ESP32-H2

技术参考手册

预发布 v0.4
乐鑫信息科技

关于本文档

ESP32-H2 技术参考手册面向使用 ESP32-H2 系列产品进行底层软件开发的人员，介绍了 ESP32-H2 系列产品中内置的硬件模块，包括概述、功能列表、硬件架构、编程指南、寄存器列表等信息。

本文档中的跳转

在本文档中实现跳转，请参考以下建议：

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

发布进度速览

注意本文档尚未全部完成，具体发布进度见下表：

No.	ESP32-H2 章节	最新进度
1	ESP-RISC-V CPU	已发布
2	RISC-V 追踪编码器 (TRACE)	已发布
3	通用 DMA 控制器 (GDMA)	已发布
4	系统和存储器	已发布
5	eFuse 控制器 (EFUSE)	已发布
6	IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)	已发布
7	复位和时钟	已发布
8	芯片 Boot 控制	已发布
9	中断矩阵 (INTMTX)	已发布
10	事件任务矩阵 (SOC_ETM)	已发布
11	低功耗管理 (RTC_CNTL) [to be added later]	0%
12	系统定时器 (SYSTIMER)	已发布
13	定时器组 (TIMG)	已发布
14	看门狗定时器 (WDT)	已发布
15	访问权限管理 (APM)	已发布
16	系统寄存器	已发布
17	辅助调试 (ASSIST_DEBUG，MEM_MONITOR)	已发布
18	AES 加速器 (AES)	已发布
19	ECC 加速器 (ECC)	已发布
20	HMAC 加速器 (HMAC)	已发布
21	RSA 加速器 (RSA)	已发布
22	SHA 加速器 (SHA)	已发布
23	数字签名算法 (DSA)	已发布
24	椭圆曲线数字签名算法 (ECDSA)	已发布
25	片外存储器加密与解密 (XTS_AES)	已发布
26	随机数发生器 (RNG) [to be added later]	64%
27	UART 控制器 (UART)	已发布
28	SPI 控制器 (SPI)	已发布
29	I2C 控制器 (I2C)	已发布
30	I2S 控制器 (I2S)	已发布
31	脉冲计数控制器 (PCNT)	已发布
32	USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)	已发布
33	双线汽车接口 (TWAI)	已发布
34	LED PWM 控制器 (LEDC)	已发布
35	电机控制脉宽调制器 (MCPWM)	已发布
36	红外遥控 (RMT)	已发布
37	并行 IO 控制器 (PARL_IO)	已发布
38	SAR ADC 转换器与温度传感器	已发布

说明：

点击链接或扫描二维码确保您使用的是最新版本的文档：
https://www.espressif.com/documentation/esp32-h2_technical_reference_manual_cn.pdf

目录

表格

插图

1 ESP-RISC-V CPU

1.1 概述

ESP-RISC-V CPU 是一个基于 RISC-V 指令集架构 (ISA) 的 32 位内核，包括基本整数 (I)、乘法/除法 (M)、原子 (A) 和压缩 (C) 标准扩展。ESP-RISC-V CPU 内核具有 4 级有序标量流水线，针对面积、功耗、性能等进行了优化。CPU 内核架构包含调试模块 (DM)、中断控制器 (INTC)、核心本地中断 (CLINT) 和用于访问存储器和外设的系统总线 (SYS BUS) 接口。

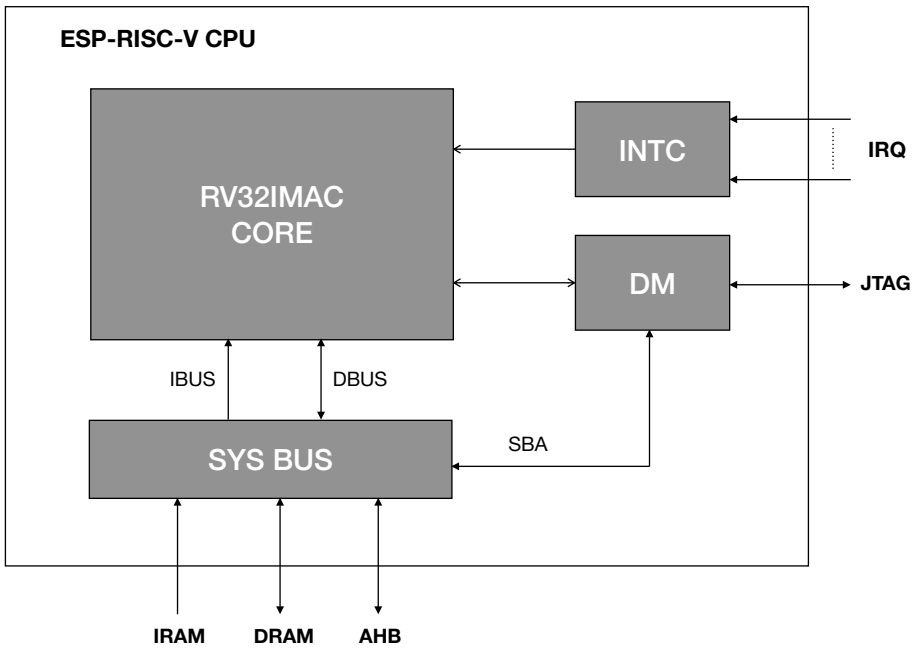


图 1-1. CPU 框图

1.2 特性

- RISC-V RV32IMAC ISA，四级流水线，时钟工作频率高达 96 MHz
- 符合 RISC-V 指令集手册 v2.2 第一卷“非特权架构”(RISC-V Instruction Set Manual Volume I: Unprivileged ISA, Version 2.2) 和 RISC-V 指令集手册 v1.10 第二卷“特权架构”(RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10)
- 通过 IRAM/DRAM 接口零等待周期访问片上 SRAM 和缓存中的程序和数据
- 具有静态分支预测功能的分支目标缓冲区 (BTB)
- 支持用户模式 (user mode) 以及中断委托
- 中断控制器 (INTC) 具有多达 28 个外部向量中断，适用于机器模式 (machine mode) 和用户模式，可配置 16 个优先级和阈值级别
- 专用于各个特权模式的核心本地中断 (CLINT)
- 调试模块 (DM) 符合 RISC-V 外部调试支持规范 (RISC-V External Debug Support) v0.13，支持通过行业标准的 JTAG/USB 端口连接外部调试器

- 支持指令追踪
- 调试器通过系统总线 (SBA) 直接访问存储器和外设
- 硬件触发器符合 RISC-V 调试规范 v0.13，具有多达 4 个断点/观察点
- 物理存储器保护 (PMP) 和物理存储器属性 (PMA)，最多可配置 16 个区域
- 32 位 AHB 系统总线，用于访问外设
- 可配置的核心性能指标事件

1.3 术语

分支 (branch) 在一定条件下改变执行流程的指令

delta 指令在存储器中的存储地址不连续时，程序计数器的地址偏移

hart RISC-V 硬件线程

退役 (retire) 机器状态更新时，执行指令的最后一个阶段

陷阱 (trap) 即异常或中断，会触发陷阱处理程序

1.4 地址分布

下表列出了 CPU 可访问的指令地址空间、数据地址空间、调试地址空间和通过系统总线访问的外设地址空间。

表 1-2. CPU 地址分布

名称	描述	起始地址	结束地址	访问
IRAM/DRAM	指令/数据空间	0x4000_0000	0x4FFF_FFFF	读/写
CPU	CPU 子系统空间	0x2000_0000	0x2FFF_FFFF	读/写
AHB	AHB 外设空间	* 默认	* 默认	读/写

* 默认：IRAM、DRAM、CPU 地址范围以外的地址空间通过 AHB 总线访问。

1.5 配置与状态寄存器 (CSR)

1.5.1 寄存器列表

下表为 CPU 可访问的 CSR 列表。除了自定义的性能计数器 CSR 外，所有已实现的 CSR 都遵循 RISC-V 指令集手册 v1.10 第二卷 “特权架构” 中所述的位域标准映射，但是，由于 CPU 并未实现所有的功能子集，所以某些标准 CSR 中的部分位域并未实现。有关详细的 CSR 寄存器描述，请参阅下一小节。

名称	描述	地址	访问
机器模式信息 CSR			
mvendorid	机器模式供应商编号寄存器	0xF11	RO
marchid	机器模式架构编号寄存器	0xF12	RO
mimpid	机器模式硬件实现编号寄存器	0xF13	RO
mhartid	机器模式 hart 编号寄存器	0xF14	RO
机器模式陷阱设置 CSR			
mstatus	机器模式状态寄存器	0x300	R/W

名称	描述	地址	访问
misa [1]	机器模式 ISA 寄存器	0x301	R/W
mideleg	机器模式中断委托寄存器	0x303	R/W
mie	机器模式中断使能寄存器	0x304	R/W
mtvec [2]	机器模式异常向量寄存器	0x305	R/W
机器模式陷阱处理 CSR			
mscratch	机器模式暂存寄存器	0x340	R/W
mepc	机器模式异常程序计数器	0x341	R/W
mcause [3]	机器模式异常原因寄存器	0x342	R/W
mtval	机器模式异常值寄存器	0x343	R/W
mip	机器模式中断等待寄存器	0x344	R/W
用户模式陷阱设置 CSR			
ustatus	用户模式状态寄存器	0x000	R/W
uie	用户模式中断使能寄存器	0x004	R/W
utvec	用户模式异常向量寄存器	0x005	R/W
用户模式陷阱处理 CSR			
uscratch	用户模式暂存寄存器	0x040	R/W
uepc	用户模式异常程序计数器	0x041	R/W
ucause	用户模式异常原因寄存器	0x042	R/W
uip	用户模式中断等待寄存器	0x044	R/W
物理存储保护 (PMP) CSR			
pmpcfg0	物理存储器保护配置寄存器	0x3A0	R/W
pmpcfg1	物理存储器保护配置寄存器	0x3A1	R/W
pmpcfg2	物理存储器保护配置寄存器	0x3A2	R/W
pmpcfg3	物理存储器保护配置寄存器	0x3A3	R/W
pmpaddr0	物理存储器保护地址寄存器	0x3B0	R/W
pmpaddr1	物理存储器保护地址寄存器	0x3B1	R/W
……			
pmpaddr15	物理存储器保护地址寄存器	0x3BF	R/W
触发器模块 CSR（与调试模块共用）			
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
tcontrol	全局触发器控制寄存器	0x7A5	R/W
调试模式 CSR			
dcsr	调试模式控制与状态寄存器	0x7B0	R/W
dpc	调试 PC 寄存器	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试模式暂存寄存器 1	0x7B3	R/W
性能计数 CSR （自定义） [4]			
mpcer	机器模式性能计数器事件寄存器	0x7E0	R/W

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

[2] mtvec 仅支持在向量模式下对异常处理进行配置，基地址为 256 字节对齐。

[3] mcause 中反映的外部中断 ID 也包括 RISC-V 标准为核心内部中断源预留的 ID。

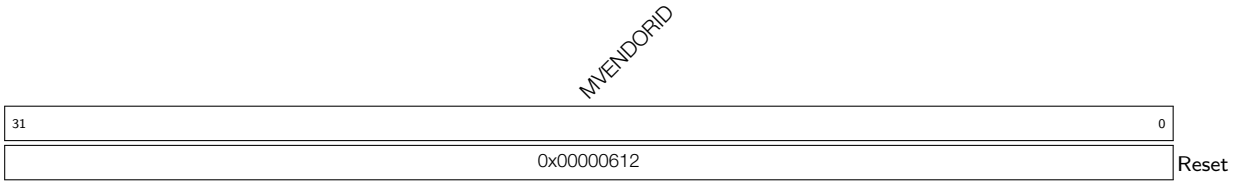
[4] 这些自定义机器模式 CSR 已经在 RISC-V 标准为用户保留的地址空间中实现。

名称	描述	地址	访问
mpcmr	机器模式性能计数器模式寄存器	0x7E1	R/W
mpccr	机器模式性能计数器计数寄存器	0x7E2	R/W
GPIO 访问 CSR（自定义）			
cpu_gpio_oen	GPIO 输出使能寄存器	0x803	R/W
cpu_gpio_in	GPIO 读输入值寄存器	0x804	RO
cpu_gpio_out	GPIO 写输出值寄存器	0x805	R/W
物理存储器属性检查器 (PMAC) CSR			
pma_cfg0	物理存储器属性配置寄存器	0xBC0	R/W
pma_cfg1	物理存储器属性配置寄存器	0xBC1	R/W
pma_cfg2	物理存储器属性配置寄存器	0xBC2	R/W
pma_cfg3	物理存储器属性配置寄存器	0xBC3	R/W
……			
pma_cfg15	物理存储器属性配置寄存器	0xBCF	R/W
pma_addr0	物理存储器属性地址寄存器	0xBD0	R/W
pma_addr1	物理存储器属性地址寄存器	0xBD1	R/W
……			
pma_addr15	物理存储器属性地址寄存器	0xBDF	R/W

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

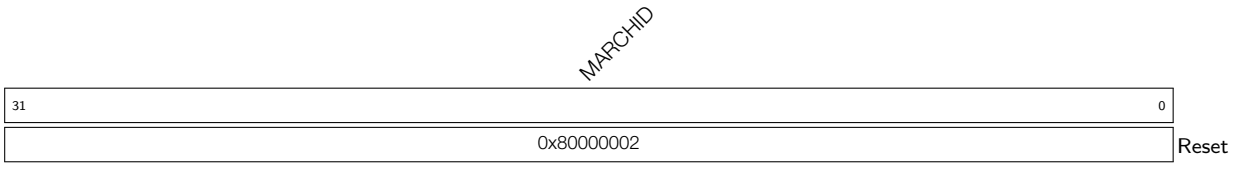
1.5.2 寄存器

Register 1.1. mvendorid (0xF11)



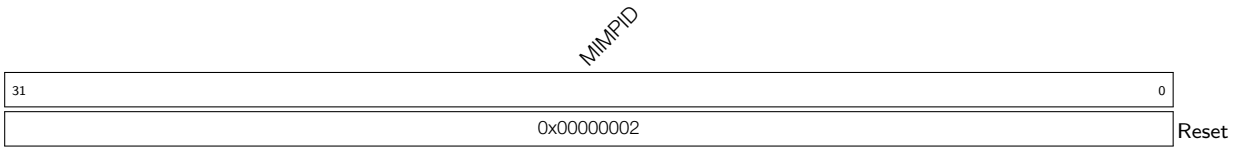
MVENDORID 表示供应商编号。(RO)

Register 1.2. marchid (0xF12)



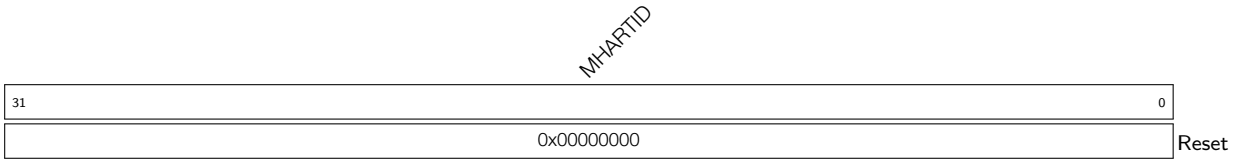
MARCHID 表示架构编号。(RO)

Register 1.3. mimpid (0xF13)



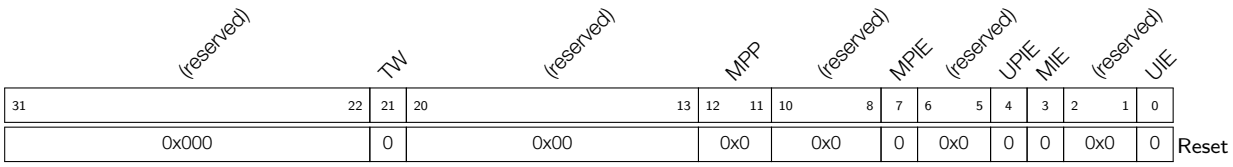
MIMPID 表示硬件实现编号。(RO)

Register 1.4. mhartid (0xF14)



MHARTID 表示硬件线程编号。(RO)

Register 1.5. mstatus (0x300)



UIE 写 1 使能全局用户模式中断。(R/W)

MIE 写 1 使能全局机器模式中断。(R/W)

UPIE 写 1 使能（异常）之前的用户模式中断。(R/W)

MPIE 写 1 使能（异常）之前的机器模式中断。(R/W)

MPP 配置（异常）之前的机器特权模式。

0x0：用户模式

0x3：机器模式

注意：仅低位可写。由于高位直接绑定低位，写入高位将被忽略。

(R/W)

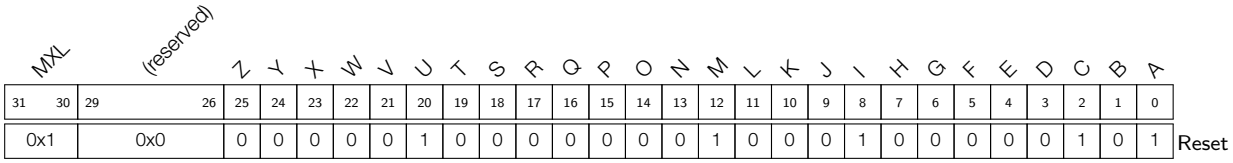
TW 配置在用户模式下执行 WFI（等待中断）指令是否导致非法指令异常。

0：不导致非法指令异常

1：导致非法指令异常

(R/W)

Register 1.6. misa (0x301)



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

Reset

MXL 机器 XLEN = 1（32 位）。(RO)

Z 保留 = 0。(RO)

Y 保留 = 0。(RO)

X 非标准扩展 = 0。(RO)

W 保留 = 0。(RO)

V 保留 = 0。(RO)

