道的功能相似。每个通道均有一个输入信号和一个控制输入信号，都能连接到芯片引脚。上升沿和下降沿中的计数工作模式可以分别进行增加、不增不减或者减少计数值的配置行为。对控制信号而言，通过配置硬件可以更改上升沿和下降沿的工作模式，包括：反转、禁止和保持。该计数器本身是一个带符号的 16-bit 加减计数器。它的值可以由软件直接读取，硬件通过将该值与一组比较器进行比较，可以产生中断。

17.2.2 计数器通道输入信号

如上文所述，一个通道里的两组输入信号能够以多种方式影响脉冲计数器：LCTRL_MODE 和 HCTRL_MODE 分别用于配置低控制信号和高控制信号；POS_MODE 和 NEG_MODE 分别用于配置输入信号的上升沿和下降沿。POS_MODE 和 NEG_MODE 配置为 1，计数器递增；若将它们配置为 2 时，则计数器递减；其它的值表示计数器保持原始值，既不递增，也不递减。当 LCTRL_MODE 或 HCTRL_MODE 为 0，表示不修改 NEG_MODE 和 POS_MODE 的工作模式；为 1 表示反转（即若原来计数器处于递增状态，当配置 POS_MODE 或 NEG_MODE 为 1 后，计数器将处于递减状态，反之亦然）；其它的值会禁止计数器计数作用。

下表列出了一些关于上升沿对计数器作用的例子，包括低/高电平控制信号以及各种配置选择。为了清晰可见，下表数值后面的括号内添加了一些描述，x 代表了“无关项”。

POS_ MODE	LCTRL_ MODE	HCTRL_ MODE	sig l→h when ctrl=0	sig l→h when ctrl=1
1 (inc)	0 (-)	0 (-)	Inc ctr	Inc ctr
2 (dec)	0 (-)	0 (-)	Dec ctr	Dec ctr
0 (-)	x	x	No action	No action
1 (inc)	0 (-)	1 (inv)	Inc ctr	Dec ctr
1 (inc)	1 (inv)	0 (-)	Dec ctr	Inc ctr
2 (dec)	0 (-)	1 (inv)	Dec ctr	Inc ctr
1 (inc)	0 (-)	2 (dis)	Inc ctr	No action
1 (inc)	2 (dis)	0 (-)	No action	Inc ctr

该表对下降沿 (sig h→l) 也同样适用，用 NEG_MODE 来代替 POS_MODE。

每个脉冲计数器单元在这 4 个输入中均有一个滤波器，可以滤除噪声。单元的 4 个输入信号可以通过置位 PCNT_FILTER_EN_U*n* 来打开滤波功能。一旦滤波器被启动，任何宽度比 REG_FILTER_THRES_U*n* 个时钟周期窄的脉冲都会被过滤掉，这些被过滤掉的脉冲将不会对计数器起任何作用。

除了输入通道以外，软件也能对计数器进行一部分控制。比如通过置位 PCNT_CNT_PAUSE_U*n*，可以暂停计数器。通过置位 PCNT_PLUS_CNT_RST_U*n* 实现计数器清零功能。

17.2.3 观察点

PULSE_CNT 可以设置 5 个观察点，5 个观察点共用一个中断，可以通过各自的中断使能信号开启或屏蔽中断。这些观察点分别是：

- 最大计数值。当 PULSE_CNT 大于等于 PCNT_CNT_H_LIM_U*n* 时，清空 PULSE_CNT。其中 PCNT_CNT_H_LIM_U*n* 应设为正数。
- 最小计数值。当 PULSE_CNT 小于等于 PCNT_CNT_L_LIM_U*n* 时，清空 PULSE_CNT。其中 PCNT_CNT_L_LIM_U*n* 应设为负数。
- 两个中间阈值。当 PULSE_CNT 等于 PCNT_THR_THRES0_U*n* 或者 PCNT_THR_THRES1_U*n* 时，产生相应的 thr_event 信号。
- 零。当 PULSE_CNT 等于 0 时，产生相应的 thr_event 信号。

17.2.4 举例

图 17-2 为仅仅使用通道 0 进行递增计数的示意图，通道 0 的配置如下所示。

- CNT_CH0_POS_MODE_U*n*=1，即在 sig_ch0_u*n* 的上升沿进行递增计数。

- PCNT_CH0_NEG_MODE_U*n*=0，即在 sig_ch0_u*n* 的下升沿不进行计数。
- PCNT_CH0_LCTRL_MODE_U*n*=0，即当 ctrl_ch0_u*n* 为低电平时，对输入 sig_ch0_u*n* 进行递增控制。
- PCNT_CH0_HCTRL_MODE_U*n*=2，即当 ctrl_ch0_u*n* 为高电平时，对输入 sig_ch0_u*n* 不进行控制。
- PCNT_CNT_H_LIM_U*n*=5，当 PULSE_CNT 的值递增到 PCNT_CNT_H_LIM_U*n*，PULSE_CNT 返回到 0。

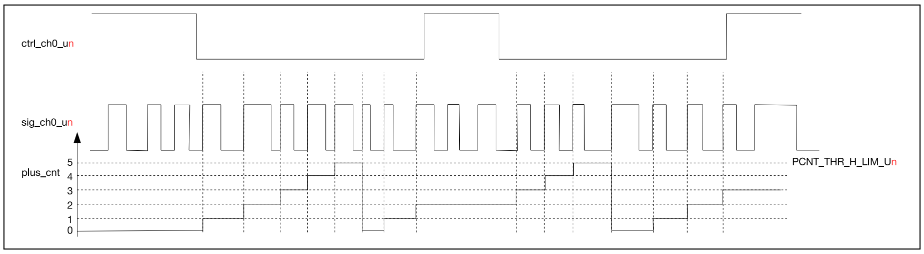


图 17-2. PULSE_CNT 递增计数图

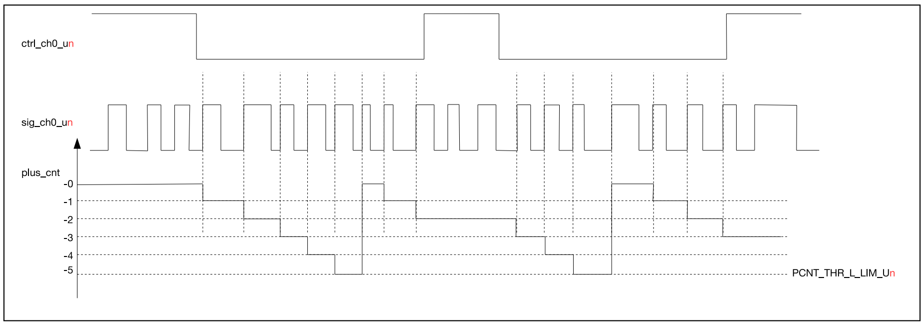


图 17-3. PULSE_CNT 递减计数图

图 17-3 为仅仅使用通道 0 进行递减计数的示意图，图 17-3 中通道 0 的配置与图 17-2 中的配置区别为：

- PCNT_CH0_LCTRL_MODE_U*n*=1，即当 ctrl_ch0_u*n* 为低电平时，对输入 sig_ch0_u*n* 进行递减控制。
- PCNT_CNT_H_LIM_U*n*=–5，当 PULSE_CNT 的值递减到 PCNT_CNT_H_LIM_U*n*，PULSE_CNT 返回到 0。

17.2.5 溢出中断

PCNT_CNT_THR_EVENT_U*n*_IN：该中断有 5 个中断源，即一个最大计数值中断，一个最小计数值中断，两个中间阈值中断以及一个过零中断，它们可以通过各自的中断使能信号开启或屏蔽中断。

17.3 寄存器列表

名称	描述	地址	访问
配置寄存器			
PCNT_U0_CONF0_REG	单元 0 的配置寄存器 0	0x3FF57000	读/写
PCNT_U1_CONF0_REG	单元 1 的配置寄存器 0	0x3FF5700C	读/写
PCNT_U2_CONF0_REG	单元 2 的配置寄存器 0	0x3FF57018	读/写
PCNT_U3_CONF0_REG	单元 3 的配置寄存器 0	0x3FF57024	读/写
PCNT_U4_CONF0_REG	单元 4 的配置寄存器 0	0x3FF57030	读/写
PCNT_U5_CONF0_REG	单元 5 的配置寄存器 0	0x3FF5703C	读/写
PCNT_U6_CONF0_REG	单元 6 的配置寄存器 0	0x3FF57048	读/写

名称	描述	地址	访问
PCNT_U7_CONF0_REG	单元 7 的配置寄存器 0	0x3FF57054	读/写
PCNT_U0_CONF1_REG	单元 0 的配置寄存器 1	0x3FF57004	读/写
PCNT_U1_CONF1_REG	单元 1 的配置寄存器 1	0x3FF57010	读/写
PCNT_U2_CONF1_REG	单元 2 的配置寄存器 1	0x3FF5701C	读/写
PCNT_U3_CONF1_REG	单元 3 的配置寄存器 1	0x3FF57028	读/写
PCNT_U4_CONF1_REG	单元 4 的配置寄存器 1	0x3FF57034	读/写
PCNT_U5_CONF1_REG	单元 5 的配置寄存器 1	0x3FF57040	读/写
PCNT_U6_CONF1_REG	单元 6 的配置寄存器 1	0x3FF5704C	读/写
PCNT_U7_CONF1_REG	单元 7 的配置寄存器 1	0x3FF57058	读/写
PCNT_U0_CONF2_REG	单元 0 的配置寄存器 2	0x3FF57008	读/写
PCNT_U1_CONF2_REG	单元 1 的配置寄存器 2	0x3FF57014	读/写
PCNT_U2_CONF2_REG	单元 2 的配置寄存器 2	0x3FF57020	读/写
PCNT_U3_CONF2_REG	单元 3 的配置寄存器 2	0x3FF5702C	读/写
PCNT_U4_CONF2_REG	单元 4 的配置寄存器 2	0x3FF57038	读/写
PCNT_U5_CONF2_REG	单元 5 的配置寄存器 2	0x3FF57044	读/写
PCNT_U6_CONF2_REG	单元 6 的配置寄存器 2	0x3FF57050	读/写
PCNT_U7_CONF2_REG	单元 7 的配置寄存器 2	0x3FF5705C	读/写
计数器值			
PCNT_U0_CNT_REG	单元 0 的计数值	0x3FF57060	只读
PCNT_U1_CNT_REG	单元 1 的计数值	0x3FF57064	只读
PCNT_U2_CNT_REG	单元 2 的计数值	0x3FF57068	只读
PCNT_U3_CNT_REG	单元 3 的计数值	0x3FF5706C	只读
PCNT_U4_CNT_REG	单元 4 的计数值	0x3FF57070	只读
PCNT_U5_CNT_REG	单元 5 的计数值	0x3FF57074	只读
PCNT_U6_CNT_REG	单元 6 的计数值	0x3FF57078	只读
PCNT_U7_CNT_REG	单元 7 的计数值	0x3FF5707C	只读
控制寄存器			
PCNT_CTRL_REG	所有计数器的控制寄存器	0x3FF570B0	读/写
中断寄存器			
PCNT_INT_RAW_REG	原始中断状态	0x3FF57080	只读
PCNT_INT_ST_REG	屏蔽中断状态	0x3FF57084	只读
PCNT_INT_ENA_REG	中断使能位	0x3FF57088	读/写
PCNT_INT_CLR_REG	中断清除位	0x3FF5708C	只写
状态寄存器			
PCNT_U*n*_STATUS_REG	指示计数器的状态	0x3FF57090	只读

17.4 寄存器

本小节括号中的地址均为相对于 PCNT 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 17.3 寄存器列表。

Register 17.1. PCNT_U*n*_CONF0_REG (*n*: 0-7) (0x0+0x0C**n*)

PCNT_CH1_LCTRL_MODE_U*n*	PCNT_CH1_HCTRL_MODE_U*n*	PCNT_CH1_POS_MODE_U*n*	PCNT_CH1_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_NEG_MODE_U*n*	PCNT_THR_THRES1_EN_U*n*	PCNT_THR_THRES0_EN_U*n*	PCNT_THR_L_LIM_EN_U*n*	PCNT_THR_H_LIM_EN_U*n*	PCNT_THR_ZERO_EN_U*n*	PCNT_FILTER_EN_U*n*	PCNT_FILTER_THRES_U*n*	
31 30	29 28	27 26	25 24	23 22	21 20	19 18	17 16	15	14	13	12	11	10	9 0	
0	0	0	0	0	0	0	0	0	0	1	1	1	1	0x010	Reset

PCNT_CH1_LCTRL_MODE_U*n* 当控制信号为低电平时，用于改变 CH1_POS_MODE/CH1_NEG_MODE 的设置。(读/写)
0：不做修改；1：反转（增加转为减少，减少转为增加）；2, 3：禁止计数器计数。

PCNT_CH1_HCTRL_MODE_U*n* 当控制信号为高电平时，用于改变 CH1_POS_MODE/CH1_NEG_MODE 的设置。(读/写)
0：不做修改；1：反转（增加转为减少，减少转为增加）；2, 3：禁止计数器计数。

PCNT_CH1_POS_MODE_U*n* 用于设置通道 1 检测输入信号上升沿的工作模式。(读/写)
1：增加计数器；2：减少计数器；0, 3: 对计数器无任何影响。

PCNT_CH1_NEG_MODE_U*n* 用于设置通道 1 检测输入信号下降沿的工作模式。(读/写)
1：增加计数器；2：减少计数器；0, 3: 对计数器无任何影响。

PCNT_CH0_LCTRL_MODE_U*n* 当控制信号为低电平时，用于配置 CH0_POS_MODE/CH0_NEG_MODE 的设置修改方式。(读/写)
0：不做修改；1：反转（增加转为减少，减少转为增加）；2, 3：禁止计数修改。

PCNT_CH0_HCTRL_MODE_U*n* 当控制信号为高电平时，用于配置 CH0_POS_MODE/CH0_NEG_MODE 的设置修改方式。(读/写)
0：不做修改；1：反转（增加转为减少，减少转为增加）；2, 3：禁止计数修改。

PCNT_CH0_POS_MODE_U*n* 用于设置通道 1 检测输入信号上升沿的工作模式。(读/写)
1：增加计数器；2：减少计数器；0, 3: 对计数器无任何影响。

PCNT_CH0_NEG_MODE_U*n* T 当通道 0 的输入信号探测到一个下降沿时，用于设置工作模式。(读/写) 1：增加计数器；2：减少计数器；0, 3: 对计数器无任何影响。

PCNT_THR_THRES1_EN_U*n* 为单元 *n* 的阈值 1 比较器的使能位。(读/写)

PCNT_THR_THRES0_EN_U*n* 为单元 *n* 的阈值 0 比较器的使能位。(读/写)

PCNT_THR_L_LIM_EN_U*n* 为单元 *n* 的阈值 1 比较器的使能位。(读/写)

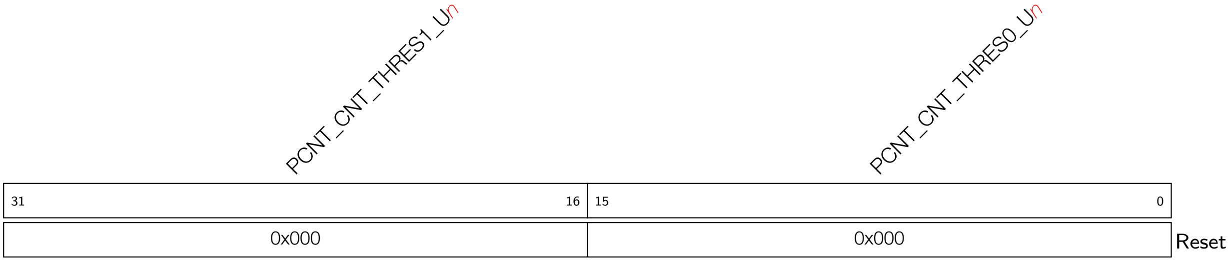
PCNT_THR_H_LIM_EN_U*n* 为单元 *n* 的 thr_h_lim 比较器的使能位。(读/写)

PCNT_THR_ZERO_EN_U*n* 为单元 *n* 的过零比较器的使能位。(读/写)

PCNT_FILTER_EN_U*n* 为单元 *n* 的输入滤波器的使能位。(读/写)

PCNT_FILTER_THRES_U*n* 在 APB_CLK 个时钟周期内为滤波器设置最大阈值。在滤波器启动时，任何比 APB_CLK 个时钟周期窄的脉冲都将被过滤掉。(读/写)

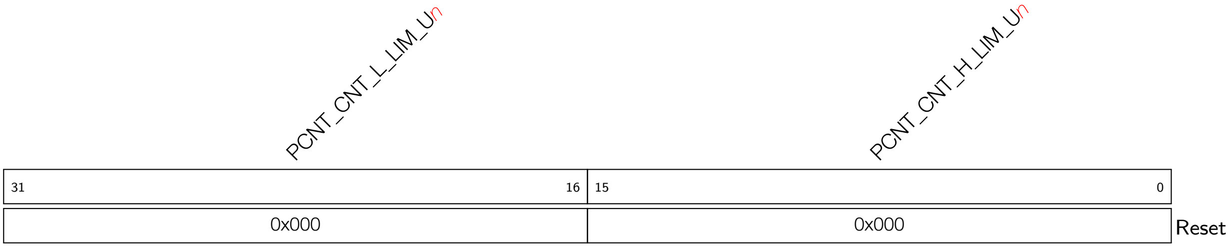
Register 17.2. PCNT_U*n*_CONF1_REG (*n*: 0-7) (0x4+0x0Cn*)**



PCNT_CNT_THRES1_U*n* 用于配置单元 *n* 的阈值 1 的值。（读/写）

PCNT_CNT_THRES0_U*n* 用于配置单元 *n* 的阈值 0 的值。（读/写）

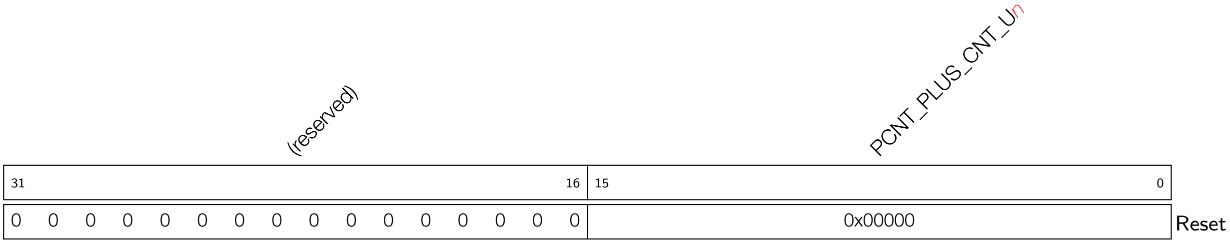
Register 17.3. PCNT_U*n*_CONF2_REG (*n*: 0-7) (0x8+0x0Cn*)**



PCNT_CNT_L_LIM_U*n* 用于配置单元 *n* 的计数器的最小值。（读/写）

PCNT_CNT_H_LIM_U*n* 用于配置单元 *n* 的计数器的最大值。（读/写）

Register 17.4. PCNT_U*n*_CNT_REG (*n*: 0-7) (0x28+0x0Cn*)**



PCNT_PLUS_CNT_U*n* 该寄存器存储着单元 *n* 的当前计数器的值，可供软件读取。（只读）

Register 17.5. PCNT_INT_RAW_REG (0x0080)

(reserved)	PCNT_CNT_THR_EVENT_U7_INT_RAW	PCNT_CNT_THR_EVENT_U6_INT_RAW	PCNT_CNT_THR_EVENT_U5_INT_RAW	PCNT_CNT_THR_EVENT_U4_INT_RAW	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 8	7	6	5	4	3	2	1	0	
0x0000000	0	0	0	0	0	0	0	0	Reset

PCNT_CNT_THR_EVENT_U*n*_INT_RAW PCNT_CNT_THR_EVENT_U*n*_INT 中断的原始中断状态位。（只读）

Register 17.6. PCNT_INT_ST_REG (0x0084)

(reserved)	PCNT_CNT_THR_EVENT_U7_INT_ST	PCNT_CNT_THR_EVENT_U6_INT_ST	PCNT_CNT_THR_EVENT_U5_INT_ST	PCNT_CNT_THR_EVENT_U4_INT_ST	PCNT_CNT_THR_EVENT_U3_INT_ST	PCNT_CNT_THR_EVENT_U2_INT_ST	PCNT_CNT_THR_EVENT_U1_INT_ST	PCNT_CNT_THR_EVENT_U0_INT_ST	
31 8	7	6	5	4	3	2	1	0	
0x0000000	0	0	0	0	0	0	0	0	Reset

PCNT_CNT_THR_EVENT_U*n*_INT_ST PCNT_CNT_THR_EVENT_U*n*_INT 中断的屏蔽中断状态位。（只读）

Register 17.7. PCNT_INT_ENA_REG (0x0088)

(reserved)	PCNT_CNT_THR_EVENT_U7_INT_ENA	PCNT_CNT_THR_EVENT_U6_INT_ENA	PCNT_CNT_THR_EVENT_U5_INT_ENA	PCNT_CNT_THR_EVENT_U4_INT_ENA	PCNT_CNT_THR_EVENT_U3_INT_ENA	PCNT_CNT_THR_EVENT_U2_INT_ENA	PCNT_CNT_THR_EVENT_U1_INT_ENA	PCNT_CNT_THR_EVENT_U0_INT_ENA	
31 8	7	6	5	4	3	2	1	0	
0x0000000	0	0	0	0	0	0	0	0	Reset

PCNT_CNT_THR_EVENT_U*n*_INT_ENA PCNT_CNT_THR_EVENT_U*n*_INT 中断的使能位。（读/写）

反馈文档意见

Register 17.8. PCNT_INT_CLR_REG (0x008c)

(reserved)	PCNT_CNT_THR_EVENT_U7_INT_CLR	PCNT_CNT_THR_EVENT_U6_INT_CLR	PCNT_CNT_THR_EVENT_U5_INT_CLR	PCNT_CNT_THR_EVENT_U4_INT_CLR	PCNT_CNT_THR_EVENT_U3_INT_CLR	PCNT_CNT_THR_EVENT_U2_INT_CLR	PCNT_CNT_THR_EVENT_U1_INT_CLR	PCNT_CNT_THR_EVENT_U0_INT_CLR	
31 8	7	6	5	4	3	2	1	0	
0x0000000	0	0	0	0	0	0	0	0	Reset

PCNT_CNT_THR_EVENT_U*n*_INT_CLR 用于清除 PCNT_CNT_THR_EVENT_U*n*_INT 中断。（只写）

Register 17.9. PCNT_CTRL_REG (0x00b0)

(reserved)	PCNT_CLK_EN	PCNT_CNT_PAUSE_U7	PCNT_PLUS_CNT_RST_U7	PCNT_CNT_PAUSE_U6	PCNT_PLUS_CNT_RST_U6	PCNT_CNT_PAUSE_U5	PCNT_PLUS_CNT_RST_U5	PCNT_CNT_PAUSE_U4	PCNT_PLUS_CNT_RST_U4	PCNT_CNT_PAUSE_U3	PCNT_PLUS_CNT_RST_U3	PCNT_CNT_PAUSE_U2	PCNT_PLUS_CNT_RST_U2	PCNT_CNT_PAUSE_U1	PCNT_PLUS_CNT_RST_U1	PCNT_CNT_PAUSE_U0	PCNT_PLUS_CNT_RST_U0	
31 17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0x0000	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	Reset

PCNT_CLK_EN 配置寄存器时钟门控。
0：仅在应用写寄存器时开启时钟
1：一直强制为寄存器开启时钟
（读/写）

PCNT_CNT_PAUSE_U*n* 用于暂停单元 *n* 的计数器。（读/写）

PCNT_PLUS_CNT_RST_U*n* 用于清零单元 *n* 的计数器。（读/写）

Register 17.10. PCNT_U*n*_STATUS_REG (*n*: 0-7) (0x90+0x0C**n*)

(reserved)	PCNT_THR_ZERO_LAT_Un	PCNT_THR_H_LIM_LAT_Un	PCNT_THR_L_LIM_LAT_Un	PCNT_THR_THRES0_LAT_Un	PCNT_THR_THRES1_LAT_Un	PCNT_THR_ZERO_MODE_Un	
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	00	Reset

PCNT_THR_ZERO_LAT_U*n* 上一次中断发生时，计数器计到 0。(只读)

PCNT_THR_H_LIM_LAT_U*n* 上一次中断发生时，计数器计到最大。(只读)

PCNT_THR_L_LIM_LAT_U*n* 上一次中断发生时，计数器计到最小。(只读)

PCNT_THR_THRES0_LAT_U*n* 上一次中断发生时，计数器计到 PCNT_THR_THRES0_U*n*。(只读)

PCNT_THR_THRES1_LAT_U*n* 上一次中断发生时，计数器计到 PCNT_THR_THRES1_U*n*。(只读)

PCNT_THR_ZERO_MODE_U*n* 该寄存器记录着当前计数器的状态。0：计数值为 +0（计数值为有符号数）；1：计数值为 -0；2：计数值为负数；3：计数值为正数。(只读)

18 定时器组 (TIMG)

18.1 概述

ESP32 内置 4 个 64-bit 通用定时器。每个定时器包含一个 16-bit 预分频器和一个 64-bit 可自动重新加载向上／向下计数器。

ESP32 的定时器分为 2 组，每组 2 个。TIMG*n*_T*x* 的 *n* 代表组别，*x* 代表定时器编号。

定时器特性：

- 16-bit 时钟预分频器，分频系数为 2-65536
- 64-bit 时基计数器
- 可配置的向上／向下时基计数器：增加或减少
- 暂停和恢复时基计数器
- 报警时自动重新加载
- 当报警值溢出/低于保护值时报警
- 软件控制的即时重新加载
- 电平触发中断和边沿触发中断

18.2 功能描述

18.2.1 16-bit 预分频器

每个定时器都以 APB 时钟（缩写 APB_CLK，频率通常为 80 MHz）作为基础时钟。16-bit 预分频器对 APB 时钟进行分频，产生时基计数器时钟（TB_clk）。TB_clk 每过一个周期，时基计数器会向上数一或者向下数一。在使用寄存器 TIMG*n*_T*x*_DIVIDER 配置分频器除数前，必须关闭定时器（清零 TIMG*n*_T*x*_DIVIDER）。定时器使能时配置预分频器会导致不可预知的结果。预分频器可以对 APB 时钟进行 2 到 65536 的分频。具体来说，TIMG*n*_T*x*_DIVIDER 为 1 或 2 时，时钟分频器是 2；TIMG*n*_T*x*_DIVIDER 为 0 时，时钟分频器是 65536。如 TIMG*n*_T*x*_DIVIDER 为其他任意值，时钟会被相同数值分频。

18.2.2 64-bit 时基计数器

TIMG*n*_T*x*_INCREASE 置 1 或清零可以将 64-bit 时基计数器分别配置为向上计数或向下计数。同时，64-bit 时基计数器支持自动重新加载和软件即时重新加载，计数器达到软件设定值时会触发报警事件。

TIMG*n*_T*x*_EN 置 1 或清零可以使能或关闭计数。清零后计数器暂停计数，并会在 TIMG*n*_T*x*_ EN 重新置 1 前保持其值不变。清零 TIMG*n*_T*x* _EN 会重新加载计数器并改变计数器的值，但在设置 TIMG*n*_T*x*_EN 前计数不会恢复。

软件可以通过寄存器 TIMG*n*_T*x*_LOAD_LO 和 TIMG*n*_T*x*_LOAD_HI 重置计数器的值。重新加载时，寄存器

TIMG*n*_T*x*_LOAD_LO 和 TIMG*n*_T*x*_LOAD_HI 的值才会被更新到 64-bit 时基计数器内。报警（报警时自动重新加载）或软件（软件即时重新加载）会触发重新加载。寄存器 TIMG*n*_T*x*_AUTORELOAD 置 1 可以使能报警时自动重新加载。如果报警时自动重新加载未被使能，时基计数器会在报警后继续向上计数或向下计数。在寄存器 TIMG*n*_T*x*_LOAD_REG 上写任意值可以触发软件即时重新加载，写值时计数器值会立刻改变。软件也能通过改变 TIMG*n*_T*x*_INCREASE 的值立刻改变时基计数器计数方向。

软件也能读取时基计数器。但由于计数器是 64-bit，CPU 只能以两个 32-bit 值的形式读值。计数器值首先需要被锁入 TIMG*n*_T*x*LO_REG 和 TIMG*n*_T*x*HI_REG。在 TIMG*n*_T*x*UPDATE_REG 上写任意值可以马上将 64-bit 定时器值锁入两个寄存器。之后，软件可以在任何时间读取寄存器。这样可以防止读取计时器低字和高字时出现读值错误。

18.2.3 报警产生

定时器可以触发报警，报警则会引发重新加载和 / 或触发中断。如报警寄存器 TIMG*n*_T*x*_ALARMLO_REG 和 TIMG*n*_T*x*_ALARMHI_REG 的值等于当前定时器的值，则报警触发。为解决寄存器设置过晚，计数器值超过报警值的问题，当前定时器值高于（适用于向上定时器）或低于（适用于向下定时器）当前报警值也会触发报警。在这种情况下，使能报警功能会马上触发报警。报警使能后，报警使能位自动清零。

18.2.4 MWDT

每个定时器模块另包含一个主系统看门狗定时器（缩写为 MWDT）和有关寄存器。本章列出了寄存器相关信息，功能描述请参阅看门狗定时器章节。

18.2.5 中断

- TIMG*n*_T*x*_INT_WDT_INT 该中断在看门狗定时器中断阶段超时后产生。
- TIMG*n*_T*x*_INT_T1_INT 该中断由定时器 1 上的报警事件产生。
- TIMG*n*_T*x*_INT_T0_INT 该中断由定时器 0 上的报警事件产生。

18.3 寄存器列表

名称	描述	TIMG0	TIMG1	访问
定时器 0 配置和控制寄存器				
TIMG*n*_T0CONFIG_REG	定时器 0 配置寄存器	0x3FF5F000	0x3FF60000	读/写
TIMG*n*_T0LO_REG	定时器 0 的当前值，低 32 位	0x3FF5F004	0x3FF60004	只读
TIMG*n*_T0HI_REG	定时器 0 的当前值，高 32 位	0x3FF5F008	0x3FF60008	只读
TIMG*n*_T0UPDATE_REG	写值将当前定时器的值复制到 TIMG*n*_T0_(LO/HI)_REG	0x3FF5F00C	0x3FF6000C	只写
TIMG*n*_T0ALARMLO_REG	定时器 0 的报警值，低 32 位	0x3FF5F010	0x3FF60010	读/写
TIMG*n*_T0ALARMHI_REG	定时器 0 的报警值，高 32 位	0x3FF5F014	0x3FF60014	读/写
TIMG*n*_T0LOADLO_REG	定时器 0 的重新加载值，低 32 位	0x3FF5F018	0x3FF60018	读/写
TIMG*n*_T0LOADHI_REG	定时器 0 的重新加载值，高 32 位	0x3FF5F01C	0x3FF6001C	读/写
TIMG*n*_T0LOAD_REG	写值从 TIMG*n*_T0_(LOADLO/LOADHI)_REG 上加载定时器	0x3FF5F020	0x3FF60020	只写
定时器 1 配置和控制寄存器				
TIMG*n*_T1CONFIG_REG	定时器 1 的配置寄存器	0x3FF5F024	0x3FF60024	读/写
TIMG*n*_T1LO_REG	定时器 1 的当前值，低 32 位	0x3FF5F028	0x3FF60028	只读
TIMG*n*_T1HI_REG	定时器 1 的当前值，高 32 位	0x3FF5F02C	0x3FF6002C	只读
TIMG*n*_T1UPDATE_REG	写值将当前定时器的值复制到 TIMG*n*_T1_(LO/HI)_REG	0x3FF5F030	0x3FF60030	只写
TIMG*n*_T1ALARMLO_REG	定时器 1 的报警值，低 32 位	0x3FF5F034	0x3FF60034	读/写
TIMG*n*_T1ALARMHI_REG	定时器 1 的报警值，高 32 位	0x3FF5F038	0x3FF60038	读/写
TIMG*n*_T1LOADLO_REG	定时器 1 的重新加载值，低 32 位	0x3FF5F03C	0x3FF6003C	读/写

名称	描述	TIMG0	TIMG1	访问
TIMG*n*_T1LOADHI_REG	定时器 1 的重新加载值，高 32 位	0x3FF5F040	0x3FF60040	读/写
TIMG*n*_T1LOAD_REG	写值从 TIMG*n*_T1_(LOADLO/LOADHI)_REG 上加载定时器	0x3FF5F044	0x3FF60044	只写
系统看门狗定时器配置和控制寄存器				
TIMG*n*_T*x*_WDTCONFIG0_REG	看门狗定时器配置寄存器	0x3FF5F048	0x3FF60048	读/写
TIMG*n*_T*x*_WDTCONFIG1_REG	看门狗定时器预分频器寄存器	0x3FF5F04C	0x3FF6004C	读/写
TIMG*n*_T*x*_WDTCONFIG2_REG	看门狗定时器阶段 0 超时时间	0x3FF5F050	0x3FF60050	读/写
TIMG*n*_T*x*_WDTCONFIG3_REG	看门狗定时器阶段 1 超时时间	0x3FF5F054	0x3FF60054	读/写
TIMG*n*_T*x*_WDTCONFIG4_REG	看门狗定时器阶段 2 超时时间	0x3FF5F058	0x3FF60058	读/写
TIMG*n*_T*x*_WDTCONFIG5_REG	看门狗定时器阶段 3 超时时间	0x3FF5F05C	0x3FF6005C	读/写
TIMG*n*_T*x*_WDTFEED_REG	写值来喂看门狗定时器	0x3FF5F060	0x3FF60060	只写
TIMG*n*_T*x*_WDTWPROTECT_REG	看门狗写保护寄存器	0x3FF5F064	0x3FF60064	读/写
RTC 校准配置寄存器				
TIMG*n*_RTCCALICFG_REG	RTC 校准配置寄存器	0x3FF5F068	0x3FF60068	不定
TIMG*n*_RTCCALICFG1_REG	RTC 校准配置寄存器 1	0x3FF5F06C	0x3FF6006C	只读
中断寄存器				
TIMG*n*_T*x*_INT_ENA_REG	中断使能位	0x3FF5F098	0x3FF60098	读/写
TIMG*n*_T*x*_INT_RAW_REG	原始中断状态	0x3FF5F09C	0x3FF6009C	只读
TIMG*n*_T*x*_INT_ST_REG	屏蔽中断状态	0x3FF5F0A0	0x3FF600A0	只读
TIMG*n*_T*x*_INT_CLR_REG	中断清除位	0x3FF5F0A4	0x3FF600A4	只写

18.4 寄存器

本小节括号中的地址均为相对于 TIMG 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 18.3 寄存器列表。

Register 18.1. TIMG*n*_T*x*CONFIG_REG (*x*: 0-1) (0x0+0x24**x*)

TIMG*n*_T*x*_EN	TIMG*n*_T*x*_INCREASE	TIMG*n*_T*x*_AUTORELOAD	TIMG*n*_T*x*_DIVIDER	TIMG*n*_T*x*_EDGE_INT_EN	TIMG*n*_T*x*_LEVEL_INT_EN	TIMG*n*_T*x*_ALARM_EN	
31	30	29	28 13	12	11	10	
0	1	1	0x00001	0	0	0	Reset

TIMG*n*_T*x*_EN 置 1 后，定时器 *x* 时基计数器使能。（读/写）

TIMG*n*_T*x*_INCREASE 置 1 后，定时器 *x* 的时基计数器会在每个时钟周期后增加。清零后，定时器 *x* 时基计数器会在每个时钟周期后减少。（读/写）

TIMG*n*_T*x*_AUTORELOAD 置 1 后，定时器 *x* 报警时自动重新加载使能。（读/写）

TIMG*n*_T*x*_DIVIDER 计时器 *x* 时钟 (T*x*_clk) 的预分频器值。（读/写）

TIMG*n*_T*x*_EDGE_INT_EN 置 1 后，报警会产生一个边沿触发中断。（读/写）

TIMG*n*_T*x*_LEVEL_INT_EN 置 1 后，报警会产生一个电平触发中断。（读/写）

TIMG*n*_T*x*_ALARM_EN 置 1 后，报警使能。报警使能后，此位自动清零。（读/写）

Register 18.2. TIMG*n*_T*x*LO_REG (*x*: 0-1) (0x4+0x24**x*)

31 0	
0x000000000	Reset

TIMG*n*_T*x*LO_REG 在 TIMG*n*_T*x*UPDATE_REG 上写值后，定时器 *x* 时基计数器的低 32 位可以被读取。（只读）

Register 18.3. TIMG*n*_T*x*HI_REG (*x*: 0-1) (0x8+0x24**x*)

31 0	
0x000000000	Reset

TIMG*n*_T*x*HI_REG 在 TIMG*n*_T*x*UPDATE_REG 上写值后，定时器 *x* 时基计数器的高 32 位可以被读取。（只读）

Register 18.4. TIMG*n*_T*x*UPDATE_REG (*x*: 0-1) (0xC+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*UPDATE_REG 写任何值触发定时器*x*时基计数器值更新（定时器*x*当前值会被存储到以上寄存器）。（只写）

Register 18.5. TIMG*n*_T*x*ALARMLO_REG (*x*: 0-1） (0x10+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*ALARMLO_REG 定时器*x*时基计数器触发警报值的低32位。（读/写）

Register 18.6. TIMG*n*_T*x*ALARMHI_REG (*x*: 0-1) (0x14+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*ALARMHI_REG 定时器*x*时基计数器触发警报值的高32位。(读/写）

Register 18.7. TIMG*n*_T*x*LOADLO_REG (*x*: 0-1) (0x18+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*LOADLO_REG 定时器*x*时基计数器重新加载的低32-bit值。(读/写)

Register 18.8. TIMG*n*_T*x*LOADHI_REG (*x*: 0-1) (0x1C+0x24**x*)

31	0	
0x000000000		Reset

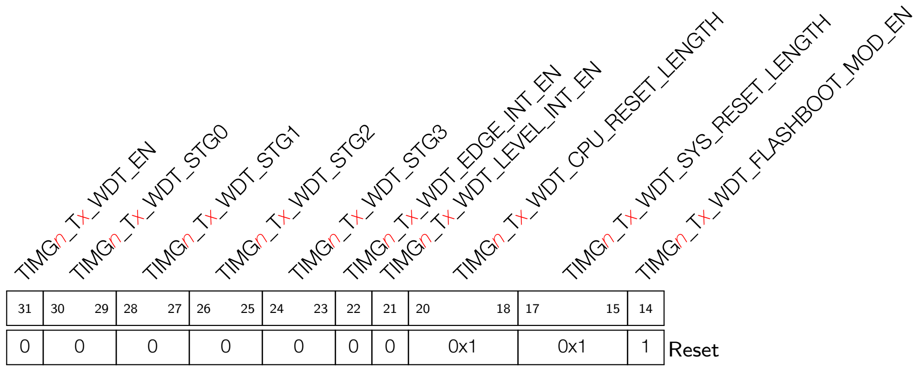
TIMG*n*_T*x*LOADHI_REG 定时器*x*时基计数器重新加载的高32-bit值。（读/写）

Register 18.9. TIMG*n*_T*x*LOAD_REG (*x*: 0-1) (0x20+0x24**x*)

31	0	
0x000000000		Reset

TIMG*n*_T*x*LOAD_REG 写任何值触发定时器*x*时基计数器重新加载。（只写）

Register 18.10. TIMGn_Tx_WDTCONFIG0_REG (0x0048)



TIMGn_Tx_WDT_EN 置 1 后, SWDT 使能。(读/写)

TIMGn_TxS_WDT_STG0 阶段 0 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

TIMGn_Tx_WDT_STG1 阶段 1 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

TIMGn_Tx_WDT_STG2 阶段 2 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

TIMGn_Tx_WDT_STG3 阶段 3 配置。0: 关闭, 1: 中断, 2: 复位 CPU, 3: 复位系统。(读/写)

TIMGn_Tx_WDT_EDGE_INT_EN 置 1 后, 如超过定时器 x 的阶段中断产生时间, 会产生边沿触发中断。(读/写)

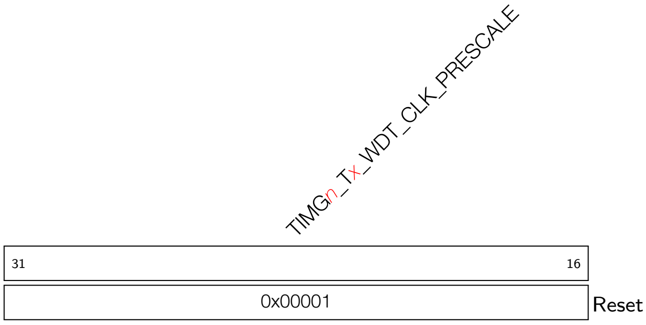
TIMGn_Tx_WDT_LEVEL_INT_EN 置 1 后, 如超过设置的阶段中断产生时间, 会产生电平触发中断。(读/写)

TIMGn_Tx_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 μs, 7: 3.2 μs。(读/写)

TIMGn_Tx_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 μs, 7: 3.2 μs。(读/写)

TIMGn_Tx_WDT_FLASHBOOT_MOD_EN 置 1 后, Flash 启动保护使能。(读/写)

Register 18.11. TIMGn_Tx_WDTCONFIG1_REG (0x004c)



TIMGn_Tx_WDT_CLK_PRESCALE SWDT 时钟预分频器值, 时钟周期 = 时钟源周期 * TIMGn_Tx_WDT_CLK_PRESCALE。(读/写)

Register 18.12. TIMG*n*_T*x*_WDTCONFIG2_REG (0x0050)

31	0	
26000000		Reset

TIMG*n*_T*x*_WDTCONFIG2_REG SWDT 时钟周期中阶段 0 超时时间。（读/写）

Register 18.13. TIMG*n*_T*x*_WDTCONFIG3_REG (0x0054)

31	0	
0x007FFFFFF		Reset

TIMG*n*_T*x*_WDTCONFIG3_REG SWDT 时钟周期中阶段 1 超时时间。（读/写）

Register 18.14. TIMG*n*_T*x*_WDTCONFIG4_REG (0x0058)

31	0	
0x0000FFFFF		Reset

TIMG*n*_T*x*_WDTCONFIG4_REG SWDT 时钟周期中阶段 2 超时时间。（读/写）

Register 18.15. TIMG*n*_T*x*_WDTCONFIG5_REG (0x005c)

31	0	
0x0000FFFFF		Reset

TIMG*n*_T*x*_WDTCONFIG5_REG SWDT 时钟周期中阶段 3 超时时间。（读/写）

Register 18.16. TIMG*n*_T*x*_WDTFEED_REG (0x0060)

31	0	
0x000000000		Reset

TIMG*n*_T*x*_WDTFEED_REG 写任何值驱动 SWDT。（只写）

Register 18.17. TIMG*n*_T*x*_WDTWPROTECT_REG (0x0064)

31	0	
0x050D83AA1		Reset

TIMG*n*_T*x*_WDTWPROTECT_REG 如果寄存器中有和复位值不同的值，写保护使能。（读/写）

Register 18.18. TIMG*n*_RTCCALICFG_REG (0x0068)

TIMG*n*_RTC_CALI_START	TIMG*n*_RTC_CALI_MAX	TIMG*n*_RTC_CALI_RDY	TIMG*n*_RTC_CALI_CLK_SEL	TIMG*n*_RTC_CALI_START_CYCLING	(reserved)	
31	30 16	15	14 13	12	11 0	
0	0x01	0	0x1	1	0 0 0 0 0 0 0 0 0 0 0 0	Reset

TIMG*n*_RTC_CALI_START_CYCLING 保留。(读/写)

TIMG*n*_RTC_CALI_CLK_SEL 选择待校准时钟。0：RC_SLOW_CLK。1：RC_FAST_DIV_CLK。2：XTAL32K_CLK。(读/写)

TIMG*n*_RTC_CALI_RDY 标记本次校准完成。(只读)

TIMG*n*_RTC_CALI_MAX 校准时间，计数单位为待校准时钟周期。(读/写)

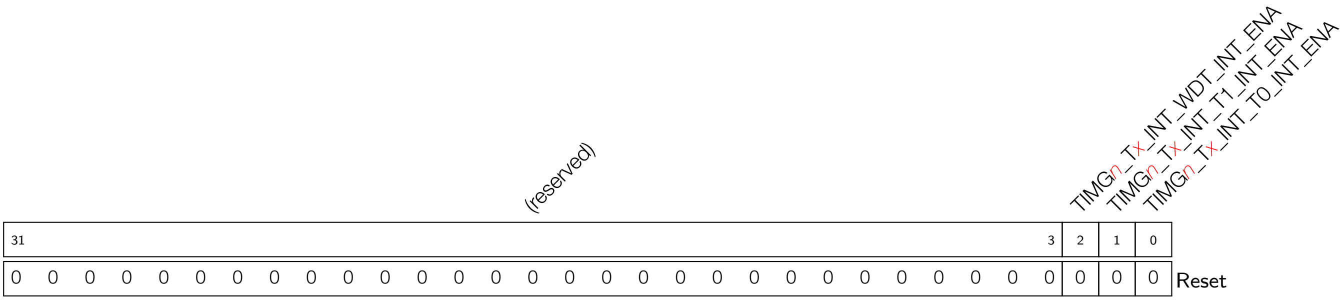
TIMG*n*_RTC_CALI_START 软件开启校准。(读/写)

Register 18.19. TIMG*n*_RTCCALICFG1_REG (0x006C)

TIMG*n*_RTC_CALI_VALUE	(reserved)	
31 7	5 0	
0x00000	0 0 0 0 0 0	Reset

TIMG*n*_RTC_CALI_VALUE 待校准时钟周期数达到 TIMG*n*_RTC_CALI_MAX 时的校准结果，单位是 XTAL_CLK 周期。(只读)

Register 18.20. TIMG*n*_T*x*_INT_ENA_REG (0x0098)

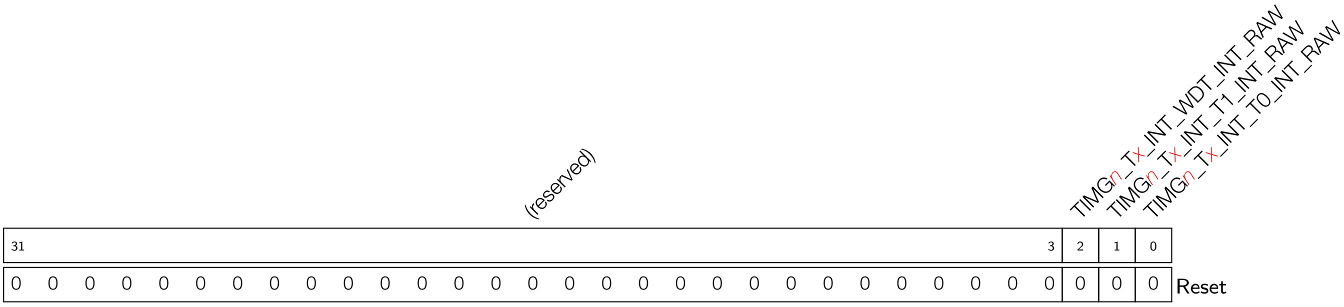


TIMG*n*_T*x*_INT_WDT_INT_ENA TIMG*n*_T*x*_INT_WDT_INT 中断的中断使能位。（读/写）

TIMG*n*_T*x*_INT_T1_INT_ENA TIMG*n*_T*x*_INT_T1_INT 中断的中断使能位。（读/写）

TIMG*n*_T*x*_INT_T0_INT_ENA TIMG*n*_T*x*_INT_T0_INT 中断的中断使能位。（读/写）

Register 18.21. TIMG*n*_T*x*_INT_RAW_REG (0x009c)

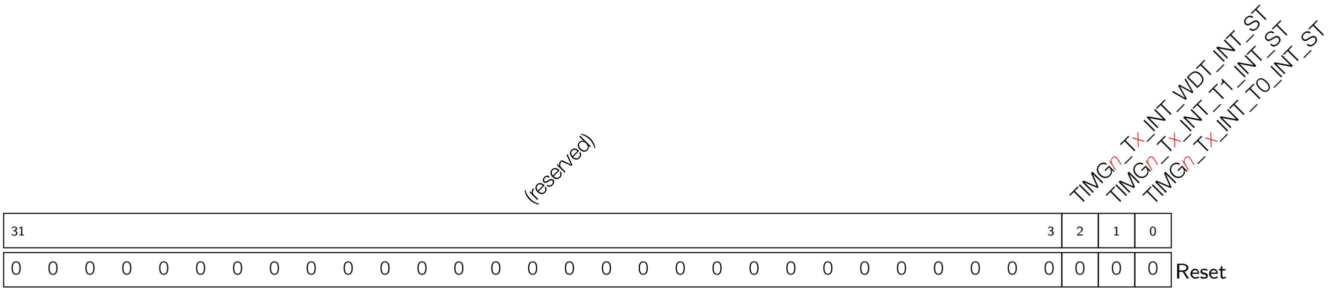


TIMG*n*_T*x*_INT_WDT_INT_RAW TIMG*n*_T*x*_INT_WDT_INT 中断的原始中断状态位。（只读）

TIMG*n*_T*x*_INT_T1_INT_RAW TIMG*n*_T*x*_INT_T1_INT 中断的原始中断状态位。（只读）

TIMG*n*_T*x*_INT_T0_INT_RAW TIMG*n*_T*x*_INT_T0_INT 中断的原始中断状态位。（只读）

Register 18.22. TIMG*n*_T*x*_INT_ST_REG (0x00a0)

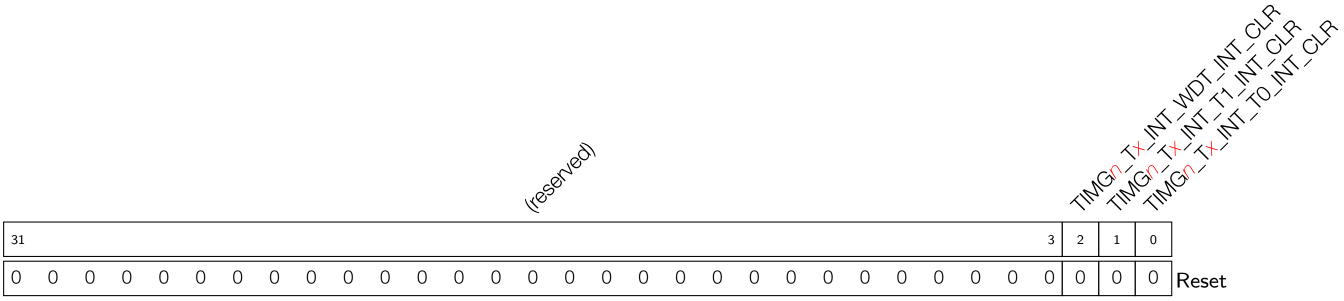


TIMG*n*_T*x*_INT_WDT_INT_ST TIMG*n*_T*x*_INT_WDT_INT 中断的屏蔽中断状态位。（只读）

TIMG*n*_T*x*_INT_T1_INT_ST TIMG*n*_T*x*_INT_T1_INT 中断的屏蔽中断状态位。（只读）

TIMG*n*_T*x*_INT_T0_INT_ST TIMG*n*_T*x*_INT_T0_INT 中断的屏蔽中断状态位。（只读）

Register 18.23. TIMGn_Tx_INT_CLR_REG (0x00a4)



TIMGn_Tx_INT_WDT_INT_CLR 置 1 该位以清除 TIMGn_Tx_INT_WDT_INT 中断。（只写）

TIMGn_Tx_INT_T1_INT_CLR 置 1 该位以清除 TIMGn_Tx_INT_T1_INT 中断。（只写）

TIMGn_Tx_INT_T0_INT_CLR 置 1 该位以清除 TIMGn_Tx_INT_T0_INT 中断。（只写）

19 看门狗定时器 (WDT)

19.1 概述

ESP32 中有三个看门狗定时器：两个定时器模块中各一个（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）。不可预知的软件或硬件问题会导致应用程序工作失常，看门狗定时器可以帮助系统从中恢复。看门狗定时器有四个阶段。如果当前阶段超过预定时间，但没有喂狗或关闭看门狗定时器，每个阶段可能被配成以下三或四种动作中的一种。这些动作是：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，将复位包括 RTC 和主系统在内的整个芯片。每个阶段的超时时间都可单独设置。

在 Flash 启动期间，RWDT 和第一个 MWDT 会自动开启以检测和修复启动问题。

19.2 主要特性

- 4 个阶段，每阶段都可被单独配置或关闭
- 各阶段超时时间可配置
- 如阶段超时，会采取三到四种可能动作中的一种（中断、CPU 复位、内核复位和系统复位）
- 32-bit 超时计数器
- 写保护，防止 RWDT 和 MWDT 配置被不小心改变
- Flash 启动保护
  如果在预定时间内，SPI Flash 的启动过程没有完成，看门狗会重启整个主系统

19.3 功能描述

19.3.1 时钟

RWDT 的时钟源是 RTC 慢速时钟 RTC_SLOW_CLK。MWDT 时钟源来自于 APB 时钟 APB_CLK，经过可配置的 16-bit 预分频器输出给 MWDT。RWDT 和 MWDT 的时钟源用作驱动 32-bit 超时计数器。当计数器接近当前阶段的超时时间，将执行当前阶段配置的动作，超时计数器复位，下一阶段启动。

19.3.2 运行过程

RWDT 和 MWDT 使能时会循环工作，从阶段 0 进行到阶段 3，再回到阶段 0 重新开始。每阶段超时动作和超时时间可以被单独配置。

如果超时计时器接近阶段超时时间，以下动作可以被配置到每个阶段：

- 触发中断
  如阶段超时，中断被触发。
- 复位 CPU 内核
  如阶段超时，复位指定 CPU 内核。复位 MWDT0 CPU 只能复位 PRO CPU，复位 MWDT1 CPU 只能复位 APP CPU。根据不同配置，复位 RWDT CPU 可以复位两个，一个或不复位 CPU 内核。
- 复位主系统
  如阶段超时，包括 MWDT 在内的主系统都会被复位。在本文中，主系统指的是 CPU 和所有外设。但 RTC 是一个例外，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。

当软件喂狗时，看门狗定时器重新回到阶段 0，超时计数器从 0 重新开始。

19.3.3 写保护

两个 MWDT 和 RWDT 都可被保护不受误写影响。为实现该功能，两个看门狗都配有写密匙保护寄存器。MWDT 的寄存器为 TIMG*n*_T*x*_WDTWPROTECT_REG, RWDT 的寄存器为 RTC_CNTL_WDTWPROTECT_REG。复位时，这些寄存器的初始值为 0x50D83AA1。当寄存器的值被改变，写保护使能。此时，除了写密匙保护寄存器以外，在包括喂狗寄存器的其他任意 WDT 寄存器上写值操作会被忽略。推荐按以下步骤访问 WDT：

1. 关闭写保护
2. 根据需要修改寄存器或喂狗
3. 重新使能写保护

19.3.4 Flash 启动保护

在 Flash 启动过程中，定时器组 0（TIMG0）中的 MWDT 和 RWDT 自动使能。两个看门狗定时器的阶段 0 默认为在超时后复位系统。启动后，应该清零寄存器 TIMERS_WDT_FLASHBOOT_MOD_EN 来关闭 MWDT Flash 启动保护程序。对于 RWDT，则应该清零 RTC_CNTL_WDT_FLASHBOOT_MOD_EN 。然后，软件可以配置 MWDT 和 RWDT。

19.3.5 寄存器

MWDT 寄存器是定时器子模块的一部分，在定时器寄存器中有详细描述。RWDT 寄存器是 RTC 子模块的一部分，在 RTC 寄存器中有详细描述。

20 eFuse 控制器 (eFuse)

20.1 概述

ESP32 中有多个 eFuse，其中存储着系统参数。作为一种非易失性存储单位，eFuse 的 bit 一旦被烧写为 1，不能恢复为 0。eFuse 控制器按照软件操作完成对 eFuse 中各个系统参数中的各个 bit 的烧写。

这些系统参数有些可以通过 eFuse 控制器被软件读取，有些直接由硬件模块使用。

20.2 主要特性

- 33 个系统参数
- 烧写保护（可选）
- 软件读取保护（可选）

20.3 功能描述

20.3.1 结构

eFuse 中存储了 33 个系统参数，各个系统参数的位宽不同，其名称与位宽如表 20-1 所示。其中，efuse_wr_disable、efuse_rd_disable、coding_scheme、BLK3_part_reserve 这 4 个系统参数与 eFuse 控制器直接相关。

表 20-1. 系统参数

参数	位宽	efuse_wr_disable 烧写保护	efuse_rd_disable 软件读取保护	描述
efuse_wr_disable	16	1	-	控制 eFuse 控制器
efuse_rd_disable	4	0	-	控制 eFuse 控制器
flash_crypt_cnt	7	2	-	管理 flash 加密/解密
WIFI_MAC_Address	56	3	-	Wi-Fi MAC 地址和 CRC
SPI_pad_config_hd	5	3	-	将 SPI I/O 配置到某个管脚
XPD_SDIO_REG	1	5	-	给 flash 调节器上电
SDIO_TIEH	1	5	-	配置 flash 调节器 电压：置 1 时为 3.3 V； 置 0 时为 1.8 V
sdio_force	1	5	-	决定 XPD_SDIO_REG 和 SDIO_TIEH 是否控制 flash 调节器
BLK3_part_reserve	1	5	-	使用 BLOCK3
SPI_pad_config_clk	5	6	-	将 SPI I/O 配置到某个管脚
SPI_pad_config_q	5	6	-	将 SPI I/O 配置到某个管脚
SPI_pad_config_d	5	6	-	将 SPI I/O 配置到某个管脚
SPI_pad_config_cs0	5	6	-	将 SPI I/O 配置到某个管脚
flash_crypt_config	4	10	3	管理 flash 解密/加密
coding_scheme*	2	10	3	控制 eFuse 控制器
console_debug_disable	1	15	-	置 1 时，禁用 ROM BASIC 控制台调试 fallback 模式。

参数	位宽	efuse_wr_disable 烧写保护	efuse_rd_disable 软件读取保护	描述
abstract_done_0	1	12	-	决定 Secure Boot 的状态
abstract_done_1	1	13	-	决定 Secure Boot 的状态
JTAG_disable	1	14	-	禁用 JTAG
download_dis_encrypt	1	15	-	管理 flash 解密/加密
download_dis_decrypt	1	15	-	管理 flash 解密/加密
download_dis_cache	1	15	-	在 Download 模式中关闭 Cache
key_status	1	10	3	决定 BLOCK3 是否被用户使用
BLOCK1*	256/192/128	7	0	管理 flash 解密/加密
BLOCK2*	256/192/128	8	1	Secure Boot 密钥
BLOCK3*	256/192/128	9	2	用户使用密钥
disable_app_cpu	1	3	-	禁用 APP CPU
disable_bt	1	3	-	禁用蓝牙
pkg_version	4	3	-	封装版本
disable_cache	1	3	-	禁用 cache
CK8M Frequency	8	4	-	RC_FAST_CLK 频率
vol_level_hp_inv	2	3	-	用于标识 CPU 跑 240 MHz 或 flash/PSRAM 跑 80 MHz 需要的电压档位
dig_vol_l6	4	11	-	用于存放档位 6 的 dig regulator 的电压相对于 1.2 V 的差值
uart_download_dis	1	2	-	置 1 时，永久禁用下载启动模式（仅 ESP32 ECO V3 支持）

20.3.1.1 系统参数 efuse_wr_disable

系统参数 efuse_wr_disable 决定所有的系统参数是否处于烧写保护状态。作为一个系统参数，efuse_wr_disable 也决定其本身是否处于烧写保护状态。

若某个系统参数未处于烧写保护状态，则此系统参数未被烧写的 bit 能够从 0 被烧写成 1。其已被烧写的 bit 已经为 1，且不可更改。若某个系统参数处于烧写保护状态，则此系统参数的每一个 bit 都无法再被更改。其未被烧写的 bit 永远为 0，已被烧写的 bit 永远为 1。

每个系统参数的烧写保护状态对应 efuse_wr_disable 的一个 bit。当某个系统参数对应的 bit 为 0 时，表示此系统参数未处于烧写保护状态。当某个系统参数对应的 bit 为 1 时，表示此系统参数处于烧写保护状态。如果某个系统参数已经处于烧写保护状态，则将永远处于此状态。表 20-1 描述了各个系统参数的烧写保护状态具体由 efuse_wr_disable 的哪个 bit 决定。

20.3.1.2 系统参数 efuse_rd_disable

33 个系统参数中有 27 个不受软件读取保护状态的约束，这 27 个系统参数即表 20-1 中“efuse_rd_disable 软件读取保护”一列中，对应值为“-”的系统参数。这些系统参数在任何时候都可通过 eFuse 控制器由软件读取，

其中有部分同时也由硬件模块使用。

其余的 6 个系统参数在未处于软件读取保护状态时，既可以被软件读取也可以被硬件模块使用。当它们处于软件读取保护状态时，只能被硬件模块使用。

表 20-1 中的“efuse_rd_disable 软件读取保护”一列描述了这 6 个系统参数的软件读取保护状态具体由 efuse_rd_disable 的哪个 bit 决定。系统参数 efuse_rd_disable 中的某个 bit 为 0，表示此 bit 管理的系统参数未处于软件读取保护状态。若系统参数 efuse_rd_disable 中的某个 bit 为 1，表示此 bit 管理的系统参数处于软件读取保护状态。如果某个系统参数已处于软件读取保护状态，则将永远处于此状态。

20.3.1.3 系统参数 coding_scheme

如表 20-1 所示，系统参数 BLOCK1、BLOCK2、BLOCK3 的位宽是变化的。它们的位宽由另一个系统参数 coding_scheme 决定。虽然 BLOCK1、BLOCK2、BLOCK3 的位宽是变化的，但这 3 个系统参数在 eFuse 中占用的 bit 数始终不变。这 3 个系统参数与它们在 eFuse 中的存储值之间存在着编码映射关系。具体参见表 20-2。

表 20-2. BLOCK1/2/3 编码

coding_scheme[1:0]	BLOCK1/2/3 位宽	编码方式	eFuse 中的 bit 数
00/11	256	非编码	256
01	192	3/4 编码	256
10	128	重复编码	256

以下对三种编码方式进行解释，其中，

- $BLOCKN$ 表示系统参数 BLOCK1、BLOCK2 或 BLOCK3。
- $BLOCKN[255:0]$、$BLOCKN[191:0]$、$BLOCKN[127:0]$ 分别表示三种编码方式下这几个系统参数的各个 bit。
- ${}^{e}BLOCKN[255:0]$ 表示这几个系统参数经过编码之后的各个 bit，即在 eFuse 中的各个 bit。

非编码方式

$${}^{e}BLOCKN[255:0] = BLOCKN[255:0]$$

3/4 编码方式

$$\begin{aligned} BLOCKN_i^j[7:0] &= BLOCKN[48i+8j+7:48i+8j] & i \in \{0,1,2,3\} & \quad j \in \{0,1,2,3,4,5\} \\ {}^{e}BLOCKN_i^j[7:0] &= {}^{e}BLOCKN[64i+8j+7:64i+8j] & i \in \{0,1,2,3\} & \quad j \in \{0,1,2,3,4,5,6,7\} \end{aligned}$$

$$^{e}BLOCKN_i^j[7:0] = \begin{cases} BLOCKN_i^j[7:0] & j \in \{0,1,2,3,4,5\} \\ BLOCKN_i^0[7:0] \oplus BLOCKN_i^1[7:0] & \\ \quad \oplus BLOCKN_i^2[7:0] \oplus BLOCKN_i^3[7:0] & j \in \{6\} \\ \quad \oplus BLOCKN_i^4[7:0] \oplus BLOCKN_i^5[7:0] & \\ \sum_{l=0}^{5}(l+1)\sum_{k=0}^{7} BLOCKN_i^l[k] & j \in \{7\} \end{cases} \qquad i \in \{0,1,2,3\}$$

$\oplus$ 表示按位异或

$\sum$ 和 $+$ 表示加

重复编码方式

$$^{e}BLOCKN[255:128] = {}^{e}BLOCKN[127:0] = BLOCKN[127:0]$$

20.3.1.4 系统参数 BLK3_part_reserve

系统参数 coding_scheme、BLOCK1、BLOCK2、BLOCK3 受系统参数 BLK3_part_reserve 约束。

当 BLK3_part_reserve 为 0 时，coding_scheme、BLOCK1、BLOCK2、BLOCK3 可以是允许的任意值。

当 BLK3_part_reserve 为 1 时，coding_scheme、BLOCK1、BLOCK2、BLOCK3 必然被固定为 3/4 编码。另外此时 $BLOCK3[143:96]$ 即 $^{e}BLOCK3[191:128]$ 不允许使用。

20.3.2 烧写系统参数

烧写变长系统参数 BLOCK1、BLOCK2、BLOCK3 不同于烧写定长系统参数。**我们并不直接烧写系统参数 BLOCK1、BLOCK2、BLOCK3 本身，而是烧写其编码之后的值 $^{e}BLOCKN[255:0]$，这个值的位宽始终是 256**。而烧写定长系统参数则是直接烧写系统参数本身。

30 个定长系统参数的每一个 bit 与 3 个变长系统参数编码之后的每一个 bit 都分别对应一个烧写寄存器 bit，对应关系如表 20-3 所示。烧写系统参数的时候需要使用到这些寄存器 bit。

表 20-3. 烧写寄存器

系统参数			寄存器	
名称	位宽	Bit	名称	Bit
efuse_wr_disable	16	[15:0]	EFUSE_BLK0_WDATA0_REG	[15:0]
efuse_rd_disable	4	[3:0]		[19:16]
flash_crypt_cnt	7	[6:0]		[26:20]
uart_download_dis	1	[0]		[27]
WIFI_MAC_Address	56	[31:0]	EFUSE_BLK0_WDATA1_REG	[31:0]
		[55:32]	EFUSE_BLK0_WDATA2_REG	[23:0]
disable_app_cpu	1	[0]	EFUSE_BLK0_WDATA3_REG	[0]
disable_bt	1	[0]		[1]
pkg_version	4	[3:0]		[2], [11:9]
disable_cache	1	[0]		[3]
SPI_pad_config_hd	5	[4:0]		[8:4]
BLK3_part_reserve	1	[0]		[14]
CK8M Frequency	8	[7:0]	EFUSE_BLK0_WDATA4_REG	[7:0]
XPD_SDIO_REG	1	[0]		[14]

系统参数			寄存器	
名称	位宽	Bit	名称	Bit
SDIO_TIEH	1	[0]		[15]
sdio_force	1	[0]		[16]
SPI_pad_config_clk	5	[4:0]	EFUSE_BLK0_WDATA5_REG	[4:0]
SPI_pad_config_q	5	[4:0]		[9:5]
SPI_pad_config_d	5	[4:0]		[14:10]
SPI_pad_config_cs0	5	[4:0]		[19:15]
vol_level_hp_inv	2	[1:0]		[23:22]
dig_vol_l6	4	[3:0]		[27:24]
flash_crypt_config	4	[3:0]		[31:28]
coding_scheme	2	[1:0]	EFUSE_BLK0_WDATA6_REG	[1:0]
console_debug_disable	1	[0]		[2]
abstract_done_0	1	[0]		[4]
abstract_done_1	1	[0]		[5]
JTAG_disable	1	[0]		[6]
download_dis_encrypt	1	[0]		[7]
download_dis_decrypt	1	[0]		[8]
download_dis_cache	1	[0]		[9]
key_status	1	[0]		[10]
BLOCK1	256/192/128	[31:0]	EFUSE_BLK1_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK1_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK1_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK1_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK1_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK1_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK1_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK1_WDATA7_REG	[31:0]
BLOCK2	256/192/128	[31:0]	EFUSE_BLK2_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK2_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK2_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK2_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK2_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK2_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK2_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK2_WDATA7_REG	[31:0]
BLOCK3	256/192/128	[31:0]	EFUSE_BLK3_WDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK3_WDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK3_WDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK3_WDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK3_WDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK3_WDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK3_WDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK3_WDATA7_REG	[31:0]

烧写系统参数的流程如下：

1. 配置寄存器 EFUSE_CLK 的 EFUSE_CLK_SEL0 位、EFUSE_CLK_SEL1 位与寄存器 EFUSE_DAC_CONF 的 EFUSE_DAC_CLK_DIV 位。
2. 将需要烧写的 bit 对应的寄存器 bit 置 1。
3. 对寄存器 EFUSE_CONF 写入 0x5A5A。
4. 对寄存器 EFUSE_CMD 写入 0x2。
5. 轮询寄存器 EFUSE_CMD 直到其为 0x0，或者等待烧写完成中断产生。
6. 对寄存器 EFUSE_CONF 写入 0x5AA5。
7. 对寄存器 EFUSE_CMD 写入 0x1。
8. 轮询寄存器 EFUSE_CMD 直到其为 0x0，或者等待读取完成中断产生。
9. 将已烧写了的 bit 对应的寄存器 bit 置 0。

如表 20-4 所示，寄存器 EFUSE_CLK 的 EFUSE_CLK_SEL0 位、EFUSE_CLK_SEL1 位与寄存器 EFUSE_DAC_CONF 的 EFUSE_DAC_CLK_DIV 位的配置值是以当前 APB_CLK 的频率为依据的。

表 20-4. 时序配置

寄存器	对应寄存器位的配置值	APB_CLK 频率		
		26 MHz	40 MHz	80 MHz
EFUSE_CLK	EFUSE_CLK_SEL0[7:0]	250	160	80
	EFUSE_CLK_SEL1[7:0]	255	255	128
EFUSE_DAC_CONF	EFUSE_DAC_CLK_DIV[7:0]	52	80	100

以下两种方法可以识别烧写/读取完成中断的产生：

方法一：

1. 轮询寄存器 EFUSE_INT_RAW 的 bit 1/0，直到 bit 1/0 为 1，表示烧写/读取完成中断产生。
2. 将寄存器 EFUSE_INT_CLR 的 bit 1/0 置 1 以清除烧写/读取完成中断。

方法二：

1. 将寄存器 EFUSE_INT_ENA 的 bit 1/0 置 1，使 eFuse 控制器能够发出烧写/读取完成中断。
2. 配置 Interrupt Matrix 使 CPU 能够响应 EFUSE_INT 中断。
3. 烧写/读取完成中断产生。
4. 查询寄存器 EFUSE_INT_ST 的 bit 1/0 以判断烧写/读取完成中断产生。
5. 对寄存器 EFUSE_INT_CLR 的 bit 1/0 置 1 以清烧写/读取完成中断。

烧写不同的系统参数，甚至是同一个系统参数中的不同 bit 都可以在多次烧写中分别完成。但我们建议尽量减少烧写次数，即某个系统参数中的所有需要烧写的 bit 都在一次烧写中完成。并且当 efuse_wr_disable 的某个 bit 管理的所有系统参数都烧写之后，就立即烧写 efuse_wr_disable 的此 bit。甚至可以在同一次烧写中既烧写

efuse_wr_disable 的某个 bit 管理的所有系统参数，同时也烧写 efuse_wr_disable 的此 bit。**另外，严禁对已经烧写了的 bit 重复烧写**。

20.3.3 软件读取系统参数

30 个定长系统参数的每一个 bit 与 3 个变长系统参数本身的每一个 bit 都分别对应一个软件读取寄存器 bit，对应关系如表 20-5 所示。通过软件读取这些寄存器即可获知系统参数的值。 系统参数 BLOCK1、BLOCK2、BLOCK3 的位宽是不定的。虽然如表 20-5 所示，这 3 个参数各自对应 256 个寄存器 bit，但是在 3/4 编码和重复编码模式下，这 256 个寄存器 bit 中的有部分是无效的。在非编码模式下，$BLOCKN[255:0]$ 的每一个 bit 对应的寄存器 bit 都是有用的。在 3/4 编码模式下，只有 $BLOCKN[191:0]$ 对应的寄存器 bit 是有用的。在重复编码模式下，只有 $BLOCKN[127:0]$ 对应的寄存器 bit 有用。在不同编码方式下，软件从无效的寄存器 bit 中读出的值无意义。**软件从有用的寄存器 bit 读出的值是系统参数 BLOCK1、BLOCK2、BLOCK3 本身，不是这几个系统参数编码之后的值**。

