ESP32 技术参考手册

Espressif Systems

2016 年 8 月 31 日

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。

发布说明

日期	版本	发布说明
2016.08	V1.0	首次发布。

免责申明和版权公告

本文中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。文档“按现状”提供，不负任何担保责任，包括对适销性、适用于特定用途或非侵权性的任何担保，和任何提案、规格或样品在他处提到的任何担保。

本文档不负任何责任，包括使用本文档内信息产生的侵犯任何专利权行为的责任。本文档在此未以禁止反言或其他方式授予任何知识产权使用许可，不管是明示许可还是暗示许可。Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

目录

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 描述了系统结构。图 2 描述了地址映射结构。

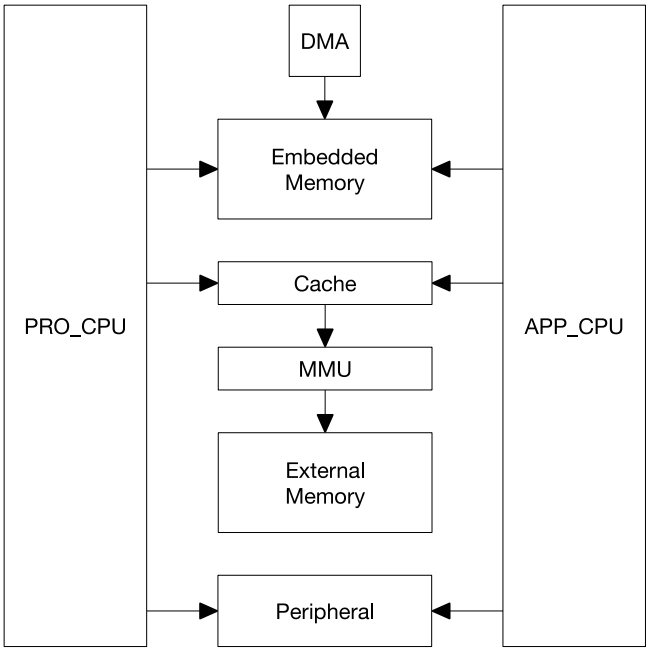


图 1: 系统结构

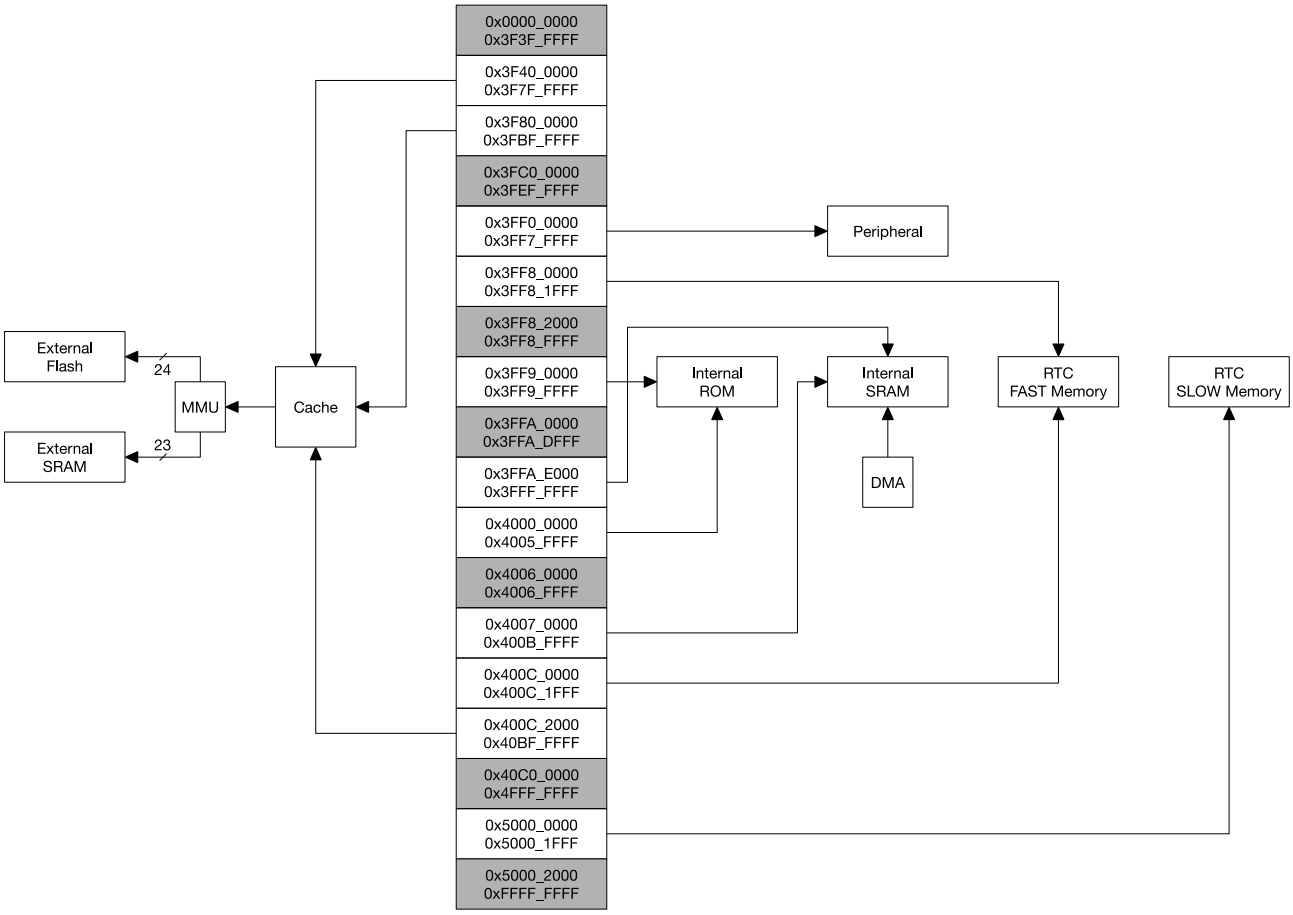


图 2: 地址映射结构

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 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: 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 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 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 3 列出了所有具有 DMA 功能的模块。

表 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 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 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 外设

ESP32 共有 41 个外设模块。表 5 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 5: 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	PWM0	
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	
数据	0x3FF6_6000	0x3FF6_6FFF	4 KB	SYSCON	

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_7000	0x3FF6_7FFF	4 KB	I2C1	
数据	0x3FF6_8000	0x3FF6_8FFF	4 KB	SDMMC	
数据	0x3FF6_9000	0x3FF6_AFFF	8 KB	EMAC	
	0x3FF6_B000	0x3FF6_BFFF	4 KB	保留	
数据	0x3FF6_C000	0x3FF6_CFFF	4 KB	PWM1	
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	PWM2	
数据	0x3FF7_0000	0x3FF7_0FFF	4 KB	PWM3	
	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	

1.3.4.1 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。

1.3.4.2 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。

1.3.4.3 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

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 3 所示。

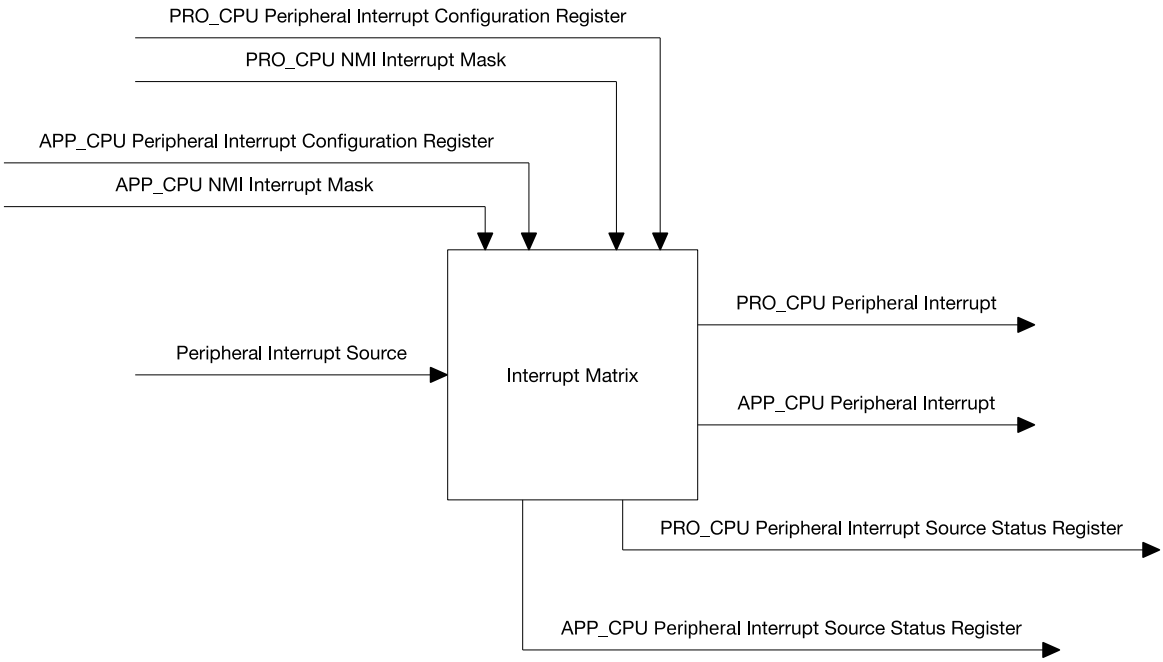


图 3: 中断矩阵结构图

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 6 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 6: 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			Peripheral Interrupt Source			APP_CPU		
Peripheral Interrupt Configuration Register	Bit	Status Register Name	No.	Name	No.	Status Register Name	Bit	Peripheral Interrupt Configuration Register
PRO_MAC_INTR_MAP_REG	0	PRO_INTR_STATUS_REG_0	0	MAC_INTR	0	APP_INTR_STATUS_REG_0	0	APP_MAC_INTR_MAP_REG
PRO_MAC_NMI_MAP_REG	1		1	MAC_NMI	1		1	APP_MAC_NMI_MAP_REG
PRO_BB_INT_MAP_REG	2		2	BB_INT	2		2	APP_BB_INT_MAP_REG
PRO_BT_MAC_INT_MAP_REG	3		3	BT_MAC_INT	3		3	APP_BT_MAC_INT_MAP_REG
PRO_BT_BB_INT_MAP_REG	4		4	BT_BB_INT	4		4	APP_BT_BB_INT_MAP_REG
PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5		5	APP_BT_BB_NMI_MAP_REG
PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6		6	APP_RWBT_IRQ_MAP_REG
PRO_BT_BB_NMI_MAP_REG	5		5	BT_BB_NMI	5		5	APP_BT_BB_NMI_MAP_REG
PRO_RWBT_IRQ_MAP_REG	6		6	RWBT_IRQ	6		6	APP_RWBT_IRQ_MAP_REG
PRO_RWBLE_IRQ_MAP_REG	7		7	RWBLE_IRQ	7		7	APP_RWBLE_IRQ_MAP_REG
PRO_RWBT_NMI_MAP_REG	8		8	RWBT_NMI	8		8	APP_RWBT_NMI_MAP_REG
PRO_RWBLE_NMI_MAP_REG	9		9	RWBLE_NMI	9		9	APP_RWBLE_NMI_MAP_REG
PRO_SLC0_INTR_MAP_REG	10		10	SLC0_INTR	10		10	APP_SLC0_INTR_MAP_REG
PRO_SLC1_INTR_MAP_REG	11		11	SLC1_INTR	11		11	APP_SLC1_INTR_MAP_REG
PRO_UHCI0_INTR_MAP_REG	12		12	UHCI0_INTR	12		12	APP_UHCI0_INTR_MAP_REG
PRO_UHCI1_INTR_MAP_REG	13		13	UHCI1_INTR	13		13	APP_UHCI1_INTR_MAP_REG
PRO_TG_T0_LEVEL_INT_MAP_REG	14		14	TG_T0_LEVEL_INT	14		14	APP_TG_T0_LEVEL_INT_MAP_REG
PRO_TG_T1_LEVEL_INT_MAP_REG	15		15	TG_T1_LEVEL_INT	15		15	APP_TG_T1_LEVEL_INT_MAP_REG
PRO_TG_WDT_LEVEL_INT_MAP_REG	16		16	TG_WDT_LEVEL_INT	16		16	APP_TG_WDT_LEVEL_INT_MAP_REG
PRO_TG_LACT_LEVEL_INT_MAP_REG	17		17	TG_LACT_LEVEL_INT	17		17	APP_TG_LACT_LEVEL_INT_MAP_REG
PRO_TG1_T0_LEVEL_INT_MAP_REG	18		18	TG1_T0_LEVEL_INT	18		18	APP_TG1_T0_LEVEL_INT_MAP_REG
PRO_TG1_T1_LEVEL_INT_MAP_REG	19		19	TG1_T1_LEVEL_INT	19		19	APP_TG1_T1_LEVEL_INT_MAP_REG
PRO_TG1_WDT_LEVEL_INT_MAP_REG	20		20	TG1_WDT_LEVEL_INT	20		20	APP_TG1_WDT_LEVEL_INT_MAP_REG
PRO_TG1_LACT_LEVEL_INT_MAP_REG	21		21	TG1_LACT_LEVEL_INT	21		21	APP_TG1_LACT_LEVEL_INT_MAP_REG
PRO_GPIO_INTERRUPT_PRO_MAP_REG	22		22	**GPIO_INTERRUPT_PRO** / **GPIO_INTERRUPT_APP**	22		22	APP_GPIO_INTERRUPT_APP_MAP_REG
PRO_GPIO_INTERRUPT_PRO_NMI_MAP_REG	23		23	**GPIO_INTERRUPT_PRO_NMI** / **GPIO_INTERRUPT_APP_NMI**	23		23	APP_GPIO_INTERRUPT_APP_NMI_MAP_REG
PRO_CPU_INTR_FROM_CPU_0_MAP_REG	24		24	CPU_INTR_FROM_CPU_0	24		24	APP_CPU_INTR_FROM_CPU_0_MAP_REG
PRO_CPU_INTR_FROM_CPU_1_MAP_REG	25		25	CPU_INTR_FROM_CPU_1	25		25	APP_CPU_INTR_FROM_CPU_1_MAP_REG
PRO_CPU_INTR_FROM_CPU_2_MAP_REG	26		26	CPU_INTR_FROM_CPU_2	26		26	APP_CPU_INTR_FROM_CPU_2_MAP_REG
PRO_CPU_INTR_FROM_CPU_3_MAP_REG	27		27	CPU_INTR_FROM_CPU_3	27		27	APP_CPU_INTR_FROM_CPU_3_MAP_REG
PRO_SPI_INTR_0_MAP_REG	28		28	SPI_INTR_0	28		28	APP_SPI_INTR_0_MAP_REG
PRO_SPI_INTR_1_MAP_REG	29		29	SPI_INTR_1	29		29	APP_SPI_INTR_1_MAP_REG
PRO_SPI_INTR_2_MAP_REG	30		30	SPI_INTR_2	30		30	APP_SPI_INTR_2_MAP_REG
PRO_SPI_INTR_3_MAP_REG	31		31	SPI_INTR_3	31		31	APP_SPI_INTR_3_MAP_REG
PRO_I2S0_INT_MAP_REG	0	PRO_INTR_STATUS_REG_1	32	I2S0_INT	32	APP_INTR_STATUS_REG_1	0	APP_I2S0_INT_MAP_REG
PRO_I2S1_INT_MAP_REG	1		33	I2S1_INT	33		1	APP_I2S1_INT_MAP_REG
PRO_UART_INTR_MAP_REG	2		34	UART_INTR	34		2	APP_UART_INTR_MAP_REG
PRO_UART1_INTR_MAP_REG	3		35	UART1_INTR	35		3	APP_UART1_INTR_MAP_REG
PRO_UART2_INTR_MAP_REG	4		36	UART2_INTR	36		4	APP_UART2_INTR_MAP_REG
PRO_SDIO_HOST_INTERRUPT_MAP_REG	5		37	SDIO_HOST_INTERRUPT	37		5	APP_SDIO_HOST_INTERRUPT_MAP_REG
PRO_EMAC_INT_MAP_REG	6		38	EMAC_INT	38		6	APP_EMAC_INT_MAP_REG
PRO_PWM0_INTR_MAP_REG	7		39	PWM0_INTR	39		7	APP_PWM0_INTR_MAP_REG
PRO_PWM1_INTR_MAP_REG	8		40	PWM1_INTR	40		8	APP_PWM1_INTR_MAP_REG
PRO_PWM2_INTR_MAP_REG	9		41	PWM2_INTR	41		9	APP_PWM2_INTR_MAP_REG
PRO_PWM3_INTR_MAP_REG	10		42	PWM3_INTR	42		10	APP_PWM3_INTR_MAP_REG
PRO_LEDC_INT_MAP_REG	11		43	LEDC_INT	43		11	APP_LEDC_INT_MAP_REG
PRO_EFUSE_INT_MAP_REG	12		44	EFUSE_INT	44		12	APP_EFUSE_INT_MAP_REG
PRO_CAN_INT_MAP_REG	13		45	CAN_INT	45		13	APP_CAN_INT_MAP_REG
PRO_RTC_CORE_INTR_MAP_REG	14		46	RTC_CORE_INTR	46		14	APP_RTC_CORE_INTR_MAP_REG
PRO_RMT_INTR_MAP_REG	15		47	RMT_INTR	47		15	APP_RMT_INTR_MAP_REG
PRO_PCNT_INTR_MAP_REG	16		48	PCNT_INTR	48		16	APP_PCNT_INTR_MAP_REG
PRO_I2C_EXT0_INTR_MAP_REG	17		49	I2C_EXT0_INTR	49		17	APP_I2C_EXT0_INTR_MAP_REG
PRO_I2C_EXT1_INTR_MAP_REG	18		50	I2C_EXT1_INTR	50		18	APP_I2C_EXT1_INTR_MAP_REG
PRO_RSA_INTR_MAP_REG	19		51	RSA_INTR	51		19	APP_RSA_INTR_MAP_REG
PRO_SPI1_DMA_INT_MAP_REG	20		52	SPI1_DMA_INT	52		20	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	
PRO_SPI2_DMA_INT_MAP_REG	21	PRO_INTR_STATUS_REG_1	53	SPI2_DMA_INT	53	APP_INTR_STATUS_REG_1	21	APP_SPI2_DMA_INT_MAP_REG
PRO_SPI3_DMA_INT_MAP_REG	22		54	SPI3_DMA_INT	54		22	APP_SPI3_DMA_INT_MAP_REG
PRO_WDG_INT_MAP_REG	23		55	WDG_INT	55		23	APP_WDG_INT_MAP_REG
PRO_TIMER_INT1_MAP_REG	24		56	TIMER_INT1	56		24	APP_TIMER_INT1_MAP_REG
PRO_TIMER_INT2_MAP_REG	25		57	TIMER_INT2	57		25	APP_TIMER_INT2_MAP_REG
PRO_TG_T0_EDGE_INT_MAP_REG	26		58	TG_T0_EDGE_INT	58		26	APP_TG_T0_EDGE_INT_MAP_REG
PRO_TG_T1_EDGE_INT_MAP_REG	27		59	TG_T1_EDGE_INT	59		27	APP_TG_T1_EDGE_INT_MAP_REG
PRO_TG_WDT_EDGE_INT_MAP_REG	28		60	TG_WDT_EDGE_INT	60		28	APP_TG_WDT_EDGE_INT_MAP_REG
PRO_TG_LACT_EDGE_INT_MAP_REG	29		61	TG_LACT_EDGE_INT	61		29	APP_TG_LACT_EDGE_INT_MAP_REG
PRO_TG1_T0_EDGE_INT_MAP_REG	30		62	TG1_T0_EDGE_INT	62		30	APP_TG1_T0_EDGE_INT_MAP_REG
PRO_TG1_T1_EDGE_INT_MAP_REG	31		63	TG1_T1_EDGE_INT	63		31	APP_TG1_T1_EDGE_INT_MAP_REG
PRO_TG1_WDT_EDGE_INT_MAP_REG	0	PRO_INTR_STATUS_REG_2	64	TG1_WDT_EDGE_INT	64	APP_INTR_STATUS_REG_2	0	APP_TG1_WDT_EDGE_INT_MAP_REG
PRO_TG1_LACT_EDGE_INT_MAP_REG	1		65	TG1_LACT_EDGE_INT	65		1	APP_TG1_LACT_EDGE_INT_MAP_REG
PRO_MMU_IA_INT_MAP_REG	2		66	MMU_IA_INT	66		2	APP_MMU_IA_INT_MAP_REG
PRO_MPU_IA_INT_MAP_REG	3		67	MPU_IA_INT	67		3	APP_MPU_IA_INT_MAP_REG
PRO_CACHE_IA_INT_MAP_REG	4		68	CACHE_IA_INT	68		4	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 7 列出了每个 CPU 所有的中断。

表 7: 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 6 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 6 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 4 展示了整个子系统的结构以及每种复位方式：

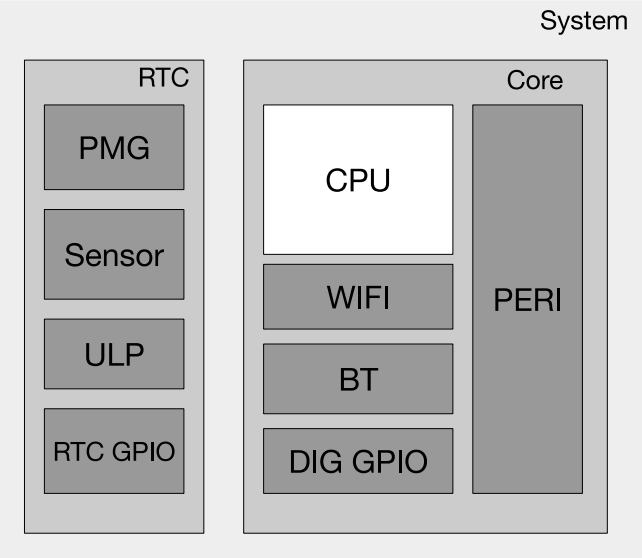


图 4: 系统复位

- 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：APP_CNTL_RESET_CAUSE_PROCPU 来获取复位源。

表 8 列出了从这些寄存器中可能读出的复位源。

表 8: 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 Rest	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 5 为系统时钟结构。

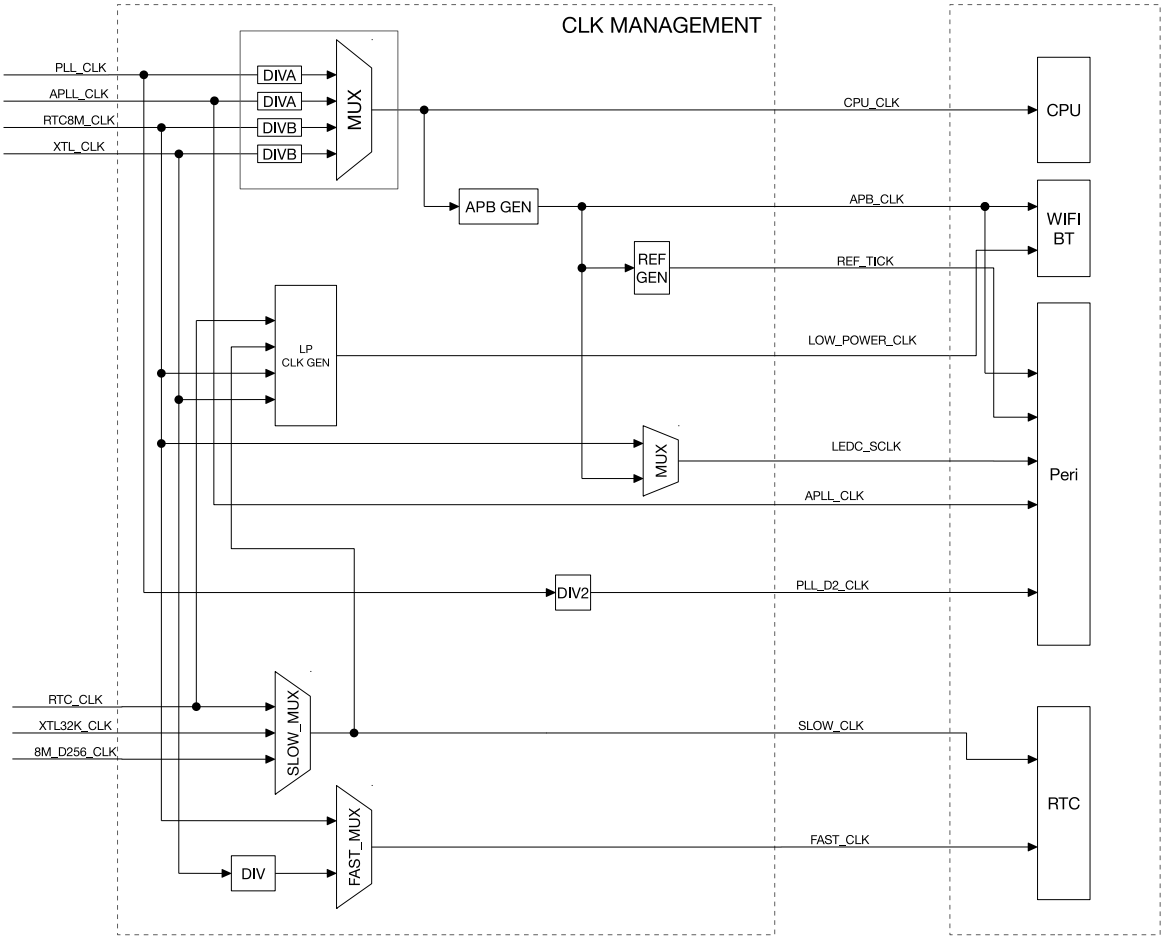


图 5: 系统时钟

3.2.2 时钟源

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

- 快速时钟
  - PLL_CLK，320 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
  - RTC8M_CLK，8 MHz 内部时钟，频率可调
  - RTC8M_D256_CLK 由 RTC8M_CLK 256 分频所得，频率为（RTC8M_CLK 256）。当 RTC8M_CLK 的初始频率为 8 MHz 时，该时钟以 31.250 KHz 的频率运行。
  - RTC_CLK，150 KHz 内部低功耗时钟，频率可调
- 音频时钟
  - APLL_CLK，16 ~ 128 MHz 内部 Audio PLL 时钟

3.2.3 CPU 时钟

如图5所示，CPU_CLK 为 CPU 主时钟，它在高效工作模式下，主频可以达到 160 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，RTC8M_CLK，XTL_CLK 作为 CPU_CLK 的时钟源。具体请参考表 9 和表 10。

表 9: CPU_CLK 源

RTC_CNTL_SOC_CLK_SEL 值	时钟源
0	XTL_CLK
1	PLL_CLK
2	RTC8M_CLK
3	APLL_CLK

表 10: CPU_CLK 源

时钟源	SEL*	CPU 时钟
0 / XTL_CLK	-	CPU_CLK = XTL_CLK / (APB_CTRL_PRE_DIV_CNT+1) APB_CTL_PRE_DIV_CNT 默认值为 0，范围 0 ~ 1023。
1 / PLL_CLK	0	CPU_CLK = PLL_CLK / 4 CPU_CLK 频率为 80 MHz。
1 / PLL_CLK	1	CPU_CLK = PLL_CLK / 2 CPU_CLK 频率为 160 MHz。
2 / RTC8M_CLK	-	CPU_CLK = RTC8M_CLK / (APB_CTRL_PRE_DIV_CNT+1) APB_CTL_PRE_DIV_CNT 默认值为 0，范围 0 ~ 1023。
3 / APLL_CLK	0	CPU_CLK = APLL_CLK / 4。
3 / APLL_CLK	1	CPU_CLK = APLL_CLK / 2。

*SEL: DPORT_CPUPERIOD _SEL 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_D2_CLK。
下表 11 为接入各个外设的时钟。

表 11: 外设时钟用法

外设	APB_CLK	REF_TICK	LEDC_SCLK	APLL_CLK	PLL_D2_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	N
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 源

如表 12 所示，APB_CLK 由 CPU_CLK 产生，分频系数由 CPU_CLK 源决定：

表 12: APB_CLK 源

CPU_CLK 源	APB_CLK
PLL_CLK	CPU_CLK / 2
APLL_CLK	CPU_CLK / 2
XTAL_CLK	CPU_CLK
RTC8M_CLK	CPU_CLK

3.2.4.2 REF_TICK 源

REF_TICK 由 APB_CLK 分频产生，分频值由 APB_CLK 源和 CPU_CLK 源共同决定。用户通过配置合理的分频系数，可以保证 REF_TICK 在 APB_CLK 切换时维持频率不变。寄存器配置如表 13 所示：

表 13: REF_TICK 源

CPU_CLK & APB_CLK 源	时钟分频寄存器
PLL_CLK	APB_CTRL_PLL_TICK_NUM
XTAL_CLK	APB_CTRL_XTAL_TICK_NUM
APLL_CLK	APB_CTRL_APLL_TICK_NUM
RTC8M_CLK	APB_CTRL_CK8M_TICK_NUM

3.2.4.3 LEDC_SCLK 源

LEDC_SCLK 时钟源由寄存器 LEDC_APB_CLK_SEL 决定，如表 14 所示。

表 14: LEDC_SCLK 源

LEDC_APB_CLK_SEL 值	LEDC_SCLK 源
1	RTC8M_CLK
0	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_D2_CLK 源

PLL_D2_CLK 是 PLL_CLK 的二分频时钟。

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 11。

LED PWM 模块能将 RTC8M_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 Power Management 模块），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 RTC8M_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 RTC_CLK、SLOW_CLK、RTC8M_CLK 或 XTL_CLK，用于 Wi-Fi 和 BT 的低功耗模式。

3.2.6 RTC 时钟

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

SLOW_CLK 允许选择 RTC_CLK，XTL32K_CLK 或 RTC8M_D256_CLK，用于驱动 Power Management 模块。

FAST_CLK 允许选择 XTL_CLK 的分频时钟或 RTC8M_CLK，用于驱动 On-chip Sensor 模块。

4. IO_MUX 和 GPIO 交换矩阵

4.1 概述

ESP32 芯片有 40 个物理 GPIO pad，有部分 GPIO pad 不可用或者没有对应的芯片封装上的管脚。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。

此章内容描述了数字 pad（控制信号：FUNC_SEL、IE、OE、WPU、WDU 等）和 256 个外设输入 / 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：SIG_IN_SEL、SIG_OUT_SEL、IE、OE 等）以及 RTC IO_MUX 之间的信号选择和连接关系。

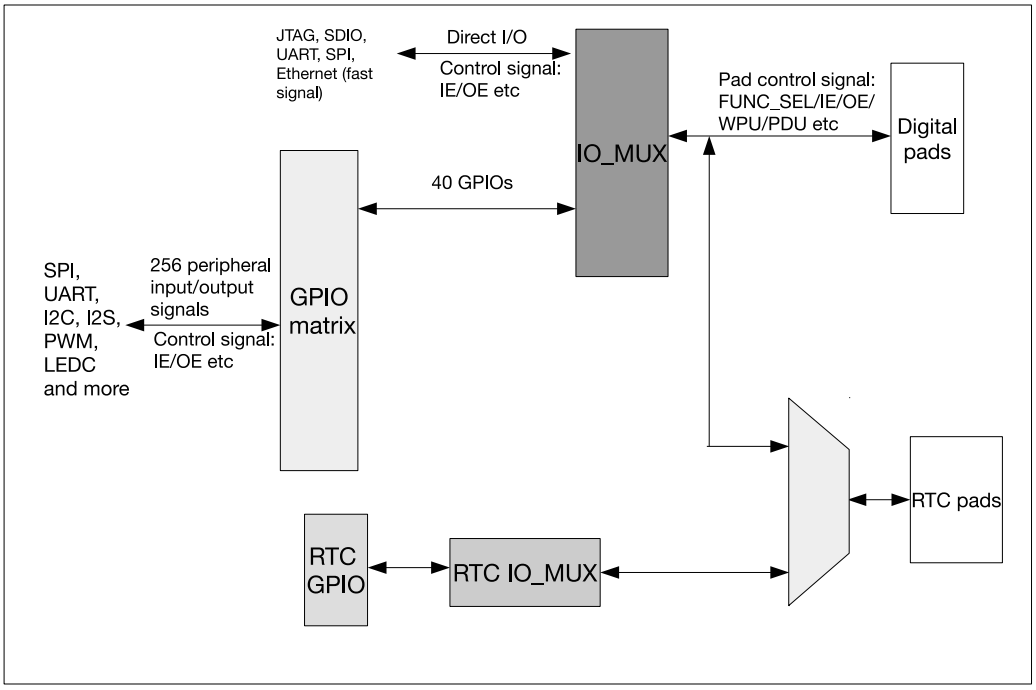


图 6: IO_MUX、RTC IO_MUX 和 GPIO 交换矩阵结构框图

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：256 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 256 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 40 个 GPIO（0-39）中获取外设输入信号的索引号（0-255）。

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 7 为通过 GPIO 交换矩阵的外设输入的示意图。

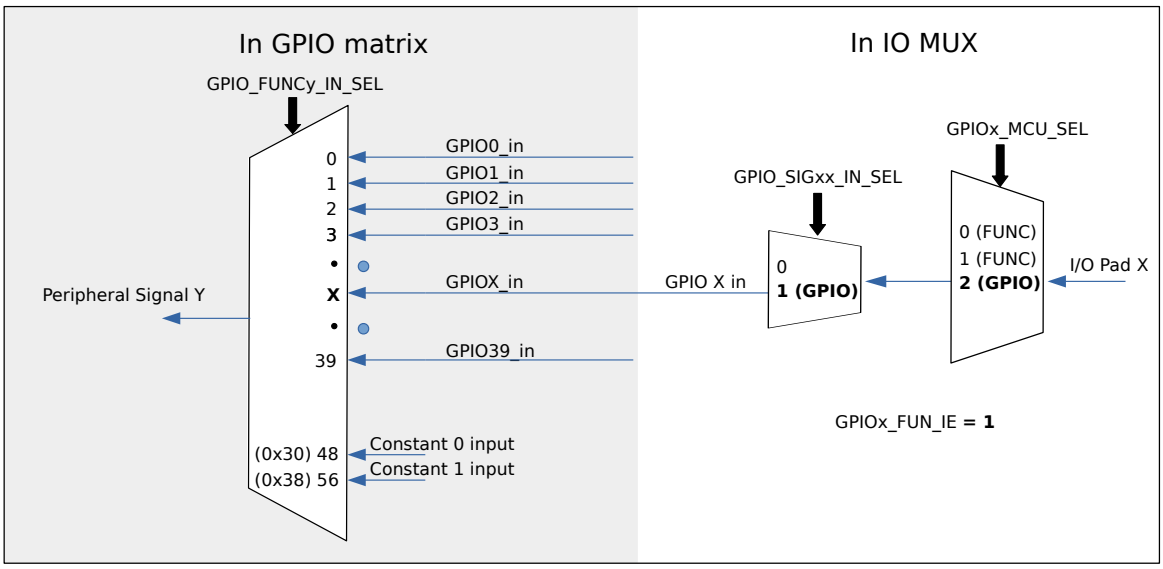


图 7: 通过 IO_MUX、GPIO 交换矩阵的外设输入

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：
   - 设置 GPIO_FUNC*x*_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 寄存器和 GPIO_ENABLE_DATA[*x*] 寄存器：
   - 若为纯单向数据输入，则可以关闭输出：将 GPIO_FUNC*x*_OEN_SEL 位置为 1，并且将 GPIO_ENABLE_DATA[*x*] 寄存器置为 0。否则，不需要关闭输出。
3. 配置 GPIO pad *X* 的 IO_MUX 寄存器：
   - 设置功能字段为 GPIO。
   - 置位 *xx*_FUN_IE 使能输入。
   - 设置 *xx*_FUN_WPU 和 *xx*_FUN_WPD 字段为需要的值，使能内部上拉 / 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*x*_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*x*_IN_SEL 是 0x30 时，input_signal_*x* 始终为 0。
  - 当 GPIO_FUNC*x*_IN_SEL 是 0x38 时，input_signal_*x* 始终为 1。