U 实现用户模式 = 1。(RO)

T 保留 = 0。(RO)

S 实现监督模式 = 0。(RO)

R 保留 = 0。(RO)

Q 四精度浮点扩展 = 0。(RO)

P 保留 = 0。(RO)

O 保留 = 0。(RO)

N 支持用户级别中断 = 0。(RO)

M 整数乘除法标准扩展 = 1。(RO)

L 保留 = 0。(RO)

K 保留 = 0。(RO)

J 保留 = 0。(RO)

I RV32I 基本 ISA = 1。(RO)

H 虚拟机管理程序扩展 = 0。(RO)

G 其他标准扩展 = 0。(RO)

F 单精度浮点扩展 = 0。(RO)

E RV32E 基本 ISA = 0。(RO)

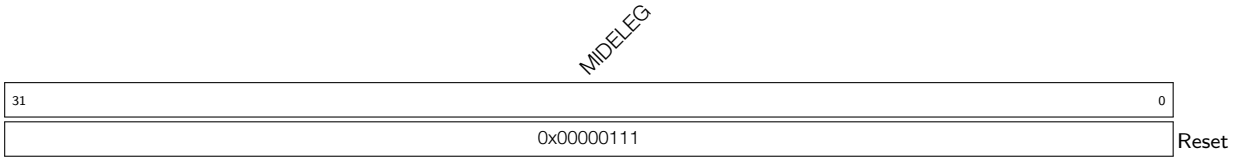
D 双精度浮点扩展 = 0。(RO)

C 压缩标准扩展 = 1。(RO)

B 保留 = 0。(RO)

A 原子标准扩展 = 1。(RO)

Register 1.7. mideleg (0x303)



MIDELEG 配置是否委托相应中断给用户模式。以下中断默认委托给 U 模式：

Bit 0：用户软件中断 (CLINT)

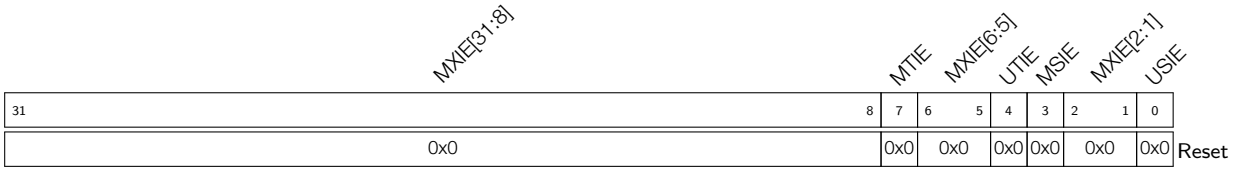
Bit 4：用户定时器中断 (CLINT)

Bit 8：用户外部中断

可根据运行时的实际需求修改默认委托给用户模式的中断。

(R/W)

Register 1.8. mie (0x304)



USIE 写 1 使能用户软件中断。(R/W)

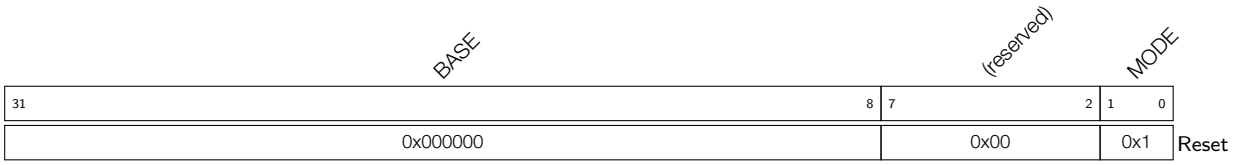
MSIE 写 1 使能机器软件中断。(R/W)

UTIE 写 1 使能用户定时器中断。(R/W)

MTIE 写 1 使能机器定时器中断。(R/W)

MXIE 写 1 使能 28 个外部中断。(R/W)

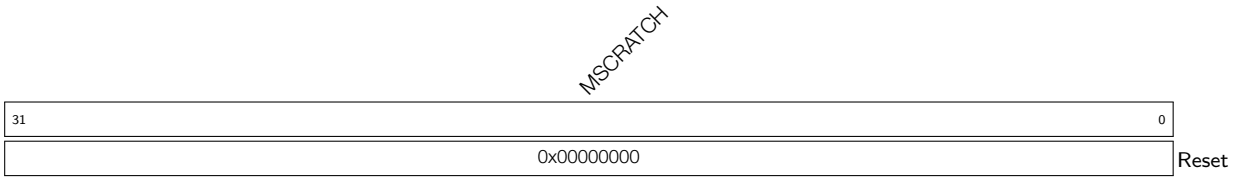
Register 1.9. mtvec (0x305)



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

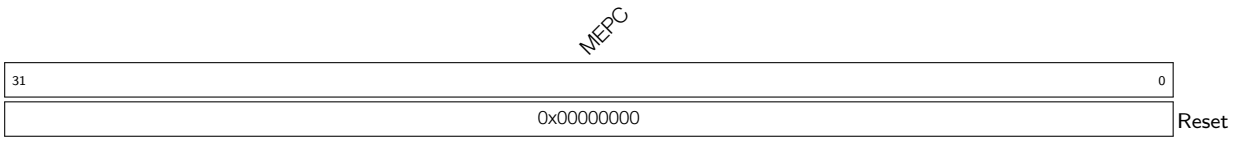
BASE 配置异常向量基址的高 24 位，地址为 256 字节对齐。(R/W)

Register 1.10. mscratch (0x340)



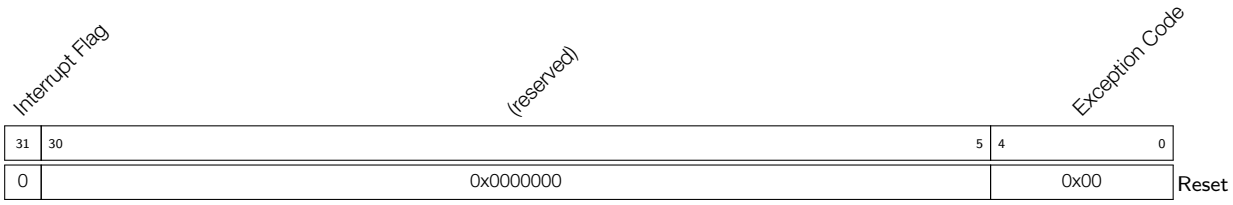
MSCRATCH 配置用户自定义的机器暂存信息。(R/W)

Register 1.11. mepc (0x341)



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

Register 1.12. mcause (0x342)



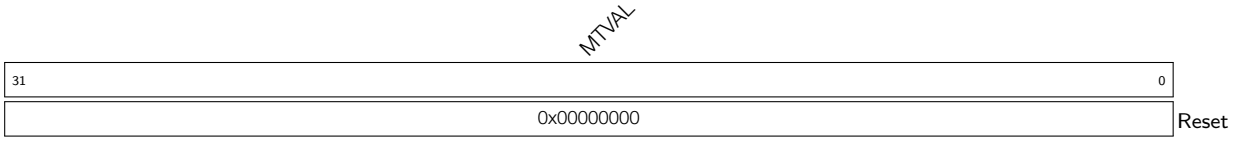
Exception Code CPU 进入异常时，此域将自动更新为最近的异常或中断的唯一 ID。可能的异常 ID：

0x1：PMP 指令访问错误
0x2：非法指令
0x3：硬件断点/观察点或 EBREAK
0x5：PMP 读存储器访问错误
0x6：写存储器地址或 AMO 地址未对齐
0x7：PMP 写存储器访问错误
0x8：用户模式环境调用 (ECALL)
0xb：机器模式环境调用
其他值：保留
注意：异常 ID 0x0（指令地址非对齐）不存在，因为 CPU 在取指时始终屏蔽地址的最低位。
(R/W)

Interrupt Flag CPU 进入异常时，此标志位将自动更新。

如果被置位，则表示最近的陷阱是由中断引起。在异常情况下保持为 0。
注意：中断控制器将中断编号 1-2、5-6、8-31 全部用于外部中断源，而 RISC-V 标准则为内核的内部中断源预留了编号 0-15。本地中断源 (CLINT) 使用 0、3、4、7 等保留编号。
(R/W)

Register 1.13. mtval (0x343)



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

0x1：指令虚拟地址错误

0x2：指令 opcode 错误

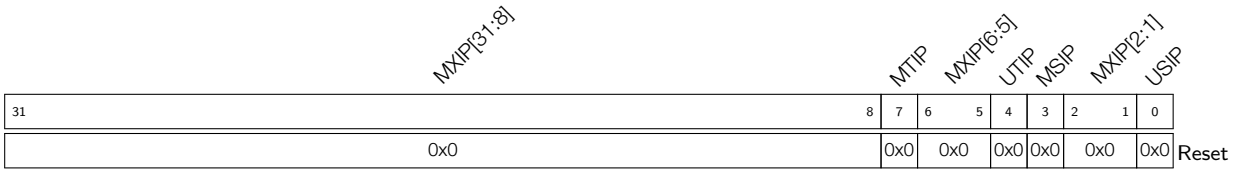
0x5：存储器读操作的数据地址错误

0x7：存储器写操作的数据地址错误

注意：该寄存器不支持其他异常 ID 和中断。

(R/W)

Register 1.14. mip (0x344)



USIP 配置用户软件中断的等待状态。

0：不等待

1：等待

(R/W)

MSIP 配置机器软件中断的等待状态。

0：不等待

1：等待

(R/W)

UTIP 配置用户定时器中断的等待状态。

0：不等待

1：等待

(R/W)

MTIP 配置机器定时器中断的等待状态。

0：不等待

1：等待

(R/W)

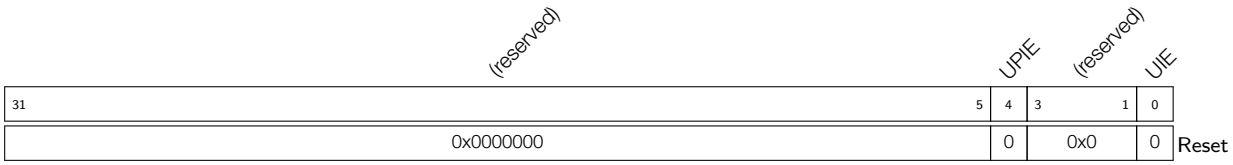
MXIP 配置 28 个外部中断的等待状态。

0：不等待

1：等待

(R/W)

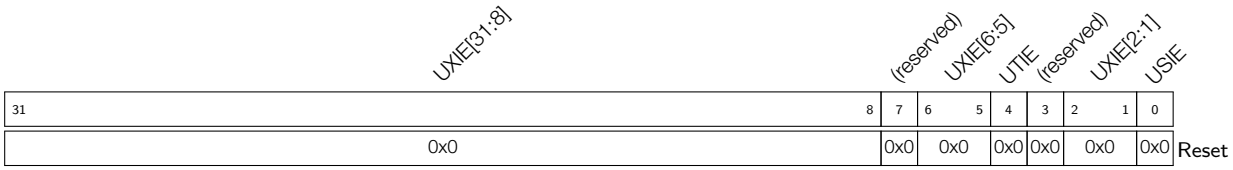
Register 1.15. ustatus (0x300)



UIE 写 1 使能全局用户模式中断。(R/W)

UPIE 写 1 使能（异常）之前的用户中断。(R/W)

Register 1.16. uie (0x004)

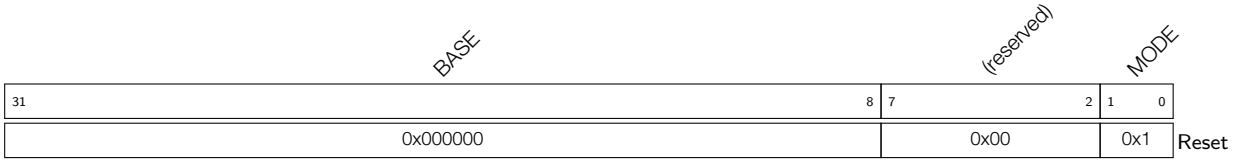


USIE 写 1 使能用户软件中断。(R/W)

UTIE 写 1 使能用户定时器中断。(R/W)

UXIE 写 1 使能 28 个委托给用户模式的外部中断。(R/W)

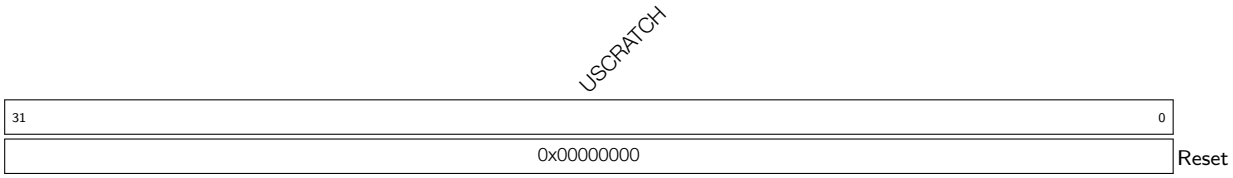
Register 1.17. utvec (0x005)



MODE 表示用户模式中断是否为向量模式，仅支持向量模式 **0x1**。(RO)

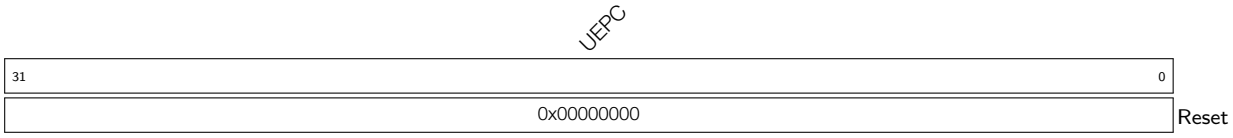
BASE 配置异常向量基址的高 24 位，地址为 256 字节对齐。(R/W)

Register 1.18. uscratch (0x040)



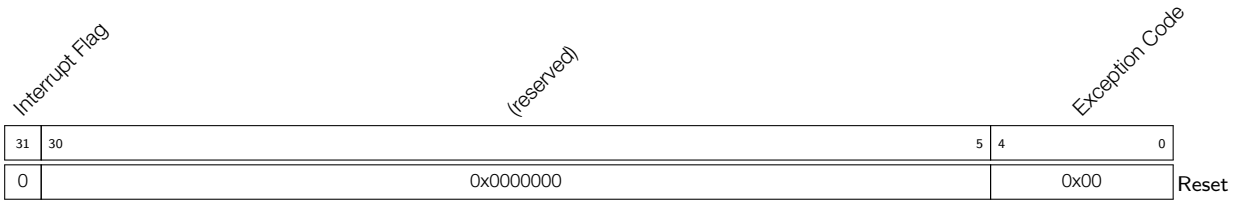
USCRATCH 配置用户自定义的机器暂存信息。(R/W)

Register 1.19. uepc (0x041)



UEPC 配置用户异常程序计数器。当 CPU 遇到异常时，此域将自动更新为 CPU 将要执行的指令的地址。(R/W)

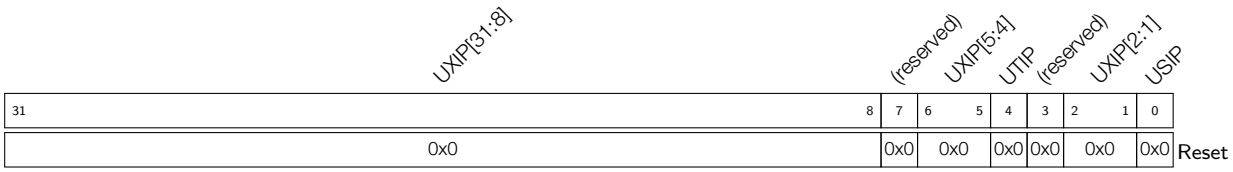
Register 1.20. ucause (0x042)



Interrupt ID 进入异常时，此域将自动更新为最近的异常或中断的唯一 ID。(R/W)

Interrupt Flag 此标志将始终设置，因为 CPU 只能由于用户模式中断而进入陷阱，因为不支持异常委托。(R/W)

Register 1.21. uip (0x044)



USIP 配置用户软件中断的等待状态。

0：不等待

1：等待

(R/W)

UTIP 配置用户定时器中断的等待状态。

0：不等待

1：等待

(R/W)

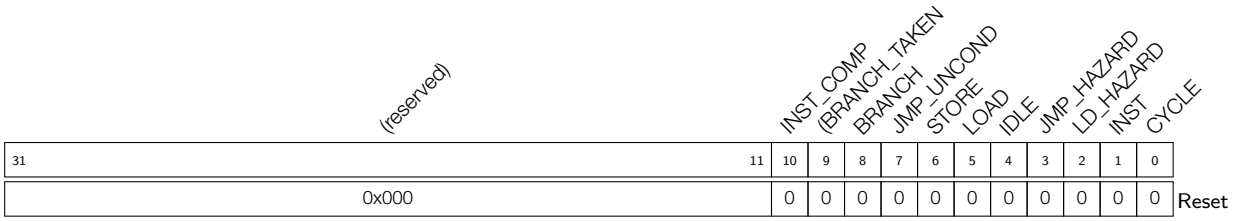
UXIP 配置 28 个委托给用户模式的外部中断。

0：不等待

1：等待

(R/W)

Register 1.22. mpcer (0x7E0)



INST_COMP 计数压缩指令。(R/W)

BRANCH_TAKEN 计数分支跳转。(R/W)

BRANCH 计数分支。(R/W)

JMP_UNCOND 计数无条件跳转。(R/W)

STORE 计数存储器写操作。(R/W)

LOAD 计数存储器读操作。(R/W)

IDLE 计数 IDLE 周期。(R/W)

JMP_HAZARD 计数跳转冒险。(R/W)

LD_HAZARD 计数存储器读操作冒险。(R/W)