表 20-5. 软件读取寄存器

系统参数			寄存器	
名字	位宽	Bit	名称	Bit
efuse_wr_disable	16	[15:0]	EFUSE_BLK0_RDATA0_REG	[15:0]
efuse_rd_disable	4	[3:0]		[19:16]
flash_crypt_cnt	7	[6:0]		[26:20]
uart_download_dis	1	[0]		[27]
WIFI_MAC_Address	56	[31:0]	EFUSE_BLK0_RDATA1_REG	[31:0]
		[55:32]	EFUSE_BLK0_RDATA2_REG	[23:0]
disable_app_cpu	1	[0]	EFUSE_BLK0_RDATA3_REG	[0]
disable_bt	1	[0]		[1]
pkg_version	4	[3:0]		[2], [11:9]
disable_cache	1	[0]		[3]
SPI_pad_config_hd	5	[4:0]		[8:4]
BLK3_part_reserve	1	[0]		[14]
CK8M Frequency	8	[7:0]	EFUSE_BLK0_RDATA4_REG	[7:0]
XPD_SDIO_REG	1	[0]		[14]
SDIO_TIEH	1	[0]		[15]
sdio_force	1	[0]		[16]
SPI_pad_config_clk	5	[4:0]	EFUSE_BLK0_RDATA5_REG	[4:0]
SPI_pad_config_q	5	[4:0]		[9:5]
SPI_pad_config_d	5	[4:0]		[14:10]
SPI_pad_config_cs0	5	[4:0]		[19:15]
vol_level_hp_inv	2	[1:0]		[23:22]
dig_vol_l6	4	[3:0]		[27:24]
flash_crypt_config	4	[3:0]		[31:28]
coding_scheme	2	[1:0]	EFUSE_BLK0_RDATA6_REG	[1:0]
console_debug_disable	1	[0]		[2]
abstract_done_0	1	[0]		[4]
abstract_done_1	1	[0]		[5]
JTAG_disable	1	[0]		[6]
download_dis_encrypt	1	[0]		[7]
download_dis_decrypt	1	[0]		[8]

系统参数			寄存器	
名字	位宽	Bit	名称	Bit
download_dis_cache	1	[0]		[9]
key_status	1	[0]		[10]
BLOCK1	256/192/128	[31:0]	EFUSE_BLK1_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK1_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK1_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK1_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK1_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK1_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK1_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK1_RDATA7_REG	[31:0]
BLOCK2	256/192/128	[31:0]	EFUSE_BLK2_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK2_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK2_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK2_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK2_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK2_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK2_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK2_RDATA7_REG	[31:0]
BLOCK3	256/192/128	[31:0]	EFUSE_BLK3_RDATA0_REG	[31:0]
		[63:32]	EFUSE_BLK3_RDATA1_REG	[31:0]
		[95:64]	EFUSE_BLK3_RDATA2_REG	[31:0]
		[127:96]	EFUSE_BLK3_RDATA3_REG	[31:0]
		[159:128]	EFUSE_BLK3_RDATA4_REG	[31:0]
		[191:160]	EFUSE_BLK3_RDATA5_REG	[31:0]
		[223:192]	EFUSE_BLK3_RDATA6_REG	[31:0]
		[255:224]	EFUSE_BLK3_RDATA7_REG	[31:0]

20.3.4 硬件模块使用系统参数

硬件模块使用系统参数是通过电路连接实现的，软件无法干预这个过程。硬件模块使用的都是系统参数本身。**对于 BLOCK1、BLOCK2、BLOCK3 而言，硬件模块使用的是解码之后的值，不是它们编码之后的值。**

20.3.5 中断

- EFUSE_PGM_DONE_INT：当 eFuse 烧写完成后，此中断被触发。
- EFUSE_READ_DONE_INT：当 eFuse 读取完成后，此中断被触发。

20.4 寄存器列表

本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。

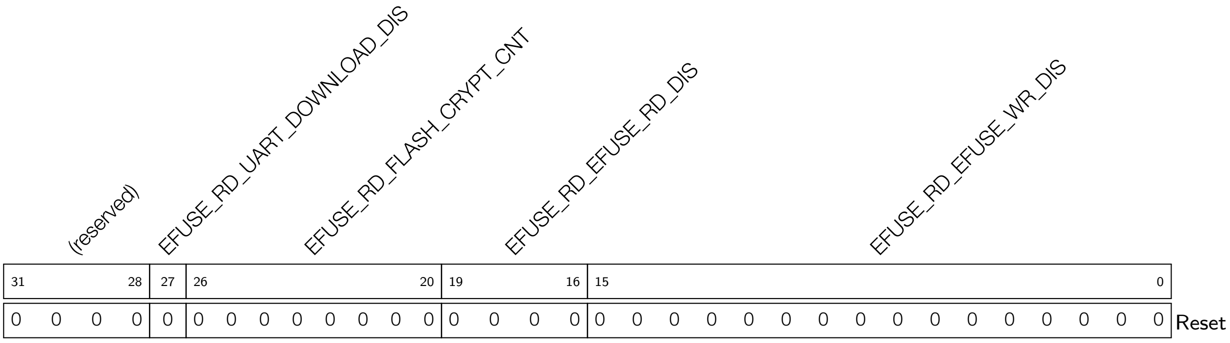
名称	描述	地址	访问
eFuse 读取寄存器			
EFUSE_BLK0_RDATA0_REG	返回 eFuse BLOCK 0 中 word 0 的值	0x3FF5A000	只读
EFUSE_BLK0_RDATA1_REG	返回 eFuse BLOCK 0 中 word 1 的值	0x3FF5A004	只读
EFUSE_BLK0_RDATA2_REG	返回 eFuse BLOCK 0 中 word 2 的值	0x3FF5A008	只读
EFUSE_BLK0_RDATA3_REG	返回 eFuse BLOCK 0 中 word 3 的值	0x3FF5A00C	只读
EFUSE_BLK0_RDATA4_REG	返回 eFuse BLOCK 0 中 word 4 的值	0x3FF5A010	只读
EFUSE_BLK0_RDATA5_REG	返回 eFuse BLOCK 0 中 word 5 的值	0x3FF5A014	只读
EFUSE_BLK0_RDATA6_REG	返回 eFuse BLOCK 0 中 word 6 的值	0x3FF5A018	只读
EFUSE_BLK1_RDATA0_REG	返回 eFuse BLOCK 1 中 word 0 的值	0x3FF5A038	只读
EFUSE_BLK1_RDATA1_REG	返回 eFuse BLOCK 1 中 word 1 的值	0x3FF5A03C	只读
EFUSE_BLK1_RDATA2_REG	返回 eFuse BLOCK 1 中 word 2 的值	0x3FF5A040	只读
EFUSE_BLK1_RDATA3_REG	返回 eFuse BLOCK 1 中 word 3 的值	0x3FF5A044	只读
EFUSE_BLK1_RDATA4_REG	返回 eFuse BLOCK 1 中 word 4 的值	0x3FF5A048	只读
EFUSE_BLK1_RDATA5_REG	返回 eFuse BLOCK 1 中 word 5 的值	0x3FF5A04C	只读
EFUSE_BLK1_RDATA6_REG	返回 eFuse BLOCK 1 中 word 6 的值	0x3FF5A050	只读
EFUSE_BLK1_RDATA7_REG	返回 eFuse BLOCK 1 中 word 7 的值	0x3FF5A054	只读
EFUSE_BLK2_RDATA0_REG	返回 eFuse BLOCK 2 中 word 0 的值	0x3FF5A058	只读
EFUSE_BLK2_RDATA1_REG	返回 eFuse BLOCK 2 中 word 1 的值	0x3FF5A05C	只读
EFUSE_BLK2_RDATA2_REG	返回 eFuse BLOCK 2 中 word 2 的值	0x3FF5A060	只读
EFUSE_BLK2_RDATA3_REG	返回 eFuse BLOCK 2 中 word 3 的值	0x3FF5A064	只读
EFUSE_BLK2_RDATA4_REG	返回 eFuse BLOCK 2 中 word 4 的值	0x3FF5A068	只读
EFUSE_BLK2_RDATA5_REG	返回 eFuse BLOCK 2 中 word 5 的值	0x3FF5A06C	只读
EFUSE_BLK2_RDATA6_REG	返回 eFuse BLOCK 2 中 word 6 的值	0x3FF5A070	只读
EFUSE_BLK2_RDATA7_REG	返回 eFuse BLOCK 2 中 word 7 的值	0x3FF5A074	只读
EFUSE_BLK3_RDATA0_REG	返回 eFuse BLOCK 3 中 word 0 的值	0x3FF5A078	只读
EFUSE_BLK3_RDATA1_REG	返回 eFuse BLOCK 3 中 word 1 的值	0x3FF5A07C	只读
EFUSE_BLK3_RDATA2_REG	返回 eFuse BLOCK 3 中 word 2 的值	0x3FF5A080	只读
EFUSE_BLK3_RDATA3_REG	返回 eFuse BLOCK 3 中 word 3 的值	0x3FF5A084	只读
EFUSE_BLK3_RDATA4_REG	返回 eFuse BLOCK 3 中 word 4 的值	0x3FF5A088	只读
EFUSE_BLK3_RDATA5_REG	返回 eFuse BLOCK 3 中 word 5 的值	0x3FF5A08C	只读
EFUSE_BLK3_RDATA6_REG	返回 eFuse BLOCK 3 中 word 6 的值	0x3FF5A090	只读
EFUSE_BLK3_RDATA7_REG	返回 eFuse BLOCK 3 中 word 7 的值	0x3FF5A094	只读
eFuse 烧写寄存器			
EFUSE_BLK0_WDATA0_REG	烧写 eFuse BLOCK 0 中 word 0 的值	0x3FF5A01c	读/写
EFUSE_BLK0_WDATA1_REG	烧写 eFuse BLOCK 0 中 word 1 的值	0x3FF5A020	读/写
EFUSE_BLK0_WDATA2_REG	烧写 eFuse BLOCK 0 中 word 2 的值	0x3FF5A024	读/写
EFUSE_BLK0_WDATA3_REG	烧写 eFuse BLOCK 0 中 word 3 的值	0x3FF5A028	读/写
EFUSE_BLK0_WDATA4_REG	烧写 eFuse BLOCK 0 中 word 4 的值	0x3FF5A02c	读/写
EFUSE_BLK0_WDATA5_REG	烧写 eFuse BLOCK 0 中 word 5 的值	0x3FF5A030	读/写
EFUSE_BLK0_WDATA6_REG	烧写 eFuse BLOCK 0 中 word 6 的值	0x3FF5A034	读/写

名称	描述	地址	访问
EFUSE_BLK1_WDATA0_REG	烧写 eFuse BLOCK 1 中 word 0 的值	0x3FF5A098	读/写
EFUSE_BLK1_WDATA1_REG	烧写 eFuse BLOCK 1 中 word 1 的值	0x3FF5A09c	读/写
EFUSE_BLK1_WDATA2_REG	烧写 eFuse BLOCK 1 中 word 2 的值	0x3FF5A0a0	读/写
EFUSE_BLK1_WDATA3_REG	烧写 eFuse BLOCK 1 中 word 3 的值	0x3FF5A0a4	读/写
EFUSE_BLK1_WDATA4_REG	烧写 eFuse BLOCK 1 中 word 4 的值	0x3FF5A0a8	读/写
EFUSE_BLK1_WDATA5_REG	烧写 eFuse BLOCK 1 中 word 5 的值	0x3FF5A0ac	读/写
EFUSE_BLK1_WDATA6_REG	烧写 eFuse BLOCK 1 中 word 6 的值	0x3FF5A0b0	读/写
EFUSE_BLK1_WDATA7_REG	烧写 eFuse BLOCK 1 中 word 7 的值	0x3FF5A0b4	读/写
EFUSE_BLK2_WDATA0_REG	烧写 eFuse BLOCK 2 中 word 0 的值	0x3FF5A0b8	读/写
EFUSE_BLK2_WDATA1_REG	烧写 eFuse BLOCK 2 中 word 1 的值	0x3FF5A0bc	读/写
EFUSE_BLK2_WDATA2_REG	烧写 eFuse BLOCK 2 中 word 2 的值	0x3FF5A0c0	读/写
EFUSE_BLK2_WDATA3_REG	烧写 eFuse BLOCK 2 中 word 3 的值	0x3FF5A0c4	读/写
EFUSE_BLK2_WDATA4_REG	烧写 eFuse BLOCK 2 中 word 4 的值	0x3FF5A0c8	读/写
EFUSE_BLK2_WDATA5_REG	烧写 eFuse BLOCK 2 中 word 5 的值	0x3FF5A0cc	读/写
EFUSE_BLK2_WDATA6_REG	烧写 eFuse BLOCK 2 中 word 6 的值	0x3FF5A0d0	读/写
EFUSE_BLK2_WDATA7_REG	烧写 eFuse BLOCK 2 中 word 7 的值	0x3FF5A0d4	读/写
EFUSE_BLK3_WDATA0_REG	烧写 eFuse BLOCK 3 中 word 0 的值	0x3FF5A0d8	读/写
EFUSE_BLK3_WDATA1_REG	烧写 eFuse BLOCK 3 中 word 1 的值	0x3FF5A0dc	读/写
EFUSE_BLK3_WDATA2_REG	烧写 eFuse BLOCK 3 中 word 2 的值	0x3FF5A0e0	读/写
EFUSE_BLK3_WDATA3_REG	烧写 eFuse BLOCK 3 中 word 3 的值	0x3FF5A0e4	读/写
EFUSE_BLK3_WDATA4_REG	烧写 eFuse BLOCK 3 中 word 4 的值	0x3FF5A0e8	读/写
EFUSE_BLK3_WDATA5_REG	烧写 eFuse BLOCK 3 中 word 5 的值	0x3FF5A0ec	读/写
EFUSE_BLK3_WDATA6_REG	烧写 eFuse BLOCK 3 中 word 6 的值	0x3FF5A0f0	读/写
EFUSE_BLK3_WDATA7_REG	烧写 eFuse BLOCK 3 中 word 7 的值	0x3FF5A0f4	读/写
控制寄存器			
EFUSE_CLK_REG	时序配置寄存器	0x3FF5A0F8	读/写
EFUSE_CONF_REG	操作码寄存器	0x3FF5A0FC	读/写
EFUSE_CMD_REG	读/写指令寄存器	0x3FF5A104	读/写
中断寄存器			
EFUSE_INT_RAW_REG	原始中断状态	0x3FF5A108	只读
EFUSE_INT_ST_REG	屏蔽中断状态	0x3FF5A10C	只读
EFUSE_INT_ENA_REG	中断使能位	0x3FF5A110	读/写
EFUSE_INT_CLR_REG	中断清除位	0x3FF5A114	只写
其它寄存器			
EFUSE_DAC_CONF_REG	eFuse 时序配置	0x3FF5A118	读/写
EFUSE_DEC_STATUS_REG	3/4 编码方式的状态	0x3FF5A11C	只读

20.5 寄存器

本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。

Register 20.1. EFUSE_BLK0_RDATA0_REG (0x000)



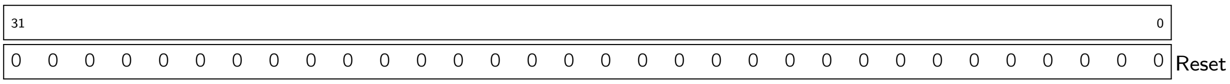
EFUSE_RD_UART_DOWNLOAD_DIS uart_download_dis 的值，仅对 ESP32 ECO V3 有效。（只读）

EFUSE_RD_FLASH_CRYPT_CNT 烧写 flash_crypt_cnt 的值。（只读）

EFUSE_RD_EFUSE_RD_DIS efuse_rd_disable 的值。（只读）

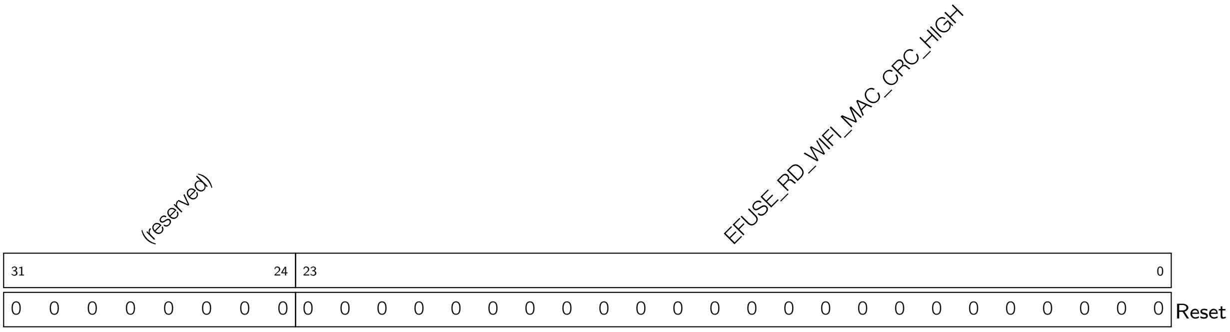
EFUSE_RD_EFUSE_WR_DIS efuse_wr_disable 的值。（只读）

Register 20.2. EFUSE_BLK0_RDATA1_REG (0x004)



EFUSE_BLK0_RDATA1_REG WIFI_MAC_Address 低 32 位的值。（只读）

Register 20.3. EFUSE_BLK0_RDATA2_REG (0x008)



EFUSE_RD_WIFI_MAC_CRC_HIGH WIFI_MAC_Address 高 24 位的值。（只读）

Register 20.4. EFUSE_BLK0_RDATA3_REG (0x00c)

(reserved)	EFUSE_RD_CHIP_VER_PKG	EFUSE_RD_SPI_PAD_CONFIG_HD	EFUSE_RD_CHIP_VER_DIS_CACHE	EFUSE_RD_CHIP_VER_PKG	EFUSE_RD_CHIP_VER_DIS_BT	EFUSE_RD_CHIP_VER_DIS_APP_CPU	
31 12	11 9	8 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

EFUSE_RD_CHIP_VER_PKG 共有四个 bit 标记封装版本，这是前三位。（只读）

EFUSE_RD_SPI_PAD_CONFIG_HD SPI_pad_config_hd 的值。（只读）

EFUSE_RD_CHIP_VER_DIS_CACHE 禁用 cache 功能。（只读）

EFUSE_RD_CHIP_VER_PKG 共有四个 bit 标记封装版本，这是最高位。（只读）

EFUSE_RD_CHIP_VER_DIS_BT 禁用蓝牙。（只读）

EFUSE_RD_CHIP_VER_DIS_APP_CPU 禁用 APP CPU。（只读）

Register 20.5. EFUSE_BLK0_RDATA4_REG (0x010)

(reserved)	EFUSE_RD_SDIO_FORCE	EFUSE_RD_SDIO_TIEH	EFUSE_RD_XPD_SDIO	(reserved)	ESFUSE_RD_CK8M_FREQ	
31 17	16	15	14	13 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0 0 0	0 0 0 0 0 0 0 0	Reset

EFUSE_RD_SDIO_FORCE sdio_force 的值。（只读）

EFUSE_RD_SDIO_TIEH SDIO_TIEH 的值。（只读）

EFUSE_RD_XPD_SDIO XPD_SDIO_REG 的值。（只读）

ESFUSE_RD_CK8M_FREQ RC_FAST_CLK 频率。（只读）

Register 20.6. EFUSE_BLK0_RDATA5_REG (0x014)

EFUSE_RD_FLASH_CRYPT_CONFIG	EFUSE_RD_DIG_VOL_L6	EFUSE_RD_VOL_LEVEL_HP_INV	(reserved)	EFUSE_RD_SPI_PAD_CONFIG_CS0	EFUSE_RD_SPI_PAD_CONFIG_D	EFUSE_RD_SPI_PAD_CONFIG_Q	EFUSE_RD_SPI_PAD_CONFIG_CLK	
31 28	27 24	23 22	21 20	19 15	14 10	9 5	4 0	
0 0 0 0	0 0 0 0	0 0	0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	Reset

EFUSE_RD_FLASH_CRYPT_CONFIG flash_crypt_config 的值。（只读）

EFUSE_RD_DIG_VOL_L6 用于存放档位 6 的 dig regulator 的电压相对于 1.2 V 的差值。（只读）

EFUSE_RD_VOL_LEVEL_HP_INV 用于标识 CPU 跑 240 MHz 或 flash/PSRAM 跑 80 MHz 需要的电压档位。0x0：采用档位 7；0x1：采用档位 6；0x2：采用档位 5；0x3：采用档位 4。（只读）

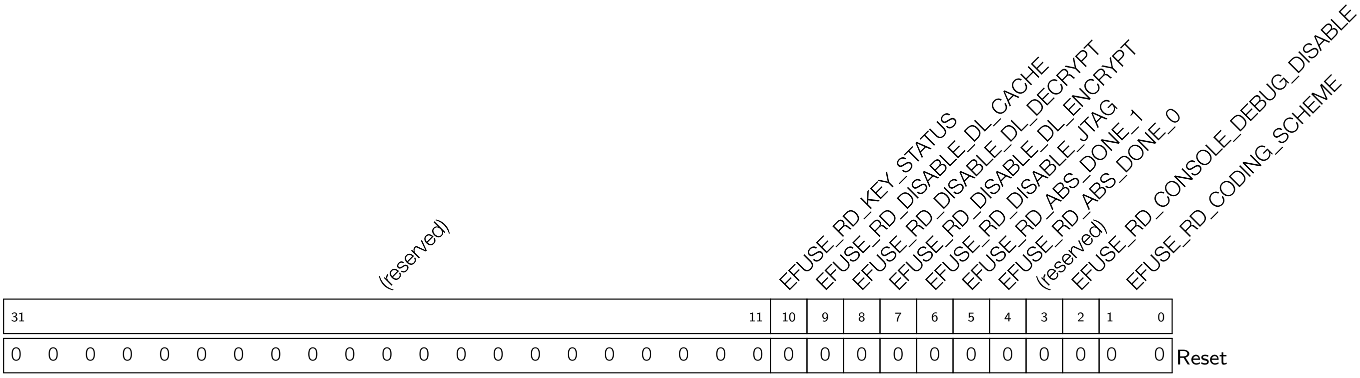
EFUSE_RD_SPI_PAD_CONFIG_CS0 SPI_pad_config_cs0 的值。（只读）

EFUSE_RD_SPI_PAD_CONFIG_D SPI_pad_config_d 的值。（只读）

EFUSE_RD_SPI_PAD_CONFIG_Q SPI_pad_config_q 的值。（只读）

EFUSE_RD_SPI_PAD_CONFIG_CLK SPI_pad_config_clk 的值。（只读）

Register 20.7. EFUSE_BLK0_RDATA6_REG (0x018)



EFUSE_RD_KEY_STATUS key_status 的值。（只读）

EFUSE_RD_DISABLE_DL_CACHE download_dis_cache 的值。（只读）

EFUSE_RD_DISABLE_DL_DECRYPT download_dis_decrypt 的值。（只读）

EFUSE_RD_DISABLE_DL_ENCRYPT download_dis_encrypt 的值。（只读）

EFUSE_RD_DISABLE_JTAG JTAG_disable 的值。（只读）

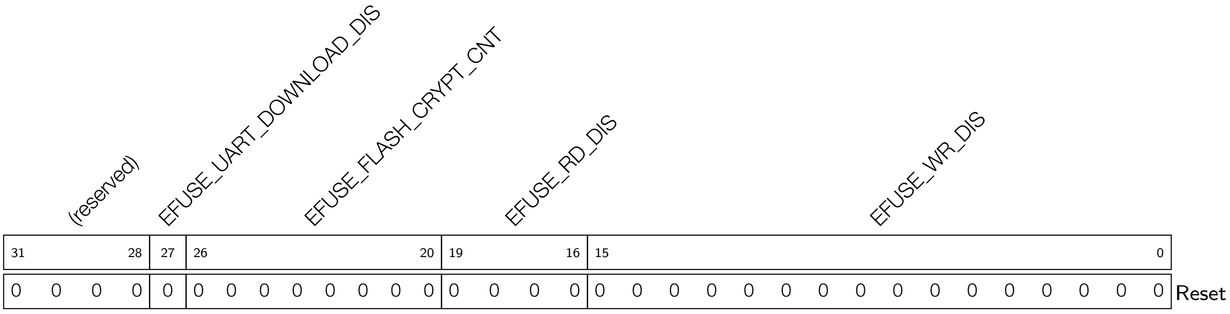
EFUSE_RD_ABS_DONE_1 abstract_done_1 的值。（只读）

EFUSE_RD_ABS_DONE_0 abstract_done_0 的值。（只读）

EFUSE_RD_CONSOLE_DEBUG_DISABLE console_debug_disable 的值。（只读）

EFUSE_RD_CODING_SCHEME coding_scheme 的值。（只读）

Register 20.8. EFUSE_BLK0_WDATA0_REG (0x01c)



EFUSE_UART_DOWNLOAD_DIS 烧写 uart_download_dis 的值，仅对 ESP32 ECO V3 有效。（读/写）

EFUSE_FLASH_CRYPT_CNT 烧写 flash_crypt_cnt 的值。（读/写）

EFUSE_RD_DIS 烧写 efuse_rd_disable 的值。（读/写）

EFUSE_WR_DIS 烧写 efuse_wr_disable 的值。（读/写）

Register 20.9. EFUSE_BLK0_WDATA1_REG (0x020)

31 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

EFUSE_BLK0_WDATA1_REG 烧写 WIFI_MAC_Address 低 32 位的值。（读/写）

Register 20.10. EFUSE_BLK0_WDATA2_REG (0x024)

(reserved)	EFUSE_WIFI_MAC_CRC_HIGH	
31 24	23 0	
0 0 0 0 0 0 0 0	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_WIFI_MAC_CRC_HIGH 烧写 WIFI_MAC_Address 高 24 位的值。（读/写）

Register 20.11. EFUSE_BLK0_WDATA3_REG (0x028)

(reserved)	EFUSE_CHIP_VER_PKG	EFUSE_SPI_PAD_CONFIG_HD	EFUSE_CHIP_VER_DIS_CACHE	EFUSE_CHIP_VER_PKG	EFUSE_CHIP_VER_DIS_BT	EFUSE_CHIP_VER_DIS_APP_CPU	
31 12	11 9	8 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	Reset

EFUSE_CHIP_VER_PKG 共有四个 bit 供烧写封装版本，这是前三位。（读/写）

EFUSE_SPI_PAD_CONFIG_HD 烧写 SPI_pad_config_hd 的值。（读/写）

EFUSE_CHIP_VER_DIS_CACHE 烧写后禁用 cache 功能。（读/写）

EFUSE_CHIP_VER_PKG 共有四个 bit 供烧写封装版本，这是最高位。读/写）

EFUSE_CHIP_VER_DIS_BT 烧写后禁用蓝牙。（读/写）

EFUSE_CHIP_VER_DIS_APP_CPU 烧写后禁用 APP CPU。（读/写）

Register 20.12. EFUSE_BLK0_WDATA4_REG (0x02c)

(reserved)	EFUSE_SDIO_FORCE	EFUSE_SDIO_TIEH	EFUSE_XPD_SDIO	(reserved)	ESFUSE_CK8M_FREQ	
31 17	16	15	14	13 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0 0 0	0 0 0 0 0 0 0 0	Reset

EFUSE_SDIO_FORCE 烧写 SDIO_TIEH 的值。(读/写)

EFUSE_SDIO_TIEH 烧写 SDIO_TIEH 的值。(读/写)

EFUSE_XPD_SDIO 烧写 XPD_SDIO_REG 的值。(读/写)

ESFUSE_CK8M_FREQ 烧写 RC_FAST_CLK 频率。(读/写)

Register 20.13. EFUSE_BLK0_WDATA5_REG (0x030)

EFUSE_FLASH_CRYPT_CONFIG	EFUSE_DIG_VOL_L6	EFUSE_VOL_LEVEL_HP_INV	(reserved)	EFUSE_SPI_PAD_CONFIG_CS0	EFUSE_SPI_PAD_CONFIG_D	EFUSE_SPI_PAD_CONFIG_Q	EFUSE_SPI_PAD_CONFIG_CLK	
31 28	27 24	23 22	21 20	19 15	14 10	9 5	4 0	
0 0 0 0	0 0 0 0	0 0	0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	Reset

EFUSE_FLASH_CRYPT_CONFIG 烧写 flash_crypt_config 的值。(读/写)

EFUSE_DIG_VOL_L6 烧写档位 6 的 dig regulator 的电压相对于 1.2 V 的差值。(读/写)

EFUSE_VOL_LEVEL_HP_INV 烧写 CPU 跑 240 MHz 或 flash/PSRAM 跑 80 MHz 需要的电压档位。0x0:采用档位 7;0x1:采用档位 6;0x2:采用档位 5;0x3:采用档位 4。(读/写)

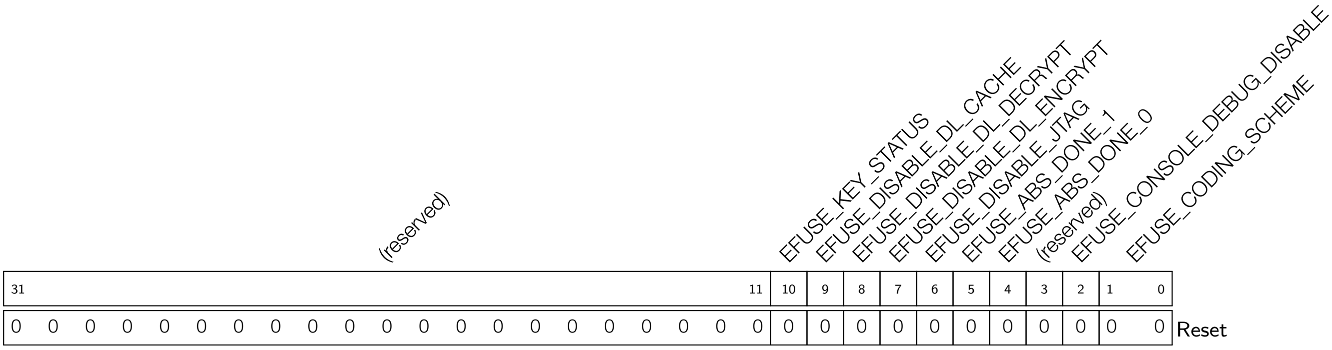
EFUSE_SPI_PAD_CONFIG_CS0 烧写 SPI_pad_config_cs0 的值。(读/写)

EFUSE_SPI_PAD_CONFIG_D 烧写 SPI_pad_config_d 的值。(读/写)

EFUSE_SPI_PAD_CONFIG_Q 烧写 SPI_pad_config_q 的值。(读/写)

EFUSE_SPI_PAD_CONFIG_CLK 烧写 SPI_pad_config_clk 的值。(读/写)

Register 20.14. EFUSE_BLK0_WDATA6_REG (0x034)



EFUSE_KEY_STATUS 烧写 key_status 的值。(读/写)

EFUSE_DISABLE_DL_CACHE 烧写 download_dis_cache 的值。(读/写)

EFUSE_DISABLE_DL_DECRYPT 烧写 download_dis_decrypt 的值。(读/写)

EFUSE_DISABLE_DL_ENCRYPT 烧写 download_dis_encrypt 的值。(读/写)

EFUSE_DISABLE_JTAG 烧写 JTAG_disable 的值。(读/写)

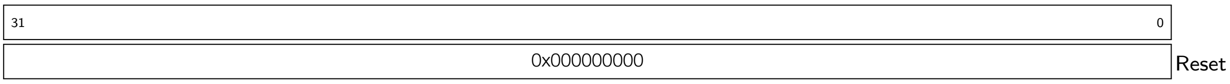
EFUSE_ABS_DONE_1 烧写 abstract_done_1 的值。(读/写)

EFUSE_ABS_DONE_0 烧写 abstract_done_0 的值。(读/写)

EFUSE_CONSOLE_DEBUG_DISABLE 烧写 console_debug_disable 的值。(读/写)

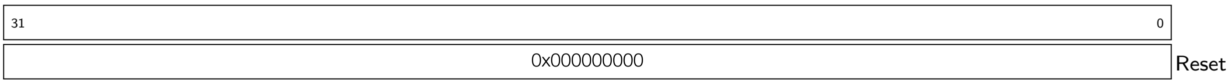
EFUSE_CODING_SCHEME 烧写 coding_scheme 的值。(读/写)

Register 20.15. EFUSE_BLK1_RDATA*n*_REG (*n*: 0-7) (0x38+4**n*)



EFUSE_BLK1_RDATA*n*_REG BLOCK1 中 word *n* 的值。(只读)

Register 20.16. EFUSE_BLK2_RDATA*n*_REG (*n*: 0-7) (0x58+4**n*)



EFUSE_BLK2_RDATA*n*_REG BLOCK2 中 word *n* 的值。(只读)

Register 20.17. EFUSE_BLK3_RDATA*n*_REG (*n*: 0-7) (0x78+4**n*)

31 0	
0x000000000	Reset

EFUSE_BLK3_RDATA*n*_REG BLOCK3 中 word *n* 的值。（只读）

Register 20.18. EFUSE_BLK1_WDATA*n*_REG (*n*: 0-7) (0x98+4**n*)

31 0	
0x000000000	Reset

EFUSE_BLK1_WDATA*n*_REG BLOCK1 中 word *n* 的值。（读/写）

Register 20.19. EFUSE_BLK2_WDATA*n*_REG (*n*: 0-7) (0xB8+4**n*)

31 0	
0x000000000	Reset

EFUSE_BLK2_WDATA*n*_REG BLOCK2 中 word *n* 的值。（读/写）

Register 20.20. EFUSE_BLK3_WDATA*n*_REG (*n*: 0-7) (0xD8+4**n*)

31 0	
0x000000000	Reset

EFUSE_BLK3_WDATA*n*_REG BLOCK3 中 word *n* 的值。（读/写）

Register 20.21. EFUSE_CLK_REG (0x0f8)

(reserved)	EFUSE_CLK_SEL1	EFUSE_CLK_SEL0	
31 16	15 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x040	0x052	Reset

EFUSE_CLK_SEL1 eFuse 时钟配置字段。（读/写）

EFUSE_CLK_SEL0 eFuse 时钟配置字段。（读/写）

Register 20.22. EFUSE_CONF_REG (0x0fc)

(reserved)	EFUSE_OP_CODE	
31 16	15 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00000	Reset

EFUSE_OP_CODE eFuse 操作码寄存器。（读/写）

Register 20.23. EFUSE_CMD_REG (0x104)

(reserved)	EFUSE_PGM_CMD	EFUSE_READ_CMD	
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	Reset

EFUSE_PGM_CMD 将此位置 1 以开始烧写操作；烧写完成后恢复为 0。（读/写）

EFUSE_READ_CMD 将此位置 1 以开始读取操作；读取完成后恢复为 0。（读/写）

Register 20.24. EFUSE_INT_RAW_REG (0x108)

(reserved)	EFUSE_PGM_DONE_INT_RAW	EFUSE_READ_DONE_INT_RAW	
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	Reset

EFUSE_PGM_DONE_INT_RAW EFUSE_PGM_DONE_INT 中断的原始中断状态位。（只读）

EFUSE_READ_DONE_INT_RAW EFUSE_READ_DONE_INT 中断的原始中断状态位。（只读）

Register 20.25. EFUSE_INT_ST_REG (0x10c)

(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	Reset

EFUSE_PGM_DONE_INT_ST EFUSE_PGM_DONE_INT 中断的屏蔽中断状态位。（只读）

EFUSE_READ_DONE_INT_ST EFUSE_READ_DONE_INT 中断的屏蔽中断状态位。（只读）

Register 20.26. EFUSE_INT_ENA_REG (0x110)

(reserved)	EFUSE_PGM_DONE_INT_ENA	EFUSE_READ_DONE_INT_ENA	
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	Reset

EFUSE_PGM_DONE_INT_ENA EFUSE_PGM_DONE_INT 中断的中断使能位。（读/写）

EFUSE_READ_DONE_INT_ENA EFUSE_READ_DONE_INT 中断的中断使能位。（读/写）

Register 20.27. EFUSE_INT_CLR_REG (0x114)

(reserved)	EFUSE_PGM_DONE_INT_CLR	EFUSE_READ_DONE_INT_CLR	
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	Reset

EFUSE_PGM_DONE_INT_CLR 将此位置 1 以清除 EFUSE_PGM_DONE_INT 中断。（只写）

EFUSE_READ_DONE_INT_CLR 将此位置 1 以清除 EFUSE_READ_DONE_INT 中断。（只写）

Register 20.28. EFUSE_DAC_CONF_REG (0x118)

(reserved)	EFUSE_DAC_CLK_DIV	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	40	Reset

EFUSE_DAC_CLK_DIV eFuse 时序配置寄存器。（读/写）

Register 20.29. EFUSE_DEC_STATUS_REG (0x11c)

(reserved)	EFUSE_DEC_WARNINGS	
31 12	11 0	
0 0 0 0 0 0 0 0 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_DEC_WARNINGS 若此寄存器的某位置 1，则解码 3/4 编码方式时更正某些错误。（只读）

21 双线汽车接口 (TWAI)

21.1 概述

双线汽车接口 (Two-wire Automotive Interface, TWAI ®) 协议是一种多主机、多播的通信协议，具有检测错误、发送错误信号以及内置报文优先仲裁等功能。TWAI 协议适用于汽车和工业应用（可参见 TWAI 协议描述）。

ESP32 包含一个 TWAI 控制器，可通过外部收发器连接到 TWAI 总线。TWAI 控制器包含一系列先进的功能，用途广泛，可用于如汽车产品、工业自动化控制、楼宇自动化等。

21.2 主要特性

ESP32 TWAI 控制器具有以下特性：

- 兼容 ISO 11898-1 协议（CAN 规范 2.0）
- 支持标准格式（11-bit 标识符）和扩展格式（29-bit 标识符）
- 支持速率：
  - 芯片版本 v0.0/v1.0/v1.1 支持 25 Kbit/s ~ 1 Mbit/s 位速率
  - 芯片版本 v3.0/v3.1 支持 12.5 Kbit/s ~ 1 Mbit/s 位速率
- 支持多种操作模式
  - 正常模式
  - 只听模式（不影响总线）
  - 自测模式（发送数据时不需应答）
- 64-byte 接收 FIFO
- 特殊发送
  - 单次发送（发生错误时不会自动重新发送）
  - 自发自收（TWAI 控制器同时发送和接收报文）
- 接收滤波器（支持单滤波器和双滤波器模式）
- 错误检测与处理
  - 错误计数
  - 错误报警限制可配置
  - 错误代码捕捉
  - 仲裁丢失捕捉

21.3 功能性协议

21.3.1 TWAI 性能

TWAI 协议连接总线网络中的两个或多个节点，并允许各节点以延迟限制的形式进行报文交互。TWAI 总线具有以下性能：

单通道通信与不归零编码： TWAI 总线由承载着位的单通道组成，因此为半双工通信。同步调整也在单通道中进行，因此不需其他通道（如时钟通道和使能通道）。TWAI 上报文的位流采用不归零编码 (NRZ) 方式。

位值： 单通道可处于显性状态或隐性状态，显性状态的逻辑值为 0，隐性状态的逻辑值为 1。发送显性状态数据的节点总是比发送隐性状态数据的节点优先级高。总线上的其他物理功能（如，差分电平）由其各自应用实现。

位填充： TWAI 报文的某些域已经过位填充。每发送某个相同值（如显性数值或隐性数值）的连续五个位后，需自动插入一个互补位。同理，接收到 5 个连续位的接收器应将下一个位视为填充位。位填充应用于以下域：SOF、仲裁域、控制域、数据域和 CRC 序列（可参见第 21.3.2 章）。

多播： 当各节点连接到同个总线上时，这些节点将接收相同的位。各节点上的数据将保持一致，除非发生总线错误（可参见第 21.3.3 章）。

多主机： 任意节点都可发起数据传输。如果当前已有正在进行的数据传输，则节点将等待当前传输结束后再发起其数据传输。

报文优先级与仲裁： 若两个或多个节点同时发起数据传输，则 TWAI 协议将确保其中一个节点获得总线的优先仲裁权。各节点所发送报文的仲裁域决定哪个节点可以获得优先仲裁。

错误检测与通报： 各节点将积极检测总线上的错误，并通过发送错误帧来通报检测到的错误。

故障限制： 若一组错误计数依据规定增加/减少时，各节点将维护该组错误计数。当错误计数超过一定阈值时，对应节点将自动关闭以退出网络。

可配置位速率： 单个 TWAI 总线的位速率是可配置的。但是，同个总线中的所有节点须以相同位速率工作。

发送器与接收器： 不论何时，任意 TWAI 节点都可作为发送器和接收器。

- 产生报文的节点为发送器。且该节点将一直作为发送器，直到总线空闲或该节点失去仲裁。请注意，未丢失仲裁的多个节点都可作为发送器。
- 所有非发送器的节点都将作为接收器。

21.3.2 TWAI 报文

TWAI 节点使用报文发送数据，并在监测到总线上存在错误时向其他节点发送错误信号。报文分为不同的帧类型，某些帧类型将具有不同的帧格式。

TWAI 协议有以下帧类型：

- 数据帧
- 远程帧
- 错误帧
- 过载帧
- 帧间距

TWAI 协议有以下帧格式：

- 标准格式 (SFF) 由 11-bit 标识符组成
- 扩展格式 (EFF) 由 29-bit 标识符组成

21.3.2.1 数据帧和远程帧

节点使用数据帧向其他节点发送数据，可负载 0 ~ 8 字节数据。节点使用远程帧向其他节点请求具有相同标识符的数据帧，因此远程帧中不包含任何数据字节。但是，数据帧和远程帧中包含许多相同域。下图 21-1 所示为不同帧类型和不同帧格式中包含的域和子域。

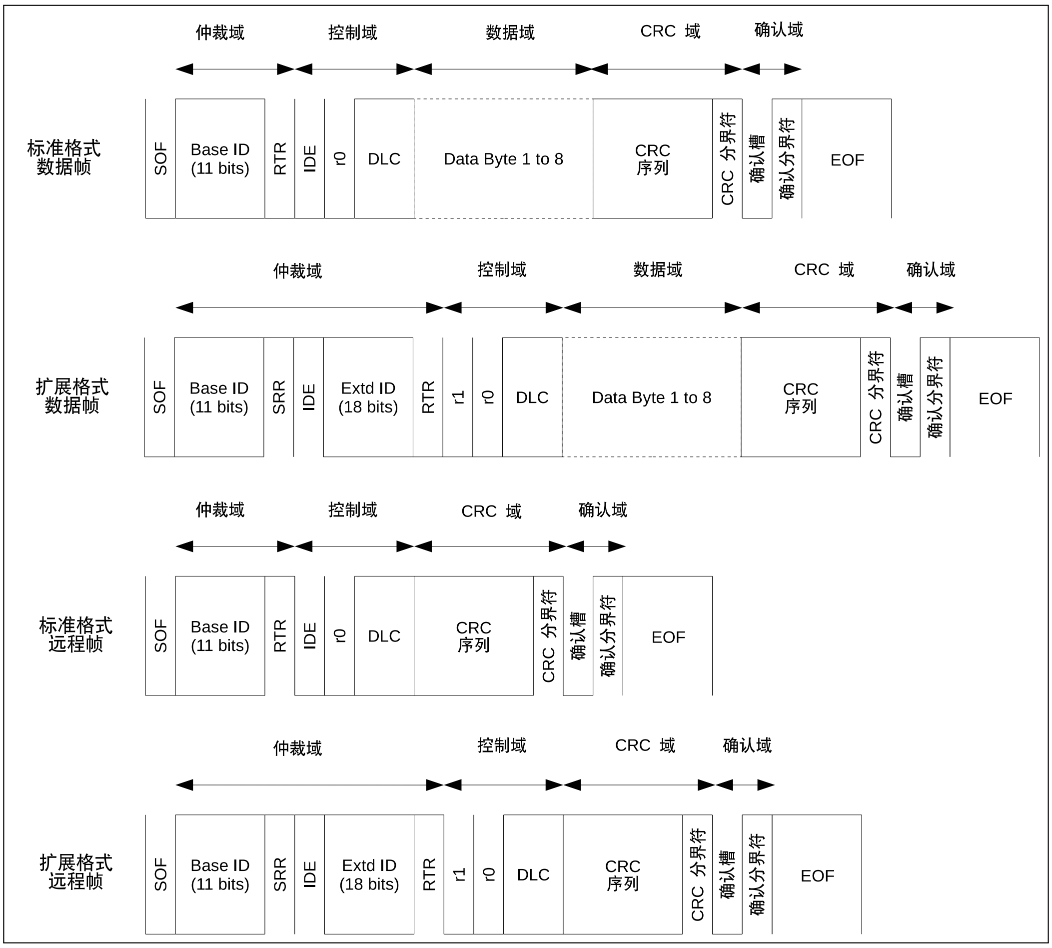


图 21-1. 数据帧和远程帧中的位域

仲裁域

当两个或多个节点同时发送数据帧和远程帧时，将根据仲裁域的位信息来决定总线上获得优先仲裁的节点。在仲裁域作用时，如果一个节点在发送隐性位的同时检测到了一个显性位，这表示有其他节点优先于了这个隐性位。那么，这个发送隐性位的节点已丢失总线仲裁，应立即转为接收器。

仲裁域主要由帧标识符组成。根据显性位代表的逻辑值为 0，隐性位代表的逻辑值为 1，有以下规律:

- ID 值最小的帧将总是获得仲裁。
- 如果 ID 和格式相同，由于数据帧的 RTR 位为显性位，数据帧将优先于远程帧。
- 如果 ID 的前 11 位相同，由于扩展帧的 SRR 位是隐性，因而标准格式帧将总优先于扩展格式帧。

控制域

控制域主要由数据长度代码 (DLC) 组成，DLC 表示一个数据帧中的负载的数据字节数量，或一个远程帧请求的数据字节数量。DLC 优先发送最高有效位。

数据域

数据域中包含一个数据帧真实负载的数据字节。远程帧中不包含数据域。

CRC 域

CRC 域主要由 CRC 序列组成。CRC 序列是一个 15-bit 的循环冗余校验编码，根据数据帧或远程帧中的未填充内容（从 SOF 到数据域末尾的所有内容）中计算而来。

确认域

确认 (ACK) 域由确认槽和确认分界符组成，主要功能为：接收器向发送器报告已正确接收到有效报文。

表 21-1. SFF 和 EFF 中的数据帧和远程帧

数据/远程帧	描述
SOF	帧起始 (SOF) 是一个用于同步总线上节点的单个显性位。
Base ID	基标识符 (ID.28 ~ ID.18) 是 SFF 的 11-bit 标识符，或者是 EFF 中 29-bit 标识符的前 11-bit。
RTR	远程发送请求位 (RTR) 显示当前报文是数据帧（显性）还是远程帧（隐性）。这意味着，当某个数据帧和一个远程帧有相同标识符时，数据帧始终优先于远程帧仲裁。
SRR	在 EFF 中发送替代远程请求位 (SRR)，以替代 SFF 中相同位置的 RTR 位。
IDE	标识符扩展位 (IED) 显示当前报文是 SFF （显性）还是 EFF（隐性）。这意味着，当某 SFF 帧和 EFF 帧有相同基标识符时，SFF 帧将始终优先于 EFF 帧仲裁。
Extd ID	扩展标识符 (ID.17 ~ ID.0) 是 EFF 中 29-bit 标识符的剩余 18-bit。
r1	r1（保留位 1）始终是显性位。
r0	r0（保留位 0）始终是显性位。
DLC	数据长度代码 (DLC) 为 4-bits，且应包含 0 ~ 8 中任一数值。数据帧使用 DLC 表示自身包含的数据字节数量。远程帧使用 DLC 表示从其他节点请求的数据字节数量。
数据字节	表示数据帧的数据负载量。该字节数量应与 DLC 的值匹配。首先发送数据字节 0，各数据字节优先发送最高有效位。
CRC 序列	CRC 序列是一个 15-bit 的循环冗余校验编码。
CRC 分界符	CRC 分界符是遵循 CRC 序列的单一隐性位。
确认槽	确认槽用于接收器节点，表示是否已成功接收数据帧或远程帧。发送器节点将在确认槽中发送一个隐性位，如果接收到的帧没有错误，则接收器节点应用一个显性位替代确认槽。
确认分界符	确认分界符是一个单一的隐性位。
EOF	帧结束 (EOF) 标志着数据帧或远程帧的结束，由七个隐性位组成。

21.3.2.2 错误帧和过载帧

错误帧

当某节点检测到总线错误时，将发送一个错误帧。错误帧由一个特殊的错误标志构成，该标志由某相同值的六个连续位组成，因而违反了位填充的规则。所以，当某节点检测到总线错误并发送错误帧时，其余节点也将相应地检测到一个填充错误并各自发送错误帧。也就是说，当发生总线错误时，通过上述过程可将该报文传递至总线上的所有节点。

当某节点检测到总线错误时，该节点将于下一个位发送错误帧。特例：如果总线错误类型为 CRC 错误，那么错误帧将从确认分界符的下一个位开始（可参见第 21.3.3 章）。下图 21-2 所示为一个错误帧所包含的不同域：

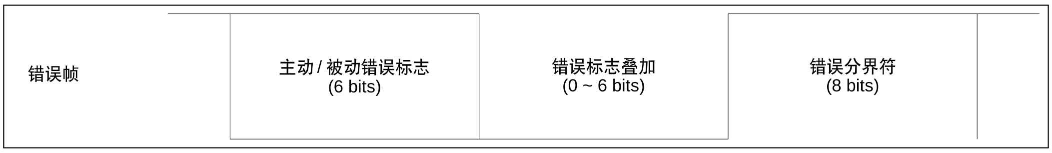


图 21-2. 错误帧中的位域

表 21-2. 错误帧

错误帧	描述
错误标志	错误标志包括两种形式: 主动错误标志和被动错误标志，主动错误标志由 6 个显性位组成，被动错误标志由 6 个隐性位组成（被其他节点的显性位优先仲裁时除外）。主动错误节点发送主动错误标志，被动错误节点发送被动错误标志。
错误标志叠加	错误标志叠加域的主要目的是允许总线上的其他节点发送各自的主动错误标志。叠加域的范围可以是 0 ~ 6 位，在检测到第一个隐性位时结束（如检测到分界符上的第一个位时）。
错误分界符	分界符域标志着错误/过载帧结束，由 8 个隐性位构成。

过载帧

过载帧与包含主动错误标志的错误帧有着相同的位域。二者主要区别在于触发发送过载帧的条件。下图 21-3 所示为过载帧中包含的位域：

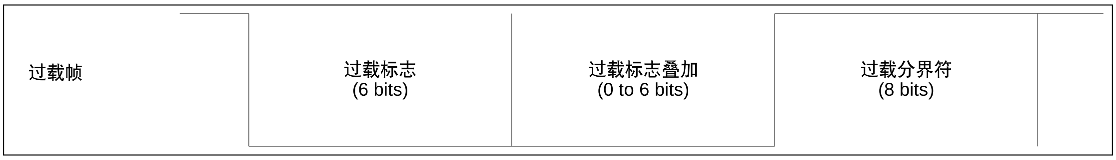


图 21-3. 过载帧中的位域

表 21-3. 过载帧

过载帧	描述
过载标志	由 6 个显性位构成。与主动错误标志相同。
过载标志叠加	允许其他节点发送过载标志的叠加，与错误标志叠加相似。
过载分界符	由 8 个隐性位构成。与错误分界符相同。

下列情况将触发发送过载帧：

1. 接收器内部要求延迟发送下一个数据帧或远程帧。
2. 在间歇域后的首个和第二个位上检测到显性位。
3. 如果在错误分界符的第八个（最后一个）位上检测到显性位。请注意，在这种情况下 TEC 和 REC 的值将不会增加（可参见第 21.3.3 章）。

由于上述情况发送过载帧时，须满足以下规定：

- 第 1 条情况下发送的过载帧只能从间歇域后的第一个位开始。
- 第 2、3 条情况下发送的过载帧须从检测到显性位的后一个位开始。
- 要延迟发送下一个数据帧或远程帧，最多可生成两个过载帧。

21.3.2.3 帧间距

帧间距充当各帧之间的分隔符。数据帧和远程帧必须与前一帧用一个帧间距分隔开，不论前面的帧是何类型（数据帧、远程帧、错误帧、过载帧）。但是，错误帧和过载帧则无需与前一个帧分隔开。

下图 21-4 所示为帧间距中包含的域：

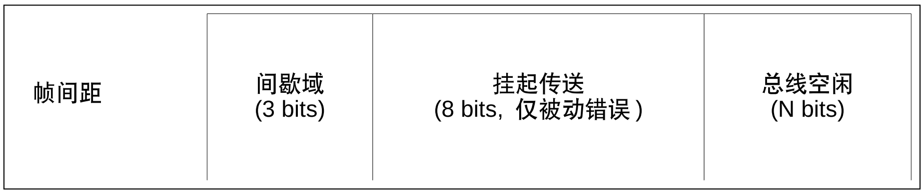


图 21-4. 帧间距中的域

表 21-4. 帧间距

帧间距	描述
间歇域	间歇域由 3 个隐性位构成。
挂起传送	被动错误节点发送报文后，节点中须包含一个挂起传送域，由 8 个隐性位构成。主动错误节点中不含这个域。
总线空闲	总线空闲域长度任意。发送 SOF 时，总线空闲结束。若节点中有挂起传送，则 SOF 应在间歇域后的第一位发送。

21.3.3 TWAI 错误

21.3.3.1 错误类型

TWAI 中的总线错误包括以下类型：

位错误
当节点发送一个位值（显性位或隐性位）但检测到相反的位时（如，发送显性位时检测到了隐性位），就会发生位错误。但是，如果发送的位是隐性位，且位于仲裁域或确认槽或被动错误标志中，那么此时检测到显性位的话也不会认定为位错误。

填充错误
当检测到相同值的 6 个连续位时（违反位填充的编码规则），发生填充错误。

CRC 错误
数据帧和远程帧的接收器将根据接收到的位计算 CRC 值。当接收器计算的值与接收到的数据帧和远程帧中的 CRC 序列不匹配时，会发生 CRC 错误。

格式错误
当某个报文中的固定格式位中包含非法位时，可检测到格式错误。比如，r1 和 r0 域必须固定为显性。

确认错误
当发送器无法在确认槽中检测到显性位时，将发生确认错误。

21.3.3.2 错误状态

TWAI 通过每个节点维护两个错误计数来实现故障界定，计数数值决定错误状态。这两个错误计数分别为：发送错误计数 (TEC) 和接收错误计数 (REC)。TWAI 包含以下错误状态。

主动错误
主动错误节点可参与到总线交互中，且在检测到错误时可以发送主动错误标志。

被动错误
被动错误节点可参与到总线交互中，但在检测到错误时只能发送一次被动错误标志。被动错误节点发送数据帧或远程帧后，须在后续的帧间距中设置挂起传送域。

离线
禁止离线节点以任意方式干扰总线（如，不允许其进行数据传输）。

21.3.3.3 错误计数

TEC 和 REC 根据以下规则递增/递减。**请注意，一条报文传输中可应用多个规则。**

1. 当接收器检测到错误时，REC 数值将增加 1。当检测到的错误为发送主动错误标志或过载标志期间的位错误除外。
2. 发送错误标志后，当接收器第一个检测到的位是显性位时，REC 数值将增加 8。
3. 当发送器发送错误标志时，TEC 数值增加 8。但是，以下情况不适用于该规则：
    - 发送器为被动错误状态，因为在应答槽未检测到显性位而产生应答错误，且在发送被动错误标志时检测到显性位时，则 TEC 数值不应增加。

  - 发送器在仲裁期间因填充错误而发送错误标志，且填充位本该是隐性位但是检测到显性位，则 TEC 数值不应增加。

4. 若发送器在发送主动错误标志和过载标志时检测到位错误，则 TEC 数值增加 8。
5. 若接收器在发送主动错误标志和过载标志时检测到位错误，则 REC 数值增加 8。
6. 任意节点在发送主动/被动错误标志或过载标志后，节点仅能承载最多 7 个连续显性位。在（发送主动错误标志或过载标志时）检测到第 14 个连续显性位，或在被动错误标志后检测到第 8 个连续显性位后，发送器将使其 TEC 数值增加 8，而接收器将使其 REC 数值增加 8。每增加 8 个连续显性位的同时，（发送器的）TEC 和（接收器的）REC 数值也将增加 8。
7. 每当发送器成功发送报文后（接收到 ACK，且直到 EOF 完成未发生错误），TEC 数值将减小 1，除非 TEC 的数值已经为 0。
8. 当接收器成功接收报文后（确认槽前未检测到错误，且成功发送 ACK），则 REC 数值将相应减小。
    - 若 REC 数值位于 1 ~ 127 之间，则其值减小 1。
    - 若 REC 数值大于 127，则其值减小到 127。
    - 若 REC 数值为 0，则仍保持为 0。
9. 当一个节点的 TEC 和/或 REC 数值大于等于 128 时，该节点变为被动错误节点。导致节点发生上述状态切换的错误，该节点仍发送主动错误标志。请注意，一旦 REC 数值到达 128，后续任何增加该值的动作都是无效的，直到 REC 数值返回到 128 以下。
10. 当某节点的 TEC 数值大于等于 256 时，该节点将变为离线节点。
11. 当某被动错误节点的 TEC 和 REC 数值都小于等于 127，则该节点将变为主动错误节点。
12. 当离线节点在总线上检测到 128 次 11 个连续隐性位后，该节点可变为主动错误节点（TEC 和 REC 数值都重设为 0）。

21.3.4 TWAI 位时序

21.3.4.1 名义位

TWAI 协议允许 TWAI 总线以特定的位速率运行。但是，总线内的所有节点必须以统一位速率运行。

- **名义位速率**为每秒发送比特数量。
- **名义位时间**为 **1/名义位速率**。

每个名义位时间中含多个段，每段由多个时间定额 (Time Quanta) 组成。**时间定额**为最小时间单位，作为一种预分频时钟信号应用于各个节点中。下图 21-5 所示为一个名义位时间内所包含的段。

TWAI 控制器将在一个时间定额的时间步长中进行操作，每个时间定额中都会分析 TWAI 的总线状态。如果两个连续的时间定额中总线状态不同（隐性-显性，或反之），意味着有边沿产生。PBS1 和 PBS2 的交点将被视为采样点，且采样的总线数值即为这个位的数值。

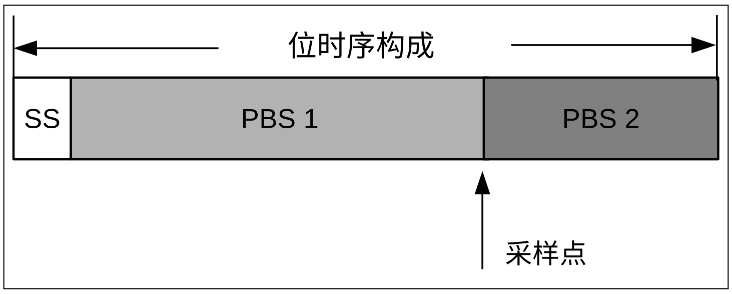


图 21-5. 位时序构成

表 21-5. 名义位时序中包含的段

段	描述
同步段 (SS)	SS（同步段）的长度为 1 个时间定额。若所有节点都同步正常，则位边沿应位于该段内。
缓冲时期段 1 (PBS1)	PBS1 的长度可为 1 ~ 16 个时间定额，用于补偿网络中的物理延迟时间。可增加 PBS1 的长度，从而更好地实现同步。
缓冲时期段 2 (PBS2)	PBS2 的长度可为 1 ~ 8 个时间定额，用于补偿节点中的信息处理时间。可缩短 PBS2 的长度，从而更好地实现同步。

21.3.4.2 硬同步与再同步

由于时钟偏移和抖动，同一总线上节点的位时序可能会脱离相位段。因而，位边沿可能会偏移到同步段的前后。针对上述位边沿偏移的问题 TWAI 提供多种同步方式。设位边沿偏移的 TQ（时间定额）数量为**相位错误 “e”**，该值与 SS 相关。

- 主动相位错误 (e > 0)：位边沿位于同步段之后采样点之前（即，边沿向后偏移）。
- 被动相位错误 (e < 0)：位边沿位于前个位的采样点之后同步段之前（即，边沿向前偏移）。

为解决相位错误，可进行两种同步方式，即**硬同步**与**再同步**。**硬同步**与**再同步**遵守以下规则：

- 单个位时序中仅可发生一次同步。
- 同步仅可发生在隐性位到显性位的边沿上。

硬同步
总线空闲期间，硬同步发生在隐性位到显性位的变化边沿上（如 SOF 位上）。此时，所有节点都将重启其内部位时序，从而使该变化边沿位于重启位时序的同步段内。

再同步
非总线空闲期间，再同步发生在隐性位到显性位的变化边沿上。如果边沿上有主动相位错误 (e > 0)，则 PBS1 长度将增加。如果边沿上有被动相位错误 (e < 0)，则 PBS2 长度将减小。

PBS1/PBS2 具体增加和减小的时间定额取决于相位错误的绝对值，同时也受可配置的同步跳宽 (SJW) 数值限制。

- 当相位错误的绝对值小于等于 SJW 数值时，PBS1/PBS2 将增加/减小 **e** 个时间定额。该过程与硬同步具有相同效果。
- 当相位错误的绝对值大于 SJW 数值时，PBS1/PBS2 将增加/减小与 SJW 相同数值的时间定额。这意味着，在完全解决相位错误之前，可能需要多个同步位。

21.4 结构概述

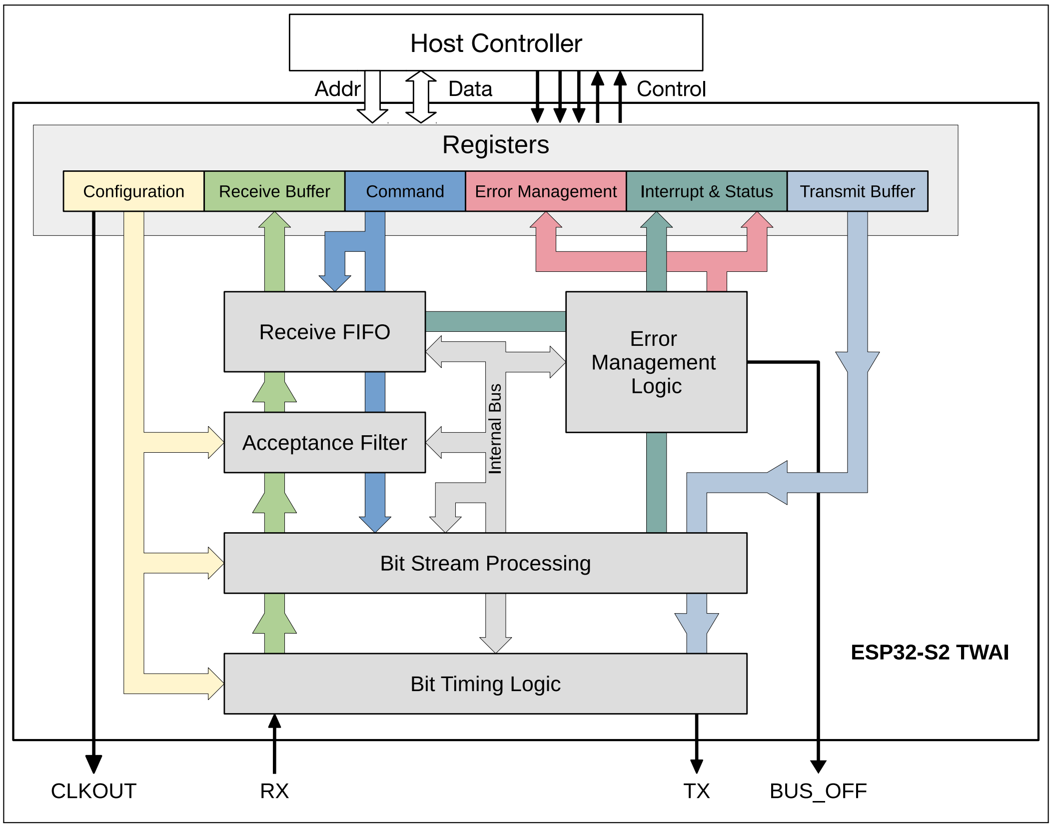


图 21-6. TWAI 概略图

ESP32 中包含一个 TWAI 控制器。图 21-6 所示为 TWAI 控制器的主要功能模块。

21.4.1 寄存器模块

ESP32 的 CPU 使用 32-bit 对齐字访问外设。但是，TWAI 控制器中的大部分寄存器仅存储最低有效字节 (bits [7:0]) 上的有用数据。因此在这些寄存器中，bits [31:8] 在写入时被忽略，在读取时返回 0。

配置寄存器

配置寄存器存储 TWAI 控制器的各配置项，如位速率、操作模式、接收滤波器等。只有在 TWAI 控制器处于复位模式时，才可修改配置寄存器（可参见第 21.5.1 章）。

指令寄存器

CPU 通过指令寄存器驱动 TWAI 控制器执行任务，如发送报文或清除接收缓冲器。只有在 TWAI 控制器处于操作模式时，才可修改指令寄存器（可参见第 21.5.1 章）。

中断 & 状态寄存器

中断寄存器显示 TWAI 控制器中发生的事件（每个事件由一个单独的位表示）。状态寄存器显示 TWAI 控制器的当前状态。

错误管理寄存器

错误管理寄存器包括错误计数和捕捉寄存器。错误计数寄存器表示 TEC 和 REC 的数值。捕捉寄存器负责记录相关信息，如 TWAI 控制器在何处检测到总线错误，或何时丢失仲裁。

发送缓冲寄存器

发送缓冲器大小为 13 字节，用于存储 TWAI 的待发送报文。

接收缓冲寄存器

接收缓冲器大小为 13 字节，用于存储单个报文。接收缓冲器是进入接收 FIFO 的窗口，接收 FIFO 中的第一个报文将被映射到接收缓冲器中。

请注意，发送缓冲寄存器、接收缓冲寄存器和接收滤波寄存器的地址范围相同（地址偏移包含 0x0040 ~ 0x0070）。这些寄存器的访问权限遵循以下规则：

- 当 TWAI 控制器处于复位模式时，该地址范围被映射到接收滤波寄存器中。
- TWAI 控制器处于操作模式时：
  - 对地址范围的所有读取都映射于接收缓冲寄存器中。
  - 对地址范围的所有写入都映射于发送缓冲寄存器中。

21.4.2 位流处理器

位流处理 (BSP) 模块负责对发送缓冲器的数据进行帧处理 (如，位填充和附加 CRC 域) 并为位时序逻辑 (BTL) 模块生成位流。同时，BSP 模块还负责处理从 BTL 模块中接收的位流（如，去填充和验证 CRC），并将处理报文置于接收 FIFO。BSP 还负责检测 TWAI 总线上的错误并将此类错误报告给错误管理逻辑 (EML)。

21.4.3 错误管理逻辑

错误管理逻辑 (EML) 模块负责更新 TEC 和 REC 数值，记录错误信息（如，错误类型和错误位置），更新控制器的错误状态，确保 BSP 模块发送正确的错误标志。此外，该模块还负责记录 TWAI 控制器丢失仲裁时的 bit 位置。

21.4.4 位时序逻辑

位时序逻辑 (BTL) 模块负责以预先配置的位速率发送和接受报文。BTL 模块还负责同步位时序，确保数据传输的稳定性。位速率由多个可编程的段组成，且用户可设置每个段的 TQ（时间定额）长度，来调整报文传输速率。

21.4.5 接收滤波器

接收滤波器是一个可编程的报文过滤单元，允许 TWAI 控制器根据报文的标识符域接收或拒绝该报文。通过接收滤波器的报文才能被存储到接收 FIFO 中。用户可配置接收滤波器的模式：单滤波器、双滤波器。

21.4.6 接收 FIFO

接收 FIFO 是大小为 64-byte 的缓冲器（位于 TWAI 控制器内部），负责存储通过接收滤波器的接收报文。接收 FIFO 中存储的报文大小可以不同（3 ~ 13 byte 范围之间）。当接收 FIFO 为满时（或剩余的空间不足以完全存储下一个接收报文），将触发溢出中断，后续的接收报文将丢失，直到接收 FIFO 中清除出足够的存储空间。接收 FIFO 中的第一条报文将被映射到 13-byte 的接收缓冲器中，直到该报文被清除（通过释放接收缓冲器指令）。清除后，接收缓冲器将继续映射接收 FIFO 中的下一条报文，接收 FIFO 中上一条已清除报文的空间将被释放。

21.5 功能描述

21.5.1 模式

ESP32 TWAI 控制器有两种工作模式：复位模式和操作模式。将 TWAI_RESET_MODE 位置 1，进入复位模式；置 0，进入操作模式。

21.5.1.1 复位模式

要修改 TWAI 控制器的各种配置寄存器，需进入复位模式。进入复位模式时，TWAI 控制器彻底与 TWAI 总线断开连接。复位模式下，TWAI 控制器将无法发送任何报文（包括错误信号）。任何正在进行的报文传输将立即被终止。同样的，TWAI 控制器在该模式下也将无法接收任何报文。

21.5.1.2 操作模式

进入操作模式后，TWAI 控制器与总线相连，并且写保护各配置寄存器，以确保控制器的配置在运行期间保持一致。操作模式下，TWAI 控制器可以发送和接收报文（包括错误信号），但具体取决于 TWAI 控制器配置于哪种运行子模式。TWAI 控制器支持以下三种子模式：

- **正常模式：** TWAI 控制器可以发送和接收包含错误信号在内的报文（如，错误帧和过载帧）。
- **自测模式：**与正常模式相同，但在该模式下，TWAI 控制器发送报文时，即使在 CRC 域之后没有接收到应答信号，也不会产生应答错误。通常在 TWAI 控制器自测时使用该模式。
- **只听模式：** TWAI 控制器可以接收报文，但在 TWAI 总线上保持完全被动。因此，TWAI 控制器将无法发送任何报文、应答或错误信号。错误计数将保持冻结状态。该模式用于 TWAI 总线监控。

请注意，退出复位模式后（如，进入操作模式时），TWAI 控制器需等待 11 个连续隐性位出现，才能完全连接上 TWAI 总线（即，可以发送或接收报文）。

21.5.2 位时序

TWAI 控制器的工作位速率必须在控制器处于复位模式时进行配置。在寄存器 TWAI_BUS_TIMING_0_REG 和 TWAI_BUS_TIMING_1_REG 中配置位速率，这两个寄存器包含以下域：

下表 21-6 所示为 TWAI_BUS_TIMING_0_REG 包含的位域。

表 21-6. TWAI_CLOCK_DIVIDER_REG 的 bit 信息; TWAI 地址 0x18

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	SJW.1	SJW.0	BRP.5	BRP.4	BRP.3	BRP.2	BRP.1	BRP.0

说明：

- 预分频值 (BRP)：TWAI 时间定额时钟由 APB 时钟分频得到，APB 时钟通常为 80 MHz。可通过以下公式计算分频数值，其中 t_{Tq} 为时间定额的时钟周期，t_{CLK} 为 APB 时钟周期：
  $t_{Tq} = 2 \times t_{CLK} \times (2^5 \times BRP.5 + 2^4 \times BRP.4 + 2^3 \times BRP.3 + 2^2 \times BRP.2 + 2^1 \times BRP.1 + 2^0 \times BRP.0 + 1)$
- 同步跳宽 (SJW)：SJW 数值在 SJW.0 和 SJW.1 中配置，计算公式为：SJW = (2 x SJW.1 + SJW.0 + 1)。

下表 21-7 所示为 TWAI_BUS_TIMING_1_REG 包含的位域。

表 21-7. TWAI_BUS_TIMING_1_REG 的 bit 信息; TWAI 地址 0x1c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

说明：

- PBS1：根据以下公式计算缓冲时期段 1 中的时间定额数量：(8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1)。

- PBS2: 根据以下公式计算缓冲时期段 2 中的时间定额数量：(4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1)。
- SAM: 该值置 1 启动三点采样。用于低/中速总线，有利于过滤总线上的尖峰信号。

21.5.3 中断管理

ESP32 TWAI 控制器提供了七种中断，每种中断由寄存器 TWAI_INT_RAW_REG 中的一个位表示。要触发某个特定的中断，须设置 TWAI_INT ENA_REG 中相应的使能位。

TWAI 控制器提供了以下七种中断：

- 接收中断
- 发送中断
- 错误报警中断
- 数据溢出中断
- 被动错误中断
- 仲裁丢失中断
- 总线错误中断

只要在 TWAI_INT_RAW_REG 一个或多个中断位为 1，TWAI 控制器中的中断信号即为有效，当 TWAI_INT_RAW_REG 中的所有位都被清除时，TWAI 控制器中的中断信号则失效。寄存器 TWAI_INT_RAW_REG 被读取后，其中的大多数中断位将自动清除。但是，接收中断不包括在内，直到通过 TWAI_RELEASE_BUF 指令位清除所有接收报文后，接收中断位才能被清除。

21.5.3.1 接收中断 (RXI)

当 TWAI 接收 FIFO 中有待读取报文时（TWAI_RX_MESSAGE_CNT_REG > 0），都会触发 RXI。TWAI_RX_MESSAGE_CNT_REG 中记录的报文数量包括接收 FIFO 中的有效报文和溢出报文。直到通过 TWAI_RELEASE_BUF 指令位清除所有挂起接收报文后，RXI 才会失效。

21.5.3.2 发送中断 (TXI)

每当发送缓冲器空闲，将其他报文加载到发送缓冲器中等待发送时，都会触发 TXI。以下情况下，发送缓冲器将变为空闲，同时 TXI 将失效：

- 报文发送已成功完成（如，应答未发现错误）。任何发送失败将自动重发。
- 单次发送已完成（TWAI_TX_COMPLETE 位指示发送成功与否）。
- 使用 TWAI_ABORT_TX 指令位终止报文发送。

21.5.3.3 错误报警中断 (EWI)

每当寄存器 TWAI_STATUS_REG 中 TWAI_ERR_ST 或 TWAI_BUS_OFF_ST 的位值改变时（如，从 0 变为 1 或反之），都会触发 EWI。根据 EWI 触发时 TWAI_ERR_ST 或 TWAI_BUS_OFF_ST 的值分成以下几种情况：