4.2.3 简单 GPIO 输入

GPIO_IN_DATA 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* 配置 *xx*_FUN_IE 寄存器，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（0-255）的外设信号输出到前 34 个（0-33）GPIO。GPIO pad 34-39 不能用于输出信号。

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。

4.3.2 功能描述

图 8 所示为 256 个输入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。

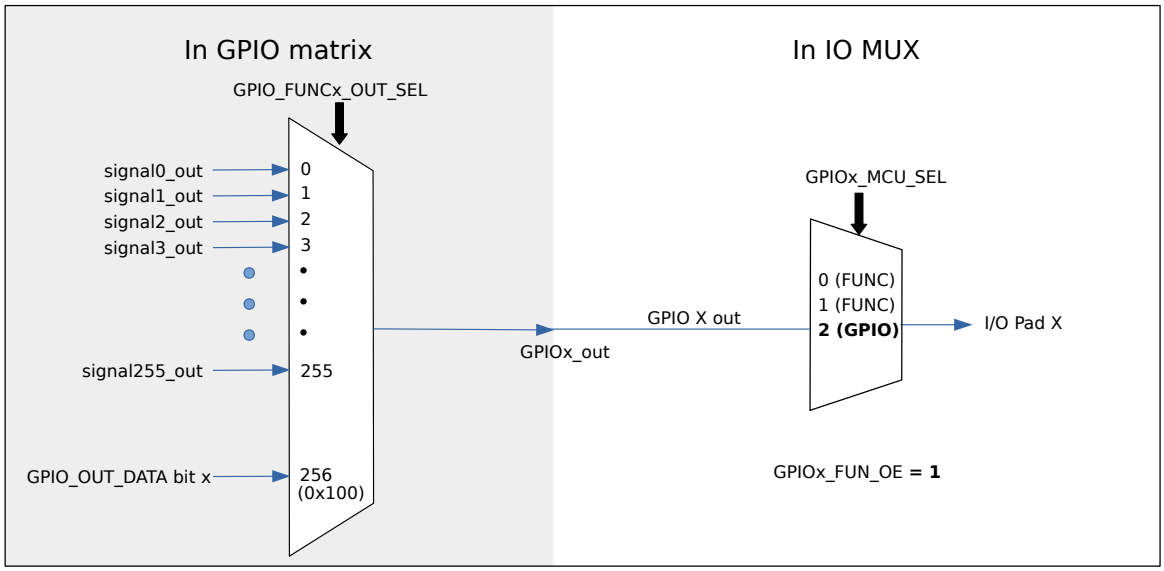


图 8: 通过 GPIO 交换矩阵输出信号

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*] 寄存器：

- 设置 GPIO_FUNC*x*_OUT_SEL 为外设输出信号 *Y* 的索引号。
- 置位 GPIO_FUNC*x*_OEN_SEL 和 GPIO_ENABLE_DATA[*x*]，此设置为强制使能输出模式；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。

2. 可以通过设置 GPIO_PIN*x* 寄存器中的 GPIO_PIN*x*_PAD_DRIVER 位来选择是否以开漏方式输出。
3. 配置 GPIO pad *X* 的 I/O Mux 寄存器：
   - 设置功能字段为 GPIO。
   - 设置 *xx*_FUN_DRV 字段为特定的输出强度值，值越大，输出驱动能力越强。在开漏模式下，通过配置 *xx*_FUNC_WPU 和 *xx*_FUNC_WPD 寄存器上拉 / 下拉 pad。

说明：

- 某一个外设的输出信号可以同时从多个 pad 输出。
- 只有前 34 个 GPIO（0-33）可以用于输出信号。
- 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_FUNC*x*_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，必须置位 SIG_IN_SEL 寄存器，直接将输入信号输出到外设。

4.5 RTC IO_MUX 的低功耗和模拟 I/O 功能

4.5.1 概述

18 个管脚具有低功耗（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 功能描述

每个 pad 的模拟和 RTC 功能是由 RTC_GPIO_PIN*x* 寄存器中的 RTC_IO_TOUCH_PAD*x*_TO_GPIO 位控制的。此位默认置为 1，通过 IO_MUX 子系统输入输出信号，如前文所述。

如果清零 RTC_IO_TOUCH_PAD*x*_TO_GPIO 位，则输入输出信号会经过 RTC 子系统。在这种模式下，RTC_GPIO_PIN*x* 寄存器用于数字 I/O，pad 的模拟功能也可以实现。章节 4.11 列出了 RTC 管脚的功能。

表 4.11 列出了 GPIO pad 与相应的 RTC 管脚和模拟功能的映射关系。请注意 RTC_IO_PIN*x* 寄存器使用的是 RTC GPIO 管脚的序号，不是 GPIO pad 的序号。

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 GPIO*xx*_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 15: IO_MUX Light-sleep 管脚功能寄存器

IO_MUX 功能	正常工作模式 或者 GPIO*xx*_SLP_SEL = 0	Light-sleep 模式 并且 GPIO*xx*_SLP_SEL = 1
Output Drive Strength	GPIO*xx*_FUNC_DRV	GPIO*xx*_MCU_DRV
Pullup Resistor	GPIO*xx*_FUNC_WPU	GPIO*xx*_MCU_WPU
Pulldown Resistor	GPIO*xx*_FUNC_WPD	GPIO*xx*_MCU_WPD
Output Enable	(From GPIO Matrix _OEN field)	GPI*xx*_MCU_OE

如果 GPIO*xx*_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和系统复位时或者 Deep-sleep 时 pad 的状态不被改变，就需要提前把 hold 置上。

4.8 I/O Pad 供电

IO pad 供电如图 9 所示。

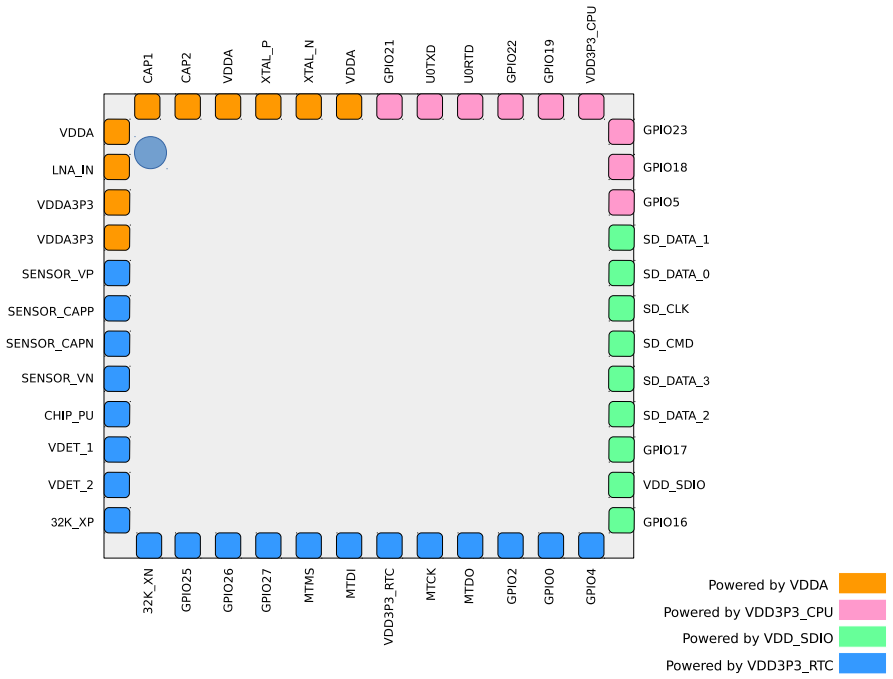


图 9: ESP32 I/O Pad 供电源

- 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。
- 粉红色和绿色的 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 1.8V 或 3.3V（与 VRTC 相同），这取决于 MTDI pad 在复位时的状态——高电平时为 1.8V，低电平时为 3.3V。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 16 列出了 GPIO 交换矩阵的外设输入 / 输出信号。

表 16: 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

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
2	SPID_in	SPID_out	YES
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		rmt_sig_out7	
95	I2CEXT1_SCL_in	I2CEXT1_SCL_out	
96	I2CEXT1_SDA_in	I2CEXT1_SDA_out	

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
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	pwm2_out1h	
115	pwm2_flta	pwm2_out1l	
116	pwm2_fltb	pwm2_out2h	
117	pwm2_cap1_in	pwm2_out2l	
118	pwm2_cap2_in	pwm2_out3h	
119	pwm2_cap3_in	pwm2_out3l	
120	pwm3_flta	pwm2_out4h	
121	pwm3_fltb	pwm2_out4l	
122	pwm3_cap1_in		
123	pwm3_cap2_in		
124	pwm3_cap3_in		
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	
153	I2S0I_DATA_in13	I2S0O_DATA_out13	
154	I2S0I_DATA_in14	I2S0O_DATA_out14	
155	I2S0I_DATA_in15	I2S0O_DATA_out15	

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
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	pwm3_out1h	
191	I2S0I_V_SYNC	pwm3_out1l	
192	I2S0I_H_ENABLE	pwm3_out2h	
193	I2S1I_H_SYNC	pwm3_out2l	
194	I2S1I_V_SYNC	pwm3_out3h	
195	I2S1I_H_ENABLE	pwm3_out3l	
196		pwm3_out4h	
197		pwm3_out4l	
198	U2RXD_in	U2TXD_out	YES
199	U2CTS_in	U2RTS_out	YES

Signal	Input Signal	Output Signal	Direct I/O in IO_MUX
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 17 列出了每个 I/O pad 的 IO_MUX 功能。

表 17: IO_MUX Pad 列表

GPIO	Pad Name	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	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	-
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	1	R
14	MTMS	MTMS	HSPICLK	GPIO14	HS2_CLK	SD_CLK	EMAC_TXD2	1	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	-

GPIO	Pad Name	Function 1	Function 2	Function 3	Function 4	Function 5	Function 6	Reset	Notes
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	-
20	GPIO20	GPIO20	-	GPIO20	-	-	-	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	1	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。

参考 ESP32 管脚清单 获取管脚功能的完整表格。

4.11 RTC_MUX 管脚清单

表 18 列出了 RTC 管脚和对应 GPIO pad。

表 18: RTC_MUX 管脚清单

RTC GPIO Num	GPIO Num	Pad Name	Analog Function		
			1	2	3
0	36	SENSOR_VP	ADC_H	ADC1_CH0	-
1	37	SENSOR_CAPP	ADC_H	ADC1_CH1	-
2	38	SENSOR_CAPN	ADC_H	ADC1_CH2	-
3	39	SENSOR_VN	ADC_H	ADC1_CH3	-
4	34	VDET_1	-	ADC1_CH6	-
5	35	VDET_2	-	ADC1_CH7	-
6	25	GPIO25	DAC_1	ADC2_CH8	-

RTC GPIO Num	GPIO Num	Pad Name	Analog Function		
			1	2	3
7	26	GPIO26	DAC_2	ADC2_CH9	-
8	33	32K_XN	XTAL_32K_N	ADC1_CH5	TOUCH8
9	32	32K_XP	XTAL_32K_P	ADC1_CH4	TOUCH9
10	4	GPIO4	-	ADC2_CH0	TOUCH0
11	0	GPIO0	-	ADC2_CH1	TOUCH1
12	2	GPIO2	-	ADC2_CH2	TOUCH2
13	15	MTDO	-	ADC2_CH3	TOUCH3
14	13	MTCK	-	ADC2_CH4	TOUCH4
15	12	MTDI	-	ADC2_CH5	TOUCH5
16	14	MTMS	-	ADC2_CH6	TOUCH6
17	27	GPIO27	-	ADC2_CH7	TOUCH7

4.12 寄存器列表

名称	描述	地址	访问
GPIO_OUT_REG	GPIO 0-31 output register_REG	0x3FF44004	读/写
GPIO_OUT_W1TS_REG	GPIO 0-31 output register_W1TS_REG	0x3FF44008	只读
GPIO_OUT_W1TC_REG	GPIO 0-31 output register_W1TC_REG	0x3FF4400C	只读
GPIO_OUT1_REG	GPIO 0-31 output register1_REG	0x3FF44010	读/写
GPIO_OUT1_W1TS_REG	GPIO 0-31 output register1_W1TS_REG	0x3FF44014	只读
GPIO_OUT1_W1TC_REG	GPIO 0-31 output register1_W1TC_REG	0x3FF44018	只读
GPIO_OUT_W1TS_REG	GPIO 0-31 output bit set register_REG	0x3FF44008	只读
GPIO_OUT_W1TC_REG	GPIO 0-31 output bit clear register_REG	0x3FF4400C	只读
GPIO_OUT1_REG	GPIO 32-39 output register_REG	0x3FF44010	读/写
GPIO_OUT1_W1TS_REG	GPIO 32-39 output register_W1TS_REG	0x3FF44014	只读
GPIO_OUT1_W1TC_REG	GPIO 32-39 output register_W1TC_REG	0x3FF44018	只读
GPIO_OUT1_W1TS_REG	GPIO 32-39 output bit set register_REG	0x3FF44014	只读
GPIO_OUT1_W1TC_REG	GPIO 32-39 output bit clear register_REG	0x3FF44018	只读
GPIO_ENABLE_REG	GPIO 0-31 output enable register_REG	0x3FF44020	读/写
GPIO_ENABLE_W1TS_REG	GPIO 0-31 output enable register_W1TS_REG	0x3FF44024	只读
GPIO_ENABLE_W1TC_REG	GPIO 0-31 output enable register_W1TC_REG	0x3FF44028	只读
GPIO_ENABLE1_REG	GPIO 0-31 output enable register1_REG	0x3FF4402C	读/写
GPIO_ENABLE1_W1TS_REG	GPIO 0-31 output enable register1_W1TS_REG	0x3FF44030	只读
GPIO_ENABLE1_W1TC_REG	GPIO 0-31 output enable register1_W1TC_REG	0x3FF44034	只读
GPIO_ENABLE_W1TS_REG	GPIO 0-31 output enable bit set register_REG	0x3FF44024	只读
GPIO_ENABLE_W1TC_REG	GPIO 0-31 output enable bit clear register_REG	0x3FF44028	只读
GPIO_ENABLE1_REG	GPIO 32-39 output enable register_REG	0x3FF4402C	读/写
GPIO_ENABLE1_W1TS_REG	GPIO 32-39 output enable register_W1TS_REG	0x3FF44030	只读
GPIO_ENABLE1_W1TC_REG	GPIO 32-39 output enable register_W1TC_REG	0x3FF44034	只读
GPIO_ENABLE1_W1TS_REG	GPIO 32-39 output enable bit set register_REG	0x3FF44030	只读
GPIO_ENABLE1_W1TC_REG	GPIO 32-39 output enable bit clear register_REG	0x3FF44034	只读
GPIO_STRAP_REG	Bootstrap pin value register_REG	0x3FF44038	只读

名称	描述	地址	访问
GPIO_IN_REG	GPIO 0-31 input register_REG	0x3FF4403C	只读
GPIO_IN1_REG	GPIO 0-31 input register1_REG	0x3FF44040	只读
GPIO_IN1_REG	GPIO 32-39 input register_REG	0x3FF44040	只读
GPIO_STATUS_REG	GPIO 0-31 interrupt status register_REG	0x3FF44044	读/写
GPIO_STATUS_W1TS_REG	GPIO 0-31 interrupt status register_W1TS_REG	0x3FF44048	只读
GPIO_STATUS_W1TC_REG	GPIO 0-31 interrupt status register_W1TC_REG	0x3FF4404C	只读
GPIO_STATUS1_REG	GPIO 0-31 interrupt status register1_REG	0x3FF44050	读/写
GPIO_STATUS1_W1TS_REG	GPIO 0-31 interrupt status register1_W1TS_REG	0x3FF44054	只读
GPIO_STATUS1_W1TC_REG	GPIO 0-31 interrupt status register1_W1TC_REG	0x3FF44058	只读
GPIO_STATUS_W1TS_REG	GPIO 0-31 interrupt status bit set register_REG	0x3FF44048	只读
GPIO_STATUS_W1TC_REG	GPIO 0-31 interrupt status bit clear register_REG	0x3FF4404C	只读
GPIO_STATUS1_REG	GPIO 32-39 interrupt status register_REG	0x3FF44050	读/写
GPIO_STATUS1_W1TS_REG	GPIO 32-39 interrupt status register_W1TS_REG	0x3FF44054	只读
GPIO_STATUS1_W1TC_REG	GPIO 32-39 interrupt status register_W1TC_REG	0x3FF44058	只读
GPIO_STATUS1_W1TS_REG	GPIO 32-39 interrupt status bit set register_REG	0x3FF44054	只读
GPIO_STATUS1_W1TC_REG	GPIO 32-39 interrupt status bit clear register_REG	0x3FF44058	只读
GPIO_ACPU_INT_REG	GPIO 0-31 APP_CPU interrupt status_REG	0x3FF44060	只读
GPIO_ACPU_INT1_REG	GPIO 0-31 APP_CPU interrupt status1_REG	0x3FF44074	只读
GPIO_ACPU_NMI_INT_REG	GPIO 0-31 APP_CPU non-maskable interrupt status_REG	0x3FF44064	只读
GPIO_ACPU_NMI_INT1_REG	GPIO 0-31 APP_CPU non-maskable interrupt status1_REG	0x3FF44078	只读
GPIO_PCPU_INT_REG	GPIO 0-31 PRO_CPU interrupt status_REG	0x3FF44068	只读
GPIO_PCPU_INT1_REG	GPIO 0-31 PRO_CPU interrupt status1_REG	0x3FF4407C	只读
GPIO_PCPU_NMI_INT_REG	GPIO 0-31 PRO_CPU non-maskable interrupt status_REG	0x3FF4406C	只读
GPIO_PCPU_NMI_INT1_REG	GPIO 0-31 PRO_CPU non-maskable interrupt status1_REG	0x3FF44080	只读
GPIO_ACPU_INT1_REG	GPIO 32-39 APP_CPU interrupt status_REG	0x3FF44074	只读
GPIO_ACPU_NMI_INT1_REG	GPIO 32-39 APP_CPU non-maskable interrupt status_REG	0x3FF44078	只读
GPIO_PCPU_INT1_REG	GPIO 32-39 PRO_CPU interrupt status_REG	0x3FF4407C	只读
GPIO_PCPU_NMI_INT1_REG	GPIO 32-39 PRO_CPU non-maskable interrupt status_REG	0x3FF44080	只读
GPIO_PIN0_REG	Configuration for GPIO pin 0_REG	0x3FF44088	读/写
GPIO_PIN1_REG	Configuration for GPIO pin 1_REG	0x3FF4408C	读/写
GPIO_PIN2_REG	Configuration for GPIO pin 2_REG	0x3FF44090	读/写
...	...		
GPIO_PIN38_REG	Configuration for GPIO pin 38_REG	0x3FF44120	读/写
GPIO_PIN39_REG	Configuration for GPIO pin 39_REG	0x3FF44124	读/写
GPIO_FUNC0_IN_SEL_CFG_REG	Peripheral function 0 input selection register_REG	0x3FF44130	读/写
GPIO_FUNC1_IN_SEL_CFG_REG	Peripheral function 1 input selection register_REG	0x3FF44134	读/写
...	...		

名称	描述	地址	访问
GPIO_FUNC254_IN_SEL_CFG_REG	Peripheral function 254 input selection register_REG	0x3FF44528	读/写
GPIO_FUNC255_IN_SEL_CFG_REG	Peripheral function 255 input selection register_REG	0x3FF4452C	读/写
GPIO_FUNC0_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 0_REG	0x3FF44530	读/写
GPIO_FUNC1_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 1_REG	0x3FF44534	读/写
...	...		
GPIO_FUNC38_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 38_REG	0x3FF445C8	读/写
GPIO_FUNC39_OUT_SEL_CFG_REG	Peripheral output selection for GPIO 39_REG	0x3FF445CC	读/写

名称	描述	地址	访问
IO_MUX_GPIO36_REG	Configuration register for pad GPIO36	0x3FF53004	读/写
IO_MUX_GPIO37_REG	Configuration register for pad GPIO37	0x3FF53008	读/写
IO_MUX_GPIO38_REG	Configuration register for pad GPIO38	0x3FF5300C	读/写
IO_MUX_GPIO39_REG	Configuration register for pad GPIO39	0x3FF53010	读/写
IO_MUX_GPIO34_REG	Configuration register for pad GPIO34	0x3FF53014	读/写
IO_MUX_GPIO35_REG	Configuration register for pad GPIO35	0x3FF53018	读/写
IO_MUX_GPIO32_REG	Configuration register for pad GPIO32	0x3FF5301C	读/写
IO_MUX_GPIO33_REG	Configuration register for pad GPIO33	0x3FF53020	读/写
IO_MUX_GPIO25_REG	Configuration register for pad GPIO25	0x3FF53024	读/写
IO_MUX_GPIO26_REG	Configuration register for pad GPIO26	0x3FF53028	读/写
IO_MUX_GPIO27_REG	Configuration register for pad GPIO27	0x3FF5302C	读/写
IO_MUX_MTMS_REG	Configuration register for pad MTMS	0x3FF53030	读/写
IO_MUX_MTDI_REG	Configuration register for pad MTDI	0x3FF53034	读/写
IO_MUX_MTCK_REG	Configuration register for pad MTCK	0x3FF53038	读/写
IO_MUX_MTDO_REG	Configuration register for pad MTDO	0x3FF5303C	读/写
IO_MUX_GPIO2_REG	Configuration register for pad GPIO2	0x3FF53040	读/写
IO_MUX_GPIO0_REG	Configuration register for pad GPIO0	0x3FF53044	读/写
IO_MUX_GPIO4_REG	Configuration register for pad GPIO4	0x3FF53048	读/写
IO_MUX_GPIO16_REG	Configuration register for pad GPIO16	0x3FF5304C	读/写
IO_MUX_GPIO17_REG	Configuration register for pad GPIO17	0x3FF53050	读/写
IO_MUX_SD_DATA2_REG	Configuration register for pad SD_DATA2	0x3FF53054	读/写
IO_MUX_SD_DATA3_REG	Configuration register for pad SD_DATA3	0x3FF53058	读/写
IO_MUX_SD_CMD_REG	Configuration register for pad SD_CMD	0x3FF5305C	读/写
IO_MUX_SD_CLK_REG	Configuration register for pad SD_CLK	0x3FF53060	读/写
IO_MUX_SD_DATA0_REG	Configuration register for pad SD_DATA0	0x3FF53064	读/写
IO_MUX_SD_DATA1_REG	Configuration register for pad SD_DATA1	0x3FF53068	读/写
IO_MUX_GPIO5_REG	Configuration register for pad GPIO5	0x3FF5306C	读/写
IO_MUX_GPIO18_REG	Configuration register for pad GPIO18	0x3FF53070	R/W
IO_MUX_GPIO19_REG	Configuration register for pad GPIO19	0x3FF53074	读/写
IO_MUX_GPIO20_REG	Configuration register for pad GPIO20	0x3FF53078	读/写
IO_MUX_GPIO21_REG	Configuration register for pad GPIO21	0x3FF5307C	读/写
IO_MUX_GPIO22_REG	Configuration register for pad GPIO22	0x3FF53080	读/写
IO_MUX_U0RXD_REG	Configuration register for pad U0RXD	0x3FF53084	读/写

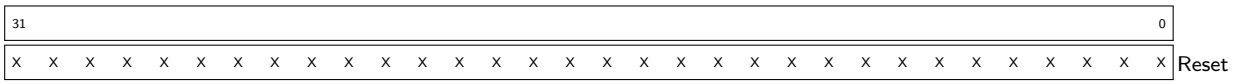
名称	描述	地址	访问
IO_MUX_U0TXD_REG	Configuration register for pad U0TXD	0x3FF53088	读/写
IO_MUX_GPIO23_REG	Configuration register for pad GPIO23	0x3FF5308C	读/写
IO_MUX_GPIO24_REG	Configuration register for pad GPIO24	0x3FF53090	读/写

名称	描述	地址	访问
GPIO 配置 / 数据寄存器			
RTCIO_RTC_GPIO_OUT_REG	RTC GPIO output register_REG	0x3FF48000	读/写
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO output register_W1TS_REG	0x3FF48001	只写
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO output register_W1TC_REG	0x3FF48002	只写
RTCIO_RTC_GPIO_OUT_W1TS_REG	RTC GPIO output bit set register_REG	0x3FF48001	只写
RTCIO_RTC_GPIO_OUT_W1TC_REG	RTC GPIO output bit clear register_REG	0x3FF48002	只写
RTCIO_RTC_GPIO_ENABLE_REG	RTC GPIO output enable register_REG	0x3FF48003	读/写
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable register_W1TS_REG	0x3FF48004	只写
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable register_W1TC_REG	0x3FF48005	只写
RTCIO_RTC_GPIO_ENABLE_W1TS_REG	RTC GPIO output enable bit setregister_REG	0x3FF48004	只写
RTCIO_RTC_GPIO_ENABLE_W1TC_REG	RTC GPIO output enable bit clear register_REG	0x3FF48005	只写
RTCIO_RTC_GPIO_STATUS_REG	RTC GPIO interrupt status register_REG	0x3FF48006	读/写
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status register_W1TS_REG	0x3FF48007	只写
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status register_W1TC_REG	0x3FF48008	只写
RTCIO_RTC_GPIO_STATUS_W1TS_REG	RTC GPIO interrupt status bit set register_REG	0x3FF48007	只写
RTCIO_RTC_GPIO_STATUS_W1TC_REG	RTC GPIO interrupt status bit clear register_REG	0x3FF48008	只写
RTCIO_RTC_GPIO_IN_REG	RTC GPIO input register_REG	0x3FF48009	只读
RTCIO_RTC_GPIO_PIN0_REG	RTC configuration for pin 0_REG	0x3FF4800A	读/写
RTCIO_RTC_GPIO_PIN1_REG	RTC configuration for pin 1_REG	0x3FF4800B	读/写
RTCIO_RTC_GPIO_PIN2_REG	RTC configuration for pin 2_REG	0x3FF4800C	读/写
RTCIO_RTC_GPIO_PIN3_REG	RTC configuration for pin 3_REG	0x3FF4800D	读/写
RTCIO_RTC_GPIO_PIN4_REG	RTC configuration for pin 4_REG	0x3FF4800E	读/写
RTCIO_RTC_GPIO_PIN5_REG	RTC configuration for pin 5_REG	0x3FF4800F	读/写
RTCIO_RTC_GPIO_PIN6_REG	RTC configuration for pin 6_REG	0x3FF48010	读/写
RTCIO_RTC_GPIO_PIN7_REG	RTC configuration for pin 7_REG	0x3FF48011	读/写
RTCIO_RTC_GPIO_PIN8_REG	RTC configuration for pin 8_REG	0x3FF48012	读/写
RTCIO_RTC_GPIO_PIN9_REG	RTC configuration for pin 9_REG	0x3FF48013	读/写
RTCIO_RTC_GPIO_PIN10_REG	RTC configuration for pin 10_REG	0x3FF48014	读/写
RTCIO_RTC_GPIO_PIN11_REG	RTC configuration for pin 11_REG	0x3FF48015	读/写
RTCIO_RTC_GPIO_PIN12_REG	RTC configuration for pin 12_REG	0x3FF48016	读/写
RTCIO_RTC_GPIO_PIN13_REG	RTC configuration for pin 13_REG	0x3FF48017	读/写
RTCIO_RTC_GPIO_PIN14_REG	RTC configuration for pin 14_REG	0x3FF48018	读/写
RTCIO_RTC_GPIO_PIN15_REG	RTC configuration for pin 15_REG	0x3FF48019	读/写
RTCIO_RTC_GPIO_PIN16_REG	RTC configuration for pin 16_REG	0x3FF4801A	读/写
RTCIO_RTC_GPIO_PIN17_REG	RTC configuration for pin 17_REG	0x3FF4801B	读/写
RTCIO_DIG_PAD_HOLD_REG	RTC GPIO hold register_REG	0x3FF4801D	读/写

名称	描述	地址	访问
GPIO RTC 功能配置寄存器			
RTCIO_HALL_SENS_REG	Hall sensor configuration_REG	0x3FF4801E	读/写
RTCIO_SENSOR_PADS_REG	Sensor pads configuration register_REG	0x3FF4801F	读/写
RTCIO_ADC_PAD_REG	ADC configuration register_REG	0x3FF48020	读/写
RTCIO_PAD_DAC1_REG	DAC1 configuration register_REG	0x3FF48021	读/写
RTCIO_PAD_DAC2_REG	DAC2 configuration register_REG	0x3FF48022	读/写
RTCIO_XTAL_32K_PAD_REG	32KHz crystal pads configuration register_REG	0x3FF48023	读/写
RTCIO_TOUCH_CFG_REG	Touch sensor configuration register_REG	0x3FF48024	读/写
RTCIO_EXT_WAKEUP0_REG	External wake up configuration register_REG	0x3FF4802F	读/写
RTCIO_XTL_EXT_CTR_REG	Crystal power down enable gpio source_REG	0x3FF48030	读/写
RTCIO_SAR_I2C_IO_REG	RTC I2C pad selection_REG	0x3FF48031	读/写

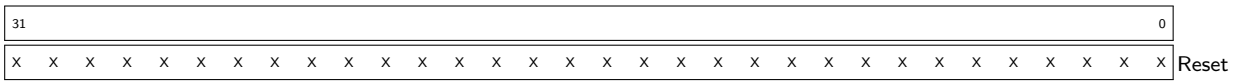
4.13 寄存器

Register 4.1: GPIO_OUT_REG (0x0004)



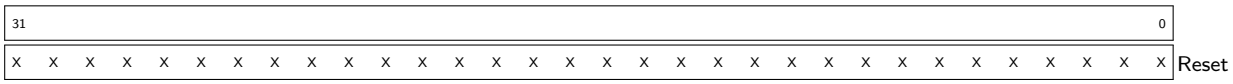
GPIO_OUT_REG GPIO0-31 输出值。(读 / 写)

Register 4.2: GPIO_OUT_W1TS_REG (0x0008)



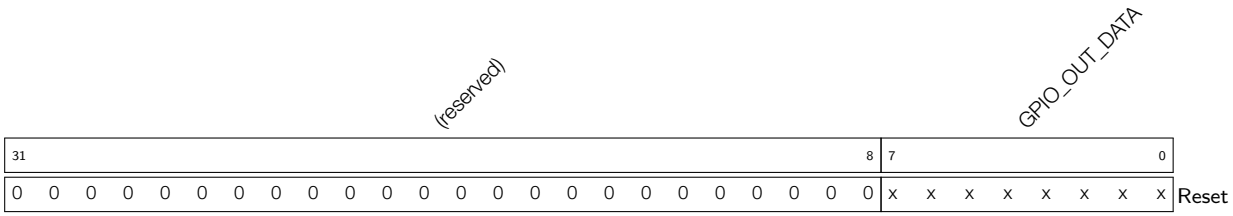
GPIO_OUT_W1TS_REG GPIO0-31 输出置位寄存器。每一位置 1，则 GPIO_OUT_DATA 中的相应位也会置 1。(只读)

Register 4.3: GPIO_OUT_W1TC_REG (0x000c)



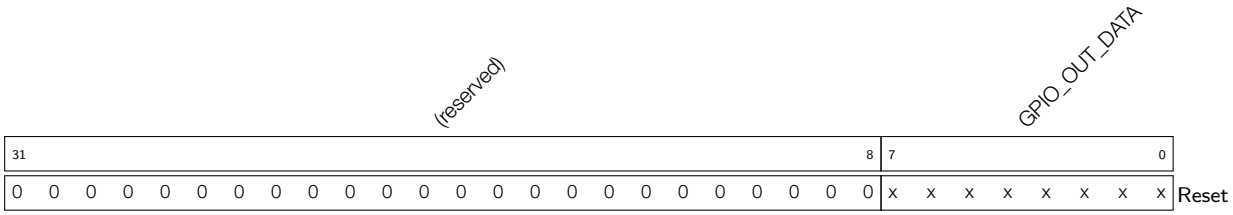
GPIO_OUT_W1TC_REG GPIO0-31 输出清零寄存器。每一位置 1，则 GPIO_OUT_DATA 中的相应位会清零。(只读)

Register 4.4: GPIO_OUT1_REG (0x0010)



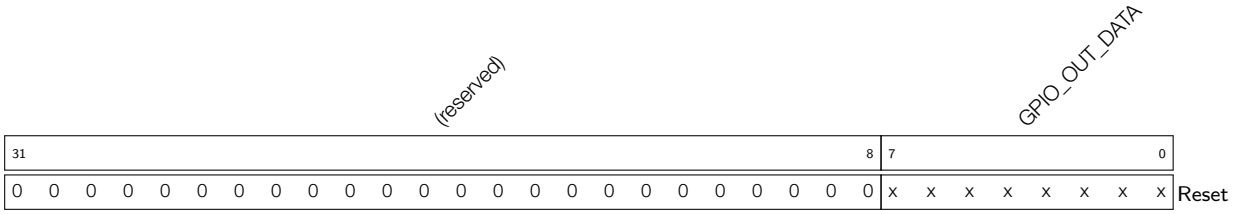
GPIO_OUT_DATA GPIO32-39 输出值。(只读)

Register 4.5: GPIO_OUT1_W1TS_REG (0x0014)



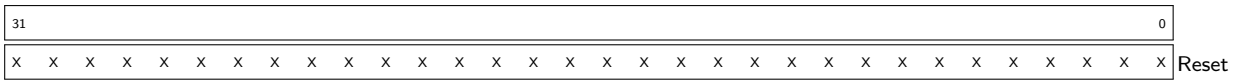
GPIO_OUT_DATA GPIO32-39 输出值置位寄存器。每一位置 1，则 GPIO_OUT1_DATA 中的相应位也会置 1。(只读)

Register 4.6: GPIO_OUT1_W1TC_REG (0x0018)



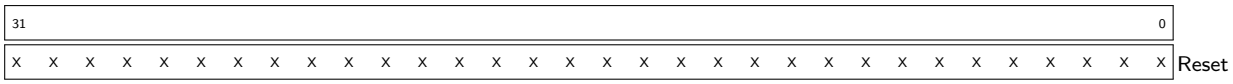
GPIO_OUT_DATA GPIO32-39 输出值清零寄存器。每一位置 1，则 GPIO_OUT1_DATA 中的相应位会清零。(只读)

Register 4.7: GPIO_ENABLE_REG (0x0020)



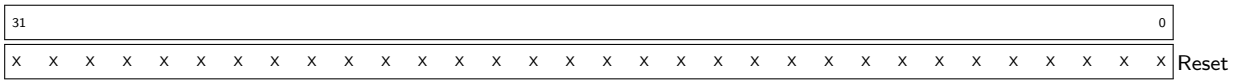
GPIO_ENABLE_REG GPIO0-31 输出使能。(读 / 写)