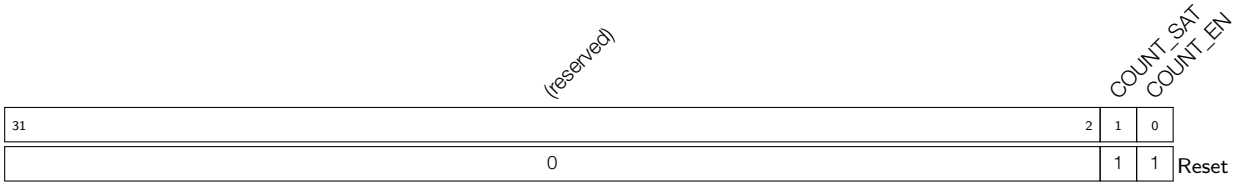
INST 计数指令。(R/W)

CYCLE 计数时钟周期，WFI 模式下周期计数不增加。

注意：每个位选择一个特定事件由计数器递增计数。如果多个事件被选择且同时发生，则计数器只递增 1。

(R/W)

Register 1.23. mpcmr (0x7E1)



COUNT_SAT 配置计数器饱

0：超出最大值上溢

1：超出最大值暂停

(R/W)

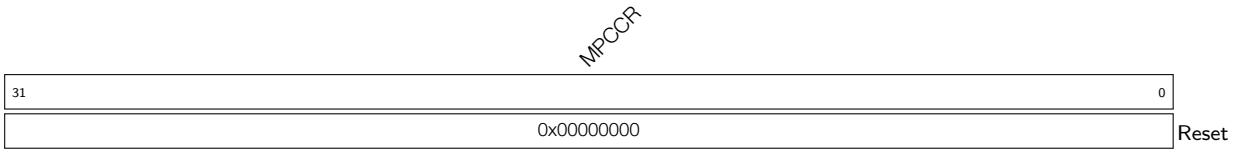
COUNT_EN 配置是否使能计数器。

0：不使能

1：使能

(R/W)

Register 1.24. mpccr (0x7E2)



MPCCR 表示机器性能计数器数值。(R/W)

1.6 中断控制器

1.6.1 特性

中断控制器能够捕获、屏蔽来自 RISC-V CPU 外部的中断源，并对中断源的优先级进行动态仲裁。中断控制器具有以下特性：

- 多达 28 个外部异步中断和 4 个核心本地中断源 (CLINT)，皆有唯一 ID (0-31)
- 可通过读写存储器映射寄存器配置中断控制器
- 可委托给用户模式
- 15 个优先级级别，可以分配给不同的中断
- 支持电平触发或边沿触发的中断源
- 可配置的全局阈值，用于屏蔽优先级较低的中断
- 与陷阱向量地址偏移量匹配的中断 ID

中断寄存器的完整列表及详细描述请见章节 9 中断矩阵 *(INTMTX)* > 章节 9.6.2。

1.6.2 功能描述

每个中断 ID 都有 6 个属性，对于 28 个外部中断（1-2、5-6、8-31）来说，这些属性皆可配置，但是对于 4 个本地 CLINT 中断（0、3、4、7）来说，这些属性是静态的（模式除外）。这些属性如下：

1. 模式 (M/U)：
    - 决定中断在哪种模式下被处理。
    - 通过设置或清除 mideleg CSR 中的相应位进行配置。
    - 如果某个中断在 mideleg CSR 中的对应位被清除，则该中断将在机器模式下被捕获。
    - 如果某个中断在 mideleg CSR 中的对应位被设置，则该中断将被委托到用户模式。
2. 使能状态 (0-1)：
    - 决定是否允许由 CPU 捕获和处理中断。
    - 通过写入 INTPRI_CORE0_CPU_INT_ENABLE_REG 相应的位进行配置。
    - 由于本地 CLINT 中断在存储器映射寄存器中保留了相应的位，因此这些中断在中断控制器级别始终为使能状态。
    - 机器模式下的外部中断和本地中断还需要通过在 mie CSR 中设置相应的位来在核心级别取消屏蔽。
    - 用户模式下的外部中断和本地中断还需要通过在 uie CSR 中设置相应的位来在核心级别取消屏蔽。
3. 类型 (0-1)：
    - 在中断信号的上升沿锁存中断状态。
    - 通过写入 INTPRI_CORE0_CPU_INT_TYPE_REG 相应的位进行配置。
    - 类型保持为 0 的中断称为电平触发中断。
    - 类型保持为 1 的中断称为边沿触发中断。
    - 本地 CLINT 中断始终为电平触发中断，因此在上述寄存器中保留了相应的位。

4. 优先级 (1-15):
    - 当有多个中断在等待时，决定 CPU 先处理哪一个中断。
    - 通过写入外部中断 ID *n* 的 INTPRI_CORE0_CPU_INT_PRI_*n*_REG 进行配置。
    - 优先级小于 INTPRI_CORE0_CPU_INT_THRESH_REG 指定阈值的外部中断将被屏蔽。
    - 优先级最低为 0，最高为 15。
    - 具有相同优先级的中断通过其 ID 静态确定优先级，ID 越小，优先级越高。
    - 本地 CLINT 中断具有静态的优先级，因此保留了相应的优先级寄存器。
    - 本地 CLINT 中断在两种模式下均不会被优先级阈值屏蔽。
5. 等待状态 (0-1):
    - 反映已使能且未被屏蔽的外部中断信号被捕获时的状态。
    - 通过读取 INTPRI_CORE0_CPU_INT_EIP_STATUS_REG 中的相应位获得每个外部中断 ID 的等待状态。
    - 也可通过读取机器模式中断的 mip CSR 或用户模式中断的 uip CSR 中的相应位来获取外部中断和本地中断 ID 的等待状态。
    - 如果没有更高优先级的中断在等待，则当前在等待的中断将导致 CPU 进入陷阱。
    - 如果在等待的中断抢占 CPU 并导致其跳转到相应的陷阱向量地址，则称该中断为“已声明”。
    - 所有在等待的中断都为“未声明”。
6. 清除状态 (0-1):
    - 切换此属性将仅清除已声明的边沿类型中断的等待状态。
    - 通过先置位然后清零 INTPRI_CORE0_CPU_INT_CLEAR_REG 中的相应位进行切换。
    - 电平触发中断的等待状态不受切换操作的影响，而必须从中断源中清除。
    - 未声明的边沿类型中断的等待状态可以被清空，方法是先清零 INTPRI_CORE0_CPU_INT_ENABLE_REG 中的相应位再置位 INTPRI_CORE0_CPU_INT_CLEAR_REG 中的相同位。

有关核心本地中断源的详细说明，请参阅章节 1.7。

当 CPU 处理在等待的机器模式或用户模式中断时，会进行以下操作:

- 将当前未执行指令的地址保存在 mepc 或 uepc 中，以便之后恢复执行。
- 将 mcause 或 ucause 的值更新为正在处理的中断 ID。
- 将 MIE 或 UIE 的状态复制到 MPIE 或 UPIE，然后清零 MIE 或 UIE，从而全局禁用中断。
- 通过跳转到 mtvec 或 utvec 中存储地址的字对齐偏移量进入陷阱。

机器模式中断 $ID = i$ 按字对齐的陷阱地址 = $(mtvec + 4i)$，用户模式中断 $ID = i$ 按字对齐的陷阱地址 = $(utvec + 4i)$。

在跳转到对应模式的陷阱向量之后，执行流程取决于软件实现，但一般来说该中断将在某个中断服务程序 (ISR) 中被处理（并清除），然后在 CPU 遇到 MRET 或 URET 指令后恢复正常程序流。

执行 MRET 或 URET 指令后，CPU 将进行以下操作:

- 将 MPIE 或 UPIE 的状态复制回 MIE 或 UIE，然后清零 MPIE 或 UPIE。这意味着，如果之前置位了 MPIE 或 UPIE，则执行 MRET 或 URET 后 MIE 或 UIE 将被置位，进而全局使能中断。
- 跳转到 mepc 或 uepc中存储的地址，然后恢复执行。

软件可以在 ISR 内部实现中断嵌套，具体请参考章节 1.6.3。

中断控制器具有以下行为特点：

- 仅当中断优先级大于或等于阈值寄存器中的值时，它才会反映在 INTPRI_CORE0_CPU_INT_EIP_STATUS_REG 中。
- 如果一个中断反映在 INTPRI_CORE0_CPU_INT_EIP_STATUS_REG 中但是还未被处理，则可以通过降低它的优先级或提高全局阈值将其屏蔽（进而防止 CPU 对其进行处理）。
- 如果一个中断反映在 INTPRI_CORE0_CPU_INT_EIP_STATUS_REG 中，要清除它（防止被处理），则必须将其禁用（如果是边沿属性的中断则需要清除）。

1.6.3 建议操作

1.6.3.1 延迟

配置中断控制器时应考虑延迟问题。

在稳态操作中，中断控制器的等待时间固定为 4 个周期。稳态操作的意思是最近没有对中断控制器寄存器作任何更改。这意味着一个中断从被中断控制器触发 (assert) 到被 CPU 开始处理刚好消耗 4 个周期。这也意味着，在抢占发生之前，CPU 最多可以执行 5 条指令。

当寄存器被修改时，中断控制器会进入临时状态，然后需要最多 4 个周期才能再次进入稳态。在临时状态期间，中断的顺序可能无法预测，因此，需要软件采取一些安全措施以避免任何同步问题。

还须注意的是，中断控制器的配置寄存器位于 APB 地址范围内，因此对这些寄存器的读写访问可能需要消耗几个周期。

考虑到上述特征，建议用户在修改中断控制器寄存器时遵循以下操作顺序：

1. 保存 MIE 的状态，然后将其清零
2. 通过“读-修改-写”的方式写中断控制器寄存器
3. 执行 FENCE 指令以等待所有未完成的写操作完成
4. 最后，恢复 MIE 的状态

如上述步骤显示，建议用户在配置中断控制器寄存器之前先全局禁用中断 (MIE=0)，然后立即恢复 MIE。

执行完上述操作后，中断控制器将恢复稳态操作。

1.6.3.2 配置流程

默认情况下，mstatus 里的 MIE 为 0，即全局禁用中断。在中断堆栈初始化（包括将 mtvec 设置为中断向量地址）完成之后，软件必须将 MIE 置为 1。

在 INTPRI_CORE0_CPU_INT_THRESH_REG 中，外部中断的阈值默认为 0。对于基于优先级的中断屏蔽，可在 CPU 复位后将对应位初始化为 1。这样所有默认优先级为 0 的中断源都将被屏蔽。

在正常情况下，如果要使能某个外部中断 *n*，可以遵循以下步骤：

1. 保存 MIE 的状态，然后将其清零
2. 根据中断的类型（边沿/电平），置位或清零 INTPRI_CORE0_CPU_INT_TYPE_REG 中的第 *n* 个位
3. 通过写入 INTPRI_CORE0_CPU_INT_PRI_*n*_REG 指定优先级（最低为 1，最高为 15）
4. 置位 INTPRI_CORE0_CPU_INT_ENABLE_REG 中的第 *n* 个位
5. 执行 FENCE 指令
6. 恢复 MIE 的状态

当一个或多个中断在等待时，CPU 将确认（声明）最高优先级的中断，然后跳转到与该中断 ID 相对应的陷阱向量地址。软件可以通过读取 mcause 来推断陷阱类型（mcause(31) 为 1 代表中断，为 0 代表异常）和中断 ID（mcause(4-0) 提供中断或异常的 ID）。如果陷阱向量中的每个表项都是指向不同陷阱处理程序的跳转指令，则软件无需做此推断。最后，陷阱处理程序会将程序指引到该中断相应的 ISR。

进入 ISR 后，如果是边沿触发中断，则软件必须置位 INTPRI_CORE0_CPU_INT_CLEAR_REG 中的第 *n* 个位。如果是电平触发中断，则必须清除相应的中断源。

软件还可以更新 INTPRI_CORE0_CPU_INT_THRESH_REG 的值并置位 MIE 来让更高优先级的中断抢占当前 ISR（即嵌套），但是，在此之前，必须先保存所有状态 CSR（mepc、mstatus、mcause 等），这是由于发生嵌套时状态 CSR 的值会被覆盖。之后，在退出 ISR 时，再恢复这些 CSR 的值。

最后，程序从 ISR 返回到陷阱处理程序之后，可以执行 MRET 指令以恢复正常程序流。

如果不再需要中断 *n* 并且需要将其禁用，则可以遵循以下操作步骤：

1. 保存 MIE 的状态，然后将其清零
2. 读取 INTPRI_CORE0_CPU_INT_EIP_STATUS_REG 检查中断是否在等待
3. 置位/复位 INTPRI_CORE0_CPU_INT_ENABLE_REG 中的第 *n* 个位
4. 如果中断属于边沿类型并且在等待，则必须切换 INTPRI_CORE0_CPU_INT_CLEAR_REG 中的第 *n* 个位以清空它的等待状态
5. 执行 FENCE 指令
6. 恢复 MIE 的状态

以上只是建议的操作方案，实际操作由软件实现决定。

1.6.4 寄存器

有关中断寄存器的完整列表及详细描述，请分别参考章节 9.6.2 和章节 9.7.2。

1.7 核心本地中断 (CLINT)

1.7.1 概述

CPU 支持 4 个本地电平触发中断源，它们具有静态的优先级。

表 1-4. 核心本地中断 (CLINT) 源

ID	描述	优先级
0	用户模式软件中断	1
3	机器模式软件中断	3
4	用户模式定时器中断	0
7	机器模式定时器中断	2

这些中断源具有固定的 ID 和优先级，在两种模式下均不会被中断控制器的阈值寄存器屏蔽。

根据 mideleg CSR 中相应位的复位值，其中有两个中断（0 和 4）默认委托给用户模式。

必须注意的是，虽然 CLINT 中断有固定的优先级，但是等待中的外部中断源总是比 CLINT 源具有更高的优先级。

1.7.2 特性

- 4 个具有静态优先级和 ID 的电平触发中断源
- 映射到存储器的配置和状态寄存器
- 支持机器模式和用户模式下的中断
- 64 位定时器，支持中断溢出标志
- 软件中断

1.7.3 软件中断

机器模式和用户模式软件中断源分别通过设置或清除存储器映射寄存器 MSIP 或 USIP 来控制。

若需在某一模式下使能核心级别的中断，则必须设置 mie/uie CSR 中的 MSIE/USIE。

若需获取某一模式下的中断等待状态，可读取 mip/uip CSR 中的 MSIP/USIP。

请注意，ID 为 0 的用户模式软件中断在 mideleg CSR 中的相应位默认置 1，即中断 0 默认委托给用户模式，但可以通过切换该位的值在机器模式下使用该中断。同样地，也可以设置机器模式软件中断对应的位，以便在用户模式下使用。

1.7.4 定时器计数器与中断

MTIME 是 CPU 本地的机器模式定时器计数器寄存器，是一个可读可写的 64 位存储器映射寄存器。定时器计数器可以通过设置 MTIMECTL 中的 MTCE 位来使能。

UTIME 是一个只读存储器映射寄存器，可从用户模式读取定时器计数器，不过读取的值总是和 MTIME 相同。

若需使能机器模式/用户模式的定时器中断，请设置 MTIMECTL/UTIMECTL 中的 MTIE/UTIE。若需在核心级别使能该中断，还必须设置 mie 中的 MTIE/UTIE。

当 64 位定时器的值超过 MTIMECMP/UTIMECMP 中写入的 64 位比较值时，机器模式/用户模式的中断被触发 (assert)。

若需获取机器模式/用户模式定时器中断的等待状态，请读取 MTIMECTL/UTIMECTL 中的 MTIP/UTIP。

为了在机器模式/用户模式下解除 (de-assert) 定时器中断，必须清除 MTIE/UTIE 位或更新 MTIMECMP/UTIMECMP 寄存器。

若需获取不同模式中断在核心级别的等待状态，可读取 mip/uip 中的 MTIP/UTIP。

64 位定时器计数器溢出时，MTIMECTL/UTIMECTL 中的 MTOF/UTOF 会立即置位，在处理完溢出情况后，可以将其清除。

注意，ID 为 4 的用户模式定时器中断在 mideleg CSR 中的相应位默认置 1，即中断 4 默认委托给用户模式，但可以通过切换该位的值在机器模式下使用该中断。同样地，也可以设置机器模式软件中断对应的位，以便在用户模式下使用。

1.7.5 寄存器列表

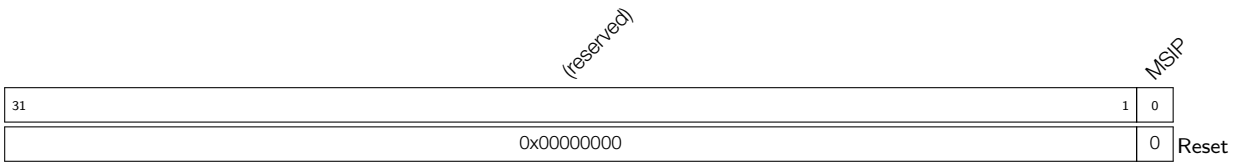
本小节的所有地址均为相对于 CPU 子系统基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的图 4-1。

名称	描述	地址	访问
MSIP	机器模式核心本地软件中断等待寄存器	0x1800	R/W
MTIMECTL	机器模式核心本地定时器中断控制/等待寄存器	0x1804	R/W
MTIME	64 位核心本地定时计数器值寄存器	0x1808	R/W
MTIMECMP	64 位机器模式核心本地定时器比较值寄存器	0x1810	R/W
USIP	用户模式核心本地软件中断等待寄存器	0x1C00	R/W
UTIMECTL	用户模式核心本地定时器中断控制/等待寄存器	0x1C04	R/W
UTIME	64 位只读核心本地定时计数器值寄存器	0x1C08	RO
UTIMECMP	64 位用户模式核心本地定时器比较值寄存器	0x1C10	R/W