- 如果 TWAI_ERR_ST = 0 或 TWAI_BUS_OFF_ST = 0：

  - 如果 TWAI 控制器处于主动错误状态，则表示 TEC 和 REC 的值都返回到了 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值之下。
  - 如果 TWAI 控制器此前正处于总线恢复状态，则表示此时总线恢复已成功完成。

- 如果 TWAI_ERR_ST = 1 或 TWAI_BUS_OFF_ST = 0：表示 TEC 或 REC 数值已超过 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值。
- 如果 TWAI_ERR_ST = 1 或 TWAI_BUS_OFF_ST = 1：表示 TWAI 控制器已进入 BUS_OFF 状态（因 TEC >= 256）。
- 如果 TWAI_ERR_ST = 0 或 TWAI_BUS_OFF_ST = 1：表示 BUS_OFF 恢复期间，TWAI 控制器的 TEC 数值已低于 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值。

21.5.3.4 数据溢出中断 (DOI)

软件读取接收缓冲器中的报文时，若该报文数据为溢出无效报文，将触发 DOI。

21.5.3.5 被动错误中断 (TXI)

每当 TWAI 控制器从主动错误变为被动错误，或反之时，都会触发 EPI。

21.5.3.6 仲裁丢失中断 (ALI)

每当 TWAI 控制器尝试发送报文且丢失仲裁时，都会触发 ALI。TWAI 控制器丢失仲裁的 bit 位置将自动记录在仲裁丢失捕捉寄存器 (TWAI_ARB LOST CAP_REG) 中。仲裁丢失捕捉寄存器被清除（通过 CPU 读取该寄存器）之前，将不会再记录新发生的仲裁失败时的 bit 位置。

21.5.3.7 总线错误中断 (BEI)

每当 TWAI 控制器在 TWAI 总线上检测到错误时，都会触发 BEI。发生总线错误时，总线错误的类型和发生错误时的 bit 位置都将自动记录在错误捕捉寄存器 (TWAI_ERR_CODE_CAP_REG) 中。错误捕捉寄存器被清除（通过 CPU 的读取）之前，将不会再记录新的总线错误信息。

21.5.4 发送缓冲器与接收缓冲器

21.5.4.1 缓冲器概述

表 21-8. SFF 与 EFF 的缓冲器布局

标准格式 (SFF)		扩展格式 (EFF)	
TWAI 地址	内容	TWAI 地址	内容
0x40	TX/RX 帧信息	0x40	TX/RX 帧信息
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3

标准格式 (SFF)		扩展格式 (EFF)	
TWAI 地址	内容	TWAI 地址	内容
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	保留	0x6c	TX/RX data byte 7
0x70	保留	0x70	TX/RX data byte 8

表 21-8 所示为发送缓冲器和接收缓冲器的寄存器布局。发送和接收缓冲寄存器的访问地址范围相同，且只有当 TWAI 控制器处于操作模式时才可访问。CPU 的写入操作将访问发送缓冲寄存器，CPU 的读取操作将访问接收缓冲寄存器。

发送缓冲寄存器用于配置 TWAI 的待发送报文。CPU 会在发送缓冲寄存器进行写入操作，指定报文的帧类型、帧格式、帧 ID 和帧数据（有效载荷）。一旦发送缓冲器配置完成后，CPU 会将 TWAI_CMD_REG 中的 TWAI_TX_REQ 位置 1，以开始报文发送。

- 若是自发自收请求，变更为将 TWAI_SELF_RX_REQ 置 1。
- 若是单次发送，需要同时将 TWAI_TX_REQ 和 TWAI_ABORT_TX 置 1。

接收缓冲寄存器映射到接收 FIFO 中的第一条报文。CPU 会在接收缓冲寄存器中进行读取操作，获取第一条报文的帧类型、帧格式、帧 ID 和帧数据（有效载荷）。读取完接收缓冲寄存器中的报文后，CPU 通过将 TWAI_CMD_REG 中的 TWAI_RELEASE_BUF 位置 1 来来清除接收缓冲寄存器，若接收 FIFO 中仍有待处理的报文，按照接收报文的先后次序将最早接收到的报文映射到接收缓冲寄存器。

21.5.4.2 帧信息

帧信息的长度为 1-byte，主要用于明确报文的帧类型、帧格式以及数据长度。下表 21-9 所示为帧信息域。

表 21-9. TX/RX 帧信息 (SFF/EFF)；TWAI 地址 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	FF	RTR	X	X	XDLC.3	DLC.2	DLC.1	DLC.0

说明：

- FF: 主要明确某报文属于 EFF 还是 SFF。当 FF 位为 1 时，该报文为 EFF，当 FF 位为 0 时，该报文为 SFF。
- RTR: 主要明确某报文是数据帧还是远程帧。当 RTR 位为 1 时，该报文为远程帧，当 RTR 位为 0 时，该报文为数据帧。
- DLC: 主要明确数据帧中的数据字节数量，或从远程帧中请求的数据字节数量。TWAI 数据帧的最大载荷为 8 个数据字节，因此 DLC 的数值范围应是 0 ~ 8。
- X: 无关 bit，可以是任意值。

21.5.4.3 帧标识符

若报文为 SFF，则对应的帧标识符域为 2-bytes (11-bits)；若报文为 EFF，则对应的帧标识符域为 4-bytes (29-bits)。

下表 Table 21-10-21-11 所示为 SFF (11-bits) 报文的帧标识符域。

表 21-10. TX/RX 标识符 1 (SFF); TWAI 地址 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 21-11. TX/RX 标识符 2 (SFF); TWAI 地址 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.20	ID.19	ID.18	X[1]	X[2]	X[2]	X[2]	X[2]

下表 21-12-21-15 所示为 EFF (29-bits) 报文的帧标识符域。

表 21-12. TX/RX 标识符 1 (EFF); TWAI 地址 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 21-13. TX/RX 标识符 2 (EFF); TWAI 地址 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

表 21-14. TX/RX 标识符 3 (EFF); TWAI 地址 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

表 21-15. TX/RX 标识符 4 (EFF); TWAI 地址 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

21.5.4.4 帧数据

帧数据域包含发送或接收的数据帧，范围为 0 ~ 8 bytes。其中的有效字节数应与 DLC 相同。但是，如果 DLC 数值大于 8，则帧数据域的有效字节数仍为 8。远程帧中不包含数据载荷，因此不存在帧数据域。

比如，当发送 5 个数据字节的数据帧时，CPU 应在 DLC 域中写入数值 5，并将数据写入数据域 1 ~ 5 字节对应的寄存器。同样，当接收 DLC 为 5 的数据帧时，只有 1 ~ 5 数据字节中包含 CPU 可以读取的有效载荷数据。

21.5.5 接收 FIFO 和数据溢出

接收 FIFO 是一个 64-bytes 的内部缓冲器，用于以先进先出的原则存储接收到的报文。一条接收报文可在接收 FIFO 中占 3 ~ 13 bytes 空间，且其中字节序与接收缓冲器的寄存器地址顺序相同。接收缓冲寄存器将被映射到

接收 FIFO 中第一条报文。

当 TWAI 控制器接收到一条报文时，TWAI_RX_MESSAGE_COUNTER 的值将增加 1，最大值为 64。如果接收 FIFO 中有足够的剩余空间，报文内容将被写入到接收 FIFO 中。读取接收缓冲器中的消息后，通过将 TWAI_RELEASE_BUF 的位置 1，释放接收 FIFO 第一条报文所占的空间，TWAI_RX_MESSAGE_COUNTER 的值也将减小 1。然后，接收缓冲器将映射接收 FIFO 中的下一条报文。

当 TWAI 控制器接收到一条报文，但接收 FIFO 没有足够空间完整地存储这条接收报文时（不论是因为报文内容大小大于接收 FIFO 中的空闲空间，还是因为接收 FIFO 已满），接收 FIFO 将在内部将溢出报文标记为无效。后续接收到的溢出报文仍将增加 TWAI_RX_MESSAGE_COUNTER 的值到最大值 64。

为了清除接收 FIFO 中的溢出报文，应重复调用 TWAI_RELEASE_BUF，直到 TWAI_RX_MESSAGE_COUNTER 为 0。这样可以读取接收 FIFO 中的所有有效报文，并清除所有溢出报文。

21.5.6 接收滤波器

接收滤波器允许 TWAI 控制器根据报文 ID 过滤接收报文（有时可以过滤报文的第一个数据字节和帧类型）。只有通过过滤的报文才能存储到接收 FIFO 中。接收滤波器的使用可以一定程度地减轻 TWAI 控制器的运行负荷（如，可减少使用接收 FIFO 和发生接收中断的次数），因为 TWAI 控制器将只需要操作一小部分过滤后的报文。

只有当 TWAI 控制器处于复位模式时，才可以访问接收滤波器的配置寄存器，因为这些配置寄存器和发送/接收缓冲寄存器的地址空间相同。

接收滤波器的配置寄存器由 32-bit 的 Code 值和 32-bit 的 Mask 值组成。Code 值将指定一种位排列模式，每条过滤报文中的位都必须匹配该模式，才能使该报文通过过滤。Mask 值可屏蔽 Code 值中的某些位（将屏蔽位设置为“不相关”的位）。如图 21-7 所示，为了使报文通过过滤，每条过滤报文的 ID 都必须匹配 Code 值所设模式或者被 Mask 值屏蔽。

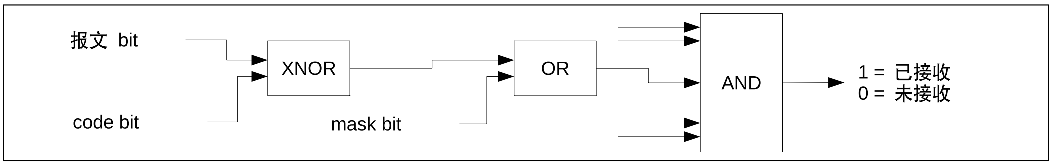


图 21-7. 接收滤波器

TWAI 控制器的接收滤波器允许 32-bit 的 Code 值和 Mask 值定义单个滤波器（单滤波模式），或两个滤波器（双滤波模式）。接收滤波器如何解析 32-bit 的 code 值和 mask 值，取决于滤波模式以及接收报文的格式（如，SFF 还是 EFF）。

21.5.6.1 单滤波模式

将 TWAI_RX_FILTER_MODE 的位置 1，可启动单滤波模式。此后，32-bit code/mask 的值将定义单个滤波器。

单个滤波器可过滤数据帧和远程帧中的以下位：

- SFF
  - 11-bit ID 整体
  - RTR bit

  - 数据字节 1 和数据字节 2
- EFF
  - 29-bit ID 整体
  - RTR bit

下图 21-8 所示为单滤波模式下如何解析 32-bit code/mask 的值。

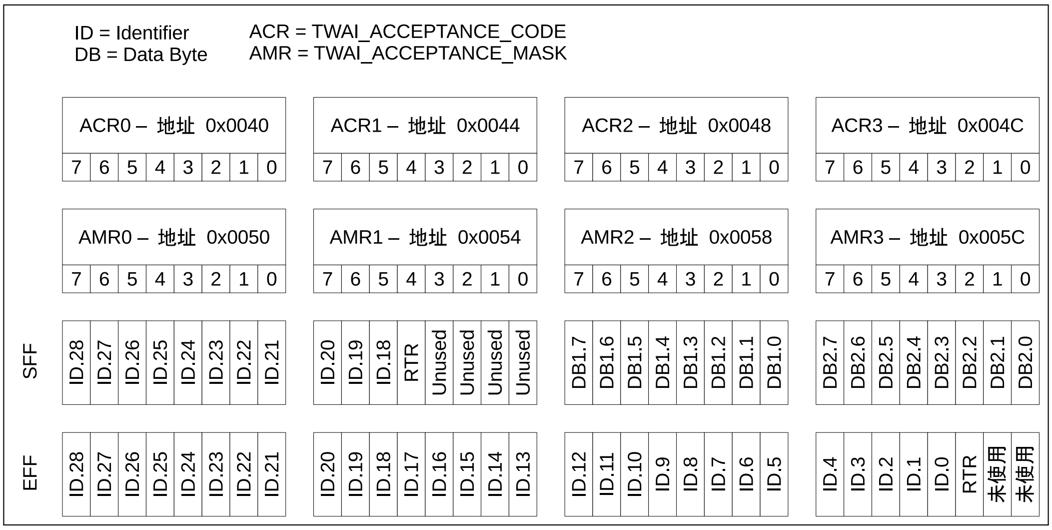


图 21-8. 单滤波模式

21.5.6.2 双滤波模式

将 TWAI_RX_FILTER_MODE 的位置 0，可启动双滤波模式。此后，32-bit code/mask 的值将定义两个滤波器之一，即滤波器 1 或滤波器 2。双滤波模式下，如果报文通过这两个滤波器中的至少一个，则表示该报文已成功通过过滤。

这两个滤波器可以过滤数据帧和远程帧中的以下位：

- SFF
  - 11-bit ID 整体
  - RTR bit
  - 数据字节 1 (仅适用于滤波器 1)
- EFF
  - 29-bit ID 的前 16-bit

下图 21-9 所示为双滤波模式下如何解析 32-bit code/mask 的值。

21.5.7 错误管理

TWAI 协议要求每个 TWAI 节点中都包含发送错误计数 (TEC) 和接收错误计数 (REC)。这两个错误计数的数值决定了 TWAI 控制器当前的错误状态（如，主动错误、被动错误、离线）。TWAI 控制器将 TEC 和 REC 的数值分别存储在 TWAI_TX_ERR_CNT_REG 和 TWAI_RX_ERR_CNT_REG 中，CPU 可随时进行读取。除了错误状态之外，

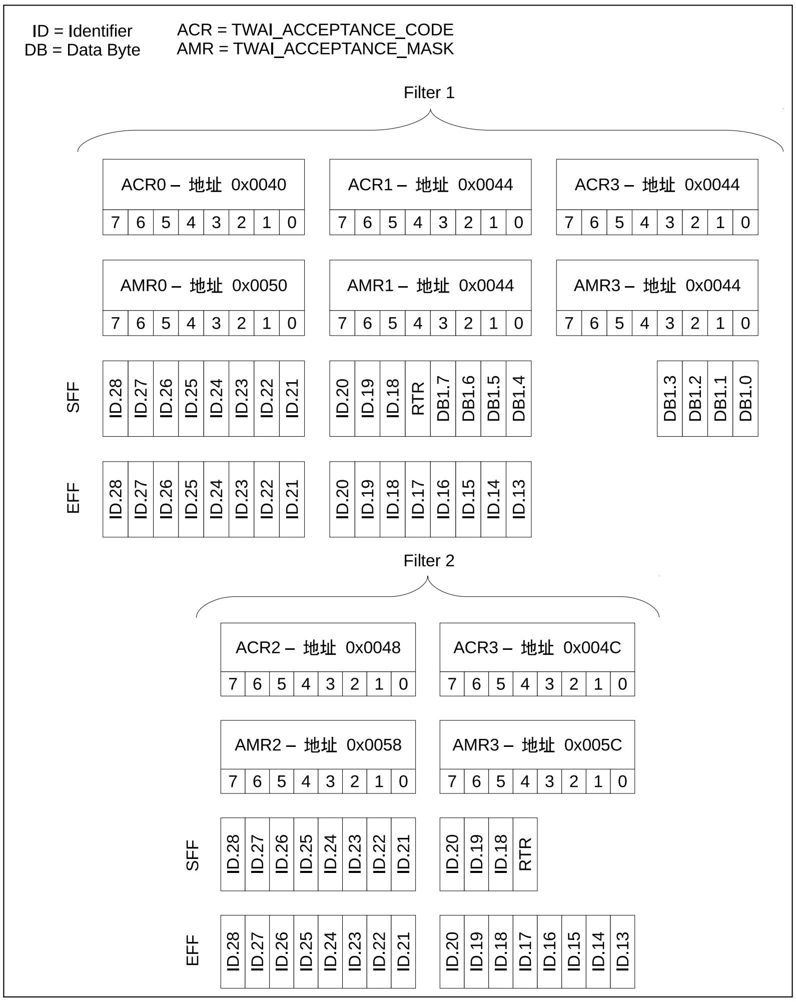


图 21-9. 双滤波模式

TWAI 控制器还提供错误报警限制 (EWL) 的功能，这个功能可在 TWAI 控制器进入被动错误状态之前，提醒用户当前发生的严重总线错误。

TWAI 控制器的当前错误状态通过以下各数值和状态位体现，即：TEC、REC、TWAI_ERR_ST 和 TWAI_BUS_OFF_ST。这些数值和状态位的变化也将触发中断，从而提醒用户当前的错误状态变化（可参见第 21.5.3 章）。下图 21-10 所示为错误状态、上述数值和状态位以及错误状态相关中断之间的关系。

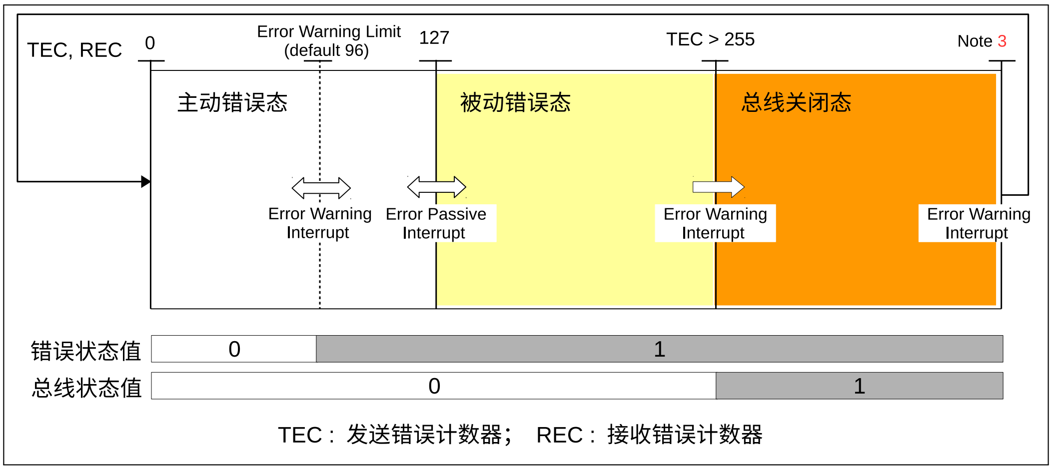


图 21-10. 错误状态变化

21.5.7.1 错误报警限制

错误报警限制 (EWL) 为 TEC 和 REC 的可配置阈值，若错误计数数值超过该阈值，将触发 EWI 中断。EWL 将作为一个报警功能提示当前发生的严重 TWAI 总线错误，且在 TWAI 控制器进入被动错误状态之前被触发。EWL 数值应在寄存器 TWAI_ERR_WARNING_LIMIT_REG 中进行配置，配置同时 TWAI 控制器必须处于复位模式下。TWAI_ERR_WARNING_LIMIT_REG 默认数值为 96。

当 TEC 和/或 REC 数值大于等于 EWL 数值时，TWAI_ERR_ST 位将立即被置 1。同理，当 TEC 和 REC 数值都小于 EWL 数值时，TWAI_ERR_ST 位将立即复位为 0。只要 TWAI_ERR_ST（或 TWAI_BUS_OFF_ST）位值发生变化，便会触发错误报警中断。

21.5.7.2 被动错误

当 TEC 或 REC 数值大于 127 时，TWAI 控制器处于被动错误状态。同理，当 TEC 和 REC 数值都小于等于 127 时，TWAI 控制器进入主动错误状态。每当 TWAI 控制器从主动错误状态变为被动错误状态，或反之时，都将触发被动错误中断。

21.5.7.3 离线状态与离线恢复

当 TEC 数值大于 255 时，TWAI 控制器将进入离线状态。进入离线状态后，TWAI 控制器将自动进行以下动作：

- REC 数值置为 0
- TEC 数值置为 127
- TWAI_BUS_OFF_ST 位置 1
- 进入复位模式

每当 TWAI_BUS_OFF_ST 位（或 TWAI_ERR_ST 位）数值发生变化时，都将触发错误报警中断。

为了返回主动错误状态，TWAI 控制器必须进行离线恢复。要启动离线恢复，首先需要退出复位模式，进入操作模式。然后要求 TWAI 控制器在总线上检测到 128 次 11 个连续隐性位。

每一次 TWAI 控制器检测到 11 个连续隐性位时，TEC 数值都将减小，以追踪离线恢复进程。当离线恢复完成后（TEC 数值从 127 减小到 0），TWAI_BUS_OFF_ST 位将自动复位为 0，从而触发错误报警中断。

21.5.8 错误捕捉

错误捕捉 (ECC) 功能允许 TWAI 控制器以错误代码的形式记录 TWAI 总线错误的错误类型和 bit 位置。当检测到一个 TWAI 总线错误时，总线错误中断将被触发，相应的错误代码将记录在 TWAI_ERR_CODE_CAP_REG 中。寄存器 TWAI_ERR_CODE_CAP_REG 中存储的当前错误代码被读取之前，后续的总线错误中断触发时，将不会再记录错误代码。

下表 21-16 所示为寄存器 TWAI_ERR_CODE_CAP_REG 中的域：

表 21-16. TWAI_ERR_CODE_CAP_REG 的 bit 信息; TWAI 地址 0x30

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ERRC.1	ERRC.0	DIR	SEG.4	SEG.3	SEG.2	SEG.1	SEG.0

说明：

- 错误代码 (ERRC)：表示总线错误的类型。00 代表位错误，01 代表格式错误，10 代表填充错误，11 代表其他错误类型。
- 传输方向 (DIR)：表示总线错误发生时，TWAI 控制器处于发送器状态还是接收器状态。0 代表发送器，1 代表接收器。
- 错误段 (SEG)：表示总线错误发生在 TWAI 报文的哪个段。

下表 21-17 所示为 SEG.0 ~ SEG.4 的位信息。

表 21-17. SEG.4 - SEG.0 的位信息

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	描述
0	0	0	1	1	帧起始
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	保留位 1
0	1	0	0	1	保留位 0
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据域
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 分界符
1	1	0	0	1	确认槽
1	1	0	1	1	确认分界符
1	1	0	1	0	帧结束
1	0	0	1	0	间歇域
1	0	0	0	1	主动错误标志
1	0	1	1	0	被动错误标志
1	0	0	1	1	兼容显性位
1	0	1	1	1	错误分界符
1	1	1	0	0	过载标志

说明：

- Bit SRTR: 标准格式 RTR bit。
- Bit IDE: 标识符扩展位。0 表示标准格式。

21.5.9 仲裁丢失捕捉

仲裁丢失捕捉 (ALC) 功能允许 TWAI 控制器记录丢失仲裁的 bit 位置。当 TWAI 控制器丢失仲裁时，bit 位置将被记录在寄存器 TWAI_ARB LOST CAP_REG 中，同时触发仲裁丢失中断。

后续的仲裁丢失中断触发时，bit 位置将不会被记录在 TWAI_ARB LOST CAP_REG 中，直到

TWAI_ERR_CODE_CAP_REG 中的当前仲裁丢失捕捉被读取。

下表 21-18 所示为 TWAI_ERR_CODE_CAP_REG 中的位域；下图 21-11 所示为一条 TWAI 报文的 bit 位置。

表 21-18. TWAI_ARB LOST CAP_REG 中的位信息; TWAI 地址 0x2c

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	BITNO.4	BITNO.3	BITNO.2	BITNO.1	BITNO.0

说明：

- 位号 (BITNO)：表示丢失仲裁的 TWAI 报文的第 n 个位。

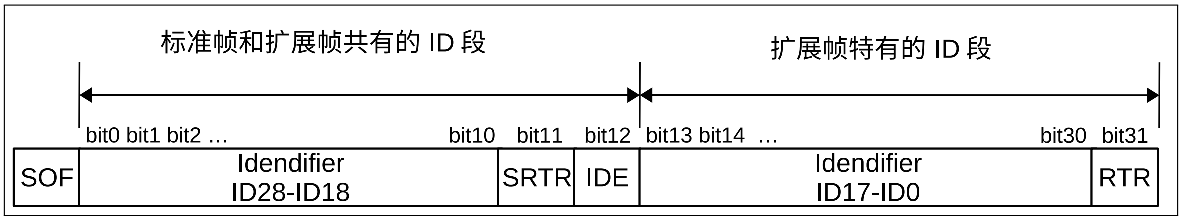


图 21-11. 丢失仲裁的 bit 位置

21.6 寄存器列表

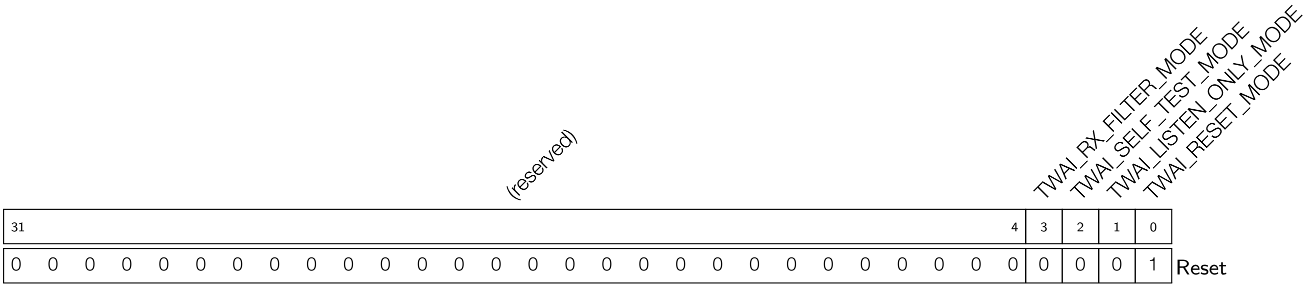
名称	描述	地址	访问
配置寄存器			
TWAI_MODE_REG	模式寄存器	0x3FF6B000	读/写
TWAI_BUS_TIMING_0_REG	时序配置寄存器 0	0x3FF6B018	只读 I 读/写
TWAI_BUS_TIMING_1_REG	时序配置寄存器 1	0x3FF6B01C	只读 I 读/写
TWAI_ERR_WARNING_LIMIT_REG	错误寄存器	0x3FF6B034	只读 I 读/写
TWAI_DATA_0_REG	数据寄存器 0	0x3FF6B040	只写 I 读/写
TWAI_DATA_1_REG	数据寄存器 1	0x3FF6B044	只写 I 读/写
TWAI_DATA_2_REG	数据寄存器 2	0x3FF6B048	只写 I 读/写
TWAI_DATA_3_REG	数据寄存器 3	0x3FF6B04C	只写 I 读/写
TWAI_DATA_4_REG	数据寄存器 4	0x3FF6B050	只写 I 读/写
TWAI_DATA_5_REG	数据寄存器 5	0x3FF6B054	只写 I 读/写
TWAI_DATA_6_REG	数据寄存器 6	0x3FF6B058	只写 I 读/写
TWAI_DATA_7_REG	数据寄存器 7	0x3FF6B05C	只写 I 读/写
TWAI_DATA_8_REG	数据寄存器 8	0x3FF6B060	只写 I 只读
TWAI_DATA_9_REG	数据寄存器 9	0x3FF6B064	只写 I 只读
TWAI_DATA_10_REG	数据寄存器 10	0x3FF6B068	只写 I 只读
TWAI_DATA_11_REG	数据寄存器 11	0x3FF6B06C	只写 I 只读
TWAI_DATA_12_REG	数据寄存器 12	0x3FF6B070	只写 I 只读
TWAI_CLOCK_DIVIDER_REG	时钟分频寄存器	0x3FF6B07C	不定
控制寄存器			
TWAI_CMD_REG	指令寄存器	0x3FF6B004	只写
状态寄存器			
TWAI_STATUS_REG	状态寄存器	0x3FF6B008	只读

名称	描述	地址	访问
TWAI_ARB_LOST_CAP_REG	仲裁丢失寄存器	0x3FF6B02C	只读
TWAI_ERR_CODE_CAP_REG	错误捕获寄存器	0x3FF6B030	只读
TWAI_RX_ERR_CNT_REG	接收错误寄存器	0x3FF6B038	只读 | 读/写
TWAI_TX_ERR_CNT_REG	发送错误寄存器	0x3FF6B03C	只读 | 读/写
TWAI_RX_MESSAGE_CNT_REG	接收数据寄存器	0x3FF6B074	只读
中断寄存器			
TWAI_INT_RAW_REG	中断寄存器	0x3FF6B00C	只读
TWAI_INT_ENA_REG	中断使能寄存器	0x3FF6B010	读/写

21.7 寄存器

本小节括号中的地址均为相对于 TWAI 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 21.6 寄存器列表。

Register 21.1. TWAI_MODE_REG (0x0000)



TWAI_RESET_MODE 配置 TWAI 控制器操作模式。1：复位模式；0：操作模式（读/写）

TWAI_LISTEN_ONLY_MODE 置 1 进入只听模式，处于该模式下的节点只接收总线上数据，不产生应答信号，也不更新接收错误计数。(读/写)

TWAI_SELF_TEST_MODE 置 1 启动自测模式，此模式下发送节点发送完数据后无需应答信号反馈。该模式常配合自接自收指令测试某个节点。(读/写)

TWAI_RX_FILTER_MODE 配置滤波模式。0: 双滤波模式；1: 单滤波模式（读/写）

Register 21.2. TWAI_BUS_TIMING_0_REG (0x0018)

(reserved)	TWAI_SYNC_JUMP_WIDTH	TWAI_BAUD_PRESC	
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	0x0	0x0	Reset

TWAI_BAUD_PRESC 预分频值，决定分频比例。（只读 | 读/写）

TWAI_SYNC_JUMP_WIDTH 同步跳宽 (SJW)，范围为 1 ~ 4 个时间定额。（只读 | 读/写）

Register 21.3. TWAI_BUS_TIMING_1_REG (0x001C)

(reserved)	TWAI_TIME_SAMP	TWAI_TIME_SEG2	TWAI_TIME_SEG1	
31 8	7	6 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	0x0	0x0	Reset

TWAI_TIME_SEG1 缓冲时期段 1 的宽度。（只读 | 读/写）

TWAI_TIME_SEG2 缓冲时期段 2 的宽度。（只读 | 读/写）

TWAI_TIME_SAMP 采样点数目。0：采样 1 次；1：采样三次（只读 | 读/写）

Register 21.4. TWAI_ERR_WARNING_LIMIT_REG (0x0034)

(reserved)	TWAI_ERR_WARNING_LIMIT	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x60	Reset

TWAI_ERR_WARNING_LIMIT 错误报警阈值，当任一错误计数数值超过该阈值或者所有错误计数数值都小于该阈值时，将触发错误报警中断（使能信号有效情况下）。（只读 | 读/写）

Register 21.5. TWAI_DATA_0_REG (0x0040)

(reserved)	TWAI_TX_BYTE_0 | TWAI_ACCEPTANCE_CODE_0	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_0 操作模式下，存储着待发送数据的第 0 个字节内容。（只写）

TWAI_ACCEPTANCE_CODE_0 复位模式下，存储着滤波编码的第 0 个字节。（读/写）

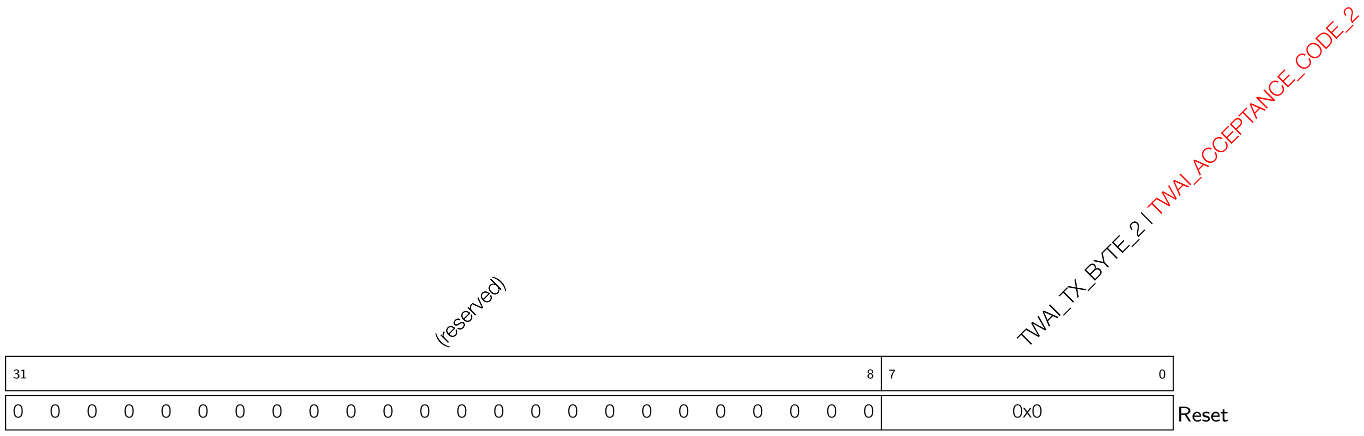
Register 21.6. TWAI_DATA_1_REG (0x0044)

(reserved)	TWAI_TX_BYTE_1 | TWAI_ACCEPTANCE_CODE_1	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_1 操作模式下，存储着待发送数据的第 1 个字节内容。（只写）

TWAI_ACCEPTANCE_CODE_1 复位模式下，存储着滤波编码的第 1 个字节。（读/写）

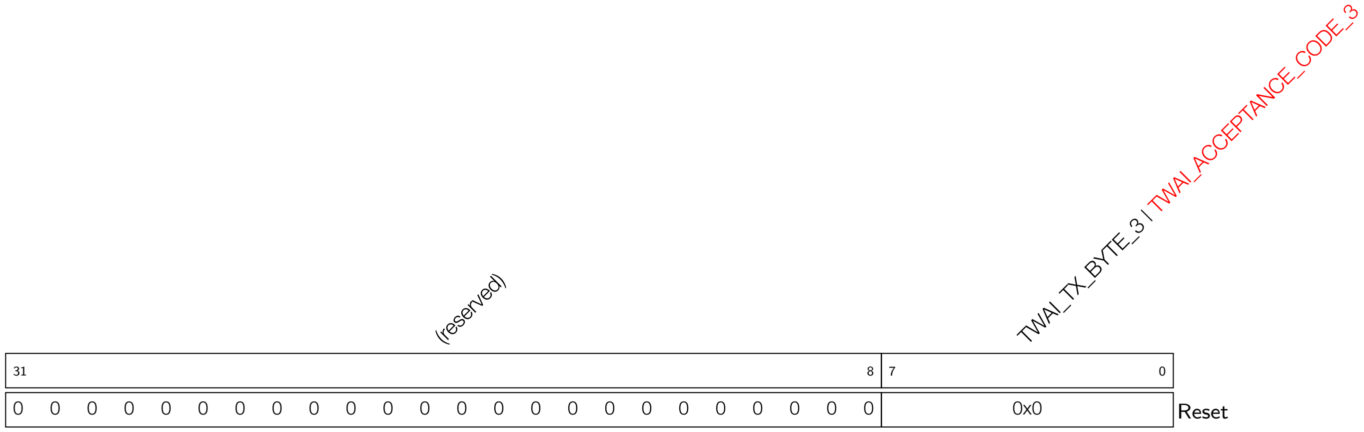
Register 21.7. TWAI_DATA_2_REG (0x0048)



TWAI_TX_BYTE_2 操作模式下，存储着待发送数据的第 2 个字节内容。(只写)

TWAI_ACCEPTANCE_CODE_2 复位模式下，存储着滤波编码的第 2 个字节。(读/写)

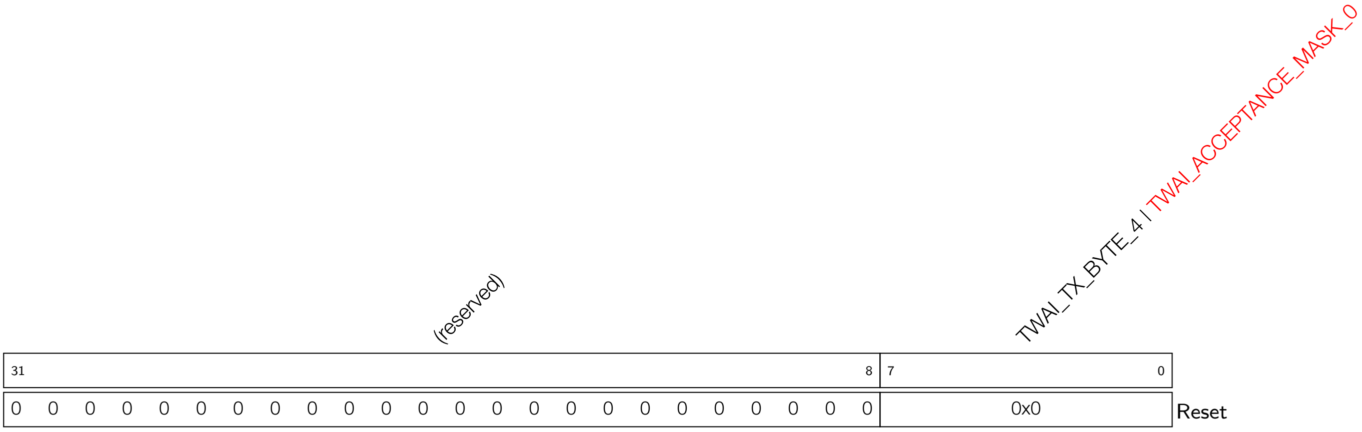
Register 21.8. TWAI_DATA_3_REG (0x004C)



TWAI_TX_BYTE_3 操作模式下，存储着待发送数据的第 3 个字节内容。(只写)

TWAI_ACCEPTANCE_CODE_3 复位模式下，存储着滤波编码的第 3 个字节。(读/写)

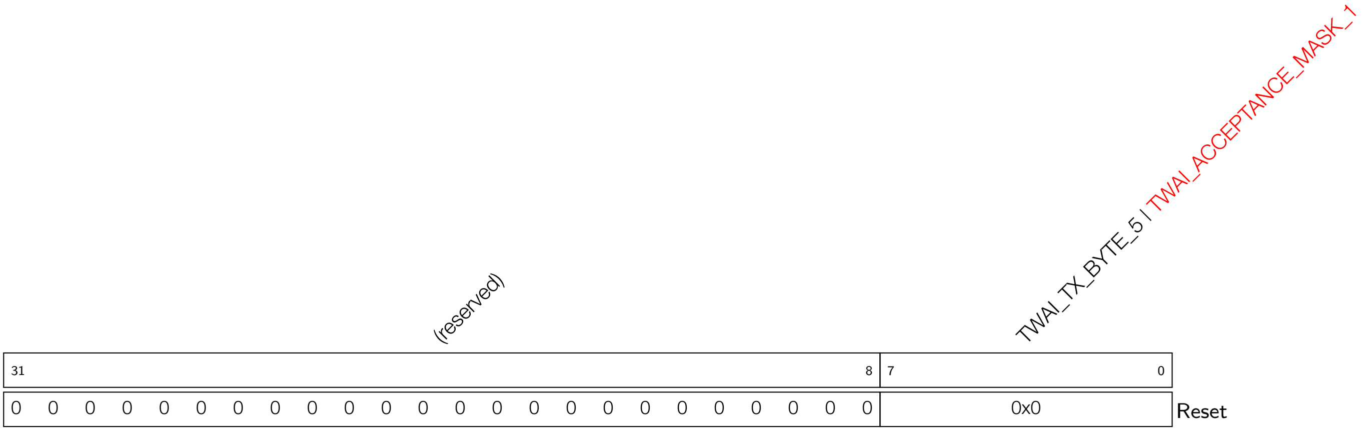
Register 21.9. TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 操作模式下，存储着待发送数据的第 4 个字节内容。（只写）

TWAI_ACCEPTANCE_MASK_0 复位模式下，存储着滤波编码的第 0 个字节。（读/写）

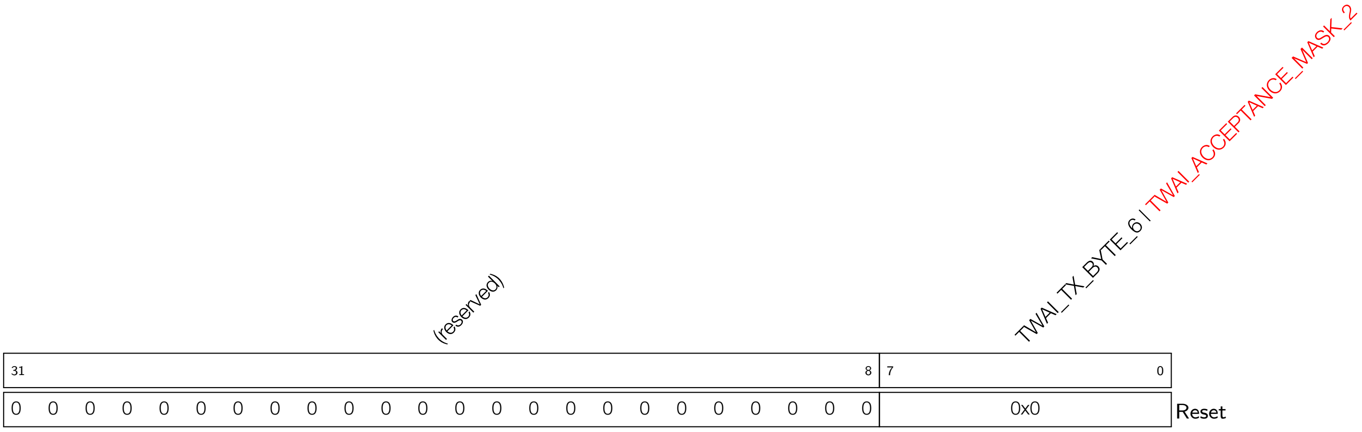
Register 21.10. TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 操作模式下，存储着待发送数据的第 5 个字节内容。（只写）

TWAI_ACCEPTANCE_MASK_1 复位模式下，存储着滤波编码的第 1 个字节。（读/写）

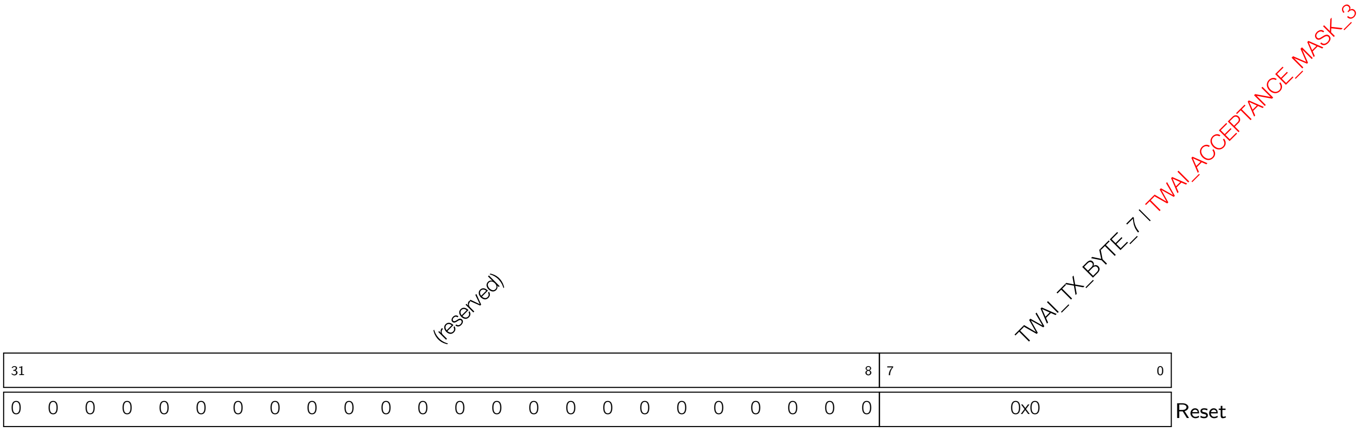
Register 21.11. TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 操作模式下，存储着待发送数据的第 6 个字节内容。（只写）

TWAI_ACCEPTANCE_MASK_2 复位模式下，存储着滤波编码的第 2 个字节。（读/写）

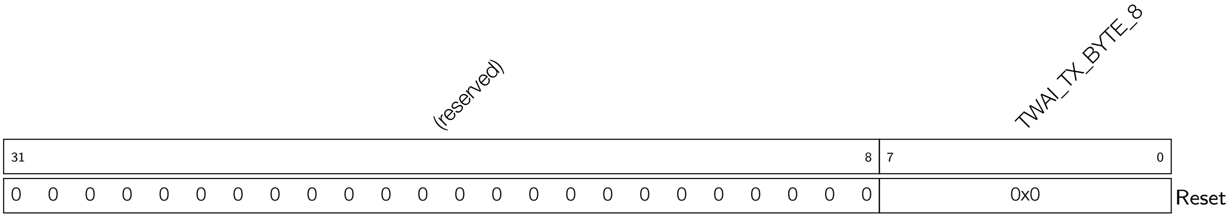
Register 21.12. TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 操作模式下，存储着待发送数据的第 7 个字节内容。（只写）

TWAI_ACCEPTANCE_MASK_3 复位模式下，存储着滤波编码的第 3 个字节。（读/写）

Register 21.13. TWAI_DATA_8_REG (0x0060)



TWAI_TX_BYTE_8 操作模式下，存储着待发送数据的第 8 个字节内容。（只写）

Register 21.14. TWAI_DATA_9_REG (0x0064)

(reserved)	TWAI_TX_BYTE_9	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_9 操作模式下，存储着待发送数据的第 9 个字节内容。（只写）

Register 21.15. TWAI_DATA_10_REG (0x0068)

(reserved)	TWAI_TX_BYTE_10	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_10 操作模式下，存储着待发送数据的第 10 个字节内容。（只写）

Register 21.16. TWAI_DATA_11_REG (0x006C)

(reserved)	TWAI_TX_BYTE_11	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_11 操作模式下，存储着待发送数据的第 11 个字节内容。（只写）

Register 21.17. TWAI_DATA_12_REG (0x0070)

(reserved)	TWAI_TX_BYTE_12	
31 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

TWAI_TX_BYTE_12 操作模式下，存储着待发送数据的第 12 个字节内容。（只写）

Register 21.18. TWAI_CLOCK_DIVIDER_REG (0x007C)

(reserved)	TWAI_EXT_MODE	(reserved)	TWAI_CLOCK_OFF	TWAI_CD	
31 8	7	6 4	3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	0	0	0x0	Reset

TWAI_CD 配置输出时钟 CLKOUT 的分频系数。(读/写)

TWAI_CLOCK_OFF 复位模式下可配。1: 关闭输出的 CLKOUT 时钟; 0: 打开 CLKOUT 时钟(只读 | 读/写)

TWAI_EXT_MODE 复位模式下可配,默认值为 1。0: 基础模式; 1: 扩展模式,兼容 CAN 2.0B(只读 | 读/写)

Register 21.19. TWAI_CMD_REG (0x0004)

(reserved)	TWAI_SELF_RX_REQ	TWAI_CLR_OVERRUN	TWAI_RELEASE_BUF	TWAI_ABORT_TX	TWAI_TX_REQ	
31 5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	Reset

TWAI_TX_REQ 置 1 驱动节点开始发送数据任务。(只写)

TWAI_ABORT_TX 置 1 取消当前还未开始的发送任务。(只写)

TWAI_RELEASE_BUF 置 1 释放接收缓冲器。(只写)

TWAI_CLR_OVERRUN 置 1 清除数据溢出状态。(只写)

TWAI_SELF_RX_REQ 自接自收命令。置 1 允许发送节点发送数据的同时接收总线上的数据。(只写)

Register 21.20. TWAI_STATUS_REG (0x0008)

	(reserved)	TWAI_BUS_OFF_ST	TWAI_ERR_ST	TWAI_TX_ST	TWAI_RX_ST	TWAI_TX_COMPLETE	TWAI_TX_BUF_ST	TWAI_OVERRUN_ST	TWAI_RX_BUF_ST
	31 8	7	6	5	4	3	2	1	0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	1	1	0	0

TWAI_RX_BUF_ST 若值为 1，表明接收缓冲器中数据不为空，至少有一个已经接收到的数据包。（只读）

TWAI_OVERRUN_ST 若值为 1，表明接收 FIFO 中存储的数据已满，产生了溢出。（只读）

TWAI_TX_BUF_ST 若值为 1，表明发送缓冲器为空，允许写入待发送数据。（只读）

TWAI_TX_COMPLETE 若值为 1，表明成功从总线上接收到一个数据包。（只读）

TWAI_RX_ST 若值为 1，表明节点正在从总线上接收数据。（只读）

TWAI_TX_ST 若值为 1，表明节点正在往总线上发送数据。（只读）

TWAI_ERR_ST 若值为 1，表明接收错误计数和发送错误计数中至少有一个数值大于等于寄存器 TWAI_ERR_WARNING_LIMIT_REG 中配置的数值（只读）

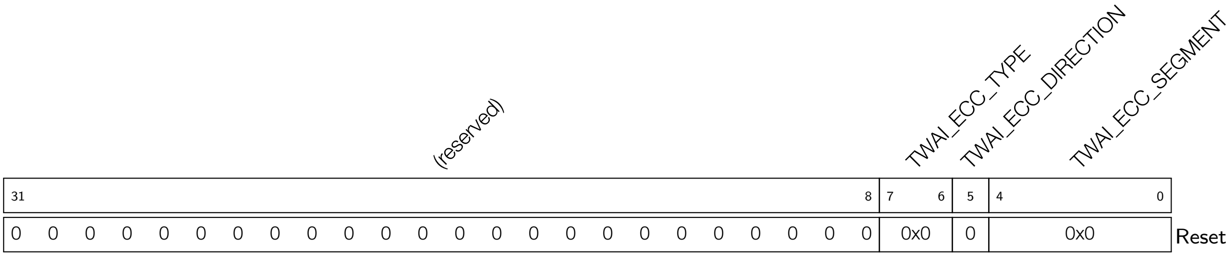
TWAI_BUS_OFF_ST 若值为 1，表明节点处于离线状态，不再响应总线上的数据传输。（只读）

Register 21.21. TWAI_ARB LOST CAP_REG (0x002C)

	(reserved)	TWAI_ARB_LOST_CAP
	31 5	4 0
Reset	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0

TWAI_ARB_LOST_CAP 记录着发送节点仲裁丢失的 bit 位置。（只读）

Register 21.22. TWAI_ERR_CODE_CAP_REG (0x0030)

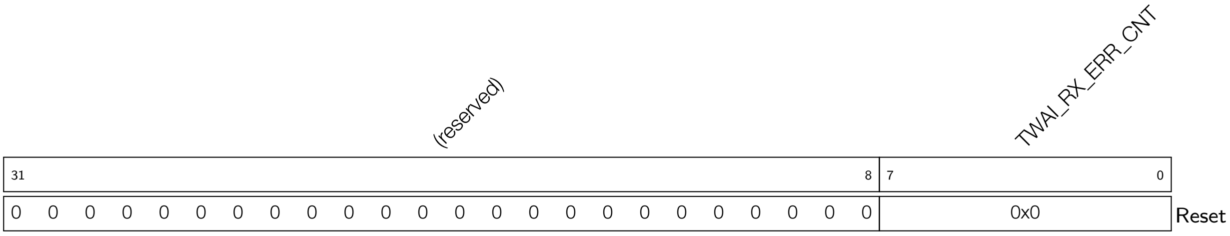


TWAI_ECC_SEGMENT 记录错误发生的位置，详见表 21-16。（只读）

TWAI_ECC_DIRECTION 记录错误时节点的数据传输方向。1：接收数据时发生错误；0：发送数据时发生错误（只读）

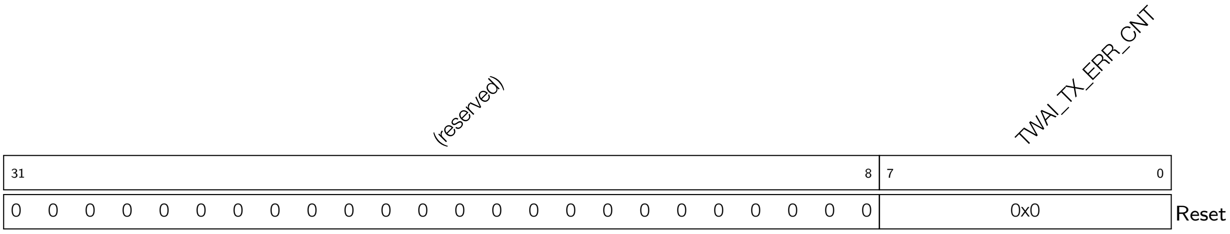
TWAI_ECC_TYPE 记录错误类别：00：位错误；01：格式错误；10：填充错误；11：其他错-（只读）

Register 21.23. TWAI_RX_ERR_CNT_REG (0x0038)



TWAI_RX_ERR_CNT 接收错误计数，数值变化发生在接收状态下。（只读 | 读/写）

Register 21.24. TWAI_TX_ERR_CNT_REG (0x003C)



TWAI_TX_ERR_CNT 发送错误计数，数值变化发生在发送状态下。（只读 | 读/写）

Register 21.25. TWAI_RX_MESSAGE_CNT_REG (0x0074)

(reserved)	TWAI_RX_MESSAGE_COUNTER	
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	0x0	Reset

TWAI_RX_MESSAGE_COUNTER 存储着接收 FIFO 中数据包的个数。（只读）

Register 21.26. TWAI_INT_RAW_REG (0x000C)

(reserved)	TWAI_BUS_ERR_INT_ST	TWAI_ARB_LOST_INT_ST	TWAI_ERR_PASSIVE_INT_ST	(reserved)	TWAI_OVERRUN_INT_ST	TWAI_ERR_WARN_INT_ST	TWAI_TX_INT_ST	TWAI_RX_INT_ST	
31 … 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

TWAI_RX_INT_ST 接收中断。若值为 1，表明接收 FIFO 不为空，有接收数据待处理。（只读）

TWAI_TX_INT_ST 发送中断。若值为 1，表明节点数据发送任务结束，可以执行新的数据发送任务。（只读）

TWAI_ERR_WARN_INT_ST 错误报警中断。若值为 1，表明状态寄存器中错误状态信号和离线信号发生变化（0 变为 1 或 1 变为 0）。（只读）

TWAI_OVERRUN_INT_ST 数据溢出中断。若值为 1，表明当前接收缓存中存储的报文为溢出失效报文。（只读）

TWAI_ERR_PASSIVE_INT_ST 被动错误中断。若值为 1，表明节点由于错误计数数值的变化，在主动错误状态与被动错误状态间发生了切换。（只读）

TWAI_ARB_LOST_INT_ST 仲裁丢失中断。若值为 1，表明发送节点丢失仲裁。（只读）

TWAI_BUS_ERR_INT_ST 错误中断。若值为 1，表明节点检测到总线上发生了错误。（只读）

Register 21.27. TWAI_INT ENA_REG (0x0010)

(reserved)	TWAI_BUS_ERR_INT_ENA	TWAI_ARB_LOST_INT_ENA	TWAI_ERR_PASSIVE_INT_ENA	(reserved)	TWAI_OVERRUN_INT_ENA	TWAI_ERR_WARN_INT_ENA	TWAI_TX_INT_ENA	TWAI_RX_INT_ENA	
31 8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	Reset

TWAI_RX_INT_ENA 置 1 使能接收中断。(读/写)

TWAI_TX_INT_ENA 置 1 使能发送中断。(读/写)

TWAI_ERR_WARN_INT_ENA 置 1 使能错误报警中断。(读/写)

TWAI_OVERRUN_INT_ENA 置 1 使能数据溢出中断。(读/写)

TWAI_ERR_PASSIVE_INT_ENA 置 1 使能被动错误中断。(读/写)

TWAI_ARB_LOST_INT_ENA 置 1 使能仲裁丢失中断。(读/写)

TWAI_BUS_ERR_INT_ENA 置 1 使能错误中断。(读/写)

22 AES 加速器 (AES)

22.1 概述

ESP32 内置 AES（高级加密标准）加速器，相比于只用软件进行 AES 运算，AES 加速器能够极大地提高运算速度。AES 加速器支持 FIPS PUB 197 标准，能够实现 AES-128，AES-192，AES-256 加密与解密运算。

22.2 主要特性

- 支持 AES-128 加解密运算
- 支持 AES-192 加解密运算
- 支持 AES-256 加解密运算
- 支持 4 种密钥字节序和 4 种文本字节序

22.3 功能描述

22.3.1 运算模式

AES 加速器支持 AES-128/192/256 加解密 6 种运算。配置寄存器 AES_MODE_REG 以实现不同运算。寄存器 AES_MODE_REG 的值与各种运算的对应关系如表 22-1 所示。

表 22-1. 运算模式

AES_MODE_REG[2:0]	运算
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

22.3.2 密钥、明文、密文

寄存器 AES_KEY_*n*_REG 存放密钥。每个寄存器位宽都是 32 位，共有 8 个寄存器。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_*m*_REG 存放明文或密文。每个寄存器位宽都是 32 位，共有 4 个寄存器。如果为 AES-128/192/265 加密运算，则运算开始之前用明文初始化寄存器 AES_TEXT_*m*_REG。运算完成之后，AES 加速器将把密文更新入寄存器 AES_TEXT_*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_*m*_REG。运算完成之后，AES 加速器将把明文更新入寄存器 AES_TEXT_*m*_REG。

22.3.3 字节序

密钥字节序

寄存器 AES_ENDIAN_REG 的 Bit 0、Bit 1 控制密钥的字节序，具体见表 22-3、表 22-4、表 22-5。表 22-3 中的 w[0] ~ w[3]、表 22-4 中的 w[0] ~ w[5]、表 22-5 中的 w[0] ~ w[7] 皆为标准 FIPS PUB 197 中 “5.2 Key

Expansion” 所述 “the first Nk words of the expanded key”。Bit 一列指明 w[0] ~ w[7] 每个 word 中的各个字节。三张表表明了在四种不同字节序下，寄存器 AES_KEY_*n*_REG 中的每个字节如何构成 “the first Nk words of the expanded key”。

文本字节序

寄存器 AES_ENDIAN_REG 的 Bit 2、Bit 3 控制输入文本的字节序，Bit 4、Bit 5 控制输出文本的字节序。输入文本在 AES-128/192/256 加密运算时指的是明文，在 AES-128/192/256 解密运算时指的是密文。输出文本在 AES-128/192/256 加密运算时指的是密文，在 AES-128/192/256 解密运算时指的是明文。具体见表 22-2。表 22-2 中的 State 为标准 FIPS PUB 197 中 “3.4 The State” 所述 “the AES algorithm’s operations are performed on a tow-dimensional array of bytes called the State”。此表表明了在四种不同字节序下，寄存器 AES_TEXT_*m*_REG 中的每个字节所存放的明文或密文如何构成 State。

表 22-2. AES 文本字节序

AES_ENDIAN_REG[3]/[5]	AES_ENDIAN_REG[2]/[4]	Plaintext/Ciphertext					
0	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_3_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_0_REG[31:24]
			1	AES_TEXT_3_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_0_REG[23:16]
			2	AES_TEXT_3_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_0_REG[15:8]
			3	AES_TEXT_3_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_0_REG[7:0]
0	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_3_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_0_REG[7:0]
			1	AES_TEXT_3_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_0_REG[15:8]
			2	AES_TEXT_3_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_0_REG[23:16]
			3	AES_TEXT_3_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_0_REG[31:24]
1	0	State		c			
				0	1	2	3
		r	0	AES_TEXT_0_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_3_REG[31:24]
			1	AES_TEXT_0_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_3_REG[23:16]
			2	AES_TEXT_0_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_3_REG[15:8]
			3	AES_TEXT_0_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_3_REG[7:0]
1	1	State		c			
				0	1	2	3
		r	0	AES_TEXT_0_REG[7:0]	AES_TEXT_1_REG[7:0]	AES_TEXT_2_REG[7:0]	AES_TEXT_3_REG[7:0]
			1	AES_TEXT_0_REG[15:8]	AES_TEXT_1_REG[15:8]	AES_TEXT_2_REG[15:8]	AES_TEXT_3_REG[15:8]
			2	AES_TEXT_0_REG[23:16]	AES_TEXT_1_REG[23:16]	AES_TEXT_2_REG[23:16]	AES_TEXT_3_REG[23:16]
			3	AES_TEXT_0_REG[31:24]	AES_TEXT_1_REG[31:24]	AES_TEXT_2_REG[31:24]	AES_TEXT_3_REG[31:24]

表 22-3. AES-128 密钥字节序

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]
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]

表 22-4. AES-192 密钥字节序

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]
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]

表 22-5. AES-256 密钥字节序

AES_ENDIAN_REG[1]	AES_ENDIAN_REG[0]	Bit	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7]
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[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]
		[23:16]	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]
		[15:8]	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]
		[7:0]	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]
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[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]	AES_KEY_4_REG[7:0]	AES_KEY_5_REG[7:0]	AES_KEY_6_REG[7:0]	AES_KEY_7_REG[7:0]
		[23:16]	AES_KEY_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]
		[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]	AES_KEY_6_REG[23:16]	AES_KEY_7_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]	AES_KEY_6_REG[31:24]	AES_KEY_7_REG[31:24]

22.3.4 加密与解密运算

单次运算

1. 初始化寄存器 AES_MODE_REG、AES_KEY_*n*_REG、AES_TEXT_*m*_REG、AES_ENDIAN_REG。
2. 对寄存器 AES_START_REG 写入 1。
3. 轮询寄存器 AES_IDLE_REG，直到从寄存器 AES_IDLE_REG 读出 1。
4. 从寄存器 AES_TEXT_*m*_REG 读取结果。

连续运算

每次运算完成之后，只有寄存器 AES_TEXT_*m*_REG 会被 AES 加速器更新，即寄存器 AES_MODE_REG、AES_KEY_*n*_REG、AES_ENDIAN_REG 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. 更新寄存器 AES_MODE_REG、AES_KEY_*n*_REG、AES_ENDIAN_REG 中需要更新的，其余不做变动。
2. 更新寄存器 AES_TEXT_*m*_REG。
3. 对寄存器 AES_START_REG 写入 1。
4. 轮询寄存器 AES_IDLE_REG，直到从寄存器 AES_IDLE_REG 读出 1。
5. 从寄存器 AES_TEXT_*m*_REG 读取结果。

22.3.5 运行效率

AES 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。

22.4 寄存器列表

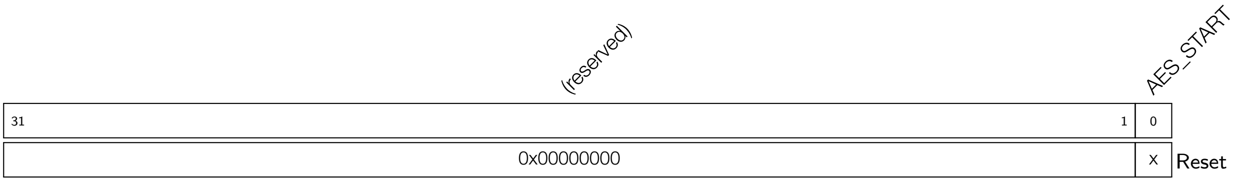
名称	描述	地址	访问
配置寄存器			
AES_MODE_REG	AES 加速器的运算模式	0x3FF01008	读 / 写
AES_ENDIAN_REG	字节序配置寄存器	0x3FF01040	读 / 写
密钥寄存器			
AES_KEY_0_REG	AES 密钥寄存器 0	0x3FF01010	读 / 写
AES_KEY_1_REG	AES 密钥寄存器 1	0x3FF01014	读 / 写
AES_KEY_2_REG	AES 密钥寄存器 2	0x3FF01018	读 / 写
AES_KEY_3_REG	AES 密钥寄存器 3	0x3FF0101C	读 / 写
AES_KEY_4_REG	AES 密钥寄存器 4	0x3FF01020	读 / 写
AES_KEY_5_REG	AES 密钥寄存器 5	0x3FF01024	读 / 写
AES_KEY_6_REG	AES 密钥寄存器 6	0x3FF01028	读 / 写
AES_KEY_7_REG	AES 密钥寄存器 7	0x3FF0102C	读 / 写
加密 / 解密数据寄存器			
AES_TEXT_0_REG	AES 加密/解密数据寄存器 0	0x3FF01030	读 / 写
AES_TEXT_1_REG	AES 加密/解密数据寄存器 1	0x3FF01034	读 / 写
AES_TEXT_2_REG	AES 加密/解密数据寄存器 2	0x3FF01038	读 / 写
AES_TEXT_3_REG	AES 加密/解密数据寄存器 3	0x3FF0103C	读 / 写
控制 / 状态寄存器			
AES_START_REG	AES 运算启动控制寄存器	0x3FF01000	只写

名称	描述	地址	访问
AES_IDLE_REG	AES 空闲状态寄存器	0x3FF01004	只读

22.5 寄存器

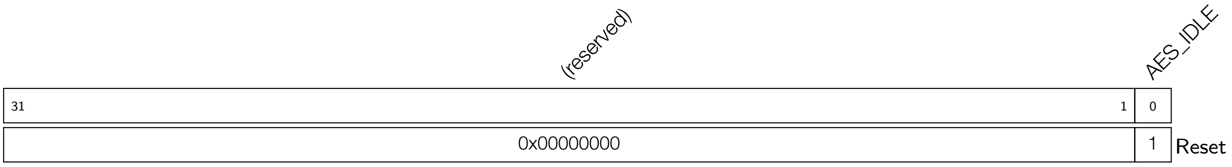
本小节括号中的地址均为相对于 AES 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 22.4 寄存器列表。

Register 22.1. AES_START_REG (0x000)



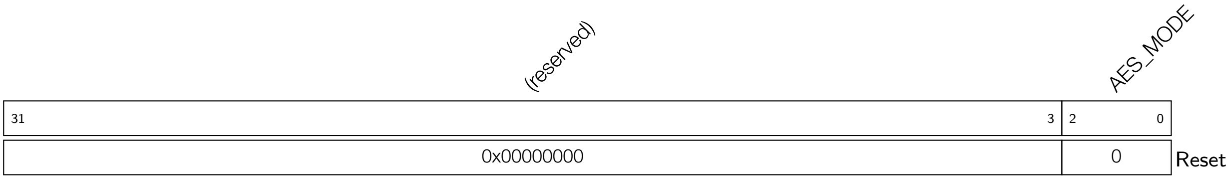
AES_START 写入 1 使能 AES 运算。（只写）

Register 22.2. AES_IDLE_REG (0x004)



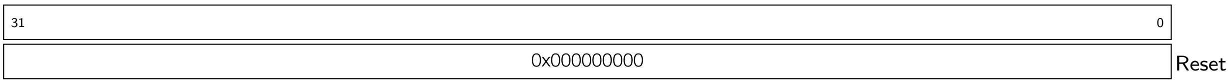
AES_IDLE AES 空闲寄存器。AES 加速器运行时读出 0，空闲时读出 1。（只读）

Register 22.3. AES_MODE_REG (0x008)



AES_MODE 选择 AES 加速器运算模式。详情请见表 22-1。（读 / 写）

Register 22.4. AES_KEY_*n*_REG (*n*: 0-7) (0x10+4**n*)



AES_KEY_*n*_REG (*n*: 0-7) AES 密钥寄存器。（读 / 写）

Register 22.5. AES_TEXT_*m*_REG (*m*: 0-3) (0x30+4**m*)

31	0
0x000000000	Reset

AES_TEXT_*m*_REG (*m*: 0-3) 明文和密文寄存器。(读 / 写)

Register 22.6. AES_ENDIAN_REG (0x040)

(reserved)	AES_ENDIAN						
31 6	5					0	
0x0000000	1	1	1	1	1	1	Reset

AES_ENDIAN 字节序选择寄存器。详情请见表 22-2。(读 / 写)

23 SHA 加速器 (SHA)

23.1 概述

相比于在软件中单独运行 SHA （安全哈希算法），SHA 加速器能够快速提升 SHA 的运行速度。SHA 加速器支持四种标准 FIPS PUB 180-4 运算：SHA-1、SHA-256、SHA-384 和 SHA-512。

23.2 主要特性

- 支持 SHA-1 运算
- 支持 SHA-256 运算
- 支持 SHA-384 运算
- 支持 SHA-512 运算

23.3 功能描述

23.3.1 填充解析信息

SHA 加速器每次只能接受一个信息块。软件将整个信息按照标准“FIPS PUB 180-4”中“5.2 Parsing the Message”的要求划分为一个一个的信息块，每次只将一个信息块写入寄存器 SHA_TEXT_*n*_REG。如果是 SHA-1、SHA-256 运算，则软件每次将 512 bit 的块写入寄存器 SHA_TEXT_0_REG ～ SHA_TEXT_15_REG。如果是 SHA-384、SHA-512 运算，则软件每次将 1024 bit 的块写入寄存器 SHA_TEXT_0_REG ～ SHA_TEXT_31_REG。

SHA 加速器不能自动完成标准“FIPS PUB 180-4”中“5.1 Padding the Message”所要求的填充操作；在将信息输入到加速器之前，需由软件来完成填充操作。

标准“FIPS PUB 180-4”中“2.2.1 Parameters”描述“$M_0^{(i)}$ is the left-most word of message block i”，此 word 存放在寄存器 SHA_TEXT_0_REG 中。以此类推寄存器 SHA_TEXT_1_REG 中存放的是信息中某个块从左向右的第二个 word⋯⋯

23.3.2 信息摘要

哈希运算完成之后，信息摘要被 SHA 加速器更新入寄存器 SHA_TEXT_*n*_REG。如果是 SHA-1 运算，则 160 bit 信息摘要存放在寄存器 SHA_TEXT_0_REG ～ SHA_TEXT_4_REG。如果是 SHA-256 运算，则 256 bit 信息摘要存放在寄存器 SHA_TEXT_0_REG ～ SHA_TEXT_7_REG。如果是 SHA-384 运算，则 384 bit 信息摘要存放在寄存器 SHA_TEXT_0_REG ～ SHA_TEXT_11_REG。如果是 SHA-512 运算，则 512 bit 信息摘要存放在寄存器 SHA_TEXT_0_REG ～ SHA_TEXT_15_REG。

标准“FIPS PUB 180-4”中“2.2.1 Parameters”描述“$H^{(N)}$ is the final hash value and is used to determine the message digest”、“$H_0^{(i)}$ is the left-most word of hash value i”。则信息摘要中最左边 word $H_0^{(N)}$ 在寄存器 SHA_TEXT_0_REG 中。以此类推信息摘要中从左至右的第二个 word $H_1^{(N)}$ 在寄存器 SHA_TEXT_1_REG 中⋯⋯

23.3.3 哈希运算

运算 SHA-1、SHA-256、SHA-384 和 SHA-512 各有一组控制寄存器；不同的哈希运算使用不同的控制寄存器。运算 SHA-1 使用寄存器 SHA_SHA1_START_REG、SHA_SHA1_CONTINUE_REG、SHA_SHA1_LOAD_REG 和 SHA_SHA1_BUSY_REG。运算 SHA-256 使用寄存器 SHA_SHA256_START_REG、SHA_SHA256_CONTINUE_REG、SHA_SHA256_LOAD_REG 和 SHA_SHA256_BUSY_REG。运算 SHA-384 使

用寄存器 SHA_SHA384_START_REG、SHA_SHA384_CONTINUE_REG、SHA_SHA384_LOAD_REG 和 SHA_SHA384_BUSY_REG。运算 SHA-512 使用寄存器 SHA_SHA512_START_REG、SHA_SHA512_CONTINUE_REG、SHA_SHA512_LOAD_REG 和 SHA_SHA512_BUSY_REG。具体操作流程如下。

1. 操作第一个信息块
   (a) 用第一个信息块初始化寄存器 SHA_TEXT_*n*_REG。
   (b) 对寄存器 SHA_*X*_START_REG 写入 1。
   (c) 轮询寄存器 SHA_*X*_BUSY_REG，直到从寄存器 SHA_*X*_BUSY_REG 读出 0。
2. 循环操作信息的后续每一个块
   (a) 用后续的信息块初始化寄存器 SHA_TEXT_*n*_REG。
   (b) 对寄存器 SHA_*X*_CONTINUE_REG 写入 1。
   (c) 轮询寄存器 SHA_*X*_BUSY_REG，直到从寄存器 SHA_*X*_BUSY_REG 读出 0。
3. 获取信息摘要
   (a) 对寄存器 SHA_*X*_LOAD_REG 写入 1。
   (b) 轮询寄存器 SHA_*X*_BUSY_REG，直到从寄存器 SHA_*X*_BUSY_REG 读出 0。
   (c) 从寄存器 SHA_TEXT_*n*_REG 取出信息摘要。

23.3.4 运行效率

SHA 加速器需要 60 至 100 个时钟周期来处理一个信息块以及 8 至 20 个时钟周期来计算最后的信息摘要。

23.4 寄存器列表

名称	描述	地址	访问
加密 / 解密数据寄存器			
SHA_TEXT_0_REG	SHA 加密/解密数据寄存器 0	0x3FF03000	读/写
SHA_TEXT_1_REG	SHA 加密/解密数据寄存器 1	0x3FF03004	读/写
SHA_TEXT_2_REG	SHA 加密/解密数据寄存器 2	0x3FF03008	读/写
SHA_TEXT_3_REG	SHA 加密/解密数据寄存器 3	0x3FF0300C	读/写
SHA_TEXT_4_REG	SHA 加密/解密数据寄存器 4	0x3FF03010	读/写
SHA_TEXT_5_REG	SHA 加密/解密数据寄存器 5	0x3FF03014	读/写
SHA_TEXT_6_REG	SHA 加密/解密数据寄存器 6	0x3FF03018	读/写
SHA_TEXT_7_REG	SHA 加密/解密数据寄存器 7	0x3FF0301C	读/写
SHA_TEXT_8_REG	SHA 加密/解密数据寄存器 8	0x3FF03020	读/写
SHA_TEXT_9_REG	SHA 加密/解密数据寄存器 9	0x3FF03024	读/写
SHA_TEXT_10_REG	SHA 加密/解密数据寄存器 10	0x3FF03028	读/写
SHA_TEXT_11_REG	SHA 加密/解密数据寄存器 11	0x3FF0302C	读/写
SHA_TEXT_12_REG	SHA 加密/解密数据寄存器 12	0x3FF03030	读/写
SHA_TEXT_13_REG	SHA 加密/解密数据寄存器 13	0x3FF03034	读/写
SHA_TEXT_14_REG	SHA 加密/解密数据寄存器 14	0x3FF03038	读/写