Register 4.8: GPIO_ENABLE_W1TS_REG (0x0024)



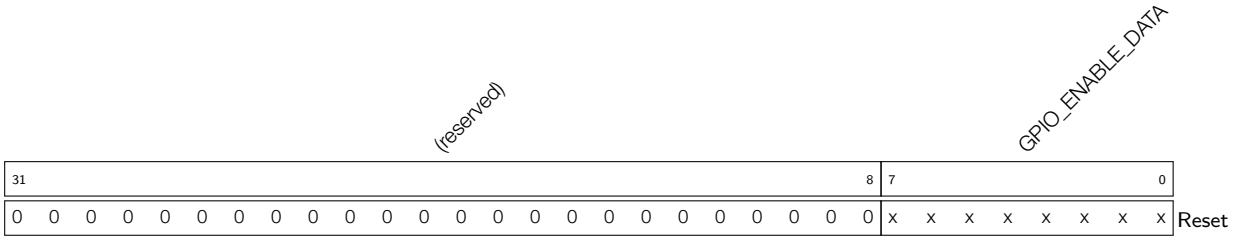
GPIO_ENABLE_W1TS_REG GPIO0-31 输出使能置位寄存器。每一位置 1，则 GPIO_ENABLE 中的相应位也置 1。(只读)

Register 4.9: GPIO_ENABLE_W1TC_REG (0x0028)



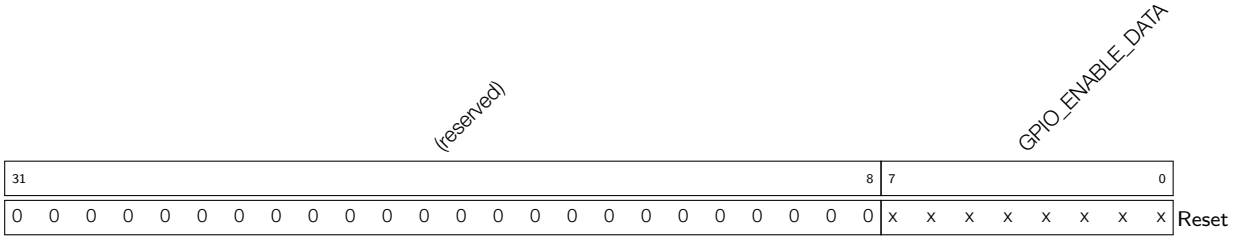
GPIO_ENABLE_W1TC_REG GPIO0-31 输出使能清零寄存器。每一位置 1，则 GPIO_ENABLE 中的相应位会清零。(只读)

Register 4.10: GPIO_ENABLE1_REG (0x002c)



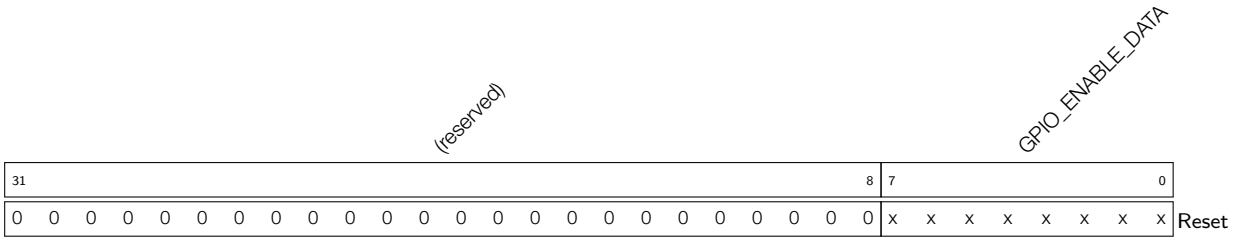
GPIO_ENABLE_DATA GPIO32-39 输出使能。(读 / 写)

Register 4.11: GPIO_ENABLE1_W1TS_REG (0x0030)



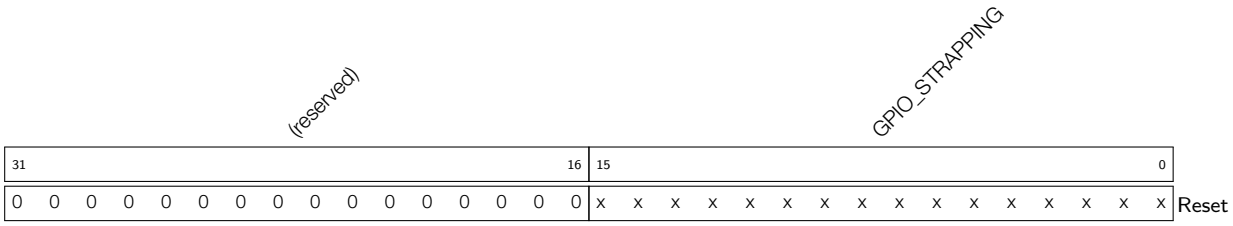
GPIO_ENABLE_DATA GPIO32-39 输出使能置位寄存器。每一位置 1，则 GPIO_ENABLE1 中的相应位也置 1。(只读)

Register 4.12: GPIO_ENABLE1_W1TC_REG (0x0034)



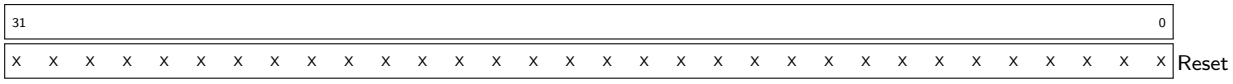
GPIO_ENABLE_DATA GPIO32-39 输出使能清零寄存器。每一位置 1，则 GPIO_ENABLE1 中的相应位会清零。(只读)

Register 4.13: GPIO_STRAP_REG (0x0038)



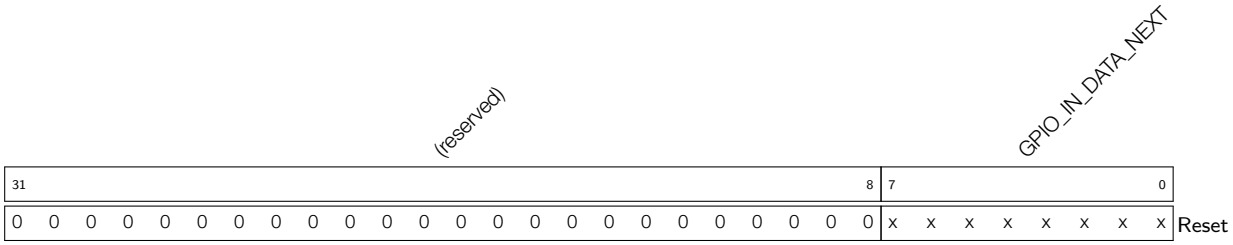
GPIO_STRAPPING GPIO strapping 结果：boot_sel_chip[5:0]: MTDI, GPIO0, GPIO2, GPIO4, MTDO, GPIO5

Register 4.14: GPIO_IN_REG (0x003c)



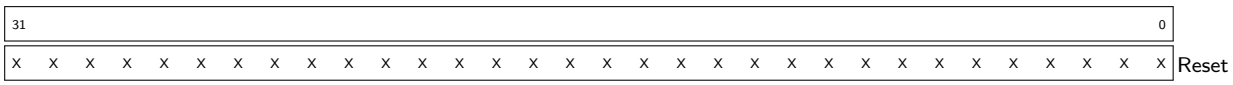
GPIO_IN_REG GPIO0-31 输入值。每个 bit 代表 pad 的片外输入值，比如片外引脚为高电平，此 bit 值应为 1，片外引脚为低电平，此 bit 值应为 0。（只读）

Register 4.15: GPIO_IN1_REG (0x0040)



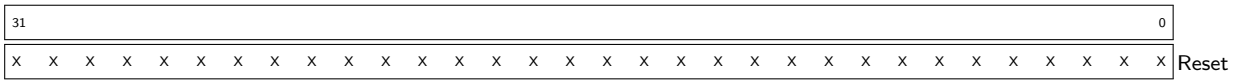
GPIO_IN_DATA_NEXT GPIO32-39 输入值。每个 bit 代表 pad 的片外输入值。（只读）

Register 4.16: GPIO_STATUS_REG (0x0044)



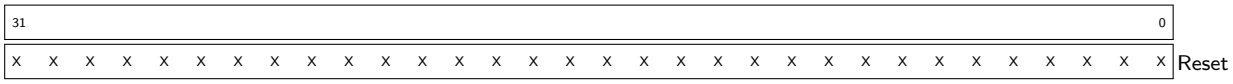
GPIO_STATUS_REG GPIO0-31 中断状态寄存器。每个 bit 都可以作为两个 CPU 的两种中断源，同时应该把 GPIO_PIN*n*_REG 的 0-4 bit 相应的 GPIO_STATUS_INTERRUPT 的使能位置为 1。（读 / 写）

Register 4.17: GPIO_STATUS_W1TS_REG (0x0048)



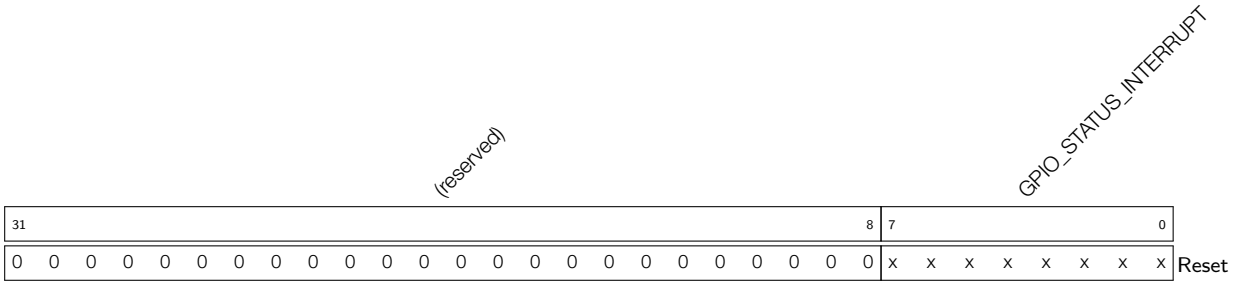
GPIO_STATUS_W1TS_REG GPIO0-31 中断状态置位寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT 中的相应位也置 1。(只读)

Register 4.18: GPIO_STATUS_W1TC_REG (0x004c)



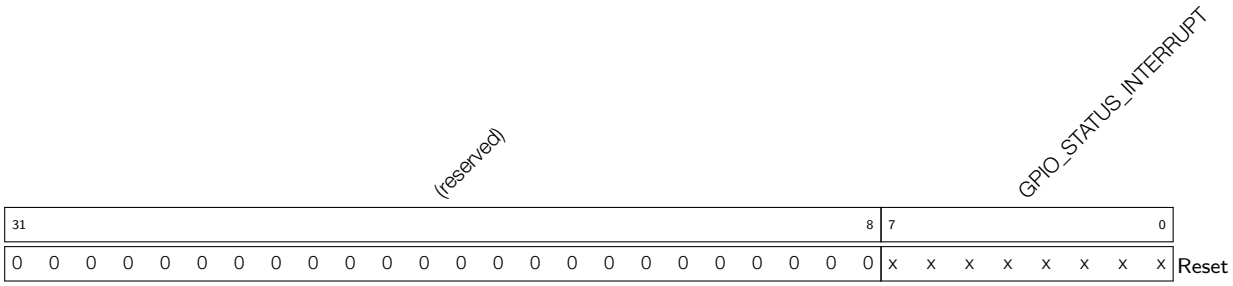
GPIO_STATUS_W1TC_REG GPIO0-31 中断状态清除寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT 中的相应位会清零。(只读)

Register 4.19: GPIO_STATUS1_REG (0x0050)



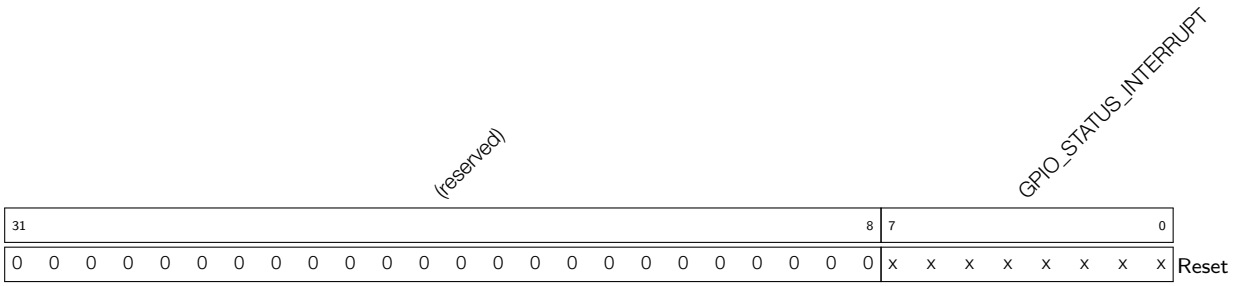
GPIO_STATUS_INTERRUPT GPIO32-39 中断状态。(读 / 写)

Register 4.20: GPIO_STATUS1_W1TS_REG (0x0054)



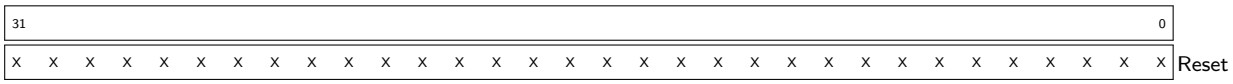
GPIO_STATUS_INTERRUPT GPIO32-39 中断状态置位寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT1 中的相应位也置 1。(只读)

Register 4.21: GPIO_STATUS1_W1TC_REG (0x0058)



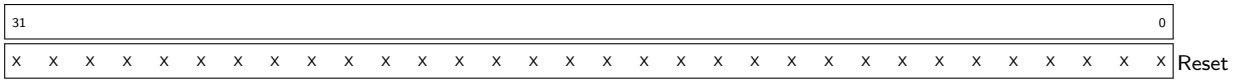
GPIO_STATUS_INTERRUPT GPIO32-39 中断状态清除寄存器。每一位置 1，则 GPIO_STATUS_INTERRUPT1 中的相应位会清零。(只读)

Register 4.22: GPIO_ACPU_INT_REG (0x0060)



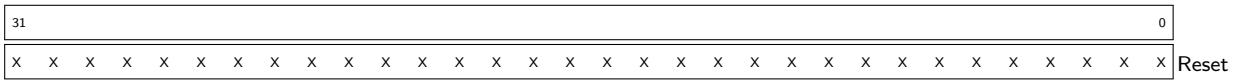
GPIO_ACPU_INT_REG GPIO0-31 APP CPU 中断状态寄存器。(只读)

Register 4.23: GPIO_ACPU_NMI_INT_REG (0x0064)



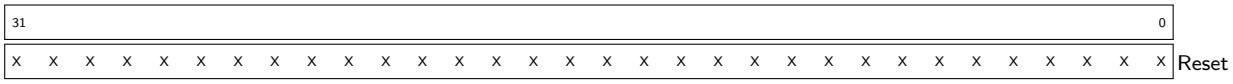
GPIO_ACPU_NMI_INT_REG GPIO0-31 APP CPU 非屏蔽中断状态寄存器。(只读)

Register 4.24: GPIO_PCPU_INT_REG (0x0068)



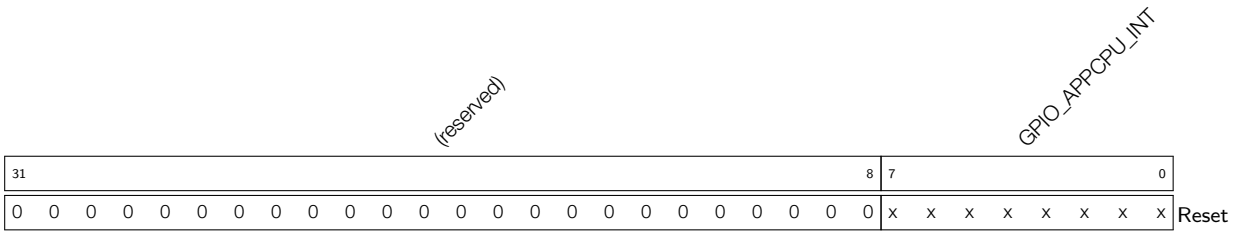
GPIO_PCPU_INT_REG GPIO0-31 PRO CPU 中断状态寄存器。(只读)

Register 4.25: GPIO_PCPU_NMI_INT_REG (0x006c)



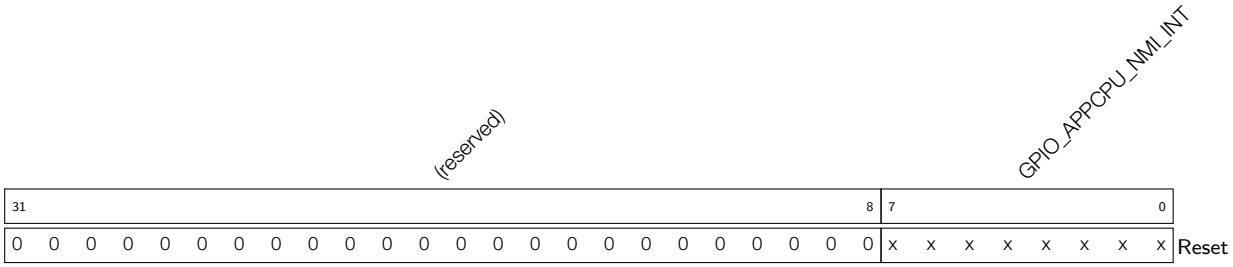
GPIO_PCPU_NMI_INT_REG GPIO0-31 PRO CPU 非屏蔽中断状态寄存器。(只读)

Register 4.26: GPIO_ACPU_INT1_REG (0x0074)



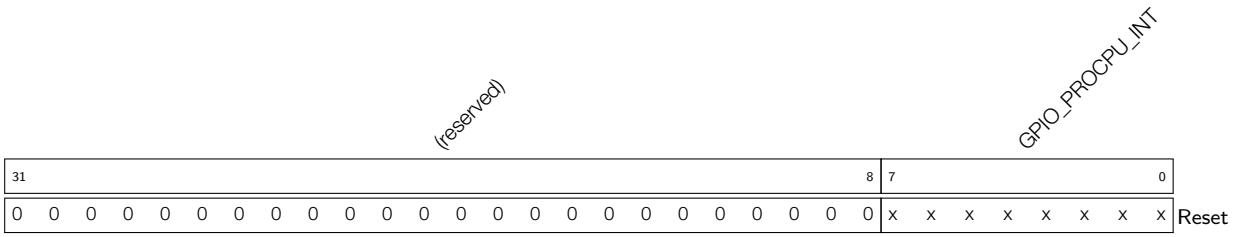
GPIO_APPCPU_INT GPIO32-39 APP CPU 中断状态寄存器。(只读)

Register 4.27: GPIO_ACPU_NMI_INT1_REG (0x0078)



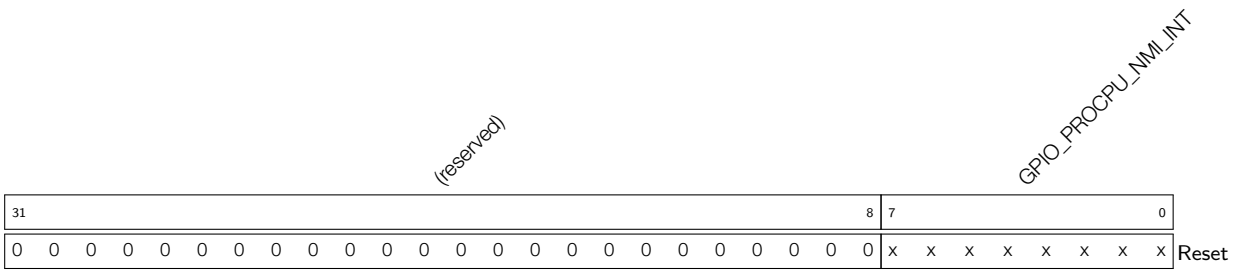
GPIO_APPCPU_NMI_INT GPIO32-39 APP CPU 非屏蔽中断状态寄存器。(只读)

Register 4.28: GPIO_PCPU_INT1_REG (0x007c)



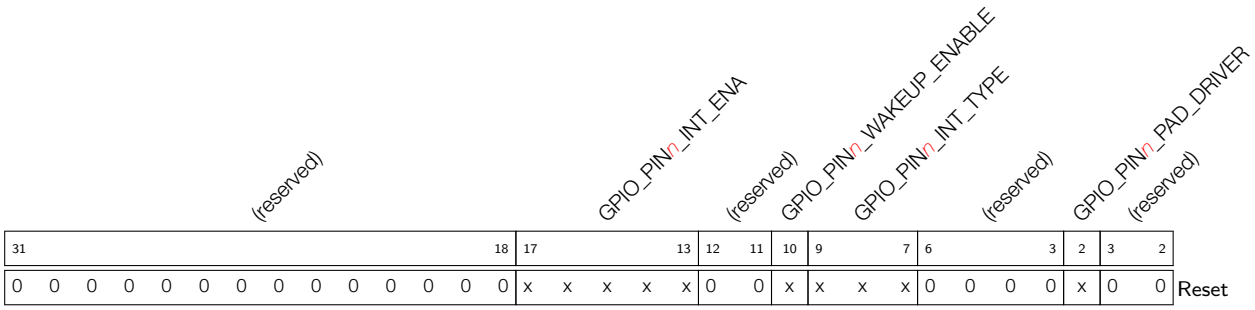
GPIO_PROCPU_INT GPIO32-39 PRO CPU 中断状态寄存器。(只读)

Register 4.29: GPIO_PCPU_NMI_INT1_REG (0x0080)



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+0x4n*)**



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

bit1: APP CPU 非屏蔽中断使能；

bit3: PRO CPU 中断使能；

bit4: 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；

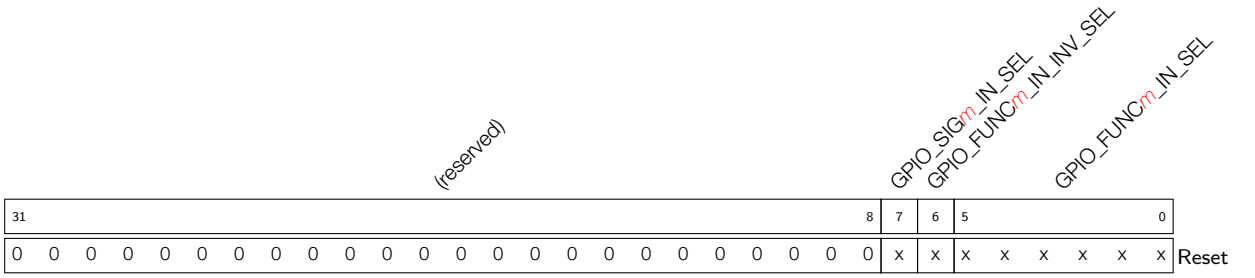
3：任一沿触发；

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*m*_IN_SEL_CFG_REG (*m*: 0-255) (0x130+0x4m*)**

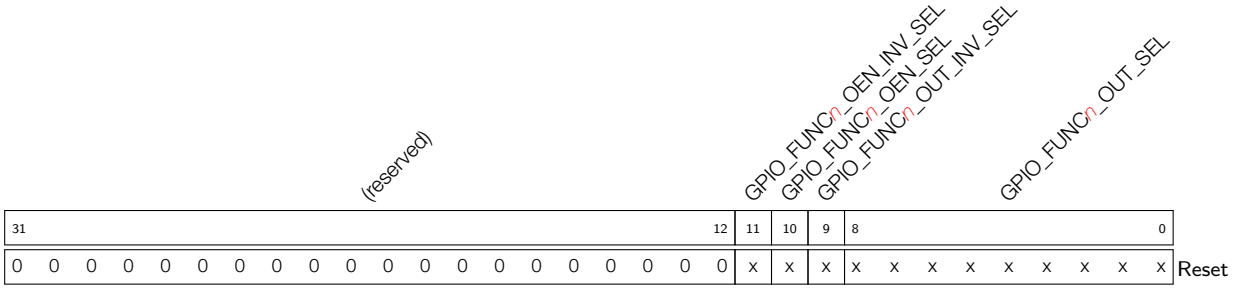


GPIO_SIG*m*_IN_SEL 旁路 GPIO 交换矩阵。0：通过 GPIO 交换矩阵；1：直接通过 IO_MUX 连接信号与外设。（读 / 写）

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

GPIO_FUNC*m*_IN_SEL 外设输入 *m* 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*: 0-39) (0x530+0x4n*)**



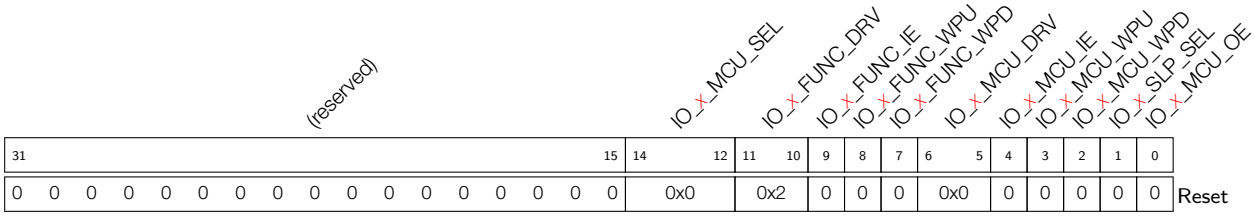
GPIO_FUNC*n*_OEN_INV_SEL 1: 反转输出使能信号；0: 不反转输出使能信号。(读 / 写)

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_DATA_REG 和 GPIO_ENABLE_REG 的 bit *n* 做为输出值和输出使能。(读 / 写)

Register 4.33: IO_MUX_*x*_REG (*x*: GPIO0-GPIO39) (0x10+4x*)**



IO_*x*_MCU_SEL 为信号选择 IO_MUX 功能。0: 选择功能 1；1: 选择功能 2；以此类推。(读 / 写)

IO_*x*_FUNC_DRV 选择驱动强度，值越大，强大越大。(读 / 写)

IO_*x*_FUNC_IE pad 的输入使能。1：输入使能；0：输入关闭。(读 / 写)

IO_*x*_FUNC_WPU pad 的上拉使能。1：内部上拉使能；0：内部上拉关闭。(读 / 写)

IO_*x*_FUNC_WPD pad 的下拉使能。1：内部下拉使能；0：内部下拉关闭。(读 / 写)

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

IO_*x*_MCU_IE 睡眠模式下 pad 的输入使能。1：输入使能；0：输入关闭。(读 / 写)

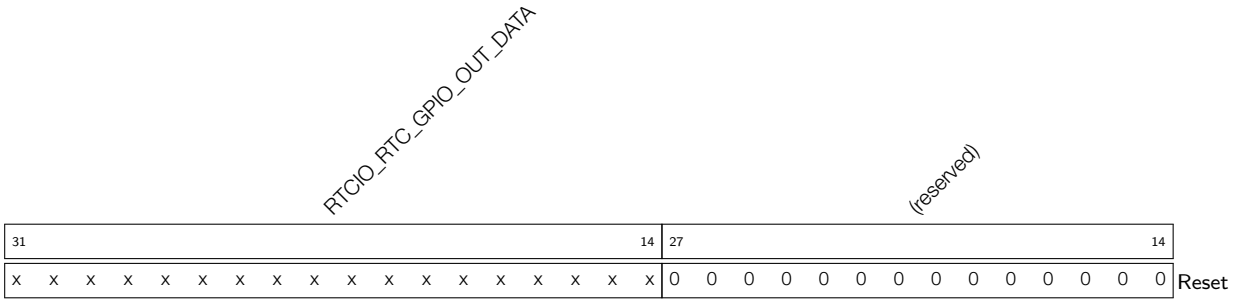
IO_*x*_MCU_WPU 睡眠模式下 pad 的上拉使能。1：内部上拉使能；0：内部上拉关闭。(读 / 写)

IO_*x*_MCU_WPD 睡眠模式下 pad 的下拉使能。1：内部下拉使能；0：内部下拉关闭。(读 / 写)

IO_*x*_SLP_SEL pad 的睡眠模式选择。置 1 将使能睡眠模式。(读 / 写)

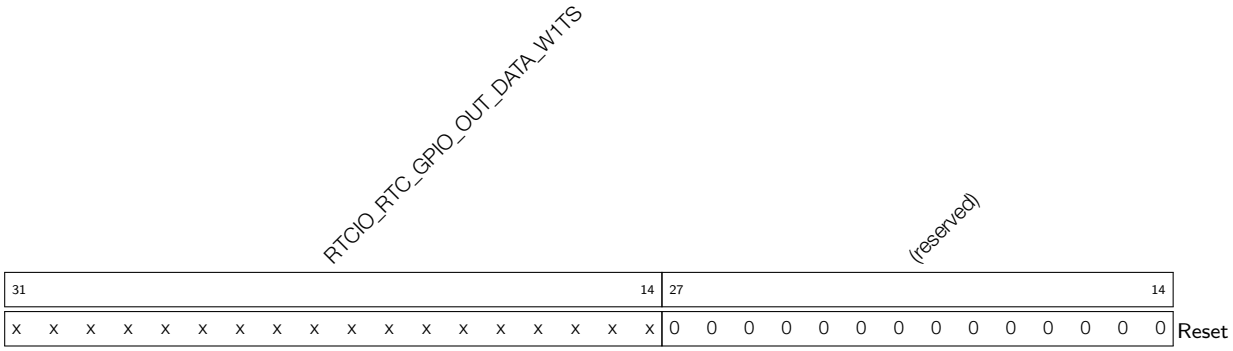
IO_*x*_MCU_OE 睡眠模式下 pad 的输出使能。1: 输出使能；2: 输出关闭。(读 / 写)

Register 4.34: RTCIO_RTC_GPIO_OUT_REG (0x0000)



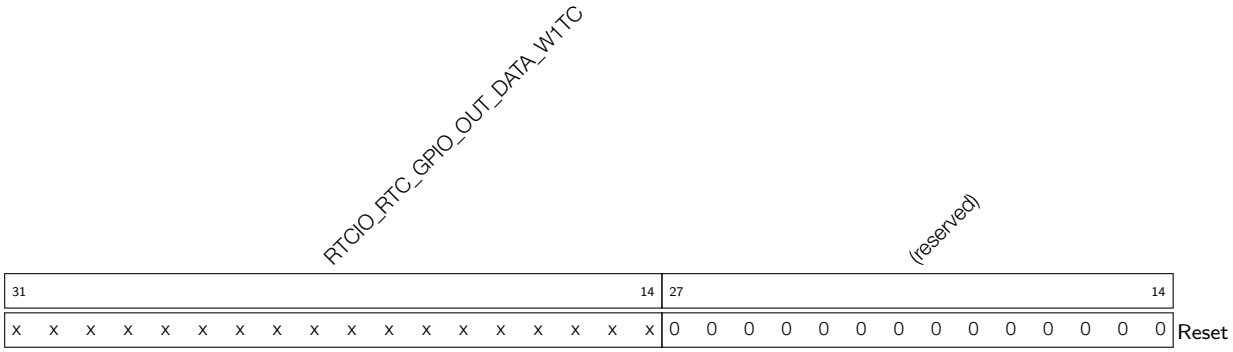
RTCIO_RTC_GPIO_OUT_DATA GPIO0-17 输出寄存器。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，以此类推。(读 / 写)

Register 4.35: RTCIO_RTC_GPIO_OUT_W1TS_REG (0x0001)



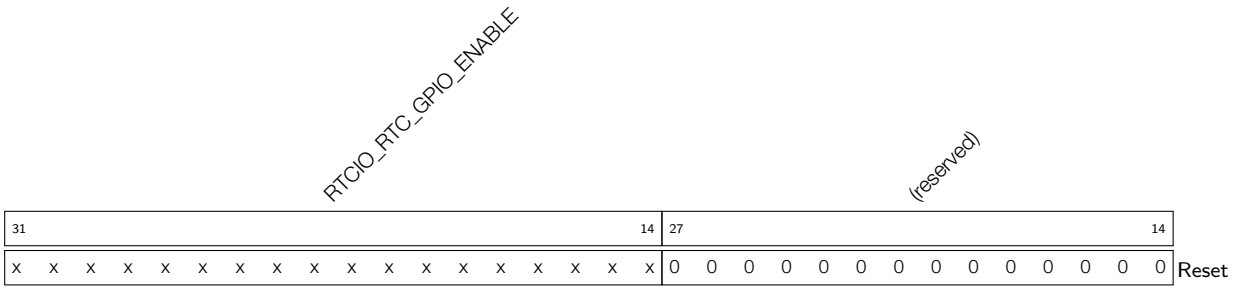
RTCIO_RTC_GPIO_OUT_DATA_W1TS GPIO0-17 输出设置寄存器。每一位置 1，则 RTCIO_RTC_GPIO_OUT 中的相应位也会置 1。(只写)

Register 4.36: RTCIO_RTC_GPIO_OUT_W1TC_REG (0x0002)



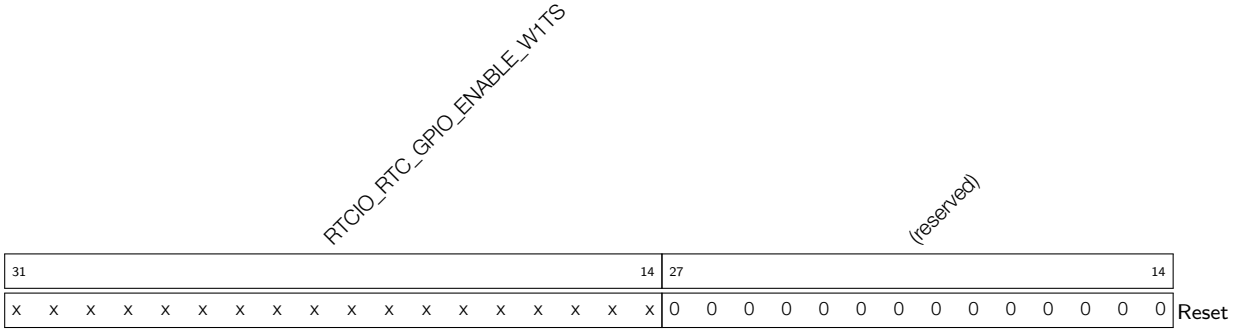
RTCIO_RTC_GPIO_OUT_DATA_W1TC GPIO0-17 输出清除寄存器。每一位置 1，则 RTCIO_RTC_GPIO_OUT 中的相应位会清零。(只写)

Register 4.37: RTCIO_RTC_GPIO_ENABLE_REG (0x0003)