1.7.6 寄存器

本小节的所有地址均为相对于 CPU 子系统基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的图 4-1。

Register 1.25. MSIP (0x1800)



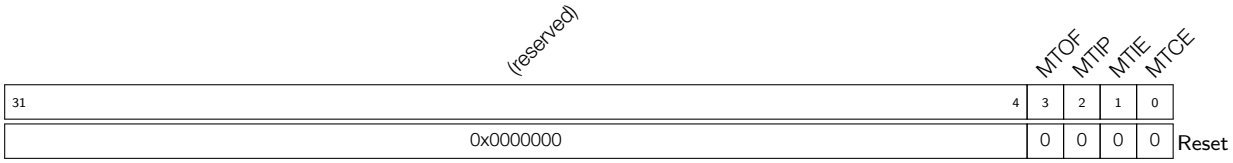
MSIP 配置机器模式软件中断等待状态。

0：不等待

1：等待

(R/W)

Register 1.26. MTIMECTL (0x1804)



MTCE 配置是否使能 CLINT 定时器计数器。

0：不使能

1：使能

(R/W)

MTIE 写 1 使能机器模式定时器中断。(R/W)

MTIP 配置机器模式定时器中断等待状态。

0：不等待

1：等待

(RO)

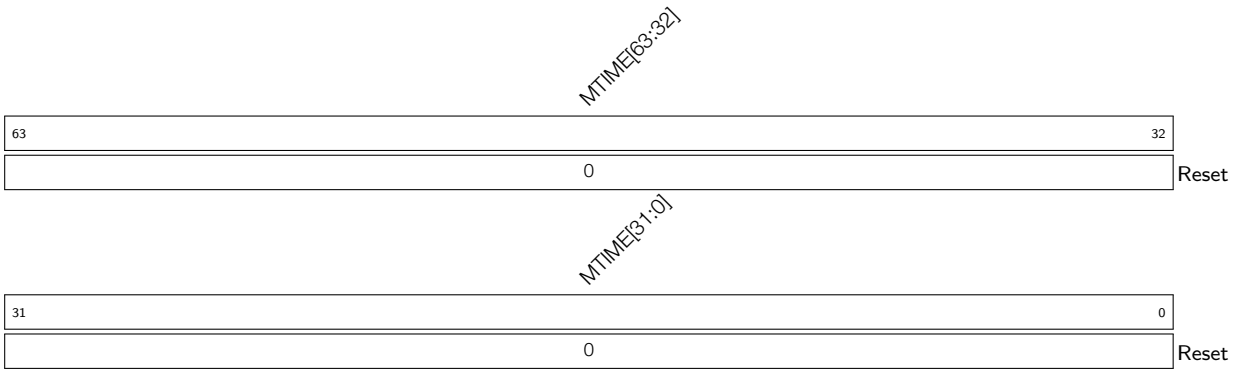
MTOF 配置机器模式定时器是否上溢。

0：不上溢

1：上溢

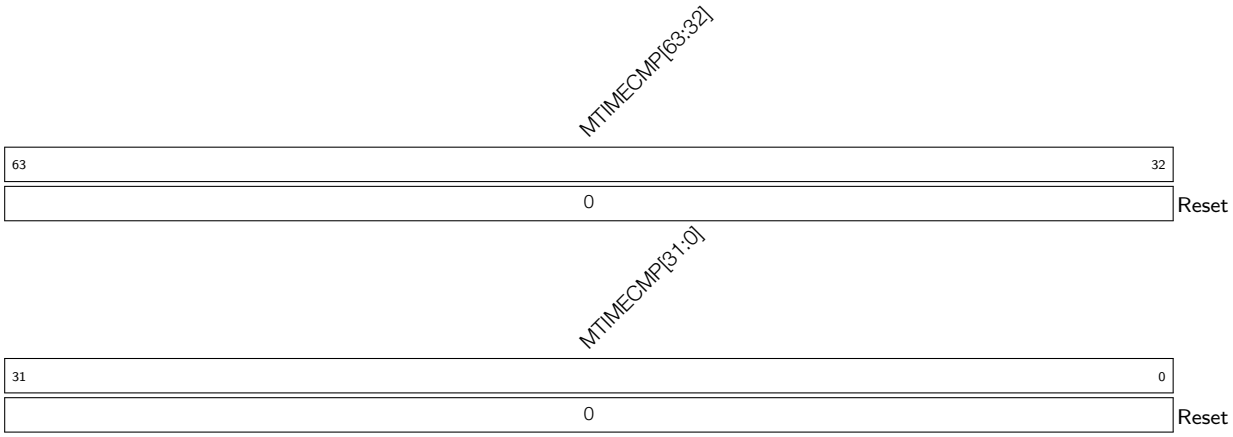
(R/W)

Register 1.27. MTIME (0x1808)



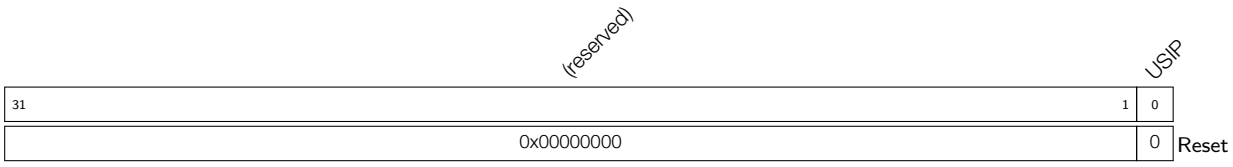
MTIME 配置 64 位 CLINT 定时器计数器数值。(R/W)

Register 1.28. MTIMECMP (0x1810)



MTIMECMP 配置 64 位机器模式定时器比较值。(R/W)

Register 1.29. USIP (0x1C00)



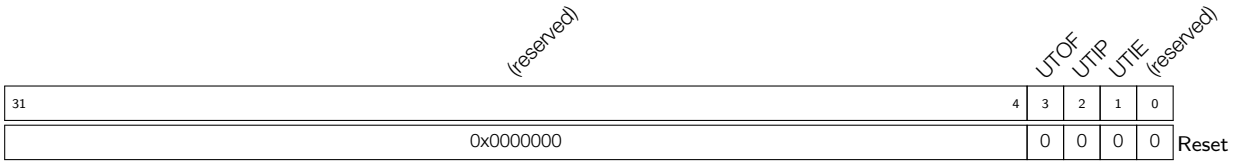
USIP 配置用户模式软件中断等待状态。

0：不等待

1：等待

(R/W)

Register 1.30. UTIMECTL (0x1C04)



UTIE 写 1 使能用户模式定时器中断。(R/W)

UTIP 表示用户模式定时器中断等待状态。(RO)

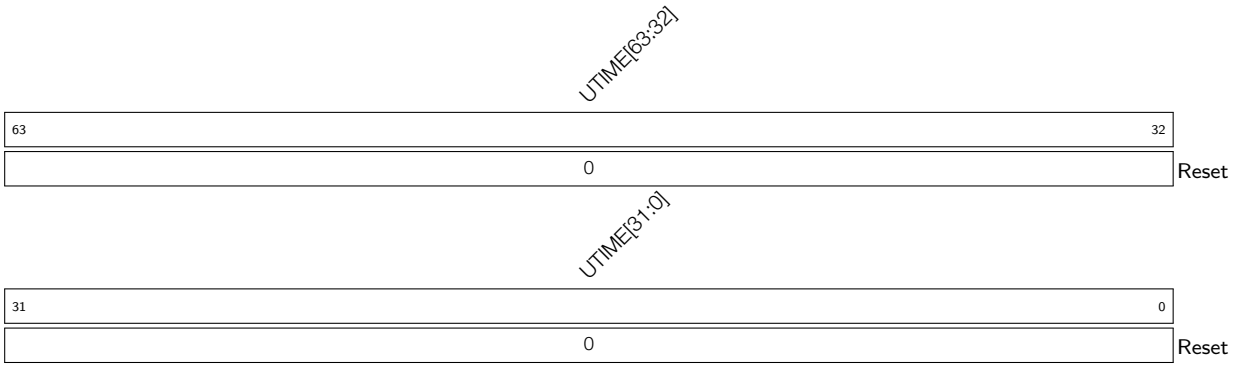
UTOF 表示用户模式定时器是否上溢。

0：不上溢

1：上溢

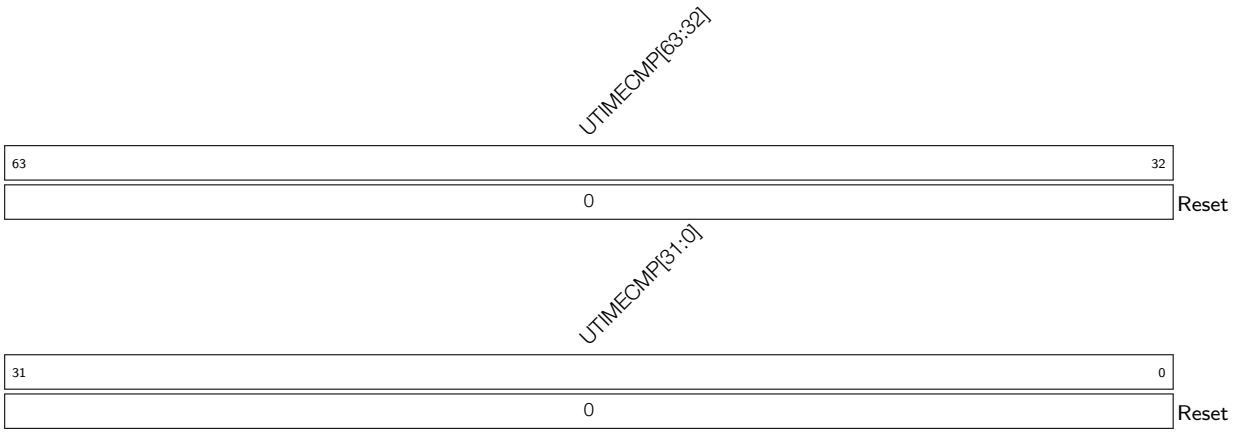
(R/W)

Register 1.31. UTIME (0x1C08)



UTIME 表示 64 位只读 CLINT 定时器计数器数值。(RO)

Register 1.32. UTIMECMP (0x1C10)



UTIMECMP 配置 64 位用户模式定时器比较值。(R/W)

1.8 物理存储器保护

1.8.1 概述

CPU 内核包含一个物理存储器保护单元，完全遵循 RISC-V 指令集手册 V1.10 第二卷 “特权架构”，可以供软件设置存储器访问特权（读、写、执行权限）。除了标准的 PMP 检查之外，CPU 内核还实现了自定义物理存储器属性 (PMA) 检查器，从而根据预定义的属性提供额外的权限检查。

1.8.2 特性

PMP 单元用于控制对物理存储器的访问。它支持 16 个区域，最小可以对 4 个字节大小的区域进行权限设定，支持的最大 NAPOT 范围是 4 GB。

1.8.3 功能描述

软件可以设置 PMP 单元的配置和地址寄存器，以保存错误并确保安全执行。配置 PMP CSR 可使能 PMP 单元，但只能在机器模式下进行配置，一旦 PMP 单元被使能，则将根据 16 个 pmpcfgX 和 pmpaddrX 寄存器（见 寄存器列表）中的配置值对用户模式下的所有存储器访问进行写入、读取、访问权限检查。

默认情况下，PMP 允许机器模式下的所有存储器访问，而撤销用户模式下的所有访问。这意味着必须通过 pmpcfg 和 pmpaddr 寄存器（见寄存器列表）设置用户模式可以访问的地址范围和有效权限，以确保访问成功。但是在机器模式下没有此要求，因为机器模式下默认 PMP 允许所有访问。如果在机器模式下也需要 PMP 检查，则软件可以将所需 PMP 表项的锁定位置位来使能权限检查。锁定位一旦置位，就只能通过 CPU 复位被清零。

如果从存储器区域提取指令而没有执行权限，则会在处理器级别生成异常，并且在 mcause CSR 中异常原因被设置为指令访问错误。同样，任何没有有效读/写权限的读写访问都将生成异常，并且 mcause 会更新为读取存储器访问错误或写入存储器访问错误。如果发生存储器读写异常，则存储器访问地址会更新到 mtval CSR 中。

1.8.4 寄存器列表

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

名称	描述	地址	访问
pmpcfg0	物理存储器保护配置寄存器	0x3A0	R/W
pmpcfg1	物理存储器保护配置寄存器	0x3A1	R/W
pmpcfg2	物理存储器保护配置寄存器	0x3A2	R/W
pmpcfg3	物理存储器保护配置寄存器	0x3A3	R/W
pmpaddr0	物理存储器保护地址寄存器	0x3B0	R/W
pmpaddr1	物理存储器保护地址寄存器	0x3B1	R/W
pmpaddr2	物理存储器保护地址寄存器	0x3B2	R/W
pmpaddr3	物理存储器保护地址寄存器	0x3B3	R/W
pmpaddr4	物理存储器保护地址寄存器	0x3B4	R/W
pmpaddr5	物理存储器保护地址寄存器	0x3B5	R/W
pmpaddr6	物理存储器保护地址寄存器	0x3B6	R/W
pmpaddr7	物理存储器保护地址寄存器	0x3B7	R/W
pmpaddr8	物理存储器保护地址寄存器	0x3B8	R/W
pmpaddr9	物理存储器保护地址寄存器	0x3B9	R/W

名称	描述	地址	访问
pmpaddr10	物理存储器保护地址寄存器	0x3BA	R/W
pmpaddr11	物理存储器保护地址寄存器	0x3BB	R/W
pmpaddr12	物理存储器保护地址寄存器	0x3BC	R/W
pmpaddr13	物理存储器保护地址寄存器	0x3BD	R/W
pmpaddr14	物理存储器保护地址寄存器	0x3BE	R/W
pmpaddr15	物理存储器保护地址寄存器	0x3BF	R/W

1.8.5 寄存器

PMP 单元实现了 RISC-V 指令集手册 V1.10 第二卷“特权架构”中定义的所有 pmpcfg0-3 和 pmpaddr0-15 CSR。

1.9 物理存储器属性检查器 (PMAC)

1.9.1 概述

CPU 核心还实现了自定义物理内存属性检查器 (PMAC)，可根据 CSR 预先定义的存储器类型提供额外的权限检查。

1.9.2 特性

PMAC 支持以下特性：

- 可为指定的存储器空间配置存储器类型
- 可为指定的存储器空间配置属性

1.9.3 功能描述

软件可以设置 PMAC 单元的配置和地址寄存器，避免因访问无效存储器区域造成的错误。PMAC CSR 只能在机器模式下进行配置。一旦 PMAC 被使能，则将根据 16 个 pma_cfgX 和 pma_addrX 寄存器（见Register Summary）中的配置值对各个模式下的所有存储器访问进行写入、读取、访问权限检查。访问被设置为无效的存储器的表项将导致获取错误或读写错误异常。

PMA 检查相关的的异常生成及错误处理与 PMP 检查类似。如果从配置为 null 或无效的存储器区域提取指令，则会在处理器级别生成异常，并且在 mcause CSR 中将异常原因设置为指令访问错误。同样，读写任何 null 或无效的存储器区域都将生成异常，并且 mcause 会更新为读取存储器访问错误或写入存储器访问错误。如果发生存储器读写异常，则存储器访问地址会更新到 mtval CSR 中。对于配置为有效存储器区域的 PMA 检查表项，处理方式与 PMP 检查相同。

如果软件想要关闭对 PMAC 寄存器的写入，可使用表项的锁定位。一旦任何 pma_cfgX 寄存器中的锁定位被设置，相应的 pma_cfgX 和 pma_addrX 寄存器就不能被写入，除非 CPU 复位。

PMAC CSR 中还提供了一个 4 位的域来定义存储器区域的属性。这些位不为 CPU 内核使用，而是基于地址匹配在读写接口上作为边带信号，供 cache 控制器用于其内部操作。

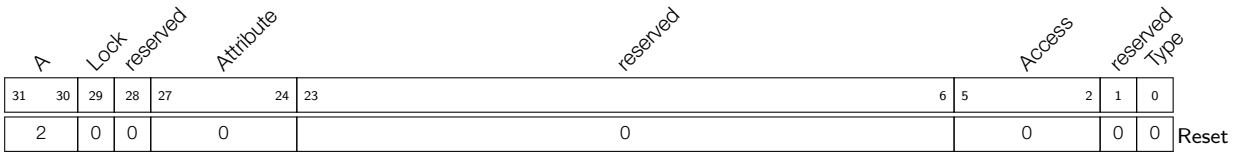
1.9.4 寄存器列表

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

名称	描述	地址	访问
pma_cfg0	物理存储器属性配置寄存器	0xBC0	R/W
pma_cfg1	物理存储器属性配置寄存器	0xBC1	R/W
pma_cfg2	物理存储器属性配置寄存器	0xBC2	R/W
pma_cfg3	物理存储器属性配置寄存器	0xBC3	R/W
pma_cfg4	物理存储器属性配置寄存器	0xBC4	R/W
pma_cfg5	物理存储器属性配置寄存器	0xBC5	R/W
pma_cfg6	物理存储器属性配置寄存器	0xBC6	R/W
pma_cfg7	物理存储器属性配置寄存器	0xBC7	R/W
pma_cfg8	物理存储器属性配置寄存器	0xBC8	R/W
pma_cfg9	物理存储器属性配置寄存器	0xBC9	R/W
pma_cfg10	物理存储器属性配置寄存器	0xBCA	R/W
pma_cfg11	物理存储器属性配置寄存器	0xBCB	R/W
pma_cfg12	物理存储器属性配置寄存器	0xBCC	R/W
pma_cfg13	物理存储器属性配置寄存器	0xBCD	R/W
pma_cfg14	物理存储器属性配置寄存器	0xBCE	R/W
pma_cfg15	物理存储器属性配置寄存器	0xBCF	R/W
pma_addr0	物理存储器属性地址寄存器	0xBD0	R/W
pma_addr1	物理存储器属性地址寄存器	0xBD1	R/W
pma_addr2	物理存储器属性地址寄存器	0xBD2	R/W
pma_addr3	物理存储器属性地址寄存器	0xBD3	R/W
pma_addr4	物理存储器属性地址寄存器	0xBD4	R/W
pma_addr5	物理存储器属性地址寄存器	0xBD5	R/W
pma_addr6	物理存储器属性地址寄存器	0xBD6	R/W
pma_addr7	物理存储器属性地址寄存器	0xBD7	R/W
pma_addr8	物理存储器属性地址寄存器	0xBD8	R/W
pma_addr9	物理存储器属性地址寄存器	0xBD9	R/W
pma_addr10	物理存储器属性地址寄存器	0xBDA	R/W
pma_addr11	物理存储器属性地址寄存器	0xBDB	R/W
pma_addr12	物理存储器属性地址寄存器	0xBDC	R/W
pma_addr13	物理存储器属性地址寄存器	0xBDD	R/W
pma_addr14	物理存储器属性地址寄存器	0xBDE	R/W
pma_addr15	物理存储器属性地址寄存器	0xBDF	R/W