名称	描述	地址	访问
SHA_TEXT_15_REG	SHA 加密/解密数据寄存器 15	0x3FF0303C	读/写
SHA_TEXT_16_REG	SHA 加密/解密数据寄存器 16	0x3FF03040	读/写
SHA_TEXT_17_REG	SHA 加密/解密数据寄存器 17	0x3FF03044	读/写
SHA_TEXT_18_REG	SHA 加密/解密数据寄存器 18	0x3FF03048	读/写
SHA_TEXT_19_REG	SHA 加密/解密数据寄存器 19	0x3FF0304C	读/写
SHA_TEXT_20_REG	SHA 加密/解密数据寄存器 20	0x3FF03050	读/写
SHA_TEXT_21_REG	SHA 加密/解密数据寄存器 21	0x3FF03054	读/写
SHA_TEXT_22_REG	SHA 加密/解密数据寄存器 22	0x3FF03058	读/写
SHA_TEXT_23_REG	SHA 加密/解密数据寄存器 23	0x3FF0305C	读/写
SHA_TEXT_24_REG	SHA 加密/解密数据寄存器 24	0x3FF03060	读/写
SHA_TEXT_25_REG	SHA 加密/解密数据寄存器 25	0x3FF03064	读/写
SHA_TEXT_26_REG	SHA 加密/解密数据寄存器 26	0x3FF03068	读/写
SHA_TEXT_27_REG	SHA 加密/解密数据寄存器 27	0x3FF0306C	读/写
SHA_TEXT_28_REG	SHA 加密/解密数据寄存器 28	0x3FF03070	读/写
SHA_TEXT_29_REG	SHA 加密/解密数据寄存器 29	0x3FF03074	读/写
SHA_TEXT_30_REG	SHA 加密/解密数据寄存器 30	0x3FF03078	读/写
SHA_TEXT_31_REG	SHA 加密/解密数据寄存器 31	0x3FF0307C	读/写
控制 / 状态寄存器			
SHA_SHA1_START_REG	发起 SHA1 运算的控制寄存器	0x3FF03080	只写
SHA_SHA1_CONTINUE_REG	继续 SHA1 运算的控制寄存器	0x3FF03084	只写
SHA_SHA1_LOAD_REG	计算最终 SHA1 哈希运算结果的控制寄存器	0x3FF03088	只写
SHA_SHA1_BUSY_REG	SHA1 运算的状态寄存器	0x3FF0308C	只读
SHA_SHA256_START_REG	发起 SHA256 运算的控制寄存器	0x3FF03090	只写
SHA_SHA256_CONTINUE_REG	继续 SHA256 运算的控制寄存器	0x3FF03094	只写
SHA_SHA256_LOAD_REG	计算最终 SHA256 哈希运算结果的控制寄存器	0x3FF03098	只写
SHA_SHA256_BUSY_REG	SHA256 运算的状态寄存器	0x3FF0309C	只读
SHA_SHA384_START_REG	发起 SHA384 运算的控制寄存器	0x3FF030A0	只写
SHA_SHA384_CONTINUE_REG	继续 SHA384 运算的控制寄存器	0x3FF030A4	只写
SHA_SHA384_LOAD_REG	计算最终 SHA384 哈希运算结果的控制寄存器	0x3FF030A8	只写
SHA_SHA384_BUSY_REG	SHA384 运算的状态寄存器	0x3FF030AC	只读
SHA_SHA512_START_REG	发起 SHA512 运算的控制寄存器	0x3FF030B0	只写
SHA_SHA512_CONTINUE_REG	继续 SHA512 运算的控制寄存器	0x3FF030B4	只写
SHA_SHA512_LOAD_REG	计算最终 SHA512 哈希运算结果的控制寄存器	0x3FF030B8	只写
SHA_SHA512_BUSY_REG	SHA512 运算的状态寄存器	0x3FF030BC	只读

23.5 寄存器

本小节括号中的地址均为相对于 SHA 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 23.4 寄存器列表。

Register 23.1. SHA_TEXT_*n*_REG (*n*: 0-31) (0x0+4n*)**

31 0	
0x000000000	Reset

SHA_TEXT_*n*_REG (*n*: 0-31) SHA 信息块和哈希运算结果寄存器。（读/写）

Register 23.2. SHA_SHA1_START_REG (0x080)

(reserved)	SHA_SHA1_START	
31 1	0	
0x00000000	0	Reset

SHA_SHA1_START 写入 1 对第一个信息块进行 SHA-1 运算。（只写）

Register 23.3. SHA_SHA1_CONTINUE_REG (0x084)

(reserved)	SHA_SHA1_CONTINUE	
31 1	0	
0x00000000	0	Reset

SHA_SHA1_CONTINUE 写入 1 对后续的信息块继续进行 SHA-1 运算。（只写）

Register 23.4. SHA_SHA1_LOAD_REG (0x088)

(reserved)	SHA_SHA1_LOAD	
31 1	0	
0x00000000	0	Reset

SHA_SHA1_LOAD 写入 1 结束 SHA-1 运算，计算最终的运算结果。（只写）

Register 23.5. SHA_SHA1_BUSY_REG (0x08C)

(reserved)		SHA_SHA1_BUSY	
31	1	0	
0x00000000		0	Reset

SHA_SHA1_BUSY SHA-1 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。（只读）

Register 23.6. SHA_SHA256_START_REG (0x090)

(reserved)		SHA_SHA256_START	
31	1	0	
0x00000000		0	Reset

SHA_SHA256_START 写入 1 对第一个信息块进行 SHA-256 运算。（只写）

Register 23.7. SHA_SHA256_CONTINUE_REG (0x094)

(reserved)		SHA_SHA256_CONTINUE	
31	1	0	
0x00000000		0	Reset

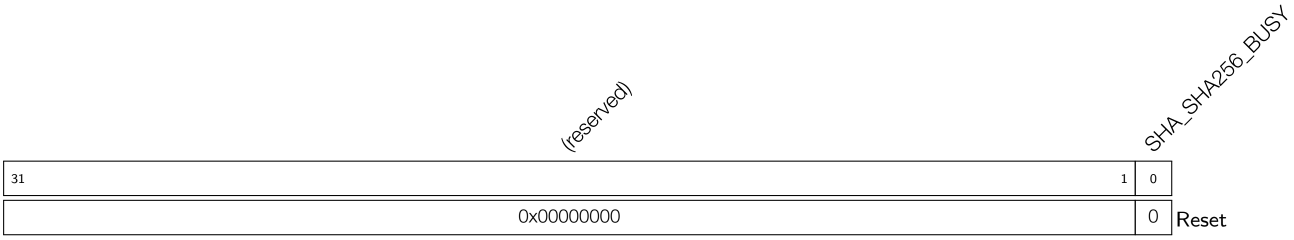
SHA_SHA256_CONTINUE 写入 1 对后续的信息块继续进行 SHA-256 运算。（只写）

Register 23.8. SHA_SHA256_LOAD_REG (0x098)

(reserved)		SHA_SHA256_LOAD	
31	1	0	
0x00000000		0	Reset

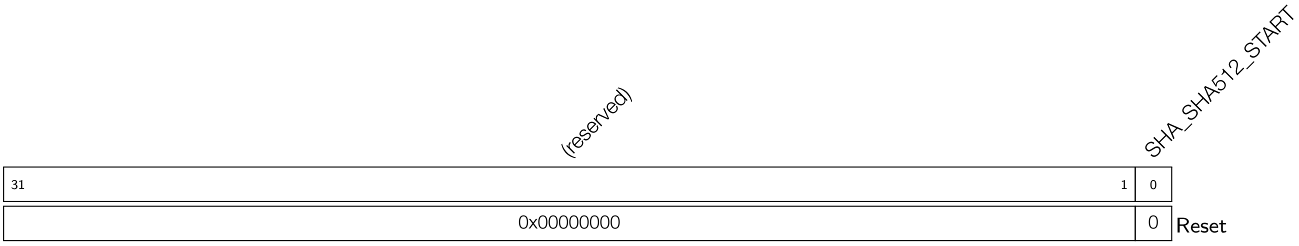
SHA_SHA256_LOAD 写入 1 结束 SHA-256 运算，计算最终的运算结果。（只写）

Register 23.9. SHA_SHA256_BUSY_REG (0x09C)



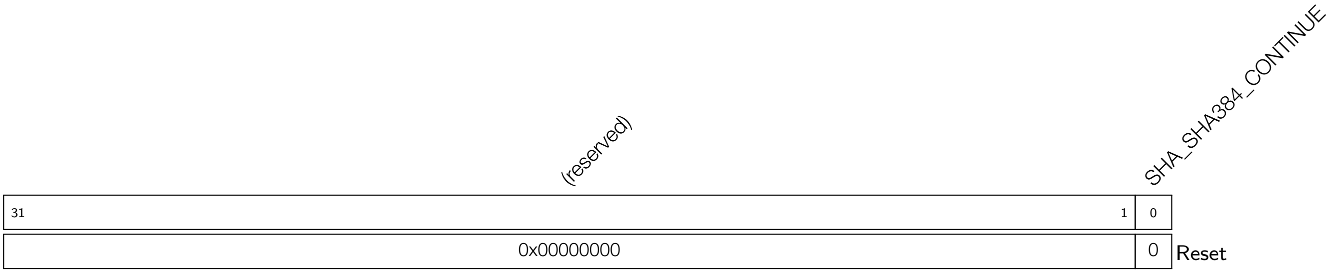
SHA_SHA256_BUSY SHA-256 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。（只读）

Register 23.10. SHA_SHA384_START_REG (0x0A0)



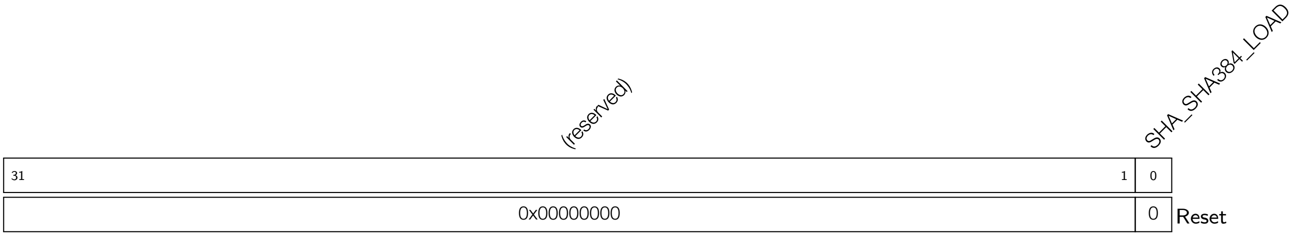
SHA_SHA384_START 写入 1 对第一个信息块进行 SHA-384 运算。（只写）

Register 23.11. SHA_SHA384_CONTINUE_REG (0x0A4)



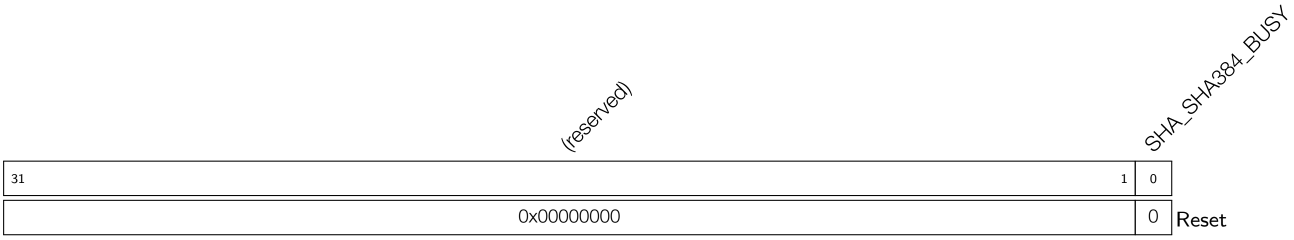
SHA_SHA384_CONTINUE 写入 1 对后续的信息块继续进行 SHA-384 运算。（只写）

Register 23.12. SHA_SHA384_LOAD_REG (0x0A8)



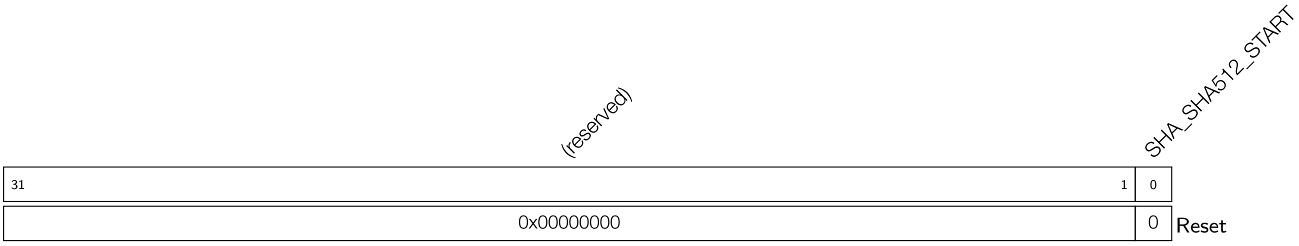
SHA_SHA384_LOAD 写入 1 结束 SHA-384 运算，计算最终的运算结果。（只写）

Register 23.13. SHA_SHA384_BUSY_REG (0x0AC)



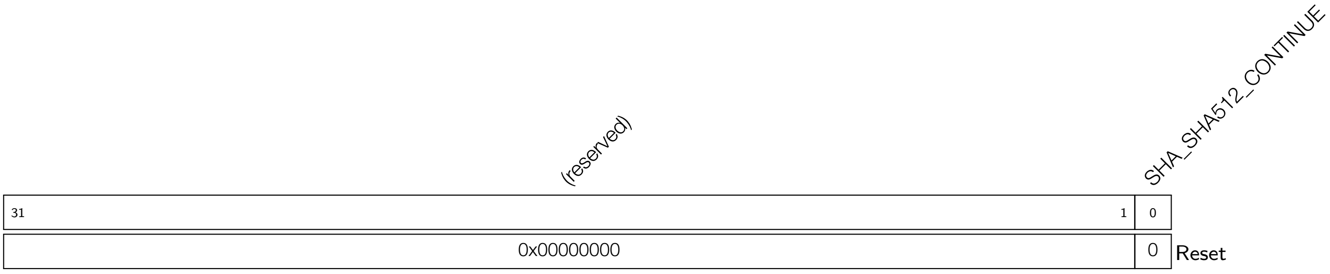
SHA_SHA384_BUSY SHA-384 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。（只读）

Register 23.14. SHA_SHA512_START_REG (0x0B0)



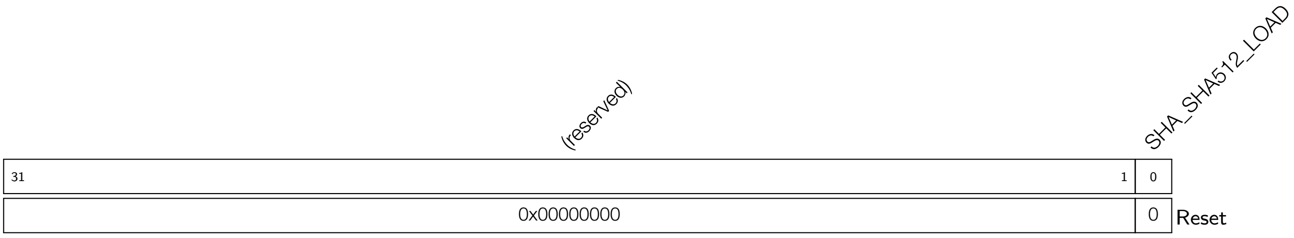
SHA_SHA512_START 写入 1 对第一个信息块进行 SHA-512 运算。（只写）

Register 23.15. SHA_SHA512_CONTINUE_REG (0x0B4)



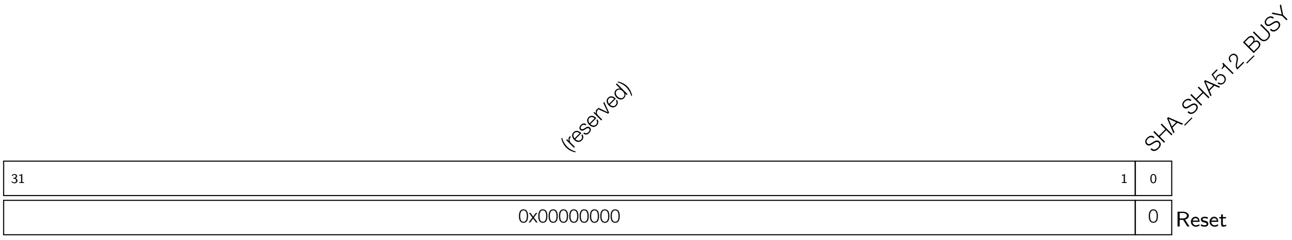
SHA_SHA512_CONTINUE 写入 1 对后续的信息块继续进行 SHA-512 运算。（只写）

Register 23.16. SHA_SHA512_LOAD_REG (0x0B8)



SHA_SHA512_LOAD 写入 1 结束 SHA-512 运算，计算最终的运算结果。（只写）

Register 23.17. SHA_SHA512_BUSY_REG (0x0BC)



SHA_SHA512_BUSY SHA-512 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。（只读）

24 RSA 加速器 (RSA)

24.1 概述

RSA 加速器为多种运用于 RSA 非对称式加密演算法的高精度计算提供硬件支持。这些高精度计算包括大数模幂运算、大数模乘运算和大数乘法运算等。

RSA 加速器极大地降低了这三种运算的软件复杂度，并且支持多种运算子长度，使得运算效率不被浪费。

24.2 主要特性

- 支持大数模幂运算
- 支持大数模乘运算
- 支持大数乘法运算
- 支持多种运算子长度

24.3 功能描述

24.3.1 初始化

通过使能对应的外设时钟并且清零 DPORT_RSA_PD_CTRL_REG 寄存器中的 DPORT_RSA_PD 位即可以复位释放并使能 RSA 加速器。

当 RSA 加速器被复位释放后，寄存器 RSA_CLEAN_REG 读到 0，初始化开始。硬件将 4 块存储器初始化为 0。初始化过程完成后，寄存器 RSA_CLEAN_REG 读到 1。因此，RSA 加速器被复位之后第一次使用时，软件需要先查询寄存器 RSA_CLEAN_REG，以确保 RSA 加速器处于可工作状态。

24.3.2 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。所以对于大数模幂运算，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 $N \in \{512, 1024, 1536, 2048, 2560, 3072, 3584, 4096\}$ 8 种长度的大数模幂运算。即 Z、X、Y、M 和 $\bar{r}$ 的位宽为这 8 种中的任意一种，但是它们的位宽必须都相同，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 $Z_{n-1} \sim Z_0$、$X_{n-1} \sim X_0$、$Y_{n-1} \sim Y_0$、$M_{n-1} \sim M_0$、$\overline{r}_{n-1} \sim \overline{r}_0$ 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}、$\overline{r}_{n-1}$ 分别为 Z、X、Y、M、$\overline{r}$ 最高位的 b 进制数，而 Z_0、X_0、Y_0、M_0、$\overline{r}_0$ 分别为 Z、X、Y、M、$\overline{r}$ 最低位的 b 进制数。

另设

$$R = b^n$$

则计算得参数：

$$\overline{r} = R^2 \text{ mod } M \tag{1}$$

$$\begin{cases} M'' \times M + 1 = R \times R^{-1} \\ M' = M'' \text{ mod } b \end{cases} \tag{2}$$

（公式 (2) 的形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_MODEXP_MODE_REG 写入 $(\frac{N}{512} - 1)$。
2. 将 X_i、Y_i、M_i、$\overline{r}_i$ $(i \in [0, n) \cap \mathbb{N})$ 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 word。每块存储器的每一个 word 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。
3. 将 M' 写入寄存器 RSA_M_PRIME_REG。
4. 对寄存器 RSA_MODEXP_START_REG 写入 1。
5. 等待运算结束。轮询寄存器 RSA_INTERRUPT_REG 直到读到 1，或者等待 RSA_INTR 中断产生。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in [0, n) \cap \mathbb{N})$。
7. 对寄存器 RSA_INTERRUPT_REG 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODEXP_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 $\overline{r}_i$ 都已经被覆盖了。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。

24.3.3 大数模乘运算

大数模乘运算也是基于 Montgomery Multiplication 实现运算 $Z = X \times Y \text{ mod } M$，所以也需要如式 1、式 2 预先通过软件计算得到 $\overline{r}$ 和 M'。

RSA 加速器也支持 8 种运算子长度的大数模乘运算。大数模乘运算采用软硬件相结合的方式，运算期间需要软件介入一次。

大数模乘运算的软件流程为：

1. 对寄存器 RSA_MULT_MODE_REG 写入 $(\frac{N}{512} - 1)$。
2. 将 X_i、M_i、$\overline{r}_i$ $(i \in [0, n) \cap \mathbb{N})$ 分别写入存储器 RSA_X_MEM、RSA_M_MEM、RSA_Z_MEM。同样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。

3. 将 M' 写入寄存器 RSA_M_PRIME_REG。
4. 对寄存器 RSA_MULT_START_REG 写入 1。
5. 等待第一轮运算结束。轮询寄存器 RSA_INTERRUPT_REG 直到读到 1，或者等待 RSA_INTR 中断产生。
6. 对寄存器 RSA_INTERRUPT_REG 写入 1 以清除中断。
7. 将 Y_i $(i \in [0, n) \cap \mathbb{N})$ 写入存储器 RSA_X_MEM。只需要根据运算子长度，将运算子中有效的数据写入存储器，没有使用到的存储器不用更改。
8. 对寄存器 RSA_MULT_START_REG 写入 1。
9. 等待第二轮运算结束。轮询寄存器 RSA_INTERRUPT_REG 直到读到 1，或者等待 RSA_INTR 中断产生。
10. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in [0, n) \cap \mathbb{N})$。
11. 对寄存器 RSA_INTERRUPT_REG 写入 1 以清除中断。

运算结束后，只有寄存器 RSA_MULT_MODE_REG 中存储的运算模式与运算子长度信息以及存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 没有被覆盖。所以连续运算时，可以不再对这些寄存器与存储器重复写入。

24.3.4 大数乘法运算

大数乘法运算实现了 $Z = X \times Y$。其中运算子 Z 的长度是运算子 X、Y 长度的两倍。所以 RSA 加速器只支持 4 种 $N \in \{512, 1024, 1536, 2048\}$ 运算子长度的大数乘法运算。运算子 Z 的长度 $\hat{N}$ 为 $2 \times N$。

首先构造 $\hat{X}$、$\hat{Y}$，它们与运算子 Z 的长度一致 $(\hat{N})$。

$$n = \frac{N}{32}$$

$$\hat{N} = 2 \times N$$

$$\hat{n} = \frac{\hat{N}}{32} = 2n$$

$$\hat{X} = (\hat{X}_{\hat{n}-1}\hat{X}_{\hat{n}-2}\cdots\hat{X}_0)_b = (\underbrace{00\cdots0}_{n}X)_b = (\underbrace{00\cdots0}_{n}X_{n-1}X_{n-2}\cdots X_0)_b$$

$$\hat{Y} = (\hat{Y}_{\hat{n}-1}\hat{Y}_{\hat{n}-2}\cdots\hat{Y}_0)_b = (Y\underbrace{00\cdots0}_{n})_b = (Y_{n-1}Y_{n-2}\cdots Y_0\underbrace{00\cdots0}_{n})_b$$

大数乘法运算的软件流程为：

1. 对寄存器 RSA_MULT_MODE_REG 写入 $(\frac{\hat{N}}{512} - 1 + 8)$。
2. 将 $\hat{X}_i$、$\hat{Y}_i$ $(i \in [0, \hat{n}) \cap \mathbb{N})$ 分别写入存储器 RSA_X_MEM、RSA_Z_MEM。

   只需要根据运算子长度，将这两个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。写入存储器的若干 b 进制数中有一半都是 0，这些 0 不可或缺。
3. 对寄存器 RSA_MULT_START_REG 写入 1。
4. 等待运算结束。轮询寄存器 RSA_INTERRUPT_REG 直到读到 1，或者等待 RSA_INTR 中断产生。
5. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in [0, \hat{n}) \cap \mathbb{N})$。
6. 对寄存器 RSA_INTERRUPT_REG 写入 1 以清除中断。

运算结束后，只有寄存器 RSA_MULT_MODE_REG 中存储的运算模式与运算子长度信息没被更改。

24.4 寄存器列表

名称	描述	地址	访问
配置寄存器			
RSA_M_PRIME_REG	存储 M' 的寄存器	0x3FF02800	读/写
模幂运算寄存器			
RSA_MODEXP_MODE_REG	模幂运算模式	0x3FF02804	读/写
RSA_MODEXP_START_REG	起始位	0x3FF02808	只写
模乘运算寄存器			
RSA_MULT_MODE_REG	模乘运算模式	0x3FF0280C	读/写
RSA_MULT_START_REG	起始位	0x3FF02810	只写
其他寄存器			
RSA_INTERRUPT_REG	RSA 中断寄存器	0x3FF02814	读/写
RSA_CLEAN_REG	RSA 清除寄存器	0x3FF02818	只读

24.5 寄存器

本小节括号中的地址均为相对于 RSA 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 24.4 寄存器列表。

Register 24.1. RSA_M_PRIME_REG (0x800)

31	0	
0x000000000		Reset

RSA_M_PRIME_REG 此寄存器包含 M’。（读/写）

Register 24.2. RSA_MODEXP_MODE_REG (0x804)

(reserved)	RSA_MODEXP_MODE	
31 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0	Reset

RSA_MODEXP_MODE 此寄存器包含模幂运算的模式。（读/写）

Register 24.3. RSA_MODEXP_START_REG (0x808)

(reserved)	RSA_MODEXP_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_MODEXP_START 写入 1 以开始模幂运算。（只写）

Register 24.4. RSA_MULT_MODE_REG (0x80C)

(reserved)	RSA_MULT_MODE	
31 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0	Reset

RSA_MULT_MODE 此寄存器包含模乘和乘法运算。（只写）

Register 24.5. RSA_MULT_START_REG (0x810)

(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 以开始模乘和乘法运算。(只写)

Register 24.6. RSA_INTERRUPT_REG (0x814)

(reserved)	RSA_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_INTERRUPT RSA 中断状态寄存器。一次运算结束即读到 1。(读/写)

Register 24.7. RSA_CLEAN_REG (0x818)

(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	Reset

RSA_CLEAN 一旦存储器初始化结束，此位即读到 1。(只读)

25 随机数发生器 (RNG)

25.1 概述

ESP32 内置一个真随机数发生器，其生成的 32 位随机数可作为加密等操作的基础。

25.2 主要特性

ESP32 的随机数发生器可通过物理过程而非算法生成真随机数，所有生成的随机数在特定范围内出现的概率完全一致。

25.3 功能描述

系统可以从随机数发生器的寄存器 RNG_DATA_REG 中读取随机数，每个读到的 32 位随机数都是真随机数，噪声源为系统中的热噪声和异步时钟。

具体来说，这些热噪声可以来自 SAR ADC 或高速 ADC 或两者兼有。当芯片的 SAR ADC 或高速 ADC 工作时，就会产生比特流，并通过异或 (XOR) 逻辑运算作为随机数种子进入随机数生成器。

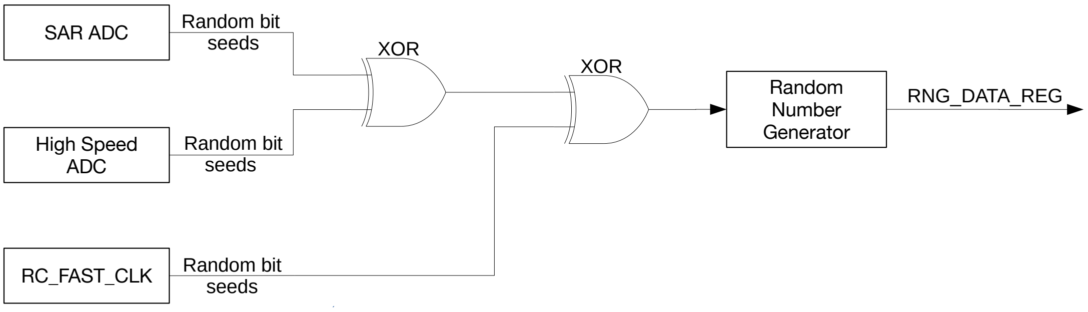


图 25-1. 噪声源

当 SAR ADC 打开时，每个 RC_FAST_CLK （8 MHz）时钟周期内（来自内部 RC 振荡器，详见 3 复位和时钟章节），随机数发生器将获得 2 位的熵。因此，为了获得最大的熵值，建议读取 RNG_DATA_REG 寄存器时的速率不超过 500 kHz。

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 寄存器时的速率不超过 5 MHz。

我们在仅打开高速 ADC 的状态下，以 5 MHz 的速率从 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。

25.4 编程指南

在使用 ESP32 的随机数生成器时，应该至少打开 SAR ADC 或高速 ADC，否则可能会导致产生伪随机数，应注意避免。其中，

- SAR ADC 可通过 DIG ADC 控制器打开，详见 29 片上传感器与模拟信号处理章节。
- 高速 ADC 在 Wi-Fi 或蓝牙开启时自动打开。

说明：

注意，在 Wi-Fi 开启时，极端情况下高速 ADC 有读值饱和的可能，这会降低熵值。因此，建议在 Wi-Fi 开启时，同时通过 DIG ADC1 控制器打开 SAR ADC 产生随机数。

在使用随机数生成器时，请多次读取 RNG_DATA_REG 寄存器的值，直至获得足够多的随机数。在读取寄存器时，注意控制速率不要超过上方第 25.3 小节的介绍。

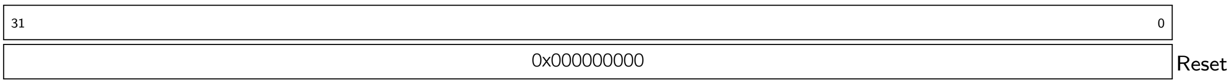
25.5 寄存器列表

名称	描述	地址	访问
RNG_DATA_REG	随机数数据	0x3FF75144	只读

25.6 寄存器

本小节括号中的地址均为相对于 RNG 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 25.5 寄存器列表。

Register 25.1. RNG_DATA_REG (0x144)



RNG_DATA_REG 随机数来源。（只读）

26 片外存储器加密与解密 (FLASH)

26.1 概述

ESP32 系列的多款芯片需要将程序和数据存储在片外 flash。片外 flash 可以用来存储专有软件、敏感的用户数据（比如用来访问私有网络的凭据）等。ESP32 的 flash 加密模块能够将代码进行加密，然后将加密之后的代码写入片外 flash。当 CPU 通过 cache 读片外 flash 时，flash 解密模块能够将读取到的指令和数据自动进行解密。ESP32 通过 flash 加解密模块为用户的应用代码提供了安全保障。

26.2 主要特性

- 多种密钥生成方式
- 加密过程需要软件参与
- 高速的解密过程，无需软件参与
- 寄存器配置、系统参数、启动 (boot) 模式共同决定 flash 加解密功能

26.3 功能描述

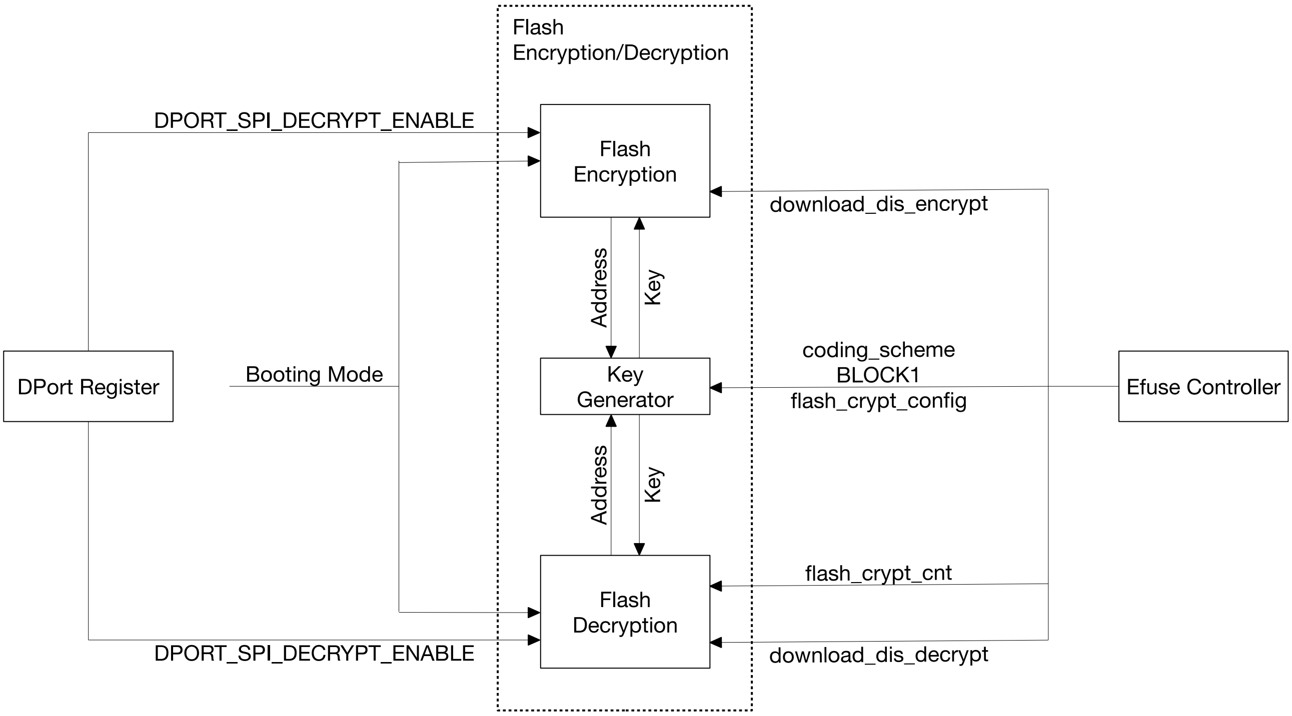


图 26-1. Flash 加解密模块架构

Flash 加解密模块包含三个部分，分别是密钥生成模块 (Key Generator)、flash 加密 (Flash Encryption) 模块和 flash 解密 (Flash Decryption) 模块，如图 26-1 所示。Key Generator 被 Flash Encryption 与 Flash Decryption 共同使用。Flash Encryption 与 Flash Decryption 可以同时工作。

外设 DPort 寄存器中与 flash 加解密相关的寄存器是 DPORT_SLAVE_SPI_CONFIG_REG 中的 DPORT_SPI_ENCRYPT_ENABLE 位和 DPORT_SPI_DECRYPT_ENABLE 位。Flash 加解密模块还会从外设 eFuse 控制器中获取 6 个系统参数，它们是 coding_scheme、BLOCK1、flash_crypt_config、download_dis_encrypt、flash_crypt_cnt 和 download_dis_decrypt。

26.3.1 Key Generator 模块

Key Generator 首先依据系统参数 coding_scheme、BLOCK1 生成
$Key_o = f(coding\_scheme, BLOCK1)$。

然后 Key Generator 根据系统参数 flash_crypt_config 和 flash 加解密模块访问的片外 flash 的实地址 $Addr_e$、$Addr_d$ 分别运算出
$Key_e = g(Key_o, flash\_crypt\_config, Addr_e)$、
$Key_d = g(Key_o, flash\_crypt\_config, Addr_d)$。

当系统参数 flash_crypt_config 为全 0 时，Key_e、Key_d 与片外 flash 的实地址无关。当系统参数 flash_crypt_config 为除全 0 之外任意值时，片外 flash 每 8 个 word 具有一个其独特的 Key_e 和 Key_d。

26.3.2 Flash Encryption 模块

Flash Encryption 是一个外设模块，自身带有寄存器，这些寄存器可以被 CPU 直接访问。模块内的寄存器、外设 DPort 寄存器、系统参数、boot 模式共同配置并使用这一模块。

此模块工作时需要软件参与，其流程为：

1. 将寄存器 DPORT_SLAVE_SPI_CONFIG_REG 的 DPORT_SPI_ENCRYPT_ENABLE 位置 1。
2. 将准备写片外 flash 的实地址填入寄存器 FLASH_ENCRYPT_ADDRESS_REG，此地址值必须是 8-word 对齐的。
3. 将待加密的 8-word 代码中的最低地址的 word 填入寄存器 FLASH_ENCRYPT_BUFFER_0_REG，次低地址的 word 填入 FLASH_ENCRYPT_BUFFER_1_REG，以此类推直至填入 FLASH_ENCRYPT_BUFFER_7_REG。
4. 将寄存器 FLASH_ENCRYPT_START_REG 置 1。
5. 轮询寄存器 FLASH_ENCRYPT_DONE_REG，直到读到 1。
6. 调用函数，通过外设 SPI0 对片外 flash 的 8-word 对齐的地址写入任意 8-word 代码。

步骤 1 至 5 操作 Flash Encryption 模块对 8-word 代码进行加密。加密算法使用的密钥就是 Key_e。加密结果也是 8 个 word。步骤 6 操作外设 SPI0 将 Flash Encryption 的加密结果写入片外 flash。步骤 6 中调用的函数会有一个参数是写片外 flash 的实地址，这个实地址必须是 8-word 对齐的，且必须与步骤 2 中填入寄存器 FLASH_ENCRYPT_ADDRESS_REG 的值一致。虽然步骤 6 中调用的函数还会有一个参数是 8-word 代码，但是在执行了步骤 1 至 5 的情况下，这个参数无意义，外设 SPI0 此时使用的是加密后的结果。若 Flash Encryption 处于未工作状态或者不执行步骤 1 至 5，那么步骤 6 不会使用加密结果而是函数中的参数。

Flash Encryption 是否处于工作状态取决于：

- SPI Flash Boot 模式下

  当寄存器 DPORT_SLAVE_SPI_CONFIG_REG 的 DPORT_SPI_ENCRYPT_ENABLE 位为 1 时，Flash Encryption 处于工作状态，否则处于未工作状态。

- Download Boot 模式下

  当寄存器 DPORT_SLAVE_SPI_CONFIG_REG 的 DPORT_SPI_ENCRYPT_ENABLE 位为 1 且系统参数 download_dis_encrypt 为 0 时，Flash Encryption 处于工作状态，否则处于未工作状态。

虽然整个过程都有软件参与，但是加密后的代码无法被软件直接读取。加密后的代码被直接写到片外 flash 中。虽然 CPU 可以不通过 cache 而直接读片外 flash 从而得到加密代码，但软件还是绝对无法获取到密钥

Key_e。

26.3.3 Flash Decryption 模块

Flash Decryption 并非外设模块，自身不带寄存器，因此不能被 CPU 直接访问。外设 DPort 寄存器、系统参数、boot 模式共同配置并使用这一模块。

当此模块工作时，CPU 通过 cache 读取片外 flash 中的指令与数据。Flash Decryption 将自动把读取到的指令与数据进行解密。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就是 Key_d，此密钥同样无法被软件获取。

当此模块未工作时，不对 flash 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 flash 中的原始内容。

Flash Decryption 是否处于工作状态取决于：

- SPI Flash Boot 模式下

  当系统参数 flash_crypt_cnt（7 bit 宽）中有奇数个 bit 为 1 时，Flash Decryption 处于工作状态，否则处于未工作状态。

- Download Boot 模式下

  当寄存器 DPORT_SLAVE_SPI_CONFIG_REG 的 DPORT_SPI_DECRYPT_ENABLE 位为 1 且系统参数 download_dis_decrypt 为 0 时，Flash Decryption 处于工作状态，否则处于未工作状态。

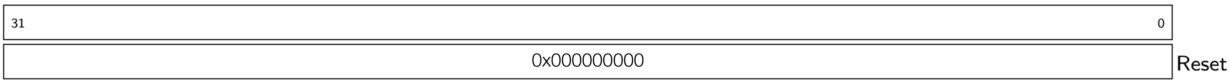
26.4 寄存器列表

名称	描述	地址	访问
FLASH_ENCRYPTION_BUFFER_0_REG	Flash 加密 buffer 寄存器 0	0x3FF5B000	只写
FLASH_ENCRYPTION_BUFFER_1_REG	Flash 加密 buffer 寄存器 1	0x3FF5B004	只写
FLASH_ENCRYPTION_BUFFER_2_REG	Flash 加密 buffer 寄存器 2	0x3FF5B008	只写
FLASH_ENCRYPTION_BUFFER_3_REG	Flash 加密 buffer 寄存器 3	0x3FF5B00C	只写
FLASH_ENCRYPTION_BUFFER_4_REG	Flash 加密 buffer 寄存器 4	0x3FF5B010	只写
FLASH_ENCRYPTION_BUFFER_5_REG	Flash 加密 buffer 寄存器 5	0x3FF5B014	只写
FLASH_ENCRYPTION_BUFFER_6_REG	Flash 加密 buffer 寄存器 6	0x3FF5B018	只写
FLASH_ENCRYPTION_BUFFER_7_REG	Flash 加密 buffer 寄存器 7	0x3FF5B01C	只写
FLASH_ENCRYPTION_START_REG	加密操作控制寄存器	0x3FF5B020	只写
FLASH_ENCRYPTION_ADDRESS_REG	片外 flash 地址寄存器	0x3FF5B024	只写
FLASH_ENCRYPTION_DONE_REG	加密操作状态寄存器	0x3FF5B028	只读

26.5 寄存器

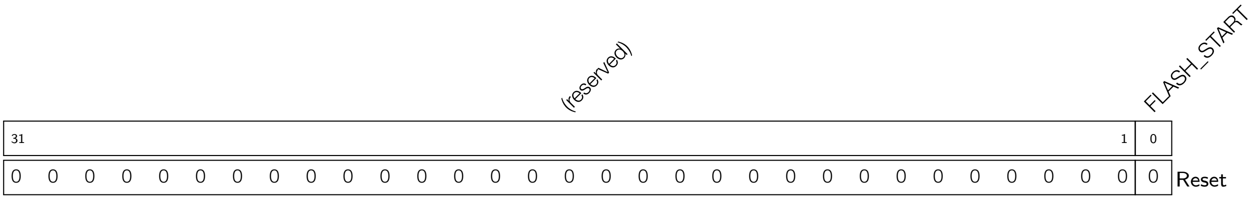
本小节括号中的地址均为相对于 FLASH 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器中的表 1-6 外设地址映射。寄存器绝对地址见章节 26.4 寄存器列表。

Register 26.1. FLASH_ENCRYPTION_BUFFER_*n*_REG (*n*: 0-7) (0x0+4n*)**



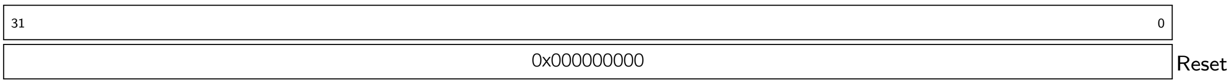
FLASH_ENCRYPTION_BUFFER_*n*_REG 解密的 buffer 数。（只写）

Register 26.2. FLASH_ENCRYPTION_START_REG (0x020)



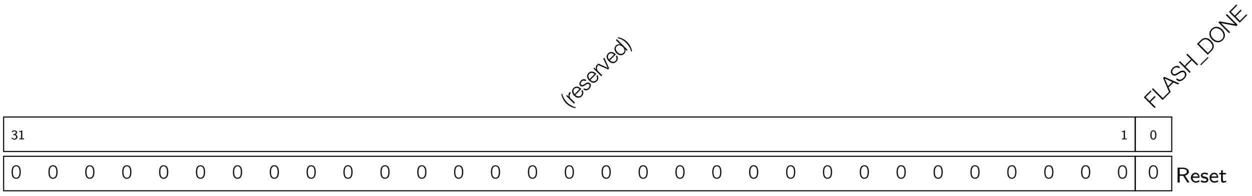
FLASH_START 将此位置为 1，启动加密操作。（只写）

Register 26.3. FLASH_ENCRYPTION_ADDRESS_REG (0x024)



FLASH_ENCRYPTION_ADDRESS_REG 片外 flash 的实地址，这个实地址必须是 8-word 对齐的。（只写）

Register 26.4. FLASH_ENCRYPTION_DONE_REG (0x028)



FLASH_DONE 加密操作完成后，此位为 1。（只读）

27 存储器管理和保护单元 (MMU, MPU)

27.1 概述

ESP32 中的每个外设和存储器通过 MMU（存储器管理单元）或 MPU（存储器保护单元）被访问。根据 OS 给予应用的权限，MPU 和 MMU 可以允许或禁止应用访问存储器某部分和外设。MMU 还可以进行虚地址和实地址的转换，将片上或片外存储器的地址范围映射到某个虚拟存储器区域。这些映射可根据应用程序配置，因此每个应用程序的内存可根据其运行所需进行配置。OS 和应用程序运行时，分别配有一个进程号（即 PID），用于区分彼此。进程号共有 8 个。每个 OS 和应用程序都有自己的映射和权限。

27.2 主要特性

- PRO_CPU 与 APP_CPU 各有 8 个进程
- MPU/MMU 基于进程的 PID 对片上存储器、片外存储器、外设进行管理
- 片上存储器由 MPU/MMU 管理
- 片外存储器由 MMU 管理
- 外设由 MPU 管理

27.3 功能描述

27.3.1 PID 控制器

在 ESP32 中，PID 控制器充当指示器，向 MMU/MPU 通知当前运行代码的程序的 PID。OS 每次将上下文切换到另一应用时，会更新 PID 控制器中的 PID。通过配置，PID 控制器可以检测中断并自动将 PID 切换到 OS 的 PID。

系统中有两个外设 PID 控制器，分别用于 ESP32 中的两个 CPU。每个 CPU 各有一个 PID 控制器，能够在需要时允许不同的 CPU 上运行不同的进程。

27.3.2 MPU/MMU

MPU 和 MMU 基于访问外设和存储器的进程管理片上存储器，片外存储器和外设。当代码试图访问受 MMU/MPU 保护的存储器区域或外设时，MMU 或 MPU 将从运行该进程的 CPU 所对应的 PID 生成器接收 PID。

MMU 和 MPU 仅基于该 PID 对片上存储器和片外存储器进行管理，并不考虑运行代码的是哪个 CPU; 因此，内部存储器和外设的 MMU 和 MPU 只有 8 个不同 PID 的配置选项。相比之下，管理片外存储器的 MMU 不仅识别 PID，而且还识别发送请求访问片外存储器的 CPU。即不论程序运行在 APP_CPU 或 PRO_CPU 上，MMU 都具有每个 PID 的配置选项。实际应用中对于同一个进程的配置，使得来自两个 CPU 的访问能够获取到相同的内容，但这样做并不是硬件要求。

基于 PID 的配置选项，MPU 可以允许或拒绝进程访问存储器区域或外设。MMU 也具有该功能，并且能将进程的虚地址访问转换为实地址访问，从而访问可能完全不同的物理存储器区域。因此，MMU 管理的存储器所实现的映射可以基于进程分别配置。

27.3.2.1 嵌入式存储器

片上存储器由功能固定的 MPU，可配置的 MPU 和 MMU 管理：

表 27-1. 片上存储器的 MPU 和 MMU 结构

存储器名字	大小	地址范围		管理存储器的 MPU
		开始	结束	
ROM0	384 KB	0x4000_0000	0x4005_FFFF	Static MPU
ROM1	64 KB	0x3FF9_0000	0x3FF9_FFFF	Static MPU
SRAM0	64 KB	0x4007_0000	0x4007_FFFF	Static MPU
	128 KB	0x4008_0000	0x4009_FFFF	SRAM0 MMU
SRAM1 (多地址访问)	128 KB	0x3FFE_0000	0x3FFF_FFFF	Static MPU
	128 KB	0x400A_0000	0x400B_FFFF	Static MPU
	32 KB	0x4000_0000	0x4000_7FFF	Static MPU
SRAM2	72 KB	0x3FFA_E000	0x3FFB_FFFF	Static MPU
	128 KB	0x3FFC_0000	0x3FFD_FFFF	SRAM2 MMU
RTC FAST (多地址访问)	8 KB	0x3FF8_0000	0x3FF8_1FFF	RTC FAST MPU
	8 KB	0x400C_0000	0x400C_1FFF	RTC FAST MPU
RTC SLOW	8 KB	0x5000_0000	0x5000_1FFF	RTC SLOW MPU

Static MPU

ROM0，ROM1，SRAM0 的低 64 KB，SRAM1 和 SRAM2 的低 72 KB 由 Static MPU 控制。这些 MPU 由硬件控制，不能由软件配置。它们仅通过当前进程的 PID 管理进程对存储器区域的访问。当进程的 PID 为 0 或 1 时，可以使用表 27-1 中指定的地址段读取（当存储器为 RAM 时用相应地址段写入）存储器。当 PID 为 2 到 7，存储器不可被访问。

RTC FAST & RTC SLOW MPU

8 KB 的 RTC FAST Memory 以及 8 KB 的 RTC SLOW Memory 由两个可配置的 MPU 控制。通过配置 RTC_CNTL_ RTC_PID_CONFIG_REG 和 DPORT_AHBLITE_MPU_TABLE_RTC_REG 寄存器，MPU 可以允许或禁止每个单独的 PID 访问存储器。通过设置这些寄存器中某个位，可以允许相应的 PID 从存储器读取或写入数据，清除该位则禁止相应的 PID 从存储器读取或写入数据。PID 为 0 和 1 的进程始终可以访问 RTC SLOW 存储器，该配置无法修改。表 27-2 和 27-3 描述了寄存器位与 PID 之间的映射关系。

表 27-2. 管理 RTC FAST Memory 的 MPU

大小	边界地址		权限
	低	高	PID RTC_CNTL_RTC_PID_CONFIG 位
8 KB	0x3FF8_0000	0x3FF8_1FFF	0 1 2 3 4 5 6 7
8 KB	0x400C_0000	0x400C_1FFF	0 1 2 3 4 5 6 7

表 27-3. 管理 RTC SLOW Memory 的 MPU

大小	边界地址		权限	
	低	高	PID = 0/1	PID DPORT_AHBLITE_MPU_TABLE_RTC_REG 位
8 KB	0x5000_0000	0x5000_1FFF	读/写	2 3 4 5 6 7 0 1 2 3 4 5

寄存器 RTC_CNTL_RTC_PID_CONFIG_REG 是 RTC 外设的一部分，只能由 PID 为 0 的进程修改; 寄存器 DPORT _AHBLITE_MPU_TABLE_RTC_REG 是一个 Dport 寄存器，可以由 PID 为 0 或 1 的进程修改。

管理 SRAM0 和 SRAM2 高 128 KB 的 MMU

SRAM0 的高 128 KB 和 SRAM 的高 128 KB 都由 MMU 管理。这些 MMU 不仅可以和 MPU 一样允许或禁止进程访问它们管理的存储器范围，而且还能够将 CPU 读取或写入的地址（虚地址）转换到存储器中可能不同的地址（实地址）。

为了实现这一点，片上 RAM MMU 将其管理的存储器范围划分为 16 页。页大小可配置为 8 KB，4 KB 和 2 KB。当页大小为 8 KB 时，16 页包含了整个 128 KB 的存储器空间; 当页大小为 4 KB 或 2 KB 时，存储器空间的末端将有分别为 64 KB 或 96 KB 区域不受 MPU 管理。类似于虚地址和实地址，页也可分为虚地址页和实地址页两种：MMU 可以将虚地址页内的地址转换为实地址页内的地址。

对于 PID 为 0 和 1 的进程，虚地址页与实地址页的地址映射是一对一的。即对某个虚地址页的读取或写入总被转换为对完全相同的实地址页的读取或写入。这允许在 PID 0 和/或 1 下运行的操作系统总是能够访问整个物理存储器范围。

但是对于 PID 为 2 到 7 的进程，MMU 可以基于每个 PID，重新配置每个虚地址页，使其映射到不同的实地址页。因此，对虚地址页内的偏移量的读取和写入可被转换为对不同实地址页内的相同偏移量的读取和写入。如图 27-1 所示，CPU（运行 PID 为 2 到 7 的进程）尝试访问存储器地址 0x3FFC_2345。该地址在虚地址页 1 的存储器区域内，偏移量为 0x0345。MMU 被指示将该 PID 的进程对虚地址页 1 的访问转换为实地址页 2 的访问。因此，存储器访问被重新定向为访问虚拟地址页 2 中相同的偏移，实际访问的实地址为 0x3FFC_**4**345。以下示例中的页大小为 8 KB。

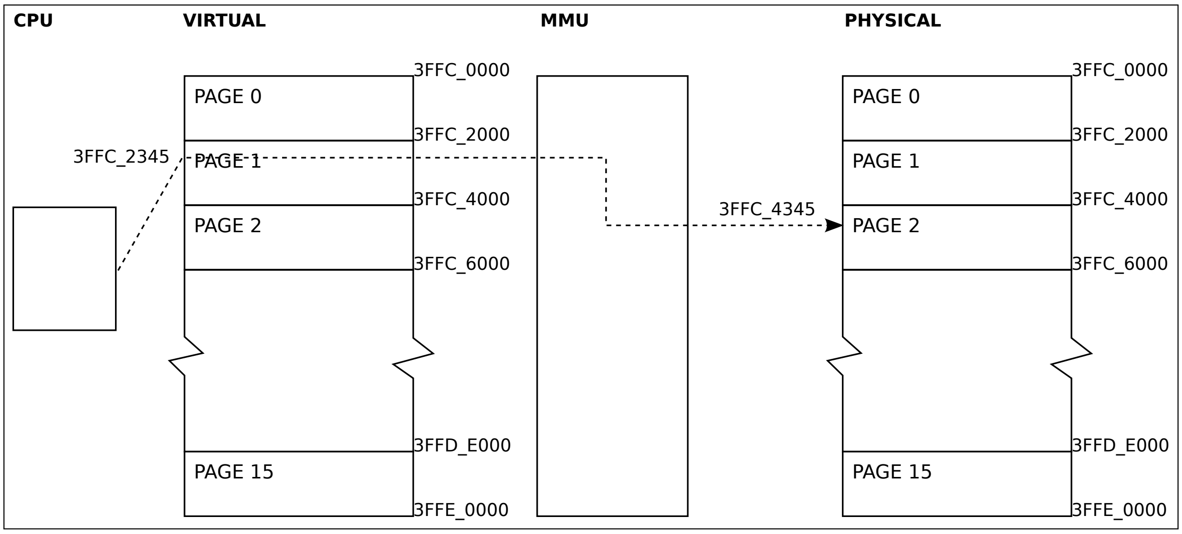


图 27-1. MMU 访问示例

表 27-4. 管理片上 SRAM 0 和 SRAM2 剩余 128 KB 的 MMU 页模式

DPORT_IMMU_PAGE_MODE	DPORT_DMMU_PAGE_MODE	页大小
0	0	8 KB
1	1	4 KB
2	2	2 KB

非 MMU 管理的存储器

对于 MMU 管理的 SRAM0 和 SRAM2 区域，页大小可配置为 8 KB，4 KB 和 2 KB。如表 27-4 所示，页大小通过寄存器 DPORT_IMMU_PAGE_MODE_REG 和 DPORT_DMMU_PAGE_MODE_REG 中的 DPORT_IMMU_PAGE_MODE (对于 SRAM0) 和 DPORT_DMMU_PAGE_MODE (对于 SRAM2) 位设置。因为每个存储器区域的页数量固定为 16，所以当页大小为 8 KB 时，这 16 页覆盖的存储器空间为 128 KB；当页大小为 4 KB 时，覆盖的存储器空间为 64 KB；当页大小为 2 KB 时，覆盖的存储器空间为 32 KB。

因此，对于 8 KB 页，整个存储器由 MMU 管理；但对于其它页大小，存储器的 128 KB 有一部分不受 MMU 管理。具体是，对于 4 KB 的页大小，不受 MMU 管理的区域是 0x4009_0000 到 0x4009_FFFF 和 0x3FFD_0000 到 0x3FFD_FFFF；对于 2 KB 的页大小，不受 MMU 管理的区域是 0x4008_8000 到 0x4009_FFFF 和 0x3FFC_8000 到 0x3FFD_FFFF。这些范围可由 PID 为 0 或 1 的进程读写；PID 为 2 到 7 的进程无法访问此存储器。

存储器空间中，页的布局是线性的，即，MMU 管理的 SRAM0 页 n 覆盖的地址范围为 $0x40080000 + (pagesize * n)$ 到 $0x40080000 + (pagesize * (n + 1) - 1)$；同理，MMU 管理的 SRAM2 页 n 覆盖的地址范围为 $0x3FFC0000 + (pagesize * n)$ 到 $0x3FFC0000 + (pagesize * (n + 1) - 1)$。表 27-5 和 27-6 描述了不同页大小模式下，由 MMU 管理的 SRAM0 和 SRAM2 所有页地址范围。

表 27-5. SRAM0 MMU 页边界地址

页	8 KB 页		4 KB 页		2 KB 页	
	低	高	低	高	低	高
0	40080000	40081FFF	40080000	40080FFF	40080000	400807FF
1	40082000	40083FFF	40081000	40081FFF	40080800	40080FFF
2	40084000	40085FFF	40082000	40082FFF	40081000	400817FF
3	40086000	40087FFF	40083000	40083FFF	40081800	40081FFF
4	40088000	40089FFF	40084000	40084FFF	40082000	400827FF
5	4008A000	4008BFFF	40085000	40085FFF	40082800	40082FFF
6	4008C000	4008DFFF	40086000	40086FFF	40083000	400837FF
7	4008E000	4008FFFF	40087000	40087FFF	40083800	40083FFF
8	40090000	40091FFF	40088000	40088FFF	40084000	400847FF
9	40092000	40093FFF	40089000	40089FFF	40084800	40084FFF
10	40094000	40095FFF	4008A000	4008AFFF	40085000	400857FF
11	40096000	40097FFF	4008B000	4008BFFF	40085800	40085FFF
12	40098000	40099FFF	4008C000	4008CFFF	40086000	400867FF
13	4009A000	4009BFFF	4008D000	4008DFFF	40086800	40086FFF
14	4009C000	4009DFFF	4008E000	4008EFFF	40087000	400877FF
15	4009E000	4009FFFF	4008F000	4008FFFF	40087800	40087FFF
剩余空间	-	-	40090000	4009FFFF	40088000	4009FFFF

表 27-6. SRAM2 MMU 页边界地址

页	8 KB 页		4 KB 页		2 KB 页	
	低	高	低	高	低	高
0	3FFC0000	3FFC1FFF	3FFC0000	3FFC0FFF	3FFC0000	3FFC07FF
1	3FFC2000	3FFC3FFF	3FFC1000	3FFC1FFF	3FFC0800	3FFC0FFF
2	3FFC4000	3FFC5FFF	3FFC2000	3FFC2FFF	3FFC1000	3FFC17FF
3	3FFC6000	3FFC7FFF	3FFC3000	3FFC3FFF	3FFC1800	3FFC1FFF
4	3FFC8000	3FFC9FFF	3FFC4000	3FFC4FFF	3FFC2000	3FFC27FF
5	3FFCA000	3FFCBFFF	3FFC5000	3FFC5FFF	3FFC2800	3FFC2FFF
6	3FFCC000	3FFCDFFF	3FFC6000	3FFC6FFF	3FFC3000	3FFC37FF
7	3FFCE000	3FFCFFFF	3FFC7000	3FFC7FFF	3FFC3800	3FFC3FFF
8	3FFD0000	3FFD1FFF	3FFC8000	3FFC8FFF	3FFC4000	3FFC47FF
9	3FFD2000	3FFD3FFF	3FFC9000	3FFC9FFF	3FFC4800	3FFC4FFF
10	3FFD4000	3FFD5FFF	3FFCA000	3FFCAFFF	3FFC5000	3FFC57FF
11	3FFD6000	3FFD7FFF	3FFCB000	3FFCBFFF	3FFC5800	3FFC5FFF
12	3FFD8000	3FFD9FFF	3FFCC000	3FFCCFFF	3FFC6000	3FFC67FF
13	3FFDA000	3FFDBFFF	3FFCD000	3FFCDFFF	3FFC6800	3FFC6FFF
14	3FFDC000	3FFDDFFF	3FFCE000	3FFCEFFF	3FFC7000	3FFC77FF
15	3FFDE000	3FFDFFFF	3FFCF000	3FFCFFFF	3FFC7800	3FFC7FFF
剩余空间	-	-	3FFD0000	3FFDFFFF	3FFC8000	3FFDFFFF

MMU 映射

SRAM0 MMU 和 SRAM2 MMU，通过一组 16 个寄存器控制访问权限和虚地址页到实地址页的映射。与其它大多数 MMU 不同的是，每个寄存器控制一个**实地址**页，而不是一个虚地址页。这些寄存器决定哪些 PID 的进程可以访问物理存储器以及哪个虚地址页映射到该寄存器控制的实地址页。表 27-7 描述了这些寄存器的位。需要注意的是，这些寄存器只控制 PID 为 2 到 7 的进程的访问权限；PID 为 0 或 1 的进程总是可以读写所有的页，并且没有虚地址到实地址的映射。即无论这些寄存器的设置如何，当 PID 为 0 或 1 的进程访问页 *x* 时，总是访问实地址页 *x*。这 16 个寄存器和控制页大小的寄存器 DPORT_IMMU_PAGE_MODE_REG 和 DPORT_DMMU_PAGE_MODE_REG 只能由 PID 为 0 或 1 的进程写入。

表 27-7. DPORT_DMMU_TABLE*n*_REG 和 DPORT_IMMU_TABLE*n*_REG

[6:4]	PID 2~ 7 的访问权限
0	PID 2 ~ 7 的进程都没有访问权限。
1	PID 2 ~ 7 的进程都有访问权限。
2	只有 PID 2 的进程有访问权限。
3	只有 PID 3 的进程有访问权限。
4	只有 PID 4 的进程有访问权限。
5	只有 PID 5 的进程有访问权限。
6	只有 PID 6 的进程有访问权限。
7	只有 PID 7 的进程有访问权限。

[3:0]	地址权限
0x00	虚拟页 0 可访问该物理页。
0x01	虚拟页 1 可访问该物理页。
0x02	虚拟页 2 可访问该物理页。
0x03	虚拟页 3 可访问该物理页。
0x04	虚拟页 4 可访问该物理页。
0x05	虚拟页 5 可访问该物理页。
0x06	虚拟页 6 可访问该物理页。
0x07	虚拟页 7 可访问该物理页。
0x08	虚拟页 8 可访问该物理页。
0x09	虚拟页 9 可访问该物理页。
0x10	虚拟页 10 可访问该物理页。
0x11	虚拟页 11 可访问该物理页。
0x12	虚拟页 12 可访问该物理页。
0x13	虚拟页 13 可访问该物理页。
0x14	虚拟页 14 可访问该物理页。
0x15	虚拟页 15 可访问该物理页。

SRAM0 和 SRAM2 MMU 的差异

由 SRAM0 MMU 管理的存储器通过处理器指令总线被访问，而处理器通过数据总线访问由 SRAM2 MMU 控制的存储器。因此，通常的做法是将代码存储在 SRAM0 的 MMU 页中，将数据存在 SRAM2 的 MMU 页中。因为通常情况下，这些 PID 的应用程序无需修改自己的代码，PID 为 2 到 7 进程对 SRAM0 的 MMU 页的访问是只读的。然而，这些应用程序必须能够修改其数据部分，因此允许它们读写位于 SRAM2 中的 MMU 页。如上所述，PID 为 0 或 1 的进程始终能够读写访问两个 SRAM0 和 SRAM2。

DMA MPU

应用程序可能需要配置 DMA 将数据直接发送到它们可以控制的外设，或者从这些外设中发送数据。通过访问 DMA，恶意进程可能将数据复制到其无法正常访问的存储器区域，或者将从其无法正常访问的存储器区域复制数据。为了防止这种情况发生，DMA MPU 可以用于禁止来自于具有敏感数据的存储器区域的 DMA 传输。

对于片上 SRAM1 和 SRAM2 存储器中的每个 8 KB 区域，MPU 通过 DPORT_AHB_MPU_TABLE_*n*_REG 寄存器中的某个位来允许或禁止 DMA 访问此区域。DMA MPU 仅使用这些位来决定是否可以开始 DMA 传输; 并不考虑进程的 PID。若 OS 以异构方式限制其进程，在上下文切换时，OS 需要根据要运行的进程来重新配置这些寄存器的值。

表 27-8 详细说明了对 SRAM1 和 SRAM2 的每个 8 KB 区域的访问进行管理的寄存器位。当寄存器位被置 1 时，DAM 可以读/写对应的 8 KB 存储器空间。当该位被清除时，DMA 对该地址空间的访问将被禁止。

表 27-8. 针对 DMA 的 MPU 设置

大小	边界地址		权限	
	低	高	寄存器	位
片上 SRAM 2				
8 KB	0x3FFA_E000	0x3FFA_FFFF	DPORT_AHB_MPU_TABLE_0_REG	0
8 KB	0x3FFB_0000	0x3FFB_1FFF	DPORT_AHB_MPU_TABLE_0_REG	1
8 KB	0x3FFB_2000	0x3FFB_3FFF	DPORT_AHB_MPU_TABLE_0_REG	2
8 KB	0x3FFB_4000	0x3FFB_5FFF	DPORT_AHB_MPU_TABLE_0_REG	3
8 KB	0x3FFB_6000	0x3FFB_7FFF	DPORT_AHB_MPU_TABLE_0_REG	4
8 KB	0x3FFB_8000	0x3FFB_9FFF	DPORT_AHB_MPU_TABLE_0_REG	5
8 KB	0x3FFB_A000	0x3FFB_BFFF	DPORT_AHB_MPU_TABLE_0_REG	6
8 KB	0x3FFB_C000	0x3FFB_DFFF	DPORT_AHB_MPU_TABLE_0_REG	7
8 KB	0x3FFB_E000	0x3FFB_FFFF	DPORT_AHB_MPU_TABLE_0_REG	8
8 KB	0x3FFC_0000	0x3FFC_1FFF	DPORT_AHB_MPU_TABLE_0_REG	9
8 KB	0x3FFC_2000	0x3FFC_3FFF	DPORT_AHB_MPU_TABLE_0_REG	10
8 KB	0x3FFC_4000	0x3FFC_5FFF	DPORT_AHB_MPU_TABLE_0_REG	11
8 KB	0x3FFC_6000	0x3FFC_7FFF	DPORT_AHB_MPU_TABLE_0_REG	12
8 KB	0x3FFC_8000	0x3FFC_9FFF	DPORT_AHB_MPU_TABLE_0_REG	13
8 KB	0x3FFC_A000	0x3FFC_BFFF	DPORT_AHB_MPU_TABLE_0_REG	14
8 KB	0x3FFC_C000	0x3FFC_DFFF	DPORT_AHB_MPU_TABLE_0_REG	15
8 KB	0x3FFC_E000	0x3FFC_FFFF	DPORT_AHB_MPU_TABLE_0_REG	16
8 KB	0x3FFD_0000	0x3FFD_1FFF	DPORT_AHB_MPU_TABLE_0_REG	17
8 KB	0x3FFD_2000	0x3FFD_3FFF	DPORT_AHB_MPU_TABLE_0_REG	18
8 KB	0x3FFD_4000	0x3FFD_5FFF	DPORT_AHB_MPU_TABLE_0_REG	19
8 KB	0x3FFD_6000	0x3FFD_7FFF	DPORT_AHB_MPU_TABLE_0_REG	20
8 KB	0x3FFD_8000	0x3FFD_9FFF	DPORT_AHB_MPU_TABLE_0_REG	21
8 KB	0x3FFD_A000	0x3FFD_BFFF	DPORT_AHB_MPU_TABLE_0_REG	22
8 KB	0x3FFD_C000	0x3FFD_DFFF	DPORT_AHB_MPU_TABLE_0_REG	23
8 KB	0x3FFD_E000	0x3FFD_FFFF	DPORT_AHB_MPU_TABLE_0_REG	24
片上 SRAM 1				
8 KB	0x3FFE_0000	0x3FFE_1FFF	DPORT_AHB_MPU_TABLE_0_REG	25
8 KB	0x3FFE_2000	0x3FFE_3FFF	DPORT_AHB_MPU_TABLE_0_REG	26
8 KB	0x3FFE_4000	0x3FFE_5FFF	DPORT_AHB_MPU_TABLE_0_REG	27
8 KB	0x3FFE_6000	0x3FFE_7FFF	DPORT_AHB_MPU_TABLE_0_REG	28
8 KB	0x3FFE_8000	0x3FFE_9FFF	DPORT_AHB_MPU_TABLE_0_REG	29
8 KB	0x3FFE_A000	0x3FFE_BFFF	DPORT_AHB_MPU_TABLE_0_REG	30
8 KB	0x3FFE_C000	0x3FFE_DFFF	DPORT_AHB_MPU_TABLE_0_REG	31
8 KB	0x3FFE_E000	0x3FFE_FFFF	DPORT_AHB_MPU_TABLE_1_REG	0
8 KB	0x3FFF_0000	0x3FFF_1FFF	DPORT_AHB_MPU_TABLE_1_REG	1
8 KB	0x3FFF_2000	0x3FFF_3FFF	DPORT_AHB_MPU_TABLE_1_REG	2
8 KB	0x3FFF_4000	0x3FFF_5FFF	DPORT_AHB_MPU_TABLE_1_REG	3
8 KB	0x3FFF_6000	0x3FFF_7FFF	DPORT_AHB_MPU_TABLE_1_REG	4
8 KB	0x3FFF_8000	0x3FFF_9FFF	DPORT_AHB_MPU_TABLE_1_REG	5
8 KB	0x3FFF_A000	0x3FFF_BFFF	DPORT_AHB_MPU_TABLE_1_REG	6

大小	边界地址		权限	
	低	高	寄存器	位
8 KB	0x3FFF_C000	0x3FFF_DFFF	DPORT_AHB_MPU_TABLE_1_REG	7
8 KB	0x3FFF_E000	0x3FFF_FFFF	DPORT_AHB_MPU_TABLE_1_REG	8

寄存器 DPROT_AHB_MPU_TABLE_0_REG 与 DPROT_AHB_MPU_TABLE_1_REG 位于 DPort 地址空间中。只有 PID 为 0 或 1 的进程可以修改这两个寄存器。

说明：

在硬件中，有 3 条指令总线分别对应 $VAddr_1$、$VAddr_2$ 和 $VAddr_3$，这 3 条总线可以同时发起读/取指访问，但是只有一个访问是真实的。如果未被屏蔽的总线超过了一个，那么所有 MMU 表项的 bit8 应该置 0。否则，当无效的 MMU 表项被某个访问使用时，即使此访问处没有程序，cache 也将被拖住。

27.3.2.2 片外存储器

对片外闪存和片外 SPI RAM 的访问通过 Cache 实现，并且由 MMU 管理。根据进程的 PID 以及运行该进程的 CPU，Cache MMU 可以实现不同的映射，做法类似于片上存储器 MMU。即，对于存储器的每个虚地址页，都有对应寄存器详细说明该虚地址页应映射到哪一个实地址页。但管理片上存储器的 MMU 和 Cache MMU 之间存在差异。首先，Cache MMU 具有固定的页面大小（片外闪存页大小为 64 KB，片外 RAM 页大小为 32 KB）；其次，Cache MMU 具有用于每个 PID 和处理器内核的显式映射表，不通过 MMU 配置项来控制访问权限。在下文中，MMU 映射配置寄存器将被统称为 “配置项”。这些寄存器只能由 PID 为 0 或 1 的进程访问；PID 为 2 到 7 的进程必须通过 PID 为 0 或 1 的进程来改变它们的 MMU 设置。

如上所述，MMU 配置项用于将对存储器虚地址页的访问映射到对存储器实地址页的访问。MMU 控制虚地址空间的五个区域，详见表 27-9。地址范围 $VAddr_1$ 到 $VAddr_4$ 用于访问片外闪存，$VAddr_{RAM}$ 用于访问片外 RAM。注意 $VAddr_4$ 是 $VAddr_0$ 的子集。

表 27-9. 片外存储器的虚地址

地址	大小	边界地址		页数量
		低	高	
$VAddr_0$	4 MB	0x3F40_0000	0x3F7F_FFFF	64
$VAddr_1$	4 MB	0x4000_0000	0x403F_FFFF	64*
$VAddr_2$	4 MB	0x4040_0000	0x407F_FFFF	64
$VAddr_3$	4 MB	0x4080_0000	0x40BF_FFFF	64
$VAddr_4$	1 MB	0x3F40_0000	0x3F4F_FFFF	16
$VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF	128