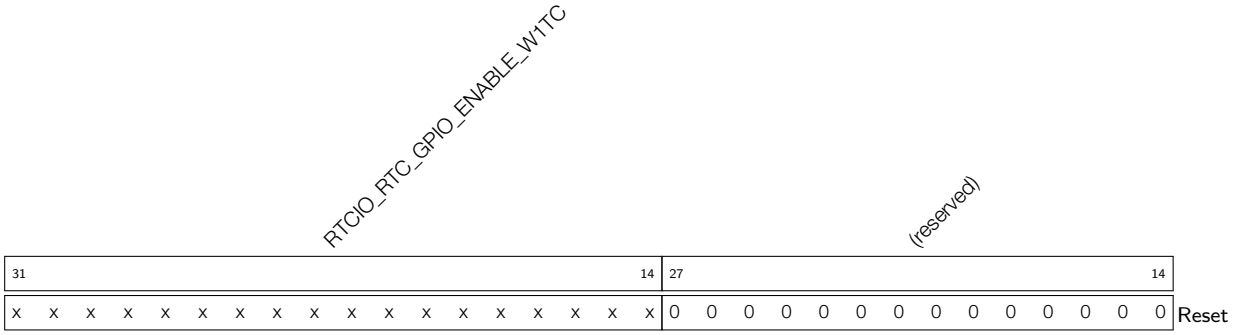
RTCIO_RTC_GPIO_ENABLE GPIO0-17 输出使能。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，以此类推。1 代表此 GPIO 为输出口。(读 / 写)

Register 4.38: RTCIO_RTC_GPIO_ENABLE_W1TS_REG (0x0004)



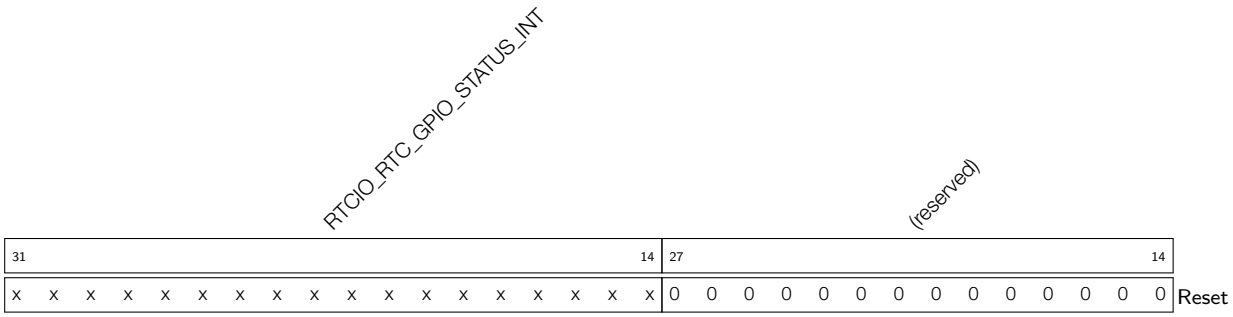
RTCIO_RTC_GPIO_ENABLE_W1TS GPIO0-17 输出使能设置寄存器。每一位置 1，则 RTCIO_RTC_GPIO_ENABLE 中的相应位也会置 1。(只写)

Register 4.39: RTCIO_RTC_GPIO_ENABLE_W1TC_REG (0x0005)



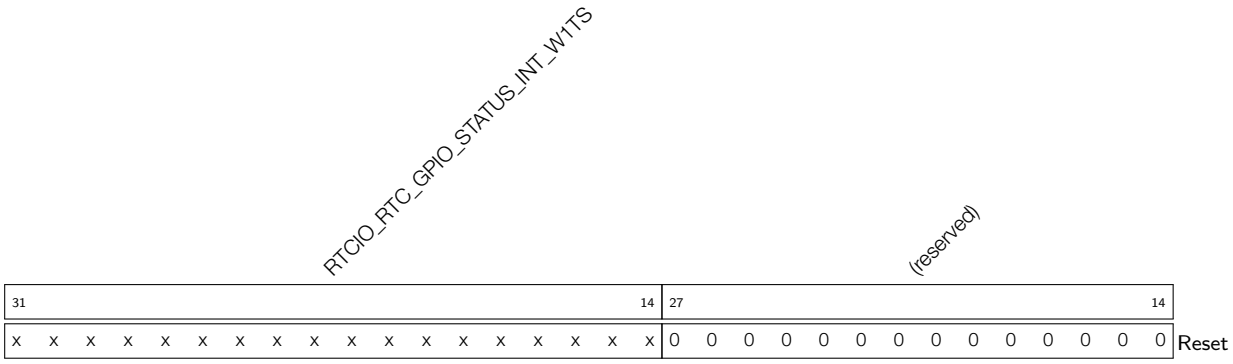
RTCIO_RTC_GPIO_ENABLE_W1TC GPIO0-17 输出使能清除寄存器。每一位置 1，则 RTCIO_RTC_GPIO_ENABLE 中的相应位会清零。(只写)

Register 4.40: RTCIO_RTC_GPIO_STATUS_REG (0x0006)



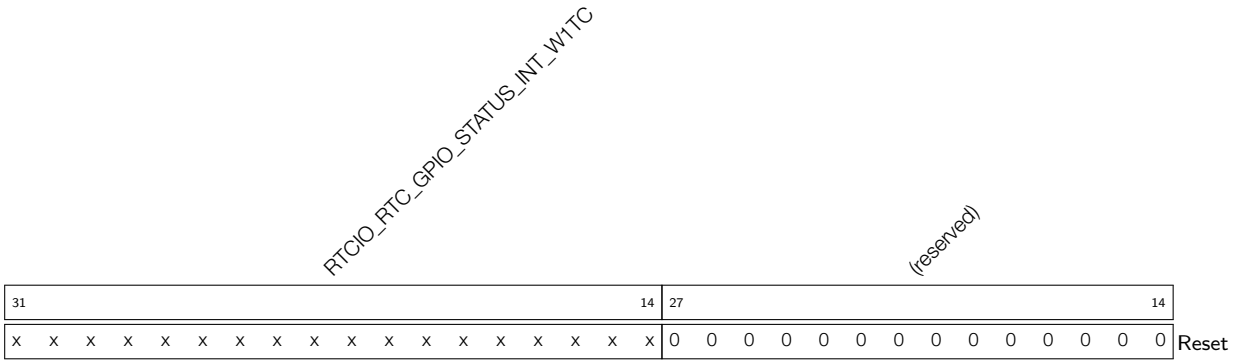
RTCIO_RTC_GPIO_STATUS_INT GPIO0-17 中断状态。Bit14 是 GPIO[0]，bit15 是 GPIO[1]，以此类推。此寄存器应同时和 RTCIO_RTC_GPIO_PIN*n*_REG 的 RTCIO_RTC_GPIO_PIN*n*_INT_TYPE 中断类型配合使用，1 代表有相应中断，0 代表没有中断。（读 / 写）

Register 4.41: RTCIO_RTC_GPIO_STATUS_W1TS_REG (0x0007)



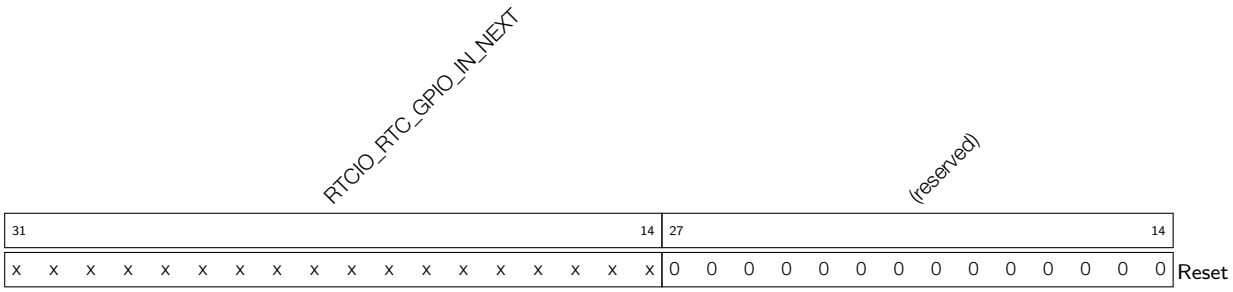
RTCIO_RTC_GPIO_STATUS_INT_W1TS GPIO0-17 中断设置寄存器。每一位置 1，则 RTCIO_RTC_GPIO_STATUS_INT 中的相应位也会置 1。（只写）

Register 4.42: RTCIO_RTC_GPIO_STATUS_W1TC_REG (0x0008)



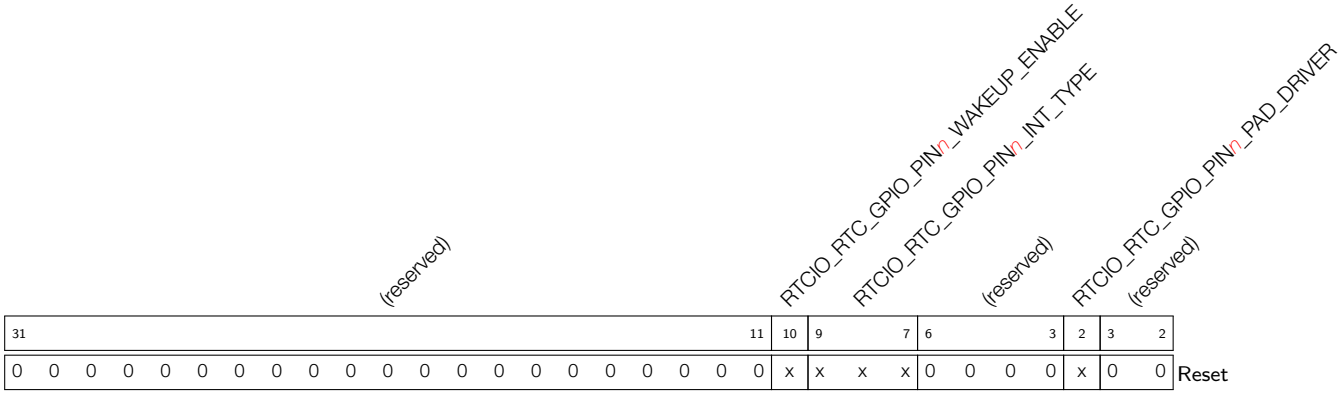
RTCIO_RTC_GPIO_STATUS_INT_W1TC GPIO0-17 中断清除寄存器。每一位置 1，则 RTCIO_RTC_GPIO_STATUS_INT 中的相应位会清零。（只写）

Register 4.43: RTCIO_RTC_GPIO_IN_REG (0x0009)



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 4.44: RTCIO_RTC_GPIO_PIN*n*_REG (*n*: 0-17) (0xA+1n*)**



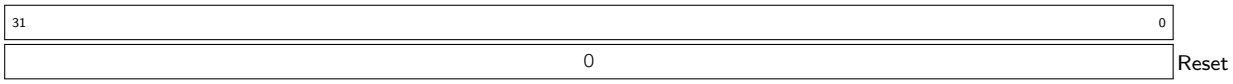
RTCIO_RTC_GPIO_PIN*n*_WAKEUP_ENABLE GPIO 唤醒使能。只能将处于 Light-sleep 的 ESP32 唤醒。（读 / 写）

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。

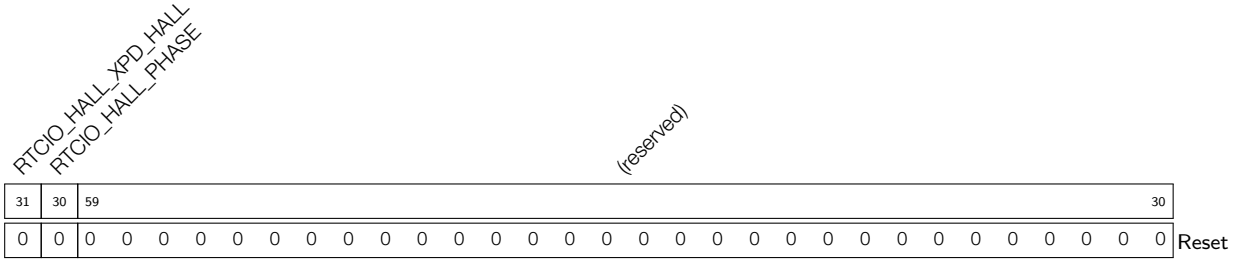
RTCIO_RTC_GPIO_PIN*n*_PAD_DRIVER Pad 驱动器选择。0: 正常输出；1: 开漏模式。（读 / 写）

Register 4.45: RTCIO_DIG_PAD_HOLD_REG (0x001d)



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）

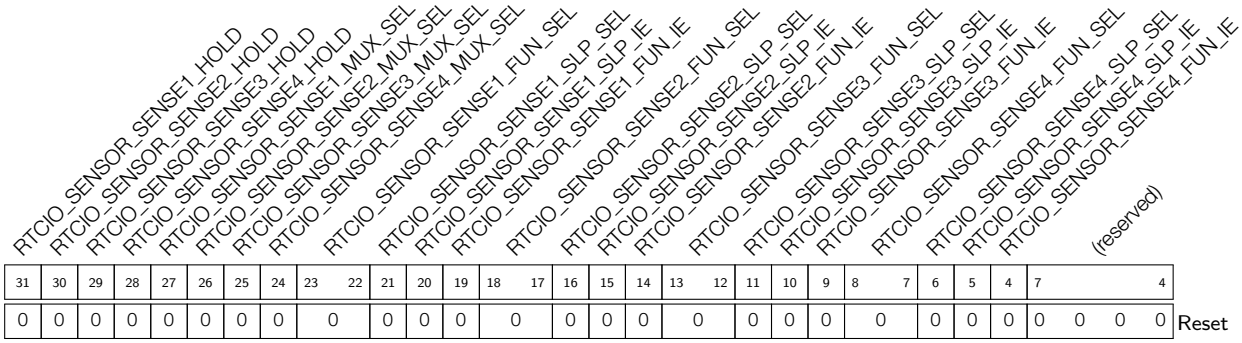
Register 4.46: RTCIO_HALL_SENS_REG (0x001e)



RTCIO_HALL_XPD_HALL 给霍尔传感器上电，连接 VP 和 VN。(读 / 写)

RTCIO_HALL_PHASE 反转霍尔传感器的极性。(读 / 写)

Register 4.47: RTCIO_SENSOR_PADS_REG (0x001f)



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)

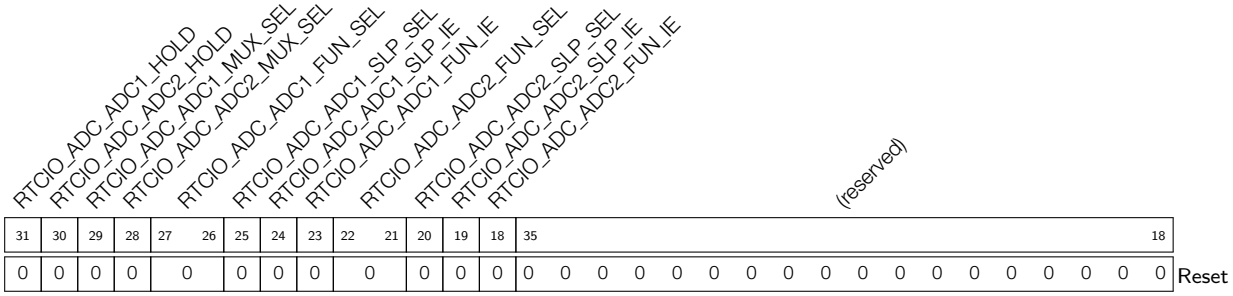
RTCIO_SENSOR_SENSE*n*_FUN_SEL 选择 RTC IO_MUX 功能。0：选择 Function 0；1: 选择 Function 1。(读 / 写)

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 (0x0020)



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

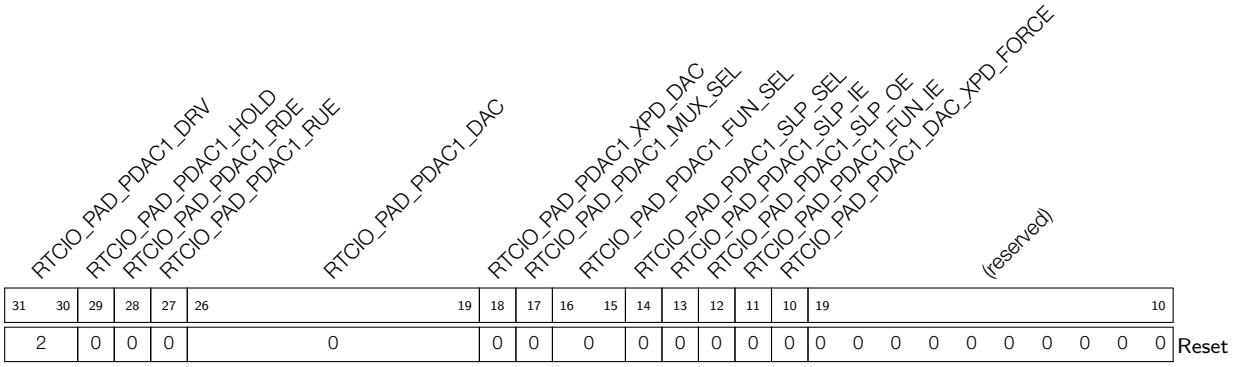
RTCIO_ADC_ADC*n*_FUN_SEL 选择 RTC 功能。0：选择 Function 0；1: 选择 Function 1。(读 / 写)

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 (0x0021)



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 1: 连接 pad 与数字 IO_MUX。(读 / 写)
0: 连接传感器 4 与 RTC 模块。

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)

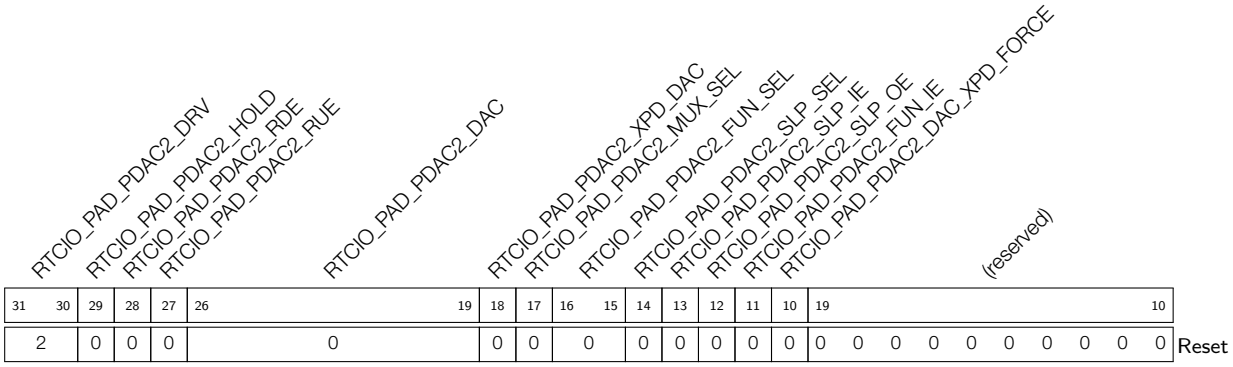
RTCIO_PAD_PDAC1_SLP_IE 睡眠模式下 pad 的输入使能。1: 使能；0: 关闭。(读 / 写)

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 (0x0022)



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 1: 连接 pad 与数字 IO_MUX。(读 / 写)
0: 连接传感器 4 与 RTC 模块。

RTCIO_PAD_PDAC2_FUN_SEL 选择 pad 的 RTC 功能，0: 选择 Function 0；1: 选择 Function 1。(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)

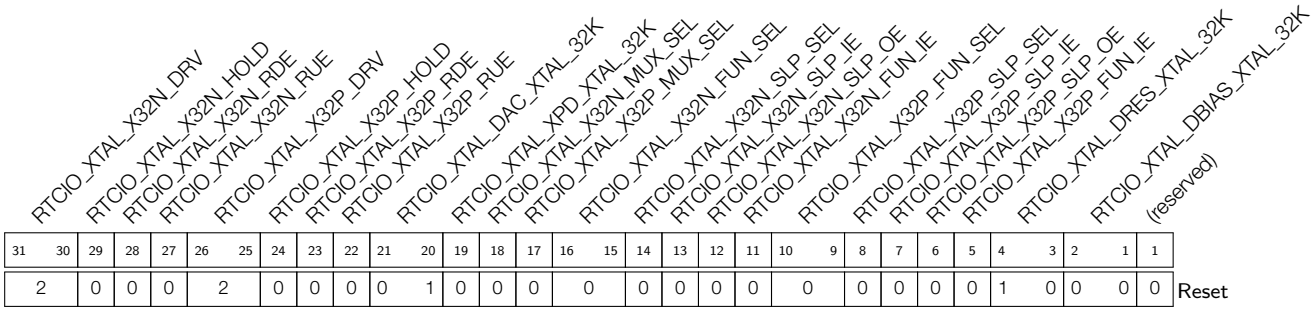
RTCIO_PAD_PDAC2_SLP_IE 睡眠模式下 pad 的输入使能。1: 使能；0: 关闭。(读 / 写)

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 (0x0023)



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

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

RTCIO_XTAL_X32N_FUN_SEL 选择 pad 的 RTC 功能，0: 选择 Function 0；1: 选择 Function 1。(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)

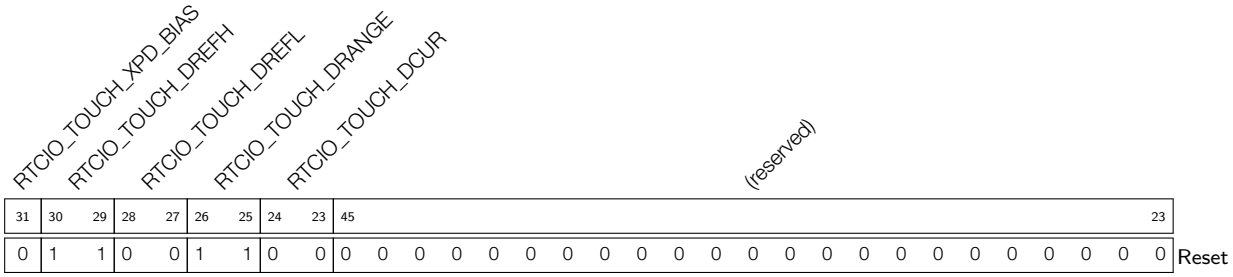
RTCIO_XTAL_X32P_SLP_OE 睡眠模式下 pad 的输出使能。1: 使能；0: 关闭。(读 / 写)

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 (0x0024)



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

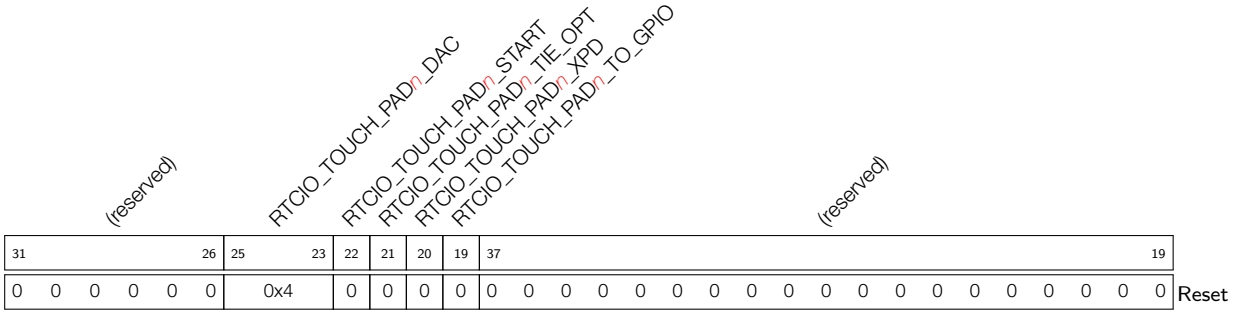
RTCIO_TOUCH_DREFH 触摸传感器波顶电压。(读 / 写)

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_PAD*n*_REG (*n*: 0-9) (0x25+1n*)**



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

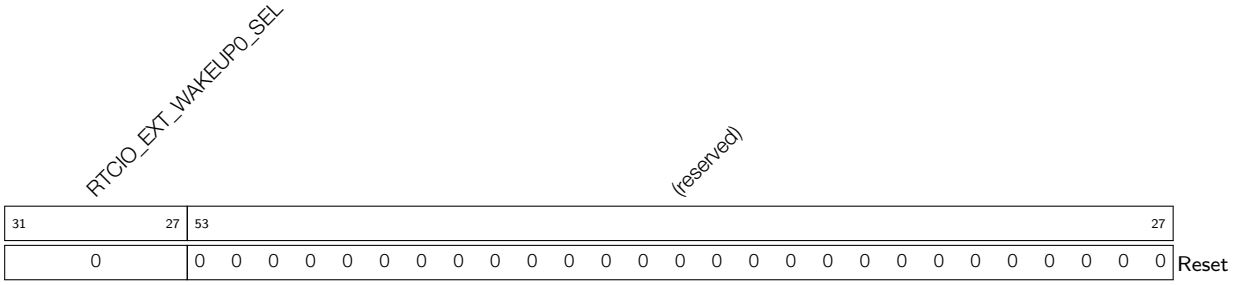
RTCIO_TOUCH_PAD*n*_START 启动触摸传感器。(读 / 写)

RTCIO_TOUCH_PAD*n*_TIE_OPT 默认触摸传感器初始电压位。0：0V；1：VDD_RTC 的电压。(读 / 写)

RTCIO_TOUCH_PAD*n*_XPD 触摸传感器上电。(读 / 写)

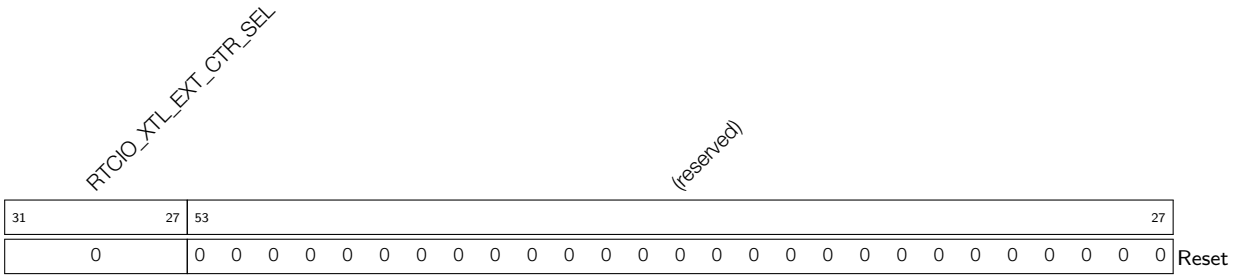
RTCIO_TOUCH_PAD*n*_TO_GPIO 连接 RTC pad 输入与数字 pad 输入，可以置 0。(读 / 写)

Register 4.54: RTCIO_EXT_WAKEUP0_REG (0x002f)



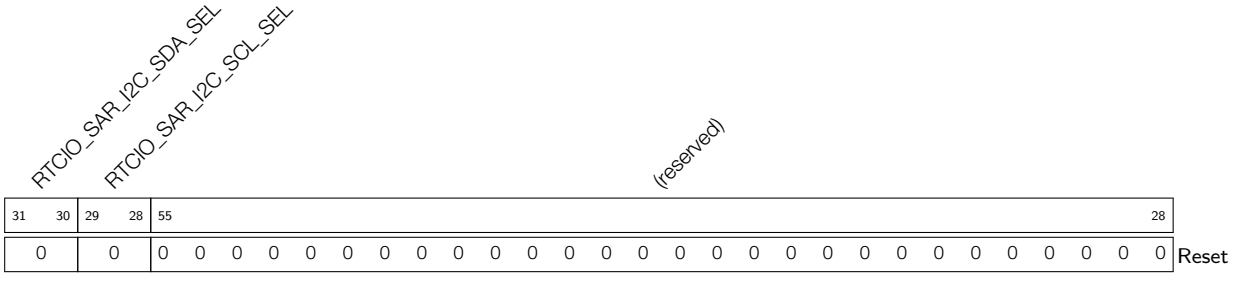
RTCIO_EXT_WAKEUP0_SEL GPIO[0-17] 可以用于将芯片从睡眠模式中唤醒。此寄存器选择 pad 源将芯片从 Deep/Light-sleep 模式中唤醒。0: 选择 GPIO0；1: 选择 GPIO2，以此类推。（读 / 写）

Register 4.55: RTCIO_XTL_EXT_CTR_REG (0x0030)



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 RTCIO_RTC_EXT_XTAL_CONF_REG[30] 上的逻辑值是晶振断电使能信号。（读 / 写）

Register 4.56: RTCIO_SAR_I2C_IO_REG (0x0031)



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]。（读 / 写）

RTCIO_SAR_I2C_SCL_SEL 选择另一个 pad 作为 RTC I2C SCL signal。0: 选择 TOUCH_PAD[1]；1: 选择 TOUCH_PAD[3]。（读 / 写）

5. LED_PWM

5.1 概述

LED_PWM 用于控制 LED 的亮度和颜色，同时也可以因其它目的产生 PWM 信号。LED_PWM 拥有 16 路通道，分别有 8 路高速通道和 8 路低速通道。在本文中，hsch*n* 指高速通道，lsch*n* 指低速通道，分别由 4 个命名为 h_timer*x* 和 l_timer*x* 的定时器驱动，能够产生独立的数字波形。高速通道和低速均能分别由各自的分频器控制。PWM 还能够自动调节占空比，在无须处理器干预的情况下实现亮度和颜色渐变。

5.2 功能描述

5.2.1 架构

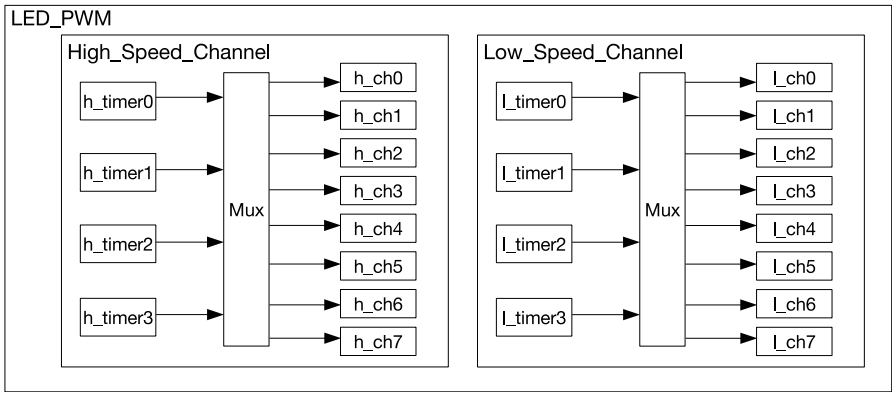


图 10: LED_PWM 架构

图 10 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*。

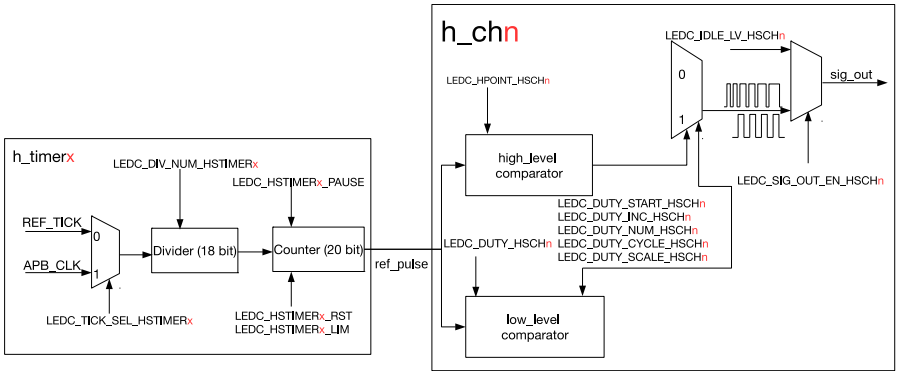


图 11: LED_PWM 高速通道框图

图 11 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。

5.2.2 分频器

一个高速时钟由选择器构成，选择器可以从 2 个时钟源中任选一个时钟源：REF_TICK 和 APB_CLK，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_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，低 8 位为小数部分，计数范围由 LEDC_HSTIMER*x*_LIM 进行配置，每次计数达到最大值 ref_pulse 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。分频器的分频系数以及计数器的计数范围共同决定了输出信号的频率：

$$f_{\text{sig\_out}} = \frac{f_{\text{REF\_TICK}} \cdot (!\text{LEDC\_TICK\_SEL\_HSTIMER}x) + f_{\text{APB\_CLK}} \cdot \text{LEDC\_TICK\_SEL\_HSTIMER}x}{\text{LEDC\_DIV\_NUM\_HSTIMER}x \cdot 2^{\text{LEDC\_HSTIMER}x\text{\_LIM}}}$$

低速通道的分频器 l_timer*x* 相对于高速通道的分频器 h_timer*x* 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_LSTIMER*x*_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。

5.2.3 通道

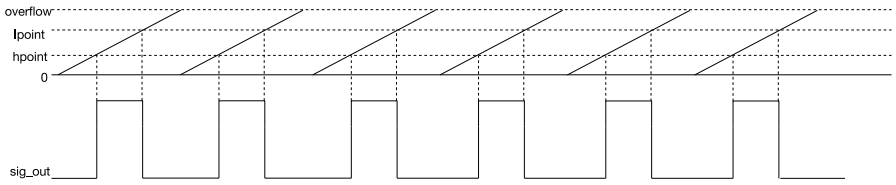


图 12: LED_PWM 输出信号图

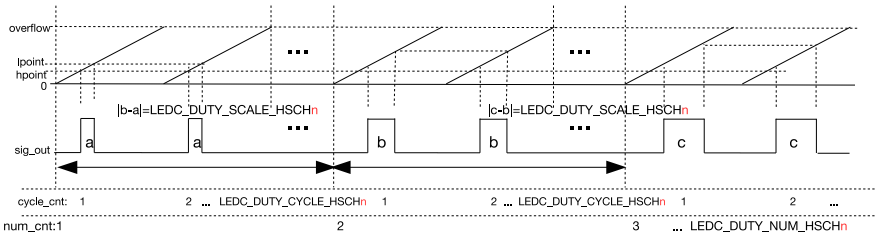


图 13: 渐变占空比输出信号图

每个通道有两个比较器，即图 11 中的 high_level_comparator 以及 low_level_comparator。high_level_comparator 的比较值 hpoint 由 LEDC_HPOINT_HSCH*n* 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 lpoint 由 LEDC_DUTY_HSCH*n*，LEDC_DUTY_START_HSCH*n*，LEDC_DUTY_INC_HSCH*n*，LEDC_DUTY_NUM_HSCH*n* 以及 LEDC_DUTY_SCALE_HSCH*n* 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 12 为 LED_PWM 输出信号图。

LEDC_DUTY_HSCH*n* 是一个具有 4 位小数的浮点寄存器，其中高 20 位用于 lpoint 的计算，低 4 位用于抖动 lpoint 的值。当低 4 位非 0 时，输出信号的脉冲宽度有 LEDC_DUTY_HSCH*n*[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_HSCH*n*，通道更新后的 LEDC_DUTY_HSCH*n* 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_HSCH*n* 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_HSCH*n* 个脉冲，LEDC_DUTY_HSCH*n*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 13 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*。