1.9.5 寄存器

Register 1.33. pma_cfgX (0xBC0-0xBCF)



A 配置地址类型。功能与 pmpcfg 寄存器的 A 字段相同。

0x0：OFF

0x1：TOR

0x2：NA4

0x3：NAPOT

(R/W)

Lock 配置是否锁定对应的 pma_cfgX 和 pma_addrX。

0：不锁定

1：锁定，即撤销写入 pma_cfgX 和 pma_addrX 的权限

只有 CPU 复位才能解锁。(R/W)

Attribute 配置要在 DRAM 属性端口上驱动的值。(R/W)

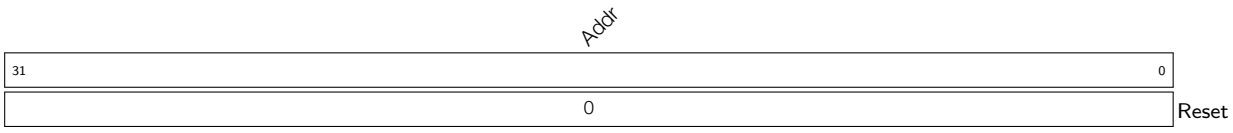
Type 配置区域类型。

0x0：存储器区域无效（RWX 访问将被视为 0，即使配置为 1）

0x1：存储器区域有效（RWX 访问可用）

(R/W)

Register 1.34. pma_addrX (0xBD0-0xBDF)



Addr 配置地址。功能与 pmpaddr 寄存器相同。(R/W)

1.10 调试

1.10.1 概述

本节介绍如何调试和测试在 ESP-RISC-V CPU 内核上运行的软件。调试功能由标准 JTAG 管脚提供，并符合 RISC-V 外部调试支持规范 v0.13。

图 1-2 为外部调试系统架构图。

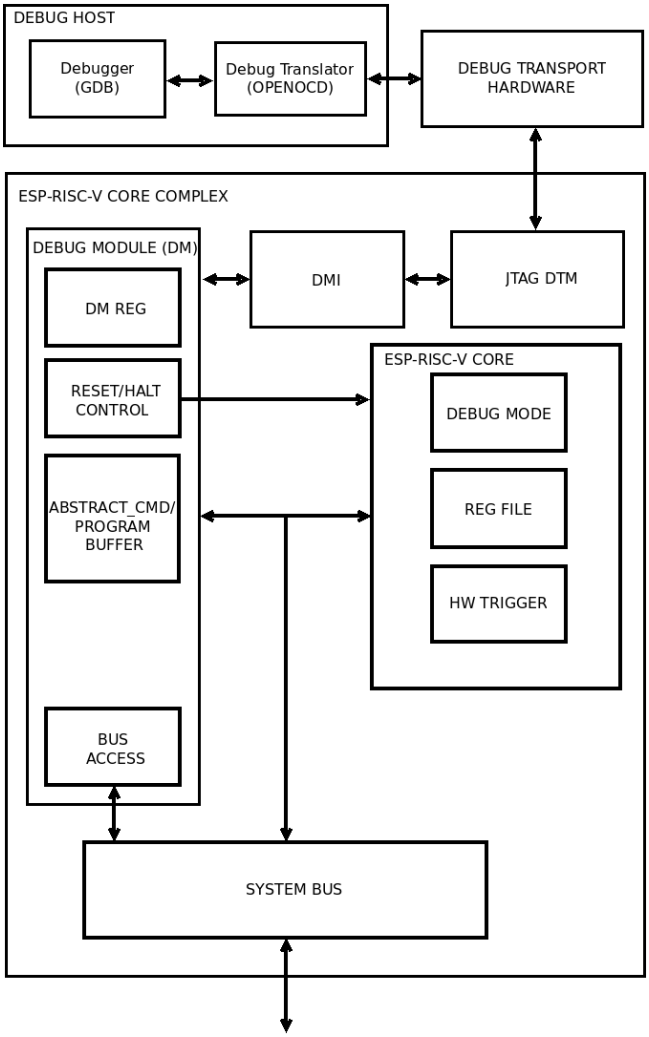


图 1-2. 调试系统架构

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

DM 允许调试器暂停选中的内核。抽象命令提供对 GPR（通用寄存器）的访问。程序缓冲区允许调试器在内核上执行任意代码，从而读取 CPU 内核的其他运行状态。CPU 内核的其他运行状态也可以由其他抽象命令读取。ESP-RISC-V 内核带有一个支持 4 个触发器的触发器模块。当满足触发条件时，内核将自发暂停并通知调试模块。

系统总线访问的 block 无需使用 RISC-V 内核即可访问存储器和外设寄存器。

1.10.2 特性

基础调试功能具有以下特性：

- 向调试器提供有关实现的必要信息
- 支持暂停和恢复 CPU 内核
- CPU 内核寄存器（包括 CSR）可以由调试器读取/写入
- CPU 可以从复位后执行的第一条指令开始就被调试
- 可以通过调试器复位 CPU 内核
- 可以通过软件断点指令暂停 CPU
- 硬件单步调试
- 通过程序缓冲区在暂停的 CPU 中执行任意指令。支持 16 字的程序缓冲区
- 支持系统总线的字对齐地址访问
- 支持 4 个硬件触发器（可用作断点/观察点），具体见章节 1.11

1.10.3 功能描述

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

1.10.4 JTAG 控制

标准的 JTAG 接口是 DTM 访问 DM 的唯一途径。硬件提供两种 JTAG 方式：PAD_to_JTAG 和 USB_to_JTAG。

- PAD_to_JTAG：指 JTAG 信号源来自芯片 IO 管脚
- USB_to_JTAG：指 JTAG 信号源来自 USB Serial/JTAG 控制器

使用哪种 JTAG 方法取决于许多因素，具体配置方法如下表所示。

暂时关闭 JTAG 3,4	EFUSE_DIS_USB_JTAG 4	EFUSE_DIS_USB_SERIAL_JTAG 4	EFUSE_DIS_PAD_JTAG 4	EFUSE_JTAG_SEL_ENABLE 4	Strapping 管脚 GPIO25 5	USB_to_JTAG 状态	PAD_to_JTAG 状态
0	0	0	0	0	x 2	可用 1	不可用 1
0	0	0	0	1	1	可用	不可用
0	0	0	0	1	0	不可用	可用
0	0	1	0	x	x	不可用	可用
0	1	0	0	x	x	不可用	可用
0	1	1	0	x	x	不可用	可用
0	0	0	1	x	x	可用	不可用
0	0	1	1	x	x	不可用	不可用
0	1	0	1	x	x	不可用	不可用
0	1	1	1	x	x	不可用	不可用
1	x	x	x	x	x	不可用	不可用

说明：

1. 可用：相应的 JTAG 功能可用
   不可用：相应的 JTAG 功能不可用
2. x：无关项
3. “暂时关闭 JTAG”表示如果 EFUSE_SOFT_DIS_JTAG[2:0] 中有偶数位“1”，则开启 JTAG 功能（表中对应值为 1），否则，JTAG 功能被关闭（表中对应的值为 0）。但是，在 ESP32-H2 HMAC 加速器的某些特殊条件下，即使 EFUSE_SOFT_DIS_JTAG[2:0] 中有奇数位“1”，JTAG 功能也可能被开启。有关 HMAC 如何影响 JTAG 功能的信息，请参阅 HMAC Accelerator 章节。
4. 有关 eFuse 的更多信息，请参考 eFuse Controller 章节。
5. 有关 strapping 管脚 GPIO25 的更多信息，请参考 Chip Boot Control 章节。

1.10.5 寄存器列表

下表列出了 ESP-RISC-V 内核支持的调试 CSR。

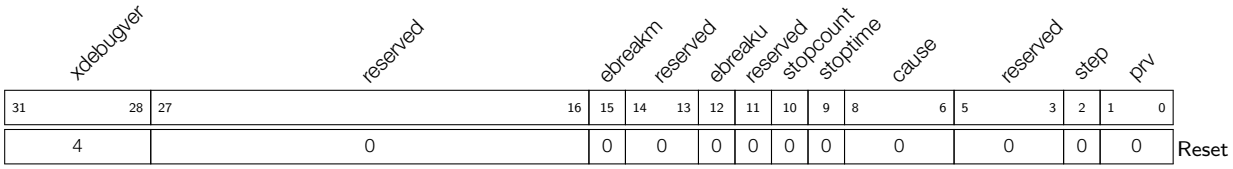
名称	描述	地址	访问
dcsr	调试控制和状态寄存器	0x7B0	R/W
dpc	调试 PC 寄存器	0x7B1	R/W
dscratch0	调试暂存寄存器 0	0x7B2	R/W
dscratch1	调试暂存寄存器 1	0x7B3	R/W

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

1.10.6 寄存器

以下是 ESP-RISC-V 内核支持的调试 CSR 的详细描述。

Register 1.35. dcsr (0x7B0)



xdebugver 表示调试版本。

4：存在外部调试支持

(RO)

ebreakm 置位后，机器模式中的 ebreak 指令进入调试模式。(R/W)

ebreaku 置位后，用户/程序模式中的 ebreak 指令进入调试模式。(R/W)

stopcount 此性能没有实现，调试器会始终读出 0。(RO)

stoptime 此性能没有实现，调试器会始终读出 0。(RO)

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

1：执行了一条 ebreak 指令（优先级 3）

2：触发模块引起暂停（优先级 4）

3：haltreq 被置位（优先级 2）

4：step 被置位导致 CPU 单步执行（优先级 1）

其他值：保留，供日后使用

(RO)

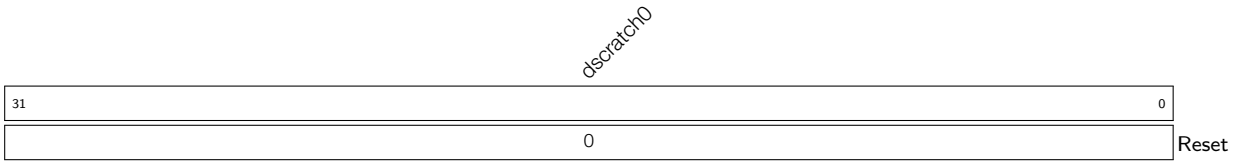
step 当被置位且不处于调试模式时，内核将仅执行单个指令，然后进入调试模式。当该位置 1 时，中断被**使能** *。如果指令由于异常而未能完成，则内核将在执行异常处理程序之前立即进入调试模式，并置位相应的异常寄存器。(R/W)

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

Register 1.36. dpc (0x7B1)

dpc 进入调试模式后，dpc 将写入遇到异常的指令的虚拟地址。恢复执行时，CPU 内核的 PC 将更新为 dpc 保存的虚拟地址。调试器可以写入 dpc 配置 CPU 恢复执行的位置。(R/W)

Register 1.37. dscratch0 (0x7B2)



dscratch0 供调试模块内部使用。(R/W)

Register 1.38. dscratch1 (0x7B3)

dscratch1 供调试模块内部使用。(R/W)

1.11 硬件触发器

1.11.1 特性

硬件触发器模块提供了断点和观察点功能，供调试使用。硬件触发器具有以下特性：

- 4 个独立触发单元
- 每个单元都可以配置为匹配程序计数器的地址或存储器访问地址
- 可以通过引起断点异常来抢占执行
- 可以暂停执行并将控制权转移给调试器
- 支持 NAPOT（2 的幂次方对齐区域）地址编码

1.11.2 功能描述

硬件触发器模块提供了 4 个 CSR，见 寄存器列表。其中，tdata1 和 tdata2 是抽象 CSR，也就是说它们是用于访问某个触发单元中的内部寄存器的影子寄存器，一次访问一个触发单元。

要选择特定的触发单元，需要将相应的编号 (0-3) 写入 tselect CSR。当写入有效数值时，抽象 CSR tdata1 和 tdata2 将自动匹配该触发单元的内部寄存器。每个触发单元都有两个内部寄存器，即 mcontrol 和 maddress，它们分别与 tdata1 和 tdata2 匹配。

向 tselect 写入超过最大编号的数值时会导致该数值被裁剪为最大的编号，此编号可以被读回。这个特性可用于枚举初始化期间或使用调试器时可用的触发器。

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

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

通过写入 mcontrol 的 action 域，可以将每个触发单元配置为引起断点异常或进入调试模式。该域只能从调试器写入，因此默认情况下，触发器（如果启用）将引起断点异常。

每个触发单元的 mcontrol 都有一个 hit 域。在 CPU 暂停或进入异常后，通过读取该域可以查明是否是触发单元触发了。触发器触发后该域会立即被置位，但在恢复操作之前必须被手动清零，不过不清零不会影响正常执行。

每个触发单元仅支持地址匹配，该地址可以是存储器访问地址，也可以是指令的虚拟地址。通过写入所选触发单元的 maddress (tdata2) CSR，可以指定区域的地址和大小。大于 1 个字节的区域大小通过 NAPOT 编码（见表 1-10）指定，并通过置位 mcontrol 中 match 域来使能。注意，根据定义，NAPOT 编码地址的起始地址与区域大小对齐（即，是区域大小的整数倍）。

表 1-10. NAPOT 编码的 maddress

maddress(31-0)	起始地址	大小（字节）
aaa...aaaaaaaaa0	aaa...aaaaaaaaa0	2
aaa...aaaaaaaa01	aaa...aaaaaaaa00	4
aaa...aaaaaaa011	aaa...aaaaaaa000	8
aaa...aaaaaa0111	aaa...aaaaaa0000	16

....		
a01...1111111111	a00...0000000000	2^{31}

tcontrol CSR 对所有触发单元都是通用的。在机器模式下，当陷阱处理程序运行时，该寄存器可用于阻止触发器重复引起异常，而且默认会禁用 ISR 内部的断点异常，但是，出于调试目的，可以在进入 ISR 之前手动使能断点异常。如果将触发器配置为进入调试模式，则此 CSR 不相关。

1.11.3 触发执行流程

当触发器触发引起 hart 暂停并进入调试模式时 (action = 1)：

- dpc 在解码阶段被设置为当前程序计数器 (PC)
- dcsr 的 cause 域被设置为 2，表示暂停是由于触发器触发引起
- 与触发的触发器对应的 hit 域被置位

当触发器触发引起 hart 进入陷阱时 (action = 0)：

- mepc 在解码阶段被设置为当前 PC
- mcause 被设置为 3，即断点异常
- mpte 被设置为陷阱发生之前的 mte 的值
- mte 被设置为 0
- 与触发的触发器对应的 hit 域被置位

说明：如果两个触发器同时触发，一个 *action = 0*，*action = 1*，则 *hart* 会暂停并进入调试模式。

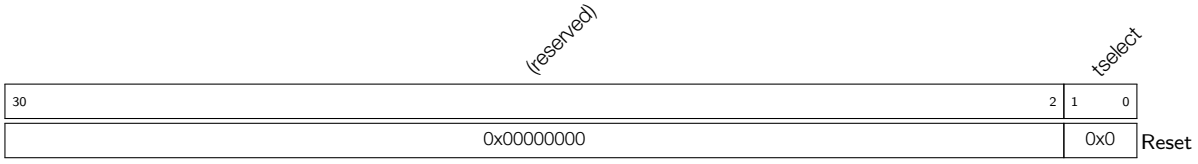
1.11.4 寄存器列表

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

名称	描述	地址	访问
tselect	触发器选择寄存器	0x7A0	R/W
tdata1	触发器抽象数据寄存器 1	0x7A1	R/W
tdata2	触发器抽象数据寄存器 2	0x7A2	R/W
tcontrol	全局触发器控制寄存器	0x7A5	R/W

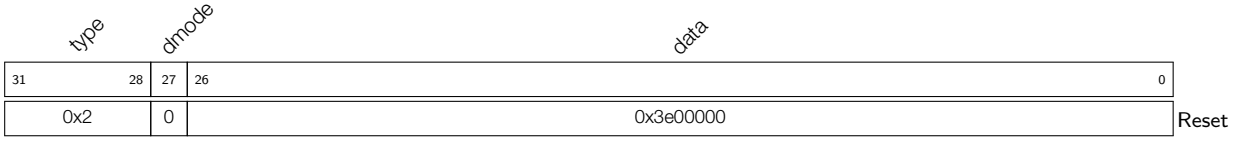
1.11.5 寄存器

Register 1.39. tselect (0x7A0)



tselect 配置触发器单元编号 (0-3)。(R/W)