* 此处为了表述方便将地址范围写作 0x4000_0000 ~ 0x403F_FFFF 这样一个完整的 4 MB 地址空间。但其中部分地址范围无法访问。地址范围 0x4000_0000 ~ 0x400C_1FFF 只访问片上存储器。即 $VAddr_1$ 的某些配置项不会被使用。

片外闪存

表 27-10 和 27-11 详细描述了配置项号，虚拟存储器范围和 PID 的关系。这两个表格列出了每个存储器区域和 PID 组合所对应的管理映射的第一个 MMU 配置项。表格中的数字是指管理第一个地址页的 MMU 配置项；“数量” 一列表示页的数量，此为对应的存储器地址范围占用的页数量。

这两个表本质上是相同的，区别在于 APP_CPU 的配置项号比对应的 PRO_CPU 配置项号大 2048。注意，地址范围 $VAddr_0$ 和 $VAddr_1$ 只能由 PID 为 0 或 1 的进程访问，$VAddr_4$ 只能由 PID 为 2 到 7 的进程访问。

表 27-10. PRO_CPU 的 MMU 配置项号

地址	数量	PID 对应的第一个 MMU 配置项						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	0	-	-	-	-	-	-
$VAddr_1$	64	64	-	-	-	-	-	-
$VAddr_2$	64	128	256	384	512	640	768	896
$VAddr_3$	64	192	320	448	576	704	832	960
$VAddr_4$	16	-	1056	1072	1088	1104	1120	1136

表 27-11. APP_CPU 的 MMU 配置项号

地址	数量	PID 对应的第一个 MMU 配置项						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	2048	-	-	-	-	-	-
$VAddr_1$	64	2112	-	-	-	-	-	-
$VAddr_2$	64	2176	2304	2432	2560	2688	2816	2944
$VAddr_3$	64	2240	2368	2496	2624	2752	2880	3008
$VAddr_4$	16	-	3104	3120	3136	3152	3168	3184

如以上表格所示，虚地址 $VAddr_1$ 只能由 PID 为 0 或 1 的进程使用。为此专门有一种模式使得 PID 为 2 到 7 的进程能够通过地址 $VAddr_1$ 读取片外闪存。当寄存器 DPORT_PRO_CACHE_CTRL_REG 中的 DPORT_PRO_SINGLE_IRAM_ENA 位置 1 时，MMU 进入此特殊模式使得 PRO_CPU 访问存储器。同理，当寄存器 DPORT_APP_ CACHE_CTRL_REG 中的 DPORT_APP_SINGLE_IRAM_ENA 位为 1 时，APP_CPU 在此种模式下访问存储器。在这种模式下，MMU 的每个配置项所支持的进程和虚地址页不同。具体参考表 27-12 和表 27-13。如这些表格所示，在此特殊模式下，不能使用 $VAddr_2$ 和 $VAddr_3$ 访问片外闪存。

表 27-12. PRO_CPU 的 MMU 配置项号（特殊模式）

地址	数量	PID 对应的第一个 MMU 配置项						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	0	-	-	-	-	-	-
$VAddr_1$	64	64	256	384	512	640	768	896
$VAddr_2$	64	-	-	-	-	-	-	-
$VAddr_3$	64	-	-	-	-	-	-	-
$VAddr_4$	16	-	1056	1072	1088	1104	1120	1136

表 27-13. APP_CPU 的 MMU 配置项号（特殊模式）

地址	数量	PID 对应的第一个 MMU 配置项						
		0/1	2	3	4	5	6	7
$VAddr_0$	64	2048	-	-	-	-	-	-
$VAddr_1$	64	2112	2304	2432	2560	2688	2816	2944
$VAddr_2$	64	-	-	-	-	-	-	-
$VAddr_3$	64	-	-	-	-	-	-	-
$VAddr_4$	16	-	3104	3120	3136	3152	3168	3184

MMU 的每个配置项将 CPU 进程的虚地址页映射到实地址页。每个配置项位宽为 32。其中，0 到 7 号位表示由虚地址页映射去的实地址页。MMU 配置项有效时必须清除 8 号位。8 号位置 1 的配置项不会将任何实地址映射到虚地址。10 号位到 32 号位不使用，应写为零。由于 MMU 项中有 8 个地址位，片外闪存的页大小为 64 KB，因此支持的最大片外闪存 256 * 64 KB = 16 MB。

示例

示例 1：PID 为 1 的 PRO_CPU 进程需要通过虚地址 0x3F70_2375 读取片外闪存地址 0x07_2375。MMU 不处于特殊模式。

- 根据表 27-9，虚地址 0x3F70_2375 位于 $VAddr_0$ 的 0x30 号页。
- 根据表 27-10，PID 为 0/1 时，对于 PRO_CPU，$VAddr_0$ 的 MMU 配置项从 0 开始。
- 被更改的 MMU 配置项为 0 + 0x30 = 0x30。
- 地址 0x07_2375 位于 7 号 64 KB 页上。
- MMU 配置项 0x30 需要设置为 7，并通过将 8 号位置为 0 来标记为有效，因此，将 0x007 写入 MMU 配置项 0x30。

示例 2：PID 为 4 的 APP_CPU 进程需要通过虚地址 0x4044_048C 读取片外闪存地址 0x44_048C。MMU 不处于特殊模式。

- 根据表 27-9，虚地址 0x4044_048C 位于 $VAddr_2$ 的 0x4 号页。
- 根据表 27-11，PID 为 4 时，对于 APP_CPU，$VAddr_2$ 的 MMU 配置项从 2560 开始。
- 被更改的 MMU 配置项是 2560 + 0x4 = 2564。
- 地址 0x44_048C 位于 0x44 号 64 KB 页上。
- MMU 配置项 2564 需要设置为 0x44 并且通过将 8 号位置为 0 来标记为有效，因此，将 0x044 写入 MMU 配置项 2564。

片外 RAM

在 PRO_CPU 和 APP_CPU 上运行的进程可以通过缓存读写片外 SRAM，读写地址为虚地址范围 $VAddr_{RAM}$，即 0x3F80_0000 ~ 0x3FBF_FFFF。与闪存 MMU 一样，地址空间和物理存储器被分成页。对于片外 RAM MMU，页大小为 32 KB，并且 MMU 能够将 256 个实地址页映射到虚地址空间，允许映射 32 KB * 256 = 8 MB 的片外 RAM 实地址。

该地址范围的虚地址页映射取决于 MMU 所处的模式：低-高模式，偶-奇模式或正常模式。在所有情况下，寄存器 DPORT_PRO_CACHE_CTRL_REG 中的 DPORT_PRO_DRAM_HL 位和 DPORT_PRO_DRAM_SPLIT 位，

DPORT _APP_CACHE_CTRL_REG 中的 DPORT_APP_DRAM_HL 位和 DPORT_APP_DRAM_SPLIT 位共同决定片外
SRAM 的虚地址模式，具体参考表 27-14。如果需要 PRO_CPU 和 APP_CPU 的映射不同，则应选择正常模式，因为它是唯一可以达到此要求的模式。如果允许 PRO_CPU 和 APP_CPU 共享相同的映射，使用高-低或偶-奇模式可以在两个 CPU 频繁访问存储器时提高速度。

APP_CPU Cache 关闭时，0x4007_8000 到 0x4007_FFFF 的区域用作正常的片上 RAM，各种 Cache 模式的可用性有所变化。正常模式下，可以在 PRO_CPU 访问片外 RAM 的同时保持 Cache 正常运行，但 APP_CPU 无法访问片外 RAM。高-低模式下，两个 CPU 都可以使用片外 RAM，但只能使用 0x3F80_0000 到 0x3F9F_FFFF 的 2 MB 存储器虚地址。不建议在 APP_CPU Cache 区域关闭的情况下使用奇-偶模式。

表 27-14. 片外 SRAM 的虚拟地址模式

模式	DPORT_PRO_DRAM_HL DPORT_APP_DRAM_HL	DPORT_PRO_DRAM_SPLIT DPORT_APP_DRAM_SPLIT
低-高	1	0
偶-奇	0	1
正常	0	0

在正常模式下，两个 CPU 的虚地址页到实地址页的映射可以是不同的。通过 $^{L}VAddr_{RAM}$ 的 MMU 配置项设置 PRO_CPU 的页映射；通过 $^{R}VAddr_{RAM}$ 的 MMU 配置项设置 APP_CPU 的页映射。在这一模式下，$^{L}VAddr$ and $^{R}VAddr$ 的 128 页都被使用，从而能够映射最大为 8 MB 的内存。其中 4 MB 映射到 PRO_CPU 地址，4 MB 映射到 APP_CPU 地址，如表 27-15 所示。

表 27-15. 片外 SRAM 的虚地址（正常模式）

虚拟地址	大小	PRO_CPU 地址	
		低	高
$^{L}VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF
虚拟地址	大小	APP_CPU 地址	
		低	高
$^{R}VAddr_{RAM}$	4 MB	0x3F80_0000	0x3FBF_FFFF

在低-高模式下，PRO_CPU 以及 APP_CPU 使用相同的映射配置项。在这种模式下，$^{L}VAddr_{RAM}$ 用于虚地址空间的低 2 MB，而 $^{R}VAddr_{RAM}$ 用于虚地址空间的高 2 MB。这也意味着 $^{L}VAddr_{RAM}$ 的高 64 个 MMU 配置项以及 $^{R}VAddr_{RAM}$ 的低 64 个配置项未使用。表 27-16 详细描述了这些地址范围。

表 27-16. 片外 SRAM 的虚地址（低-高模式）

虚拟地址	大小	PRO_CPU/APP_CPU 地址	
		低	高
$^{L}VAddr_{RAM}$	2 MB	0x3F80_0000	0x3F9F_FFFF
$^{R}VAddr_{RAM}$	2 MB	0x3FA0_0000	0x3FBF_FFFF

在偶-奇存储器中，VRAM 被拆分为 32 字节的块。偶数块通过 MMU 配置项分散到 $^{L}VAddr_{RAM}$，奇数块通过 MMU 配置项分散到 $^{R}VAddr_{RAM}$。一般来说，$^{L}VAddr_{RAM}$ 和 $^{R}VAddr_{RAM}$ 的 MMU 配置项将被设置为相同的值，因此虚地址页可以映射到物理存储器的一个连续区域。表 27-17 详细说明了这种模式。

表 27-17. 片外 SRAM 的虚地址（偶-奇模式）

虚拟地址	大小	PRO_CPU/APP_CPU 地址	
		低	高
$^{L}VAddr_{RAM}$	32 字节	0x3F80_0000	0x3F80_001F
$^{R}VAddr_{RAM}$	32 字节	0x3F80_0020	0x3F80_003F
$^{L}VAddr_{RAM}$	32 字节	0x3F80_0040	0x3F80_005F
$^{R}VAddr_{RAM}$	32 字节	0x3F80_0060	0x3F80_007F
...			
$^{L}VAddr_{RAM}$	32 字节	0x3FBF_FFC0	0x3FBF_FFDF
$^{R}VAddr_{RAM}$	32 字节	0x3FBF_FFE0	0x3FBF_FFFF

片外 RAM MMU 配置项的位配置与闪存相同：配置项为 32 位寄存器，使用低 9 位。0 号位到 7 号位包含配置项需要映射其对应虚地址页的实地址页。如果配置项有效，则 8 号位被清除；如果配置项无效，则 8 号位置 1。表 27-18 详细说明 $^{L}VAddr_{RAM}$ 和 $^{R}VAddr_{RAM}$ 对应所有 PID 的第一个 MMU 配置项号。

表 27-18. 片外 RAM 的 MMU 配置项号

地址	页数量	PID 对应的第一个 MMU 配置项						
		0/1	2	3	4	5	6	7
$^{L}VAddr_{RAM}$	128	1152	1280	1408	1536	1664	1792	1920
$^{R}VAddr_{RAM}$	128	3200	3328	3456	3584	3712	3840	3968

示例

示例 1：PRO_CPU 上运行的 PID 为 7 的进程需要通过虚地址 0x3FA7_2375 读取或写入片外 RAM 地址 0x7F_A375。MMU 处于低-高模式。

- 根据表 27-9，虚地址 0x3FA7_2375 位于 $VAddr_{RAM}$ 的 0x4E 号 32 KB 页上。
- 根据表 27-16，虚地址 0x3FA7_2375 由 $^{R}VAddr_{RAM}$ 管理。
- 根据表 27-18 对于 PRO_CPU 上运行的 PID 为 7 的进程，$^{R}VAddr_{RAM}$ 对应的 MMU 配置项始于 3968。
- 被修改的 MMU 配置项为 3968 + 0x4E = 4046。
- 地址 0x7F_A375 位于 255 号 32 KB 页上。
- MMU 配置项 4046 需要被置为 255，并通过清除 8 号位来标记为有效，因此，将 0x0FF 写入 MMU 配置项 4046。

示例 2：APP_CPU 上运行的 PID 为 5 的进程需要从虚地址 0x3F85_5805 开始读取或写入片外 RAM 地址范围 0x55_5805 到 0x55_5823。MMU 处于奇-偶模式。

- 根据表 27-9，虚地址 0x3F85_5805 位于 $VAddr_{RAM}$ 的 0x0A 号 32 KB 页上。
- 根据表 27-17，要读取或写入的地址范围包含了 $^{R}VAddr_{RAM}$ 和 $^{L}VAddr_{RAM}$ 中的各 32 字节的区域。
- 根据表 27-18，对于 PID 5，$^{L}VAddr_{RAM}$ 的 MMU 配置项始于 1664。
- 根据表 27-18，对于 PID 5，$^{R}VAddr_{RAM}$ 的 MMU 配置项始于 3712。
- 被修改的 MMU 配置项为 1664 + 0x0A = 1674 和 3712 + 0x0A = 3722。
- 地址范围 0x55_5805 到 0x55_5823 位于 0xAA 号 32 KB 页上。

- MMU 配置项 1674 和 3722 需要被设置为 0xAA，并且通过将 8 号位置为 0 来标记为有效，因此，将 0x0AA 写入 MMU 配置项 1674 和 3722。该映射适用于 PRO_CPU 以及 APP_CPU。

示例 3：PRO_CPU 上运行的 PID 为 1 的进程和 APP_CPU 上运行的 PID 为 1 的进程需要使用虚地址 0x3F80_0876 读取或写入片外 RAM。PRO_CPU 需要该虚地址来访问实地址 0x10_0876，而 APP_CPU 需要通过该虚地址访问实地址 0x20_0876。MMU 处于正常模式。

- 根据表 27-9，虚拟地址 0x3F80_0876 位于 $VAddr_{RAM}$ 的 0 号 32 KB 页上。
- 根据表 27-18, 对于 PRO_CPU 上运行的 PID 为 1 的进程，MMU 配置项始于 1152。
- 根据表 27-18, 对于 APP_CPU 上运行的 PID 为 1 的进程，MMU 配置项始于 3200。
- 对于 PRO_CPU，被修改的 MMU 配置项为 1152 + 0 = 1152，对于 APP_CPU，被修改的 MMU 配置项为 3200 + 0 = 3200。
- 地址 0x10_0876 位于 0x20 号 32 KB 页上。
- 地址 0x20_0876 位于 0x40 号 32 KB 页上。
- 对于 PRO_CPU, MMU 配置项 1152 需要被置为 0x20 并且通过清除 8 号位来标记有效，所以将 0x020 写入 MMU 配置项 1152。
- 对于 APP_CPU，MMU 配置项 3200 需要被置为 0x40 并且通过清除 8 号位来标记有效，所以将 0x040 写入 MMU 配置项 3200。
- 这样 PRO_CPU 和 APP_CPU 就可以通过相同的虚地址访问不同的物理内存区域。

27.3.2.3 外设

外设 MPU 管理 39 个外设模块。根据每个外设模块，可以通过配置 MMU，允许只有特定 PID 的进程访问外设。配置 MMU 的寄存器详细信息请参考表 27-19。

表 27-19. 管理外设的 MPU

外设	权限	
	PID = 0/1	PID = 2 ~ 7
DPort Register	访问	禁止
AES Accelerator	访问	禁止
RSA Accelerator	访问	禁止
SHA Accelerator	访问	禁止
Secure Boot	访问	禁止
Cache MMU Table	访问	禁止
PID Controller	访问	禁止
UART0	访问	DPORT_AHBLITE_MPU_TABLE_UART_REG
SPI1	访问	DPORT_AHBLITE_MPU_TABLE_SPI1_REG
SPI0	访问	DPORT_AHBLITE_MPU_TABLE_SPI0_REG
GPIO	访问	DPORT_AHBLITE_MPU_TABLE_GPIO_REG
RTC	访问	DPORT_AHBLITE_MPU_TABLE_RTC_REG
IO MUX	访问	DPORT_AHBLITE_MPU_TABLE_IO_MUX_REG
SDIO Slave	访问	DPORT_AHBLITE_MPU_TABLE_HINF_REG
UDMA1	访问	DPORT_AHBLITE_MPU_TABLE_UHCI1_REG
I2S0	访问	DPORT_AHBLITE_MPU_TABLE_I2S0_REG
UART1	访问	DPORT_AHBLITE_MPU_TABLE_UART1_REG

外设	权限	
	PID = 0/1	PID = 2 ~ 7
I2C0	访问	DPORT_AHBLITE_MPU_TABLE_I2C_EXT0_REG
UDMA0	访问	DPORT_AHBLITE_MPU_TABLE_UHCI0_REG
SDIO Slave	访问	DPORT_AHBLITE_MPU_TABLE_SLCHOST_REG
RMT	访问	DPORT_AHBLITE_MPU_TABLE_RMT_REG
PCNT	访问	DPORT_AHBLITE_MPU_TABLE_PCNT_REG
SDIO Slave	访问	DPORT_AHBLITE_MPU_TABLE_SLC_REG
LED PWM	访问	DPORT_AHBLITE_MPU_TABLE_LEDC_REG
Efuse Controller	访问	DPORT_AHBLITE_MPU_TABLE_EFUSE_REG
Flash Encryption	访问	DPORT_AHBLITE_MPU_TABLE_SPI_ENCRYPT_REG
PWM0	访问	DPORT_AHBLITE_MPU_TABLE_PWM0_REG
TIMG0	访问	DPORT_AHBLITE_MPU_TABLE_TIMERGROUP_REG
TIMG1	访问	DPORT_AHBLITE_MPU_TABLE_TIMERGROUP1_REG
SPI2	访问	DPORT_AHBLITE_MPU_TABLE_SPI2_REG
SPI3	访问	DPORT_AHBLITE_MPU_TABLE_SPI3_REG
SYSCON	访问	DPORT_AHBLITE_MPU_TABLE_APB_CTRL_REG
I2C1	访问	DPORT_AHBLITE_MPU_TABLE_I2C_EXT1_REG
SDMMC	访问	DPORT_AHBLITE_MPU_TABLE_SDIO_HOST_REG
EMAC	访问	DPORT_AHBLITE_MPU_TABLE_EMAC_REG
PWM1	访问	DPORT_AHBLITE_MPU_TABLE_PWM1_REG
I2S1	访问	DPORT_AHBLITE_MPU_TABLE_I2S1_REG
UART2	访问	DPORT_AHBLITE_MPU_TABLE_UART2_REG
RNG	访问	DPORT_AHBLITE_MPU_TABLE_PWR_REG

寄存器 DPORT_AHBLITE_MPU_TABLE_X_REG 的每个位决定每个进程是否可以访问寄存器管理的外设。详细信息请参考表 27-20。当寄存器 DPORT_AHBLITE_MPU_TABLE_X_REG 的某个位置 1 时，这意味着具有相应 PID 的进程可以访问此寄存器的相应外设。否则，进程无法访问相应的外设。

表 27-20. DPORT_AHBLITE_MPU_TABLE_X_REG

PID	2 3 4 5 6 7
DPORT_AHBLITE_MPU_TABLE_X_REG 位	0 1 2 3 4 5

所有的 DPORT_AHBLITE_MPU_TABLE_X_REG 寄存器位于外设 DPort 寄存器中。只有 PID 为 0/1 的进程可以更改这些寄存器。

28 PID 控制器 (PID)

28.1 概述

ESP32 双核芯片可以同时处理多个线程。PID 控制器在进程切换的过程中，辅助完成进程号 PID 的切换。此外，PID 控制器还可以通过记录 CPU 处理中断的状态来管理嵌套中断。因此，PID 控制器可以使得用户在应用中更有效地管理进程切换和嵌套中断。

28.2 主要特性

PID 控制器有以下特性：

- 管理进程优先级
- 切换进程号 PID
- 记录中断信息
- 管理中断嵌套

28.3 功能描述

CPU 具有 8 个进程，进程号 PID 分别为 0 ~ 7。这 8 个进程中，PID 为 0/1 的进程比 PID 为 2 ~ 7 的进程拥有更多的权限。

CPU 在两种情形下会进行进程切换。

- 当中断发生，CPU 从中断向量入口地址取指的时候。无论中断发生之前正在运行的是哪个进程，系统此时都将中断取指视为 PID 为 0 的进程。
- 当前进程主动切换进程时。能够主动进行进程切换的进程一定是 PID 为 0/1 的高权限进程。

28.3.1 中断识别

CPU 一共有 Level 1、Level 2、Level 3、Level 4、Level 5、Level 6（Debug）、NMI 七个优先级的中断。每一级中断有一个中断向量入口地址。PID 控制器识别到 CPU 从中断向量入口地址取指的时候自动切换 PID 至 0。如果 CPU 只是对中断向量入口地址做数据访问，那么 PID 控制器不采取任何动作。

PID 控制器能够识别的中断优先级取决于寄存器 PIDCTRL_INTERRUPT_ENABLE_REG。寄存器 PIDCTRL_INTERRUPT_ENABLE_REG 中的某位为 1 时，当 CPU 从此位对应的中断向量入口地址取指时，PID 控制器将切换进程；若寄存器 PIDCTRL_INTERRUPT_ENABLE_REG 中的某位为 0 时，则 PID 控制器不采取任何动作。这七级中断各自的中断向量入口地址由寄存器 PIDCTRL_INTERRUPT_ADDR_1_REG ~ PIDCTRL_INTERRUPT_ADDR_7_REG 决定，详见表 28-1。

表 28-1. 中断向量入口地址

中断优先级	中断识别使能位 PIDCTRL_INTERRUPT_ENABLE_REG 位	中断向量入口地址
Level 1	1	PIDCTRL_INTERRUPT_ADDR_1_REG
Level 2	2	PIDCTRL_INTERRUPT_ADDR_2_REG
Level 3	3	PIDCTRL_INTERRUPT_ADDR_3_REG
Level 4	4	PIDCTRL_INTERRUPT_ADDR_4_REG
Level 5	5	PIDCTRL_INTERRUPT_ADDR_5_REG
Level 6 (Debug)	6	PIDCTRL_INTERRUPT_ADDR_6_REG
NMI	7	PIDCTRL_INTERRUPT_ADDR_7_REG

28.3.2 信息记录

当 PID 控制器识别到中断时，除了自动切换 PID 为 0，还会记录 3 条信息：

1. 当前中断的优先级
2. 系统的上一次中断状态
3. CPU 运行的上一个进程

PID 控制器会将当前发生的中断的优先级记录到寄存器 PIDCTRL_LEVEL_REG，详见表 28-2。

表 28-2. PIDCTRL_LEVEL_REG

寄存器值	系统当前中断状态
0	不处于中断中
1	处于 Level 1 中断
2	处于 Level 2 中断
3	处于 Level 3 中断
4	处于 Level 4 中断
5	处于 Level 5 中断
6	处于 Level 6 中断
7	处于 NMI 中断

PID 控制器还将当前中断发生之前的状态记录进寄存器 PIDCTRL_FROM_*n*_REG。寄存器 PIDCTRL_FROM_*n*_REG 的位宽为 7。其高 4 位表示此寄存器对应中断发生之前系统的中断状态。低 3 位表示此寄存器对应中断发生之前系统处于哪个进程。详见表 28-3。

表 28-3. PIDCTRL_FROM_*n*_REG

[6:3]	当前中断发生前系统的中断状态	[2:0]	当前中断发生前系统运行的进程
0	不处于中断	0	PID 为 0 的进程
1	处于 Level 1 中断	1	PID 为 1 的进程
2	处于 Level 2 中断	2	PID 为 2 的进程
3	处于 Level 3 中断	3	PID 为 3 的进程
4	处于 Level 4 中断	4	PID 为 4 的进程
5	处于 Level 5 中断	5	PID 为 5 的进程
6	处于 Level 6 中断	6	PID 为 6 的进程
7	处于 Level 7 中断	7	PID 为 7 的进程

PID 控制器拥有寄存器 PIDCTRL_FROM_1_REG ~ PIDCTRL_FROM_7_REG，它们分别对应 Level 1、Level 2、Level 3、Level 4、Level 5、Level 6（Debug）、NMI 七级中断。系统通过这些寄存器处理中断嵌套，如图 28-1 所示。

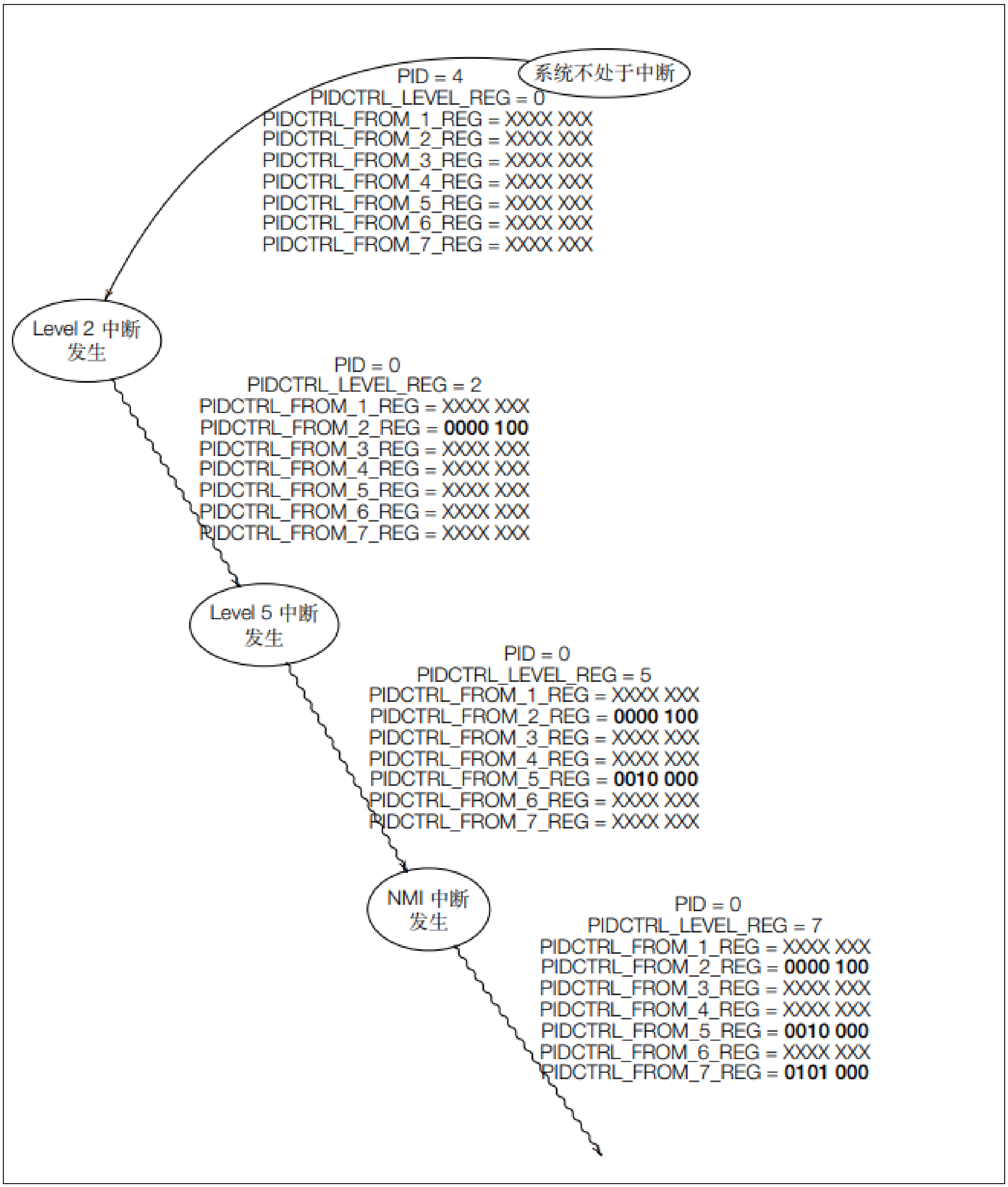


图 28-1. 中断嵌套

如果中断发生，但是由于寄存器 PIDCTRL_INTERRUPT_ENABLE_REG 的配置，而使得 PID 控制器没有识别到它，那么 PID 控制器不会做任何记录，也不会更改寄存器 PIDCTRL_LEVEL_REG、PIDCTRL_FROM_*n*_REG 的值。

28.3.3 进程主动切换进程

只有 PID 为 0/1 的进程可以主动切换进程。切换之后的进程可以是 PID 为 0 ~ 7 中的任何一个。进程主动切换进程的关键在于，从当前进程的最后一条指令跳转到新进程的第一条指令的时候，PID 要刚好从 0/1 变为新进程的 PID。

进程主动切换进程的软件流程为

1. 软件屏蔽除 NMI 中断之外的所有中断
2. 将寄存器 PIDCTRL_NMI_MASK_ENABLE_REG 置 1，生成 CPU NMI 中断屏蔽信号
3. 配置寄存器 PIDCTRL_PID_DELAY_REG、PIDCTRL_NMI_DELAY_REG
4. 配置寄存器 PIDCTRL_PID_NEW_REG
5. 配置寄存器 PIDCTRL_LEVEL_REG、PIDCTRL_FROM_*n*_REG
6. 将寄存器 PIDCTRL_PID_CONFIRM_REG、PIDCTRL_NMI_MASK_DISABLE_REG 置 1
7. 撤销对 NMI 中断之外的中断的屏蔽
8. 切换到新进程取指

虽然系统可以处理中断嵌套的情况，但 PID 为 0/1 的进程不应在进程切换时被新的中断打断，因此在步骤 1、步骤 2 中屏蔽了所有中断。

步骤 3 中配置的寄存器 PIDCTRL_PID_DELAY_REG、PIDCTRL_NMI_DELAY_REG 的值将会作用于步骤 6。

步骤 4 中配置的寄存器 PIDCTRL_PID_NEW_REG 的值将会在步骤 6 之后成为新的进程 PID。

如果当前处于嵌套中断中且要恢复到上一个中断，那么在步骤 5 中需要根据寄存器 *n* 中记录的信息恢复寄存器 PIDCTRL_LEVEL_REG。

步骤 6 中，将寄存器 PIDCTRL_PID_CONFIRM_REG、PIDCTRL_NMI_MASK_DISABLE_REG 置 1 后，PID 控制器并不会立即将 PID 切换为 PIDCTRL_PID_NEW_REG 中的值，也不会立即关闭 CPU NMI 中断屏蔽信号，而是会分别等待一定数量的时钟周期数后才会执行这两个任务。这两段等待时钟周期数即寄存器 PIDCTRL_PID_DELAY_REG、PIDCTRL_NMI_DELAY_REG 中的值。

在步骤 7 中还可以执行其他任务，只需要在步骤 3 中配置寄存器 PIDCTRL_PID_DELAY_REG、PIDCTRL_NMI_DELAY_REG 时包含这些任务的时间开销即可。

28.4 寄存器列表

名称	描述	地址	访问
PIDCTRL_INTERRUPT_ENABLE_REG	PID 中断识别使能位	0x3FF1F000	读 / 写
PIDCTRL_INTERRUPT_ADDR_1_REG	Level 1 中断向量入口地址	0x3FF1F004	读 / 写
PIDCTRL_INTERRUPT_ADDR_2_REG	Level 2 中断向量入口地址	0x3FF1F008	读 / 写
PIDCTRL_INTERRUPT_ADDR_3_REG	Level 3 中断向量入口地址	0x3FF1F00C	读 / 写
PIDCTRL_INTERRUPT_ADDR_4_REG	Level 4 中断向量入口地址	0x3FF1F010	读 / 写
PIDCTRL_INTERRUPT_ADDR_5_REG	Level 5 中断向量入口地址	0x3FF1F014	读 / 写
PIDCTRL_INTERRUPT_ADDR_6_REG	Level 6 中断向量入口地址	0x3FF1F018	读 / 写
PIDCTRL_INTERRUPT_ADDR_7_REG	NMI 中断向量入口地址	0x3FF1F01C	读 / 写
PIDCTRL_PID_DELAY_REG	新的 PID 生效前的延迟	0x3FF1F020	读 / 写
PIDCTRL_NMI_DELAY_REG	关闭 NMI 屏蔽信号前的延迟	0x3FF1F024	读 / 写
PIDCTRL_LEVEL_REG	当前中断优先级	0x3FF1F028	读 / 写
PIDCTRL_FROM_1_REG	Level 1 中断发生前的系统状态	0x3FF1F02C	读 / 写
PIDCTRL_FROM_2_REG	Level 2 中断发生前的系统状态	0x3FF1F030	读 / 写
PIDCTRL_FROM_3_REG	Level 3 中断发生前的系统状态	0x3FF1F034	读 / 写
PIDCTRL_FROM_4_REG	Level 4 中断发生前的系统状态	0x3FF1F038	读 / 写
PIDCTRL_FROM_5_REG	Level 5 中断发生前的系统状态	0x3FF1F03C	读 / 写
PIDCTRL_FROM_6_REG	Level 6 中断发生前的系统状态	0x3FF1F040	读 / 写
PIDCTRL_FROM_7_REG	NMI 中断发生前的系统状态	0x3FF1F044	读 / 写
PIDCTRL_PID_NEW_REG	配置新的 PID	0x3FF1F048	读 / 写
PIDCTRL_PID_CONFIRM_REG	确认新的 PID	0x3FF1F04C	只写
PIDCTRL_NMI_MASK_ENABLE_REG	NMI 中断屏蔽使能寄存器	0x3FF1F054	只写
PIDCTRL_NMI_MASK_DISABLE_REG	NMI 中断屏蔽关闭寄存器	0x3FF1F058	只写

28.5 寄存器

本小节括号中的地址均为相对于 PID Controller 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 28.4 寄存器列表。

Register 28.1. PIDCTRL_INTERRUPT_ENABLE_REG (0x000)

(reserved)	PIDCTRL_INTERRUPT_ENABLE	(reserved)	
31 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	Reset

PIDCTRL_INTERRUPT_ENABLE 该位用于使能中断的识别和处理。(读 / 写)

Register 28.2. PIDCTRL_INTERRUPT_ADDR_1_REG (0x004)

31 0	
0x040000340	Reset

PIDCTRL_INTERRUPT_ADDR_1_REG Level 1 中断向量入口地址。(读 / 写)

Register 28.3. PIDCTRL_INTERRUPT_ADDR_2_REG (0x008)

31 0	
0x040000180	Reset

PIDCTRL_INTERRUPT_ADDR_2_REG Level 2 中断向量入口地址。(读 / 写)

Register 28.4. PIDCTRL_INTERRUPT_ADDR_3_REG (0x00C)

31 0	
0x0400001C0	Reset

PIDCTRL_INTERRUPT_ADDR_3_REG Level 3 中断向量入口地址。(读 / 写)

Register 28.5. PIDCTRL_INTERRUPT_ADDR_4_REG (0x010)

31 0	
0x040000200	Reset

PIDCTRL_INTERRUPT_ADDR_4_REG Level 4 中断向量入口地址。(读 / 写)

Register 28.6. PIDCTRL_INTERRUPT_ADDR_5_REG (0x014)

31	0	
0x040000240		Reset

PIDCTRL_INTERRUPT_ADDR_5_REG Level 5 中断向量入口地址。（读 / 写）

Register 28.7. PIDCTRL_INTERRUPT_ADDR_6_REG (0x018)

31	0	
0x040000280		Reset

PIDCTRL_INTERRUPT_ADDR_6_REG Level 6 中断向量入口地址。（读 / 写）

Register 28.8. PIDCTRL_INTERRUPT_ADDR_7_REG (0x01C)

31	0	
0x0400002C0		Reset

PIDCTRL_INTERRUPT_ADDR_7_REG NMI 中断向量入口地址。（读 / 写）

Register 28.9. PIDCTRL_PID_DELAY_REG (0x020)

(reserved)		PIDCTRL_PID_DELAY		
31	12	11	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		20		Reset

PIDCTRL_PID_DELAY 新分配的 PID 生效前的延迟。（读 / 写）

Register 28.10. PIDCTRL_NMI_DELAY_REG (0x024)

(reserved)		PIDCTRL_NMI_DELAY		
31	12	11	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		16		Reset

PIDCTRL_NMI_DELAY 关闭 CPU NMI 中断屏蔽信号前的延迟。（读 / 写）

Register 28.11. PIDCTRL_LEVEL_REG (0x028)

(reserved)	PIDCTRL_CURRENT_STATUS	
31 4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

PIDCTRL_CURRENT_STATUS 系统当前状态。（读 / 写）

Register 28.12. PIDCTRL_FROM_*n*_REG (*n*: 1-7) (0x28+0x4*n)

(reserved)	PIDCTRL_PREVIOUS_STATUS_*n*	
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	Reset

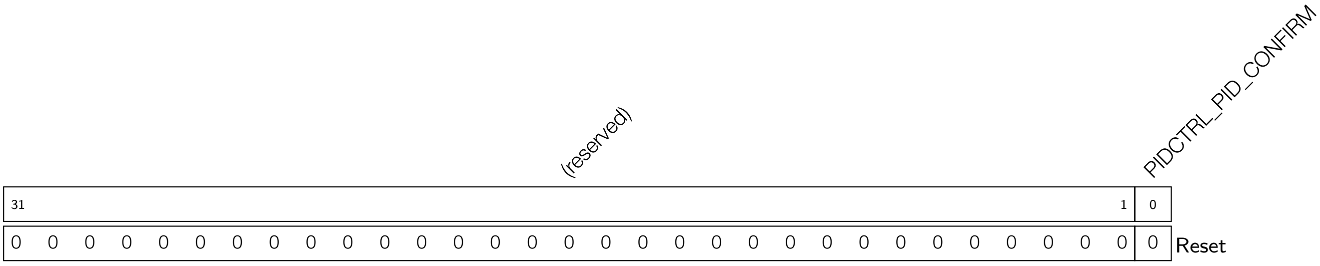
PIDCTRL_PREVIOUS_STATUS_*n* 任一 Level 1 至 Level 6，或 NMI 中断发生前的系统状态。（读 / 写）

Register 28.13. PIDCTRL_PID_NEW_REG (0x048)

(reserved)	PIDCTRL_PID_NEW	
31 3	2 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	Reset

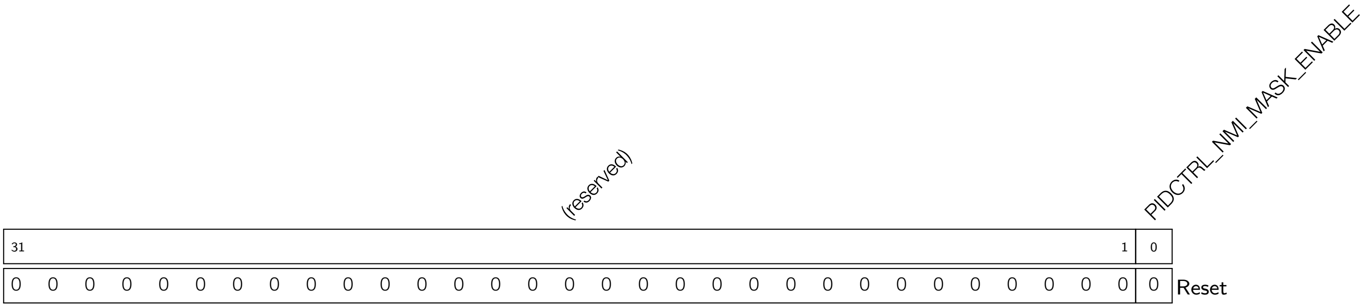
PIDCTRL_PID_NEW 新的 PID。（读 / 写）

Register 28.14. PIDCTRL_PID_CONFIRM_REG (0x04C)



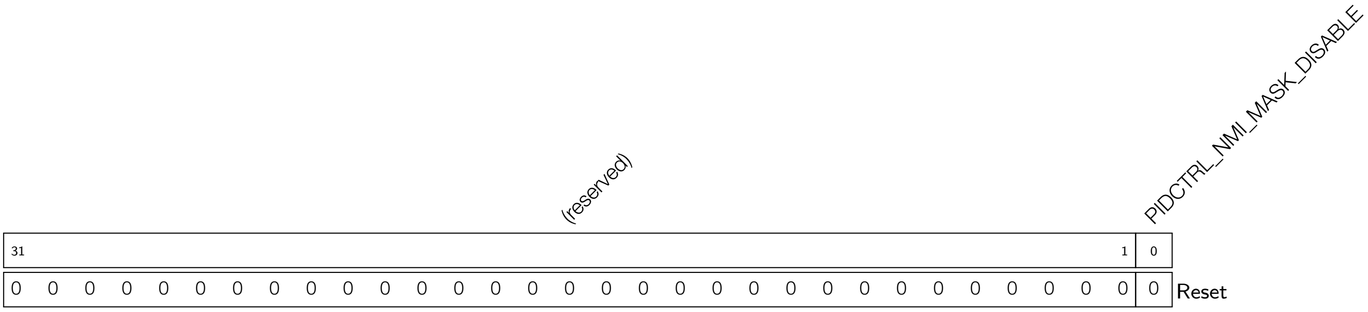
PIDCTRL_PID_CONFIRM 该位用于使新 PID 生效。（只写）

Register 28.15. PIDCTRL_NMI_MASK_ENABLE_REG (0x054)



PIDCTRL_NMI_MASK_ENABLE 使能 CPU NMI 中断屏蔽信号。（只写）

Register 28.16. PIDCTRL_NMI_MASK_DISABLE_REG (0x058)



PIDCTRL_NMI_MASK_DISABLE 关闭 CPU NMI 中断屏蔽信号。（只写）

29 片上传感器与模拟信号处理

29.1 概述

ESP32 集成电容式触摸传感器，最高支持 10 路输入。

ESP32 的模拟信号处理主要由 2 个逐次逼近模拟数字转换器 (Successive Approximation ADC, SAR ADC) 完成。系统专门内置了 5 个 ADC 专用控制器，可在转换模拟输入信号时支持高性能与低功耗 2 种模式，处理器的开销最低。

此外，ESP32 还可使用 2 个独立数字模拟转换器 (DAC)和 1 个余弦波形发生器生成模拟信号。

29.2 电容式触摸传感器

29.2.1 简介

触摸传感器系统主要由 3 个部分组成，从外到内依次为平面保护层、电极与基片，见图 29-1。当用户触碰保护层时，传感器系统的电容量会发生改变，继而生成 1 个可以反映本次触碰是否触发的二进制信号。

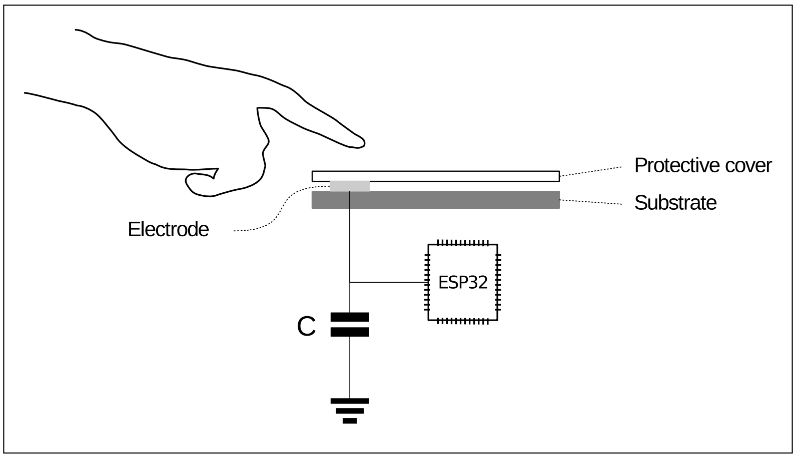


图 29-1. 触摸传感器

29.2.2 主要特性

- 最多支持 10 路电容触摸管脚/通用输入输出接口 (General Purpose Input and Output, GPIO)
- 触摸管脚可以组合使用，可覆盖更大触感区域或更多触感点
- 触摸管脚的传感由有限状态机 (FSM) 硬件控制，由软件或专用硬件计时器发起
- 触摸管脚是否受到触碰的信息可由以下方式获得：
    - 由软件直接检查触摸传感器的寄存器
    - 由触摸监测模块发起的中断信号判断
    - 由触摸监测模块上的 CPU 是否从 Deep-sleep 中唤醒判断
- 支持以下场景下的低功耗工作：
    - CPU 处于 Deep-sleep 节能模式，将在受到触碰后逐步唤醒
    - 触摸监测由超低功耗协处理器 (ULP coprocessor) 管理

ULP 用户程序可通过写入与检查特定寄存器，判断是否达到触碰阈值

说明：

ESP32 触摸传感器目前尚无法通过射频抗扰度测试系统 (CS) 认证，应用场景有所限制。

29.2.3 可用通用输入输出接口

全部 10 个可用通用输入输出接口的信息，请见表 29-1。

表 29-1. ESP32 电容式触摸传感器的管脚

触摸传感信号名	管脚
T0	GPIO4
T1	GPIO0
T2	GPIO2
T3	MTDO
T4	MTCK
T5	MTDI
T6	MTMS
T7	GPIO27
T8	32K_XN
T9	32K_XP

29.2.4 功能描述

触摸传感器的内部结构请见图 29-2，工作流程请见图 29-3。

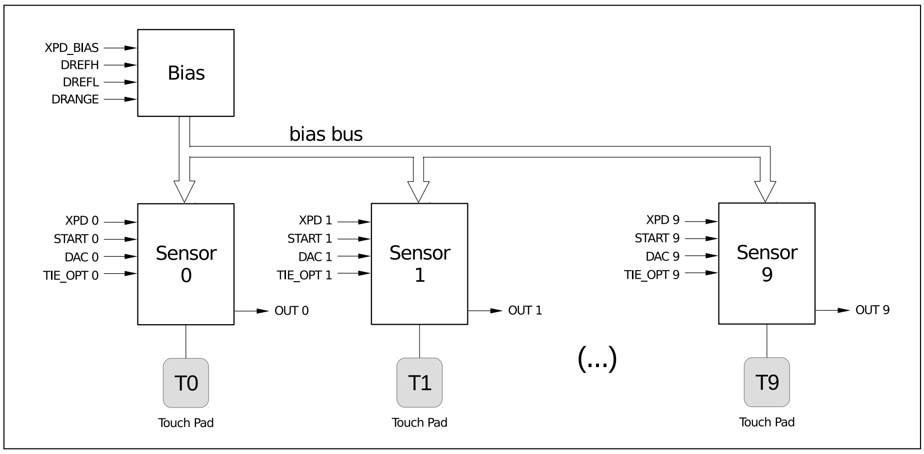


图 29-2. 触摸传感器的内部结构

触摸管脚的电容会进行周期性充放电。” 触摸管脚的内部电压” 代表充/放电电压在参考高值 (drefH) 与参考低值 (drefL) 之间的变化。在每次变化中，触摸传感器将生成一个输出脉冲 (OUT)。由于触摸管脚受到触碰（高电容）与未受到触碰（低电容）时的电压变化速率不同，我们可以通过统计同一时间间隔内出现的输出脉冲数量，判断触摸管脚是否受到触碰。可以通过 TIE_OPT 设置开始充/放电的初始电压电平。

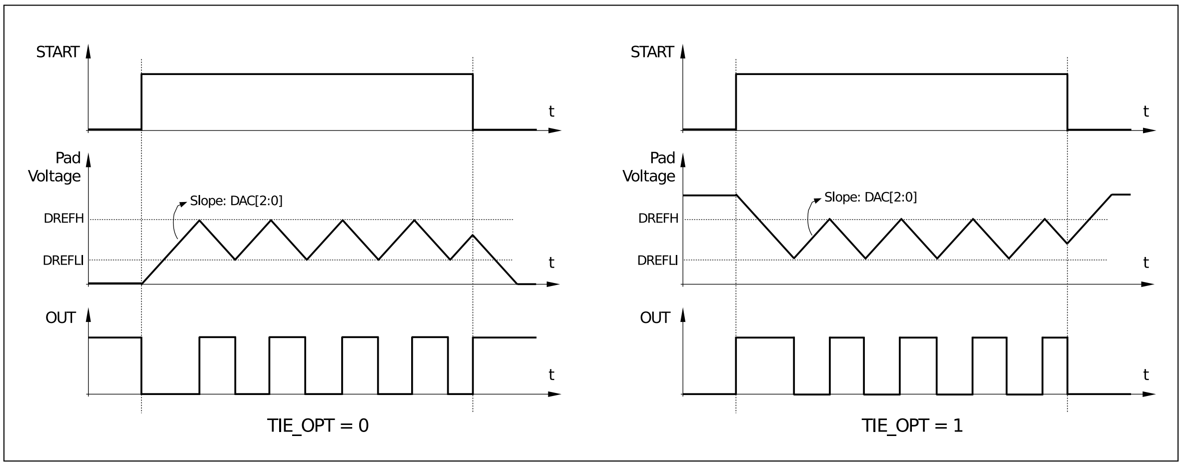


图 29-3. 触摸传感器的工作流程

29.2.5 触发传感器的状态机

有限状态机 (Finite-State Machine, FSM) 将执行 29.2.4 章节描述的序列检测。软件可通过专用寄存器操作 FSM。FSM 的内部结构可见图 29-4。

FSM 的功能包括：

- 接收软件或计时器发出的开始信号
  - 当 SENS_SAR_TOUCH_START_FORCE = 1 时，可通过设置 SENS_SAR_TOUCH_START_EN 发起一次性检测；
  - 当 SENS_SAR_TOUCH_START_FORCE = 0 时，可利用计时器实现周期性检测。

  触摸传感器在睡眠模式下也能工作。更多信息，请见功耗管理章节。可通过寄存器 SENS_SAR_TOUCH_SLEEP_CYCLES 设定检测周期。传感器由 RTC_FAST_CLK 控制，常见时钟频率为 8 MHz。更多信息，请见复位和时钟章节。

- 根据可调节时序，生成 XPD_TOUCH_BIAS / TOUCH_XPD / TOUCH_START
  在选择使能触摸管脚时，TOUCH_XPD / TOUCH_START 中的内容将被 10 位寄存器 SENS_SAR_TOUCH_PAD_WORKEN 遮掩。

- 计数 TOUCH0_OUT ~ TOUCH9_OUT 上的脉冲数量
  计数结果可见 SENS_SAR_TOUCH_MEAS_OUT*n*。全部 10 个触摸管脚可支持同时工作。

- 生成唤醒中断
  如果一个管脚的脉冲计数结果低于阈值，则 FSM 视该管脚被“触碰”。10 位寄存器 SENS_TOUCH_PAD_OUTEN1 & SENS_TOUCH_PAD_OUTEN2 可以将所有管脚定义为 2 组，即 SET1 & SET2。默认状态下，如果 SET1 中的任意管脚被“触碰”，即可生成唤醒中断，也可以配置为 SET1 和 SET2 中均有管脚被“触碰”时有效。

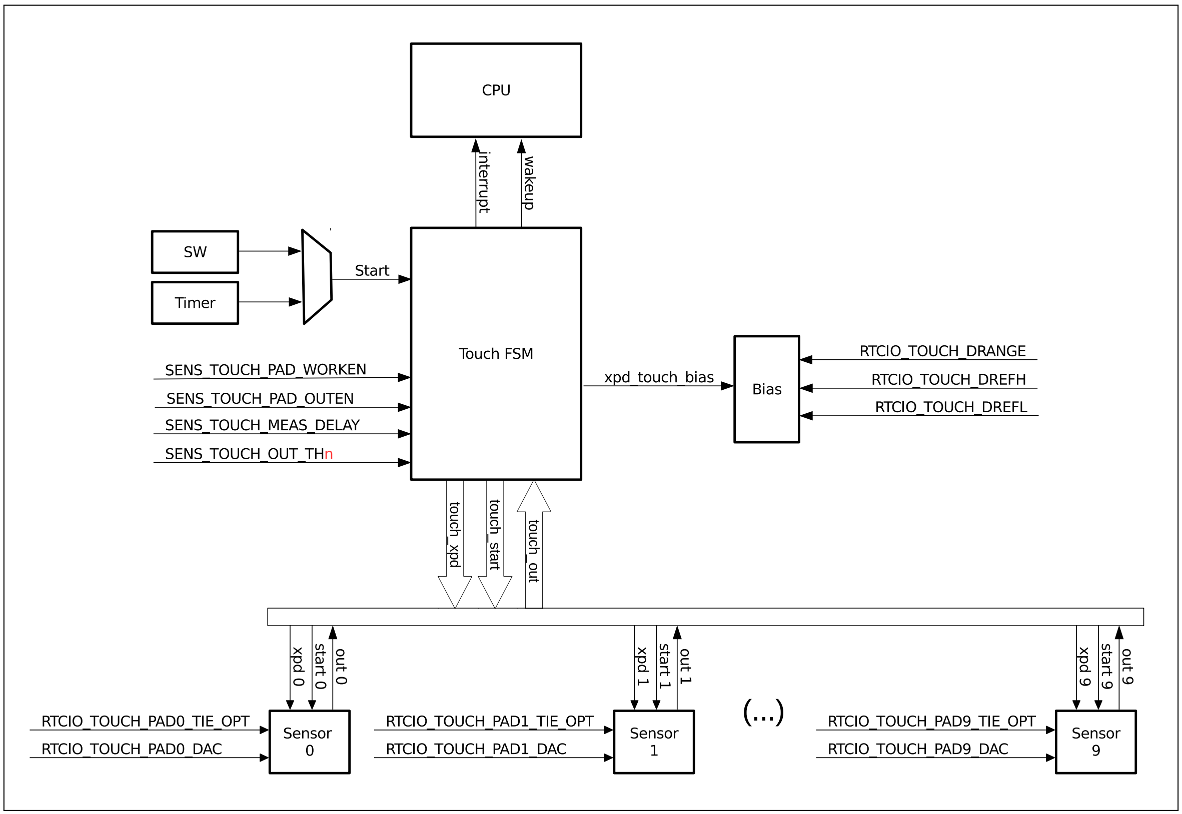


图 29-4. FSM 的内部结构

29.3 SAR ADC

29.3.1 简介

ESP32 内置了 2 个 12 位的 SAR ADC，由 5 个专用转换器控制器管理，可测量来自 18 个管脚的模拟信号。

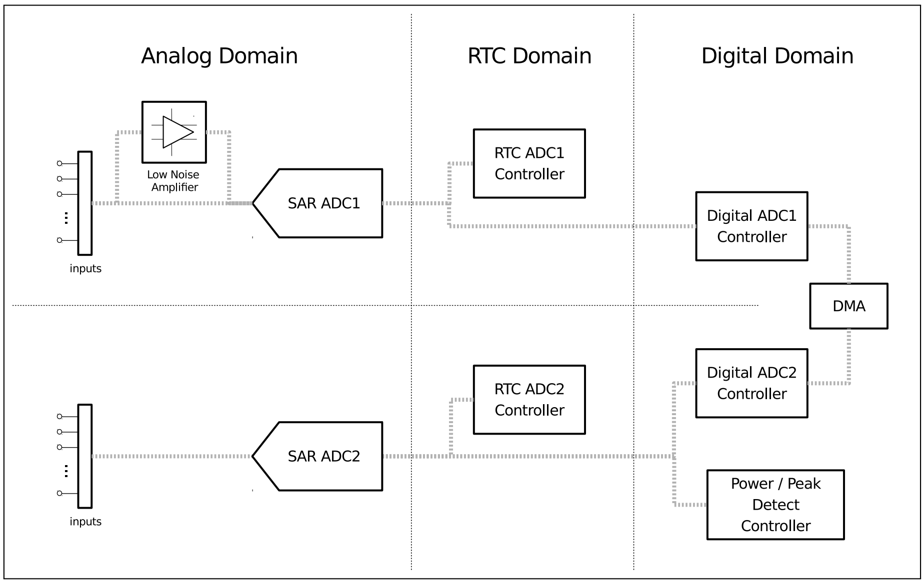


图 29-5. SAR ADC 的概况

SAR ADC 使用的 5 个控制器均为专用控制器，其中 2 个支持高性能多通道扫描、2 个经过优化可支持

Deep-sleep 模式下的低功耗运行，另外 1 个专门用于 PWDET / PKDET （功率检测和峰值监测）。SAR ADC 的基本概况见图 29-5。

说明：
PWDET/PKDET 控制器仅供 Wi-Fi 内部使用。如果 Wi-Fi 正在使用 SAR ADC2，则用户无法使用 SAR ADC2 测量管脚的模拟信号。Wi-Fi 释放 SAR ADC2 之后，用户可正常使用 SAR ADC2。

29.3.2 主要特性

- 采用 2 个 SAR ADC，可支持同时采样与转换
- 采用 5 个专用 ADC 控制器，可支持不同应用场景（比如，高性能、低功耗，或功率检测和峰值检测）
- 支持 18 个模拟输入管脚
- 可配置 12 位、11 位、10 位、9 位多种分辨率
- 支持 DMA（1 个控制器支持）
- 支持多通道扫描模式（2 个控制器支持）
- 支持 Deep-sleep 模式运行（1 个控制器支持）
- 支持 ULP 协处理器控制（2 个控制器支持）

29.3.3 功能概况

SAR ADC 的主要元件与连接情况见图 29-6。

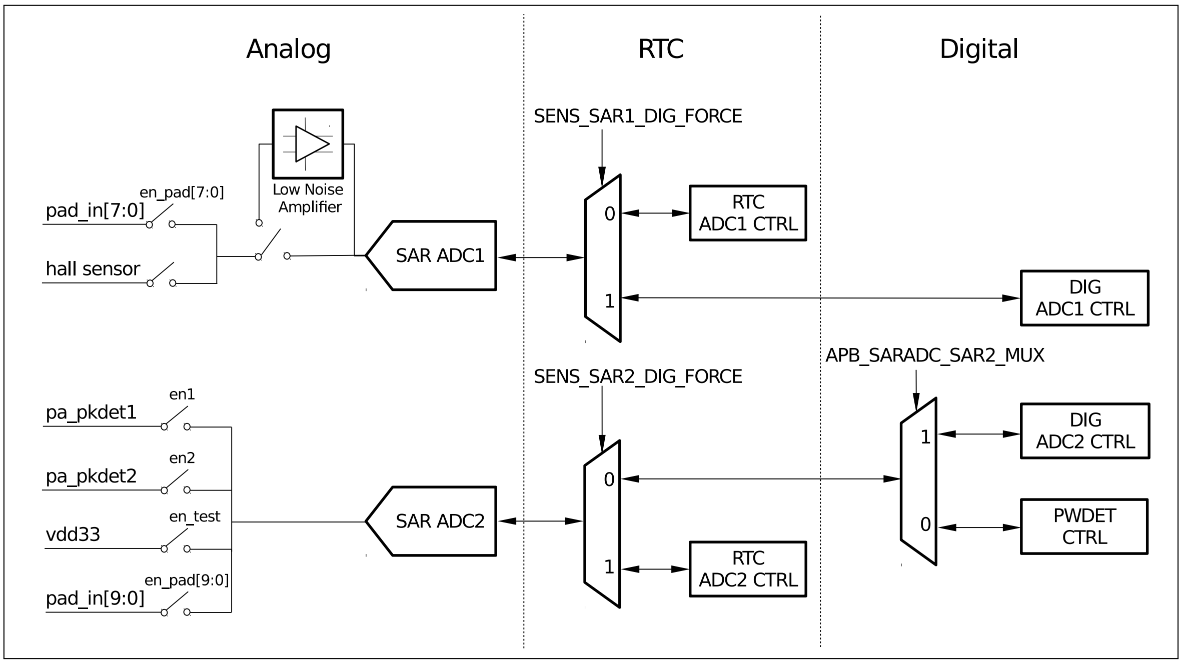


图 29-6. SAR ADC 的功能概况

表 29-2 列出了 SAR ADC 的信号输入与对应接入的 ADC 信道。

表 29-2. SAR ADC 的信号输入

信号名称	ADC 信道 #	ADC 选择
VDET_2	7	SAR ADC1
VDET_1	6	
32K_XN	5	
32K_XP	4	
SENSOR_VN	3	
SENSOR_CAPN	2	
SENSOR_CAPP	1	
SENSOR_VP	0	
GPIO26	9	SAR ADC2
GPIO25	8	
GPIO27	7	
MTMS	6	
MTDI	5	
MTCK	4	
MTDO	3	
GPIO2	2	
GPIO0	1	
GPIO4	0	

说明：

- SAR ADC2 的检测管脚中，包括 GPIO0、GPIO2 和 GPIO15 为芯片 Strapping 管脚，用户使用时需特别注意。

ESP32 内置了 5 个专用 ADC 控制器：RTC ADC1 CTRL、RTC ADC2 CTRL、DIG ADC1 CTRL、DIG ADC2 CTRL，及 PWDET CTRL。各控制器的场景支持情况见表 29-3。

表 29-3. ESP32 的 SAR ADC 控制器

	RTC ADC1	RTC ADC2	DIG ADC1	DIG ADC2	PWDET
DAC	Y	-	-	-	-
Deep-sleep	Y	Y	-	-	-
ULP 协处理器	Y	Y	-	-	-
PWDET/PKDET	-	-	-	-	Y
DMA	-	-	Y	-	-

29.3.4 RTC SAR ADC 控制器

RTC 电源域中的 SAR ADC 控制器（RTC ADC1 CTRL 和 RTC ADC2 CTRL）可在低频状态下提供最小功耗 ADC 测量。

各控制器的具体功能概况见图 29-7。对于每个控制器来说，转换是由寄存器 SENS_SAR_MEAS*n*_START_SAR 触发，测量结果可见寄存器 SENS_SAR_MEAS*n*_DATA_SAR。

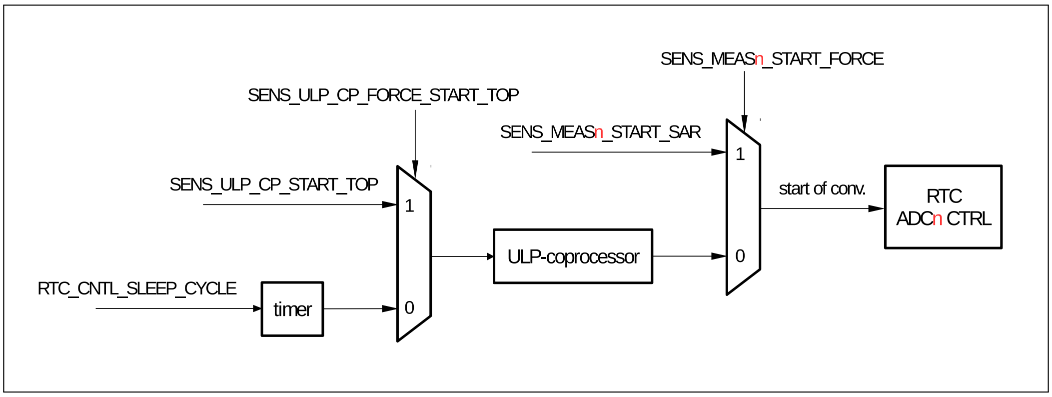


图 29-7. RTC SAR ADC 的功能概况

ULP 协处理器与控制器之间的关系非常紧密，已经内置了指令来使用 ADC。很多情况下，控制器均需要与 ULP 协处理器协同工作，比如：

- 可在 Deep-sleep 模式下对通道进行周期性检测。Deep-sleep 模式下，ULP 协处理器是唯一的触发器。
- 可按一定顺序对通道进行连续性扫描。尽管控制器无法支持连续性扫描或 DMA，但 ULP 协处理器可协助实现这部分功能。

29.3.5 DIG SAR ADC 控制器

与 RTC SAR ADC 控制器相比，DIG SAR ADC 控制器的性能和吞吐均实现了一定优化，具备以下特点：

- 高性能。时钟更快，因此采样速率实现了大幅提升。
- 支持多通道扫描模式。每个 SAR ADC 的测量规则可见样式表。扫描模式可配置为单通道模式、双通道模式或交替模式。
- 扫描可由软件或 I2S 总线发起。
- 支持 DMA。扫描完成即发生中断。

说明：

由于无法通过直接访问发起一次性 SAR ADC 转换，因此我们将在本章节中采用“开始扫描”的说法，代表我们将利用 DIG SAR ADC 控制器扫描一系列通道。

图 29-8 展示了 DIG SAR ADC 控制器的原理图。

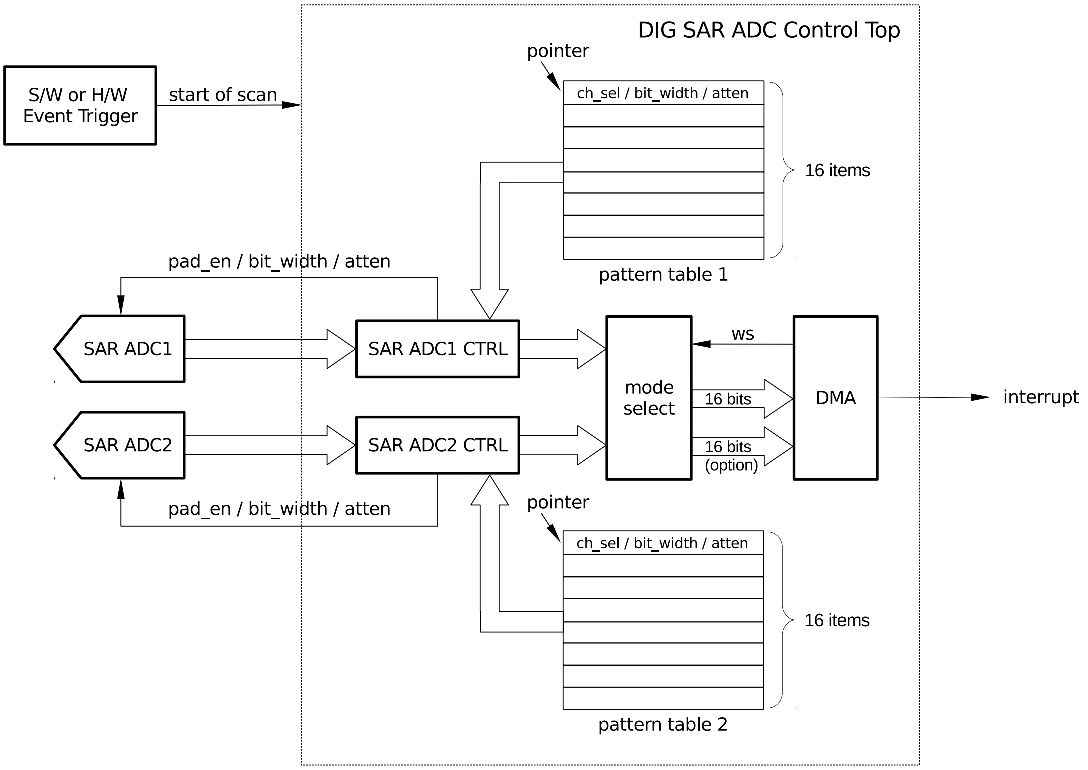


图 29-8. DIG SAR ADC 控制器的概况

样式表可以描述 DIG SAR ADC 控制器需要遵守的各项测量规则，每个表项拥有 16 个项，可存储通道选择、分辨率和衰减信息等内容。当扫描开始时，控制器将逐条读取样式表中的测量规则。对于每个控制器而言，每个扫描序列最多拥有 16 条不同规则。

样式表寄存器的长度为 8 位，共包括 3 个字段，分别存储了通道、分辨率和衰减信息的内容，具体见表 29-4。

表 29-4. 样式表寄存器的字段信息

样式表寄存器 [7:0]		
ch_sel[3:0]	bit_width[1:0]	atten[1:0]
扫描通道	分辨率	衰减

扫描模式可配置为：单通道模式、双通道模式或交替模式。

- 单通道模式：仅 SAR ADC1 或 SAR ADC2 的通道将被扫描。
- 双通道模式：SAR ADC1 和 SAR ADC2 的通道同将被扫描。
- 交替模式：SAR ADC1 和 SAR ADC2 的通道将被交替扫描。

ESP32 最高支持 12 位的 SAR ADC 分辨率，最终向 DMA 传递的 16 位数据包括 ADC 转换结果，及一些因扫描模式不同而有所差别的相关信息，具体为：

- 单通道模式：仅增加 4 位通道选择信息。
- 双通道模式或交替模式：增加 4 位通道选择信息，及 1 位 SAR ADC 选择信息。

每种扫描模式均有其对应的数据格式，即 I 型和 II 型。有关这两种数据格式的具体描述，请见表 29-5 和表 29-6。

表 29-5. I 型 DMA 数据格式

I 型 DMA 数据格式 [15:0]	
ch_sel[3:0]	data[11:0]
通道	SAR ADC 信息

表 29-6. II 型 DMA 数据格式

II 型 DMA 数据格式 [15:0]		
sar_sel	ch_sel[3:0]	SAR ADC data[10:0]
SAR ADC*n*	通道	SAR ADC 信息

I 型数据格式的 SAR ADC 分辨率最高可支持 12 位，II 型数据格式的 SAR ADC 分辨率最高可支持 11 位。

DIG SAR ADC 控制器允许通过 I2S 总线实现直接内存访问。I2S 总线的 WS 信号可用作测量触发信号。可通过 DATA 信号获得测量结果是否完成的信息。可通过软件配置 APB_SARADC_DATA_TO_I2S，将 ADC 连接至 I2S 总线。

29.4 数字模拟转换器

29.4.1 简介

数字模拟转换器 (DAC) 带有 2 个 8 位通道，可将数字值转换为最高 2 路模拟输出信号，包括集成电阻串和缓冲区。这种双通道 DAC 支持将电源当做输入电压参考，且支持双通道的独立/同时转换。

29.4.2 主要特性

DAC 的主要特性包括：

- 2 个 8 位 DAC 通道
- 支持双通道的独立/同时转换
- 可从 VDD3P3_RTC 引脚获得电压参考
- 含有余弦波型发生器
- 支持 DMA 功能
- 可通过软件或 SAR ADC FSM 开始转换。更多信息，请见 SAR ADC 章节。
- 可由 ULP 协处理器通过控制寄存器来实现完全控制。请见 ULP 协处理器章节。

单通道 DAC 的功能概况请见图 29-9。具体介绍，请见本章节下方内容。

29.4.3 结构

双通道 DAC 的 2 个 8 位通道可实现独立配置，每个通道的输出模拟电压计算方式见下：

$$\mathrm{DAC}n\_\mathrm{OUT} = \mathrm{VDD3P3\_RTC} \cdot \mathrm{PDAC}n\_\mathrm{DAC}/255$$

- VDD3P3_RTC 代表 VDD3P3_RTC 引脚的电压 (通常为 3.3V)。

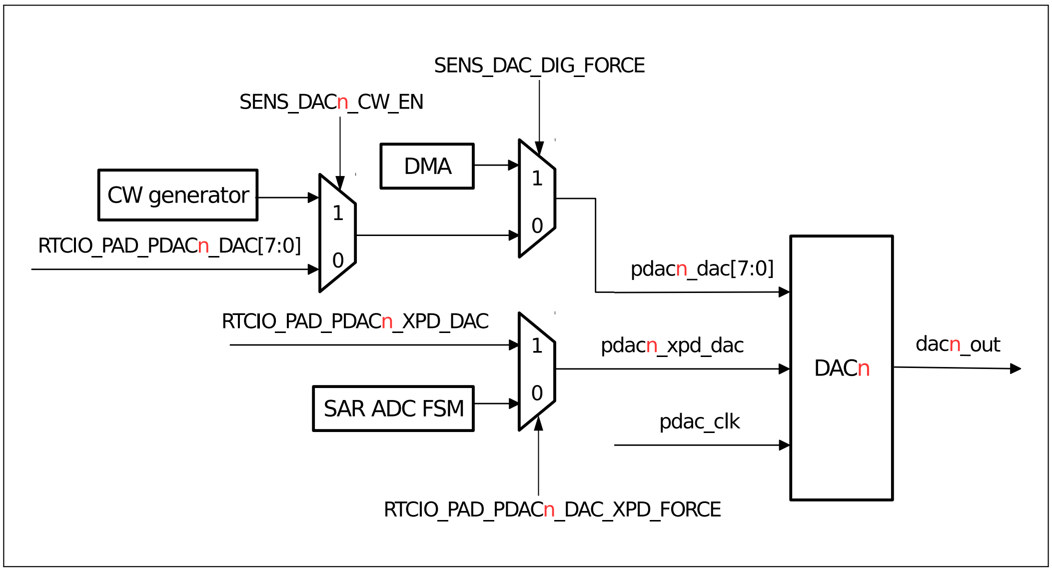


图 29-9. DAC 的功能概况

- PDAC*n*_DAC 拥有多个来源：余弦波形生成器、寄存器 RTCIO_PAD_DAC*n*_REG，及 DMA 。

可通过寄存器 RTCIO_PAD_PDAC*n*_XPD_DAC 决定转换是否开始，软件或 SAR ADC FSM 控制转换流程本身，具体请见图 29-9。

29.4.4 余弦波形生成器

余弦波形生成器可用于生成余弦波形/正弦波形，具体工作流程可见图 29-10。

余弦波形生成器的特点包括：

- 频率可调节
  余弦波的频率可通过寄存器 SENS_SAR_SW_FSTEP[15:0] 调节：

$$\text{freq} = \text{dig\_clk\_rtc\_freq} \cdot \text{SENS\_SAR\_SW\_FSTEP}/65536$$