5.2.4 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。

5.3 寄存器列表

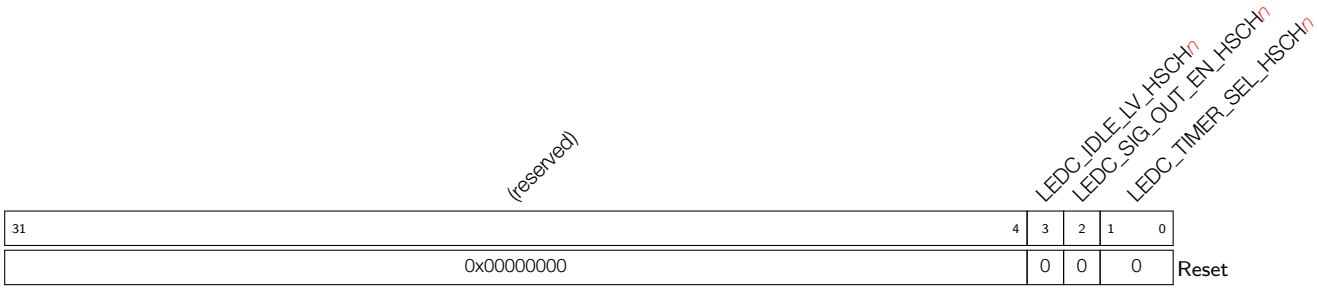
名称	描述	地址	访问
配置寄存器			
LEDC_CONF_REG	Global ledc configuration register	0x00000190	读 / 写
LEDC_HSCH0_CONF0_REG	Configuration register 0 for high-speed channel 0	0x00000000	读 / 写
LEDC_HSCH1_CONF0_REG	Configuration register 0 for high-speed channel 1	0x00000014	读 / 写
LEDC_HSCH2_CONF0_REG	Configuration register 0 for high-speed channel 2	0x00000028	读 / 写
LEDC_HSCH3_CONF0_REG	Configuration register 0 for high-speed channel 3	0x0000003C	读 / 写
LEDC_HSCH4_CONF0_REG	Configuration register 0 for high-speed channel 4	0x00000050	读 / 写
LEDC_HSCH5_CONF0_REG	Configuration register 0 for high-speed channel 5	0x00000064	读 / 写
LEDC_HSCH6_CONF0_REG	Configuration register 0 for high-speed channel 6	0x00000078	读 / 写
LEDC_HSCH7_CONF0_REG	Configuration register 0 for high-speed channel 7	0x0000008C	读 / 写
LEDC_HSCH0_CONF1_REG	Configuration register 1 for high-speed channel 0	0x0000000C	读 / 写
LEDC_HSCH1_CONF1_REG	Configuration register 1 for high-speed channel 1	0x00000020	读 / 写
LEDC_HSCH2_CONF1_REG	Configuration register 1 for high-speed channel 2	0x00000034	读 / 写
LEDC_HSCH3_CONF1_REG	Configuration register 1 for high-speed channel 3	0x00000048	读 / 写
LEDC_HSCH4_CONF1_REG	Configuration register 1 for high-speed channel 4	0x0000005C	读 / 写
LEDC_HSCH5_CONF1_REG	Configuration register 1 for high-speed channel 5	0x00000070	读 / 写
LEDC_HSCH6_CONF1_REG	Configuration register 1 for high-speed channel 6	0x00000084	读 / 写
LEDC_HSCH7_CONF1_REG	Configuration register 1 for high-speed channel 7	0x00000098	读 / 写
LEDC_LSCH0_CONF0_REG	Configuration register 0 for low-speed channel 0	0x000000A0	读 / 写
LEDC_LSCH1_CONF0_REG	Configuration register 0 for low-speed channel 1	0x000000B4	读 / 写
LEDC_LSCH2_CONF0_REG	Configuration register 0 for low-speed channel 2	0x000000C8	读 / 写
LEDC_LSCH3_CONF0_REG	Configuration register 0 for low-speed channel 3	0x000000DC	读 / 写
LEDC_LSCH4_CONF0_REG	Configuration register 0 for low-speed channel 4	0x000000F0	读 / 写
LEDC_LSCH5_CONF0_REG	Configuration register 0 for low-speed channel 5	0x00000104	读 / 写
LEDC_LSCH6_CONF0_REG	Configuration register 0 for low-speed channel 6	0x00000118	读 / 写
LEDC_LSCH7_CONF0_REG	Configuration register 0 for low-speed channel 7	0x0000012C	读 / 写
LEDC_LSCH0_CONF1_REG	Configuration register 1 for low-speed channel 0	0x000000AC	读 / 写

名称	描述	地址	访问
LEDC_LSCH1_CONF1_REG	Configuration register 1 for low-speed channel 1	0x000000C0	读 / 写
LEDC_LSCH2_CONF1_REG	Configuration register 1 for low-speed channel 2	0x000000D4	读 / 写
LEDC_LSCH3_CONF1_REG	Configuration register 1 for low-speed channel 3	0x000000E8	读 / 写
LEDC_LSCH4_CONF1_REG	Configuration register 1 for low-speed channel 4	0x000000FC	读 / 写
LEDC_LSCH5_CONF1_REG	Configuration register 1 for low-speed channel 5	0x00000110	读 / 写
LEDC_LSCH6_CONF1_REG	Configuration register 1 for low-speed channel 6	0x00000124	读 / 写
LEDC_LSCH7_CONF1_REG	Configuration register 1 for low-speed channel 7	0x00000138	读 / 写
占空比寄存器			
LEDC_HSCH0_DUTY_REG	Initial duty cycle for high-speed channel 0	0x00000008	读 / 写
LEDC_HSCH1_DUTY_REG	Initial duty cycle for high-speed channel 1	0x0000001C	读 / 写
LEDC_HSCH2_DUTY_REG	Initial duty cycle for high-speed channel 2	0x00000030	读 / 写
LEDC_HSCH3_DUTY_REG	Initial duty cycle for high-speed channel 3	0x00000044	读 / 写
LEDC_HSCH4_DUTY_REG	Initial duty cycle for high-speed channel 4	0x00000058	读 / 写
LEDC_HSCH5_DUTY_REG	Initial duty cycle for high-speed channel 5	0x0000006C	读 / 写
LEDC_HSCH6_DUTY_REG	Initial duty cycle for high-speed channel 6	0x00000080	读 / 写
LEDC_HSCH7_DUTY_REG	Initial duty cycle for high-speed channel 7	0x00000094	读 / 写
LEDC_HSCH0_DUTY_R_REG	Current duty cycle for high-speed channel 0	0x00000010	只读
LEDC_HSCH1_DUTY_R_REG	Current duty cycle for high-speed channel 1	0x00000024	只读
LEDC_HSCH2_DUTY_R_REG	Current duty cycle for high-speed channel 2	0x00000038	只读
LEDC_HSCH3_DUTY_R_REG	Current duty cycle for high-speed channel 3	0x0000004C	只读
LEDC_HSCH4_DUTY_R_REG	Current duty cycle for high-speed channel 4	0x00000060	只读
LEDC_HSCH5_DUTY_R_REG	Current duty cycle for high-speed channel 5	0x00000074	只读
LEDC_HSCH6_DUTY_R_REG	Current duty cycle for high-speed channel 6	0x00000088	只读
LEDC_HSCH7_DUTY_R_REG	Current duty cycle for high-speed channel 7	0x0000009C	只读
LEDC_LSCH0_DUTY_REG	Initial duty cycle for low-speed channel 0	0x000000A8	读 / 写
LEDC_LSCH1_DUTY_REG	Initial duty cycle for low-speed channel 1	0x000000BC	读 / 写
LEDC_LSCH2_DUTY_REG	Initial duty cycle for low-speed channel 2	0x000000D0	读 / 写
LEDC_LSCH3_DUTY_REG	Initial duty cycle for low-speed channel 3	0x000000E4	读 / 写
LEDC_LSCH4_DUTY_REG	Initial duty cycle for low-speed channel 4	0x000000F8	读 / 写
LEDC_LSCH5_DUTY_REG	Initial duty cycle for low-speed channel 5	0x0000010C	读 / 写
LEDC_LSCH6_DUTY_REG	Initial duty cycle for low-speed channel 6	0x00000120	读 / 写
LEDC_LSCH7_DUTY_REG	Initial duty cycle for low-speed channel 7	0x00000134	读 / 写
LEDC_LSCH0_DUTY_R_REG	Current duty cycle for low-speed channel 0	0x000000B0	只读
LEDC_LSCH1_DUTY_R_REG	Current duty cycle for low-speed channel 1	0x000000C4	只读
LEDC_LSCH2_DUTY_R_REG	Current duty cycle for low-speed channel 2	0x000000D8	只读
LEDC_LSCH3_DUTY_R_REG	Current duty cycle for low-speed channel 3	0x000000EC	只读
LEDC_LSCH4_DUTY_R_REG	Current duty cycle for low-speed channel 4	0x00000100	只读
LEDC_LSCH5_DUTY_R_REG	Current duty cycle for low-speed channel 5	0x00000114	只读
LEDC_LSCH6_DUTY_R_REG	Current duty cycle for low-speed channel 6	0x00000128	只读
LEDC_LSCH7_DUTY_R_REG	Current duty cycle for low-speed channel 7	0x0000013C	只读
分频器寄存器			
LEDC_HSTIMER0_CONF_REG	High-speed timer 0 configuration	0x00000140	读 / 写
LEDC_HSTIMER1_CONF_REG	High-speed timer 1 configuration	0x00000148	读 / 写
LEDC_HSTIMER2_CONF_REG	High-speed timer 2 configuration	0x00000150	读 / 写

名称	描述	地址	访问
LEDC_HSTIMER3_CONF_REG	High-speed timer 3 configuration	0x00000158	读 / 写
LEDC_HSTIMER0_VALUE_REG	High-speed timer 0 current counter value	0x00000144	只读
LEDC_HSTIMER1_VALUE_REG	High-speed timer 1 current counter value	0x0000014C	只读
LEDC_HSTIMER2_VALUE_REG	High-speed timer 2 current counter value	0x00000154	只读
LEDC_HSTIMER3_VALUE_REG	High-speed timer 3 current counter value	0x0000015C	只读
LEDC_HSTIMER0_CONF_REG	Low-speed timer 0 configuration	0x00000140	读 / 写
LEDC_HSTIMER1_CONF_REG	Low-speed timer 1 configuration	0x00000148	读 / 写
LEDC_HSTIMER2_CONF_REG	Low-speed timer 2 configuration	0x00000150	读 / 写
LEDC_HSTIMER3_CONF_REG	Low-speed timer 3 configuration	0x00000158	读 / 写
LEDC_HSTIMER0_VALUE_REG	Low-speed timer 0 current counter value	0x00000144	只读
LEDC_HSTIMER1_VALUE_REG	Low-speed timer 1 current counter value	0x0000014C	只读
LEDC_HSTIMER2_VALUE_REG	Low-speed timer 2 current counter value	0x00000154	只读
LEDC_HSTIMER3_VALUE_REG	Low-speed timer 3 current counter value	0x0000015C	只读
中断寄存器			
LEDC_INT_RAW_REG	Raw interrupt status	0x00000180	只读
LEDC_INT_ST_REG	Masked interrupt status	0x00000184	只读
LEDC_INT_ENA_REG	Interrupt enable bits	0x00000188	读 / 写
LEDC_INT_CLR_REG	Interrupt clear bits	0x0000018C	只写

5.4 寄存器

Register 5.1: LEDC_HSCH*n*_CONF0_REG (*n*: 0-7) (0x1C+0x10n*)**



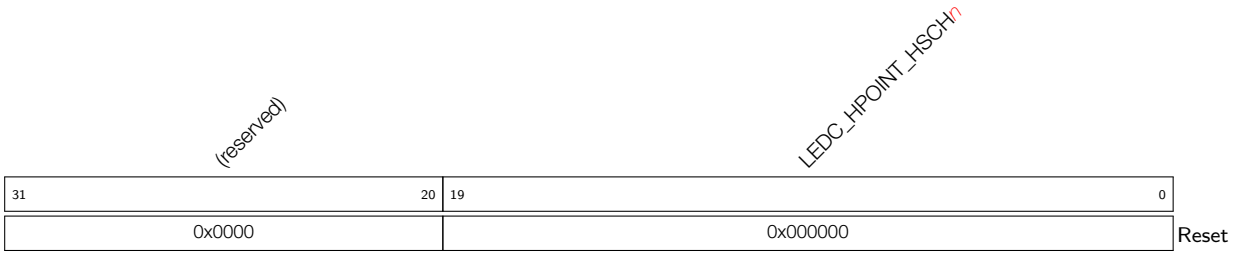
LEDC_IDLE_LV_HSCH*n* 高速通道 *n* 不工作时，用于控制输出值。（读 / 写）

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）

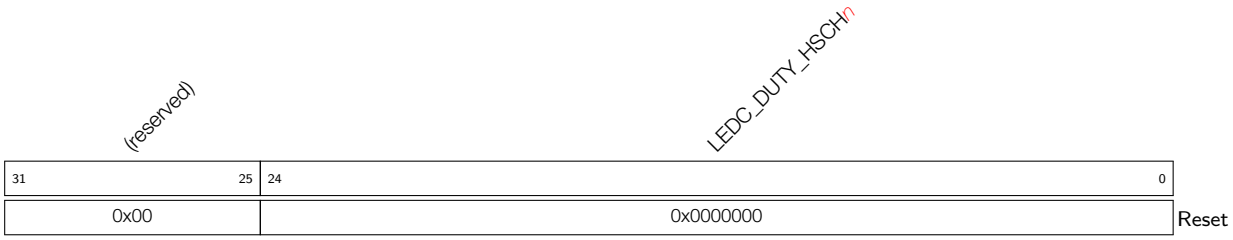
0: 选择 hstimer0；
1: 选择 hstimer1；
2: 选择 hstimer2；
3: 选择 hstimer3。

Register 5.2: LEDC_HSCH*n*_HPOINT_REG (*n*: 0-7) (0x20+0x10n*)**



LEDC_HPOINT_HSCH*n* 当高速通道 *n* 的 htimerx(x=[0,3]) 达到 reg_hpoint_hsch*n*[19:0]，输出信号翻转为高电平。（读 / 写）

Register 5.3: LEDC_HSCH*n*_DUTY_REG (*n*: 0-7) (0x24+0x10*n)



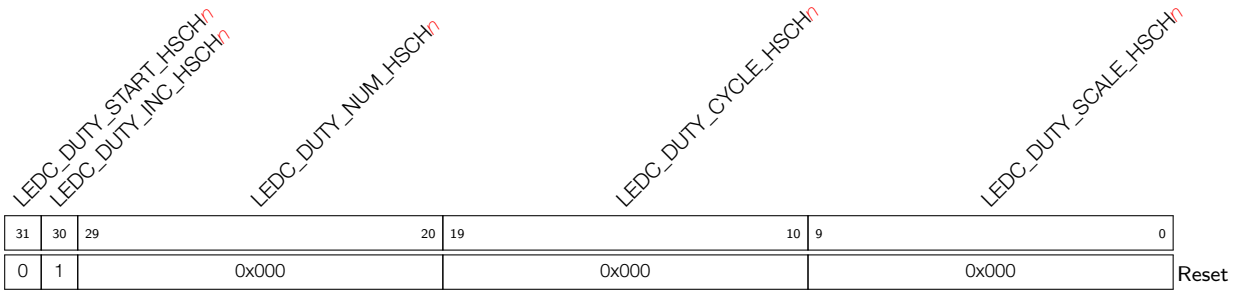
LEDC_DUTY_HSCH*n* 用于控制输出占空比。当通道 *n* 的 hstimerx(x=[0,3]) 达到 reg_lpoint_hsch*n* 时，输出信号翻转为低电平。

reg_lpoint_hsch*n*=(reg_hpoint_hsch*n*[19:0]+reg_duty_hsch*n*[24:4]) (1)

reg_lpoint_hsch*n*=(reg_hpoint_hsch*n*[19:0]+reg_duty_hsch*n*[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 5.4: LEDC_HSCH*n*_CONF1_REG (*n*: 0-7) (0x28+0x10*n)



LEDC_DUTY_START_HSCH*n* 当配置了 RREG_DUTY_NUM_HSCH*n*, REG_DUTY_CYCLE_HSCH*n* 以及 REG_DUTY_SCALE_HSCH*n* 时，需要置位 REG_DUTY_START_HSCH*n*，这些寄存器才会生效。硬件自动清零此位。(读 / 写)

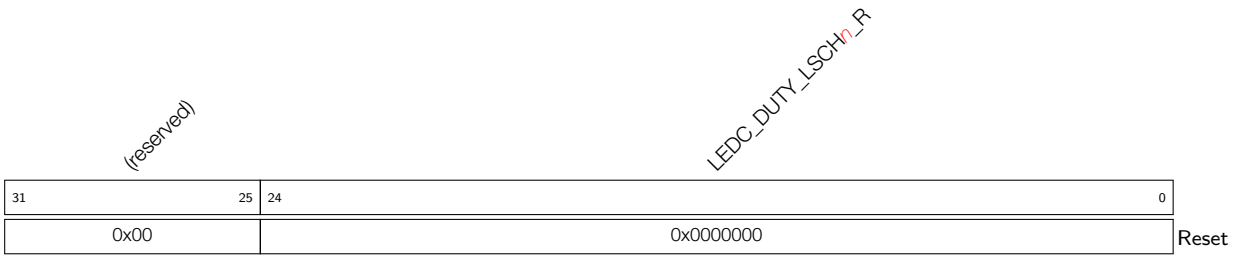
LEDC_DUTY_INC_HSCH*n* 用于递增或递减高速通道 *n* 的输出信号的占空比。(读 / 写)

LEDC_DUTY_NUM_HSCH*n* 用于控制高速通道 *n* 占空比变化的次数。(读 / 写)

LEDC_DUTY_CYCLE_HSCH*n* 每 REG_DUTY_CYCLE_HSCH*n* 个时钟周期递增或递减高速通道 *n* 的占空比。

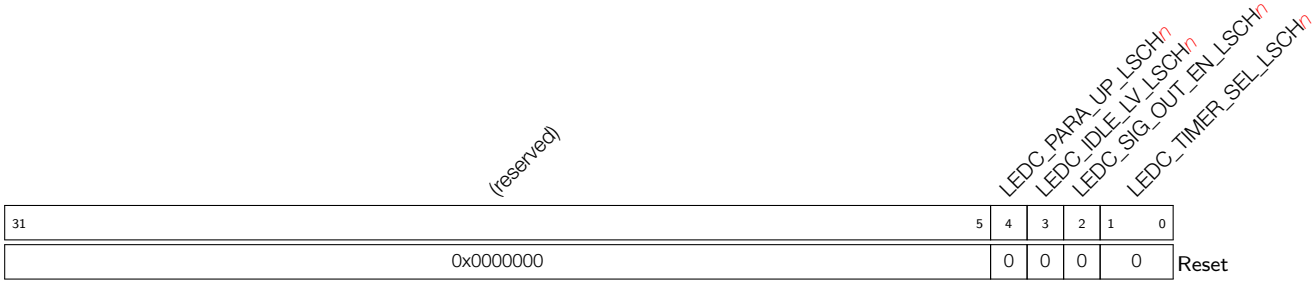
LEDC_DUTY_SCALE_HSCH*n* 用于递增或递减高速通道 *n* 的步长。(读 / 写)

Register 5.5: LEDC_HSCH*n*_DUTY_R_REG (*n*: 0-7) (0x2C+0x10n*)**



LEDC_DUTY_HSCH*n*_R 当下高速通道 *n* 的输出信号的占空比。(只读)

Register 5.6: LEDC_LSCH*n*_CONF0_REG (*n*: 0-7) (0xBC+0x10n*)**



LEDC_PARA_UP_LSCH*n* 用于更新低速通道 *n* 的 LEDC_LSCH*n*_HPOINT 和 LEDC_LSCH*n*_DUTY 寄存器。(读 / 写)

LEDC_IDLE_LV_LSCH*n* 当低速通道 *n* 不工作时，用于控制输出值。(读 / 写)

LEDC_SIG_OUT_EN_LSCH*n* 低速通道 *n* 的输出控制位。(读 / 写)

LEDC_TIMER_SEL_LSCH*n* 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)

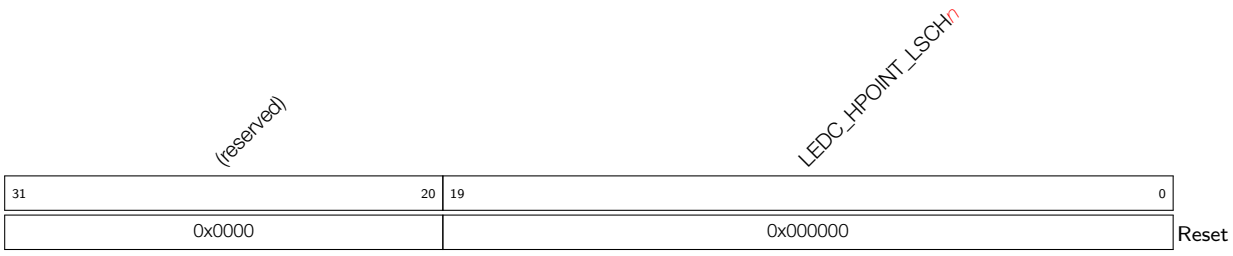
0: 选择 lstimer0；

1: 选择 lstimer1；

2: 选择 lstimer2；

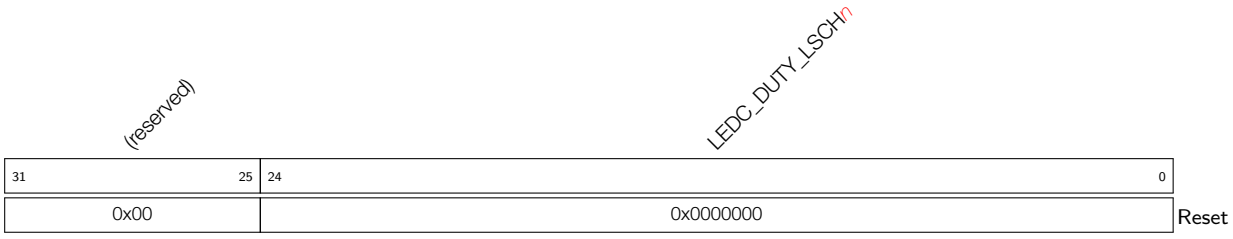
3: 选择 lstimer3。

Register 5.7: LEDC_LSCH*n*_HPOINT_REG (*n*: 0-7) (0xC0+0x10n*)**



LEDC_HPOINT_LSCH*n* 当低速通道 *n* 的 lstimerx(x=[0,3]) 达到 reg_hpoint_lsch*n*[19:0] 时，输出信号翻转为高电平。(读 / 写)

Register 5.8: LEDC_LSCHn_DUTY_REG (n: 0-7) (0xC4+0x10*n)



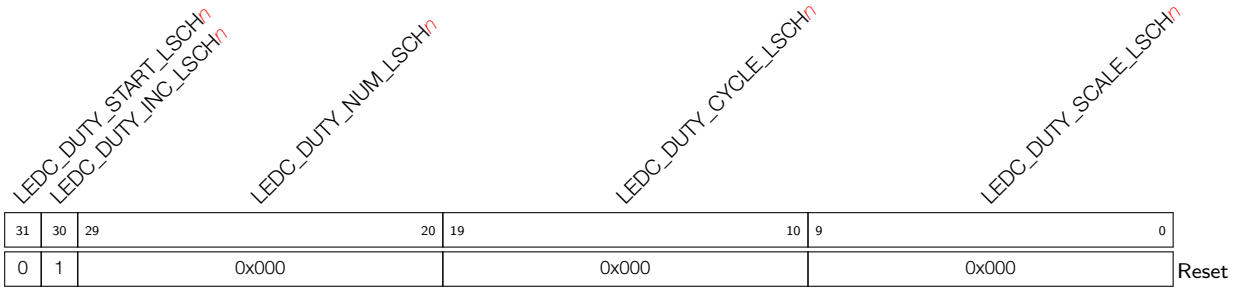
LEDC_DUTY_LSCHn 用于控制输出占空比。当低速通道 n 的 lstimerx(x=[0,3]) 达到 reg_lpoint_lschn时，输出信号翻转为低电平。(读 / 写)

reg_lpoint_lschn=(reg_hpoint_lschn[19:0]+reg_duty_lschn[24:4]) (1)

reg_lpoint_lschn=(reg_hpoint_lschn[19:0]+reg_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 5.9: LEDC_LSCHn_CONF1_REG (n: 0-7) (0xC8+0x10*n)



LEDC_DUTY_START_LSCHn 当配置了 reg_duty_num_hschn、reg_duty_cycle_hschn 以及 reg_duty_scale_hschn 时，需要置位 reg_duty_start_hschn，这些寄存器才能生效。硬件自动清零此位。(读 / 写)

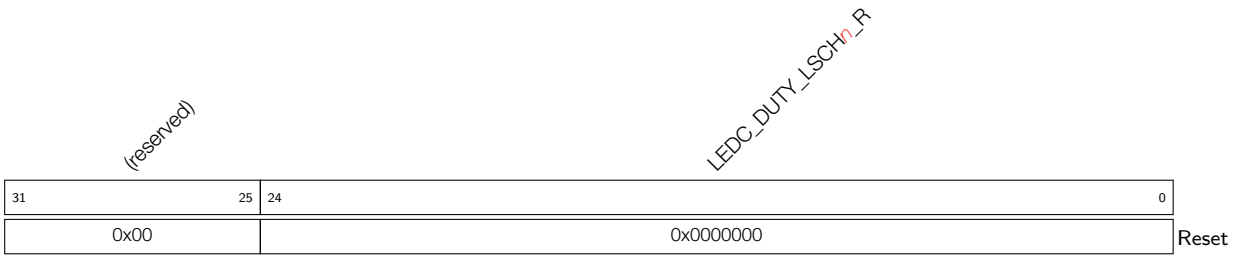
LEDC_DUTY_INC_LSCHn 用于递增或递减低速通道 n 的输出信号的占空比。(读 / 写)

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 reg_duty_cycle_lschn 个时钟周期递增或递减占空比。(读 / 写)

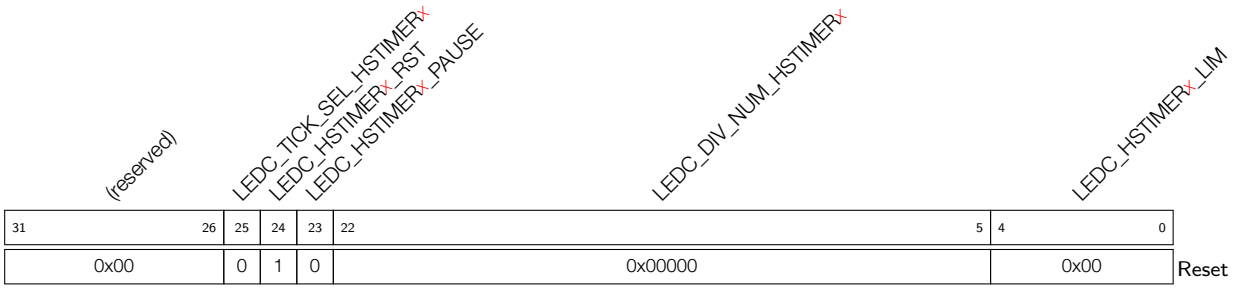
LEDC_DUTY_SCALE_LSCHn 用递增或递减低速通道 n 的步长。(读 / 写)

Register 5.10: LEDC_LSCH*n*_DUTY_R_REG (*n*: 0-7) (0xCC+0x10n*)**



LEDC_DUTY_R_LSCH*n*_R 低速通道 *n* 的输出信号的当前占空比。(只读)

Register 5.11: LEDC_HSTIMER*x*_CONF_REG (*x*: 0-3) (0x140+8x*)**



LEDC_TICK_SEL_HSTIMER*x* 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。

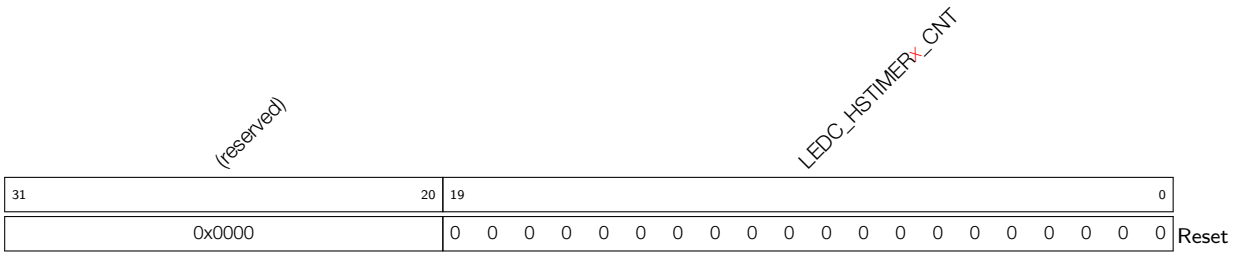
LEDC_HSTIMER*x*_RST 用于复位高速时钟计数器 *x*，复位后计数器为 0。(读 / 写)

LEDC_HSTIMER*x*_PAUSE 用于暂停高速时钟计数器 *x*。(读 / 写)

LEDC_DIV_NUM_HSTIMER*x* 用于配置高速时钟计数器 *x*，低 8 位为小数部分。(读 / 写)

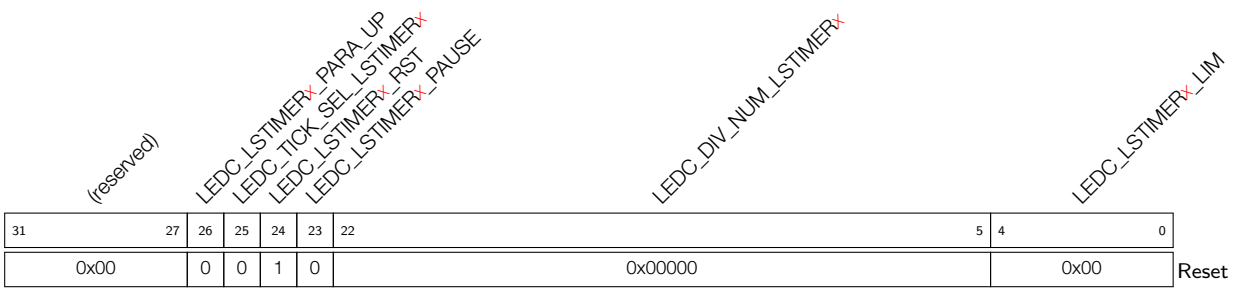
LEDC_HSTIMER*x*_LIM 用于控制高速时钟计数器 *x* 的计数范围。计数范围为 [0,2**reg_hstimer*x*_lim]，最大位宽为 20 位。(读 / 写)

Register 5.12: LEDC_HSTIMER*x*_VALUE_REG (*x*: 0-3) (0x144+8x*)**



LEDC_HSTIMER*x*_CNT 软件可以读取高速时钟计数器 *x* 的当前计数值。(只读)

Register 5.13: LEDC_LSTIMERx_CONF_REG (x: 0-3) (0x160+8*x)



LEDC_LSTIMERx_PARA_UP 用于更新 REG_DIV_NUM_LSTIMEx 和 REG_LSTIMERx_LIM。

LEDC_TICK_SEL_LSTIMERx 用于为低速时钟计数器 x 选择 SLOW_CLK 或 REF_TICK 时钟。(读 / 写)

1：SLOW_CLK；

0：REF_TICK。

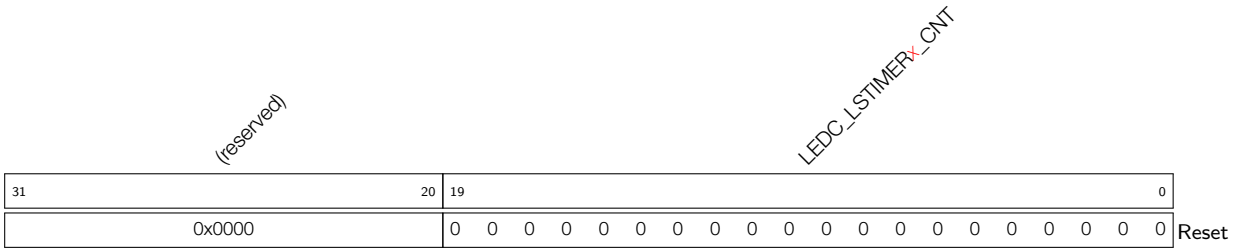
LEDC_LSTIMERx_RST 用于复位低速时钟计数器 x，复位后计数器为 0。(读 / 写)

LEDC_LSTIMERx_PAUSE 用于暂停低速时钟计数器 x。(读 / 写)

LEDC_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)

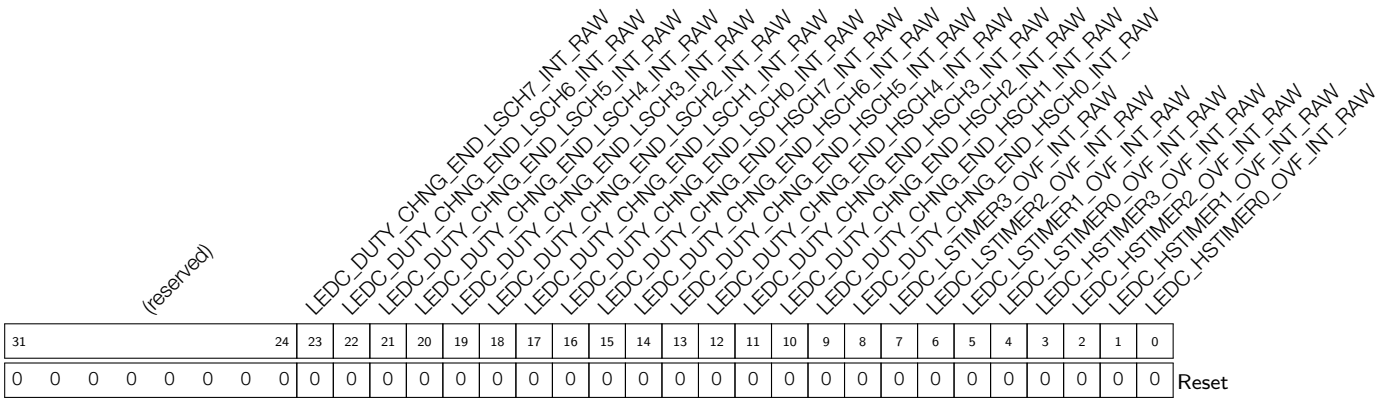
LEDC_LSTIMERx_LIM 用于控制低速时钟计数器 x 的计数范围。计数范围为 [0,2**reg_lstimerx_lim]，最大位宽为 20 位。(读 / 写)

Register 5.14: 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 5.15: LEDC_INT_RAW_REG (0x0180)



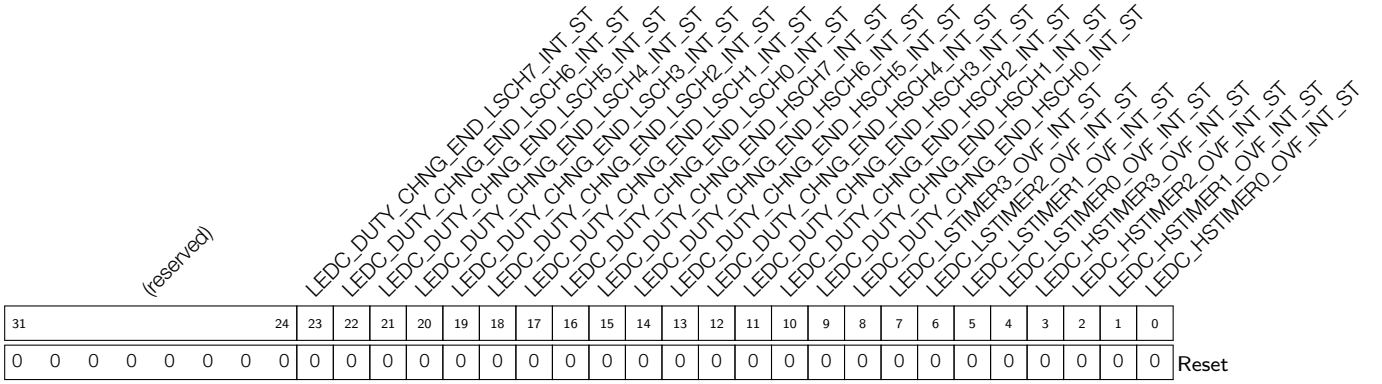
LEDC_DUTY_CHNG_END_LSCH*n*_INT_RAW LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断的原始中断状态位。(只读)