Register 1.40. tdata1 (0x7A1)



type 表示触发器类型。仅支持匹配类型 (0x2)，此域保留。(RO)

dmode 如果某触发器正在被调试器使用，则此域置为 1。

0：在调试模式和机器模式下都能写入 tdata1 和 tdata2

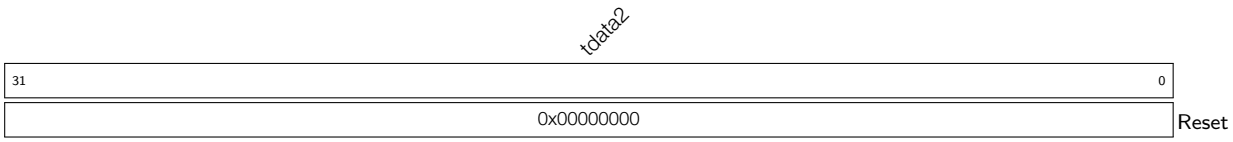
1：只有在调试模式下才能写入 tdata1 和 tdata2，其他模式下的写操作将被忽略

注意：仅支持调试模式下的写操作

(R/W)

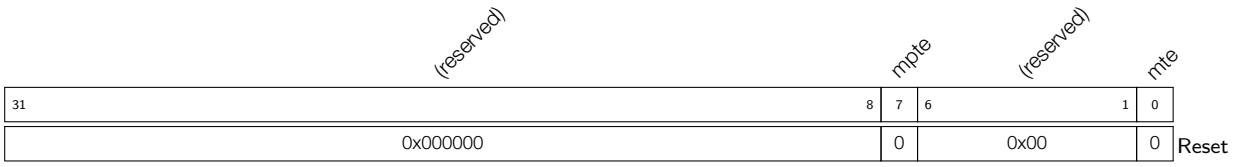
data 配置抽象 tdata1 的内容。由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 mcontrol 的域。(R/W)

Register 1.41. tdata2 (0x7A2)



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

Register 1.42. tcontrol (0x7A5)



mpte 配置是否使能机器模式下前一个触发器。

当 CPU 在机器模式下进入异常，mte 的值会自动写入此域。

当 CPU 执行 MRET，此域的值会返回 mte，此域变为 0。

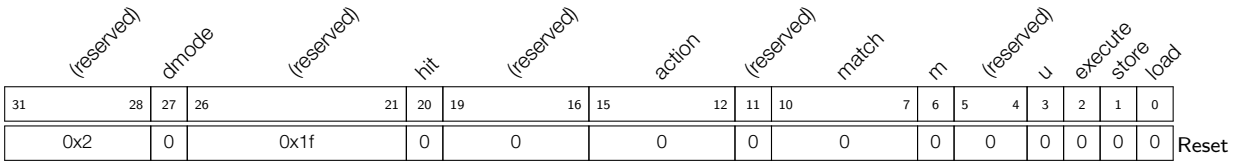
(R/W)

mte 配置是否使能机器模式下触发器。当 CPU 在机器模式下进入异常，此域的值会自动写入 mpte，然后此域变为 0，并且 action=0 的触发器被全局禁用。

当 CPU 执行 MRET，mpte 的值会自动返回此域。

(R/W)

Register 1.43. mcontrol (0x7A1)



dmode 与 tdata1 的 dmode 一致。(RW *)

hit 如果选定的触发器之前触发过，则此域为 1。此域必须手动清零。(R/W)

action 配置选定的触发器在触发时进行以下操作。有效选项为：

0x0：引起断点异常

0x1：进入调试模式（仅当 dmode = 1 时有效）

注意：写入无效数值会导致此域变为默认值 0x0。

(R/W)

match 配置触发器进行数据/指令地址的匹配操作。有效选项为：

0x0：严格字节匹配，即与访问中某个字节对应的地址必须严格匹配 maddress 的值

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

注意：写入超过最大值的数值会被裁剪为最大值 0x1。

(R/W)

m 置位使选定的触发器在机器模式下操作。(R/W)

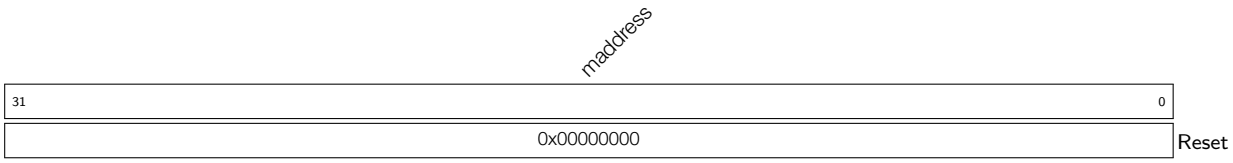
u 置位使选定的触发器在用户模式下操作。(R/W)

execute 置位使选定的触发器在 CPU 执行具有匹配的虚拟地址的指令之前触发。(R/W)

store 置位使选定的触发器在 CPU 执行具有匹配的数据地址的存储器写操作之前触发。(R/W)

load 置位使选定的触发器在 CPU 执行具有匹配的数据地址的存储器读操作之前触发。(R/W)

Register 1.44. maddress (0x7A2)



maddress 配置选定的触发器执行匹配操作时使用的地址。当 mcontrol 中的 match=1 时由 NAPOT 解码。(R/W)

1.12 追踪

1.12.1 概述

为了支持非侵入式软件调试，CPU 内核提供了指令追踪接口，为离线调试提供相关信息。追踪编码器 (Trace Encoder) 模块可通过指令追踪接口获取这些信息，并将其压缩后存储在分配的存储器中。软件解码器可直接从存储器中读取这些信息，无需中断 CPU 内核，便可重新生成 CPU 内核实际执行的程序。

1.12.2 特性

CPU 内核支持指令追踪功能，并按照 RISC-V 处理器追踪规范 v1.0 (RISC-V Processor Trace v1.0) 中的规定向追踪编码器提供以下信息：

- 退役指令数量
- 异常和中断是否发生及其原因、陷阱值
- hart 当前的特权级别
- 跳转、分支和返回退役指令的指令类型
- 程序计数器更改之前和之后退役指令的地址

1.12.3 功能描述

ESP-RISC-V CPU 核心实现了强制指令 delta 追踪，也称为分支追踪。它的工作原理是通过发送程序获取的 delta 信息来追踪从已知起始地址开始的执行情况。delta 通常由跳转、调用、返回、分支类型的指令、中断以及异常引入。所有此类 delta 以及其他详细信息，如原因和实际指令/地址，都通过高带宽指令追踪接口从 CPU 输出给追踪编码器，追踪编码器对这些信息进行操作，将信息压缩后存入存储器，供解码器离线调试。有关编码的更多信息，请参阅章节 2 *RISC-V 追踪编码器 (TRACE)*。

CPU 没有任何内部寄存器来控制指令追踪接口，所有相关控制寄存器都在 2 *RISC-V 追踪编码器 (TRACE)* 模块中。

1.13 专用 IO

1.13.1 概述

GPIO 通常是一个 APB 外围设备，这意味着对输出的更改和对输入的读取可能会卡在写入缓冲区或其他传输之后，并且通常速度较慢，因为 APB 总线的运行速度通常低于 CPU。作为替代方案，CPU 内核实现了 I/O 处理器特定的 CPU 寄存器 (CSR)，直接连接到 GPIO 矩阵或 IO 管脚。这些寄存器只需一条指令即可访问，因此十分迅速。

1.13.2 特性

- 8 个直接映射到 GPIO 的专用 IO
- 驱动输出端口无延迟
- 读取输入值有两个 CPU 周期的延迟

1.13.3 功能描述

CPU 核心有一组 8 个输入和输出（管脚值 + 管脚输出使能），这些输入和输出端口直接连接到 GPIO 矩阵，可通过 GPIO 矩阵映射到最上层的管脚。更多相关信息请参考 6 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)*。

CPU 自定义了 3 个 CSR:

- GPIO_IN 只读，表示输入值。
- GPIO_OUT 可读可写，表示 GPIO 输出值。
- GPIO_OEN 可读可写，表示 GPIO 输出使能状态，控制管脚方向，拉高表示管脚配置为输出模式，拉低表示配置为输入模式。

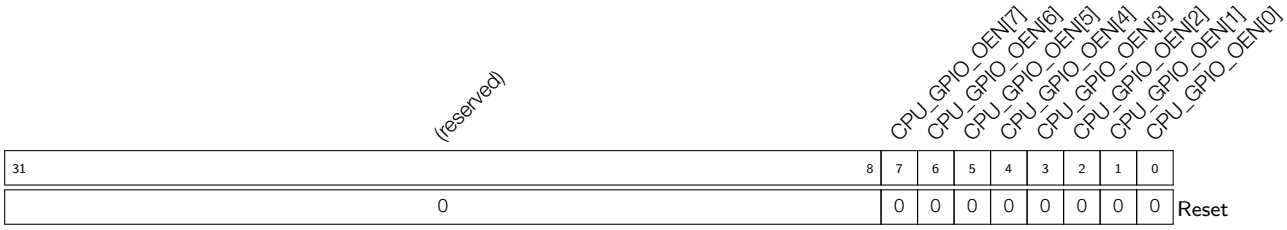
1.13.4 寄存器列表

下面是在内核中自定义的专用 IO CSR 的列表。

名称	描述	地址	访问
cpu_gpio_oen	GPIO 输出使能寄存器	0x803	R/W
cpu_gpio_in	GPIO 读输入值寄存器	0x804	RO
cpu_gpio_out	GPIO 写输出值寄存器	0x805	R/W

1.13.5 寄存器

Register 1.45. cpu_gpio_oen (0x803)



CPU_GPIO_OEN 配置是否使能 GPIOn (n=0 ~ 21) 输出。CPU_GPIO_OEN[7:0] 分别对应章节 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 中表 6-2 里的 cpu_gpio_out_oen[7:0] 输出使能信号。CPU_GPIO_OEN 的值与 cpu_gpio_out_oen 的值对应。

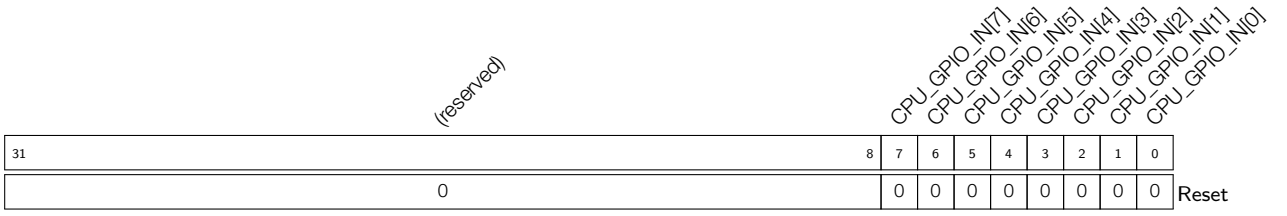
此寄存器是 CPU_GPIO_OUT 的使能寄存器。

0：关闭 GPIO 输出

1：使能 GPIO 输出

(R/W)

Register 1.46. cpu_gpio_in (0x804)

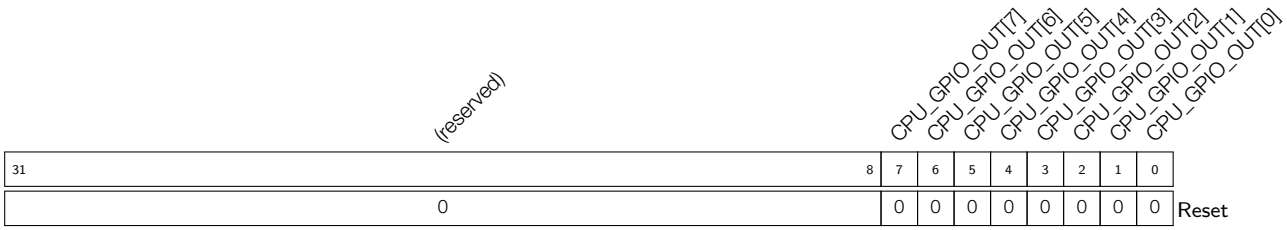


CPU_GPIO_IN 表示 SoC GPIOn (n=0 ~ 21) 的输入值（1 为高电平，0 为低电平）。CPU_GPIO_IN[7:0] 分别对应章节 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 中表 6-2 里的 cpu_gpio_in[7:0] 输入信号。

CPU_GPIO_IN[7:0] 只能通过 GPIO 交换矩阵映射到 GPIO。详细描述请参考章节 6.4。

(RO)

Register 1.47. cpu_gpio_out (0x805)



CPU_GPIO_OUT 向 SoC GPIOn (n=0～21) 写输出值（1 为高电平，0 为低电平）。CPU_GPIO_OEN 置位时，写输出值才有效。

CPU_GPIO_OUT[7:0] 分别对应章节 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 中表 6-2 里的 cpu_gpio_out[7:0] 输出信号。

CPU_GPIO_OUT[7:0] 只能通过 GPIO 交换矩阵映射到 GPIO。详细描述请参考章节 6.5。

(R/W)

1.14 原子 (A) 扩展

1.14.1 概述

对原子 (A) 扩展的支持符合 RISC-V 指令集手册 V2.2 第一卷 “非特权架构”，重点是保证向前推进，即功能上避免出现任何数据存储器被无限期锁定的情况。

原子指令目前忽略 aq（获取）和 rl（释放）位，因为这些位对于目前这种可以保证存储器排序的架构无关。

1.14.2 功能描述

1.14.2.1 加载保留字 (LR.W) 指令

LR.W 指令只是锁定正在读取的 32 位对齐存储器地址。一旦某个 4 字节的存储器空间被锁定，它将保持锁定状态，其他 hart 无法访问该区域，直到在执行过程中遇到以下任何一种情况：

- 任何读操作
- 任何写操作
- 任何中断/异常
- 向后跳转/向后分支跳转
- JALR 指令
- ECALL/EBREAK/MRET/URET 指令
- FENCE/FENCE.I 指令
- 调试模式
- 临界区超过 64 字节
- SC.W 指令中的数据地址与 LR.W 指令中的不匹配

如果发生上述任何一种情况，存储器锁将立刻被释放。如果遇到 SC.W 指令，则存储器锁不会立即被释放，但最终会被释放，有关具体如何释放的信息，请参考章节 1.14.2.2。

如果地址未对齐，则会导致异常 mcause = 6。

1.14.2.2 条件存入字 (SC.W) 指令

SC.W 指令首先检查内存锁是否仍然有效，以及地址是否与上一条 LR.W 指令指定的地址相同，只有以上条件同时满足时，SC.W 指令才会对存储器执行写操作，并且在收到存储器发出的写操作完成确认信息后立即释放锁。

如果锁已失效（失效原因见章节 1.14.2.1），SC.W 指令将在目标寄存器 rd 中设置失败代码，目前只能设置为 1。

如果地址未对齐，则会导致异常 mcause = 6。

1.14.2.3 AMO 指令

AMO (原子性存储器操作) 指令分三步执行：

1. 从 rs1 给定的存储器地址读取数据，并将数据保存到目标寄存器 rd。
2. 将 rd 和 rs2 中的数据进行运算。
3. 将上面第 2 步得到的结果写入 rs1 给定的存储器地址。

AMO 运算共有 9 种类型：SWAP、ADD、AND、OR、XOR、MAX、MIN、MAXU 和 MINU。

在整个过程中，存储器地址保持锁定状态，其他 hart 无法访问。如果遇到未对齐的地址，则会导致异常 mcause = 6。

AMO 操作的读/写访问 (PMP/PMA) 都在第一步中进行检查。若发生错误，则 mcause = 7。

2 RISC-V 追踪编码器 (TRACE)

ESP32-H2 的 CPU 通过追踪编码器 (Trace Encoder) 实现了对指令追踪接口的支持。追踪编码器连接 CPU 的指令追踪接口，将信息压缩成更小的数据包，存入内部 SRAM（详见章节 4 系统和存储器）中。