  通常，dig_clk_rtc 的频率为 8 MHz。

- 振幅可调节
  可通过寄存器 SENS_SAR_DAC_SCALE*n*[1:0] 设置波形振幅，调整为 1、1/2、1/4 或 1/8 倍。

- 直流偏移
  寄存器 SENS_SAR_DAC_DC*n*[7:0] 可能引入一些直流偏移，导致结果饱和。

- 相位移动
  可通过寄存器 SENS_SAR_DAC_INV*n*[1:0] 增加 0/90/180/270° 相位偏移。

29.4.5 支持 DMA

双通道 DAC 的直接内存存取 (DMA) 控制器可对 2 个 DAC 通道的输出进行设置。通过配置 SENS_SAR_DAC_DIG_FORCE ，I2S_clk 可连接至 DAC clk，I2S_DATA_OUT 可连接至 DAC_DATA 实现直接内存访问。

更多信息，请见 DMA 章节。

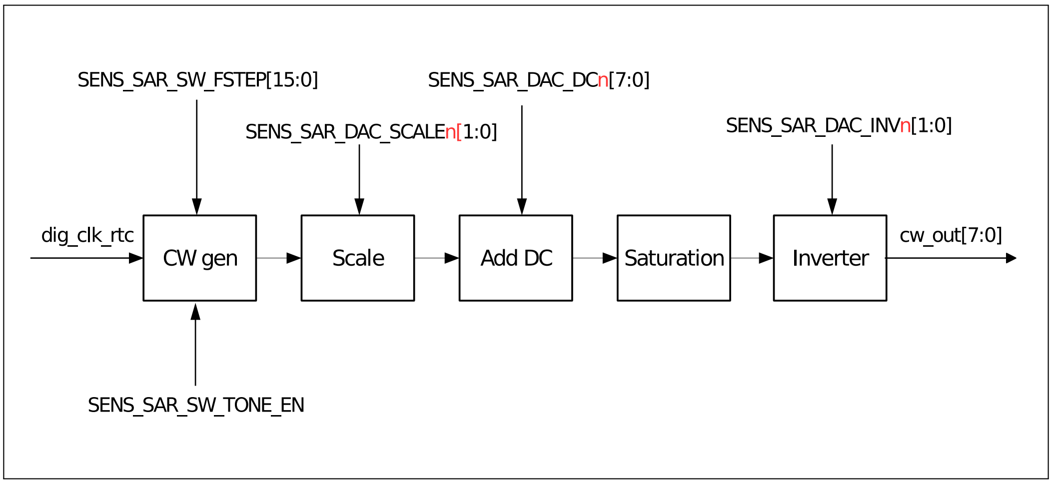


图 29-10. 余弦波形生成器的工作流程

29.5 寄存器列表

说明：下方寄存器的分类主要以功能为主，并不反映访问内存的具体顺序。

29.5.1 传感器

名称	描述	地址	访问类型
触摸管脚设置与控制寄存器			
SENS_SAR_TOUCH_CTRL1_REG	触摸板控制	0x3FF48858	读/写
SENS_SAR_TOUCH_CTRL2_REG	触摸板控制与状态	0x3FF48884	只读
SENS_SAR_TOUCH_ENABLE_REG	唤醒中断控制与工作 SET	0x3FF4888C	读/写
SENS_SAR_TOUCH_THRES1_REG	管脚 0 和管脚 1 的阈值设置	0x3FF4885C	读/写
SENS_SAR_TOUCH_THRES2_REG	管脚 2 和管脚 3 的阈值设置	0x3FF48860	读/写
SENS_SAR_TOUCH_THRES3_REG	管脚 4 和管脚 5 的阈值设置	0x3FF48864	读/写
SENS_SAR_TOUCH_THRES4_REG	管脚 6 和管脚 7 的阈值设置	0x3FF48868	读/写
SENS_SAR_TOUCH_THRES5_REG	管脚 8 和管脚 9 的阈值设置	0x3FF4886C	读/写
SENS_SAR_TOUCH_OUT1_REG	管脚 0 和管脚 1 的计数器	0x3FF48870	只读
SENS_SAR_TOUCH_OUT2_REG	管脚 2 和管脚 3 的计数器	0x3FF48874	只读
SENS_SAR_TOUCH_OUT3_REG	管脚 4 和管脚 5 的计数器	0x3FF48878	只读
SENS_SAR_TOUCH_OUT4_REG	管脚 6 和管脚 7 的计数器	0x3FF4887C	只读
SENS_SAR_TOUCH_OUT5_REG	管脚 8 和管脚 9 的计数器	0x3FF48880	只读
SAR ADC 控制寄存器			
SENS_SAR_START_FORCE_REG	SAR ADC1 和 ADC2 控制	0x3FF4882C	读/写
SAR ADC1 访问控制器			
SENS_SAR_READ_CTRL_REG	SAR ADC1 数据与采样控制	0x3FF48800	读/写
SENS_SAR_MEAS_START1_REG	SAR ADC1 转换控制与状态	0x3FF48854	只读
SAR ADC2 控制寄存器			
SENS_SAR_READ_CTRL2_REG	SAR ADC2 数据与采样控制	0x3FF48890	读/写
SENS_SAR_MEAS_START2_REG	SAR ADC2 转换控制与状态	0x3FF48894	只读
ULP 协处理器配置寄存器			
SENS_ULP_CP_SLEEP_CYC0_REG	ULP 协处理器的睡眠周期	0x3FF48818	读/写
管脚衰减配置寄存器			

SENS_SAR_ATTEN1_REG	每个管脚的 2 位衰减	0x3FF48834	读/写
SENS_SAR_ATTEN2_REG	每个管脚的 2 位衰减	0x3FF48838	读/写
DAC 控制寄存器			
SENS_SAR_DAC_CTRL1_REG	DAC 控制	0x3FF48898	读/写
SENS_SAR_DAC_CTRL2_REG	DAC 输出控制	0x3FF4889C	读/写

29.5.2 外围总线

名称	描述	地址	访问方式
SAR ADC1 和 ADC2 通用配置寄存器			
APB_SARADC_CTRL_REG	SAR ADC 通用配置	0x60002610	读/写
APB_SARADC_CTRL2_REG	SAR ADC 通用配置	0x60002614	读/写
APB_SARADC_FSM_REG	SAR ADC FSM 采样周期配置	0x60002618	读/写
SAR ADC1 样式表寄存器			
APB_SARADC_SAR1_PATT_TAB1_REG	样式表 0 - 3	0x6000261C	读/写
APB_SARADC_SAR1_PATT_TAB2_REG	样式表 4 - 7	0x60002620	读/写
APB_SARADC_SAR1_PATT_TAB3_REG	样式表 8 - 11	0x60002624	读/写
APB_SARADC_SAR1_PATT_TAB4_REG	样式表 12 - 15	0x60002628	读/写
SAR ADC2 样式表寄存器			
APB_SARADC_SAR2_PATT_TAB1_REG	样式表 0 - 3	0x6000262C	读/写
APB_SARADC_SAR2_PATT_TAB2_REG	样式表 4 - 7	0x60002630	读/写
APB_SARADC_SAR2_PATT_TAB3_REG	样式表 8 - 11	0x60002634	读/写
APB_SARADC_SAR2_PATT_TAB4_REG	样式表 12 - 15	0x60002638	读/写

29.5.3 RTC I/O

有关 RTC I/O 的相关寄存器列表，请参见 IO_MUX 和 GPIO 交换矩阵章节中寄存器列表的内容。

29.6 寄存器

29.6.1 传感器

本小节括号中的地址均为相对于 (RTC 基地址 + 0x0800) 的地址偏移量（相对地址）。RTC 基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 29.5.1 传感器。

Register 29.1. SENS_SAR_READ_CTRL_REG (0x0000)

	(reserved)	SENS_SAR1_DATA_INV	SENS_SAR1_DIG_FORCE	(reserved)	SENS_SAR1_SAMPLE_BIT	SENS_SAR1_SAMPLE_CYCLE	SENS_SAR1_CLK_DIV
	31 29	28	27	26 18	17 16	15 8	7 0
Reset	0 0 0	0	0	0 0 0 0 0 0 0 0 0	3	9	2

SENS_SAR1_DATA_INV 反转 SAR ADC1 数据。(读/写)

SENS_SAR1_DIG_FORCE 1：SAR ADC1 由 DIG ADC1 CTR 控制；0：SAR ADC1 由 RTC ADC1 CTRL 控制。(读/写)

SENS_SAR1_SAMPLE_BIT SAR ADC1 的位宽，00：9 位；01：10 位；10：11 位；11：12 位。(读/写)

SENS_SAR1_SAMPLE_CYCLE SAR ADC1 的采样周期。(读/写)

SENS_SAR1_CLK_DIV 时钟分频器。(读/写)

Register 29.2. SENS_ULP_CP_SLEEP_CYC0_REG (0x0018)

	31 0
Reset	200

SENS_ULP_CP_SLEEP_CYC0_REG ULP 协处理器计时器的睡眠周期。(读/写)

Register 29.3. SENS_SAR_START_FORCE_REG (0x002c)

(reserved)	SENS_SAR1_STOP	SENS_SAR2_STOP	SENS_PC_INIT	(reserved)	SENS_ULP_CP_START_TOP	SENS_ULP_CP_FORCE_START_TOP	SENS_SAR2_PWDET_CCT	SENS_SAR2_EN_TEST	SENS_SAR2_BIT_WIDTH	SENS_SAR1_BIT_WIDTH	
31 24	23	22	21 11	10	9	8	7 5	4	3 2	1 0	
0 0 0 0 0 0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0	0	1 1	1 1	Reset

SENS_SAR1_STOP 停止 SAR ADC1 的转换。（读/写）

SENS_SAR2_STOP 停止 SAR ADC2 的转换。（读/写）

SENS_PC_INIT ULP 协处理器的初始化 PC 。（读/写）

SENS_ULP_CP_START_TOP 1：启动 ULP 协处理器，仅在 reg_ulp_cp_force_start_top = 1 时有效。（读/写）

SENS_ULP_CP_FORCE_START_TOP 1：ULP 协处理器由软件启动；0：ULP 协处理器由计时器启动。（读/写）

SENS_SAR2_PWDET_CCT SAR2_PWDET_CCT，PA 功率监测器的电容调谐。（读/写）

SENS_SAR2_EN_TEST SAR2_EN_TEST，仅在 reg_sar2_dig_force = 0 时有效。（读/写）

SENS_SAR2_BIT_WIDTH SAR ADC2 的位宽，00：9 位；01：10 位；10：11 位；11：12 位。（读/写）

SENS_SAR1_BIT_WIDTH SAR ADC1 的位宽，00：9 位；01：10 位；10：11 位；11：12 位。（读/写）

Register 29.4. SENS_SAR_ATTEN1_REG (0x0034)

31 0	
0x0FFFFFFFF	Reset

SENS_SAR_ATTEN1_REG 每个管脚的衰减，11：1 dB；10：6 dB；01：3 dB；00：0 dB。[1:0] 用于描述 ADC1_CH0、[3:2] 用于描述 ADC1_CH1，以此类推。（读/写）

Register 29.5. SENS_SAR_ATTEN2_REG (0x0038)

31 0	
0x0FFFFFFFF	Reset

SENS_SAR_ATTEN2_REG 每个管脚的衰减，11：1 dB；10：6 dB；01：3 dB；00：0 dB。[1:0] 用于描述 ADC2_CH0、[3:2] 用于描述 ADC1_CH2，以此类推。（读/写）

Register 29.6. SENS_SAR_MEAS_START1_REG (0x0054)

SENS_SAR1_EN_PAD_FORCE	SENS_SAR1_EN_PAD	SENS_MEAS1_START_FORCE	SENS_MEAS1_START_SAR	SENS_MEAS1_DONE_SAR	SENS_MEAS1_DATA_SAR	
31	30 19	18	17	16	15 0	
0	0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SENS_SAR1_EN_PAD_FORCE 1：SAR ADC1 管脚使能位图由软件控制；0：由 ULP 协处理器控制。（读/写）

SENS_SAR1_EN_PAD SAR ADC1 管脚使能位图，仅当 reg_sar1_en_pad_force = 1 时有效。（读/写）

SENS_MEAS1_START_FORCE 1：SAR ADC1 控制器 (RTC) 由软件启动；0：由 ULP 协处理器启动。（读/写）

SENS_MEAS1_START_SAR SAR ADC1 控制器 (RTC) 开始转换，仅当 reg_meas1_start_force = 1 时有效。（读/写）

SENS_MEAS1_DONE_SAR SAR ADC1 代表转换已完成。（只读）

SENS_MEAS1_DATA_SAR SAR ADC1 数据。（只读）

Register 29.7. SENS_SAR_TOUCH_CTRL1_REG (0x0058)

(reserved)	(reserved)	(reserved)	SENS_TOUCH_OUT_1EN	SENS_TOUCH_OUT_SEL	SENS_TOUCH_XPD_WAIT	SENS_TOUCH_MEAS_DELAY	
31 28	27	26	25	24	23 16	15 0	
0 0 0 0	0	0	1	0	0x004	0x01000	Reset

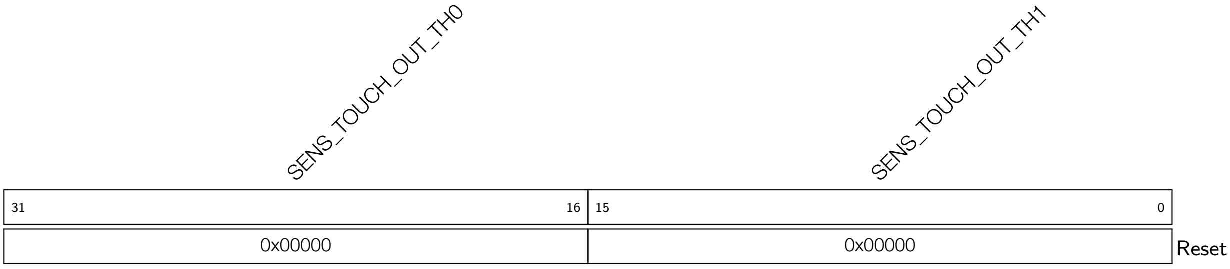
SENS_TOUCH_OUT_1EN 1：当 SET1 中有管脚受到触碰，唤醒中断产生；0：当 SET1 & SET2 中均有管脚受到触碰，唤醒中断产生。（读/写）

SENS_TOUCH_OUT_SEL 1：当计数器数值大于阈值，则视该管脚受到触碰；0：当计数器数值小于阈值，则视该管脚受到触碰。（读/写）

SENS_TOUCH_XPD_WAIT (8 MHz 周期) TOUCH_START 和 TOUCH_XPD 之间的等待时间。（读/写）

SENS_TOUCH_MEAS_DELAY (8 MHz 周期) 测量持续时长。（读/写）

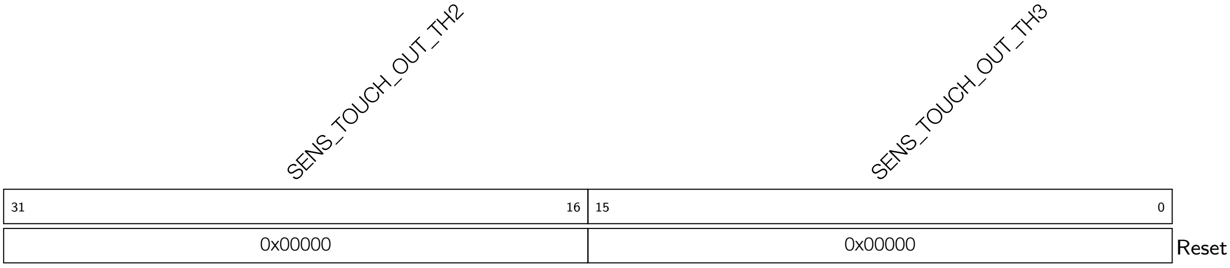
Register 29.8. SENS_SAR_TOUCH_THRES1_REG (0x005c)



SENS_TOUCH_OUT_TH0 管脚 0 的阈值。(读/写)

SENS_TOUCH_OUT_TH1 管脚 1 的阈值。(读/写)

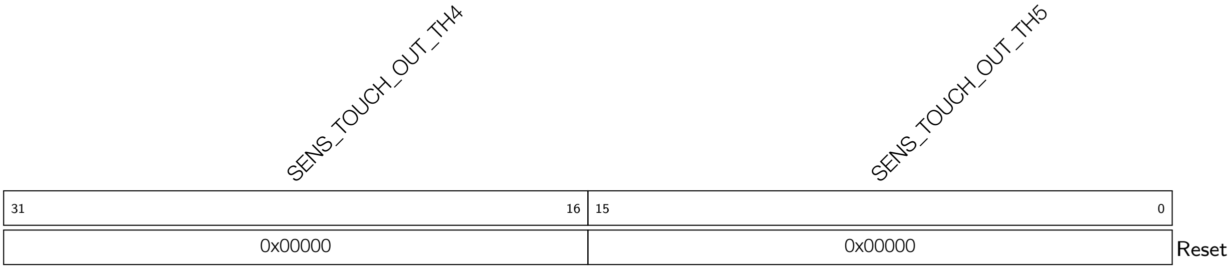
Register 29.9. SENS_SAR_TOUCH_THRES2_REG (0x0060)



SENS_TOUCH_OUT_TH2 管脚 2 的阈值。(读/写)

SENS_TOUCH_OUT_TH3 管脚 3 的阈值。(读/写)

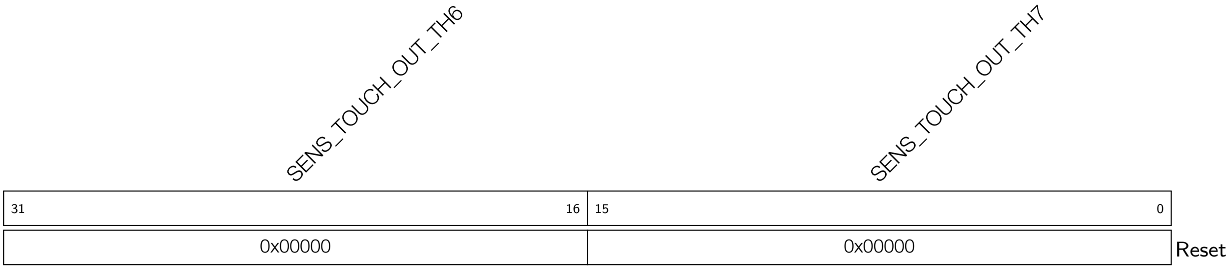
Register 29.10. SENS_SAR_TOUCH_THRES3_REG (0x0064)



SENS_TOUCH_OUT_TH4 管脚 4 的阈值。(读/写)

SENS_TOUCH_OUT_TH5 管脚 5 的阈值。(读/写)

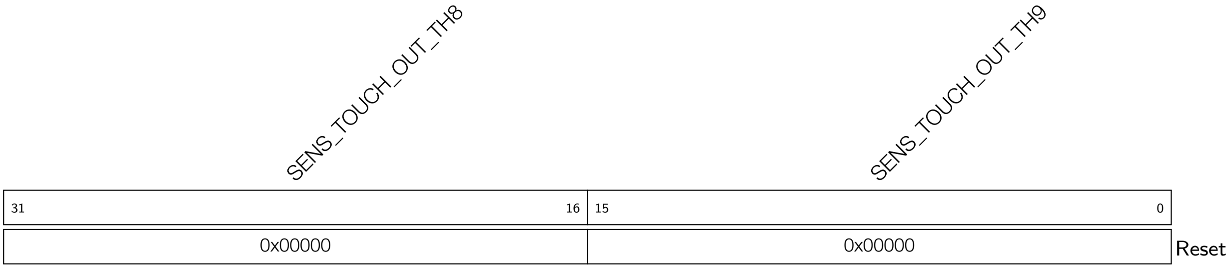
Register 29.11. SENS_SAR_TOUCH_THRES4_REG (0x0068)



SENS_TOUCH_OUT_TH6 管脚 6 的阈值。(读/写)

SENS_TOUCH_OUT_TH7 管脚 7 的阈值。(读/写)

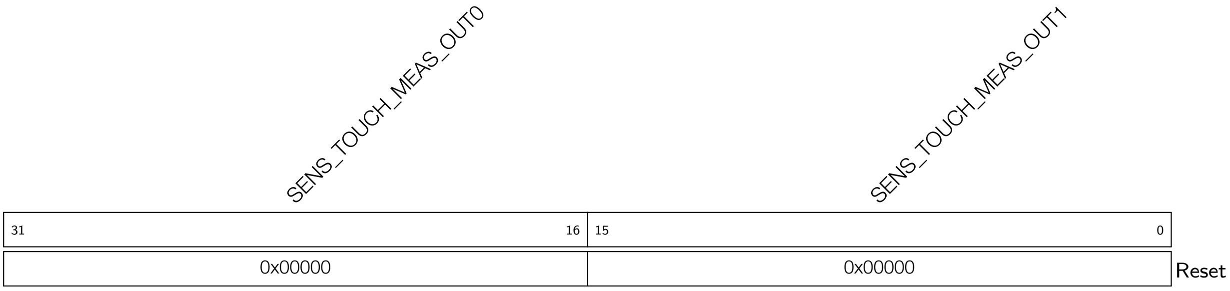
Register 29.12. SENS_SAR_TOUCH_THRES5_REG (0x006c)



SENS_TOUCH_OUT_TH8 管脚 8 的阈值。(读/写)

SENS_TOUCH_OUT_TH9 管脚 9 的阈值。(读/写)

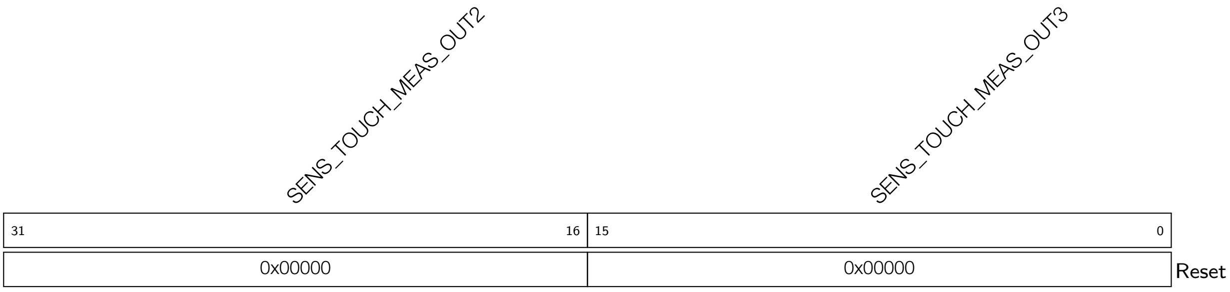
Register 29.13. SENS_SAR_TOUCH_OUT1_REG (0x0070)



SENS_TOUCH_MEAS_OUT0 管脚 0 的计数器。(只读)

SENS_TOUCH_MEAS_OUT1 管脚 1 的计数器。(只读)

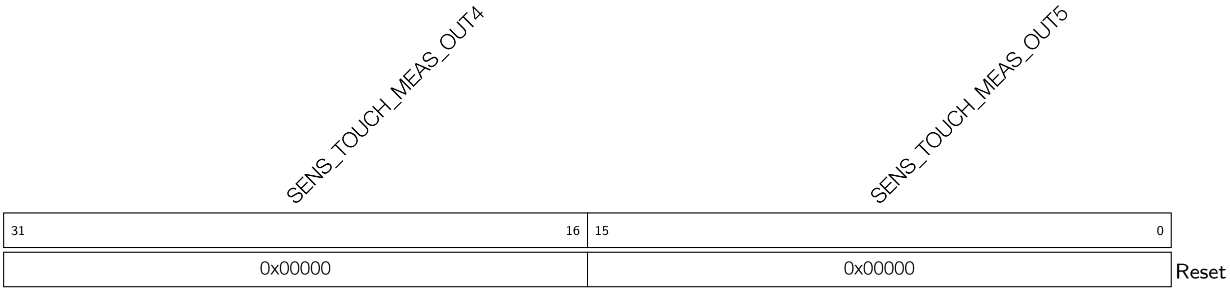
Register 29.14. SENS_SAR_TOUCH_OUT2_REG (0x0074)



SENS_TOUCH_MEAS_OUT2 管脚 2 的计数器。(只读)

SENS_TOUCH_MEAS_OUT3 管脚 3 的计数器。(只读)

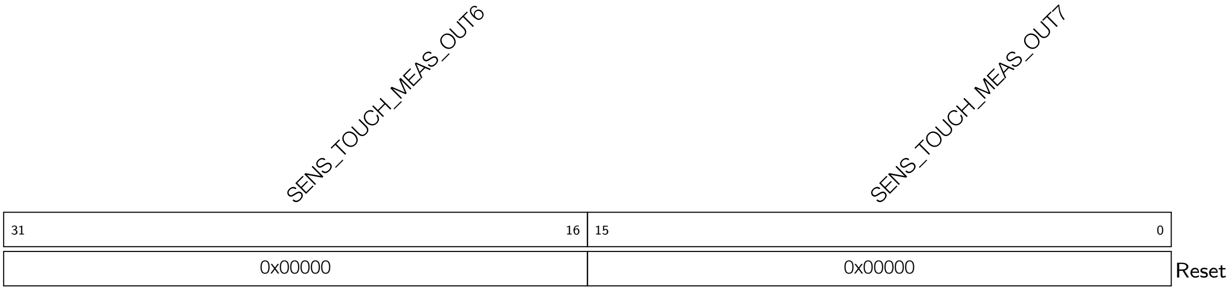
Register 29.15. SENS_SAR_TOUCH_OUT3_REG (0x0078)



SENS_TOUCH_MEAS_OUT4 管脚 4 的计数器。(只读)

SENS_TOUCH_MEAS_OUT5 管脚 5 的计数器。(只读)

Register 29.16. SENS_SAR_TOUCH_OUT4_REG (0x007c)



SENS_TOUCH_MEAS_OUT6 管脚 6 的计数器。(只读)

SENS_TOUCH_MEAS_OUT7 管脚 7 的计数器。(只读)

Register 29.17. SENS_SAR_TOUCH_OUT5_REG (0x0080)

SENS_TOUCH_MEAS_OUT8	SENS_TOUCH_MEAS_OUT9	
31 16	15 0	
0x00000	0x00000	Reset

SENS_TOUCH_MEAS_OUT8 管脚 8 的计数器。（只读）

SENS_TOUCH_MEAS_OUT9 管脚 9 的计数器。（只读）

Register 29.18. SENS_SAR_TOUCH_CTRL2_REG (0x0084)

(reserved)	SENS_TOUCH_MEAS_EN_CLR	SENS_TOUCH_SLEEP_CYCLES	SENS_TOUCH_START_FORCE	SENS_TOUCH_START_EN	SENS_TOUCH_START_FSM_EN	SENS_TOUCH_MEAS_DONE	SENS_TOUCH_MEAS_EN	
31	30	29 14	13	12	11	10	9 0	
0	0	0x00100	0	0	1	0	0x000	Reset

SENS_TOUCH_MEAS_EN_CLR 清零 reg_touch_meas_en 。（只写）

SENS_TOUCH_SLEEP_CYCLES 计时器的睡眠周期。（读/写）

SENS_TOUCH_START_FORCE 1：触摸 FSM 由软件启动；0：由计时器启动。（读/写）

SENS_TOUCH_START_EN 1：启动触摸 FSM，当 reg_touch_start_force 设置时有效。（读/写）

SENS_TOUCH_START_FSM_EN 1：TOUCH_START & TOUCH_XPD 由触摸 FSM 控制；0：由寄存器控制。（读/写）

SENS_TOUCH_MEAS_DONE 由 FSM 设置，代表触摸测量已完成。（只读）

SENS_TOUCH_MEAS_EN 10 位寄存器，代表具体受到触碰的管脚。（只读）

Register 29.19. SENS_SAR_TOUCH_ENABLE_REG (0x008c)

(reserved)	SENS_TOUCH_PAD_OUTEN1	SENS_TOUCH_PAD_OUTEN2	SENS_TOUCH_PAD_WORKEN	
31 30	29 20	19 10	9 0	
0 0	0x3FF	0x3FF	0x3FF	Reset

SENS_TOUCH_PAD_OUTEN1 定义 SET1 唤醒中断生成的位图，仅当 SET1 中至少有 1 个管脚受到触碰，才视 SET1 受到触碰。(读/写)

SENS_TOUCH_PAD_OUTEN2 定义 SET2 唤醒中断生成的位图，仅当 SET2 中至少有 1 个管脚受到触碰，才视 SET2 受到触碰。(读/写)

SENS_TOUCH_PAD_WORKEN 定义测量有效 SET 的位图。(读/写)

Register 29.20. SENS_SAR_READ_CTRL2_REG (0x0090)

(reserved)	SENS_SAR2_DATA_INV	SENS_SAR2_DIG_FORCE	(reserved)	SENS_SAR2_SAMPLE_BIT	SENS_SAR2_SAMPLE_CYCLE	SENS_SAR2_CLK_DIV	
31 30	29	28	27 18	17 16	15 8	7 0	
0 0	0	0	0 0 0 0 0 0 0 0 0 0	3	9	2	Reset

SENS_SAR2_DATA_INV 反转 SAR ADC2 数据。(读/写)

SENS_SAR2_DIG_FORCE 1：SAR ADC2 由 DIG ADC2 CTRL 或 PWDET CTRL 控制；0：由 RTC ADC2 CTRL 控制。(读/写)

SENS_SAR2_SAMPLE_BIT SAR ADC2 的位宽，00：9 位；01：10 位；10：11 位；11：12 位。(读/写)

SENS_SAR2_SAMPLE_CYCLE SAR ADC2 的采样周期。(读/写)

SENS_SAR2_CLK_DIV 时钟分频器。(读/写)

Register 29.21. SENS_SAR_MEAS_START2_REG (0x0094)

SENS_SAR2_EN_PAD_FORCE	SENS_SAR2_EN_PAD	SENS_MEAS2_START_FORCE	SENS_MEAS2_START_SAR	SENS_MEAS2_DONE_SAR	SENS_MEAS2_DATA_SAR	
31	30 19	18	17	16	15 0	
0	0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

SENS_SAR2_EN_PAD_FORCE 1：SAR ADC2 管脚使能位图由软件控制；0：由 ULP 协处理器控制。（读/写）

SENS_SAR2_EN_PAD SAR ADC2 管脚使能位图，仅当 reg_sar2_en_pad_force = 1 时有效。（读/写）

SENS_MEAS2_START_FORCE 1：SAR ADC2 控制器 (RTC) 由软件启动；0：由 ULP 协控制器启动。（读/写）

SENS_MEAS2_START_SAR SAR ADC2 控制器 (RTC) 开始转换，仅当 reg_meas2_start_force = 1 时有效。（读/写）

SENS_MEAS2_DONE_SAR 代表 SAR ADC2 转换已完成。（只读）

SENS_MEAS2_DATA_SAR SAR ADC2 数据。（只读）

Register 29.22. SENS_SAR_DAC_CTRL1_REG (0x0098)

(reserved)	SENS_DAC_CLK_INV	SENS_DAC_CLK_FORCE_HIGH	SENS_DAC_CLK_FORCE_LOW	SENS_DAC_DIG_FORCE	(reserved)	SENS_SW_TONE_EN	SENS_SW_FSTEP	
31 26	25	24	23	22	21 17	16	15 0	
0 0 0 0 0 0	0	0	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

SENS_DAC_CLK_INV 1：反转 PDAC_CLK；0；不进行反转。（读/写）

SENS_DAC_CLK_FORCE_HIGH PDAC_CLK 强制取 1。（读/写）

SENS_DAC_CLK_FORCE_LOW PDAC_CLK 强制取 0。（读/写）

SENS_DAC_DIG_FORCE 1：DAC1 & DAC2 使用 DMA；0：DAC1 & DAC2 不使用 DMA 。（读/写）

SENS_SW_TONE_EN 1：使能 CW 发生器；0：禁用 CW 发生器。（读/写）

SENS_SW_FSTEP CW 发生器的频率阶跃，可用于调节波形频率。（读/写）

Register 29.23. SENS_SAR_DAC_CTRL2_REG (0x009c)

(reserved)	SENS_DAC_CW_EN2	SENS_DAC_CW_EN1	SENS_DAC_INV2	SENS_DAC_INV1	SENS_DAC_SCALE2	SENS_DAC_SCALE1	SENS_DAC_DC2	SENS_DAC_DC1	
31 26	25	24	23 22	21 20	19 18	17 16	15 8	7 0	
0 0 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 0 0 0	Reset

SENS_DAC_CW_EN2 1：选择 CW 发生器为 PDAC2_DAC[7:0] 的数据来源；0：选择寄存器 reg_pdac2_dac[7:0] 为 PDAC2_DAC[7:0] 的数据来源。(读/写)

SENS_DAC_CW_EN1 1：选择 CW 发生器为 PDAC1_DAC[7:0] 的数据来源；0：选择寄存器 reg_pdac1_dac[7:0] 为 PDAC1_DAC[7:0] 的数据来源。(读/写)

SENS_DAC_INV2 DAC2，00：不反转任何位；01：反转所有位：10：反转 MSB；11：反转除 MSB 外的所有位。(读/写)

SENS_DAC_INV1 DAC1，00：不反转任何位；01：反转所有位：10：反转 MSB；11：反转除 MSB 外的所有位。(读/写)

SENS_DAC_SCALE2 DAC2，00：1 倍；01：1/2 倍；10：1/4 倍；11：1/8 倍。(读/写)

SENS_DAC_SCALE1 DAC1，00：1 倍；01：1/2 倍；10：1/4 倍；11：1/8 倍。(读/写)

SENS_DAC_DC2 DAC2 CW 发生器的直流偏移。(读/写)

SENS_DAC_DC1 DAC1 CW 发生器的直流偏移。(读/写)

29.6.2 高级外围总线

本小节括号中的地址均为相对于 0x6000_2600 的地址偏移量（经 AHB 总线访问）。寄存器绝对地址见章节 29.5.2 外围总线。

Register 29.24. APB_SARADC_CTRL_REG (0x10)

(reserved)	APB_SARADC_DATA_TO_I2S	APB_SARADC_DATA_SAR_SEL	APB_SARADC_SAR2_PATT_P_CLEAR	APB_SARADC_SAR1_PATT_P_CLEAR	APB_SARADC_SAR2_PATT_LEN	APB_SARADC_SAR1_PATT_LEN	APB_SARADC_SAR_CLK_DIV	APB_SARADC_SAR_CLK_GATED	APB_SARADC_SAR_SEL	APB_SARADC_WORK_MODE	APB_SARADC_SAR2_MUX	APB_SARADC_START	APB_SARADC_START_FORCE	
31 27	26	25	24	23	22 19	18 15	14 7	6	5	4 3	2	1	0	
0 0 0 0 0	0	0	0	0	15	15	4	1	0	0	0	0	0	Reset

APB_SARADC_DATA_TO_I2S 1：I2S (DMA) 输入数据来自 SAR ADC；0：I2S 输入数据来自 GPIO 矩阵。（读/写）

APB_SARADC_DATA_SAR_SEL 1：sar_sel 将由 16 位输出数据的 MSB 编码，此时分辨率不应大于 11 位；0：此时 SAR ADC 分辨率应为 12 位。（读/写）

APB_SARADC_SAR2_PATT_P_CLEAR DIG ADC2 CTRL 的样式表指针清零。（读/写）

APB_SARADC_SAR1_PATT_P_CLEAR DIG ADC1 CTRL 的样式表指针清零。（读/写）

APB_SARADC_SAR2_PATT_LEN SAR ADC2，0 - 15 代表样式表的 1 - 16 位。（读/写）

APB_SARADC_SAR1_PATT_LEN SAR ADC1，0 - 15 代表样式表的 1 - 16 位。（读/写）

APB_SARADC_SAR_CLK_DIV SAR 时钟分频器。（读/写）

APB_SARADC_SAR_CLK_GATED 保留位，请初始化为 0b1 。（读/写）

APB_SARADC_SAR_SEL 0：SAR1；1：SAR2，该设置适用单通道 SAR 模式。（读/写）

APB_SARADC_WORK_MODE 0：单通道模式；1：双通道模式；2：交替模式。（读/写）

APB_SARADC_SAR2_MUX 1：SAR ADC2 由 DIG ADC2 CTRL 控制；0：SAR ADC2 由 PWDET CTRL 控制。（读/写）

APB_SARADC_START 保留位，请初始化为 0 。（读/写）

APB_SARADC_START_FORCE 保留位，请初始化为 0 。（读/写）

Register 29.25. APB_SARADC_CTRL2_REG (0x14)

(reserved)	APB_SARADC_SAR2_INV	APB_SARADC_SAR1_INV	APB_SARADC_MAX_MEAS_NUM	APB_SARADC_MEAS_NUM_LIMIT	
31 11	10	9	8 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	255	0	Reset

APB_SARADC_SAR2_INV 1：输入 DIG ADC2 CTRL 的数据被反转；0：输入 DIG ADC2 CTRL 的数据未被反转。（读/写）

APB_SARADC_SAR1_INV 1：输入 DIG ADC1 CTRL 的数据被反转；0：输入 DIG ADC1 CTRL 的数据未被反转。（读/写）

APB_SARADC_MAX_MEAS_NUM 最大转换数量。（读/写）

APB_SARADC_MEAS_NUM_LIMIT 保留位，请初始化为 0b1 。（读/写）

Register 29.26. APB_SARADC_FSM_REG (0x18)

APB_SARADC_SAMPLE_CYCLE	(reserved)	(reserved)	(reserved)	
31 24	23 16	15 8	7 0	
2	8	255	8	Reset

APB_SARADC_SAMPLE_CYCLE 采样周期。（读/写）

Register 29.27. APB_SARADC_SAR1_PATT_TAB1_REG (0x1C)

31 0	
0x00F0F0F0F	Reset

APB_SARADC_SAR1_PATT_TAB1_REG 代表 SAR ADC1 的样式表 0 - 3，每项占用 1 字节：[31:28] pattern0_channel、[27:26] pattern0_bit_width、[25:24] pattern0_attenuation、[23:20] pattern1_channel，等。（读/写）

Register 29.28. APB_SARADC_SAR1_PATT_TAB2_REG (0x20)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR1_PATT_TAB2_REG 代表 SAR ADC1 的样式表 4 - 7，每项占用 1 字节：[31:28] pattern4_channel、[27:26] pattern4_bit_width、[25:24]。pattern4_attenuation、[23:20] pattern5_channel，等。（读/写）

Register 29.29. APB_SARADC_SAR1_PATT_TAB3_REG (0x24)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR1_PATT_TAB3_REG 代表 SAR ADC1 的样式表 8 - 11，每项占用 1 字节：[31:28] pattern8_channel、[27:26] pattern8_bit_width、[25:24] pattern8_attenuation、[23:20] pattern9_channel，等。（读/写）

Register 29.30. APB_SARADC_SAR1_PATT_TAB4_REG (0x28)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR1_PATT_TAB4_REG 代表 SAR ADC1 的样式表 12 - 15，每项占用 1 字节：[31:28] pattern12_channel、[27:26] pattern12_bit_width、[25:24] pattern12_attenuation、[23:20] pattern13_channel，等。（读/写）

Register 29.31. APB_SARADC_SAR2_PATT_TAB1_REG (0x2C)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR2_PATT_TAB1_REG 代表 SAR ADC2 的样式表 0 - 3，每项占用 1 字节：[31:28] pattern0_channel、[27:26] pattern0_bit_width、[25:24] pattern0_attenuation、[23:20] pattern1_channel，等。（读/写）

Register 29.32. APB_SARADC_SAR2_PATT_TAB2_REG (0x30)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR2_PATT_TAB2_REG 代表 SAR ADC2 的样式表 4 - 7，每项占用 1 字节：[31:28] pattern4_channel、[27:26] pattern4_bit_width、[25:24] pattern4_attenuation、[23:20] pattern5_channel，等。（读/写）

Register 29.33. APB_SARADC_SAR2_PATT_TAB3_REG (0x34)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR2_PATT_TAB3_REG 代表 SAR ADC2 的样式表 8 - 11，每项占用 1 字节：[31:28] pattern8_channel、[27:26] pattern8_bit_width、[25:24] pattern8_attenuation、[23:20] pattern9_channel，等。（读/写）

Register 29.34. APB_SARADC_SAR2_PATT_TAB4_REG (0x38)

31	0	
0x00F0F0F0F		Reset

APB_SARADC_SAR2_PATT_TAB4_REG 代表 SAR ADC2 的样式表 12 - 15，每项占用 1 字节：[31:28] pattern12_channel、[27:26] pattern12_bit_width、[25:24] pattern12_attenuation、[23:20] pattern13_channel，等。（读/写）

29.6.3 RTC I/O

有关 RTC I/O 的相关寄存器，请参见 IO_MUX 和 GPIO 交换矩阵章节中寄存器的内容。

30 超低功耗协处理器 (ULP)

30.1 概述

超低功耗协处理器 (ULP Coprocessor) 是一种功耗极低的协处理器设备，可在主系统级芯片 (SoC) 系统进入 Deep-sleep 状态时保持上电，允许开发者通过存储在 RTC 中的专用程序，访问外围设备、内部传感器及 RTC 寄存器。ULP 协处理器的主要应用场景包括一些需要在保证最低功耗的情况下，通过外部活动或计时器（或两者兼有）唤醒 CPU 的应用。

30.2 主要特性

- 可访问最多 8 KB SRAM RTC 慢速内存，储存指令和数据
- 采用 8 MHz RTC_FAST_CLK 时钟频率
- 支持正常模式和 Deep-sleep 模式
- 可唤醒 CPU 或向 CPU 发送中断
- 可访问外围设备、内部传感器及 RTC 寄存器
- 采用 4 个 16 位通用寄存器 (R0 - R3)，进行数据操作和内存访问
- 采用 1 个 8 位阶段计数器寄存器 Stage_cnt，可通过 ALU 指令进行操作并用于 JUMP 指令

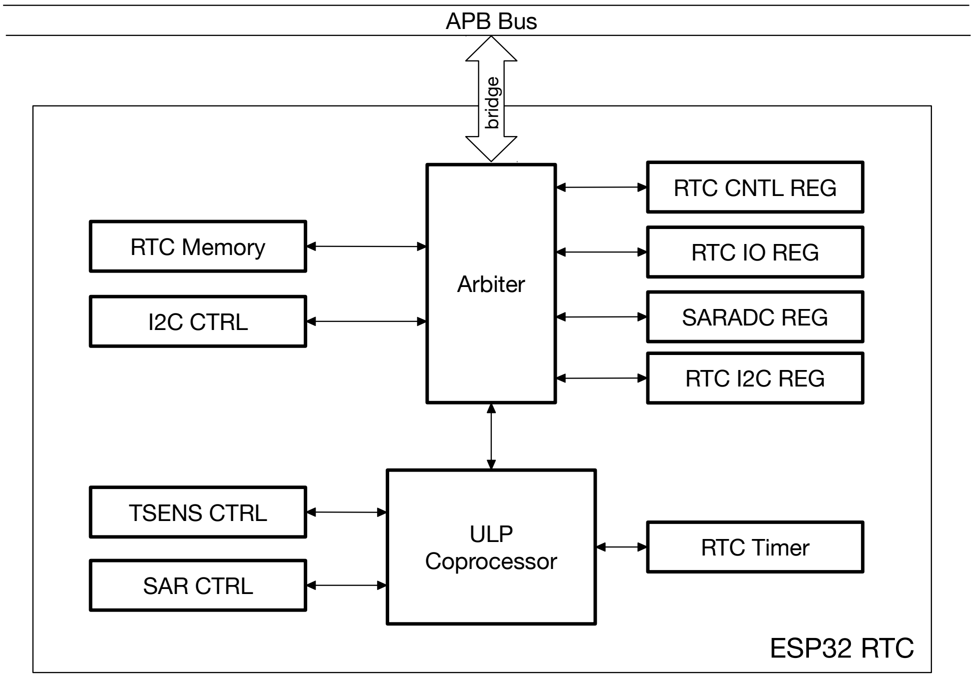


图 30-1. ULP 协处理器基本架构

30.3 功能描述

ULP 协处理器是一种可编程有限状态机 (FSM)，可在 CPU 进入 Deep-sleep 状态时工作。协处理器支持部分通用 CPU 指令，可协助进行一些复杂逻辑运算。此外，还支持一些特殊的 RTC 控制与外围设备控制指令。与 CPU 一样，ULP 协处理器也可访问 8 KB SRAM RTC 慢速内存。也正因如此，这块内存经常被用于存储一些协处理器和 CPU 的通用指令。

ULP 协处理器可由软件程序或硬件定时器周期性启动，并通过执行 HALT 指令停止。协处理器的功能非常强大，可以通过内置指令和 RTC 寄存器，访问 RTC 域中的几乎所有模块。ULP 协处理器可在很多应用场景中成为 CPU 的有力补充，甚至取代 CPU，特别是在一些对功耗很敏感的应用中。ULP 协处理器的基本架构可见图 30-1。

30.4 指令集

ULP 协处理器可支持下列指令：

- 算数与逻辑 - ALU
- 加载与数据存储 - LD、ST、REG_RD 及 REG_WR
- 跳转至某地址 - JUMP
- 管理程序执行 - WAIT 和 HALT
- 控制协处理器的睡眠周期 - SLEEP
- 唤醒 CPU 及与 SoC 通信 - WAKE
- 测量 - ADC
- I^{2}C 总线通信 - I2C_RD 和 I2C_WR

ULP 协处理器指令的格式可见图 30-2。

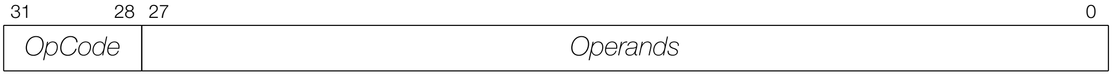


图 30-2. ULP 协处理器的指令格式

根据 *Operands* 的设置不同，同一个 *OpCode* 可对应多种不同操作。比如，ALU 能够执行 10 种不同的算数和逻辑运算，JUMP 也可执行有条件跳转、无条件跳转、绝对跳转及相对跳转等多种形式的跳转。

ULP 协处理器的所有指令均固定为 32 位。通过这一系列指令，协处理器程序即可得到执行。程序内部的执行均采用 32 位寻址。该程序具体存储在 1 块专用的慢速内存区 (RTC_SLOW_MEM)，地址范围为 0x5000_0000 到 0x5000_1FFF (8 KB)，对主 CPU 可见。

本文中 *OpCode* 的格式为 4’d*x*，其中 4 代表 4 位宽，’d 代表十进制，*x* 代表 *OpCode* 的值 (*x*: 0 ~ 15)。

30.4.1 ALU - 算数与逻辑运算

算数逻辑单元 (ALU) 可以进行算数和逻辑运算，对象为协处理器寄存器中存储的数值或指令中存储的立即值。

具体可以支持的运算类型如下：

- 算数 - 加 (ADD) 和减 (SUB)

- 逻辑 - 按位与 (AND) 和按位或 (OR)
- 移位 - 左移 (LSH) 和右移 (RSH)
- 寄存器赋值 - 移动 (MOVE)
- 计数器寄存器操作 - STAGE_RST、STAGE_INC 和 STAGE_DEC

尽管 OpCode 相同, 但可通过设置协处理器指令 [27:21] 位，选择特定的算数和逻辑运算。

30.4.1.1 对寄存器数值的运算

31 28	27 25	24 21		5 4	3 2	1 0
4’d7	3’b0	ALU_sel		Rsrc2	Rsrc1	Rdst

图 30-3. 指令类型 - 对寄存器数值的 ALU 运算

如图 30-3 所示，当协处理器指令 [27:25] 位设置为 3’b0 时，ALU 将对协处理器寄存器 R[0-3] 中存储的内容进行运算，运算类型则取决于指令的 ALU_sel [24:21] 位，具体设置方式见下表 30-1。

Operand	**描述** - 见图 30-3
ALU_sel	ALU 运算类型
Rdst	寄存器 R[0-3]，目标地址寄存器
Rsrc1	寄存器 R[0-3]，源地址寄存器
Rsrc2	寄存器 R[0-3]，源地址寄存器

ALU_sel	指令	运算类型	描述
0	ADD	*Rdst* = *Rsrc1* + *Rsrc2*	加
1	SUB	*Rdst* = *Rsrc1* - *Rsrc2*	减
2	AND	*Rdst* = *Rsrc1* & *Rsrc2*	按位与
3	OR	*Rdst* = *Rsrc1* | *Rsrc2*	按位或
4	MOVE	*Rdst* = *Rsrc1*	寄存器赋值
5	LSH	*Rdst* = *Rsrc1* «*Rsrc2*	左移
6	RSH	*Rdst* = *Rsrc1* »*Rsrc2*	右移

表 30-1. 对寄存器数值的 ALU 运算

注意：

- ADD 和 SUB 运算可用于设置或清除 ALU 溢出标志位。
- 所有 ALU 运算均可用于设置或清除 ALU 零标志位。

30.4.1.2 对指令立即值的运算

31 28	27 25	24 21		19 4	3 2	1 0
4’d7	3’b1	ALU_sel		Imm	Rsrc1	Rdst

图 30-4. 指令类型 - 对指令立即值的 ALU 运算

如图 30-4 所示，当协处理器指令 [27:25] 位设置为 3’b1 时，ALU 将对协处理器寄存器 R[0-3] 和指令 [19:4] 位存储的立即值进行运算，运算类型取决于指令的 ALU_sel [24:21] 位，具体设置方式见下表 30-2。

Operand	**描述** - 见图 30-4
ALU_sel	ALU 运算类型
Rdst	寄存器 R[0-3]，目标地址寄存器
Rsrc1	寄存器 R[0-3]，源地址寄存器
Imm	指令立即值，16 位有符号数

ALU_sel	指令	运算	描述
0	ADD	*Rdst* = *Rsrc1* + *Imm*	加
1	SUB	*Rdst* = *Rsrc1* - *Imm*	减
2	AND	*Rdst* = *Rsrc1* & *Imm*	按位与
3	OR	*Rdst* = *Rsrc1* | *Imm*	按位或
4	MOVE	*Rdst* = *Imm*	寄存器赋值
5	LSH	*Rdst* = *Rsrc1* «*Jmm*	左移
6	RSH	*Rdst* = *Rsrc1* »*Jmm*	右移

表 30-2. 对指令立即值的 ALU 运算

注意：

- ADD 和 SUB 运算可用于设置或清除 ALU 溢出标志位。
- 所有 ALU 运算均可用于设置或清除 ALU 零标志位。

30.4.1.3 对阶段计数器寄存器数值的运算

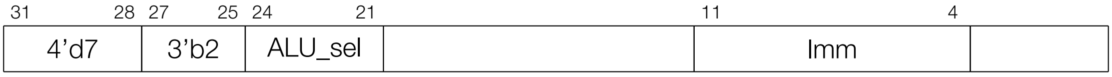


图 30-5. 指令类型 - 对阶段计数器寄存器的 ALU 运算

如图 30-5 所示，当协处理器指令 [27:25] 位设置为 3’b2 时，ALU 将对 8 位寄存器 Stage_cnt 进行递增、递减或重置操作，运算类型取决于指令的 ALU_sel [24:21] 位，具体设置方式见下表 30-5。

Operand	**描述** - 见图 30-5
ALU_sel	ALU 运算类型
Stage_cnt	专用 8 位阶段计数器寄存器，可存储循环下标等变量
Imm	指令立即值，8 位数

ALU_sel	指令	运算	描述
0	STAGE_INC	*Stage_cnt* = *Stage_cnt* + *Imm*	阶段计数器寄存器递增
1	STAGE_DEC	*Stage_cnt* = *Stage_cnt* - *Imm*	阶段计数器寄存器递减
2	STAGE_RST	*Stage_cnt* = 0	阶段计数器寄存器复位

表 30-3. 对阶段计数器寄存器的 ALU 运算

30.4.2 ST –存储数据至内存

31 28	27 25		20 10		3 2	1 0
4’d6	3’b100	4’b0	Offset	6’b0	Rdst	Rsrc

图 30-6. 指令类型 - ST

Operand	**描述** - 见图 30-6
Offset	10 位有符号数，单位为 32 位字
Rsrc	寄存器 R[0-3]，保留需存储的 16 位数
Rdst	寄存器 R[0-3]，目标寄存器地址，单位为 32 位字

描述

该指令可将 *Rsrc* 中保留的 16 位数存储至内存地址为 *Rdst* + *Offset* 的低半字中。该内存中的高半字为程序计数器 (PC) 中的内容（以字为单位）左移 5 位，逻辑或 Rdst (0..3)：

$$\text{Mem [Rdst + Offset]\{31:0\} = \{PC[10:0], 3'b0, Rdst, Rsrc[15:0]\}}$$

应用程序可通过高半字判断 ULP 程序中的哪条指令具体向内存写入了什么内容。

注意：

- 该指令仅能以 32 位字为单位进行访问。
- 任何情况下，*Rsrc* 中保留的 16 位数永远仅能存储至内存的低半字中，即永远不可能将 *Rsrc* 存储至内存的高半字中。
- Mem 写入的是 RTC_SLOW_MEM 慢速内存，ULP 协处理器的地址 0 即相当于主 CPU 的地址 0x50000000。

30.4.3 LD –从内存加载数据

31 28		20 10		3 2	1 0
4’d13		Offset		Rsrc	Rdst

图 30-7. 指令类型 - LD

Operand	**描述** - 见图 30-7
Offset	10 位有符号数，单位为 32 位字
Rsrc	寄存器 R[0-3]，保留目标寄存器的地址，单位为 32 位字
Rdst	寄存器 R[0-3]，目标寄存器

描述

该指令可将内存地址 *Rsrc* + *offset* 中的低半字加载至目标寄存器 *Rdst*：

$$\text{Rdst[15:0] = Mem[Rsrc + Offset][15:0]}$$

注意：

- 该指令仅能以 32 位字为单位进行访问。
- 任何情况下，*Rsrt* 仅可加载内存的低半字，即永远不可能将内存的高半字加载至 *Rsrt* 中。

- 加载的 Mem 将存储至 RTC_SLOW_MEM 慢速内存中，ULP 协处理器的地址 0 即相当于主 CPU 的地址 0x50000000。

30.4.4 JUMP –跳转至绝对地址

31 28	27 25	24 22	21		12 2	1 0
4’d8	3’b0	Type	Sel		ImmAddr	Rdst

图 30-8. 指令类型 - JUMP

Operand	**描述** - 见图 30-8
Rdst	寄存器 R[0-3]，储存需跳转至的目标地址
ImmAddr	11 位地址，单位为 32 位字
Sel	跳转目标地址来源： 0 - *ImmAddr* 存储的地址 1 - *Rdst* 存储的地址
Type	跳转类型： 0 - 无条件跳转 1 - 有条件跳转，仅当最后一次 ALU 运算设置了零标志位时跳转 2 - 有条件跳转，仅当最后一次 ALU 运算设置了溢出标志位时跳转

注意：

所有跳转地址均以 32 位字为单位。

描述

该指令可以让协处理器跳转至特定地址，跳转可以为无条件跳转或有条件跳转。

30.4.5 JUMPR –跳转至相对地址（基于 R0 寄存器判断）

31 28	27 25	24 17	16	15 0
4’d8	3’b1	Step	Cond	Threshold

图 30-9. 指令类型 - JUMPR

Operand	**描述** - 见图 30-9
Step	相对移位量，单位为 32 位字： 如果 Step[7] = 0，则 PC = PC + Step[6:0] 如果 Step[7] = 1，则 PC = PC - Step[6:0]
Threshold	跳转条件阈值（见下方 *Cond* ）
Cond	跳转条件： 0 - 如果 R0 < *Threshold*，即跳转 1 - 如果 R0 >= *Threshold*，即跳转

注意：

所有跳转地址均以 32 位字为单位。

描述

如果跳转条件（即比较 R0 寄存器的值与 *Threshold* 阈值）为真，该指令可以让协处理器跳转至 1 个相对地址。

30.4.6 JUMPS –跳转至相对地址（基于阶段计数器寄存器判断）

31 28	27 25	24 17	16 15	15	7 0
4’d8	3’b2	Step	Cond		Threshold

图 30-10. 指令类型 - JUMPS

Operand	**描述** - 见图 30-10
Step	相对位移量，单位为 32 位字： 如果 Step[7] = 0，则 PC = PC + Step[6:0] 如果 Step[7] = 1，则 PC = PC - Step[6:0]
Threshold	跳转条件阈值（见下方 *Cond*）
Cond	跳转条件： 1X - 如果 *Stage_cnt* <= *Threshold*，即跳转 00 - 如果 *Stage_cnt* < *Threshold*，即跳转 01 - 如果 *Stage_cnt* >= *Threshold*，即跳转

注意：

- 有关阶段计数器的相关设置，请见 30.4.1.3 ALU 阶段计数器章节。
- 所有跳转地址均以 32 位字为单位。

描述

如果跳转条件（即比较 *Stage_cnt* 阶段计数器寄存器的值与 *Threshold* 阈值）为真，该指令可以让协处理器跳转至 1 个相对地址。

30.4.7 HALT –结束程序

31 28	0
4’d11	

图 30-11. 指令类型 - HALT

描述

该指令可以让协处理器进入断电模式。

注意：

执行该指令后，ULP 协处理器的硬件计时器将开始计时。

30.4.8 WAKE –唤醒芯片

31 28	27 25		0
4’d9	3’b0		1’b1

图 30-12. 指令类型 - WAKE

描述

该指令可以让 ULP 协处理器向 RTC 控制器发送中断。

- 当 SoC 处于 Deep-sleep 模式时，该指令可唤醒 SoC。
- 当 SoC 处于 Deep-sleep 之外的模式时，如果 RTC_CNTL_INT_ENA_REG 寄存器设置了 RTC_CNTL_ULP_CP_INT_ENA 中断位，该指令即触发 RTC 中断。

30.4.9 SLEEP –设置硬件计时器的唤醒周期

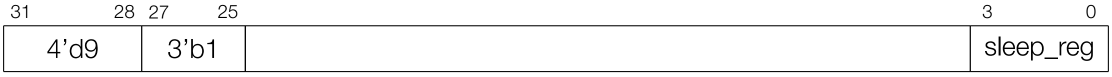


图 30-13. 指令类型 - SLEEP

Operand	**描述** - 见图 30-13
sleep_reg	在 5 个存储了不同唤醒周期的 SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) 寄存器中进行选择。

描述

该指令可选择 ULP 协处理器计时器的唤醒周期来源，即 SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) 寄存器中的 1 个。默认情况下，ULP 协处理器将采用 SENS_ULP_CP_SLEEP_CYC0_REG 中的值为唤醒周期。

30.4.10 WAIT –等待若干个周期

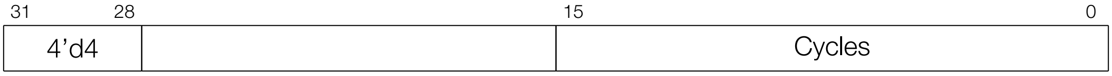


图 30-14. 指令类型 - WAIT

Operand	**描述** - 见图 30-14
Cycles	两次睡眠之间的等待周期数

描述

该指令可以设定协处理器两次睡眠之间的等待周期。

30.4.11 ADC –对 ADC 进行测量

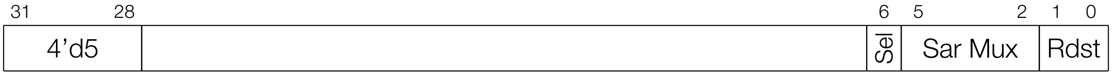


图 30-15. 指令类型 - ADC

Operand	**描述** - 见图 30-15
Rdst	目标地址寄存器 R[0-3]，将存储测量结果
Sel	选择 ADC：0 代表选择 SAR ADC1；1 代表选择 SAR ADC2，具体可见表 30-4。
Sar Mux	使能 SARADC 管脚 [Sar_Mux - 1]，见表 30-4。

表 30-4. ADC 指令的输入信号

管脚名 / 信号名 / GPIO	*Sar_Mux*	ADC 选择（*Sel*）
SENSOR_VP (GPIO36)	1	*Sel* = 0，选择 SAR ADC1
SENSOR_CAPP (GPIO37)	2	
SENSOR_CAPN (GPIO38)	3	
SENSOR_VN (GPIO39)	4	
32K_XP (GPIO33)	5	
32K_XN (GPIO32)	6	
VDET_1 (GPIO34)	7	
VDET_2 (GPIO35)	8	
GPIO4	1	*Sel* = 1，选择 SAR ADC2
GPIO0	2	

管脚名 / 信号名 / GPIO	*Sar_Mux*	ADC 选择（*Sel*）
GPIO2	3	
MTDO (GPIO15)	4	
MTCK (GPIO13)	5	
MTDI (GPIO12)	6	
MTMS (GPIO14)	7	
GPIO27	8	
GPIO25	9	
GPIO26	10	

描述

该指令可对 ADC 的信号进行测量。ADC 指令测量的管脚与信号见表 30-4。

30.4.12 I2C_RD / I2C_WR –读 / 写 I²C

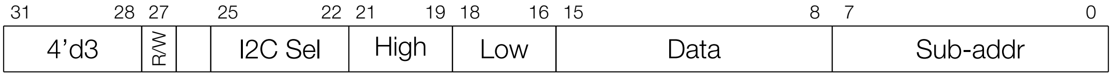


图 30-16. 指令类型 - I²C

Operand	**描述** - 见图 30-16
Sub-addr	从机的寄存器地址
Data	I2C_WR 运算中需写入的数据（但不会用于 I2C_RD 运算）
Low	位掩码的高位
High	位掩码的低位
I2C Sel	在 8 个存储 I²C 从机地址的寄存器 SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7) 中进行选择
R/W	I²C 通信类型： 1 - I²C 写 0 - I²C 读

描述

该指令可以支持与外部 I²C 从机进行通信（读/写），有关 RTC I²C 外围设备的使用，可见 30.6。

注意：

在主机模式下，RTC_I2C 可在 SCL 时钟的下降沿对 SDA 输入信号进行采样。

30.4.13 REG_RD –从外围寄存器读取

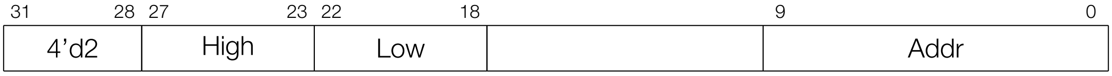


图 30-17. 指令类型 - REG_RD

Operand	**描述** - 见图 30-17
Addr	外围设备寄存器地址，单位为 32 位字
High	寄存器结束位
Low	寄存器开始位

描述

该指令可以从外围从机寄存器中读取最高 16 位的内容，并存入通用寄存器 R0。

R0 = REG[Addr][High:Low]

如需读取的内容超过 16 位，即 *High* - *Low* + 1 > 16，则该指令将返回 [Low+15:Low] 的内容。

注意：

- 该指令可访问 RTC_CNTL、RTC_IO、SENS 及 RTC_I2C 外围设备中的寄存器。ULP 协处理器可通过相同寄存器在 DPORT 总线上的地址，计算外围寄存器的地址，具体方式见下：

  addr_ulp = (addr_dport - DR_REG_RTCCNTL_BASE) / 4

- *addr_ulp* 以 32 位字（而非字节）为单位，0 可投射至 DR_REG_RTCCNTL_BASE（从主 CPU 的角度）。因此，10 位 ULP 协处理器的地址可覆盖外围寄存器空间的 4096 字节，包括 DR_REG_RTCCNTL_BASE、DR_REG_RTCIO_BASE、DR_REG_SENS_BASE 及 DR_REG_RTC_I2C_BASE 区域。

30.4.14 REG_WR –写入外围寄存器

31 28	27 23	22 18	17 10	9 0
4’d1	High	Low	Data	Addr

图 30-18. 指令类型 - REG_WR

Operand	**描述** - 见图 30-18
Addr	目标寄存器地址，单位为 32 位字
High	寄存器结束位
Low	寄存器开始位
Data	需写入的值，8 位数

描述

该指令可以向外设寄存器写入一个 8 位立即值 (Data)。

REG[Addr][High:Low] = Data

如需写入的内容超过 8 位，即 *High* - *Low* + 1 > 8，则该指令会给 8 位以上的内容填充 0。

注意：

有关 *addr_ulp* 的内容，请见 30.4.13。

30.5 ULP 协处理器程序的执行

ULP 协处理器经过专门设计，可独立于主 CPU 运行，无论后者是否处于 Deep-sleep 模式。

在典型场景中，为了降低功耗，系统可使主 CPU 进入 Deep-sleep 状态，而利用 ULP 协处理器进行必要操作。为了进一步降低功耗，ULP 协处理器自身也可以进入睡眠模式。在这种情况下，由于无法运行任何软件程序，因此系统配备了 1 个特殊的硬件计时器，可用于唤醒 ULP 协处理器。该硬件计时器必须提前进行配置，并在 5 个存储了不同唤醒周期的 SENS_ULP_CP_SLEEP_CYC*n*_REG 寄存器中进行选择。主 CPU 和 ULP 协处理器程序均可通过 REG_WR 和 SLEEP 指令进行相关设置。接着，系统可通过设置 RTC_CNTL_STATE0_REG 寄存器中的 RTC_CNTL_ULP_CP_SLP_TIMER_EN 位，启动 ULP 协处理器计时器。

ULP 协处理器可通过执行 HALT 指令进入睡眠状态，这也将同时触发 ULP 协处理器的硬件计时器 RTC_SLOW_CLK 开始统计时钟数（默认状态下，来自内部的 150 kHz RC 振荡器）。计时器一旦到期，ULP 协

处理器将上电并通过存储在 SENS_PC_INIT 中的 PC 启动。上述信号与寄存器之间的关系可见图 30-19。

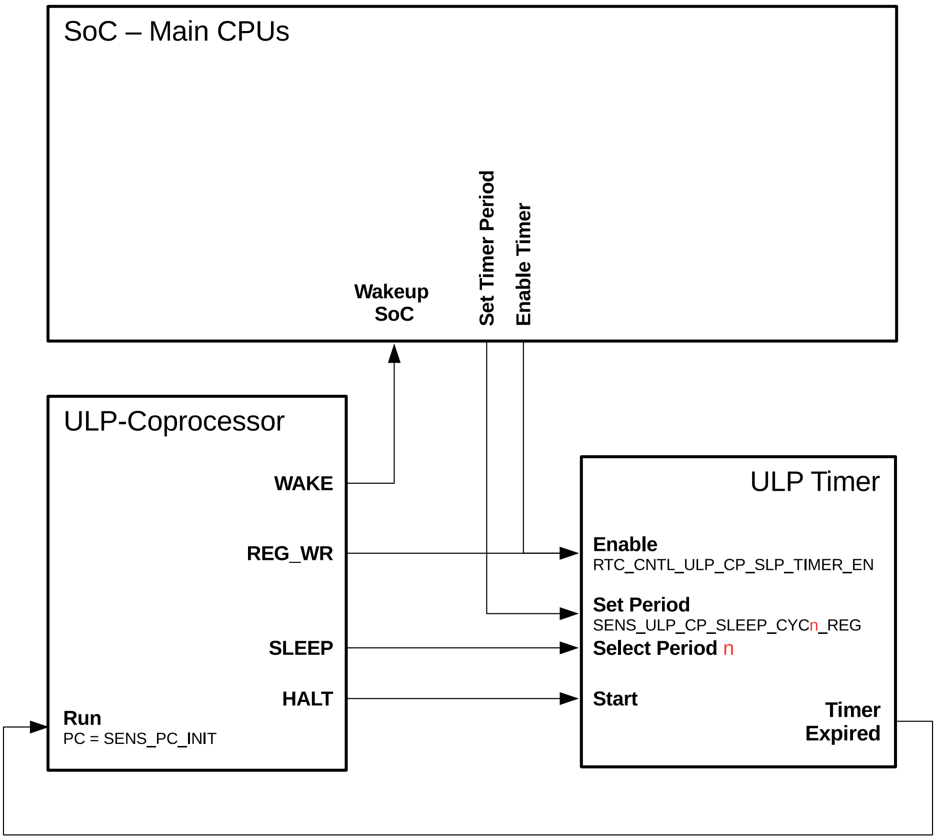


图 30-19. ULP 协处理器程序框图

复位和上电瞬间，ULP 协处理器程序仅会在 ULP 计时器的默认周期 SENS_ULP_CP_SLEEP_CYC0_REG 过期后启动。

ULP 协处理器程序的运行顺序示例可见图 30-20，其中具体步骤包括：

1. 软件通过 RTC_CNTL_ULP_CP_SLP_TIMER_EN 位启动 ULP 计时器。
2. ULP 计时器过期，ULP 协处理器开始从 PC = SENS_PC_INIT 处运行程序。
3. 执行 HALT 指令，ULP 程序停止运行，ULP 计时器再次启动。
4. 执行 SLEEP 指令，修改睡眠计时器周期寄存器。
5. ULP 协处理器程序或软件通过 RTC_CNTL_ULP_CP_SLP_TIMER_EN 位，关闭 ULP 协处理器计时器。

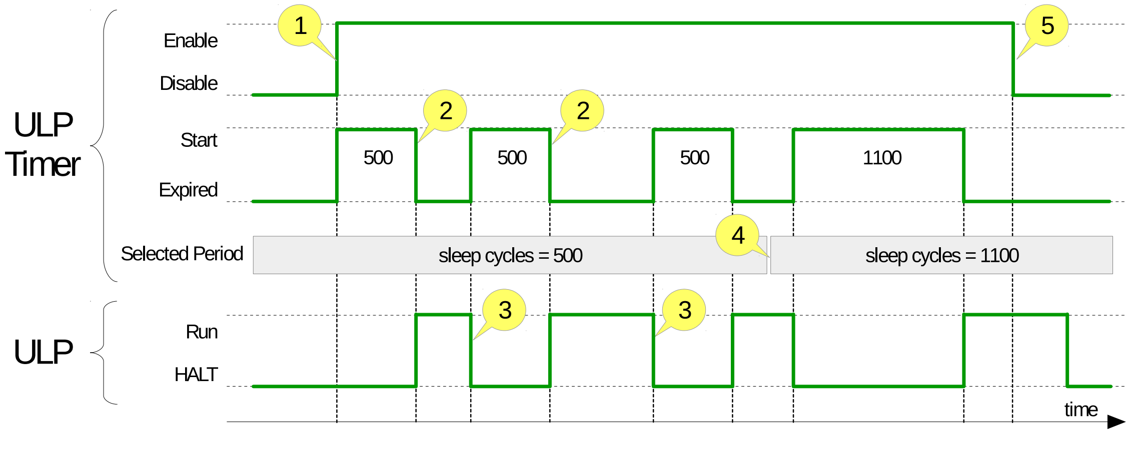


图 30-20. ULP 协处理器程序流控图

ULP 协处理器唤醒、执行程序和睡眠的具体时序由 ULP FSM 控制，具体为：

1. ULP 计时器过期时，FSM 唤醒 ULP 协处理器，这个过程持续两个时钟周期。
2. ULP 协处理器程序执行之前，FSM 等待 RTC_CNTL_TIMER2_REG 寄存器中 RTC_CNTL_ULPCP_TOUCH_START_WAIT 设置的若干个时钟周期，这个过程是为了等待 8 MHz 时钟状态稳定。
3. ULP 协处理器程序执行。
4. 调用 HALT 指令后，ULP 协处理器程序执行停止。FSM 还需要两个时钟周期来使 ULP 协处理器进入睡眠状态。

30.6 RTC_I2C 控制器

ULP 协处理器可使用 1 款位于 RTC 域的独立 I^2C 控制器，与外部 I^2C 从机进行通信。与 I2C0 / I2C1 外围设备相比，RTC_I2C 的功能集相对有限。

30.6.1 配置 RTC_I2C

ULP 协处理器在正常使用 I^2C 指令之前必须配置 RTC_I2C 中的特定参数，可通过某主 CPU 或 ULP 协处理器本身运行程序完成，具体即向 RTC_I2C 寄存器写入特定计时参数。

1. 通过 RTC_I2C_SCL_LOW_PERIOD_REG 和 RTC_I2C_SCL_HIGH_PERIOD_REG 设置 RTC_FAST_CLK 周期中 SCL 时钟的高低电平宽度和周期（例，频率为 100 kHz 时，设置 RTC_I2C_SCL_LOW_PERIOD = 40、RTC_I2C_SCL_HIGH_PERIOD = 40）。
2. 通过 RTC_FAST_CLK 中的 RTC_I2C_SDA_DUTY_REG 设置 SDA 切换前等待的周期数（例，RTC_I2C_SDA_DUTY=16）。
3. 通过 RTC_I2C_SCL_START_PERIOD_REG 设置启动信号后的等待时间（例，RTC_I2C_SCL_START_PERIOD = 30）。
4. 通过 RTC_I2C_SCL_STOP_PERIOD_REG 设置停止信号前的等待时间（例，RTC_I2C_SCL_STOP_PERIOD = 44）。
5. 通过 RTC_I2C_TIMEOUT_REG 设置通信超时参数（例，RTC_I2C_TIMEOUT = 200）。