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 5.16: LEDC_INT_ST_REG (0x0184)



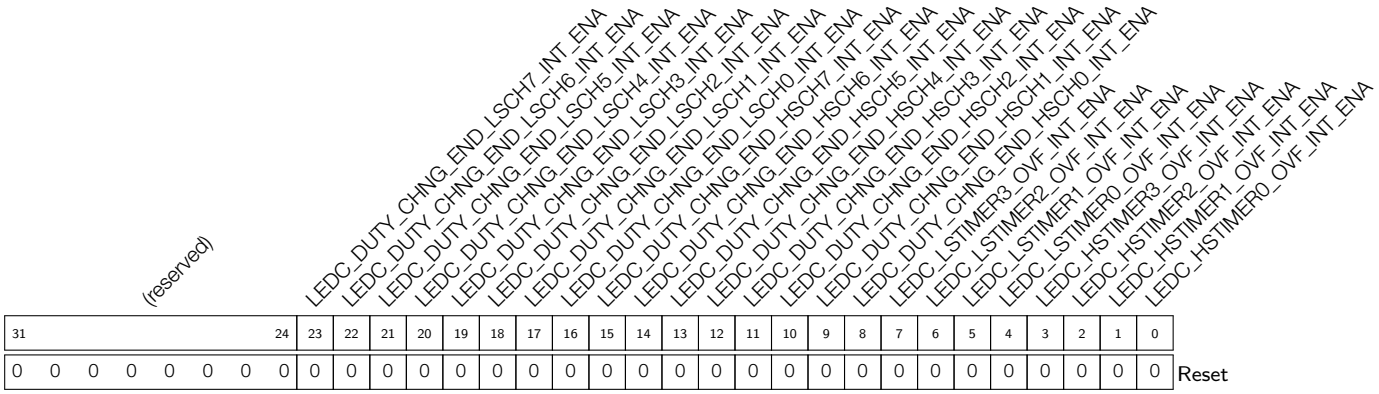
LEDC_DUTY_CHNG_END_LSCH*n*_INT_ST LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断的隐蔽中断状态位。(只读)

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 5.17: LEDC_INT_ENA_REG (0x0188)



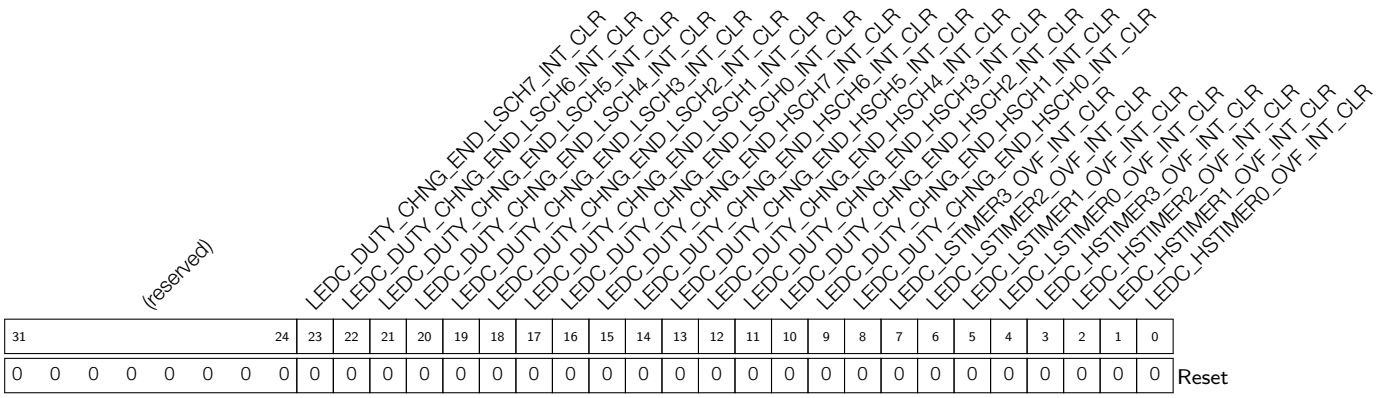
LEDC_DUTY_CHNG_END_LSCH*n*_INT_ENA LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断的使能位。(读 / 写)

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 5.18: LEDC_INT_CLR_REG (0x018C)



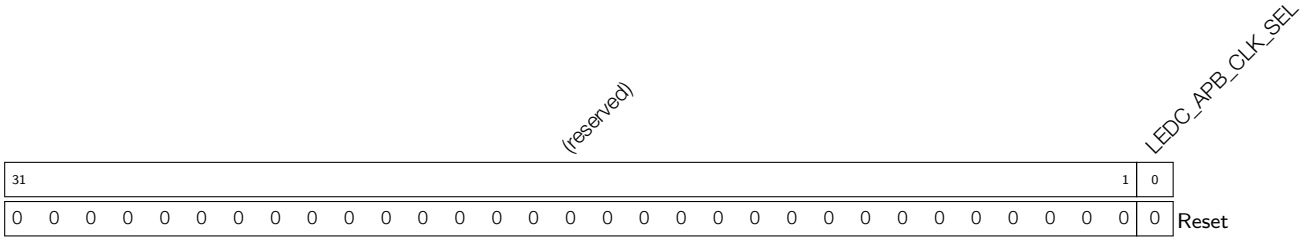
LEDC_DUTY_CHNG_END_LSCH*n*_INT_CLR 用于清除 LEDC_DUTY_CHNG_END_LSCH*n*_INT 中断。(只写)

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 5.19: LEDC_CONF_REG (0x0190)



LEDC_APB_CLK_SEL 用于设置 SLOW_CLK 的频率。(读 / 写)

0：8 MHz；

1：80 MHz。

6. 红外遥控

6.1 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。

6.2 功能描述

6.2.1 RMT 架构

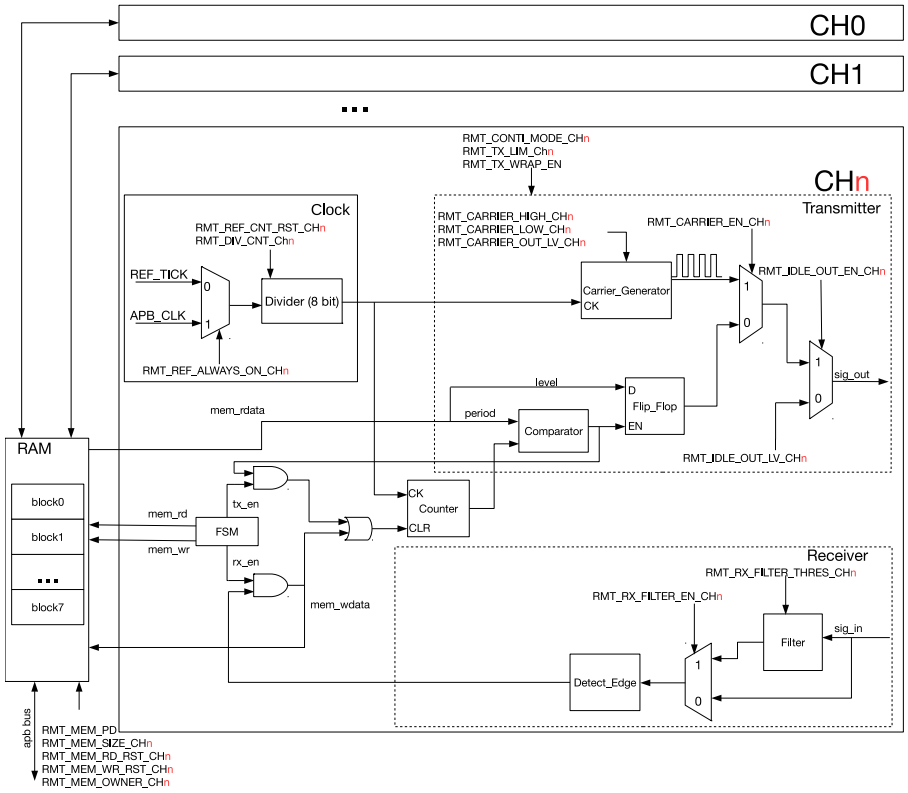


图 14: RMT 架构

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。

6.2.2 RMT RAM

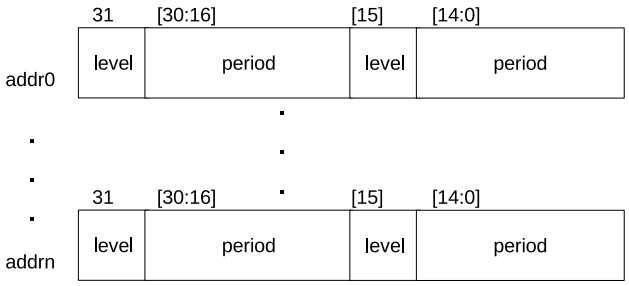


图 15: 数据结构

RAM 中数据结构如图 15 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 持续的时钟周期数。period 为 0 表示结束标志，发射器停止发送数据。

RAM 可以通过 APB 总线进行访问，其起始地址为 RMT 基地址 + 0x800。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_CHn 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:

start_addr_ch*n* = RMT base address + 0x800 + $64 * 4 * n$, and
end_addr_ch*n* = RMT base address + 0x800 + $64 * 4 * n + 64 * 4 *$ RMT_MEM_SIZE_CH*n* mod $512 * 4$

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。

6.2.3 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。

6.2.4 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。此时可以通过配置 RMT_IDLE_OUT_EN_CH*n* 和 RMT_IDLE_OUT_LV_CH*n* 来控制发射器的输出。

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_HIGH_CH*n* 进行设置。

6.2.5 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。

6.2.6 中断

- RMT_CH*n*_TX_THR_EVENT_INT：当发射器发送的数据量与 RMT_CH*n*_TX_LIM_REG 寄存器的内容一致时，即触发此中断。
- 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。

6.3 寄存器列表

名称	描述	地址	访问
配置寄存器			
RMT_CH0CONF0_REG	Channel 0 config register 0	0x3FF56020	读 / 写
RMT_CH0CONF1_REG	Channel 0 config register 1	0x3FF56024	读 / 写
RMT_CH1CONF0_REG	Channel 1 config register 0	0x3FF56028	读 / 写
RMT_CH1CONF1_REG	Channel 1 config register 1	0x3FF5602C	读 / 写
RMT_CH2CONF0_REG	Channel 2 config register 0	0x3FF56030	读 / 写
RMT_CH2CONF1_REG	Channel 2 config register 1	0x3FF56034	读 / 写
RMT_CH3CONF0_REG	Channel 3 config register 0	0x3FF56038	读 / 写
RMT_CH3CONF1_REG	Channel 3 config register 1	0x3FF5603C	读 / 写
RMT_CH4CONF0_REG	Channel 4 config register 0	0x3FF56040	读 / 写
RMT_CH4CONF1_REG	Channel 4 config register 1	0x3FF56044	读 / 写
RMT_CH5CONF0_REG	Channel 5 config register 0	0x3FF56048	读 / 写
RMT_CH5CONF1_REG	Channel 5 config register 1	0x3FF5604C	读 / 写
RMT_CH6CONF0_REG	Channel 6 config register 0	0x3FF56050	读 / 写
RMT_CH6CONF1_REG	Channel 6 config register 1	0x3FF56054	读 / 写
RMT_CH7CONF0_REG	Channel 7 config register 0	0x3FF56058	读 / 写
RMT_CH7CONF1_REG	Channel 7 config register 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	Channel 0 duty cycle configuration register	0x3FF560B0	读 / 写

名称	描述	地址	访问
RMT_CH1CARRIER_DUTY_REG	Channel 1 duty cycle configuration register	0x3FF560B4	读 / 写
RMT_CH2CARRIER_DUTY_REG	Channel 2 duty cycle configuration register	0x3FF560B8	读 / 写
RMT_CH3CARRIER_DUTY_REG	Channel 3 duty cycle configuration register	0x3FF560BC	读 / 写
RMT_CH4CARRIER_DUTY_REG	Channel 4 duty cycle configuration register	0x3FF560C0	读 / 写
RMT_CH5CARRIER_DUTY_REG	Channel 5 duty cycle configuration register	0x3FF560C4	读 / 写
RMT_CH6CARRIER_DUTY_REG	Channel 6 duty cycle configuration register	0x3FF560C8	读 / 写
RMT_CH7CARRIER_DUTY_REG	Channel 7 duty cycle configuration register	0x3FF560CC	读 / 写
发送事件配置寄存器			
RMT_CH0_TX_LIM_REG	Channel 0 Tx event configuration register	0x3FF560D0	读 / 写
RMT_CH1_TX_LIM_REG	Channel 1 Tx event configuration register	0x3FF560D4	读 / 写
RMT_CH2_TX_LIM_REG	Channel 2 Tx event configuration register	0x3FF560D8	读 / 写
RMT_CH3_TX_LIM_REG	Channel 3 Tx event configuration register	0x3FF560DC	读 / 写
RMT_CH4_TX_LIM_REG	Channel 4 Tx event configuration register	0x3FF560E0	读 / 写
RMT_CH5_TX_LIM_REG	Channel 5 Tx event configuration register	0x3FF560E4	读 / 写
RMT_CH6_TX_LIM_REG	Channel 6 Tx event configuration register	0x3FF560E8	读 / 写
RMT_CH7_TX_LIM_REG	Channel 7 Tx event configuration register	0x3FF560EC	读 / 写
其它寄存器			
RMT_APB_CONF_REG	RMT-wide configuration register	0x3FF560F0	读 / 写

6.4 寄存器

Register 6.1: RMT_CH*n*CONF0_REG (*n*: 0-7) (0x0058+8n*)**

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。1: 载波加载在低电平输出信号上；0: 载波加载在高电平输出信号上。（读 / 写）

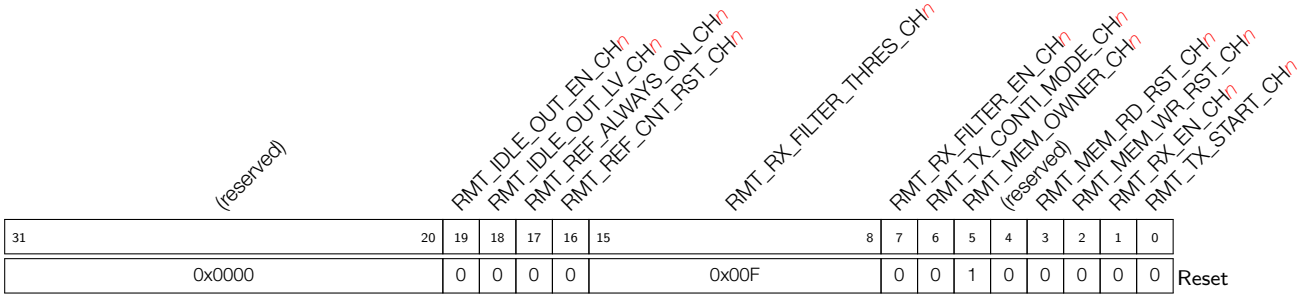
RMT_CARRIER_EN_CH*n* 通道 *n* 的载波调制使能位。1：使能载波调制；0: 关闭载波调制。（读 / 写）

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 6.2: RMT_CH*n*CONF1_REG (*n*: 0-7) (0x005c+8n*)**



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)

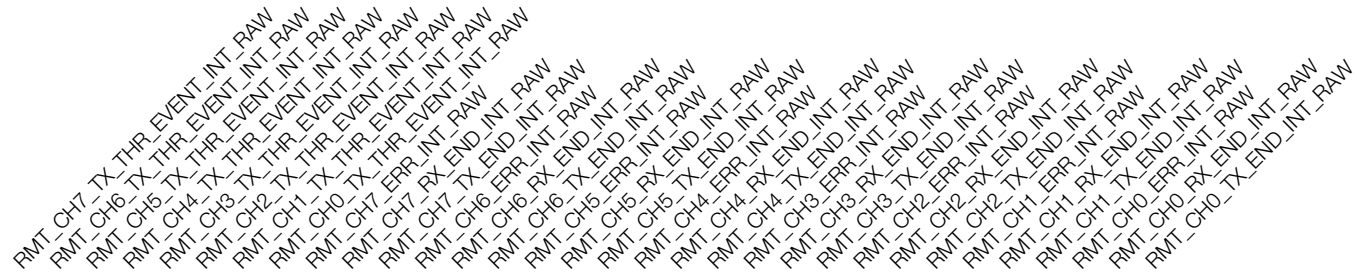
RMT_MEM_RD_RST_CH*n* 用于复位通道 *n* 的发射器的 RAM 读取地址。(读 / 写)

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 6.3: RMT_INT_RAW_REG (0x00a0)



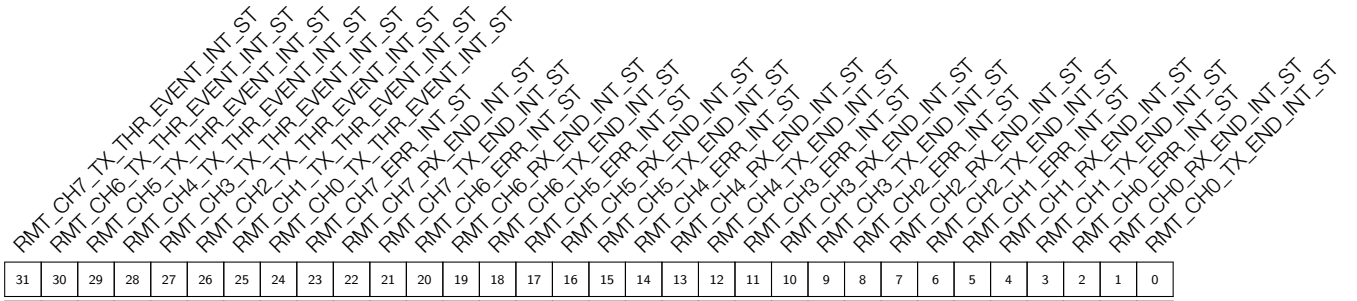
RMT_CH*n*_TX_THR_EVENT_INT_RAW RMT_CH*n*_TX_THR_EVENT_INT 中断的原始中断状态位。(只读)

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 6.4: RMT_INT_ST_REG (0x00a4)



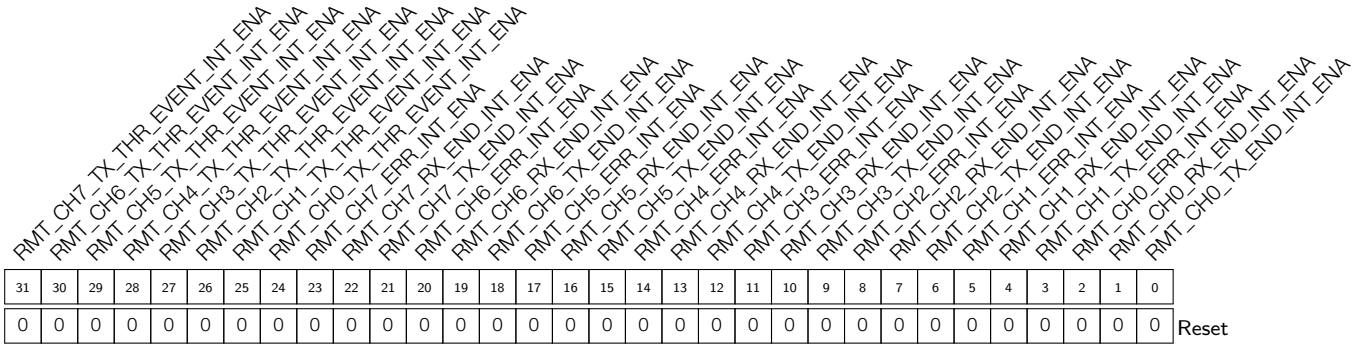
RMT_CH*n*_TX_THR_EVENT_INT_ST RMT_CH*n*_TX_THR_EVENT_INT 中断的隐蔽中断状态位。(只读)

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 6.5: RMT_INT_ENA_REG (0x00a8)



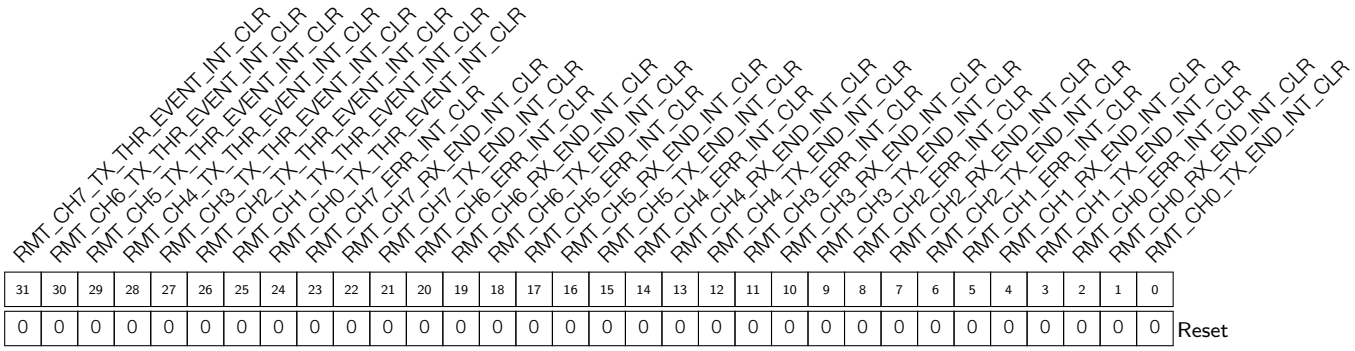
RMT_CH*n*_TX_THR_EVENT_INT_ENA RMT_CH*n*_TX_THR_EVENT_INT 中断的使能位。(读 / 写)

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 6.6: RMT_INT_CLR_REG (0x00ac)



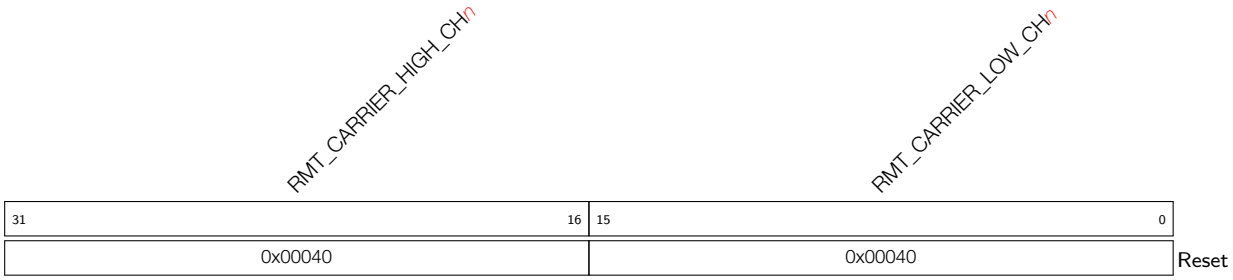
RMT_CH*n*_TX_THR_EVENT_INT_CLR 用于清除 RMT_CH*n*_TX_THR_EVENT_INT 中断。(只写)

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)

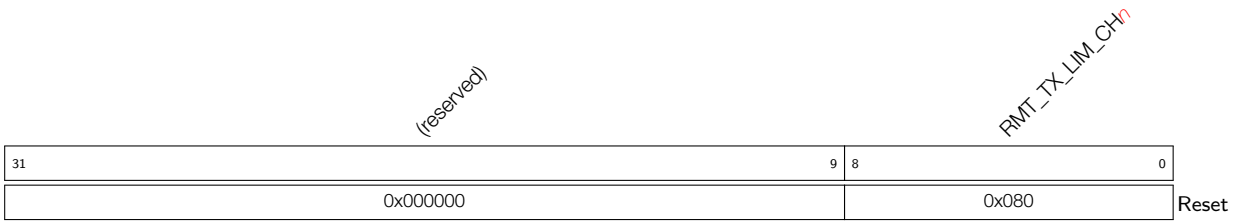
Register 6.7: RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-7) (0x00cc+4n*)**



RMT_CARRIER_HIGH_CH*n* 用于配置通道 *n* 的载波的高电平时钟周期。(读 / 写)

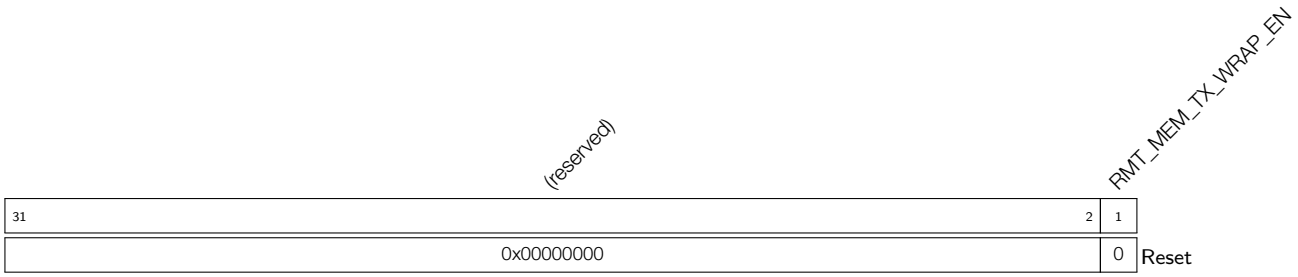
RMT_CARRIER_LOW_CH*n* 用于配置通道 *n* 的载波的低电平时钟周期。(读 / 写)

Register 6.8: RMT_CH*n*_TX_LIM_REG (*n*: 0-7) (0x00ec+4n*)**



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 6.9: 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)

7. PULSE_CNT

7.1 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*。

7.2 功能描述

7.2.1 架构图

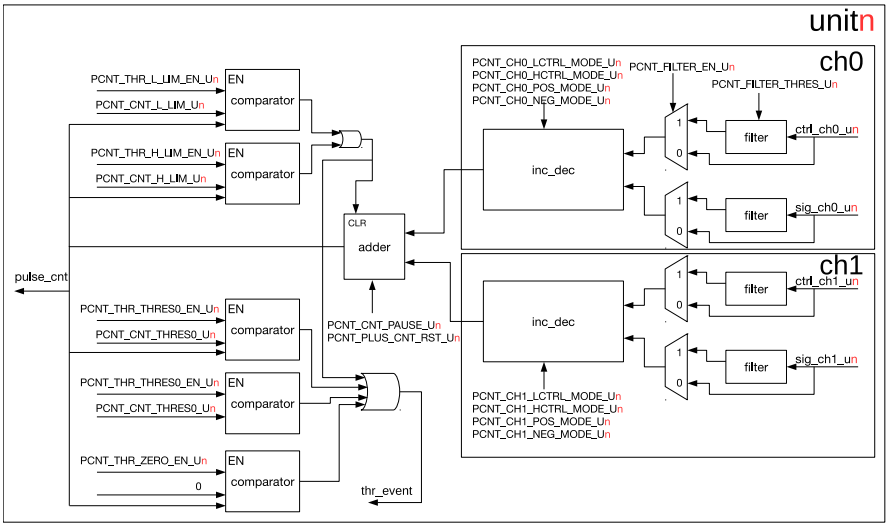


图 16: PULSE_CNT 单元基本架构图

图 16 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仪进行比较，可以产生中断。

7.2.2 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元始纸，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_PULSE_CNT_RST_U*n* 实现计数器清零功能。

7.2.3 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_THR_H_LIM_U*n* 时，清空 pulse_cnt。
- 最小计数值。当 PULSE_CNT 小于等于 PCNT_THR_L_LIM_U*n* 时，清空 pulse_cnt。其中 PCNT_THR_L_LIM_U*n* 应设为负数。
- 两个中间阈值。当 PULSE_CNT 等于 PCNT_THR_THRES0_Un 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。

7.2.4 举例

图 17 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* 为底电平时，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_THR_H_LIM_U*n*=5，当 PULSE_CNT 的值递增到 PCNT_THR_H_LIM_U*n*，PULSE_CNT 返回到 0。

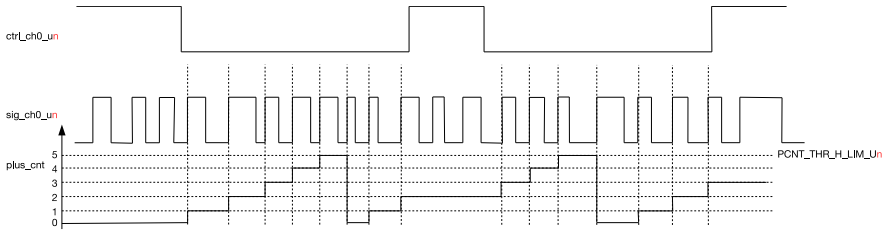


图 17: PULSE_CNT 递增计数图

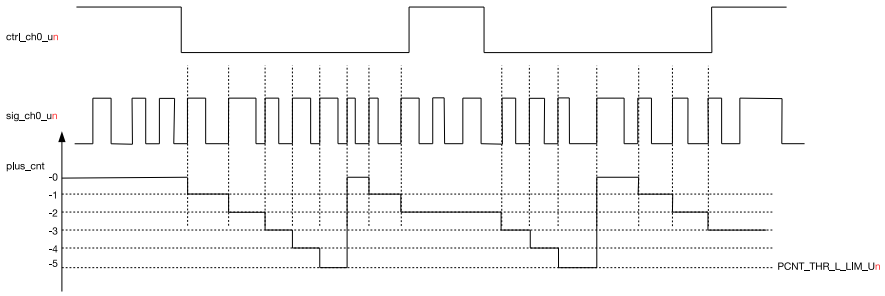


图 18: PULSE_CNT 递减计数图

图 18 为仅仅使用通道 0 进行递减计数的示意图，图 18 中通道 0 的配置与图 17 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_THR_H_LIM_U*n*=-5，当 PULSE_CNT 的值递减到 PCNT_THR_H_LIM_U*n*，plus_cnt 返回到 0。

7.2.5 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。

7.3 寄存器列表

名称	描述	地址	访问
配置寄存器			
PCNT_U0_CONF0_REG	Configuration register 0 for unit 0	0x3FF57000	读 / 写
PCNT_U1_CONF0_REG	Configuration register 0 for unit 1	0x3FF5700C	读 / 写
PCNT_U2_CONF0_REG	Configuration register 0 for unit 2	0x3FF57018	读 / 写
PCNT_U3_CONF0_REG	Configuration register 0 for unit 3	0x3FF57024	读 / 写
PCNT_U4_CONF0_REG	Configuration register 0 for unit 4	0x3FF57030	读 / 写
PCNT_U5_CONF0_REG	Configuration register 0 for unit 5	0x3FF5703C	读 / 写
PCNT_U6_CONF0_REG	Configuration register 0 for unit 6	0x3FF57048	读 / 写
PCNT_U7_CONF0_REG	Configuration register 0 for unit 7	0x3FF57054	读 / 写
PCNT_U0_CONF1_REG	Configuration register 1 for unit 0	0x3FF57004	读 / 写
PCNT_U1_CONF1_REG	Configuration register 1 for unit 1	0x3FF57010	读 / 写
PCNT_U2_CONF1_REG	Configuration register 1 for unit 2	0x3FF5701C	读 / 写
PCNT_U3_CONF1_REG	Configuration register 1 for unit 3	0x3FF57028	读 / 写
PCNT_U4_CONF1_REG	Configuration register 1 for unit 4	0x3FF57034	读 / 写

名称	描述	地址	访问
PCNT_U5_CONF1_REG	Configuration register 1 for unit 5	0x3FF57040	读 / 写
PCNT_U6_CONF1_REG	Configuration register 1 for unit 6	0x3FF5704C	读 / 写
PCNT_U7_CONF1_REG	Configuration register 1 for unit 7	0x3FF57058	读 / 写
PCNT_U0_CONF2_REG	Configuration register 2 for unit 0	0x3FF57008	读 / 写
PCNT_U1_CONF2_REG	Configuration register 2 for unit 1	0x3FF57014	读 / 写
PCNT_U2_CONF2_REG	Configuration register 2 for unit 2	0x3FF57020	读 / 写
PCNT_U3_CONF2_REG	Configuration register 2 for unit 3	0x3FF5702C	读 / 写
PCNT_U4_CONF2_REG	Configuration register 2 for unit 4	0x3FF57038	读 / 写
PCNT_U5_CONF2_REG	Configuration register 2 for unit 5	0x3FF57044	读 / 写
PCNT_U6_CONF2_REG	Configuration register 2 for unit 6	0x3FF57050	读 / 写
PCNT_U7_CONF2_REG	Configuration register 2 for unit 7	0x3FF5705C	读 / 写
计数器值			
PCNT_U0_CNT_REG	Counter value for unit 0	0x3FF57060	只读
PCNT_U1_CNT_REG	Counter value for unit 1	0x3FF57064	只读
PCNT_U2_CNT_REG	Counter value for unit 2	0x3FF57068	只读
PCNT_U3_CNT_REG	Counter value for unit 3	0x3FF5706C	只读
PCNT_U4_CNT_REG	Counter value for unit 4	0x3FF57070	只读
PCNT_U5_CNT_REG	Counter value for unit 5	0x3FF57074	只读
PCNT_U6_CNT_REG	Counter value for unit 6	0x3FF57078	只读
PCNT_U7_CNT_REG	Counter value for unit 7	0x3FF5707C	只读
控制寄存器			
PCNT_CTRL_REG	Control register for all counters	0x3FF570B0	读 / 写
中断寄存器			
PCNT_INT_RAW_REG	Raw interrupt status	0x3FF57080	只读
PCNT_INT_ST_REG	Masked interrupt status	0x3FF57084	只读
PCNT_INT_ENA_REG	Interrupt enable bits	0x3FF57088	读 / 写
PCNT_INT_CLR_REG	Interrupt clear bits	0x3FF5708C	只写

7.4 寄存器

Register 7.1: PCNT_U*n*_CONF0_REG (*n*: 0-7) (0x0+0x0C*n*)

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_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_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* 为单位 *n* 的阈值 1 比较器的使能位。(读 / 写)

PCNT_THR_THRES0_EN_U*n* 为单位 *n* 的阈值 0 比较器的使能位。(读 / 写)

PCNT_THR_L_LIM_EN_U*n* 为单位 *n* 的阈值 1 比较器的使能位。(读 / 写)

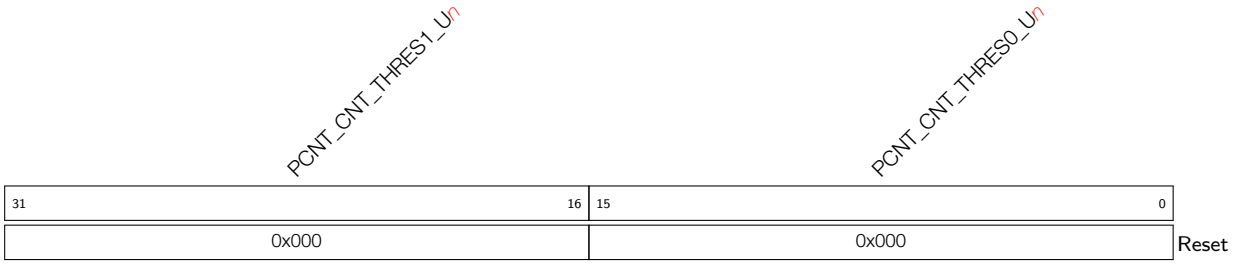
PCNT_THR_H_LIM_EN_U*n* 为单位 *n* 的 thr_h_lim 比较器的使能位。(读 / 写)