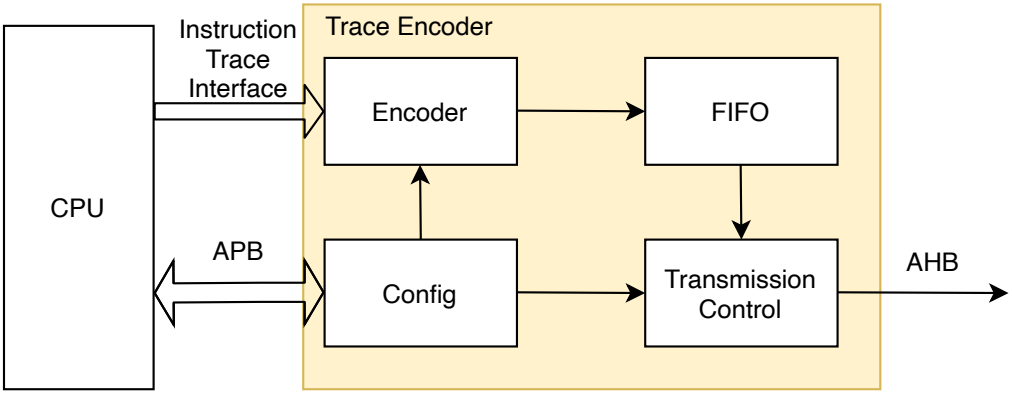


图 2-1. 追踪编码器概述图

2.1 术语

为了更好地说明 RISC-V 追踪编码器的功能，本章使用了以下术语。

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

2.2 介绍

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

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

此时，可使用指令追踪来追踪程序执行。

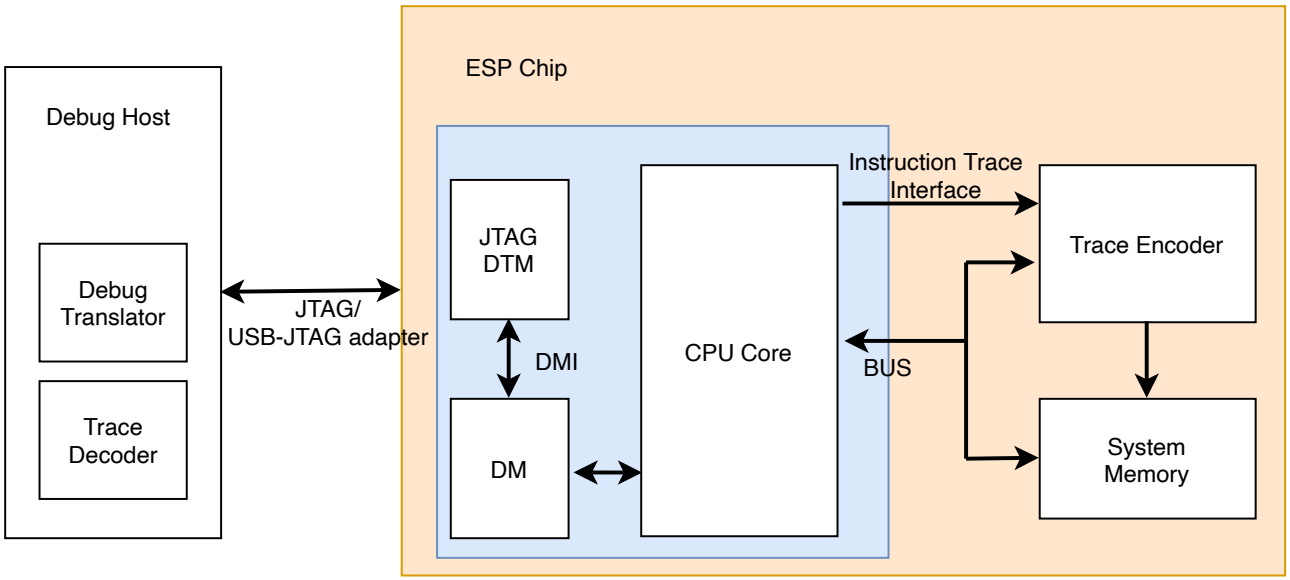


图 2-2. 指令追踪概览

图 2-2 为指令追踪的原理图：

- CPU 核有一个指令追踪接口，可以输出 CPU 执行指令的信息，包括指令地址、指令类型等。更多关于 ESP32-H2 CPU 指令追踪接口的信息，可参考章节 1 *ESP-RISC-V CPU*。
- 追踪编码器连接 CPU 的指令追踪接口，将信息压缩成带宽更小的数据包，存入系统存储器中。
- 调试器可以将系统存储器中的追踪数据包通过 JTAG 或 USB Serial/JTAG 导出，然后用解码器解压缩、重建程序的执行流。追踪解码器通常是外部 PC 上的软件，在获取追踪数据包后，根据在 hart 上运行的程序二进制数据，重建程序的指令流。解码可以离线进行，也可以在 hart 执行时实时进行。

本章节将主要介绍 ESP32-H2 追踪编码器的具体实现。

2.3 特性

- 兼容 RISC-V Processor Trace 1.0（RISC-V 追踪规范 v1.0），实现的参数详见表 2-2
- 支持任意地址范围用作追踪存储器
- 具有两个同步模式：
  - 同步计数器按包计数
  - 同步计数器按周期计数
- 支持丢包状态标识
- 支持丢包后自动重启
- 写追踪存储器时支持循环和非循环模式
- 具有两个中断：
  - 包的大小超过配置的存储器空间时触发中断
  - 丢包时触发中断
- 具有 128 × 8 位 FIFO，用于缓存数据包

表 2-2. 追踪编码器参数

参数名称	值	描述
arch_p	0	追踪器符合 RISC-V 追踪规范初始版本
bpred_size_p	0	不支持分支预测模式
cache_size_p	0	不支持跳转目标缓存模式
call_counter_size_p	0	不支持隐式返回模式
ctype_width_p	0	输出包中不包含上下文信息
context_width_p	0	输出包中不包含上下文信息
ecause_width_p	5	异常原因宽度
ecause_choice_p	0	不支持特权和异常原因多选
f0s_width_p	0	不支持格式 0 数据包
filter_context_p	0	不支持筛选功能
filter_excint_p	0	
filter_privilege_p	0	
filter_tval_p	0	
iaddress_lsb_p	1	支持压缩指令
iaddress_width_p	32	指令总线为 32 位
iretire_width_p	1	iretire 总线宽度
ilastsize_width_p	0	ilastsize 总线宽度
itype_width_p	3	itype 总线宽度
noncontext_p	1	在 te_inst 数据包中去掉上下文信息
privilege_width_p	1	仅支持机器和用户模式
retires_p	1	每个块退役指令的最大数量
return_stack_size_p	0	不支持隐式返回模式
sijump_p	0	不支持根据顺序推测跳转
taken_branches_p	1	每个时钟周期仅一条指令退役
impdef_width_p	0	未实现

上述参数的具体描述，详见 RISC-V Processor Trace Version 1.0 > 章节 Parameters and Discovery。

2.4 架构概览

如图 2-1 所示，追踪编码器包含一个编码器、一个 FIFO、一个寄存器配置模块和一个发送控制模块。

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

发送控制模块通过 AHB 总线将 FIFO 中的数据写入内部 SRAM。

FIFO 的深度为 128，宽度为 8 位。存储器写入带宽不足时，FIFO 可能会溢出，发生丢包。如果发生丢包，编码器会发送一个包通知丢包情况，之后停止工作，直至 FIFO 为空。

2.5 功能描述

2.5.1 同步

为保障追踪的有效性，追踪过程中必须定期同步。同步可以通过发送完整的指令地址实现。同步计数器的值达到 TRACE_RESYNC_PROLONGED_REG 寄存器的 TRACE_RESYNC_MODE 位的值时，编码器会发送一个同

步数据包（格式 3 子格式 0，见章节 2.6.3.1）。

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

- 0：同步计数器按周期计数
- 1：同步计数器按数据包计数

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

2.5.2 锚定

由于数据包的长度不定，为在写入存储器时分清数据包之间的边界，ESP32-H2 通过在数据包之间插入 0 ，从而能够发现数据包头：

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

2.5.3 写存储器模式

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

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

2.5.4 自动重启

因 FIFO 溢出而丢包时，编码器将停止工作，需要通过软件再次开启。如果 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位为 1，FIFO 清空后编码器会立即自动重启，无需软件重启。

如果开启了自动重启，即使编码器已经停止追踪，也可能会自动重启。因此，要关闭编码器，必须先清除 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位从而关闭自动重启。

2.6 编码器输出数据包

本节主要介绍 ESP32-H2 追踪编码器输出的数据包格式。ESP32-H2 仅实现了必需的指令 delta 追踪，不支持下列可选功能：

- delta 地址模式（支持运行时间配置模式）
- 上下文 (context) 信息和所有上下文相关的字段
- 可选的边带信号 (Sideband Signals)
- 调试模块触发输出

更多关于上述功能的细节，请参考 RISC-V Processor Trace 1.0。

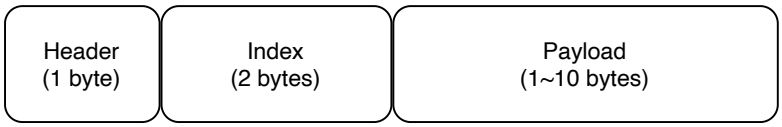


图 2-3. 追踪数据包格式

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

2.6.1 头部

头部为 1 个字节，格式如表 2-3 所示。

表 2-3. 头部格式

字段	位	描述	值
length	5	数据包的总长度	4~13
placeholder	3	保留	0

2.6.2 索引

索引为 2 个字节，格式如表 2-4 所示。

表 2-4. 索引格式

字段	位	描述	值
index	16	每个数据包的索引	0~65536

2.6.3 有效载荷

有效载荷的宽度为 1 至 10 字节。

2.6.3.1 格式 3

格式 3 的数据包用于同步和发送支持信息。格式 3 有四种子格式，ESP32-H2 仅支持 3 种。

格式 3 子格式 0 - 同步

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

表 2-5. 格式 3 子格式 0

字段名	位	描述
format	2	11 (sync)：同步
subformat	2	00 (start)：开始追踪，或开始同步

字段名	位	描述
branch	1	0：地址指向分支指令，且分支跳转 1：指令非分支指令，或分支没有跳转
privilege	1	所追踪指令的特权等级
address	31	指令的完整地址。字段值需左移 1 位，以复原字节地址
sign_extend	3	保留

格式 3 子格式 1 - 异常

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

表 2-6. 格式 3 子格式 1

字段名	位	描述
format	2	11 (sync)：同步
subformat	2	01 (exception)：异常原因和陷阱处理程序的地址
branch	1	0：地址指向分支指令，且分支跳转 1：指令非分支指令，或分支没有跳转
privilege	1	所追踪指令的特权等级
ecause	5	异常原因
interrupt	1	中断
address	31	指令的完整地址。地址必须向左偏移 1 位，以复原字节地址
tvalepc	32	ecause 为 2、interrupt 为 0 时是异常地址，否则为陷阱值
sign_extend	6	保留

格式 3 子格式 3 - 支持

该格式的数据包包含解码器需要的辅助信息，在追踪结束时发出。有效载荷的长度为 1 个字节。

表 2-7. 格式 3 子格式 3

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

2.6.3.2 格式 2

该格式的数据包仅包含指令地址，在必须注明指令地址时使用，包中没有分支信息。有效载荷的长度为 5 个字节。

表 2-8. 格式 2

字段名	位	描述
format	2	10 (addr-only)：无分支信息
address	31	指令的完整地址
notify	1	ESP32-H2 不支持通知功能，因此该位永远和地址的最高有效位一致
updiscon	1	若该位的值和 notify 的值不同，则表示指令的地址指向一个无法推测的不连续的目标地址，且该地址为异常、特权变更或同步之前的最后一条指令
sign_extend	5	

2.6.3.3 格式 1

该格式的数据包包含分支信息，在必须注明分支信息（比如分支映射已满时）或在上一个包后出现一个以上分支指令、必须注明指令地址时使用。仅支持完整地址模式。

格式 1，有地址时 branch_map 字段

有效载荷长度不定。

表 2-9. 格式 1，有地址

字段名	位	描述
format	2	01：包含分支信息
branches	5	该字段的值决定了 branch_map 的位宽（branch_map 的有效位数量），具体如下： • 0：（该格式中此值无效） • 1：1 位 • 2-3：3 位 • 4-7：7 位 • 8-15：15 位 • 16-31：31 位 比如，若 branches = 12，则 branch_map 长度为 15 位，低 12 位有效。
branch_map	不定	表示分支是否跳转的位数组。位 0 对应最早执行的分支指令。每个位的含义如下： • 0：分支已跳转 • 1：分支未跳转 该字段的位宽不定，取决于 branches 字段的值
address	31	指令的完整地址
notify	1	ESP32-H2 不支持通知功能，因此该位永远和地址的最高有效位一致

字段名	位	描述
updiscon	1	若该位的值和 notify 的值不同，则表示指令的地址指向一个无法推测的不连续的目标地址，且该地址为异常、特权变更或同步之前的最后一条指令
sign_extend	不定	该字段的位宽不定，取决于 branches 字段的值，具体如下： • 1：7 位 • 2-3：5 位 • 4-7：1 位 • 8-15：1 位 • 16-32：31 位

格式 1，无地址时 branch_map 字段

有效载荷长度为 5 个字节。

表 2-10. 格式 1，无地址

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

2.7 中断

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

使能追踪编码器中断后，如果要让 CPU 响应中断，需要通过中断矩阵将追踪编码器中断映射到 CPU 中断号，详情请参考章节 9 中断矩阵 *(INTMTX)*。

2.8 编程流程

2.8.1 使能编码器

- 通过 TRACE_MEM_START_ADDR_REG 和 TRACE_MEM_END_ADDR_REG 配置追踪存储器的地址空间
- 向 TRACE_MEM_CURRENT_ADDR_UPDATE 写 1，将 TRACE_MEM_CURRENT_ADDR_REG 的值更新为 TRACE_MEM_START_ADDR_REG 的值
- （可选）通过 TRACE_TRIGGER_REG 寄存器的 TRACE_MEM_LOOP 位配置写存储器的模式

  - 0：非循环模式
  - 1：循环模式（默认）
- 通过 TRACE_RESYNC_PROLONGED_REG 寄存器的 TRACE_RESYNC_MODE 位配置同步模式
  - 0：同步计数器按周期计数（默认）
  - 1：同步计数器按数据包计数
- （可选）通过 TRACE_RESYNC_PROLONGED_REG 配置同步计数器的阈值，默认为 128
- （可选）使能中断
  - 向 TRACE_INTR_ENA_REG 寄存器的相应位写 1 开启对应中断
  - 向TRACE_INTR_CLR_REG 寄存器的相应位写 1 清除对应中断
  - 读 TRACE_INTR_RAW_REG 寄存器的值，获取中断状态
- （可选）向 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位写 1，使能自动重启。该功能默认开启
- 向 TRACE_TRIGGER_REG 寄存器的 TRACE_TRIGGER_ON 字段写 1，使能编码器

编码器使能后，会一直追踪 CPU 的指令接口，向追踪存储器写入数据包。

2.8.2 关闭编码器

- 通过清除 TRACE_TRIGGER_REG 寄存器的 TRACE_RESTART_ENA 位关闭自动重启
- 向 TRACE_TRIGGER_REG 寄存器的 TRACE_TRIGGER_OFF 位写 1 关闭编码器
- 读取 TRACE_FIFO_EMPTY 位的值，确认 FIFO 中数据是否都已写入追踪存储器

2.8.3 解码数据包

- 找到解码的第一个地址
  - 读 TRACE_INTR_RAW_REG 寄存器的 TRACE_MEM_FULL_INTR_RAW 位，查看追踪存储器是否已满
    * 如果值为 1，从 TRACE_MEM_START_ADDR_REG 寄存器读取追踪数据包
    * 如果值为 0，且开启了存储器循环模式，则之前的数据包已被覆盖。此时，读取 TRACE_MEM_CURRENT_ADDR_REG 的值，获取最后一个写入的地址，该地址即为解码器处理的第一个地址
- 用解码器解码数据包
  - 解码器从第一个地址开始读取所有数据包，结合二进制文件重建数组
  - 如章节 2.6 所述，编码器向存储器分区边界写 14 个值为 0 的字节。因此出现 14 个值为 0 的字节时，则表示下一个非 0 字节为新数据包的头部

2.9 寄存器列表

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

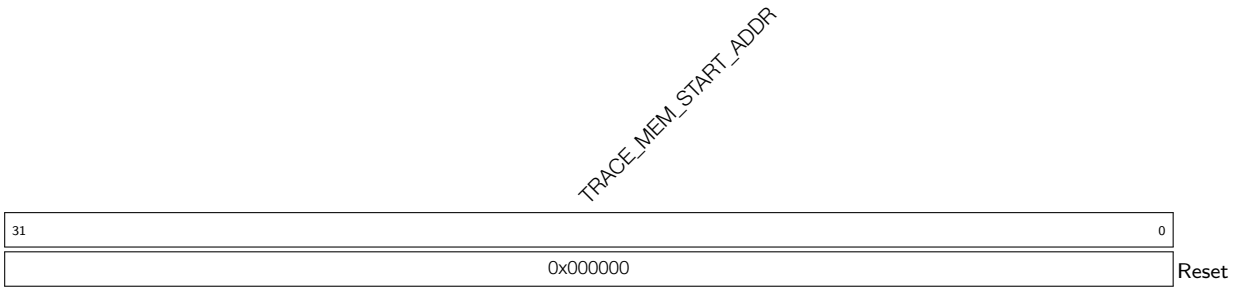
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
存储器配置寄存器			
TRACE_MEM_START_ADDR_REG	存储器起始地址	0x0000	R/W
TRACE_MEM_END_ADDR_REG	存储器结束地址	0x0004	R/W
TRACE_MEM_CURRENT_ADDR_REG	存储器当前地址	0x0008	RO
TRACE_MEM_ADDR_UPDATE_REG	存储器地址更新	0x000C	WT
FIFO 状态寄存器			
TRACE_FIFO_STATUS_REG	FIFO 装填寄存器	0x0010	RO
中断寄存器			
TRACE_INTR_ENA_REG	中断使能寄存器	0x0014	R/W
TRACE_INTR_RAW_REG	中断原始状态寄存器	0x0018	RO
TRACE_INTR_CLR_REG	中断清除寄存器	0x001C	WT
追踪配置寄存器			
TRACE_TRIGGER_REG	追踪使能寄存器	0x0020	varies
TRACE_RESYNC_PROLONGED_REG	再同步配置寄存器	0x0024	R/W
时钟门控控制和配置寄存器			
TRACE_CLOCK_GATE_REG	时钟门控控制寄存器	0x0028	R/W
版本寄存器			
TRACE_DATE_REG	版本控制寄存器	0x03FC	R/W