6. 通过 RTC_I2C_CTRL_REG 中的 RTC_I2C_MS_MODE 位启动主机模式。
7. 将外部从机的地址写入 SENS_I2C_SLAVE_ADDR*n* (*n*: 0-7)，最多可通过这种方式预编程 8 个从机地址。此后，可为每次通信选择 1 个上述地址，共同组成协处理器 I2C 指令。

完成上述 RTC_I2C 配置后，即可使用 I2C_RD / I2C_WR –读/写 I^{2}C 指令。

30.6.2 使用 RTC_I2C

ULP 协处理器的 2 个指令集（采用同 1 个 OpCode）均可使用 RTC_I2C 控制器：I2C_RD （读）和 I2C_WR（写），详见 I2C_RD / I2C_WR –读/写 I^{2}C 章节。

30.6.2.1 I2C_RD - 读取单个字节

I2C_RD 指令的执行步骤如下，见图 30-21：

1. 主机发送启动信号。
2. 主机发送命令字节，包括从机地址和读/写控制位（此时，读/写控制位置为 0，代表“写”）。从机地址可从 SENS_I2C_SLAVE_ADDR*n* 中获取，其中寄存器的具体选择可通过 I2C_RD 设置。
3. 从机发送应答信号。
4. 主机发送从机寄存器地址，其中寄存器的具体选择可通过 I2C_RD 设置。
5. 从机发送应答信号。
6. 主机发送重复启动信号。
7. 主机发送从机地址，其中读/写控制位置为 1，代表“读”。
8. 从机发送 1 个字节的数据。
9. 主机发送非应答信号。
10. 主机发送停止信号，结束读取。

	1	2	3	4	5	6	7	8	9	10
Master	START	Slave Address W		Reg Address		RSTRT	Slave Address R		NACK	STOP
Slave			ACK		ACK			Data		

图 30-21. I2C 读操作

注意：

RTC_I2C 控制器外围设备会对 SCL 时钟下降沿上的 SDA 信号进行采样。如果从机的 SDA 信号在约 0.38 ms 内发生改变，主机则将接收到不正确的数据。

从机接收到的字节将存储在 R0 寄存器中。

30.6.2.2 I2C_WR - 写入单个字节

I2C_WR 指令的执行步骤如下，见图 30-22：

1. 主机发送开始信号。

2. 主机发送命令字节，包括从机地址和读/写控制位（此时，读/写控制位置为 0，代表 “写”）。从机地址可从 SENS_I2C_SLAVE_ADDR*n* 中获取，其中寄存器的具体选择可通过 I2C_WR 设置。
3. 从机发送应答信号。
4. 主机发送从机寄存器地址，其中寄存器的具体选择可通过 I2C_WR 设置。
5. 从机发送应答信号。
6. 主机发送重复启动信号。
7. 主机发送从机地址，其中读/写位置为 0，代表 “写”。
8. 主机发送 1 个字节的数据。
9. 从机发送应答信号。
10. 主机发送停止信号，结束写入。

图 30-22. I^{2}C 写操作

30.6.2.3 检测错误条件

ULP 协处理器指令 I2C_RD 和 I2C_WR 不会通过寄存器报告从机 NACK 等错误条件。相反，应用程序可以通过查询 RTC_I2C_INT_ST_REG 寄存器中的特定位，判断指令是否成功执行。为了检查特定的通信活动，RTC_I2C_INT_EN_REG 寄存器中的相应位应进行设置。注意，系统位图将移 1。如果检测到特定通信活动，且设置了 RTC_I2C_INT_ST_REG 寄存器，则可通过 RTC_I2C_INT_CLR_REG 寄存器清零。

30.6.2.4 连接 I^{2}C 信号

SDA 和 SCL 时钟信号可通过配置 RTCIO_SAR_I2C_IO_REG 寄存器，连接至 2 个 GPIO 管脚（共 4 个），详细定义请见章节 *IO_MUX 和 GPIO 交换矩阵*中的表 *RTC_MUX 管脚清单*。

30.7 寄存器列表

30.7.1 SENS_ULP 地址空间

名称	描述	地址	访问类型
ULP 计时器周期选择			
SENS_ULP_CP_SLEEP_CYC0_REG	计时器周期设置 0	0x3FF48818	读/写
SENS_ULP_CP_SLEEP_CYC1_REG	计时器周期设置 1	0x3FF4881C	读/写
SENS_ULP_CP_SLEEP_CYC2_REG	计时器周期设置 2	0x3FF48820	读/写
SENS_ULP_CP_SLEEP_CYC3_REG	计时器周期设置 3	0x3FF48824	读/写
SENS_ULP_CP_SLEEP_CYC4_REG	计时器周期设置 4	0x3FF48828	读/写
RTC I^{2}C 从设备地址选择			
SENS_SAR_SLAVE_ADDR1_REG	I^{2}C 地址 0 和 1	0x3FF4883C	读/写
SENS_SAR_SLAVE_ADDR2_REG	I^{2}C 地址 2 和 3	0x3FF48840	读/写

SENS_SAR_SLAVE_ADDR3_REG	I^{2}C 地址 4 和 5	0x3FF48844	读/写
SENS_SAR_SLAVE_ADDR4_REG	I^{2}C 地址 6 和 7，I^{2}C 控制	0x3FF48848	读/写
RTC I^{2}C 控制			
SENS_SAR_I2C_CTRL_REG	I^{2}C 控制寄存器	0x3FF48850	读/写

30.7.2 RTC_I2C 地址空间

名称	描述	地址	访问类型
RTC I^{2}C 控制寄存器			
RTC_I2C_CTRL_REG	传输设置	0x3FF48C04	读/写
RTC_I2C_DEBUG_STATUS_REG	调试状态	0x3FF48C08	读/写
RTC_I2C_TIMEOUT_REG	超时设置	0x3FF48C0C	读/写
RTC_I2C_SLAVE_ADDR_REG	本地从设备地址设置	0x3FF48C10	读/写
RTC I^{2}C 信号设置寄存器			
RTC_I2C_SDA_DUTY_REG	配置 SCL 下降沿后的 SDA 保持时间	0x3FF48C30	读/写
RTC_I2C_SCL_LOW_PERIOD_REG	配置 SCL 时钟的低电平宽度	0x3FF48C00	读/写
RTC_I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平宽度	0x3FF48C38	读/写
RTC_I2C_SCL_START_PERIOD_REG	配置开始条件下，SDA 与 SCL 下降沿之间的延迟	0x3FF48C40	读/写
RTC_I2C_SCL_STOP_PERIOD_REG	配置停止条件下，SDA 与 SCL 下降沿之间的延迟	0x3FF48C44	读/写
RTC I^{2}C 中断寄存器 - 仅用于调试目的			
RTC_I2C_INT_CLR_REG	清除 I^{2}C 通信活动状态	0x3FF48C24	读/写
RTC_I2C_INT_EN_REG	开始捕捉 I^{2}C 通信状态活动	0x3FF48C28	读/写
RTC_I2C_INT_ST_REG	捕捉 I^{2}C 通信活动的状态	0x3FF48C2C	只读

注意：

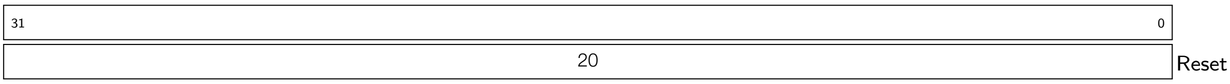
来自 RTC_I2C 控制器的中断暂时尚未连接，以上中断寄存器仅用于调试目的。

30.8 寄存器

30.8.1 SENS_ULP 地址空间

本小节括号中的地址均为相对于 (RTC 基地址 + 0x0800) 的地址偏移量（相对地址）。RTC 基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 30.7.1 *SENS_ULP 地址空间*。

Register 30.1. SENS_ULP_CP_SLEEP_CYC*n*_REG (*n*: 0-4) (0x18+0x4n*)**



SENS_ULP_CP_SLEEP_CYC*n*_REG ULP 计时器周期设置 *n*，ULP 协处理器可通过指令 SLEEP 在上述寄存器中进行选择。（读/写）

Register 30.2. SENS_SAR_START_FORCE_REG (0x002c)

(reserved)	SENS_PC_INIT	(reserved)	SENS_ULP_CP_START_TOP	SENS_ULP_CP_FORCE_START_TOP	(reserved)	
31 22	21 11	10	9	8	7 0	
0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0	0	0	0	0 0 0 0 0 0 0 0	Reset

SENS_PC_INIT ULP PC 入口地址。(读/写)

SENS_ULP_CP_START_TOP 启动 ULP 协处理器，仅当 SENS_ULP_CP_FORCE_START_TOP = 1 时有效。(读/写)

SENS_ULP_CP_FORCE_START_TOP 1：ULP 协处理器由 SENS_ULP_CP_START_TOP 启动；0：ULP 协处理器由硬件计时器启动。(读/写)

Register 30.3. SENS_SAR_SLAVE_ADDR1_REG (0x003c)

(reserved)	SENS_I2C_SLAVE_ADDR0	SENS_I2C_SLAVE_ADDR1	
31 22	21 11	10 0	
0 0 0 0 0 0 0 0 0 0	0x000	0x000	Reset

SENS_I2C_SLAVE_ADDR0 I^{2}C 从机地址 0。(读/写)

SENS_I2C_SLAVE_ADDR1 I^{2}C 从机地址 1。(读/写)

Register 30.4. SENS_SAR_SLAVE_ADDR2_REG (0x0040)

(reserved)	SENS_I2C_SLAVE_ADDR2	SENS_I2C_SLAVE_ADDR3	
31 22	21 11	10 0	
0 0 0 0 0 0 0 0 0 0	0x000	0x000	Reset

SENS_I2C_SLAVE_ADDR2 I^{2}C 从机地址 2。(读/写)

SENS_I2C_SLAVE_ADDR3 I^{2}C 从机地址 3。(读/写)

Register 30.5. SENS_SAR_SLAVE_ADDR3_REG (0x0044)

(reserved)	SENS_I2C_SLAVE_ADDR4	SENS_I2C_SLAVE_ADDR5	
31 22	21 11	10 0	
0 0 0 0 0 0 0 0 0 0	0x000	0x000	Reset

SENS_I2C_SLAVE_ADDR4 I^{2}C 从机地址 4。（读/写）

SENS_I2C_SLAVE_ADDR5 I^{2}C 从机地址 5。（读/写）

Register 30.6. SENS_SAR_SLAVE_ADDR4_REG (0x0048)

(reserved)	SENS_I2C_DONE	SENS_I2C_RDATA	SENS_I2C_SLAVE_ADDR6	SENS_I2C_SLAVE_ADDR7	
31	30	29 22	21 11	10 0	
0	0	0x000	0x000	0x000	Reset

SENS_I2C_DONE 示意 I^{2}C 已完成。（只读）

SENS_I2C_RDATA I^{2}C 读取数据。（只读）

SENS_I2C_SLAVE_ADDR6 I^{2}C 从机地址 6。（读/写）

SENS_I2C_SLAVE_ADDR7 I^{2}C 从机地址 7。（读/写）

Register 30.7. SENS_SAR_I2C_CTRL_REG (0x0050)

(reserved)	SENS_SAR_I2C_START_FORCE	SENS_SAR_I2C_START	SENS_SAR_I2C_CTRL	
31 30	29	28	27 0	
0 0	0	0	0 0 0 0 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

SENS_SAR_I2C_START_FORCE 1：I^{2}C 由软件启动；0：I^{2}C 由 FSM 启动。（读/写）

SENS_SAR_I2C_START 启动 I^{2}C，仅当 SENS_SAR_I2C_START_FORCE = 1 时有效。（读/写）

SENS_SAR_I2C_CTRL I^{2}C 控制数据，仅当 SENS_SAR_I2C_START_FORCE = 1 时有效。（读/写）

30.8.2 RTC_I2C 地址空间

本小节括号中的地址均为相对于 (RTC 基地址 + 0x0C00) 的地址偏移量（相对地址）。RTC 基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 30.7.2 *RTC_I2C 地址空间*。

Register 30.8. RTC_I2C_SCL_LOW_PERIOD_REG (0x000)

(reserved)	RTC_I2C_SCL_LOW_PERIOD	
31 … 19	18 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_I2C_SCL_LOW_PERIOD SCL 时钟信号处于低电平时的周期数。(读/写)

Register 30.9. RTC_I2C_CTRL_REG (0x004)

(reserved)	RTC_I2C_RX_LSB_FIRST	RTC_I2C_TX_LSB_FIRST	RTC_I2C_TRANS_START	RTC_I2C_MS_MODE	(reserved)	RTC_I2C_SCL_FORCE_OUT	RTC_I2C_SDA_FORCE_OUT	
31 … 8	7	6	5	4	3 … 2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0	0	0	Reset

RTC_I2C_RX_LSB_FIRST 优先接收 LSB。(读/写)

RTC_I2C_TX_LSB_FIRST 优先发送 LSB。(读/写)

RTC_I2C_TRANS_START 强制产生开始条件。(读/写)

RTC_I2C_MS_MODE 选择模式：1 代表主机模式；0 代表从机模式。(读/写)

RTC_I2C_SCL_FORCE_OUT SCL 输出模式：1 代表推挽输出；0 代表开漏输出。(读/写)

RTC_I2C_SDA_FORCE_OUT SDA 输出模式：1 代表推挽输出；0 代表开漏输出。(读/写)

Register 30.10. RTC_I2C_DEBUG_STATUS_REG (0x008)

(reserved)	RTC_I2C_SCL_STATE	RTC_I2C_MAIN_STATE	(reserved)	RTC_I2C_BYTE_TRANS	RTC_I2C_SLAVE_ADDR_MATCH	RTC_I2C_BUS_BUSY	RTC_I2C_ARB_LOST	RTC_I2C_TIMED_OUT	RTC_I2C_SLAVE_RW	RTC_I2C_ACK_VAL	
31	30 28	27 25	24 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

RTC_I2C_SCL_STATE SCL 状态机的状态。(读/写)

RTC_I2C_MAIN_STATE 主状态机的状态。(读/写)

RTC_I2C_BYTE_TRANS 示意 8 位传输已完成。(读/写)

RTC_I2C_SLAVE_ADDR_MATCH 从机模式下，判断地址是否相符。(读/写)

RTC_I2C_BUS_BUSY 示意运算正在进行中。(读/写)

RTC_I2C_ARB_LOST 主机模式下，失去 I²C 总线控制。(读/写)

RTC_I2C_TIMED_OUT I²C 通信已超时。(读/写)

RTC_I2C_SLAVE_RW 从机模式下，接收到的读/写位的值。(读/写)

RTC_I2C_ACK_VAL 总线上应答信号的值。(读/写)

Register 30.11. RTC_I2C_TIMEOUT_REG (0x00c)

(reserved)	RTC_I2C_TIMEOUT	
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	Reset

RTC_I2C_TIMEOUT 传输可以接收的 RTC_FAST_CLK 最大周期数。(读/写)

Register 30.12. RTC_I2C_SLAVE_ADDR_REG (0x010)

RTC_I2C_SLAVE_ADDR_10BIT	(reserved)	RTC_I2C_SLAVE_ADDR	
31	30 15	14 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_I2C_SLAVE_ADDR_10BIT 当本地从机地址为 10 位时设置。(读/写)

RTC_I2C_SLAVE_ADDR 本地从机地址。(读/写)

Register 30.13. RTC_I2C_INT_CLR_REG (0x024)

(reserved)	RTC_I2C_TIME_OUT_INT_CLR	RTC_I2C_TRANS_COMPLETE_INT_CLR	RTC_I2C_MASTER_TRANS_COMPLETE_INT_CLR	RTC_I2C_ARBITRATION_LOST_INT_CLR	RTC_I2C_SLAVE_TRANS_COMPLETE_INT_CLR	(reserved)	
31 9	8	7	6	5	4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0 0 0 0	Reset

RTC_I2C_TIME_OUT_INT_CLR 清除超时中断。(读/写)

RTC_I2C_TRANS_COMPLETE_INT_CLR 清除停止模式监测中断。(读/写)

RTC_I2C_MASTER_TRANS_COMPLETE_INT_CLR 主机模式下,清除动作完成中断。(读/写)

RTC_I2C_ARBITRATION_LOST_INT_CLR 主机模式下,清除失去总线控制中断。(读/写)

RTC_I2C_SLAVE_TRANS_COMPLETE_INT_CLR 从机模式下,清除动作完成中断。(读/写)

Register 30.14. RTC_I2C_INT_EN_REG (0x028)

(reserved)	RTC_I2C_TIME_OUT_INT_ENA	RTC_I2C_TRANS_COMPLETE_INT_ENA	RTC_I2C_MASTER_TRAN_COMP_INT_ENA	RTC_I2C_ARBITRATION_LOST_INT_ENA	(reserved)	
31 9	8	7	6	5	4 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0 0 0 0	Reset

RTC_I2C_TIME_OUT_INT_ENA 开启超时中断。（读/写）

RTC_I2C_TRANS_COMPLETE_INT_ENA 开启停止模式监测中断。（读/写）

RTC_I2C_MASTER_TRAN_COMP_INT_ENA 主机模式下，开启动作完成中断。（读/写）

RTC_I2C_ARBITRATION_LOST_INT_ENA 主机模式下，开启失去总线控制中断。（读/写）

Register 30.15. RTC_I2C_INT_ST_REG (0x02c)

(reserved)	RTC_I2C_TIME_OUT_INT_ST	RTC_I2C_TRANS_COMPLETE_INT_ST	RTC_I2C_MASTER_TRAN_COMP_INT_ST	RTC_I2C_ARBITRATION_LOST_INT_ST	(reserved)	
31 8	7	6	5	4	3 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0 0 0 0	Reset

RTC_I2C_TIME_OUT_INT_ST 监测到的超时。（只读）

RTC_I2C_TRANS_COMPLETE_INT_ST 监测到的 I²C 总线停止模式。（只读）

RTC_I2C_MASTER_TRAN_COMP_INT_ST 主机模式下，完成动作。（只读）

RTC_I2C_ARBITRATION_LOST_INT_ST 主机模式下，总线失去控制。（只读）

Register 30.16. RTC_I2C_SDA_DUTY_REG (0x030)

(reserved)	RTC_I2C_SDA_DUTY	
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	Reset

RTC_I2C_SDA_DUTY SCL 下降沿与 SDA 切换之间的 RTC_FAST_CLK 周期数。(读/写)

Register 30.17. RTC_I2C_SCL_HIGH_PERIOD_REG (0x038)

(reserved)	RTC_I2C_SCL_HIGH_PERIOD	
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	Reset

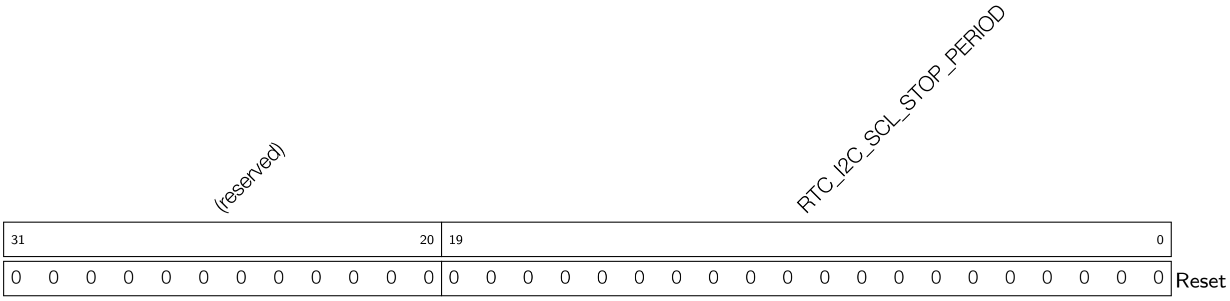
RTC_I2C_SCL_HIGH_PERIOD SCL 为高的 RTC_FAST_CLK 周期数。(读/写)

Register 30.18. RTC_I2C_SCL_START_PERIOD_REG (0x040)

(reserved)	RTC_I2C_SCL_START_PERIOD	
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	Reset

RTC_I2C_SCL_START_PERIOD 产生开始条件前，需要等待的 RTC_FAST_CLK 周期数。(读/写)

Register 30.19. RTC_I2C_SCL_STOP_PERIOD_REG (0x044)



RTC_I2C_SCL_STOP_PERIOD 产生停止条件前，需要等待的 RTC_FAST_CLK 周期数。（读/写）

31 低功耗管理 (RTC_CNTL)

31.1 概述

ESP32 采用了高效、灵活的功耗管理技术，可以在功耗控制、唤醒延迟和不同唤醒源之间实现最佳平衡。芯片的主处理器支持 5 种功耗模式，可以满足用户的不同场景需求。此外，ESP32 还支持超低功耗协处理器 (ULP Co-processor) 控制，允许主处理器进入 Deep-sleep 模式，从而最大程度降低功耗，支持用户的低功耗应用。

31.2 主要特性

- 支持 5 种预设功耗模式，适用于多种应用场景
- 支持高达 16 KB 保留内存 (Retention Memory)
- 8 x 32-bit 保留寄存器 (Retention Register)
- 所有低功耗模式均支持 ULP 协处理器控制
- 支持 RTC boot 功能，可缩短唤醒延迟

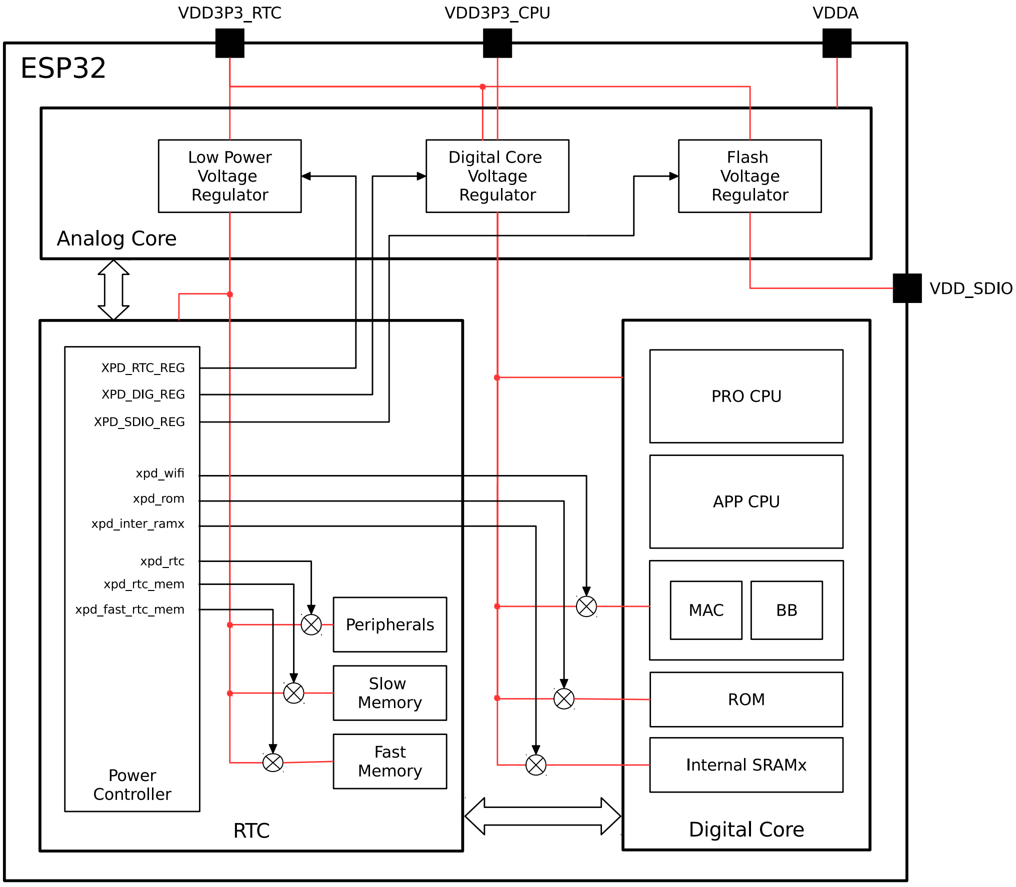


图 31-1. ESP32 功耗控制示意图

31.3 功能描述

31.3.1 简介

ESP32 的低功耗管理单元包括调压器、功耗控制器、电源开关单元、电源域隔离单元 (Isolation Cell) 等部分，具体结构示意图可见上方图 31-1。

31.3.2 数字内核调压器

ESP32 的内置数字内核调压器可以将外部电源电压（通常为 3.3V）转换为 1.1V，支持数字内核的正常工作。该调压器可以接受的外部输入电压范围为 1.8V 到 3.6V，输出电压范围为 0.90V 到 1.25V。具体结构示意图可见下方图 31-2。

1. 当 XPD_DIG_REG == 1 时，该调压器的输出电压为 1.1V，数字内核可以正常工作；当 XPD_DIG_REG == 0 时，调压器和数字内核均无法工作。
2. DIG_REG_DBIAS[2:0] 可以调节数字内核的供电电压：

$$\mathrm{VDD\_DIG} = 0.90 + \mathrm{DBIAS} \cdot 0.05\mathrm{V}$$

3. 流入数字内核的电流来自管脚 VDD3P3_CPU 和 VDD3P3_RTC。

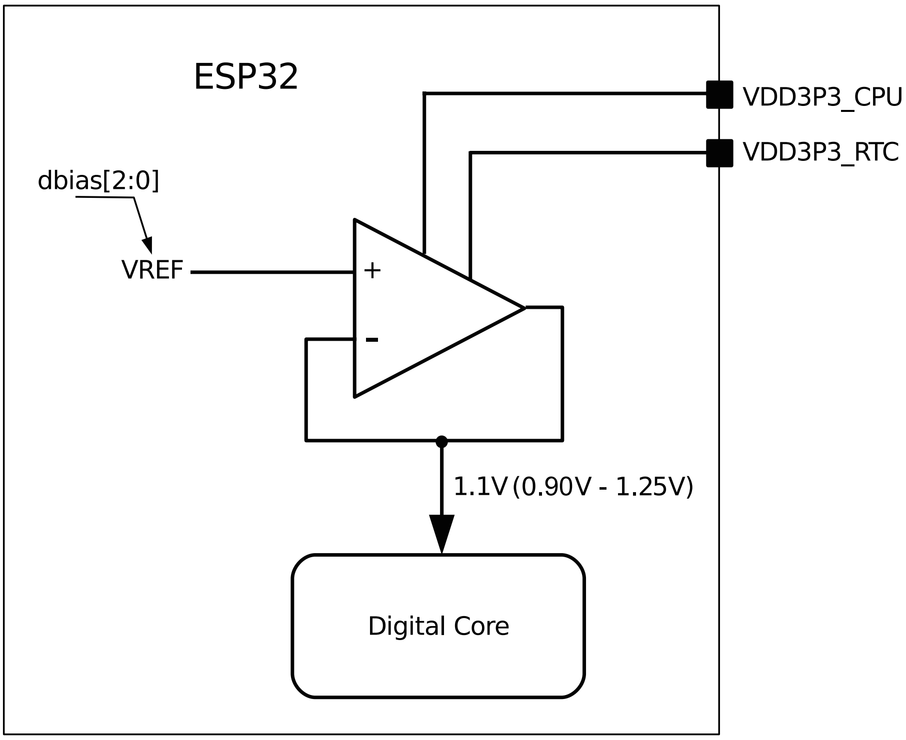


图 31-2. 数字内核调压器

31.3.3 低功耗调压器

ESP32 的内置低功耗调压器可以将外部电源电压（通常为 3.3V）转换为 1.1V，支持内部 RTC 数字内核的正常工作。为了降低功耗，该调压器可以接受的外部输入电压范围为 1.8V 到 3.6V，输出电压范围可调：正常工作模式下的电压输出范围为 0.90V 到 1.25V；Deep-sleep 和睡眠模式下的电压输出固定为 0.75V；休眠模式下的电压输出更低。具体结构示意图可见下方图 31-3。

1. 当管脚 CHIP_PU 为高电平时，低功耗调压器无法关闭，仅能在正常和 Deep-sleep 模式之间进行切换。
2. 在正常模式下，RTC_DBIAS[2:0] 可以调节输出电压：

$$\mathrm{VDD\_RTC} = 0.90 + \mathrm{DBIAS} \cdot 0.05\mathrm{V}$$

3. 在 Deep-sleep 模式下，调压器的输出电压固定为 0.75V。
4. 流向 RTC 数字内核的电流来自管脚 VDD3P3_RTC。

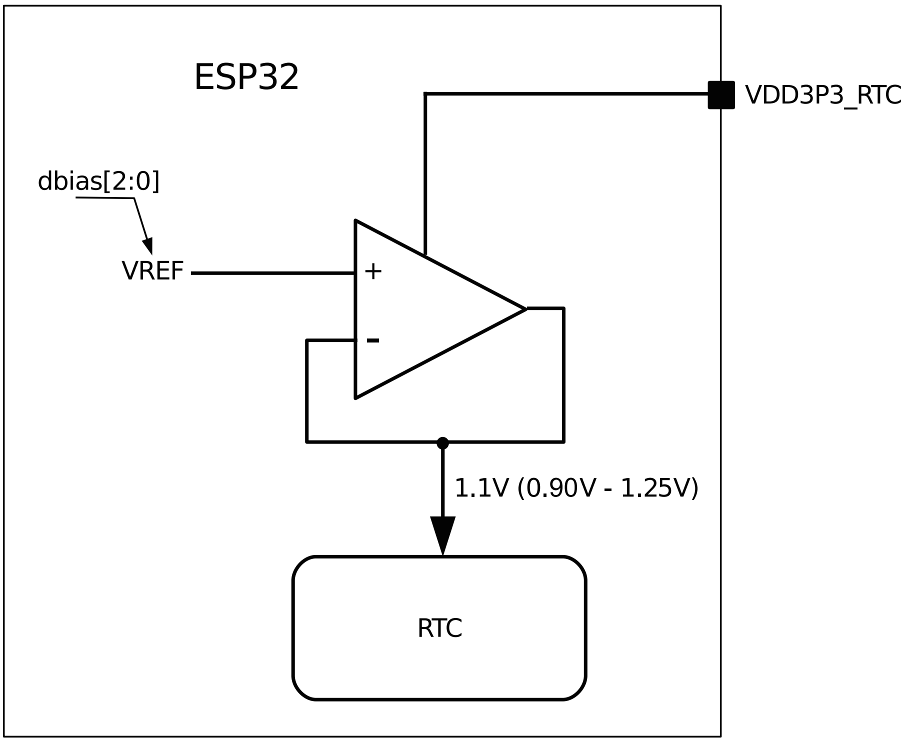


图 31-3. 低功耗调压器

31.3.4 Flash 调压器

ESP32 的内置 flash 调压器可以向系统中的其他设备输出 3.3V 或 1.8V 电压，比如 flash。该调压器的最大电流输出为 40 mA。具体结构示意图可见下方图 31-4。

1. 当 XPD_SDIO_VREG == 1 时，调压器的输出电压为 3.3V 或 1.8V；当 XPD_SDIO_VREG == 0 时，调压器的输出电压为高阻抗，电压由外部电源供应。
2. 当 SDIO_TIEH == 1 时，调压器将管脚 VDD_SDIO 和管脚 VDD3P3_RTC 短路，电压输出为 3.3V，即管脚 VDD3P3_RTC 的电压；当 SDIO_TIEH == 0 时，调压器的电压输出为参考电压 VREF，通常为 1.8V。
3. 更改 DREFH_SDIO、DREFM_SDIO 和 DREFL_SDIO 可以小幅调节参考电压 VREF，但这种操作可能会影响系统内环的稳定性，因此不推荐进行更改。
4. 当调压器输出为 3.3V 或 1.8V 时，输出电流来自管脚 VDD3P3_RTC。

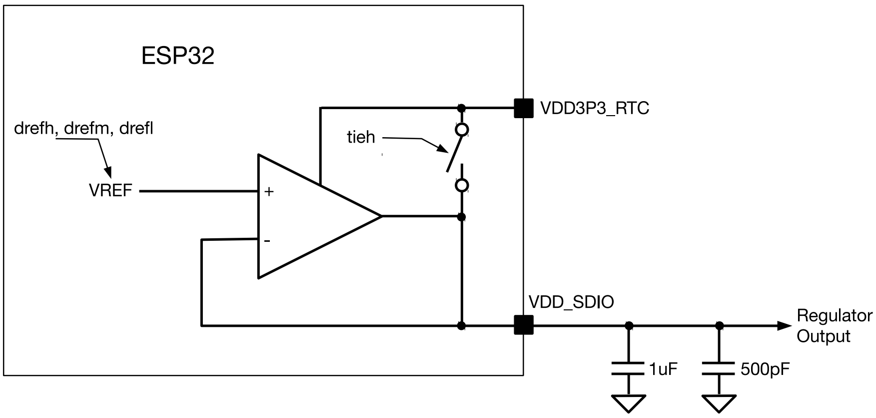


图 31-4. Flash 调压器

31.3.5 欠压检测器

ESP32 的欠压检测器可以检查管脚 VDD3P3_RTC 的电压。当电压快速下落至一定水平，欠压检测器将发出触发信号，关闭部分耗电模块（比如 LNA 和 PA 等），从而为数字模块争取更多时间，用以保存、转移重要数据。欠压检测器的功耗非常低，将在芯片开启时永远保持开启，具体的信号触发阈值可以调节，通常为 2.5V。具体结构示意图可见下方图 31-5。

1. RTC_CNTL_BROWN_OUT_DET 为欠压检测器的输出电平，将在管脚 VDD3P3_RTC 电压低于阈值时跳至高电平。
2. RTC_CNTL_DBROWN_OUT_THRES[2:0] 可用于调节欠压检测器的信号触发阈值，通常为 2.5V。

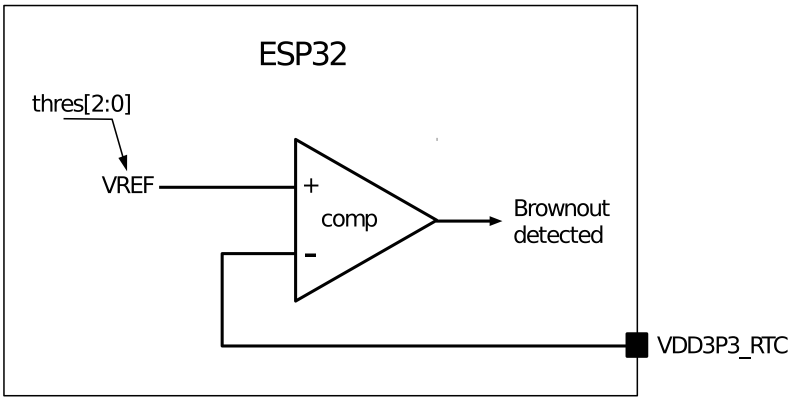


图 31-5. 欠压检测器

31.3.6 RTC 模块

ESP32 的 RTC 模块经过专门设计，可用于管理低功耗模式的进入和退出，控制时钟源、PLL、电源开关和隔离单元以产生电源门控、时钟门控和复位信号。RTC 模块的结构图可见图 31-6，主要部分包括：

- RTC 主状态机—记录电源状态。
- 数字 & 模拟电源控制器—可用于为 RTC 的数字模块和模拟模块生成电源门控 / 时钟门控信号。
- 睡眠 & 唤醒控制器—可处理低功耗模式的进入和退出。
- 计时器—包括 RTC 主计时器、ULP 协处理器计时器和触摸计时器。

- 低功耗处理器和传感器控制器—ULP 协处理器、触摸控制器、SAR ADC 控制器等。
- 保留内存
  - RTC 慢速内存—支持 8 KB SRAM，绝大部分用作保留内存或存储 ULP 协处理器的指令 & 数据内存。CPU 可通过 APB 总线访问慢速内存，起始地址为 0x50000000。
  - RTC 快速内存—支持 8 KB SRAM，绝大部分用作保留内存。CPU 可通过 IRAM0/DRAM0 访问快速内存。RTC 快速内存的速度约为 RTC 慢速内存的 10 倍。
- 保留寄存器—8 x 32 位。该寄存器永远开启，可用于数据存储。
- RTC IO 管脚—18 个 “always-on” 管脚，通常作为唤醒源。

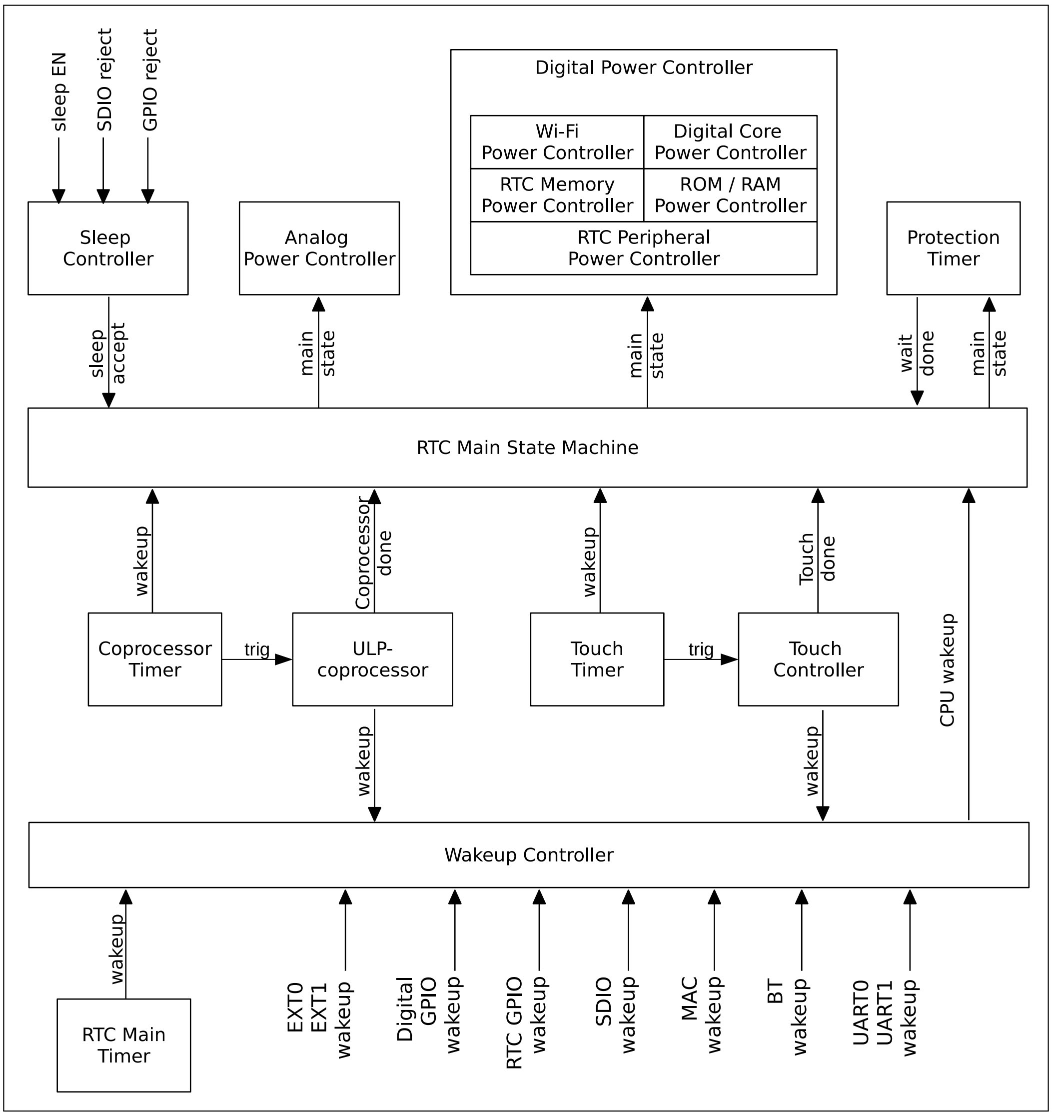


图 31-6. RTC 结构图

31.3.7 低功耗时钟

在低功耗模式下，ESP32 的 40 MHz 晶振和 PLL 通常将断电以降低功耗，但时钟仍将开启，以确保芯片在低功耗模式下的正常工作。

RTC 内核可以使用 5 个时钟源：

- 外部低速晶振时钟 XTL32K_CLK (32.768 kHz)
- 外部高速晶振时钟 XTAL_DIV_CLK (2 MHz ~ 40 MHz)
- 内部 RC 振荡器 RC_SLOW_CLK（频率可调，通常为 150 kHz）
- 内部 8 MHz 振荡器 RC_FAST_CLK
- 内部 31.25 kHz 时钟 RC_FAST_DIV_CLK（来自内部 8 MHz 振荡器，256 分频）

这些时钟可以产生 RTC_FAST_CLK 和 RTC_SLOW_CLK 。默认情况下，RTC_FAST_CLK 选择 RC_FAST_CLK，RTC_SLOW_CLK 选择 RC_SLOW_CLK，具体可见图 31-7。

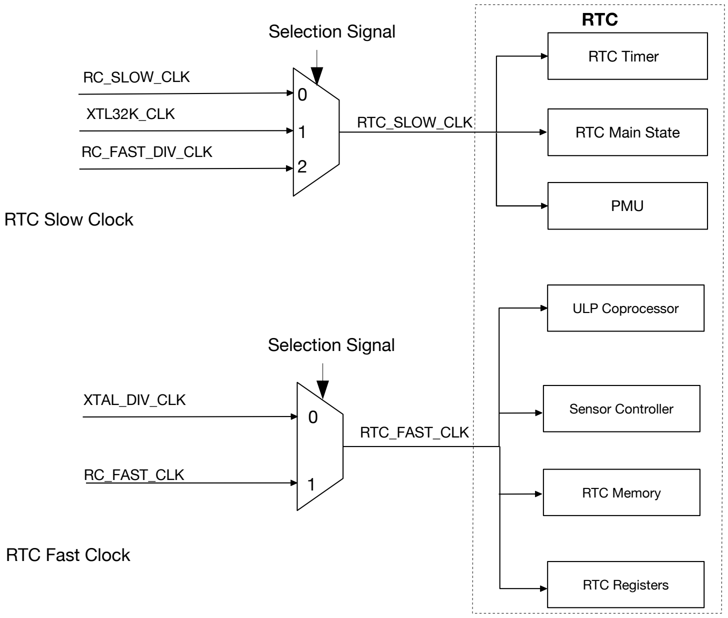


图 31-7. RTC 低功耗时钟

对于数字内核，LOW_POWER_CLK 可在 4 个时钟源之间切换，具体可见图 31-8。

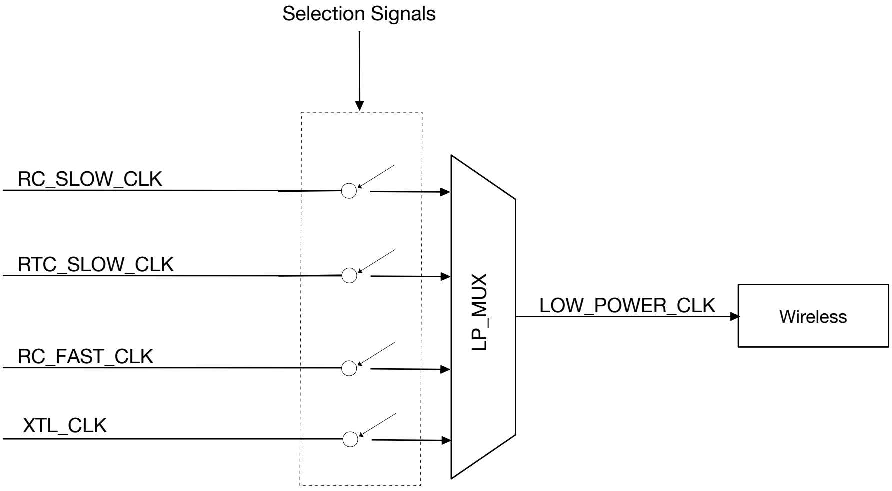


图 31-8. 数字低功耗时钟

31.3.8 电源门控的实现

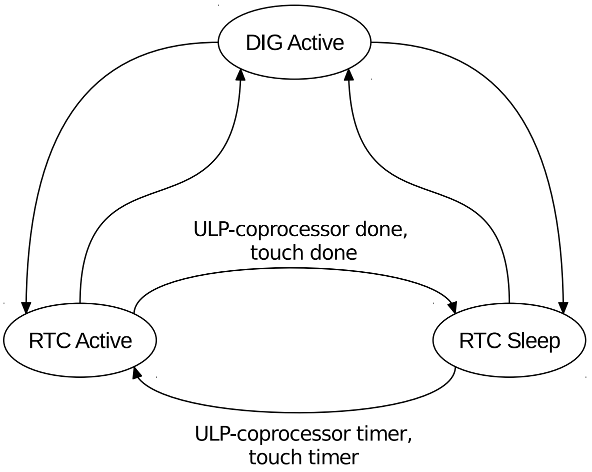


图 31-9. RTC 状态

电源门控转化图具体可见图 31-9。实际电源控制信号可通过软件设置为强制开启 (FPU) 或强制关闭 (FPD)。电源域支持独立电源门控，可针对多种不同应用场景进行具体组合。表 31-1 展示了 ESP32 电源域的控制方式。

表 31-1. RTC 电源域

电源域		RTC 主要状态			软件选项		注意 *
		DIG 活跃	RTC 活跃	RTC 睡眠	打开	关闭	
RTC	数字内核	ON	ON	ON	N	N	1
	RTC 外围	ON	ON	OFF	Y	Y	2
	RTC 慢速内存	ON	OFF	OFF	Y	Y	3
	RTC 快速内存	ON	OFF	OFF	Y	Y	4
数字	数字内核	ON	OFF	OFF	Y	Y	5
	Wi-Fi	ON	OFF	OFF	Y	Y	6
	ROM	ON	OFF	OFF	Y	Y	-
	内部 SRAM	ON	OFF	OFF	Y	Y	7
模拟	40 MHz 晶振	ON	OFF	OFF	Y	Y	-
	PLL	ON	OFF	OFF	Y	Y	-
	8 MHz 振荡器	ON	OFF	OFF	Y	Y	-
	射频	-	-	-	Y	Y	-

注意 *：

1. 电源域 RTC 数字内核为“always-on”电源域，FPU/FPD 选项不可用。

2. 电源域 RTC 外设内含 RTC 域的大多数快速逻辑模块，包括 ULP 协处理器、传感器控制器等。

3. 电源域 RTC 慢速内存在用作保留内存或在 ULP 协处理器工作时，应强制开启。

4. 电源域 RTC 快速内存在用作保留内存时，应强制开启。

5. 当电源域数字内核关闭时，电源域中的所有模块均关闭。

6. 电源域 Wi-Fi 包括 Wi-Fi MAC 和 BB。

7. 每个内部 SRAM 均支持独立电源门控。

31.3.9 预设功耗模式

ESP32 支持 5 种预设功耗模式，可以覆盖绝大多数应用场景。因此，用户在自行调整各个电源控制信号前，应首先尝试这 5 种功耗模式能否满足要求，具体包括：

- Active 模式
  - CPU 的工作时钟为 XTAL_DIV_N (40 MHz/26 MHz) 或 PLL (80 MHz/160 MHz/240 MHz)。
  - 芯片可以接收、发射或监听信号。
- Modem-sleep 模式
  - CPU 可以工作，时钟可以配置。
  - Wi-Fi ／蓝牙基带受时钟门限控制或关闭，射频模块关闭。
  - PLL 为 80 MHz 时，电流消耗：~30 mA。
  - XTAL 为 2 MHz 时，电流消耗：~3 mA。
  - 即刻唤醒。
- Light-sleep 模式
  - 内部 8 MHz 振荡器、40 MHz 高速晶振、PLL 及射频模块均禁用。
  - 数字内核时钟受门限限制，CPU 暂停工作。
  - ULP 协处理器和触摸控制器可以周期性触发，对传感器进行监测。
  - 电流消耗：~ 800 μA。
  - 唤醒延迟：低于 1 ms。
- Deep-sleep 模式
  - 内部 8 MHz 振荡器、40 MHz 高速晶振、PLL 及射频模块均禁用。
  - 数字内核断电，CPU 内容丢失。
  - RTC 内核的供电电压降至 0.7V。
  - 8 x 32 位数据保存在通用保留寄存器中。

  - RTC 内存和快速 RTC 内存可以保持。
  - 电流消耗：~ 6.5 μA。
  - 唤醒延迟：低于 1 ms。
  - 推荐用于一些非频繁连接 Wi-Fi / 蓝牙的超低功耗应用场景。
- 休眠模式
  - 内部 8 MHz 振荡器、40 MHz 高速晶振、PLL 及射频模块均禁用。
  - 数字内核断电，CPU 内容丢失。
  - RTC 外设域断电。
  - RTC 内核的供电电压降至 0.7V。
  - 8 x 32 位数据保存在通用保留寄存器中。
  - RTC 内存和快速 RTC 内存断电。
  - 电流消耗：~ 4.5 μA。
  - 唤醒源：仅支持 RTC 计时器。
  - 唤醒延迟：低于 1 ms。
  - 推荐用于一些非频繁连接 Wi-Fi / 蓝牙的超低功耗应用场景。

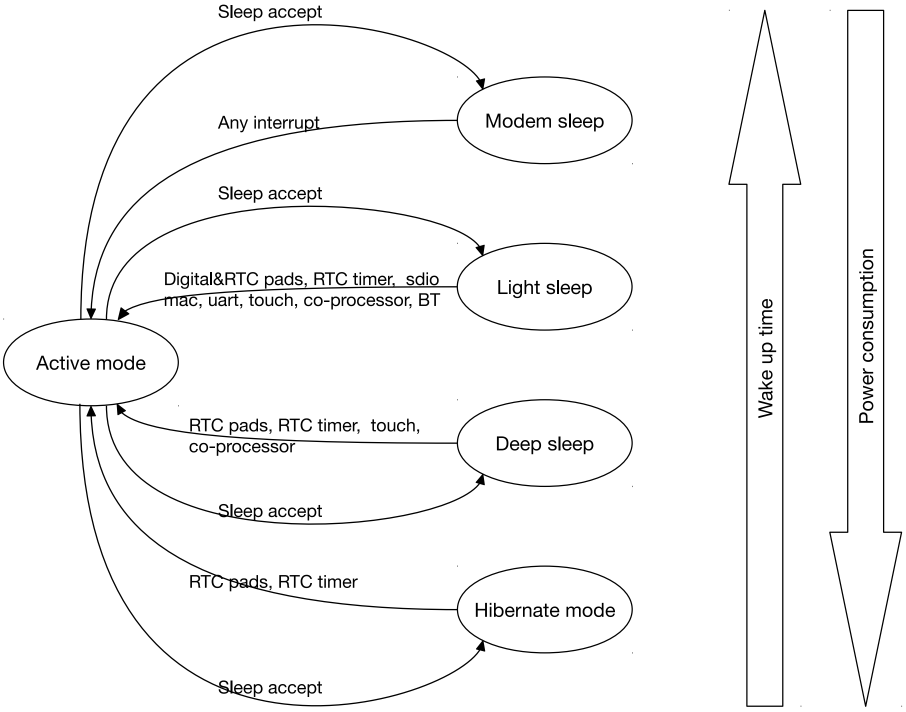


图 31-10. 功耗模式

默认情况下，ESP32 系统复位后将进入 Active 模式。当 CPU 不需要一直工作时，比如当等待外部活动唤醒时，系统可以进入多种低功耗模式。用户可根据具体功耗、唤醒延迟和可用唤醒源需求，在各种功耗模式中进行选择。详见图 31-10。

值得一提的是，ESP32 的预设功耗模式可支持进一步优化，以适应不同的应用场景。

31.3.10 唤醒源

ESP32 可支持多种唤醒源，可将 CPU 从不同睡眠模式中唤醒。唤醒源的选择由 RTC_CNTL_WAKEUP_ENA 决定，见表 31-2。

表 31-2. 唤醒源

WAKEUP_ENA	唤醒源[1]	Light-sleep	Deep-sleep	休眠
0x1	EXT0[2]	Y	Y	-
0x2	EXT1[3]	Y	Y	Y
0x4	GPIO[4]	Y	Y	-
0x8	RTC 计时器	Y	Y	Y
0x10	SDIO[5]	Y	-	-
0x20	Wi-Fi[6]	Y	-	-
0x40	UART0 [7]	Y	-	-
0x80	UART1 [7]	Y	-	-
0x100	TOUCH	Y	Y	-
0x200	ULP 协处理器	Y	Y	-
0x400	BT [6]	Y	-	-

[1] 表格中的所有唤醒源均可配置为拒绝睡眠原因，但 UART 除外。

[2] EXT0 仅可将芯片从 Light-sleep/Deep-sleep 模式中唤醒。当 RTC_CNTL_EXT_WAKEUP0_LV 为 1 时将触发管脚高电平，否则将触发管脚低电平。用户可通过设置 RTCIO_EXT_WAKEUP0_SEL[4:0]，选择将作为唤醒源的 RTC 管脚。

[3] EXT1 经过专门设计，可将芯片从任何睡眠模式中唤醒，而且还支持多个管脚的组合。首先，应按照选定作为唤醒源的管脚位图，配置 RTC_CNTL_EXT_WAKEUP1_SEL[17:0]。接着，当 RTC_CNTL_EXT_WAKEUP1_LV 为 1 时，只要有任何选择的管脚为高电压，则将触发信号唤醒芯片；当 RTC_CNTL_EXT_WAKEUP1_LV 为 0 时，则必须全部选择的管脚均为低电压才会触发信号。

[4] 在 Deep-sleep 模式下，仅有 RTC GPIO 可以作为唤醒源，而非数字 GPIO。

[5] 任何接收到的 SDIO 命令均将触发唤醒动作。

[6] 为了通过 Wi-Fi 或 BT 源唤醒芯片，芯片将在 Active、Modem-sleep 和 Light-sleep 之间进行切换，CPU、Wi-Fi、Bluetooth 和射频模块均将在预设间隔中唤醒，保证 Wi-Fi/蓝牙的正常连接。

[7] 当输入 RxD 沿变化的次数大于等于 (UART_ACTIVE_THRESHOLD+2) 时，即触发唤醒。**注意：**RxD 不能通过 GPIO 交换矩阵输入，只能通过 IO_MUX 输入。

31.3.11 拒绝睡眠

ESP32 提供了硬件拒绝睡眠的机制，防止系统在某些外设仍在工作但是未被 CPU 检测到时进入睡眠，最终导致该外设不能正常工作。

ESP32 的所有唤醒源也同时可以作为芯片拒绝睡眠的原因，因此具体的拒绝睡眠源请见表 31-2（UART 除

外)。

用户可以配置以下寄存器使能或禁用拒绝睡眠功能。

- 配置 RTC_CNTL_SLP_REJECT 整体使能或关闭拒绝休眠功能:
  - 进一步配置 RTC_CNTL_LIGHT_SLP_REJECT_EN，具体使能拒绝进入 Light_sleep;
  - 进一步配置 RTC_CNTL_DEEP_SLP_REJECT_EN，具体使能拒绝进入 Deep_sleep;
- 读取 RTC_CNTL_REJECT_CAUSE 了解拒绝睡眠的原因。

31.3.12 RTC 计时器

RTC 计时器为一个可读 48-bit 计数器，时钟为 RTC_SLOW_CLK。除上电复位外其余任何复位 / 睡眠均不会使 RTC 计时器停止或复位。

RTC 计时器可以在指定时间唤醒 CPU，并周期性唤醒触摸控制器和 ULP 协处理器。

31.3.13 RTC Boot

由于 CPU 的 ROM 和 RAM 内存均将在 Deep-sleep 和休眠模式下断电，而 ROM 解包与 SPI 启动（从 flash 中复制数据）均需要时间，因此这两种模式下的唤醒过程均比 Light-sleep 和 Modem-sleep 模式长的多。RTC 域中有 2 种 SRAM 内存，分别为 RTC 慢速内存和 RTC 快速内存。这 2 种 SRAM 内存均将在 Deep-sleep 模式下保持开启。一些代码规模不大的应用（小于 8 KB）可通过两种方法，避免 ROM 解包或 SPI 启动，从而加速芯片唤醒过程。

第一种方法：使用 RTC 慢速内存

1. 设置 PRO_CPU 寄存器 RTC_CNTL_PROCPU_STAT_VECTOR_SEL（或设置 APP_CPU 寄存器 RTC_CNTL_APPCPU_STAT_VECTOR_SEL）为 0。
2. 芯片进入睡眠模式。
3. 当 CPU 开启时，复位向量将从地址 0x50000000，而非 0x40000400 开始复位，整个过程并不需要进行 ROM 解包和 SPI boot。RTC 内存中的代码仅需在 C 语言环境中进行部分初始化操作即可。

第二种方法：使用 RTC 快速内存

1. 设置 PRO_CPU 寄存器 RTC_CNTL_PROCPU_STAT_VECTOR_SEL（或设置 APP_CPU 寄存器 RTC_CNTL_APPCPU_STAT_VECTOR_SEL）为 1。
2. 计算 RTC 快速内存的 CRC 码，并将结果保存在寄存器 RTC_CNTL_RTC_STORE6_REG[31:0] 中。
3. 将 RTC 快速内存的入口地址输入寄存器 RTC_CNTL_RTC_STORE7_REG[31:0]。
4. 芯片进入睡眠模式。
5. CPU 开启，在完成 ROM 解包及部分初始化工作后，再次计算 CRC 码。如果计算结果与寄存器 RTC_CNTL_RTC_STORE6_REG[31:0] 一致，则 CPU 将跳转至入口地址。

ESP32 的启动流程图为图 31-11。

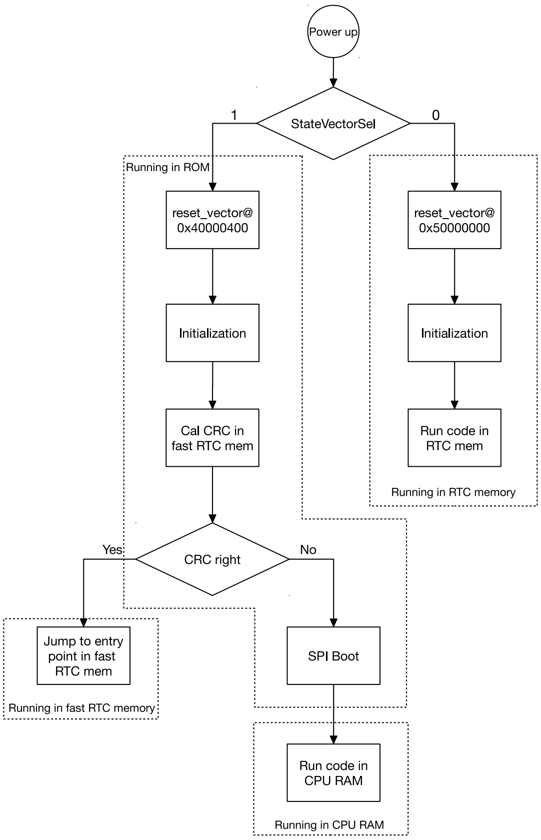


图 31-11. ESP32 启动流程图

31.4 寄存器列表

说明：

- 以下寄存器已按照具体功能进行分组，但列表中的顺序并不反映寄存器在内存中的真实顺序。
- 如果从 AHB 总线进行访问，则寄存器的基址为 0x60008000；如果从 DPORT 总线进行访问，则寄存器的基址为 0x3FF48000。

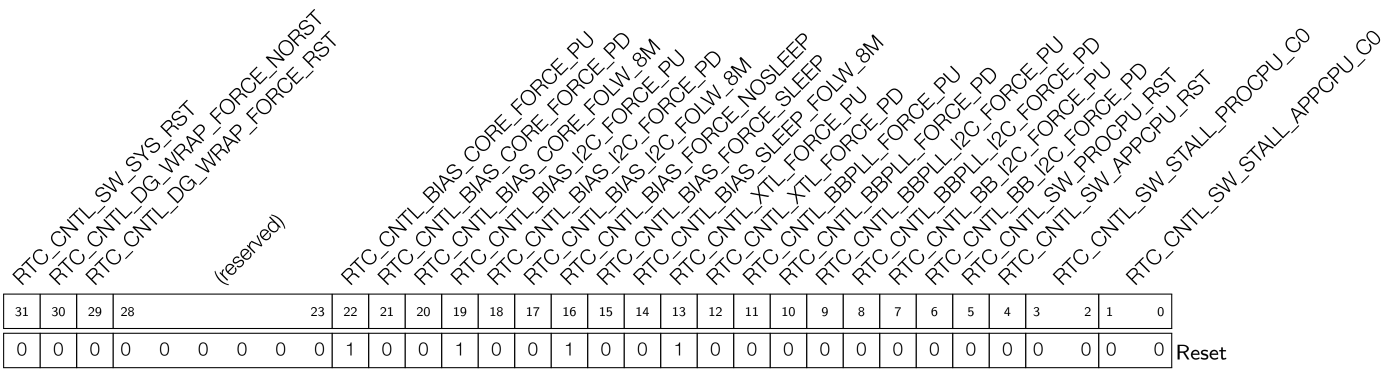
名称	描述	地址	访问方式
RTC 选项寄存器			
RTC_CNTL_OPTIONS0_REG	配置 RTC 选项	0x3FF48000	读 / 写
RTC 计时器寄存器的控制与配置			
RTC_CNTL_SLP_TIMER0_REG	RTC 睡眠计时器	0x3FF48004	读 / 写
RTC_CNTL_SLP_TIMER1_REG	RTC 睡眠计时器、警报和控制	0x3FF48008	读 / 写
RTC_CNTL_TIME_UPDATE_REG	RTC 计时器的更新控制	0x3FF4800C	只读
RTC_CNTL_TIME0_REG	RTC 计时器低 32 位	0x3FF48010	只读
RTC_CNTL_TIME1_REG	RTC 计时器高 16 位	0x3FF48014	只读
RTC_CNTL_STATE0_REG	RTC 睡眠、SDIO 和 ULP 控制	0x3FF48018	读 / 写
RTC_CNTL_TIMER1_REG	CPU 等待使能	0x3FF4801C	读 / 写
RTC_CNTL_TIMER2_REG	慢速时钟和触摸控制器配置	0x3FF48020	读 / 写
RTC_CNTL_TIMER5_REG	慢速时钟下的最小睡眠周期	0x3FF4802C	读 / 写
复位状态与唤醒控制寄存器			
RTC_CNTL_RESET_STATE_REG	CPU 复位状态控制与原因	0x3FF48034	只读
RTC_CNTL_WAKEUP_STATE_REG	唤醒过滤器、使能与原因	0x3FF48038	只读
RTC_CNTL_EXT_WAKEUP_CONF_REG	低/高电平唤醒配置	0x3FF48060	读 / 写
RTC_CNTL_EXT_WAKEUP1_REG	外部唤醒位与唤醒清除位的管脚选择	0x3FF480CC	读/写
RTC_CNTL_EXT_WAKEUP1_STATUS_REG	外部唤醒状态	0x3FF480D0	只读
RTC 中断控制与状态寄存器			
RTC_CNTL_INT_ENA_REG	中断使能位	0x3FF4803C	读 / 写
RTC_CNTL_INT_RAW_REG	原始中断状态	0x3FF48040	只读
RTC_CNTL_INT_ST_REG	屏蔽中断状态	0x3FF48044	只读
RTC_CNTL_INT_CLR_REG	中断清除位	0x3FF48048	只写
RTC 通用保留寄存器			
RTC_CNTL_STORE0_REG	通用保留寄存器 0	0x3FF4804C	读 / 写
RTC_CNTL_STORE1_REG	通用保留寄存器 1	0x3FF48050	读 / 写
RTC_CNTL_STORE2_REG	通用保留寄存器 2	0x3FF48054	读 / 写
RTC_CNTL_STORE3_REG	通用保留寄存器 3	0x3FF48058	读 / 写
RTC_CNTL_STORE4_REG	通用保留寄存器 4	0x3FF480B0	读 / 写
RTC_CNTL_STORE5_REG	通用保留寄存器 5	0x3FF480B4	读 / 写
RTC_CNTL_STORE6_REG	通用保留寄存器 6	0x3FF480B8	读 / 写
RTC_CNTL_STORE7_REG	通用保留寄存器 7	0x3FF480BC	读 / 写
内部功耗管理寄存器			
RTC_CNTL_ANA_CONF_REG	上电 / 断电配置	0x3FF48030	读 / 写
RTC_CNTL_VREG_REG	内部功耗分配与控制	0x3FF4807C	读 / 写
RTC_CNTL_PWC_REG	RTC 域功耗管理	0x3FF48080	读 / 写

名称	描述	地址	访问方式
RTC_CNTL_DIG_PWC_REG	数字域功耗管理	0x3FF48084	读 / 写
RTC_CNTL_DIG_ISO_REG	数字域隔离控制	0x3FF48088	只读
RTC 看门狗配置域控制寄存器			
RTC_CNTL_WDTCONFIG0_REG	WDT 配置寄存器 0	0x3FF4808C	读 / 写
RTC_CNTL_WDTCONFIG1_REG	WDT 配置寄存器 1	0x3FF48090	读 / 写
RTC_CNTL_WDTCONFIG2_REG	WDT 配置寄存器 2	0x3FF48094	读 / 写
RTC_CNTL_WDTCONFIG3_REG	WDT 配置寄存器 3	0x3FF48098	读 / 写
RTC_CNTL_WDTCONFIG4_REG	WDT 配置寄存器 4	0x3FF4809C	读 / 写
RTC_CNTL_WDTFEED_REG	喂狗寄存器	0x3FF480A0	只写
RTC_CNTL_WDTWPROTECT_REG	看门狗写保护寄存器	0x3FF480A4	读 / 写
其他 RTC 配置寄存器			
RTC_CNTL_EXT_XTL_CONF_REG	外部管脚 XTAL 控制	0x3FF4805C	读 / 写
RTC_CNTL_SLP_REJECT_CONF_REG	拒绝原因和使能控制	0x3FF48064	读 / 写
RTC_CNTL_CPU_PERIOD_CONF_REG	CPU 周期选择	0x3FF48068	读 / 写
RTC_CNTL_CLK_CONF_REG	RTC 时钟配置	0x3FF48070	读 / 写
RTC_CNTL_SDIO_CONF_REG	SDIO 配置	0x3FF48074	读 / 写
RTC_CNTL_SW_CPU_STALL_REG	CPU 停机	0x3FF480AC	读 / 写
RTC_CNTL_HOLD_FORCE_REG	RTC 管脚保留寄存器	0x3FF480C8	读 / 写
RTC_CNTL_BROWN_OUT_REG	欠压管理	0x3FF480D4	读 / 写

31.5 寄存器

本小节括号中的地址均为相对于低功耗管理 (RTC) 基地址的地址偏移量（相对地址），具体基地址见章节 1 系统和存储器 中的表 1-6 外设地址映射。寄存器绝对地址见章节 31.4 寄存器列表。

Register 31.1. RTC_CNTL_OPTIONS0_REG (0x0000)



RTC_CNTL_SW_SYS_RST 软件系统复位。(只写)

RTC_CNTL_DG_WRAP_FORCE_NORST Deep-sleep 模式下，数字内核强制不复位。(读 / 写)

RTC_CNTL_DG_WRAP_FORCE_RST Deep-sleep 模式下，数字内核强制复位。(读 / 写)

RTC_CNTL_BIAS_CORE_FORCE_PU BIAS_CORE 强制打开。(读 / 写)

RTC_CNTL_BIAS_CORE_FORCE_PD BIAS_CORE 强制关闭。(读 / 写)

RTC_CNTL_BIAS_CORE_FOLW_8M BIAS_CORE 随 CK8M 变化。(读 / 写)

RTC_CNTL_BIAS_I2C_FORCE_PU BIAS_I2C 强制打开。(读 / 写)

RTC_CNTL_BIAS_I2C_FORCE_PD BIAS_I2C 强制关闭。(读 / 写)

RTC_CNTL_BIAS_I2C_FOLW_8M BIAS_I2C 随 CK8M 变化。(读 / 写)

RTC_CNTL_BIAS_FORCE_NOSLEEP BIAS_SLEEP 强制不睡眠。(读 / 写)

RTC_CNTL_BIAS_FORCE_SLEEP BIAS_SLEEP 强制睡眠。(读 / 写)

RTC_CNTL_BIAS_SLEEP_FOLW_8M BIAS_SLEEP 随 CK8M 变化。(读 / 写)

RTC_CNTL_XTL_FORCE_PU Crystal 强制打开。(读 / 写)

RTC_CNTL_XTL_FORCE_PD Crystal 强制关闭。(读 / 写)

RTC_CNTL_BBPLL_FORCE_PU BB_PLL 强制打开。(读 / 写)

RTC_CNTL_BBPLL_FORCE_PD BB_PLL 强制关闭。(读 / 写)

RTC_CNTL_BBPLL_I2C_FORCE_PU BB_PLL_I2C 强制打开。(读 / 写)

RTC_CNTL_BBPLL_I2C_FORCE_PD BB_PLL _I2C 强制关闭。(读 / 写)

RTC_CNTL_BB_I2C_FORCE_PU BB_I2C 强制打开。(读 / 写)

RTC_CNTL_BB_I2C_FORCE_PD BB_I2C 强制关闭。(读 / 写)

RTC_CNTL_SW_PROCPU_RST PRO_CPU 软件复位。(只写)

寄存器描述下一页继续。

Register 31.1. RTC_CNTL_OPTIONS0_REG (0x0000)

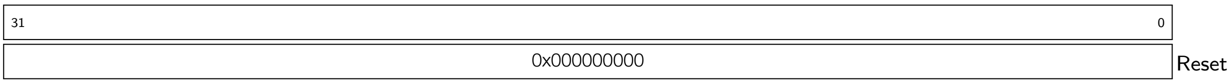
继上一页寄存器描述。

RTC_CNTL_SW_APPCPU_RST APP_CPU 软件复位。(只写)

RTC_CNTL_SW_STALL_PROCPU_C0 参见 RTC_CNTL_SW_CPU_STALL_REG。(读 / 写)

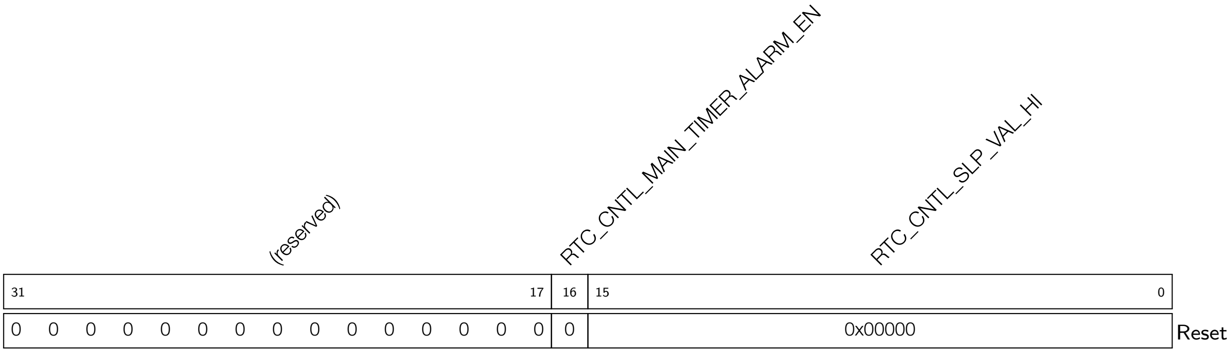
RTC_CNTL_SW_STALL_APPCPU_C0 参见 RTC_CNTL_SW_CPU_STALL_REG。(读 / 写)

Register 31.2. RTC_CNTL_SLP_TIMER0_REG (0x0004)



RTC_CNTL_SLP_TIMER0_REG RTC 睡眠计时器低 32 位。(读 / 写)

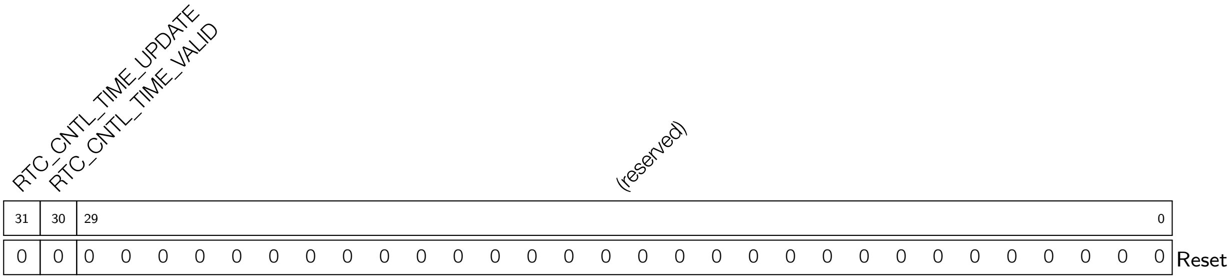
Register 31.3. RTC_CNTL_SLP_TIMER1_REG (0x0008)



RTC_CNTL_MAIN_TIMER_ALARM_EN 计时器警报使能位。(读 / 写)

RTC_CNTL_SLP_VAL_HI RTC 睡眠计时器高 16 位。(读 / 写)

Register 31.4. RTC_CNTL_TIME_UPDATE_REG (0x000C)



RTC_CNTL_TIME_UPDATE 置 1：使用 RTC 计时器更新寄存器。(只写)

RTC_CNTL_TIME_VALID 表示寄存器已更新。(只读)

Register 31.5. RTC_CNTL_TIME0_REG (0x0010)