PCNT_THR_ZERO_EN_U*n* 为单位 *n* 的过零比较器的使能位。(读 / 写)

PCNT_FILTER_EN_U*n* 为单位 *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)

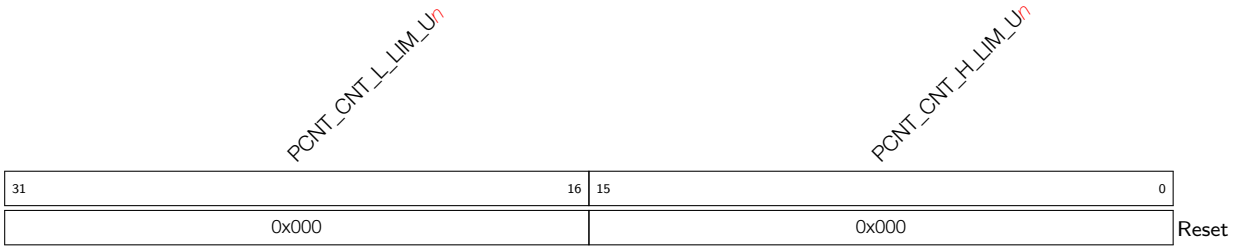
Register 7.2: PCNT_U*n*_CONF1_REG (*n*: 0-7) (0x4+0x0Cn*)**



PCNT_CNT_THRES1_U*n* 用于配置单位 *n* 的阈值 1 的值。(读 / 写)

PCNT_CNT_THRES0_U*n* 用于配置单位 *n* 的阈值 0 的值。(读 / 写)

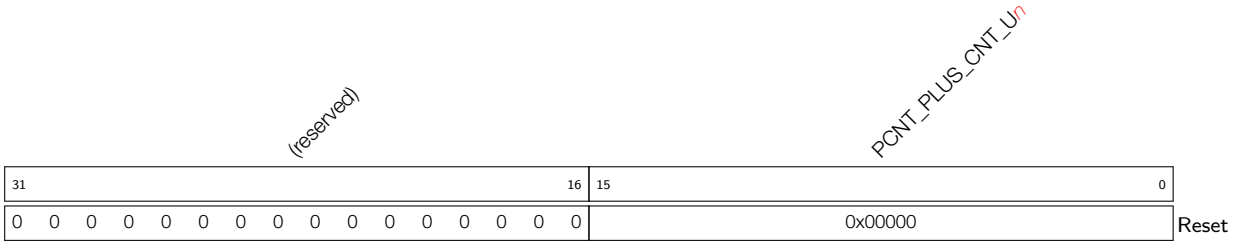
Register 7.3: PCNT_U*n*_CONF2_REG (*n*: 0-7) (0x8+0x0Cn*)**



PCNT_CNT_L_LIM_U*n* 用于配置单位 *n* 的计数器的最小值。(读 / 写)

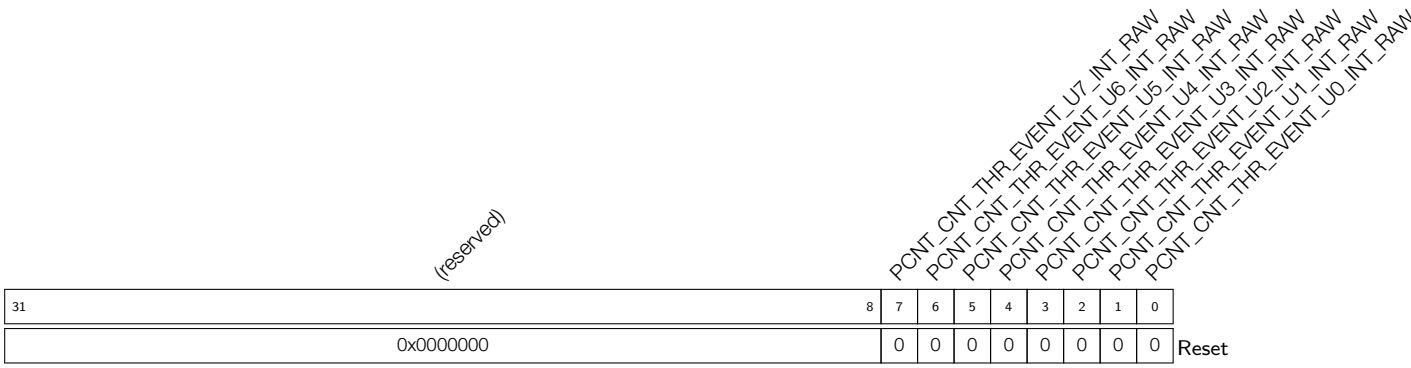
PCNT_CNT_H_LIM_U*n* 用于配置单位 *n* 的计数器的最大值。(读 / 写)

Register 7.4: PCNT_U*n*_CNT_REG (*n*: 0-7) (0x28+0x0Cn*)**



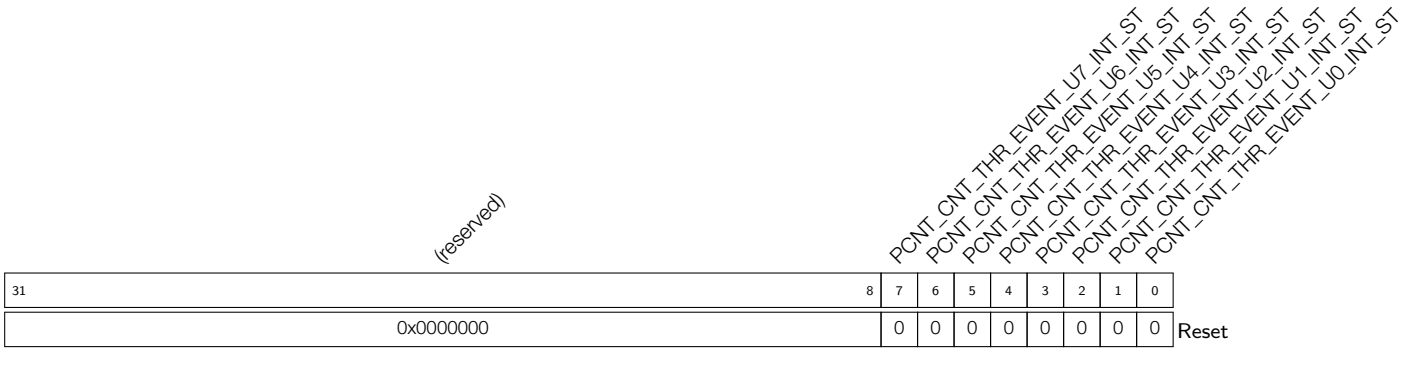
PCNT_PLUS_CNT_U*n* 该寄存器存储着单位 *n* 的当前计数器的值，可供软件读取。(只读)

Register 7.5: PCNT_INT_RAW_REG (0x0080)



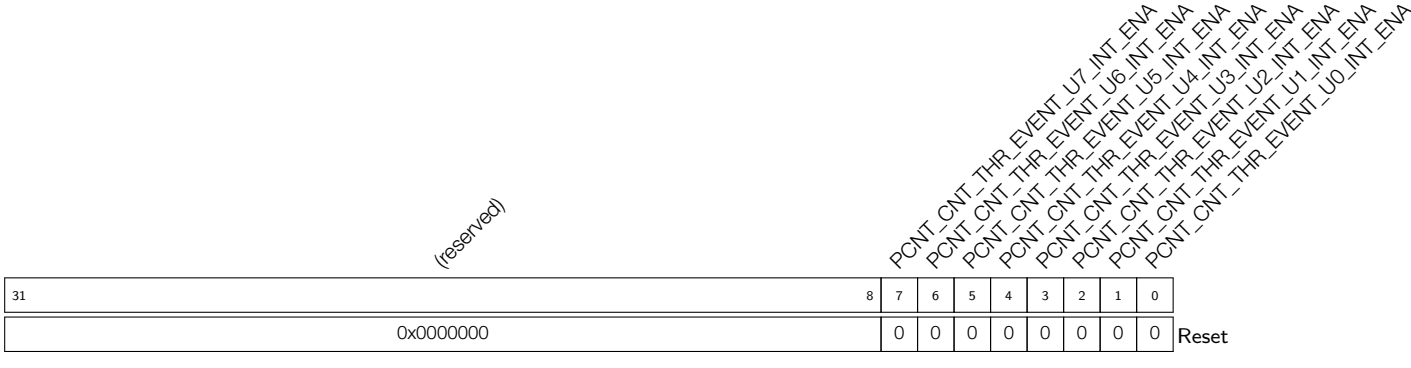
PCNT_CNT_THR_EVENT_U*n*_INT_RAW PCNT_CNT_THR_EVENT_U*n*_INT 中断的原始中断状态位。(只读)

Register 7.6: PCNT_INT_ST_REG (0x0084)



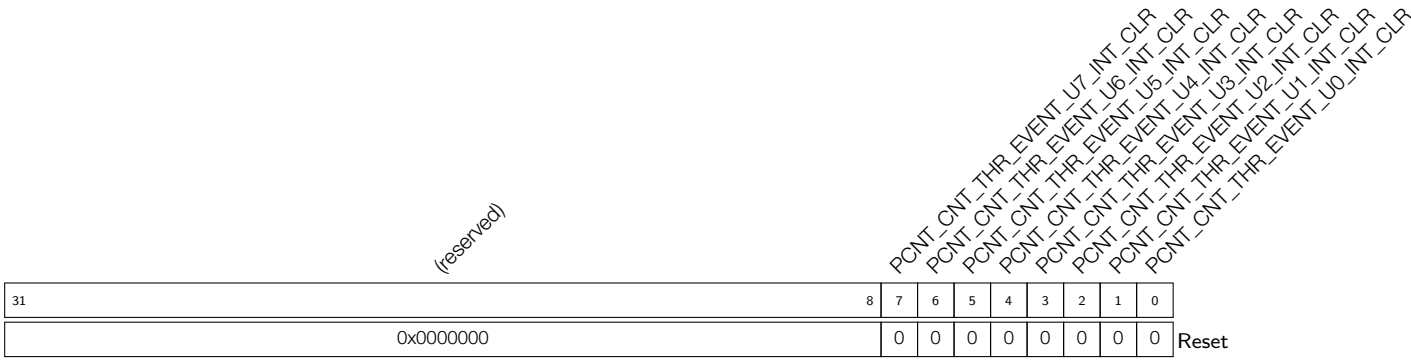
PCNT_CNT_THR_EVENT_U*n*_INT_ST PCNT_CNT_THR_EVENT_U*n*_INT 中断的屏蔽中断状态位。(只读)

Register 7.7: PCNT_INT_ENA_REG (0x0088)



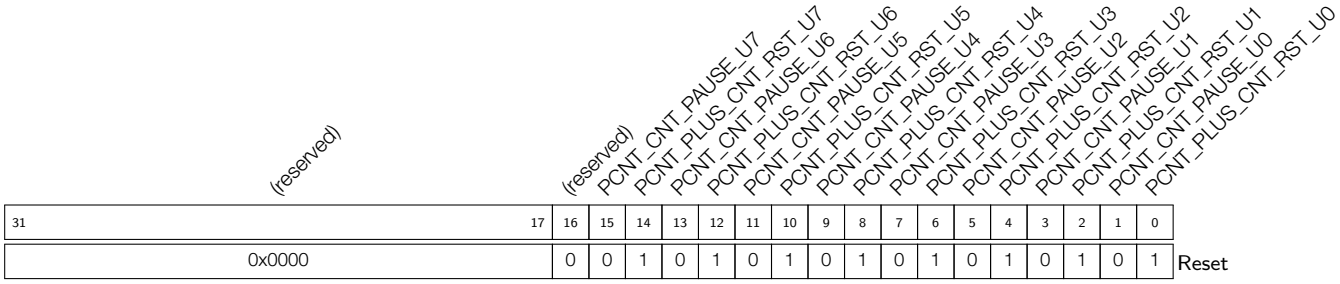
PCNT_CNT_THR_EVENT_U*n*_INT_ENA PCNT_CNT_THR_EVENT_U*n*_INT 中断的使能位。(读 / 写)

Register 7.8: PCNT_INT_CLR_REG (0x008c)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR 用于清除 PCNT_CNT_THR_EVENT_U*n*_INT 中断。(只写)

Register 7.9: PCNT_CTRL_REG (0x00b0)



PCNT_CNT_PAUSE_U*n* 用于暂停单位 *n* 的计数器。(读 / 写)

PCNT_PLUS_CNT_RST_U*n* 用于清零单位 *n* 的计数器。(读 / 写)

8. 64-bit 定时器

8.1 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

8.2 功能描述

8.2.1 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。

8.2.2 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。

8.2.3 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。

8.2.4 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。

8.2.5 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。

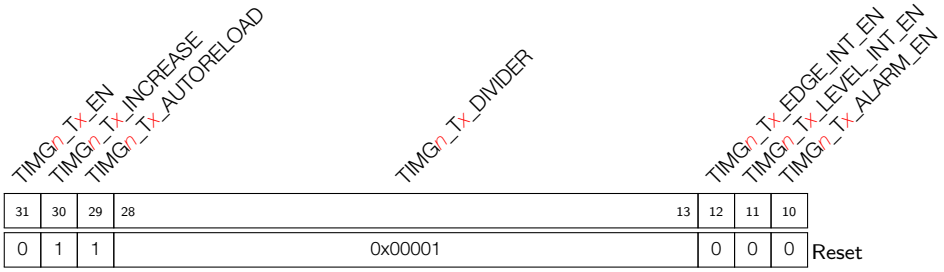
8.3 寄存器列表

名称	描述	TIMG0	TIMG1	访问
定时器 0 配置和控制寄存器				
TIMG*n*_T0CONFIG_REG	Timer 0 configuration register	0x3FF5F000	0x3FF60000	读/写
TIMG*n*_T0LO_REG	Timer 0 current value, low 32 bits	0x3FF5F004	0x3FF60004	只读
TIMG*n*_T0HI_REG	Timer 0 current value, high 32 bits	0x3FF5F008	0x3FF60008	只读
TIMG*n*_T0UPDATE_REG	Write to copy current timer value to TIMG*n*_T0_(LO/HI)_REG	0x3FF5F00C	0x3FF6000C	只写
TIMG*n*_T0ALARMLO_REG	Timer 0 alarm value, low 32 bits	0x3FF5F010	0x3FF60010	读/写
TIMG*n*_T0ALARMHI_REG	Timer 0 alarm value, high 32 bits	0x3FF5F014	0x3FF60014	读/写
TIMG*n*_T0LOADLO_REG	Timer 0 reload value, low 32 bits	0x3FF5F018	0x3FF60018	读/写
TIMG*n*_T0LOAD_REG	Write to reload timer from TIMG*n*_T0_(LOADLOLOADHI)_REG	0x3FF5F020	0x3FF60020	只写
定时器 1 配置和控制寄存器				
TIMG*n*_T1CONFIG_REG	Timer 1 configuration register	0x3FF5F024	0x3FF60024	读/写
TIMG*n*_T1LO_REG	Timer 1 current value, low 32 bits	0x3FF5F028	0x3FF60028	只读
TIMG*n*_T1HI_REG	Timer 1 current value, high 32 bits	0x3FF5F02C	0x3FF6002C	只读

名称	描述	TIMG0	TIMG1	访问
TIMG*n*_T1UPDATE_REG	Write to copy current timer value to TIMG*n*_T1_(LO/HI)_REG	0x3FF5F030	0x3FF60030	只写
TIMG*n*_T1ALARMLO_REG	Timer 1 alarm value, low 32 bits	0x3FF5F034	0x3FF60034	读/写
TIMG*n*_T1ALARMHI_REG	Timer 1 alarm value, high 32 bits	0x3FF5F038	0x3FF60038	读/写
TIMG*n*_T1LOADLO_REG	Timer 1 reload value, low 32 bits	0x3FF5F03C	0x3FF6003C	读/写
TIMG*n*_T1LOAD_REG	Write to reload timer from TIMG*n*_T1_(LOADLOLOADHI)_REG	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	Watchdog timer configuration register	0x3FF5F048	0x3FF60048	读/写
TIMG*n*_T*x*_WDTCONFIG1_REG	Watchdog timer prescaler register	0x3FF5F04C	0x3FF6004C	读/写
TIMG*n*_T*x*_WDTCONFIG2_REG	Watchdog timer stage 0 timeout value	0x3FF5F050	0x3FF60050	读/写
TIMG*n*_T*x*_WDTCONFIG3_REG	Watchdog timer stage 1 timeout value	0x3FF5F054	0x3FF60054	读/写
TIMG*n*_T*x*_WDTCONFIG4_REG	Watchdog timer stage 2 timeout value	0x3FF5F058	0x3FF60058	读/写
TIMG*n*_T*x*_WDTCONFIG5_REG	Watchdog timer stage 3 timeout value	0x3FF5F05C	0x3FF6005C	读/写
TIMG*n*_T*x*_WDTFEED_REG	Write to feed the watchdog timer	0x3FF5F060	0x3FF60060	只写
TIMG*n*_T*x*_WDTWPROTECT_REG	Watchdog write protect register	0x3FF5F064	0x3FF60064	读/写
中断寄存器				
TIMG*n*_T*x*_INT_RAW_REG	Raw interrupt status	0x3FF5F09C	0x3FF6009C	只读
TIMG*n*_T*x*_INT_ST_REG	Masked interrupt status	0x3FF5F0A0	0x3FF600A0	只读
TIMG*n*_T*x*_INT_ENA_REG	Interrupt enable bits	0x3FF5F098	0x3FF60098	读/写
TIMG*n*_T*x*_INT_CLR_REG	Interrupt clear bits	0x3FF5F0A4	0x3FF600A4	只写

8.4 寄存器

Register 8.1: TIMGn_TxCONFIG_REG (x: 0-1) (0x0+0x24*x)



TIMGn_Tx_EN 置 1 后，定时器 x 时基计数器使能。(读/写)

TIMGn_Tx_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)

TIMGn_Tx_AUTORELOAD 置 1 后，定时器 x 报警时自动重新加载使能。(读/写)

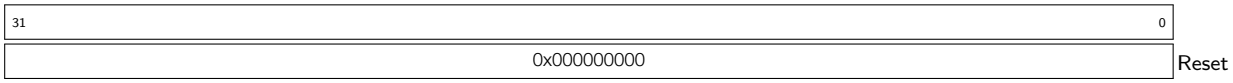
TIMGn_Tx_DIVIDER 计时器 x 时钟 (Tx_clk) 的预分频器值。(读/写)

TIMGn_Tx_EDGE_INT_EN 置 1 后，报警会产生一个边沿触发中断。(读/写)

TIMGn_Tx_LEVEL_INT_EN 置 1 后，报警会产生一个电平触发中断。(读/写)

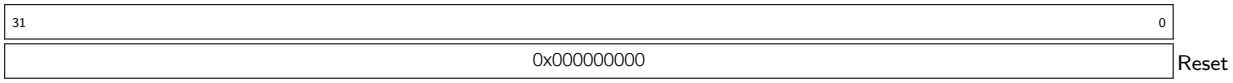
TIMGn_Tx_ALARM_EN 置 1 后，报警使能。(读/写)

Register 8.2: TIMGn_TxLO_REG (x: 0-1) (0x4+0x24*x)



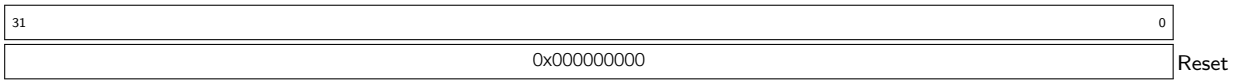
TIMGn_TxLO_REG 在 TIMGn_TxUPDATE_REG 上写值后，定时器 x 时基计数器的低 32 位可以被读取。(只读)

Register 8.3: TIMGn_TxHI_REG (x: 0-1) (0x8+0x24*x)



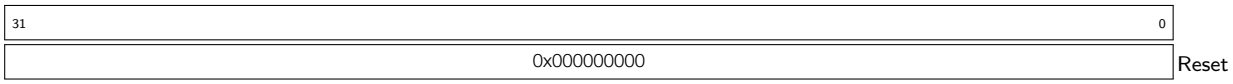
TIMGn_TxHI_REG 在 TIMGn_TxUPDATE_REG 上写值后，定时器 x 时基计数器的高 32 位可以被读取。(只读)

Register 8.4: TIMG*n*_T*x*UPDATE_REG (*x*: 0-1) (0xC+0x24*x)



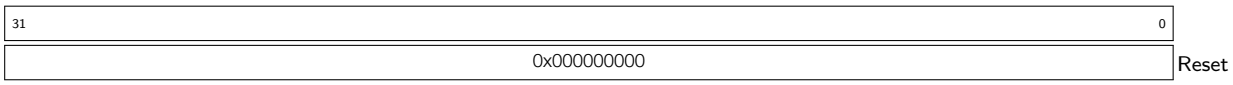
TIMG*n*_T*x*UPDATE_REG 写任何值触发定时器 *x* 时基计数器值更新（定时器 *x* 当前值会被存储到以上寄存器）。（只写）

Register 8.5: TIMG*n*_T*x*ALARMLO_REG (*x*: 0-1) (0x10+0x24*x)



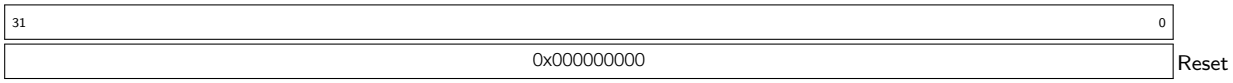
TIMG*n*_T*x*ALARMLO_REG 定时器 *x* 时基计数器触发警报值的低 32 位。（读/写）

Register 8.6: TIMG*n*_T*x*ALARMHI_REG (*x*: 0-1) (0x14+0x24*x)



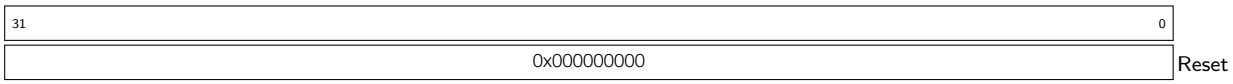
TIMG*n*_T*x*ALARMHI_REG 定时器 *x* 时基计数器触发警报值的高 32 位。(读/写)

Register 8.7: TIMG*n*_T*x*LOADLO_REG (*x*: 0-1) (0x18+0x24*x)



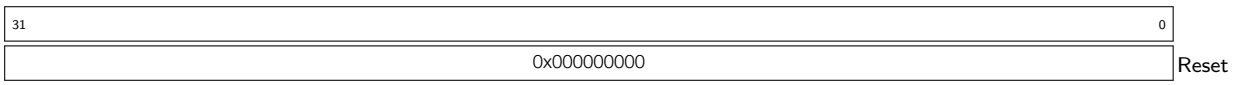
TIMG*n*_T*x*LOADLO_REG 定时器 *x* 时基计数器重新加载的低 32-bit 值。(读/写)

Register 8.8: TIMG*n*_T*x*LOADHI_REG (*x*: 0-1) (0x1C+0x24*x)



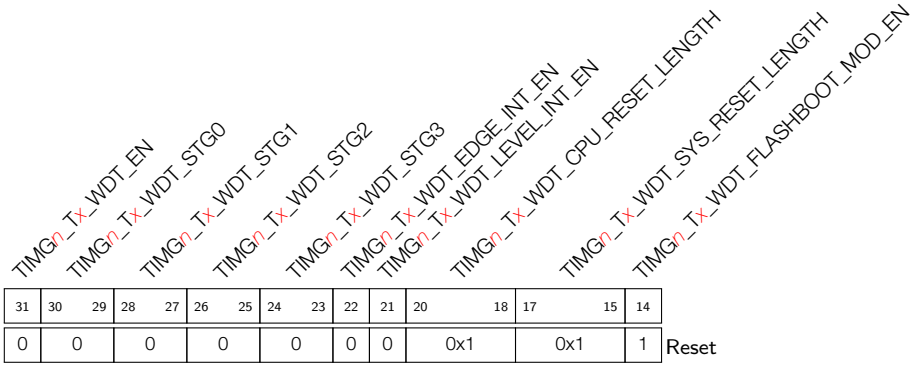
TIMG*n*_T*x*LOADHI_REG 定时器 *x* 时基计数器重新加载的高 32-bit 值。（读/写）

Register 8.9: TIMG*n*_T*x*LOAD_REG (*x*: 0-1) (0x20+0x24*x)



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 8.10: TIMG*n*_T*x*_WDTCONFIG0_REG (0x0048)



TIMG*n*_T*x*_WDT_EN 置 1 后, SWDT 使能。（读/写）

TIMG*n*_T*x*S_WDT_STG0 阶段 0 配置。0: 关闭，1: 中断, 2: 复位 CPU, 3: 复位系统。（读/写）

TIMG*n*_T*x*_WDT_STG1 阶段 1 配置。0: 关闭，1: 中断, 2: 复位 CPU, 3: 复位系统。（读/写）

TIMG*n*_T*x*_WDT_STG2 阶段 2 配置。0: 关闭，1: 中断, 2: 复位 CPU, 3: 复位系统。（读/写）

TIMG*n*_T*x*_WDT_STG3 阶段 3 配置。0: 关闭，1: 中断, 2: 复位 CPU, 3: 复位系统。（读/写）

TIMG*n*_T*x*_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）

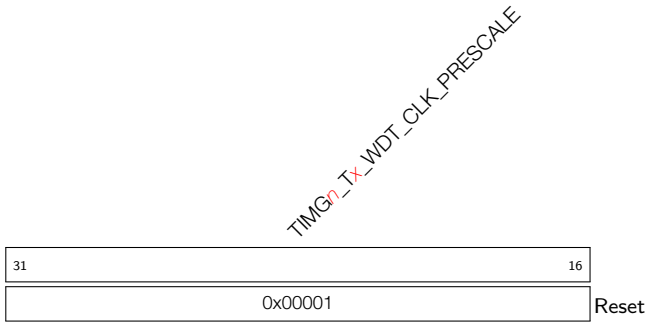
TIMG*n*_T*x*_WDT_LEVEL_INT_EN 置 1 后，如超过设置的阶段中断产生时间，会产生电平触发中断。（读/写）

TIMG*n*_T*x*_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）

TIMG*n*_T*x*_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）

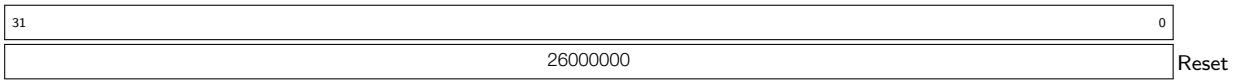
TIMG*n*_T*x*_WDT_FLASHBOOT_MOD_EN 置 1 后, Flash 启动保护使能。（读/写）

Register 8.11: TIMG*n*_T*x*_WDTCONFIG1_REG (0x004c)



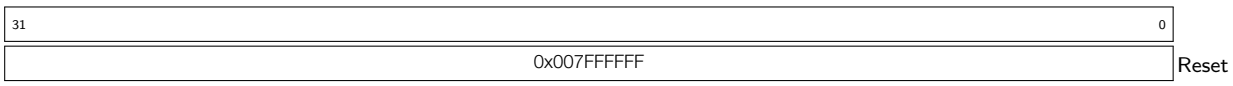
TIMG*n*_T*x*_WDT_CLK_PRESCALE SWDT 时钟预分频器值，分辨率 = 12.5 ns * TIMG*n*_T*x*_WDT_CLK_PRESCALE。（读/写）

Register 8.12: TIMG*n*_T*x*_WDTCONFIG2_REG (0x0050)



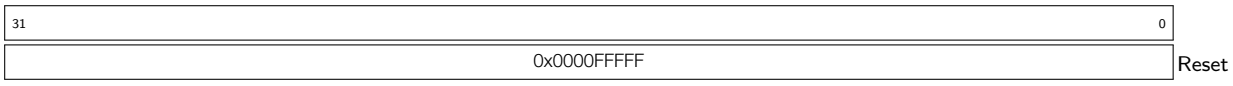
TIMG*n*_T*x*_WDTCONFIG2_REG SWDT 时钟周期中阶段 0 超时时间。(读/写)

Register 8.13: TIMG*n*_T*x*_WDTCONFIG3_REG (0x0054)



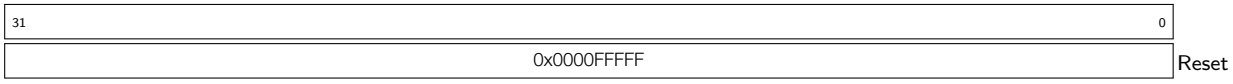
TIMG*n*_T*x*_WDTCONFIG3_REG SWDT 时钟周期中阶段 1 超时时间。(读/写)

Register 8.14: TIMG*n*_T*x*_WDTCONFIG4_REG (0x0058)



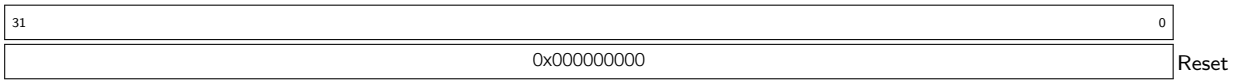
TIMG*n*_T*x*_WDTCONFIG4_REG SWDT 时钟周期中阶段 2 超时时间。(读/写)

Register 8.15: TIMG*n*_T*x*_WDTCONFIG5_REG (0x005c)



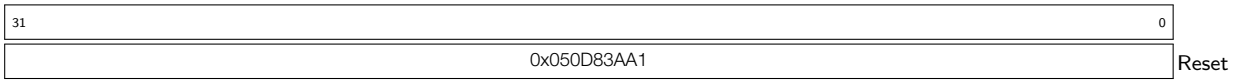
TIMG*n*_T*x*_WDTCONFIG5_REG SWDT 时钟周期中阶段 3 超时时间。(读/写)

Register 8.16: TIMG*n*_T*x*_WDTFEED_REG (0x0060)



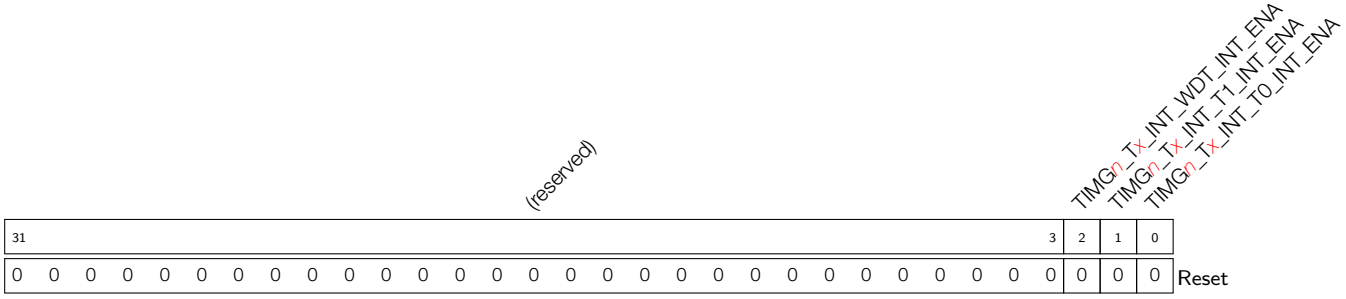
TIMG*n*_T*x*_WDTFEED_REG 写任何值驱动 SWDT。(只写)

Register 8.17: TIMG*n*_T*x*_WDTWPROTECT_REG (0x0064)



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 8.18: TIMG*n*_T*x*_INT_ENA_REG (0x0098)

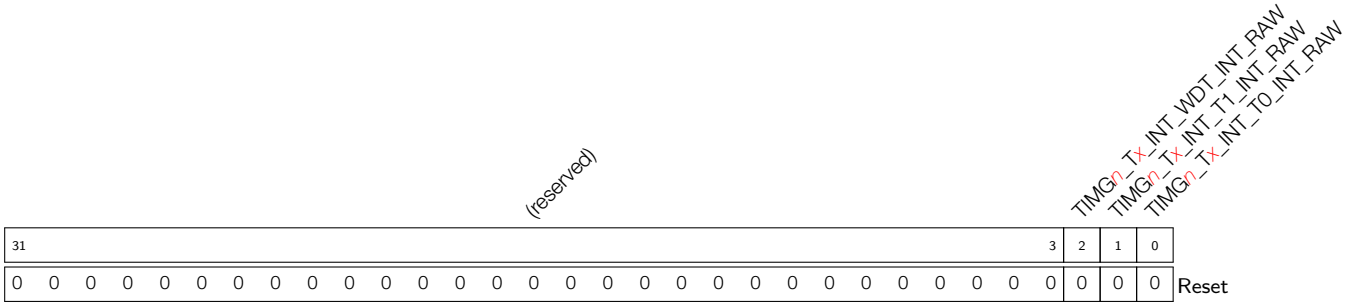


TIMG*n*_T*x*_INT_WDT_INT_ENA TIMG*n*_T*x*_INT_WDT_INT 中断的中断使能位。(读/写)

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 8.19: TIMG*n*_T*x*_INT_RAW_REG (0x009c)

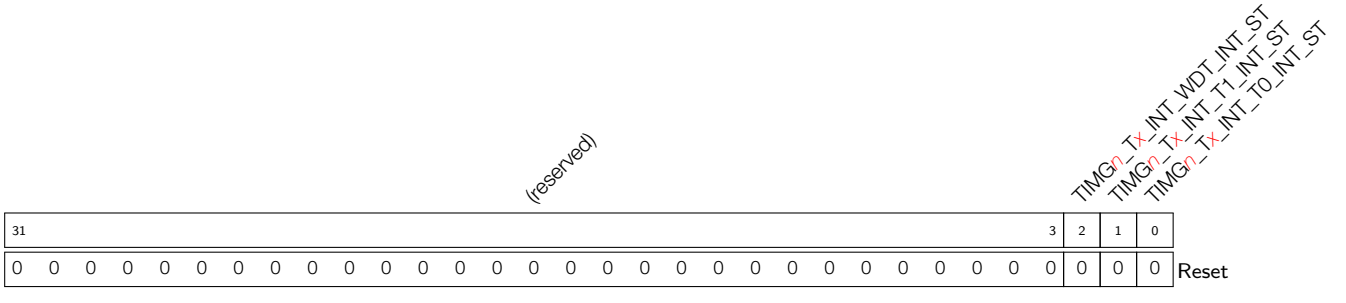


TIMG*n*_T*x*_INT_WDT_INT_RAW TIMG*n*_T*x*_INT_WDT_INT 中断的原始中断状态位。(只读)

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 8.20: TIMG*n*_T*x*_INT_ST_REG (0x00a0)