31 … 0	
0x000000000	Reset

RTC_CNTL_TIME0_REG RTC 计时器低 32 位。(只读)

Register 31.6. RTC_CNTL_TIME1_REG (0x0014)

(reserved)	RTC_CNTL_TIME_HI	
31 … 16	15 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x00000	Reset

RTC_CNTL_TIME_HI RTC 计时器高 16 位。(只读)

Register 31.7. RTC_CNTL_STATE0_REG (0x0018)

RTC_CNTL_SLEEP_EN	RTC_CNTL_SLP_REJECT	RTC_CNTL_SLP_WAKEUP	RTC_CNTL_SDIO_ACTIVE_IND	(reserved)	RTC_CNTL_ULP_CP_SLP_TIMER_EN	RTC_CNTL_TOUCH_SLP_TIMER_EN	(reserved)	
31	30	29	28	27 … 25	24	23	22 … 0	
0	0	0	0	0 0 0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_SLEEP_EN 睡眠使能位。(读 / 写)

RTC_CNTL_SLP_REJECT 睡眠拒绝位。(读 / 写)

RTC_CNTL_SLP_WAKEUP 睡眠唤醒位。(读 / 写)

RTC_CNTL_SDIO_ACTIVE_IND 表示 SDIO 激活。(只读)

RTC_CNTL_ULP_CP_SLP_TIMER_EN ULP 协处理器计时器使能位。(读 / 写)

RTC_CNTL_TOUCH_SLP_TIMER_EN 触摸传感器使能位。(读 / 写)

Register 31.8. RTC_CNTL_TIMER1_REG (0x001C)

(reserved)	RTC_CNTL_CPU_STALL_EN	
31 1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	Reset

RTC_CNTL_CPU_STALL_EN CPU 暂停使能位。（读 / 写）

Register 31.9. RTC_CNTL_TIMER2_REG (0x0020)

RTC_CNTL_MIN_TIME_CK8M_OFF	RTC_CNTL_ULPCP_TOUCH_START_WAIT	(reserved)	
31 24	23 15	14 0	
0x001	0x010	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_MIN_TIME_CK8M_OFF 关闭条件下，CK8M RTC_SLOW_CLK 的最小周期数。（读 / 写）

RTC_CNTL_ULPCP_TOUCH_START_WAIT ULP 协处理器 / 触摸传感器开始工作前，RTC_SLOW_CLK 的等待周期数。（读 / 写）

Register 31.10. RTC_CNTL_TIMER5_REG (0x002C)

(reserved)	RTC_CNTL_MIN_SLP_VAL	(reserved)	
31 16	15 8	7 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x080	0 0 0 0 0 0 0 0	Reset

RTC_CNTL_MIN_SLP_VAL RTC_SLOW_CLK 最小睡眠周期数。（读 / 写）

Register 31.11. RTC_CNTL_ANA_CONF_REG (0x0030)

RTC_CNTL_PLL_I2C_PU	RTC_CNTL_CKGEN_I2C_PU	(reserved)	RTC_CNTL_RFRX_PBUS_PU	RTC_CNTL_TXRF_I2C_PU	RTC_CNTL_PVTMON_PU	(reserved)	RTC_CNTL_PLLA_FORCE_PU	RTC_CNTL_PLLA_FORCE_PD	(reserved)	
31	30	29	28	27	26	25	24	23	22 0	
0	0	0	0	0	0	0	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_PLL_I2C_PU 1：PLL_I2C 打开；0：关闭。（读 / 写）

RTC_CNTL_CKGEN_I2C_PU 1：CKGEN_I2C 打开；0：关闭。（读 / 写）

RTC_CNTL_RFRX_PBUS_PU 1：RFRX_PBUS 打开；0：关闭。（读 / 写）

RTC_CNTL_TXRF_I2C_PU 1：TXRF_I2C 打开；0：关闭。（读 / 写）

RTC_CNTL_PVTMON_PU 1：PVTMON 打开；0：关闭。（读 / 写）

RTC_CNTL_PLLA_FORCE_PU PLLA 强制打开。（读 / 写）

RTC_CNTL_PLLA_FORCE_PD PLLA 强制关闭。（读 / 写）

Register 31.12. RTC_CNTL_RESET_STATE_REG (0x0034)

(reserved)	RTC_CNTL_PROCPU_STAT_VECTOR_SEL	RTC_CNTL_APPCPU_STAT_VECTOR_SEL	RTC_CNTL_RESET_CAUSE_APPCPU	RTC_CNTL_RESET_CAUSE_PROCPU	
31 14	13	12	11 6	5 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1	1	x x x x x x	x x x x x x	Reset

RTC_CNTL_PROCPU_STAT_VECTOR_SEL PRO_CPU 状态向量选择。（读 / 写）

RTC_CNTL_APPCPU_STAT_VECTOR_SEL APP_CPU 状态向量选择。（读 / 写）

RTC_CNTL_RESET_CAUSE_APPCPU APP_CPU 复位原因。（只读）

RTC_CNTL_RESET_CAUSE_PROCPU PRO_CPU 复位原因。（只读）

Register 31.13. RTC_CNTL_WAKEUP_STATE_REG (0x0038)

(reserved)	RTC_CNTL_GPIO_WAKEUP_FILTER	RTC_CNTL_WAKEUP_ENA	RTC_CNTL_WAKEUP_CAUSE	
31 23	22	21 11	10 0	
0 0 0 0 0 0 0 0 0	0	0 0 0 0 0 0 0 1 1 0 0	0x000	Reset

RTC_CNTL_GPIO_WAKEUP_FILTER GPIO 唤醒事件使能过滤器。（读 / 写）

RTC_CNTL_WAKEUP_ENA 唤醒使能位图。（读 / 写）

RTC_CNTL_WAKEUP_CAUSE 唤醒原因。（只读）

Register 31.14. RTC_CNTL_INT_ENA_REG (0x003C)

(reserved)	RTC_CNTL_MAIN_TIMER_INT_ENA	RTC_CNTL_BROWN_OUT_INT_ENA	RTC_CNTL_TOUCH_INT_ENA	RTC_CNTL_ULP_CP_INT_ENA	RTC_CNTL_TIME_VALID_INT_ENA	RTC_CNTL_WDT_INT_ENA	RTC_CNTL_SDIO_IDLE_INT_ENA	RTC_CNTL_SLP_REJECT_INT_ENA	RTC_CNTL_SLP_WAKEUP_INT_ENA	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_MAIN_TIMER_INT_ENA RTC_CNTL_MAIN_TIMER_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_BROWN_OUT_INT_ENA RTC_CNTL_BROWN_OUT_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_TOUCH_INT_ENA RTC_CNTL_TOUCH_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_ULP_CP_INT_ENA RTC_CNTL_ULP_CP_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_TIME_VALID_INT_ENA RTC_CNTL_TIME_VALID_INT 中断的中断使能位。（读 / 写）

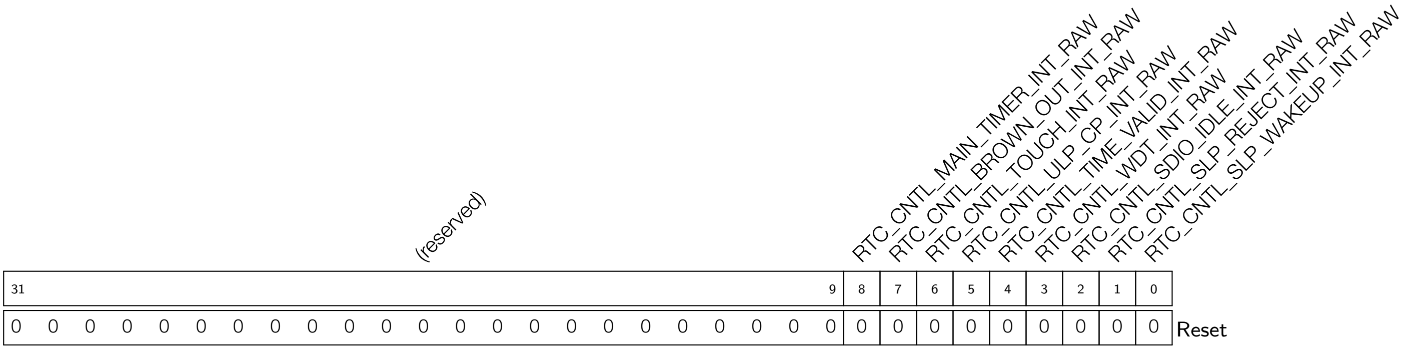
RTC_CNTL_WDT_INT_ENA RTC_CNTL_WDT_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_SDIO_IDLE_INT_ENA RTC_CNTL_SDIO_IDLE_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_SLP_REJECT_INT_ENA RTC_CNTL_SLP_REJECT_INT 中断的中断使能位。（读 / 写）

RTC_CNTL_SLP_WAKEUP_INT_ENA RTC_CNTL_SLP_WAKEUP_INT 中断的中断使能位。（读 / 写）

Register 31.15. RTC_CNTL_INT_RAW_REG (0x0040)



RTC_CNTL_MAIN_TIMER_INT_RAW RTC_CNTL_MAIN_TIMER_INT 中断的原始中断状态位。(只读)

RTC_CNTL_BROWN_OUT_INT_RAW RTC_CNTL_BROWN_OUT_INT 中断的原始中断状态位。(只读)

RTC_CNTL_TOUCH_INT_RAW RTC_CNTL_TOUCH_INT 中断的原始中断状态位。(只读)

RTC_CNTL_ULP_CP_INT_RAW RTC_CNTL_ULP_CP_INT 中断的原始中断状态位。(只读)

RTC_CNTL_TIME_VALID_INT_RAW RTC_CNTL_TIME_VALID_INT 中断的原始中断状态位。(只读)

RTC_CNTL_WDT_INT_RAW RTC_CNTL_WDT_INT 中断的原始中断状态位。(只读)

RTC_CNTL_SDIO_IDLE_INT_RAW RTC_CNTL_SDIO_IDLE_INT 中断的原始中断状态位。(只读)

RTC_CNTL_SLP_REJECT_INT_RAW RTC_CNTL_SLP_REJECT_INT 中断的原始中断状态位。(只读)

RTC_CNTL_SLP_WAKEUP_INT_RAW RTC_CNTL_SLP_WAKEUP_INT 中断的原始中断状态位。(只读)

Register 31.16. RTC_CNTL_INT_ST_REG (0x0044)

(reserved)	RTC_CNTL_MAIN_TIMER_INT_ST	RTC_CNTL_BROWN_OUT_INT_ST	RTC_CNTL_TOUCH_INT_ST	RTC_CNTL_SAR_INT_ST	RTC_CNTL_TIME_VALID_INT_ST	RTC_CNTL_WDT_INT_ST	RTC_CNTL_SDIO_IDLE_INT_ST	RTC_CNTL_SLP_REJECT_INT_ST	RTC_CNTL_SLP_WAKEUP_INT_ST	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_MAIN_TIMER_INT_ST RTC_CNTL_MAIN_TIMER_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_BROWN_OUT_INT_ST RTC_CNTL_BROWN_OUT_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_TOUCH_INT_ST RTC_CNTL_TOUCH_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_SAR_INT_ST RTC_CNTL_SAR_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_TIME_VALID_INT_ST RTC_CNTL_TIME_VALID_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_WDT_INT_ST RTC_CNTL_WDT_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_SDIO_IDLE_INT_ST RTC_CNTL_SDIO_IDLE_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_SLP_REJECT_INT_ST RTC_CNTL_SLP_REJECT_INT 中断的屏蔽中断状态位。(只读)

RTC_CNTL_SLP_WAKEUP_INT_ST RTC_CNTL_SLP_WAKEUP_INT 中断的屏蔽中断状态位。(只读)

Register 31.17. RTC_CNTL_INT_CLR_REG (0x0048)

(reserved)	RTC_CNTL_MAIN_TIMER_INT_CLR	RTC_CNTL_BROWN_OUT_INT_CLR	RTC_CNTL_TOUCH_INT_CLR	RTC_CNTL_SAR_INT_CLR	RTC_CNTL_TIME_VALID_INT_CLR	RTC_CNTL_WDT_INT_CLR	RTC_CNTL_SDIO_IDLE_INT_CLR	RTC_CNTL_SLP_REJECT_INT_CLR	RTC_CNTL_SLP_WAKEUP_INT_CLR	
31 9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_MAIN_TIMER_INT_CLR 此位置 1 清除 RTC_CNTL_MAIN_TIMER_INT 中断。(只写)

RTC_CNTL_BROWN_OUT_INT_CLR 此位置 1 清除 RTC_CNTL_BROWN_OUT_INT 中断。(只写)

RTC_CNTL_TOUCH_INT_CLR 此位置 1 清除 RTC_CNTL_TOUCH_INT 中断。(只写)

RTC_CNTL_SAR_INT_CLR 此位置 1 清除 RTC_CNTL_SAR_INT 中断。(只写)

RTC_CNTL_TIME_VALID_INT_CLR 此位置 1 清除 RTC_CNTL_TIME_VALID_INT 中断。(只写)

RTC_CNTL_WDT_INT_CLR 此位置 1 清除 RTC_CNTL_WDT_INT 中断。(只写)

RTC_CNTL_SDIO_IDLE_INT_CLR 此位置 1 清除 RTC_CNTL_SDIO_IDLE_INT 中断。(只写)

RTC_CNTL_SLP_REJECT_INT_CLR 此位置 1 清除 RTC_CNTL_SLP_REJECT_INT 中断。(只写)

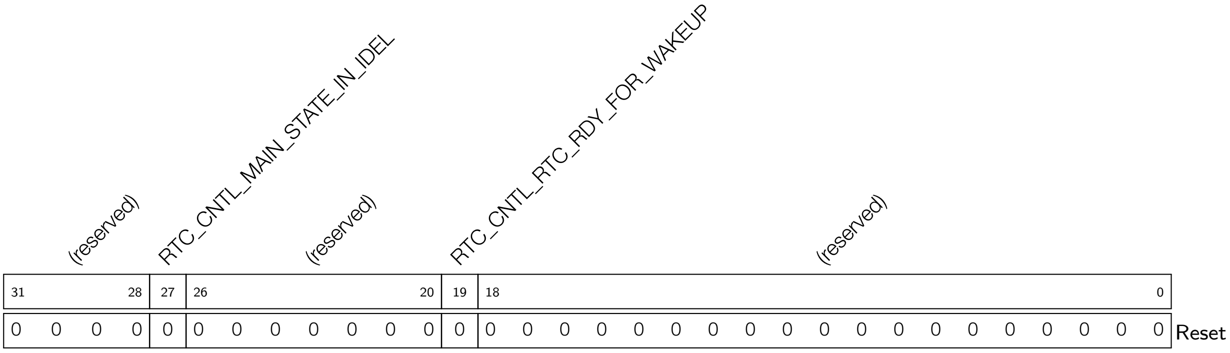
RTC_CNTL_SLP_WAKEUP_INT_CLR 此位置 1 清除 RTC_CNTL_SLP_WAKEUP_INT 中断。(只写)

Register 31.18. RTC_CNTL_STORE*n*_REG (*n*: 0-3) (0x004C+4**n*)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

RTC_CNTL_STORE*n*_REG 32-bit 通用保留寄存器。(读 / 写)

Register 31.19. RTC_CNTL_LOW_POWER_ST_REG (0x00C0)

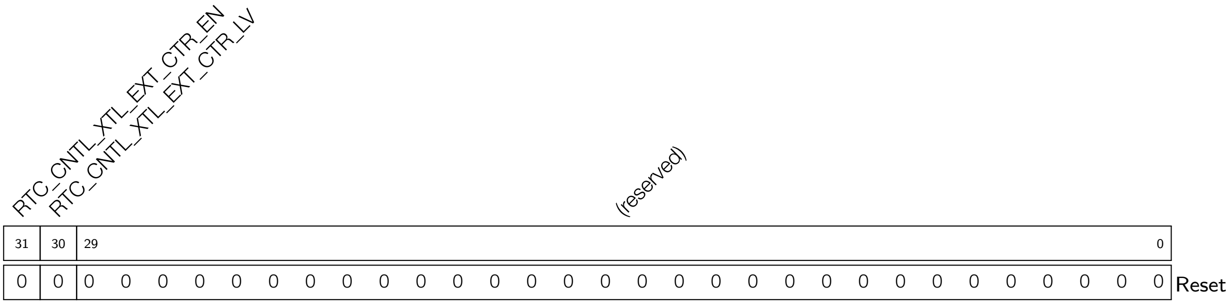


RTC_CNTL_RTC_RDY_FOR_WAKEUP 代表 RTC 已经准备好被任何唤醒源唤醒。(RO)

RTC_CNTL_MAIN_STATE_IN_IDLE 代表 RTC 状态。

- 0：芯片可能处于以下任一情况
  - 已经进入睡眠模式
  - 正在进入睡眠模式。此时，需等待 RTC_CNTL_RTC_RDY_FOR_WAKEUP 变为 1，然后可以唤醒芯片。
  - 正在退出睡眠模式。此时，RTC_CNTL_MAIN_STATE_IN_IDLE 终将变为 1。
- 1：芯片不处于睡眠模式（比如正常运行中）。

Register 31.20. RTC_CNTL_EXT_XTL_CONF_REG (0x005C)



RTC_CNTL_XTL_EXT_CTR_EN 外部管脚使能 XTAL 控制。（读 / 写）

RTC_CNTL_XTL_EXT_CTR_LV 0：高电平时，XTAL 关闭；1：低电平时，XTAL 关闭。（读 / 写）

Register 31.21. RTC_CNTL_EXT_WAKEUP_CONF_REG (0x0060)

RTC_CNTL_EXT_WAKEUP1_LV	RTC_CNTL_EXT_WAKEUP0_LV	(reserved)	
31	30	29 0	
0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_EXT_WAKEUP1_LV 0：低电平时，外部唤醒；1：高电平时，外部唤醒。（读 / 写）

RTC_CNTL_EXT_WAKEUP0_LV 0：低电平时，外部唤醒；1：高电平时，外部唤醒。（读 / 写）

Register 31.22. RTC_CNTL_SLP_REJECT_CONF_REG (0x0064)

RTC_CNTL_REJECT_CAUSE	RTC_CNTL_DEEP_SLP_REJECT_EN	RTC_CNTL_LIGHT_SLP_REJECT_EN	RTC_CNTL_SDIO_REJECT_EN	RTC_CNTL_GPIO_REJECT_EN	(reserved)	
31 28	27	26	25	24	23 0	
0 0 0 0	0	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_REJECT_CAUSE 睡眠拒绝原因。（只读）

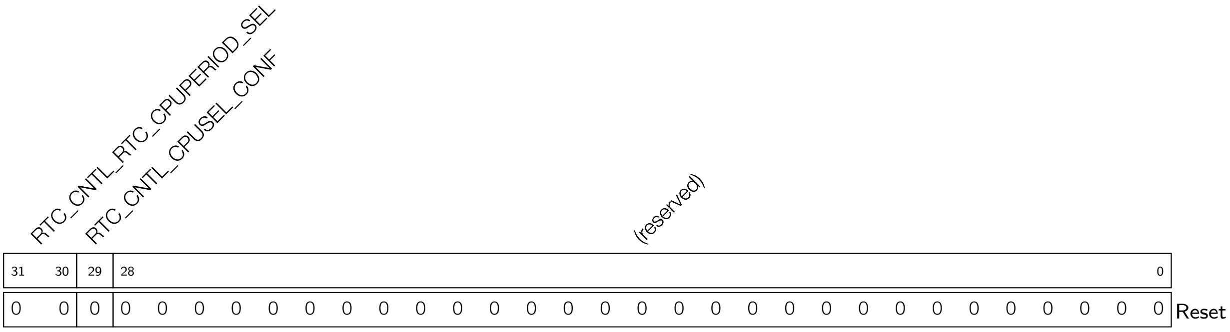
RTC_CNTL_DEEP_SLP_REJECT_EN 使能拒绝 Deep-sleep。（读 / 写）

RTC_CNTL_LIGHT_SLP_REJECT_EN 使能拒绝 Light-sleep。（读 / 写）

RTC_CNTL_SDIO_REJECT_EN 使能 SDIO 拒绝。（读 / 写）

RTC_CNTL_GPIO_REJECT_EN 使能 GPIO 拒绝。（读 / 写）

Register 31.23. RTC_CNTL_CPU_PERIOD_CONF_REG (0x0068)



RTC_CNTL_RTC_CPUPERIOD_SEL CPU 周期选择。（读 / 写）

RTC_CNTL_CPUSEL_CONF CPU 选择选项。（读 / 写）

Register 31.24. RTC_CNTL_CLK_CONF_REG (0x0070)

RTC_CNTL_ANA_CLK_RTC_SEL	RTC_CNTL_RTC_FAST_CLK_SEL	RTC_CNTL_SOC_CLK_SEL	RTC_CNTL_CK8M_FORCE_PU	RTC_CNTL_CK8M_FORCE_PD	RTC_CNTL_CK8M_DFREQ	(reserved)	RTC_CNTL_CK8M_DIV_SEL	(reserved)	RTC_CNTL_DIG_CLK8M_EN	RTC_CNTL_DIG_CLK8M_D256_EN	RTC_CNTL_DIG_XTAL32K_EN	RTC_CNTL_ENB_CK8M_DIV	RTC_CNTL_ENB_CK8M	RTC_CNTL_CK8M_DIV	(reserved)	
31 30	29	28 27	26	25	24 17	16 15	14 12	11	10	9	8	7	6	5 4	3 0	
0	0	0	0	0	0	0 0	2	0	0	1	0	0	0	0 1	0 0 0 0	Reset

RTC_CNTL_ANA_CLK_RTC_SEL 选择 RTC_SLOW_CLK。0：RC_SLOW_CLK；1：XTL32K_CLK；2：RC_FAST_DIV_CLK。(读 / 写)

RTC_CNTL_RTC_FAST_CLK_SEL 选择 RTC_FAST_CLK。0：XTAL_DIV_CLK；1：RC_FAST_CLK。(读 / 写)

RTC_CNTL_SOC_CLK_SEL 选择 SOC 时钟。0：XTAL；1：PLL；2：CK8M；3: APLL。(读 / 写)

RTC_CNTL_CK8M_FORCE_PU CK8M 强制打开。(读 / 写)

RTC_CNTL_CK8M_FORCE_PD CK8M 强制关闭。(读 / 写)

RTC_CNTL_CK8M_DFREQ CK8M_DFREQ。(读 / 写)

RTC_CNTL_CK8M_DIV_SEL Divider = reg_rtc_cntl_ck8m_div_sel + 1。(读 / 写)

RTC_CNTL_DIG_CLK8M_EN 数字内核使能 CK8M（注意：与 RTC 内核无关）。(读 / 写)

RTC_CNTL_DIG_CLK8M_D256_EN 数字内核使能 RC_FAST_DIV_CLK（注意：与 RTC 内核无关）。(读 / 写)

RTC_CNTL_DIG_XTAL32K_EN 数字内核使能 XTL32K_CLK（注意：与 RTC 内核无关）。(读 / 写)

RTC_CNTL_ENB_CK8M_DIV 1：RC_FAST_DIV_CLK 实际为 CK8M；0：RC_FAST_DIV_CLK 为 CK8M 的 256 分频。(读 / 写)

RTC_CNTL_ENB_CK8M 禁用 CK8M 和 RC_FAST_DIV_CLK。(读 / 写)

RTC_CNTL_CK8M_DIV RC_FAST_DIV_CLK 分频器。00：128 分频；01：256 分频；10：512 分频；11：1024 分频。(读 / 写)

Register 31.25. RTC_CNTL_SDIO_CONF_REG (0x0074)

RTC_CNTL_XPD_SDIO_VREG	RTC_CNTL_DREFH_SDIO	RTC_CNTL_DREFM_SDIO	RTC_CNTL_DREFL_SDIO	RTC_CNTL_REG1P8_READY	RTC_CNTL_SDIO_TIEH	RTC_CNTL_SDIO_FORCE	RTC_CNTL_SDIO_VREG_PD_EN	(reserved)	
31	30 29	28 27	26 25	24	23	22	21	20 0	
0	0 0	0 0	0 1	0	1	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_XPD_SDIO_VREG XPD_SDIO_VREG 软件选项，仅在 reg_rtc_cntl_sdio_force == 1 时激活。（读 / 写）

RTC_CNTL_DREFH_SDIO DREFH_SDIO 软件选项，仅在 reg_rtc_cntl_sdio_force == 1 时激活。（读 / 写）

RTC_CNTL_DREFM_SDIO DREFM_SDIO 软件选项，仅在 reg_rtc_cntl_sdio_force == 1 时激活。（读 / 写）

RTC_CNTL_DREFL_SDIO DREFL_SDIO 软件选项，仅在 reg_rtc_cntl_sdio_force == 1 时激活。（读 / 写）

RTC_CNTL_REG1P8_READY REG1P8_READY 只读寄存器。（只读）

RTC_CNTL_SDIO_TIEH SDIO_TIEH 软件选项，仅在 reg_rtc_cntl_sdio_force == 1 时激活。（读 / 写）

RTC_CNTL_SDIO_FORCE 1：使用软件选项控制 SDIO_VREG；0：使用状态机控制 SDIO_VREG。（读 / 写）

RTC_CNTL_SDIO_VREG_PD_EN 睡眠状态下，SDIO_VREG 关闭，仅在 reg_rtc_cntl_sdio_force == 0 时激活。（读 / 写）

Register 31.26. RTC_CNTL_VREG_REG (0x007C)

RTC_CNTL_PREG_FORCE_PU	RTC_CNTL_PREG_FORCE_PD	RTC_CNTL_DBOOST_FORCE_PU	RTC_CNTL_DBOOST_FORCE_PD	RTC_CNTL_DBIAS_WAK	RTC_CNTL_DBIAS_SLP	RTC_CNTL_SCK_DCAP	RTC_CNTL_DIG_VREG_DBIAS_WAK	RTC_CNTL_DIG_VREG_DBIAS_SLP	(reserved)	
31	30	29	28	27 25	24 22	21 14	13 11	10 8	7 0	
1	0	1	0	4	4	0	4	4	0 0 0 0 0 0 0 0	Reset

RTC_CNTL_VREG_FORCE_PU RTC 调压器 - 强制打开。(读 / 写)

RTC_CNTL_VREG_FORCE_PD RTC 调压器 - 强制关闭 (在这种情况下,“关闭” 指电压下降至 0.8V 或更低)。(读 / 写)

RTC_CNTL_DBOOST_FORCE_PU RTC_DBOOST 强制打开。(读 / 写)

RTC_CNTL_DBOOST_FORCE_PD RTC_DBOOST 强制关闭。(读 / 写)

RTC_CNTL_DBIAS_WAK 唤醒阶段的 RTC_DBIAS。(读 / 写)

RTC_CNTL_DBIAS_SLP 睡眠阶段的 RTC_DBIAS。(读 / 写)

RTC_CNTL_SCK_DCAP 调节 RTC 慢速时钟频率。(读 / 写)

RTC_CNTL_DIG_VREG_DBIAS_WAK 唤醒阶段的数字调压器 DBIAS。(读 / 写)

RTC_CNTL_DIG_VREG_DBIAS_SLP 睡眠阶段的数字调压器 DBIAS。(读 / 写)

Register 31.27. RTC_CNTL_PWC_REG (0x0080)

(reserved)	RTC_CNTL_PD_EN	RTC_CNTL_FORCE_PU	RTC_CNTL_FORCE_PD	RTC_CNTL_SLOWMEM_PD_EN	RTC_CNTL_SLOWMEM_FORCE_PU	RTC_CNTL_SLOWMEM_FORCE_PD	RTC_CNTL_FASTMEM_PD_EN	RTC_CNTL_FASTMEM_FORCE_PU	RTC_CNTL_FASTMEM_FORCE_PD	RTC_CNTL_SLOWMEM_FORCE_LPU	RTC_CNTL_SLOWMEM_FORCE_LPD	RTC_CNTL_SLOWMEM_FOLW_CPU	RTC_CNTL_FASTMEM_FORCE_LPU	RTC_CNTL_FASTMEM_FORCE_LPD	RTC_CNTL_FASTMEM_FOLW_CPU	RTC_CNTL_FORCE_NOISO	RTC_CNTL_FORCE_ISO	RTC_CNTL_SLOWMEM_FORCE_ISO	RTC_CNTL_SLOWMEM_FORCE_NOISO	RTC_CNTL_FASTMEM_FORCE_ISO	RTC_CNTL_FASTMEM_FORCE_NOISO	
31 21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	1	Reset

RTC_CNTL_PD_EN 睡眠状态下，使能关闭 rtc_peri。(读 / 写)

RTC_CNTL_FORCE_PU rtc_peri 强制打开。(读 / 写)

RTC_CNTL_FORCE_PD rtc_peri 强制关闭。(读 / 写)

RTC_CNTL_SLOWMEM_PD_EN 睡眠状态下，使能关闭 RTC 内存。(读 / 写)

RTC_CNTL_SLOWMEM_FORCE_PU RTC 内存强制打开。(读 / 写)

RTC_CNTL_SLOWMEM_FORCE_PD RTC 内存强制关闭。(读 / 写)

RTC_CNTL_FASTMEM_PD_EN 睡眠状态下，使能关闭快速 RTC 内存。(读 / 写)

RTC_CNTL_FASTMEM_FORCE_PU 快速 RTC 内存强制打开。(读 / 写)

RTC_CNTL_FASTMEM_FORCE_PD 快速 RTC 内存强制关闭。(读 / 写)

RTC_CNTL_SLOWMEM_FORCE_LPU RTC 内存强制打开低功耗模式。(读 / 写)

RTC_CNTL_SLOWMEM_FORCE_LPD RTC 内存强制关闭低功耗模式。(读 / 写)

RTC_CNTL_SLOWMEM_FOLW_CPU 1：RTC 内存低功耗模式强制关闭，随 CPU 变化；0：RTC 内存低功耗模式强制关闭 PD，随 RTC 状态机变化。(读 / 写)

RTC_CNTL_FASTMEM_FORCE_LPU 快速 RTC 内存强制打开低功耗模式。(读 / 写)

RTC_CNTL_FASTMEM_FORCE_LPD 快速 RTC 内存强制关闭低功耗模式。(读 / 写)

RTC_CNTL_FASTMEM_FOLW_CPU 1：快速 RTC 内存关闭低功耗模式，随 CPU 变化；0：快速 RTC 内存关闭低功耗模式，随 RTC 状态机变化。(读 / 写)

RTC_CNTL_FORCE_NOISO rtc_peri 强制不隔离。(读 / 写)

RTC_CNTL_FORCE_ISO rtc_peri 强制隔离。(读 / 写)

RTC_CNTL_SLOWMEM_FORCE_ISO RTC 内存强制隔离。(读 / 写)

RTC_CNTL_SLOWMEM_FORCE_NOISO RTC 内存强制不隔离。(读 / 写)

RTC_CNTL_FASTMEM_FORCE_ISO 快速 RTC 内存强制隔离。(读 / 写)

RTC_CNTL_FASTMEM_FORCE_NOISO 快速 RTC 内存强制不隔离。(读 / 写)

Register 31.28. RTC_CNTL_DIG_PWC_REG (0x0084)

RTC_CNTL_DG_WRAP_PD_EN	RTC_CNTL_WIFI_PD_EN	RTC_CNTL_INTER_RAM4_PD_EN	RTC_CNTL_INTER_RAM3_PD_EN	RTC_CNTL_INTER_RAM2_PD_EN	RTC_CNTL_INTER_RAM1_PD_EN	RTC_CNTL_INTER_RAM0_PD_EN	RTC_CNTL_ROM0_PD_EN	(reserved)	RTC_CNTL_DG_WRAP_FORCE_PU	RTC_CNTL_DG_WRAP_FORCE_PD	RTC_CNTL_WIFI_FORCE_PU	RTC_CNTL_WIFI_FORCE_PD	RTC_CNTL_INTER_RAM4_FORCE_PU	RTC_CNTL_INTER_RAM4_FORCE_PD	RTC_CNTL_INTER_RAM3_FORCE_PU	RTC_CNTL_INTER_RAM3_FORCE_PD	RTC_CNTL_INTER_RAM2_FORCE_PU	RTC_CNTL_INTER_RAM2_FORCE_PD	RTC_CNTL_INTER_RAM1_FORCE_PU	RTC_CNTL_INTER_RAM1_FORCE_PD	RTC_CNTL_INTER_RAM0_FORCE_PU	RTC_CNTL_INTER_RAM0_FORCE_PD	RTC_CNTL_ROM0_FORCE_PU	RTC_CNTL_ROM0_FORCE_PD	RTC_CNTL_LSLP_MEM_FORCE_PU	RTC_CNTL_LSLP_MEM_FORCE_PD	(reserved)	
31	30	29	28	27	26	25	24	23 21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2 0	
x	x	x	x	x	x	x	x	0 0 0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0 0 0	Reset

RTC_CNTL_DG_WRAP_PD_EN 睡眠模式下，使能关闭数字内核。(读 / 写)

RTC_CNTL_WIFI_PD_EN 睡眠模式下，使能关闭 Wi-Fi 模块。(读 / 写)

RTC_CNTL_INTER_RAM4_PD_EN 睡眠模式下，使能关闭内部 SRAM 4。(读 / 写)

RTC_CNTL_INTER_RAM3_PD_EN 睡眠模式下，使能关闭内部 SRAM 3。(读 / 写)

RTC_CNTL_INTER_RAM2_PD_EN 睡眠模式下，使能关闭内部 SRAM 2。(读 / 写)

RTC_CNTL_INTER_RAM1_PD_EN 睡眠模式下，使能关闭内部 SRAM 1。(读 / 写)

RTC_CNTL_INTER_RAM0_PD_EN 睡眠模式下，使能关闭内部 SRAM 0。(读 / 写)

RTC_CNTL_ROM0_PD_EN 睡眠模式下，使能关闭 ROM。(读 / 写)

RTC_CNTL_DG_WRAP_FORCE_PU 数字内核强制打开。(读 / 写)

RTC_CNTL_DG_WRAP_FORCE_PD 数字内核强制关闭。(读 / 写)

RTC_CNTL_WIFI_FORCE_PU Wi-Fi 强制打开。(读 / 写)

RTC_CNTL_WIFI_FORCE_PD Wi-Fi 强制关闭。(读 / 写)

RTC_CNTL_INTER_RAM4_FORCE_PU 内部 SRAM 4 强制打开。(读 / 写)

RTC_CNTL_INTER_RAM4_FORCE_PD 内部 SRAM 4 强制关闭。(读 / 写)

RTC_CNTL_INTER_RAM3_FORCE_PU 内部 SRAM 3 强制打开。(读 / 写)

RTC_CNTL_INTER_RAM3_FORCE_PD 内部 SRAM 3 强制关闭。(读 / 写)

RTC_CNTL_INTER_RAM2_FORCE_PU 内部 SRAM 2 强制打开。(读 / 写)

RTC_CNTL_INTER_RAM2_FORCE_PD 内部 SRAM 2 强制关闭。(读 / 写)

RTC_CNTL_INTER_RAM1_FORCE_PU 内部 SRAM 1 强制打开。(读 / 写)

RTC_CNTL_INTER_RAM1_FORCE_PD 内部 SRAM 1 强制关闭。(读 / 写)

RTC_CNTL_INTER_RAM0_FORCE_PU 内部 SRAM 0 强制打开。(读 / 写)

RTC_CNTL_INTER_RAM0_FORCE_PD 内部 SRAM 0 强制关闭。(读 / 写)

寄存器描述下一页继续。

Register 31.28. RTC_CNTL_DIG_PWC_REG (0x0084)

继上一页寄存器描述。

RTC_CNTL_ROM0_FORCE_PU ROM 强制打开。(读 / 写)

RTC_CNTL_ROM0_FORCE_PD ROM 强制关闭。(读 / 写)

RTC_CNTL_LSLP_MEM_FORCE_PU 睡眠模式下，数字内核内存强制打开。(读 / 写)

RTC_CNTL_LSLP_MEM_FORCE_PD 睡眠模式下，数字内核内存强制关闭。(读 / 写)

Register 31.29. RTC_CNTL_DIG_ISO_REG (0x0088)

RTC_CNTL_DG_WRAP_FORCE_NOISO	RTC_CNTL_DG_WRAP_FORCE_ISO	RTC_CNTL_WIFI_FORCE_NOISO	RTC_CNTL_WIFI_FORCE_ISO	RTC_CNTL_INTER_RAM4_FORCE_NOISO	RTC_CNTL_INTER_RAM4_FORCE_ISO	RTC_CNTL_INTER_RAM3_FORCE_NOISO	RTC_CNTL_INTER_RAM3_FORCE_ISO	RTC_CNTL_INTER_RAM2_FORCE_NOISO	RTC_CNTL_INTER_RAM2_FORCE_ISO	RTC_CNTL_INTER_RAM1_FORCE_NOISO	RTC_CNTL_INTER_RAM1_FORCE_ISO	RTC_CNTL_INTER_RAM0_FORCE_NOISO	RTC_CNTL_INTER_RAM0_FORCE_ISO	RTC_CNTL_ROM0_FORCE_NOISO	RTC_CNTL_ROM0_FORCE_ISO	RTC_CNTL_DG_PAD_FORCE_HOLD	RTC_CNTL_DG_PAD_FORCE_UNHOLD	RTC_CNTL_DG_PAD_FORCE_ISO	RTC_CNTL_DG_PAD_FORCE_NOISO	RTC_CNTL_REG_RTC_CNTL_DG_PAD_AUTOHOLD_EN	RTC_CNTL_CLR_REG_RTC_CNTL_DG_PAD_AUTOHOLD	RTC_CNTL_DG_PAD_AUTOHOLD	(reserved)	
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8 0	
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_DG_WRAP_FORCE_NOISO 数字内核强制不隔离。(读 / 写)

RTC_CNTL_DG_WRAP_FORCE_ISO 数字内核强制隔离。(读 / 写)

RTC_CNTL_WIFI_FORCE_NOISO Wi-Fi 模块强制不隔离。(读 / 写)

RTC_CNTL_WIFI_FORCE_ISO Wi-Fi 模块强制隔离。(读 / 写)

RTC_CNTL_INTER_RAM4_FORCE_NOISO 内部 SRAM 4 强制不隔离。(读 / 写)

RTC_CNTL_INTER_RAM4_FORCE_ISO 内部 SRAM 4 强制隔离。(读 / 写)

RTC_CNTL_INTER_RAM3_FORCE_NOISO 内部 SRAM 3 强制不隔离。(读 / 写)

RTC_CNTL_INTER_RAM3_FORCE_ISO 内部 SRAM 3 强制隔离。(读 / 写)

RTC_CNTL_INTER_RAM2_FORCE_NOISO 内部 SRAM 2 强制不隔离。(读 / 写)

RTC_CNTL_INTER_RAM2_FORCE_ISO 内部 SRAM 2 强制隔离。(读 / 写)

RTC_CNTL_INTER_RAM1_FORCE_NOISO 内部 SRAM 1 强制不隔离。(读 / 写)

RTC_CNTL_INTER_RAM1_FORCE_ISO 内部 SRAM 1 强制隔离。(读 / 写)

RTC_CNTL_INTER_RAM0_FORCE_NOISO 内部 SRAM 0 强制不隔离。(读 / 写)

RTC_CNTL_INTER_RAM0_FORCE_ISO 内部 SRAM 0 强制隔离。(读 / 写)

RTC_CNTL_ROM0_FORCE_NOISO ROM 强制不隔离。(读 / 写)

RTC_CNTL_ROM0_FORCE_ISO ROM 强制隔离。(读 / 写)

RTC_CNTL_DG_PAD_FORCE_HOLD 数字管脚强制保持。(读 / 写)

RTC_CNTL_DG_PAD_FORCE_UNHOLD 数字管脚强制解除保持。(读 / 写)

RTC_CNTL_DG_PAD_FORCE_ISO 数字管脚强制隔离。(读 / 写)

RTC_CNTL_DG_PAD_FORCE_NOISO 数字管脚强制不隔离。(读 / 写)

寄存器描述下一页继续。

Register 31.29. RTC_CNTL_DIG_ISO_REG (0x0088)

继上一页寄存器描述。

RTC_CNTL_REG_RTC_CNTL_DG_PAD_AUTOHOLD_EN 数字管脚使能自动保持。（读 / 写）

RTC_CNTL_CLR_REG_RTC_CNTL_DG_PAD_AUTOHOLD 只写寄存器，清除数字管脚自动保持。（只写）

RTC_CNTL_DG_PAD_AUTOHOLD 只读寄存器，表示数字管脚的自动保持状态。（只读）

Register 31.30. RTC_CNTL_WDTCONFIG0_REG (0x008C)

RTC_CNTL_WDT_EN	RTC_CNTL_WDT_STG0	RTC_CNTL_WDT_STG1	RTC_CNTL_WDT_STG2	RTC_CNTL_WDT_STG3	reserved	reserved	RTC_CNTL_WDT_CPU_RESET_LENGTH	RTC_CNTL_WDT_SYS_RESET_LENGTH	RTC_CNTL_WDT_FLASHBOOT_MOD_EN	RTC_CNTL_WDT_PROCPU_RESET_EN	RTC_CNTL_WDT_APPCPU_RESET_EN	RTC_CNTL_WDT_PAUSE_IN_SLP	reserved	
31	30 28	27 25	24 22	21 19	18	17	16 14	13 11	10	9	8	7	6 0	
0	0	0	0	0	0	0	1	1	1	0	0	1	0	Reset

RTC_CNTL_WDT_PAUSE_IN_SLP 置 1 设置“睡眠中 RTC WDT 暂停使用”。（读/写）

RTC_CNTL_WDT_APPCPU_RESET_EN 置 1 允许 RTC WDT 复位 APP_CPU。（读/写）

RTC_CNTL_WDT_PROCPU_RESET_EN 置 1 允许 RTC WDT 复位 PRO_CPU。（读/写）

RTC_CNTL_WDT_FLASHBOOT_MOD_EN 置 1 在芯片从 flash 启动时使能 RTC WDT。（读/写）

RTC_CNTL_WDT_SYS_RESET_LENGTH 设置系统复位计数器的时长，单位：RTC_SLOW_CLK，取值范围：0 ~ 7。（读/写）

RTC_CNTL_WDT_CPU_RESET_LENGTH 设置 CPU 复位计数器的时长单位：RTC_SLOW_CLK，取值范围：0 ~ 7。（读/写）

RTC_CNTL_WDT_STG3 1：在中断阶段使能；2：在 CPU 复位阶段使能；3：在系统复位阶段使能；4：在 RTC 复位阶段使能。（读/写）

RTC_CNTL_WDT_STG2 1：在中断阶段使能；2：在 CPU 复位阶段使能；3：在系统复位阶段使能；4：在 RTC 复位阶段使能。（读/写）

RTC_CNTL_WDT_STG1 1：在中断阶段使能；2：在 CPU 复位阶段使能；3：在系统复位阶段使能；4：在 RTC 复位阶段使能。（读/写）

RTC_CNTL_WDT_STG0 1：在中断阶段使能；2：在 CPU 复位阶段使能；3：在系统复位阶段使能；4：在 RTC 复位阶段使能。（读/写）

RTC_CNTL_WDT_EN 使能 RTC WDT。（读/写）

Register 31.31. RTC_CNTL_WDTCONFIGn_REG (n: 1-4) (0x008C+4*n)

31	0	
0x000000FFF		Reset

RTC_CNTL_WDTCONFIGn_REG 看门狗定时器阶段 n 的保持周期。(读 / 写)

Register 31.32. RTC_CNTL_WDTFEED_REG (0x00A0)

RTC_CNTL_WDT_FEED	(reserved)	
31	30 … 0	
0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_WDT_FEED 软件喂狗。(只写)

Register 31.33. RTC_CNTL_WDTWPROTECT_REG (0x00A4)

31	0	
0x050D83AA1		Reset

RTC_CNTL_WDTWPROTECT_REG 当该寄存器中的值不为 0x50d83aa1 时，使能 RTC 看门狗 (RWDT) 的写保护。(读 / 写)

Register 31.34. RTC_CNTL_SW_CPU_STALL_REG (0x00AC)

RTC_CNTL_SW_STALL_PROCPU_C1	RTC_CNTL_SW_STALL_APPCPU_C1	(reserved)	
31 26	25 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	Reset

RTC_CNTL_SW_STALL_PROCPU_C1 reg_rtc_cntl_sw_stall_procpu_c1[5:0]，当 reg_rtc_cntl_sw_stall_procpu_c0[1:0] == 0x86 (100001 10) 时，PRO_CPU 将暂停工作，请参见 RTC_CNTL_OPTIONS0_REG。（读 / 写）

RTC_CNTL_SW_STALL_APPCPU_C1 reg_rtc_cntl_sw_stall_appcpu_c1[5:0]，当 reg_rtc_cntl_sw_stall_appcpu_c0[1:0] == 0x86 (100001 10) 时，APP_CPU 将暂停工作，请参见 RTC_CNTL_OPTIONS0_REG。（读 / 写）

Register 31.35. RTC_CNTL_STORE*n*_REG (*n*: 4-7) (0x00B0+4**n*)

31 0	
x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x x	Reset

RTC_CNTL_STORE*n*_REG 32 位通用保留寄存器。（读 / 写）

Register 31.36. RTC_CNTL_HOLD_FORCE_REG (0x00C8)

(reserved)	RTC_CNTL_X32N_HOLD_FORCE	RTC_CNTL_X32P_HOLD_FORCE	RTC_CNTL_TOUCH_PAD7_HOLD_FORCE	RTC_CNTL_TOUCH_PAD6_HOLD_FORCE	RTC_CNTL_TOUCH_PAD5_HOLD_FORCE	RTC_CNTL_TOUCH_PAD4_HOLD_FORCE	RTC_CNTL_TOUCH_PAD3_HOLD_FORCE	RTC_CNTL_TOUCH_PAD2_HOLD_FORCE	RTC_CNTL_TOUCH_PAD1_HOLD_FORCE	RTC_CNTL_TOUCH_PAD0_HOLD_FORCE	RTC_CNTL_SENSE4_HOLD_FORCE	RTC_CNTL_SENSE3_HOLD_FORCE	RTC_CNTL_SENSE2_HOLD_FORCE	RTC_CNTL_SENSE1_HOLD_FORCE	RTC_CNTL_PDAC2_HOLD_FORCE	RTC_CNTL_PDAC1_HOLD_FORCE	RTC_CNTL_ADC2_HOLD_FORCE	RTC_CNTL_ADC1_HOLD_FORCE	
31 18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Reset

RTC_CNTL_X32N_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_X32P_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD7_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD6_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD5_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD4_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD3_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD2_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD1_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_TOUCH_PAD0_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_SENSE4_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_SENSE3_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_SENSE2_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_SENSE1_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

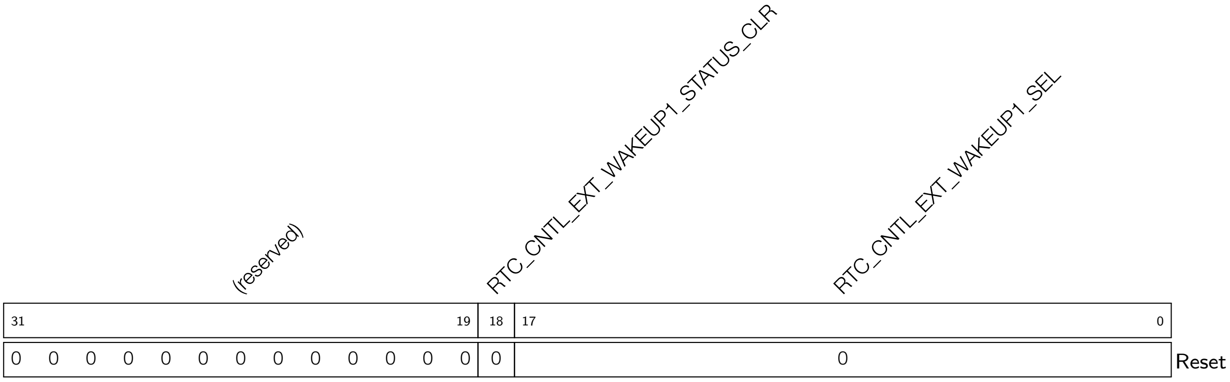
RTC_CNTL_PDAC2_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_PDAC1_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_ADC2_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

RTC_CNTL_ADC1_HOLD_FORCE 休眠状态下，此位置 1 保持管脚状态。(读 / 写)

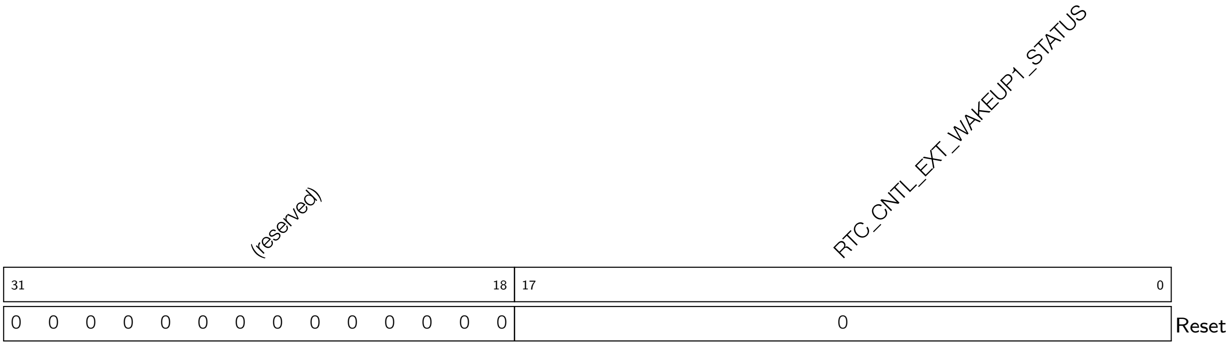
Register 31.37. RTC_CNTL_EXT_WAKEUP1_REG (0x00CC)



RTC_CNTL_EXT_WAKEUP1_STATUS_CLR 清除外部唤醒源 1 的状态。(只写)

RTC_CNTL_EXT_WAKEUP1_SEL 为外部唤醒源选择 RTC 管脚的位图。(读 / 写)

Register 31.38. RTC_CNTL_EXT_WAKEUP1_STATUS_REG (0x00D0)



RTC_CNTL_EXT_WAKEUP1_STATUS 外部唤醒源 1 的状态。(只读)

Register 31.39. RTC_CNTL_BROWN_OUT_REG (0x00D4)

RTC_CNTL_BROWN_OUT_DET	RTC_CNTL_BROWN_OUT_ENA	RTC_CNTL_DBROWN_OUT_THRES	RTC_CNTL_BROWN_OUT_RST_ENA	RTC_CNTL_BROWN_OUT_RST_WAIT	RTC_CNTL_BROWN_OUT_PD_RF_ENA	RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA	(reserved)	
31	30	29 27	26	25 16	15	14	13 0	
0	0	0x2	0	0x3FF	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0	Reset

RTC_CNTL_BROWN_OUT_DET 欠压检测。(只读)

RTC_CNTL_BROWN_OUT_ENA 使能欠压检测。(读 / 写)

RTC_CNTL_DBROWN_OUT_THRES 欠压阈值。当电源电压低于此欠压阈值时，欠压检测器将复位芯片。请注意，每个 ESP32 芯片之间可能存在欠压阈值的差异。0：2.43 V ± 0.05；1：2.48 V ± 0.05；2：2.58 V ± 0.05；3：2.62 V ± 0.05；4：2.67 V ± 0.05；5：2.70 V ± 0.05；6：2.77 V ± 0.05；7：2.80 V ± 0.05。(读 / 写)

RTC_CNTL_BROWN_OUT_RST_ENA 使能欠压复位。(读 / 写)

RTC_CNTL_BROWN_OUT_RST_WAIT 欠压复位等待周期。(读 / 写)

RTC_CNTL_BROWN_OUT_PD_RF_ENA 当出现欠压时，使能关闭 RF。(读 / 写)

RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA 当出现欠压时，使能关闭 flash（向 flash 发送 suspend 指令）。(读 / 写)

词汇列表

外设相关词汇

AES	AES 加速器
AHB	高级高性能总线 (Advanced High-performance Bus)。更多信息见 AHB 规范。
APB	高级外围总线 (Advanced Peripheral Bus)。更多信息见 APB 规范。
DMA	DMA 控制器
DPORT	数据端口 (Data Port) 的缩写，全称为 XLMI 端口 (Xtensa Local Memory Interface)。
eFuse	eFuse 控制器
EMAC	以太网 MAC
I2C	I2C 控制器
I2S	I2S 控制器
LEDC	LED 控制 PWM
MCPWM	电机控制 PWM
MMU	存储器管理单元
MPU	存储器保护单元
PCNT	脉冲计数器控制器
PERI	外设
PMU	电源管理单元
PID	进程号
RMT	红外遥控
RNG	随机数生成器
RSA	RSA 加速器
RTC	实时控制器。SOC 中的一组电路，在任何芯片模式下都能随时保持工作
SDHOST	SD/MMC 主机控制器
SHA	SHA 加速器
SPI	SPI 控制器
TIMG	定时器组
TWAI	双线汽车接口
UART	UART 控制器
ULP 协处理器	超低功耗协处理器
USB OTG	USB On-The-Go
WDT	看门狗定时器

寄存器相关词汇

ISO	隔离。当模块断电时，其输出的引脚将处于未知状态（某些中间电压）。“ISO”寄存器将使其输出引脚隔离在一个确定的电压，从而不会影响其他未掉电的工作模块的状态。
NMI	不可屏蔽中断。
REG	寄存器。
R/W	读/写，软件可读写这些位。
RO	只读，软件只能读这些位。
WO	只写，软件只能写这些位。

修订历史

日期	版本	发布说明
2024.04	v5.1	• 章节 4 *IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)*： – 更新 4.7 中的描述 – 更新以下寄存器的描述：RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7)、RTCIO_TOUCH_PAD*m*_REG (*m* = 8, 9) • 章节 7 *SPI 控制器 (SPI)*：标记寄存器 SPI_CTRL2_REG 中的 bit12 ~ bit15 为保留位 • 章节 8 *SDIO 从机控制器*：修正几处笔误 • 章节 10 *以太网 (MAC)*：删除寄存器 EMACCSTATUS_REG，将其标记为 reserved • 章节 12 *I2S 控制器 (I2S)*：更新 12.6.2 小节中的中断描述 • 章节 14 *LED PWM 控制器 (LEDC)*：更新 14.2.3 中关于不支持暂停 LEDC 渐变的说明，并更新表 14-1 中的最低精度 • 章节 18 *定时器组 (TIMG)*：更新 TIMG*n*_T*x*_WDT_CLK_PRESCALE 的描述 • 章节 29 *片上传感器与模拟信号处理*：更新寄存器 APB_SARADC_FSM_REG • 章节 31 *低功耗管理 (RTC_CNTL)*：更新寄存器 RTC_CNTL_WDTWPROTECT_REG 的描述
2023.07	v5.0	• 全文更新 APB_CTRL 寄存器前缀为 SYSCON • 章节 1.3.4 *Cache*：说明 cache 的块大小为 32 字节 • 章节 4 *IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)*：将原来的 RTCIO_TOUCH_PAD*n*_REG (*n*: 0-9) 寄存器分成两个：RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7) 和 RTCIO_TOUCH_PAD*m*_REG (*m*: 8-9)；在 RTCIO_TOUCH_PAD*n*_REG (*n*: 0-7) 中增加 bit [27-31]、[12-16] • 章节 4.5.2 *模拟功能描述*：修改描述 • 章节 5 *DPort 寄存器*：增加两个寄存器 DPORT_PRO_CACHE_CTRL1_REG、DPORT_APP_CACHE_CTRL1_REG 的描述 • 章节 5.3.7 *外设时钟门控和复位*：增加说明“使用外设前必须先把时钟打开、复位释放” • 章节 6.5 *SPI DMA 控制器*：将 SPI DMA 一次接收/发送的数据长度改为“应当与 4 字节对齐” • 章节 10 *以太网 (MAC)*：删除时间戳/PTP 功能相关内容，原因是硬件暂不支持此功能 • 章节 17 *脉冲计数器 (PCNT)*：增加 PCNT_CLK_EN 位的描述

见下页

续上页

日期	版本	发布说明

2023.04	v4.9	- 根据 PCN20221202 删除霍尔传感器相关内容、寄存器、信号等
		- 将全文出现的 PLL_D2_CLK 改为 PLL_F160M_CLK
		- 章节 4 *IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)*：在表 4-2 中增加 TWAI 信号
		- 在章节 13 *UART 控制器 (UART)* 新增终止状态 (break condition) 的相关描述，更新停止位最大位数和相关描述
		- 在章节 14 *LED PWM 控制器 (LEDC)* 新增占空比精度计算公式，并更新表 *分频器*
		- 章节 29 *片上传感器与模拟信号处理*：
		  – 在小节 29.2.2 *主要特性* 中增加应用限制的说明
		  – 删除内部信号 vdd33、pa_pkdet1、pa_pkdet2
		- 在章节 31 *低功耗管理 (RTC_CNTL)* 中增加有关“拒绝睡眠”的描述

2022.12	v4.8	更新以下章节： - 在章节 3 *复位和时钟* 中更新小节 3.2.4.2，新增寄存器小节 3.3 和 3.4 - 更新章节 10 *以太网 (MAC)* 中 MIICSRCLK 和 MIIBUSY 字段的描述 - 在章节 13 *UART 控制器 (UART)* 新增 UART_MEM_TX_STATUS_REG 和 UART_MEM_RX_STATUS_REG 寄存器的描述 - 更新 25 *随机数发生器 (RNG)* 中的时钟名称 - 更新 29 *片上传感器与模拟信号处理* 中的表 29-2 - 更新 31 *低功耗管理 (RTC_CNTL)* 中的时钟名称，并增加有关 RTC_CNTL_MAIN_STATE_IN_IDLE 的描述 - 将 20 *eFuse 控制器 (eFuse)* 中的 RTC8M_CLK 重命名为 RC_FAST_CLK - 在所有章节的寄存器小节增加了有关“相对地址”的说明

见下页

续上页

日期	版本	发布说明
2022.08	v4.7	• 更新章节 1 *系统和存储器* 中的表 1-6 • 更新章节 4 *IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)*： – 修正寄存器列表中的一处笔误 – 更新第 4.2.2 小节中的描述 – 更新对 GPIO_PIN*n*_INT_ENA 和 IO_MUX_PIN_CTRL 的描述 • 在章节 6 *DMA 控制器 (DMA)* 中的 6.4 小节新增一条关于接收链表描述符的说明 • 更新章节 7 *SPI 控制器 (SPI)* 中对 SPI_TRANS_DONE、SPI_SLV_WR_STA_DONE、SPI_SLV_RD_STA_DONE、SPI_SLV_WR_BUF_DONE、SPI_SLV_RD_BUF_DONE 的描述 • 更新时钟名称： – RTC8M_CLK 更新为 RC_FAST_CLK – RTC8M_D256_CLK 更新为 RC_FAST_DIV_CLK – RTC_CLK 更新为 RC_SLOW_CLK – SLOW_CLK 更新为 RTC_SLOW_CLK – FAST_CLK 更新为 RTC_FAST_CLK • 修正章节 12 *I2S 控制器 (I2S)* 中寄存器列表中的一处笔误 • 更新章节 18 *定时器组 (TIMG)* 中的寄存器列表 • 在章节 29 *片上传感器与模拟信号处理* 中新增两条说明
2021.11	v4.6	更新章节 1 *系统和存储器* 中的表 1-6 更新章节 2 *中断矩阵 (INTERRUPT)* 中的表 2-1 更新章节 3 *复位和时钟* 中的图 3-1 更新章节 4 *IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)* 中的表 4-2 和对 IO_MUX_PIN_CTRL 的描述 全面更新章节 5 *DPort 寄存器* 在章节 12 *I2S 控制器 (I2S)* 的表 12-1 下方新增一条说明 更新章节 8 *SDIO 从机控制器* 中的 8.3.6 小节和寄存器 SLCHOST_CONF_REG 更新章节 26 *片外存储器加密与解密 (FLASH)* 的描述 更新章节 29 *片上传感器与模拟信号处理* 中的 29.4.3 小节 在章节 4、章节 7、章节 12、章节 15、章节 29 和章节 30 中增加对寄存器地址的说明 翻译多个章节中的寄存器列表 更新词汇列表 修正笔误

见下页

续上页

日期	版本	发布说明
2021.07	v4.5	增加章节 10.6.4 在表 4-4 下方增加一条说明 在章节 14 *LED PWM 控制器 (LEDC)* 更新表 14-1 更新章节 3.2.7 的描述 将章节 10.9 中的 Ethernet MAC 基地址更新为 0x3FF6_9000 在章节 13 *UART 控制器 (UART)* 更新 UART_SW_RTS、UART_RX_FLOW_THRHD 字段的描述 在章节 21 *双线汽车接口 (TWAI)* 更新特性描述 更新章节 25 *随机数发生器 (RNG)* 更新章节 30.4.2 的描述 在章节 5 *DPort 寄存器* 和章节 2 *中断矩阵 (INTERRUPT)* 中添加中断矩阵寄存器的基地址信息 更新章节 4 *IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)* 中管脚功能编号，编号从 Function0 开始 删除章节 13.3.4 中重复的内容 修正表 2-1 和章节 6.4 中的笔误
2021.03	V4.4	更新章节 16：*电机控制脉宽调制器 (PWM)* 中寄存器 PWM_TIMER0_SYNC_REG ~ PWM_TIMER2_SYNC_REG 的描述 更新章节 18：*定时器组 (TIMG)* 中 TIMG*n*_RTCCALICFG_REG 寄存器和 TIMG*n*_RTCCALICFG1_REG 寄存器的描述 更新章节 31：*低功耗管理 (RTC_CNTL)* 中 RTC_CNTL_WDTCONFIG0_REG 寄存器的描述 更新章节 10：*以太网 (MAC)* 中对 EMACMIIADDR_REG 的描述 更新章节 4：*IO_MUX 和 GPIO 交换矩阵 (GPIO, IO_MUX)* 中对 IO_MUX_*x*_REG (*x*: GPIO0-GPIO39) 和 IO_MUX_*x*_REG (*x*: GPIO0-GPIO39) 的描述 更新章节 21：*双线汽车接口 (TWAI)* 中所用的商标符号 更新章节 31：*低功耗管理 (RTC_CNTL)* 中 RTC_CNTL_ULP_CP_TIMER_REG 寄存器的描述 将文档写作规范章节重命名为词汇列表，并移至文档结尾 更新表 4-3：*IO_MUX Pad 列表*下方的说明
2020.09	V4.3	新增章节 21 *双线汽车接口 (TWAI)* 在章节 *eFuse 控制器* 中新增 uart_download_dis 系统参数的相关信息。 在章节 25 *随机数发生器 (RNG)* 中新增 25.4 *编程指南* 小节，并更新部分描述。 更新对寄存器 SPI_ADDR_REG 和 SPI_SLV_WR_STATUS_REG 的描述

见下页

续上页

日期	版本	发布说明
2020.06	V4.2	新增文档写作规范章节 更新章节*系统和存储器*： • 在表 1-6 下方添加关于 DPORT 和 AHB 地址空间的说明 更新章节*复位和时钟*： • 更新表 3-3：增加 PLL_CLK 频率为 480 MHz，CPU_CLK 频率为 240 MHz 的描述 • 更新表 3-5：将 CPU_CLK 源为 PLL_CLK 时 APB_CLK 的频率改为 80 MHz 更新章节 *IO_MUX 和 GPIO 交换矩阵*： • 修正章节 4.4.2 中的一处笔误：对于输入信号，必须清零 SIG_IN_SEL 寄存器，直接将输入信号输出到外设。 • 修改表 4-3 中 MTCK、MTMS、GPIO27 的 reset 状态 • 更新寄存器 FUN_DRV 的描述 更新章节 *I²S*： • 更新章节 12.4.1.1 更新章节 UART 控制器： • 更新章节 13.3.3 • 更新寄存器 UART_FIFO_REG 和 UART_RX_TOUT_THRHD 的描述 更新章节 *LED_PWM*： • 增加表 14-1 更新章节*电机控制脉宽调制器 (MCPWM)*： • 修正在递增计数模式、递减计数模式、递增-递减循环模式下 PWM 的周期值 更新章节 *PULSE_CNT*： • 增加寄存器 PCNT_U*n*_STATUS_REG 的描述 更新章节 *eFuse 控制器*： • 将两个系统参数“32pad”和“chip_version”合并成一个：pkg_version • 更新寄存器 EFUSE_RD_CHIP_VER_PKG 和 EFUSE_CHIP_VER_PKG 的描述 更新章节*片上传感器与模拟信号处理*： • 修正几处笔误 更新章节*超低功耗协处理器*： • 更新章节 30.4.13 和 30.4.14 中的描述 • 修正几处笔误 更新章节低功耗管理： • 增加寄存器 RTC_CNTL_WDTCONFIG0_REG 的描述 • 修改寄存器 RTC_CNTL_WDTCONFIG*n*_REG 的描述

见下页

续上页

日期	版本	发布说明
2019.11	V4.1	更新章节 *IO_MUX 和 GPIO 交换矩阵*： • 更新表 4-4； • 在寄存器 RTCIO_TOUCH_PAD*n*_REG 描述中增加 RTCIO_TOUCH_PAD*n*_FUN_SEL 及其说明； 更新章节 *SPI*： • 修正表 7.7 中 SPI2、SPI3 的错误地址； 更新章节 *I^{2}C 控制器*： • 删除 I2C_SLAVE_TRAN_COMP_INT 中断； 更新章节 *I^{2}S*： • 在表 12-1 下增加一条说明； 更新章节 *UART 控制器*： • 修正 UART_FORCE_XOFF 和 UART_FORCE_XON 的寄存器描述； • 修正 ART_SWFC_CONF_REG 的寄存器描述； 更新章节*红外遥控*： • 更新图 15-1； 更新章节 *PULSE_CNT*： • 更新图 17-1； • 修正寄存器 PCNT_U*n*_CONF0_REG 描述里的笔误； 更新章节 *eFuse 控制器*： • 新增并修改 8 个系统参数；更新相应寄存器； • 更新表 20-4 中的寄存器配置值； • 将系统参数 flash_crypt_cnt 的位宽改为 7 位； 更新章节 *PID/MPU/MMU*： • 在表格 27-8 下方增加一条说明； 更新章节*片上传感器与模拟信号处理*： • 修正 SENS_SAR2_BIT_WIDTH 和 SENS_SAR1_BIT_WIDTH 的寄存器描述； 更新章节*超低功耗协处理器*： • 删除 TSENS 指令； • 修正 REG_WR 的 OpCode； • 更新章节 30.6.2.4； • 修正 RTC_I2C_RX_LSB_FIRST 和 RTC_I2C_TX_LSB_FIRST 的寄存器描述； • 删除 RTC_I2C_SLAVE_TRAN_COMP_INT_ENA 和 RTC_I2C_SLAVE_TRAN_COMP_INT_ST 的寄存器描述； 更新章节*低功耗管理*： • 修正寄存器 RTC_CNTL_DBROWN_OUT_THRES 的默认值和描述； • 修正寄存器 RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA 的描述； 增加文档反馈链接。

见下页

续上页

日期	版本	发布说明
2018.12	V4.0	更新章节 IO_MUX 和 GPIO 交换矩阵 中几处寄存器名称，以跟头文件保持一致； 更新章节 7 SPI： • 更新章节 7.3； • 更新章节 7.5.1； • 更新章节 7.8； 更新章节 13 UART 控制器： • 将 UART 支持的停止位修正为 1/1.5/2/3 个； • 在章节 13.3.2 末段增加一则说明； • 更新寄存器 UART_DL0_EN 的描述； 在低功耗管理一章中唤醒源的末段增加一则说明。
2018.10	V3.9	更新章节 11：I²C 控制器中图 11-3：I²C 时序图。
2018.09	V3.8	更新寄存器 TIMG*n*_T*x*_ALARM_EN 的描述； 在章节 30.5 中增加关于 ULP 协处理器唤醒时间的描述。
2018.08	V3.7	更新寄存器 UART_RX_GAP_TOUT 的描述。
2018.08	V3.6	更新章节 30.4.6 中跳转条件； 更新寄存器 UART_ACTIVE_THRESHOLD 的描述。
2018.07	V3.5	更新章节 15 RMT： • 更新章节 15.2.2：RMT RAM 中 RAM 的起始地址； • 修正几处 RMT 寄存器地址错误； • 更新寄存器 RMT_APB_CONF_REG 的描述。 更新寄存器 UART_RX_TOUT_THRHD、UART_RXFIFO_FULL_INT_CLR、UART_RXFIFO_FULL_INT_CLR 的描述。
2018.06	V3.4	更新章节 4.8 ESP32 I/O Pad 供电中的图； 更新章节 11.3.3 I²C 总线时序； 在章节 14.2.3 LEDC 通道中增加说明； 更新章节 17.2.3 PULSE_CNT 观察点中的 “最大计数值”； 删除有关温度传感器和超低噪声前置模拟放大器相关的内容。
2018.05	V3.3	更新 低功耗管理 章中 寄存器列表 节和 寄存器 节中的寄存器地址。
2018.04	V3.2	更新图 8-3 CMD53 内容。 在章节 Ethernet MAC 中增加以下 6 个寄存器： • DMAOPERATION_MODE_REG; • DMAIN_EN_REG, • DMAMISSEDFR_REG, • PMT_RWUFFR_REG, • PMT_CSR_REG, • EMACLPI_CSR_REG, and • EMACLPITIMERSCONTROL_REG.
2018.04	V3.1	更新图 15-1 RMT 架构； 在章节 4.7 中增加说明； 在章节 4.46 中增加对寄存器位的描述；
2018.03	V3.0	更新章节 30.4.2 中 ST 指令的定义图；

见下页

续上页

日期	版本	发布说明
		在章节 10.9 和 10.10 中增加寄存器 EMACADDR2HIGH_REG 到 EMACADDR7LOW_REG 说明。
2018.02	V2.9	更新章节 4.2.2、4.2.3、4.3.2； 在章节 I2S 寄存器 中增加 I2S_FIFO_WR_REG 和 I2S_FIFO_RD_REG 寄存器。
2018.01	V2.8	增加章节以太网 MAC。 在章节 eFuse 控制器 中增加系统参数 BLK3_part_reserve 的描述。
2017.12	V2.7	在章节 系统和存储器 中增加小节 Cache； 更新章节 分频器 及 LED_PWM 中多个寄存器名称； 更新章节 eFuse 控制器 中寄存器 console_debug_disable 的描述。
2017.11	V2.6	更新章节红外遥控： • 更新图 15-1 RMT 架构； • 更新章节 RMT RAM； • 更新章节 发射器； • 更新中断 RMT_CH*n*_TX_THR_EVENT_INT 的描述。 在章节 UART RAM 和寄存器 UART_CONF0_REG 中增加说明。
2017.11	V2.5	更新章节 SPI 寄存器列表中寄存器 SPI_CTRL_REG 的地址描述； 在章节 SD/MMC 主机控制器中增加时钟相位选择，增加寄存器 CLK_EDGE_SEL 的说明； 关于 I2C 控制器章节的重大修订。
2017.09	V2.4	在章节 SDIO 从机 中增加寄存器 SLC0HOST_TOKEN_RDATA 的描述； 在章节 I2S 模块时钟中增加注意事项； 在章节 GP-SPI 主机模式中增加说明； 增加章节 DPort 寄存器； 增加章节 DMA 控制器。
2017.08	V2.3	增加章节 Flash 加密与解密。
2017.07	V2.2	增加章节 低功耗管理。
2017.07	V2.1	更新章节 IO_MUX 和 GPIO 交换矩阵中 GPIO 配置/数据寄存器和 GPIO RTC 功能配置寄存器的地址； 增加章节 PID 控制器。
2017.07	V2.0	增加章节 SDIO 从机。
2017.06	V1.9	更新章节 IO_MUX 和 GPIO 交换矩阵； 增加章节 电机控制脉宽调制器（MCPWM）。
2017.06	V1.8	在章节 I2S 中增加寄存器 I2S_STATE_REG 的描述； 更新章节 IO_MUX 和 GPIO 交换矩阵； 增加章节 超低功耗协处理器。
2017.05	V1.7	增加章节 片上传感器与模拟信号处理； 增加章节 音频 PLL； 更新章节 eFuse 控制器寄存器列表； 更新章节 I2S PDM 模式和 LCD 模式； 更新章节：GP-SPI 从机支持的通信格式。
2017.03	V1.6	增加章节 SD/MMC 主机控制器； 在章节 IO_MUX 和 GPIO 交换矩阵中增加寄存器 IO_MUX_PIN_CTRL 的描述。

见下页

续上页

日期	版本	发布说明
2017.03	V1.5	增加章节 I2S。
2017.01	V1.4	增加章节 SPI； 增加章节 UART 控制器。
2016.12	V1.3	增加章节 eFuse 控制器； 增加章节 RSA 加速器； 增加章节 随机数发生器； 更新章节 I2C 控制器中断 和 I2C 控制器寄存器。
2016.11	V1.2	增加章节 PID/MPU/MMU； 更新章节 IO_MUX 和 GPIO 交换矩阵寄存器列表； 更新章节 LED_PWM 寄存器列表。
2016.09	V1.1	增加章节 I2C 控制器。
2016.08	V1.0	首次发布。

www.espressif.com

免责声明和版权公告