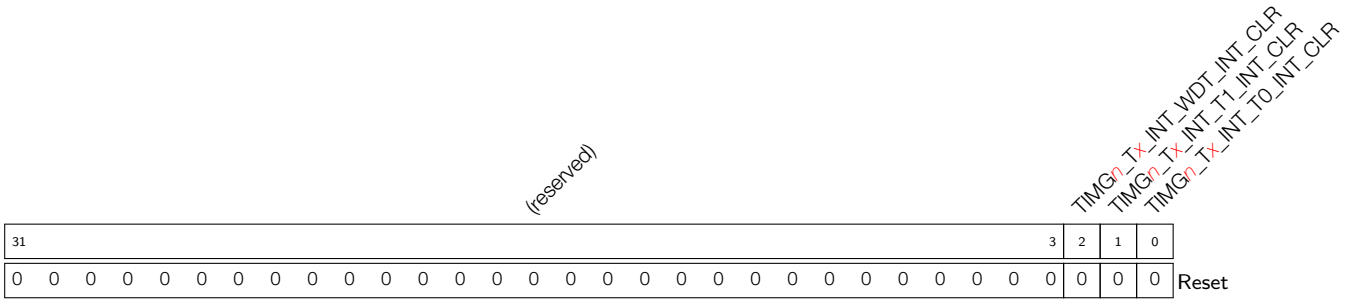


TIMG*n*_T*x*_INT_WDT_INT_ST TIMG*n*_T*x*_INT_WDT_INT 中断的屏蔽中断状态位。(只读)

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 8.21: TIMG*n*_T*x*_INT_CLR_REG (0x00a4)



TIMG*n*_T*x*_INT_WDT_INT_CLR 置 1 该位以清除 TIMG*n*_T*x*_INT_WDT_INT 中断。(只写)

TIMG*n*_T*x*_INT_T1_INT_CLR 置 1 该位以清除 TIMG*n*_T*x*_INT_T1_INT 中断。(只写)

TIMG*n*_T*x*_INT_T0_INT_CLR 置 1 该位以清除 TIMG*n*_T*x*_INT_T0_INT 中断。(只写)

9. 看门狗定时器

9.1 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。

9.2 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

9.3 功能描述

9.3.1 时钟

RWDT 的时钟源是 RTC 慢速时钟，频率通常为 32 KHz。MWDT 时钟源来自于 APB 时钟，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。

9.3.1.1 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。

9.3.1.2 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 TIMERS_WDT_WKEY, RWDT 的寄存器为 RTC_CNTL_WDT_WKEY。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

9.3.1.3 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。

9.3.1.4 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。

10. AES 加速器

10.1 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。

10.2 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

10.3 功能描述

10.3.1 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 27 所示。

表 27: 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

10.3.2 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。

10.3.3 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 29、表 30、表 31。表 29 中的 w[0] ~ w[3]、表 30 中的 w[0] ~ w[5]、表 31 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 28。表 28 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_KEY_*n*_REG 中的每个字节所存放的明文或密文如何构成 State。

表 28: AES 文本字节序

AES_ENDIAN_REG[3]/[5]	AES_ENDIAN_REG[2]/[4]	Plaintext/clphertext				
0	0	State		c		
				0	1	2	3
		r	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	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
			2	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
			3	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	0	State		c		
				0	1	2	3
		r	0	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
			1	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
			2	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
			3	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]
0	0	State		c		
				0	1	2	3
		r	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	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
			2	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
			3	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	0	State		c		
				0	1	2	3
		r	0	AES_KEY_3_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_0_REG[7:0]
			1	AES_KEY_3_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_0_REG[15:8]
			2	AES_KEY_3_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_0_REG[23:16]
			3	AES_KEY_3_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_0_REG[31:24]

表 29: 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	0	[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 30: 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	0	[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 31: 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	0	[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]

10.3.4 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。

10.3.5 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。

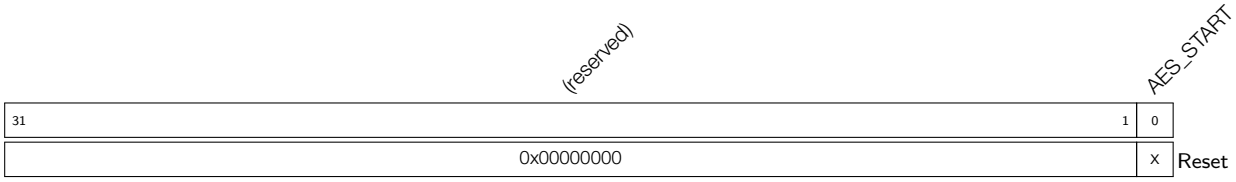
10.4 寄存器列表

名称	描述	地址	访问
配置寄存器			
AES_MODE_REG	Operation mode of the AES Accelerator	0x3FF01008	读 / 写
AES_ENDIAN_REG	Endian-ness configuration register	0x3FF01040	读 / 写
密钥寄存器			
AES_KEY_0_REG	AES key material register 0	0x3FF01010	读 / 写
AES_KEY_1_REG	AES key material register 1	0x3FF01014	读 / 写
AES_KEY_2_REG	AES key material register 2	0x3FF01018	读 / 写
AES_KEY_3_REG	AES key material register 3	0x3FF0101C	读 / 写
AES_KEY_4_REG	AES key material register 4	0x3FF01020	读 / 写
AES_KEY_5_REG	AES key material register 5	0x3FF01024	读 / 写
AES_KEY_6_REG	AES key material register 6	0x3FF01028	读 / 写
AES_KEY_7_REG	AES key material register 7	0x3FF0102C	读 / 写
加密 / 解密数据寄存器			
AES_TEXT_0_REG	AES encrypted/decrypted data register 0	0x3FF01030	读 / 写
AES_TEXT_1_REG	AES encrypted/decrypted data register 1	0x3FF01034	读 / 写
AES_TEXT_2_REG	AES encrypted/decrypted data register 2	0x3FF01038	读 / 写
AES_TEXT_3_REG	AES encrypted/decrypted data register 3	0x3FF0103C	读 / 写

名称	描述	地址	访问
控制 / 状态寄存器			
AES_START_REG	AES operation start control register	0x3FF01000	只写
AES_IDLE_REG	AES idle status register	0x3FF01004	只读

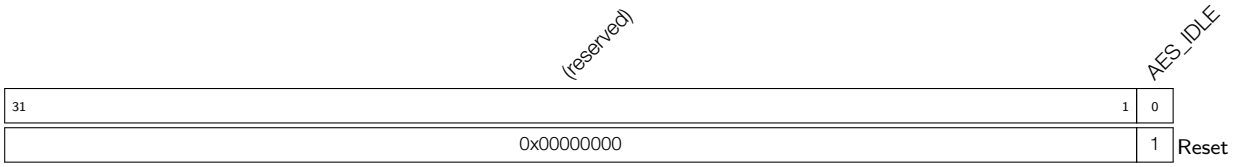
10.5 寄存器

Register 10.1: AES_START_REG (0x000)



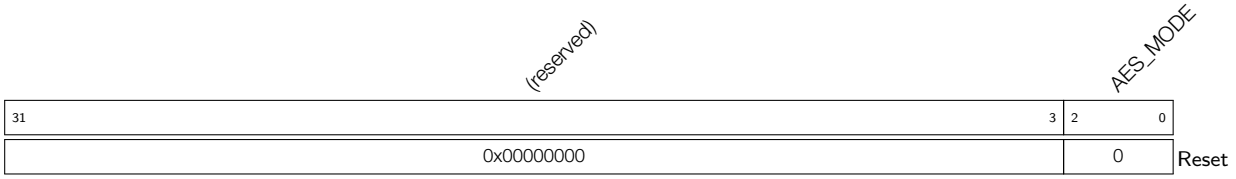
AES_START 写入 1 使能 AES 运算。（只写）

Register 10.2: AES_IDLE_REG (0x004)



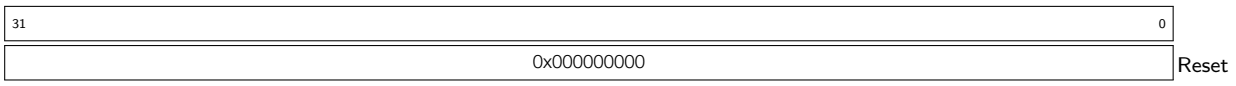
AES_IDLE AES 空闲寄存器。AES 加速器运行时读出 0，空闲时读出 1。（只读）

Register 10.3: AES_MODE_REG (0x008)



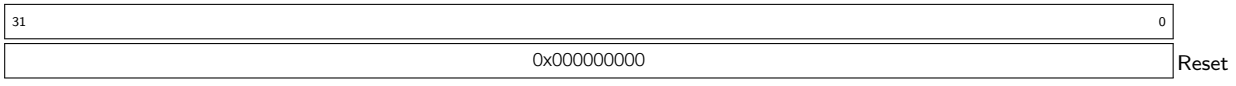
AES_MODE 选择 AES 加速器运算模式。详情请见表 27。（读 / 写）

Register 10.4: AES_KEY_*n*_REG (*n*: 0-7) (0x10+4**n*)



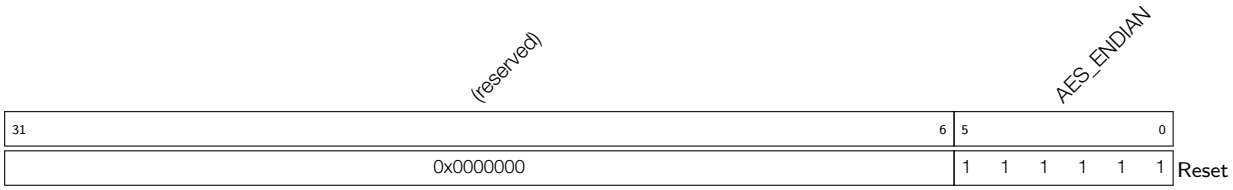
AES_KEY_*n*_REG (*n*: 0-7) AES 密钥寄存器。（读 / 写）

Register 10.5: AES_TEXT_*m*_REG (*m*: 0-3) (0x30+4**m*)



AES_TEXT_*m*_REG (*m*: 0-3) 明文和密文寄存器。（读 / 写）

Register 10.6: AES_ENDIAN_REG (0x040)



AES_ENDIAN 字节序选择寄存器。详情请见表 28。(读 / 写)

11. SHA 加速器

11.1 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。

11.2 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

11.3 功能描述

11.3.1 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……

11.3.2 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……

11.3.3 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。

11.3.4 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。

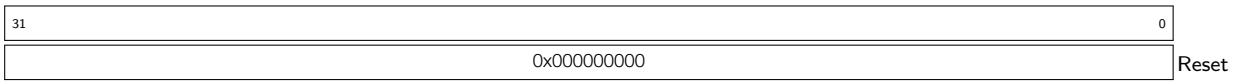
11.4 寄存器列表

名称	描述	地址	访问
加密 / 解密数据寄存器			
SHA_TEXT_0_REG	SHA encrypted/decrypted data register 0	0x3FF03000	读 / 写
SHA_TEXT_1_REG	SHA encrypted/decrypted data register 1	0x3FF03004	读 / 写
SHA_TEXT_2_REG	SHA encrypted/decrypted data register 2	0x3FF03008	读 / 写
SHA_TEXT_3_REG	SHA encrypted/decrypted data register 3	0x3FF0300C	读 / 写
SHA_TEXT_4_REG	SHA encrypted/decrypted data register 4	0x3FF03010	读 / 写
SHA_TEXT_5_REG	SHA encrypted/decrypted data register 5	0x3FF03014	读 / 写
SHA_TEXT_6_REG	SHA encrypted/decrypted data register 6	0x3FF03018	读 / 写
SHA_TEXT_7_REG	SHA encrypted/decrypted data register 7	0x3FF0301C	读 / 写
SHA_TEXT_8_REG	SHA encrypted/decrypted data register 8	0x3FF03020	读 / 写

名称	描述	地址	访问
SHA_TEXT_9_REG	SHA encrypted/decrypted data register 9	0x3FF03024	读 / 写
SHA_TEXT_10_REG	SHA encrypted/decrypted data register 10	0x3FF03028	读 / 写
SHA_TEXT_11_REG	SHA encrypted/decrypted data register 11	0x3FF0302C	读 / 写
SHA_TEXT_12_REG	SHA encrypted/decrypted data register 12	0x3FF03030	读 / 写
SHA_TEXT_13_REG	SHA encrypted/decrypted data register 13	0x3FF03034	读 / 写
SHA_TEXT_14_REG	SHA encrypted/decrypted data register 14	0x3FF03038	读 / 写
SHA_TEXT_15_REG	SHA encrypted/decrypted data register 15	0x3FF0303C	读 / 写
SHA_TEXT_16_REG	SHA encrypted/decrypted data register 16	0x3FF03040	读 / 写
SHA_TEXT_17_REG	SHA encrypted/decrypted data register 17	0x3FF03044	读 / 写
SHA_TEXT_18_REG	SHA encrypted/decrypted data register 18	0x3FF03048	读 / 写
SHA_TEXT_19_REG	SHA encrypted/decrypted data register 19	0x3FF0304C	读 / 写
SHA_TEXT_20_REG	SHA encrypted/decrypted data register 20	0x3FF03050	读 / 写
SHA_TEXT_21_REG	SHA encrypted/decrypted data register 21	0x3FF03054	读 / 写
SHA_TEXT_22_REG	SHA encrypted/decrypted data register 22	0x3FF03058	读 / 写
SHA_TEXT_23_REG	SHA encrypted/decrypted data register 23	0x3FF0305C	读 / 写
SHA_TEXT_24_REG	SHA encrypted/decrypted data register 24	0x3FF03060	读 / 写
SHA_TEXT_25_REG	SHA encrypted/decrypted data register 25	0x3FF03064	读 / 写
SHA_TEXT_26_REG	SHA encrypted/decrypted data register 26	0x3FF03068	读 / 写
SHA_TEXT_27_REG	SHA encrypted/decrypted data register 27	0x3FF0306C	读 / 写
SHA_TEXT_28_REG	SHA encrypted/decrypted data register 28	0x3FF03070	读 / 写
SHA_TEXT_29_REG	SHA encrypted/decrypted data register 29	0x3FF03074	读 / 写
SHA_TEXT_30_REG	SHA encrypted/decrypted data register 30	0x3FF03078	读 / 写
SHA_TEXT_31_REG	SHA encrypted/decrypted data register 31	0x3FF0307C	读 / 写
控制 / 状态寄存器			
SHA_SHA1_START_REG	Control register to initiate SHA1 operation	0x3FF03080	只写
SHA_SHA1_CONTINUE_REG	Control register to continue SHA1 operation	0x3FF03084	只写
SHA_SHA1_LOAD_REG	Control register to calculate the final SHA1 hash	0x3FF03088	只写
SHA_SHA1_BUSY_REG	Status register for SHA1 operation	0x3FF0308C	只写
SHA_SHA256_START_REG	Control register to initiate SHA256 operation	0x3FF03090	只写
SHA_SHA256_CONTINUE_REG	Control register to continue SHA256 operation	0x3FF03094	只写
SHA_SHA256_LOAD_REG	Control register to calculate the final SHA256 hash	0x3FF03098	只写
SHA_SHA256_BUSY_REG	Status register for SHA256 operation	0x3FF0309C	只读
SHA_SHA384_START_REG	Control register to initiate SHA384 operation	0x3FF030A0	只写
SHA_SHA384_CONTINUE_REG	Control register to continue SHA384 operation	0x3FF030A4	只写
SHA_SHA384_LOAD_REG	Control register to calculate the final SHA384 hash	0x3FF030A8	只写
SHA_SHA384_BUSY_REG	Status register for SHA384 operation	0x3FF030AC	只读
SHA_SHA512_START_REG	Control register to initiate SHA512 operation	0x3FF030B0	只写
SHA_SHA512_CONTINUE_REG	Control register to continue SHA512 operation	0x3FF030B4	只写
SHA_SHA512_LOAD_REG	Control register to calculate the final SHA512 hash	0x3FF030B8	只写
SHA_SHA512_BUSY_REG	Status register for SHA512 operation	0x3FF030BC	只读

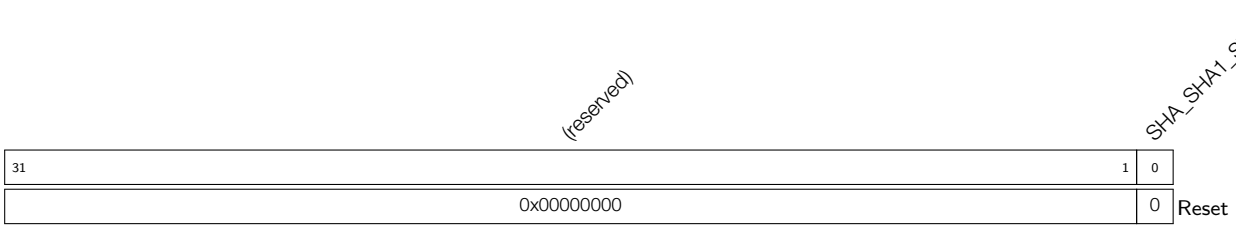
11.5 寄存器

Register 11.1: SHA_TEXT_*n*_REG (*n*: 0-31) (0x0+4**n*)



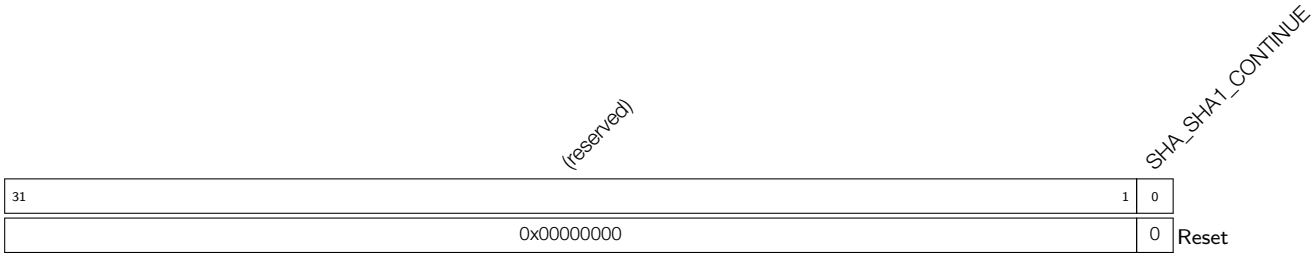
SHA_TEXT_*n*_REG (*n*: 0-31) SHA 信息块和哈希运算结果寄存器。(读 / 写)

Register 11.2: SHA_SHA1_START_REG (0x080)



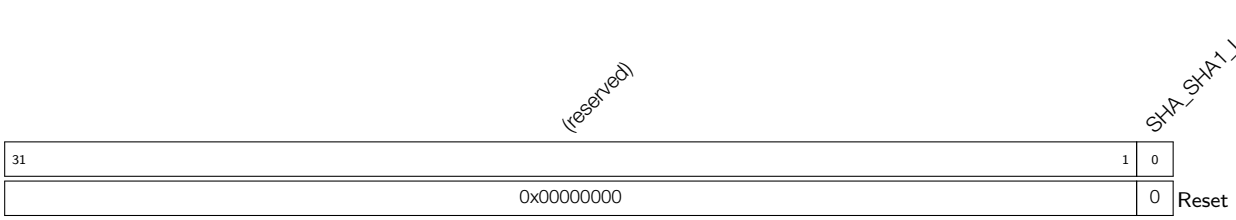
SHA_SHA1_START 写入 1 对第一个信息块进行 SHA-1 运算。(只写)

Register 11.3: SHA_SHA1_CONTINUE_REG (0x084)



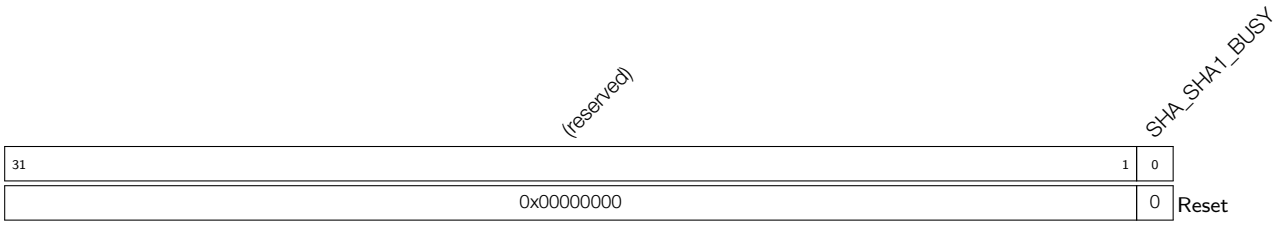
SHA_SHA1_CONTINUE 写入 1 对后续的信息块继续进行 SHA-1 运算。(只写)

Register 11.4: SHA_SHA1_LOAD_REG (0x088)



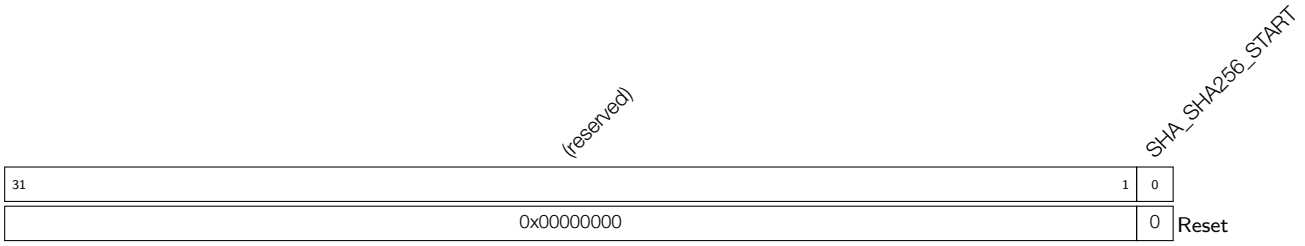
SHA_SHA1_LOAD 写入 1 结束 SHA-1 运算，计算最终的运算结果。(只写)

Register 11.5: SHA_SHA1_BUSY_REG (0x08C)



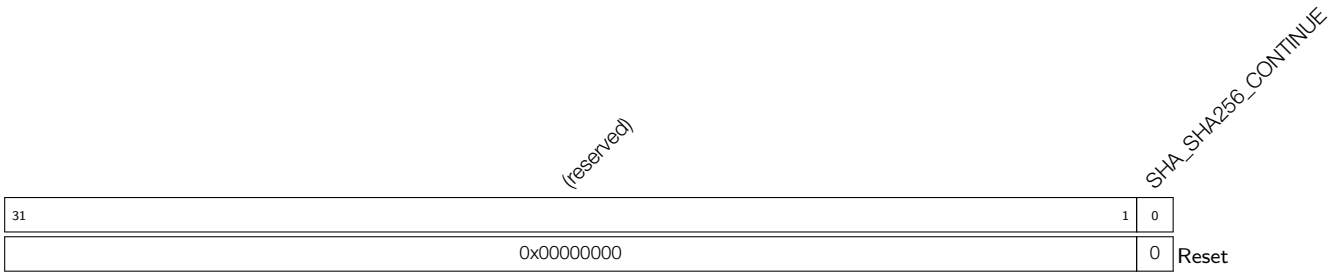
SHA_SHA1_BUSY SHA-1 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。(只读)

Register 11.6: SHA_SHA256_START_REG (0x090)



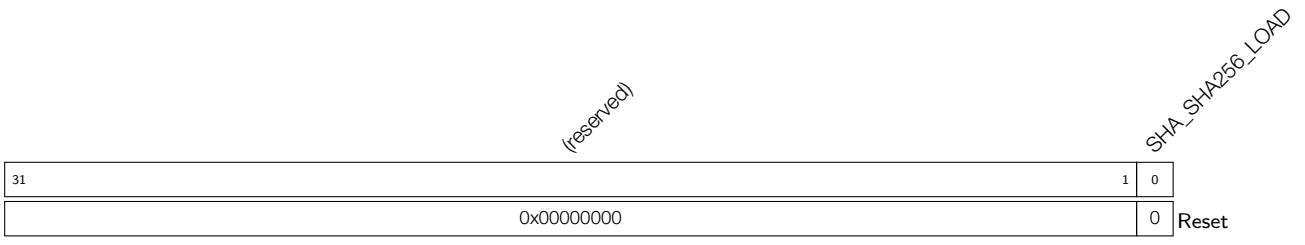
SHA_SHA256_START 写入 1 对第一个信息块进行 SHA-256 运算。(只写)

Register 11.7: SHA_SHA256_CONTINUE_REG (0x094)



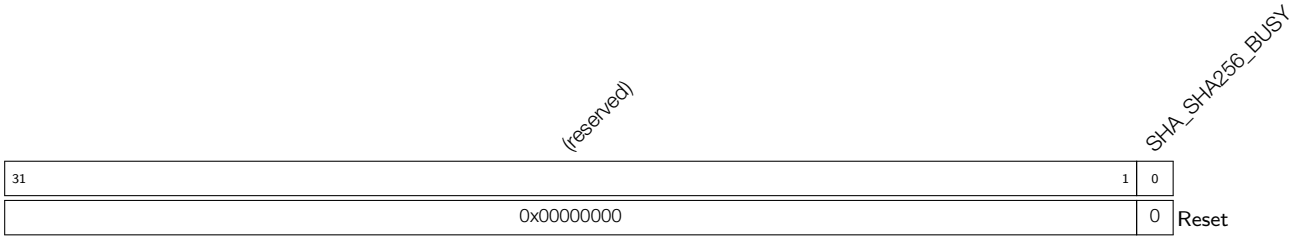
SHA_SHA256_CONTINUE 写入 1 对后续的信息块继续进行 SHA-256 运算。(只写)

Register 11.8: SHA_SHA256_LOAD_REG (0x098)



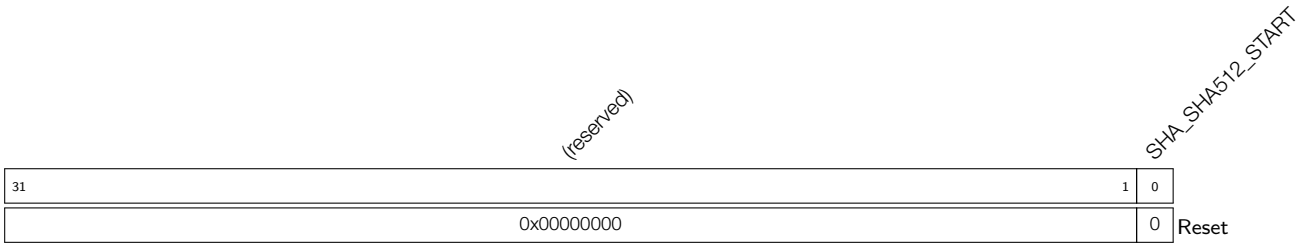
SHA_SHA256_LOAD 写入 1 结束 SHA-256 运算，计算最终的运算结果。(只写)

Register 11.9: SHA_SHA256_BUSY_REG (0x09C)



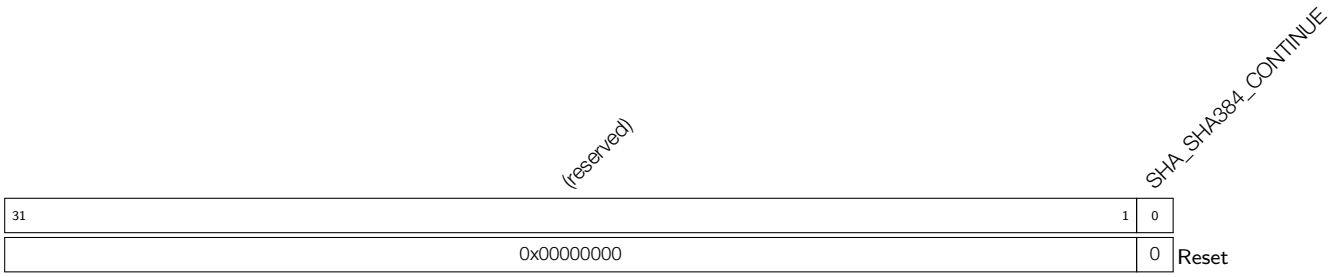
SHA_SHA256_BUSY SHA-256 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。(只读)

Register 11.10: SHA_SHA384_START_REG (0x0A0)



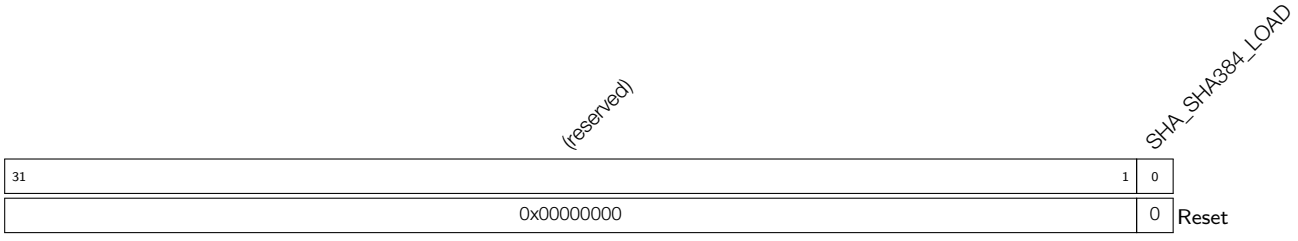
SHA_SHA384_START 写入 1 对第一个信息块进行 SHA-384 运算。(只写)

Register 11.11: SHA_SHA384_CONTINUE_REG (0x0A4)



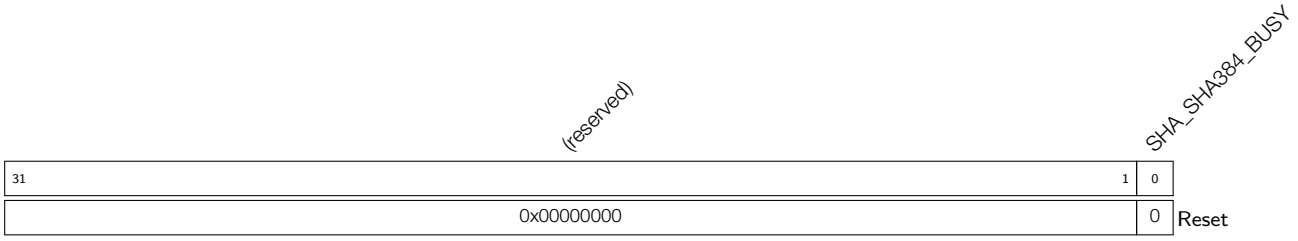
SHA_SHA384_CONTINUE 写入 1 对后续的信息块继续进行 SHA-384 运算。(只写)

Register 11.12: SHA_SHA384_LOAD_REG (0x0A8)



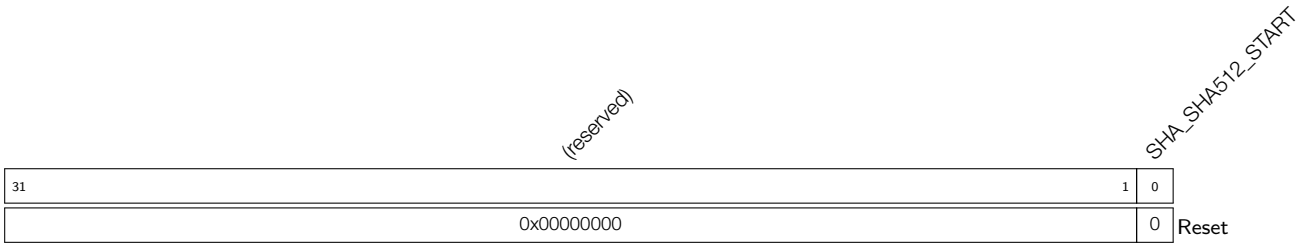
SHA_SHA384_LOAD 写入 1 结束 SHA-384 运算，计算最终的运算结果。(只写)

Register 11.13: SHA_SHA384_BUSY_REG (0x0AC)



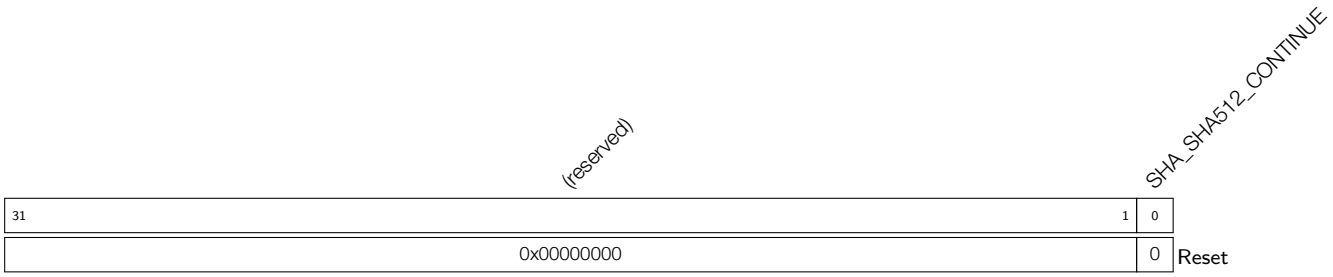
SHA_SHA384_BUSY SHA-384 运算状态寄存器：1：SHA 加速器正在处理数据；0：空闲。(只读)

Register 11.14: SHA_SHA512_START_REG (0x0B0)



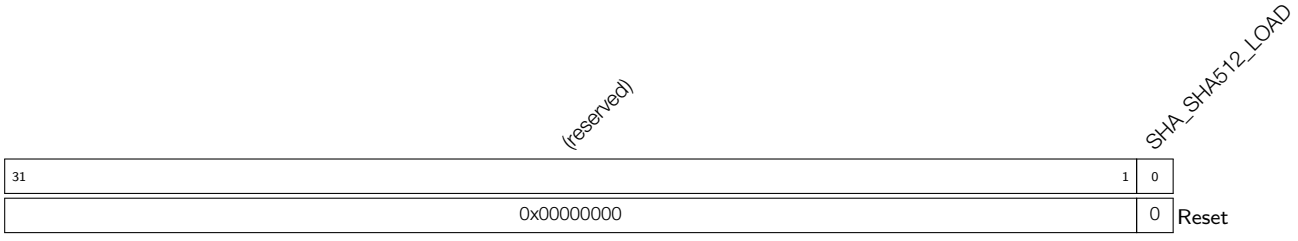
SHA_SHA512_START 写入 1 对第一个信息块进行 SHA-512 运算。(只写)

Register 11.15: SHA_SHA512_CONTINUE_REG (0x0B4)



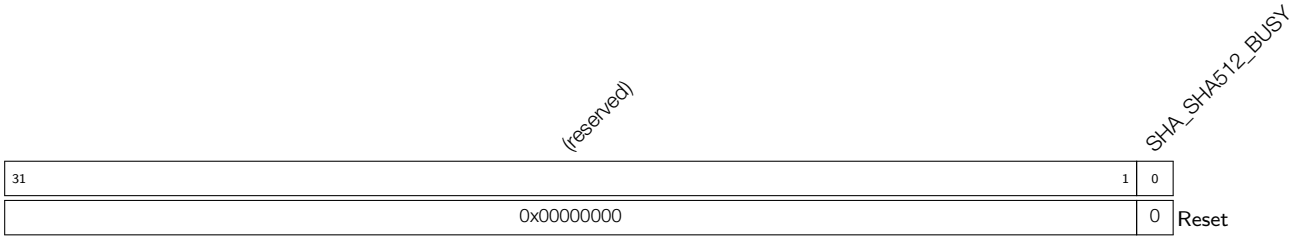
SHA_SHA512_CONTINUE 写入 1 对后续的信息块继续进行 SHA-512 运算。(只写)

Register 11.16: 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 11.17: 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)