2.10 寄存器

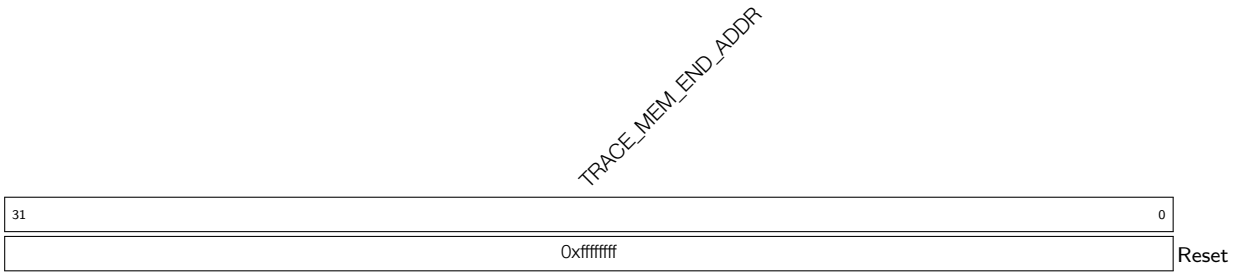
本小节的所有地址均为相对于 RISC-V 追踪编码器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 2.1. TRACE_MEM_START_ADDR_REG (0x0000)



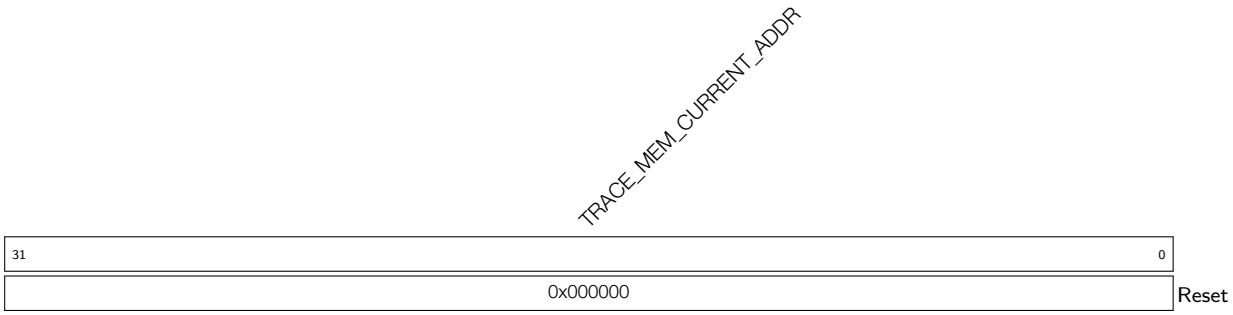
TRACE_MEM_START_ADDR 配置追踪存储器的起始地址。(R/W)

Register 2.2. TRACE_MEM_END_ADDR_REG (0x0004)



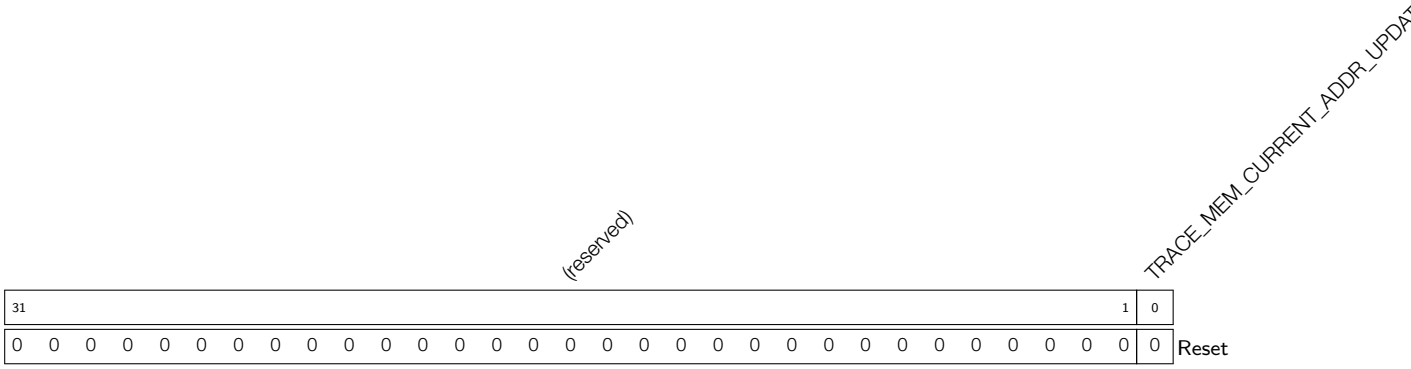
TRACE_MEM_END_ADDR 配置追踪存储器的结束地址。(R/W)

Register 2.3. TRACE_MEM_CURRENT_ADDR_REG (0x0008)



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

Register 2.4. TRACE_MEM_ADDR_UPDATE_REG (0x000C)



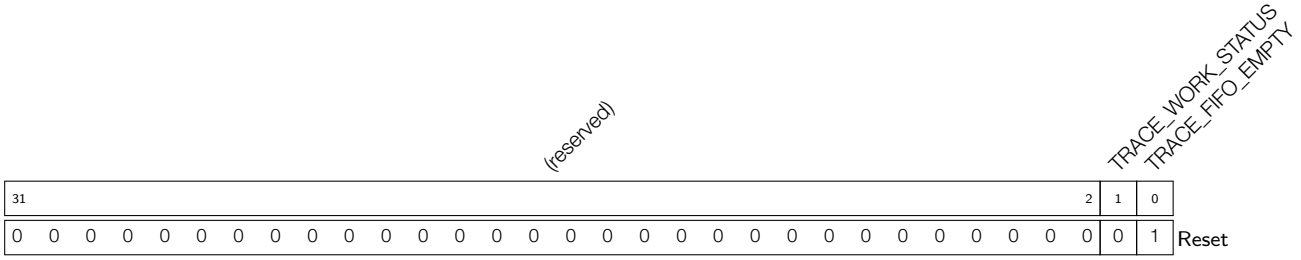
TRACE_MEM_CURRENT_ADDR_UPDATE 配置是否将当前要写入的存储器地址更新为存储器的起始地址。

0：不更新

1：更新

(WT)

Register 2.5. TRACE_FIFO_STATUS_REG (0x0010)



TRACE_FIFO_EMPTY 表示 FIFO 的状态。

0：不为空

1：为空

(RO)

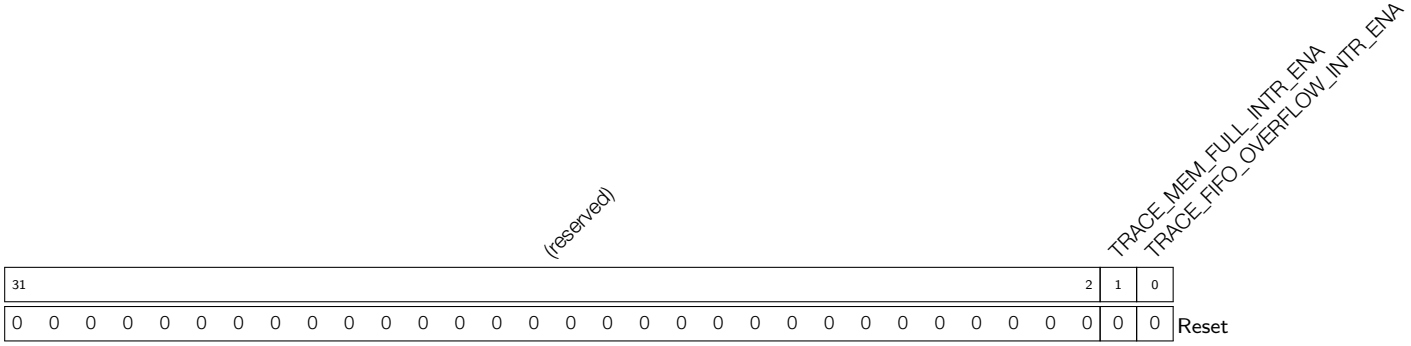
TRACE_WORK_STATUS 表示编码器的状态。

0：不追踪指令

1：追踪指令，输出数据包

(RO)

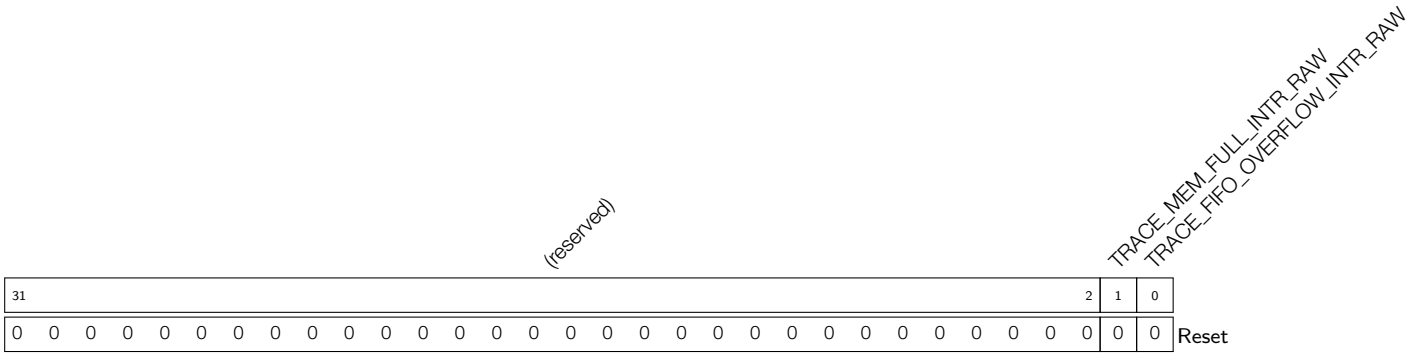
Register 2.6. TRACE_INTR_ENA_REG (0x0014)



TRACE_FIFO_OVERFLOW_INTR_ENA 写 1 使能 TRACE_FIFO_OVERFLOW_INTR。(R/W)

TRACE_MEM_FULL_INTR_ENA 写 1 使能 TRACE_MEM_FULL_INTR。(R/W)

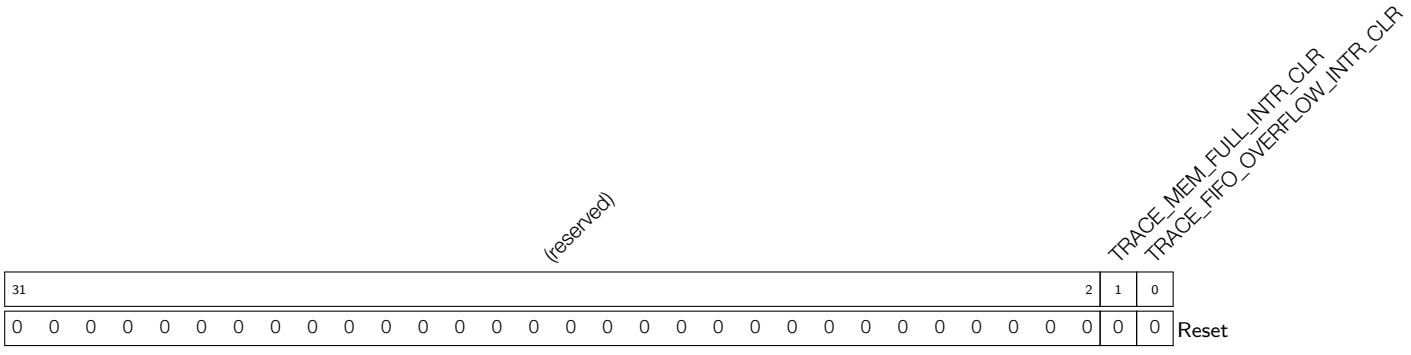
Register 2.7. TRACE_INTR_RAW_REG (0x0018)



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

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

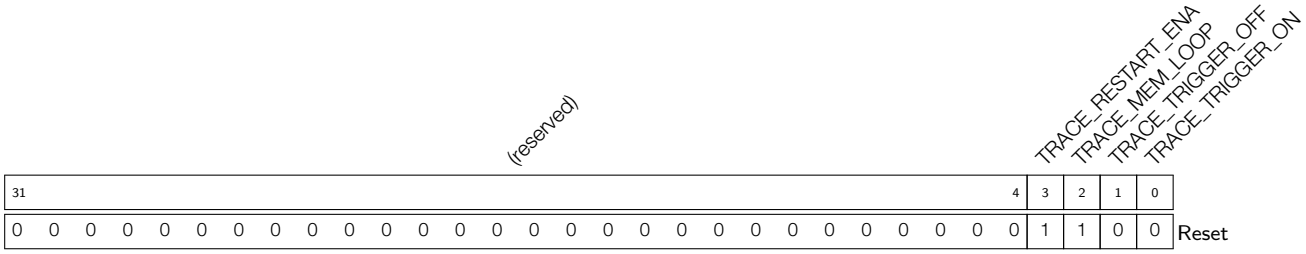
Register 2.8. TRACE_INTR_CLR_REG (0x001C)



TRACE_FIFO_OVERFLOW_INTR_CLR 写 1 清除 TRACE_FIFO_OVERFLOW_INTR。(WT)

TRACE_MEM_FULL_INTR_CLR 写 1 清除 TRACE_MEM_FULL_INTR。(WT)

Register 2.9. TRACE_TRIGGER_REG (0x0020)



TRACE_TRIGGER_ON 配置开启追踪编码器。

0：无效值，没有作用

1：开启

(WT)

TRACE_TRIGGER_OFF 配置关闭追踪编码器。

0：无效值，没有作用

1：关闭

(WT)

TRACE_MEM_LOOP 配置存储器模式。

0：非循环模式

1：循环模式

(R/W)

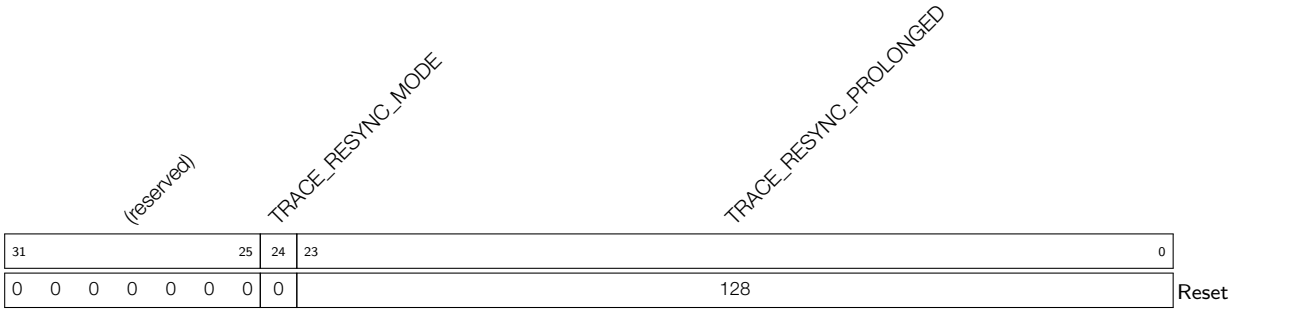
TRACE_RESTART_ENA 配置是否开启编码器自动重启功能。

0：关闭

1：开启

(R/W)

Register 2.10. TRACE_RESYNC_PROLONGED_REG (0x0024)



TRACE_RESYNC_PROLONGED 配置同步计数器的阈值。(R/W)

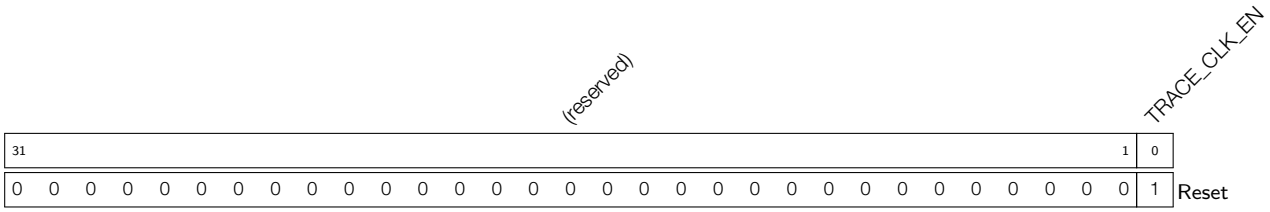
TRACE_RESYNC_MODE 配置同步模式。

0：按包计数同步

1：按周期计数同步

(R/W)

Register 2.11. TRACE_CLOCK_GATE_REG (0x0028)



TRACE_CLK_EN 配置寄存器时钟门控。

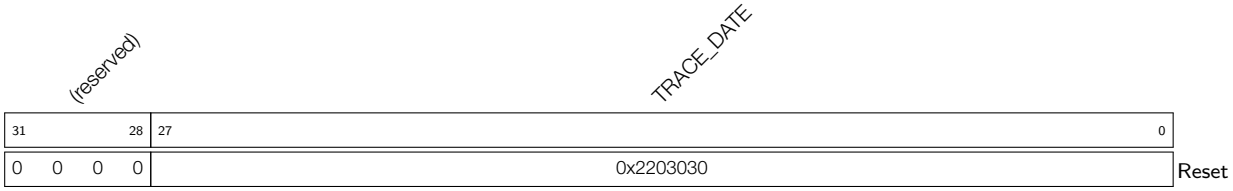
0：仅在应用写寄存器时开启时钟，从而降低功耗

1：一直强制为寄存器开启时钟

该位不影响寄存器访问。

(R/W)

Register 2.12. TRACE_DATE_REG (0x03FC)



TRACE_DATE 版本控制寄存器。(R/W)

3 通用 DMA 控制器 (GDMA)

3.1 概述

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

ESP32-H2 的 GDMA 控制器共有 6 个独立的通道，其中包括 3 个发送通道和 3 个接收通道。这 6 个通道由支持 GDMA 功能的外设共享，用户可以将通道分配给任何支持 DMA 功能的外设。这些外设包括：SPI2、UHCI (UART0/UART1)、I2S、AES、SHA、ADC 和 PARLIO。UART0 与 UART1 共用一个 UHCI 接口。

GDMA 控制器支持通道间固定优先级及轮询仲裁以管理外设不同的带宽需求。

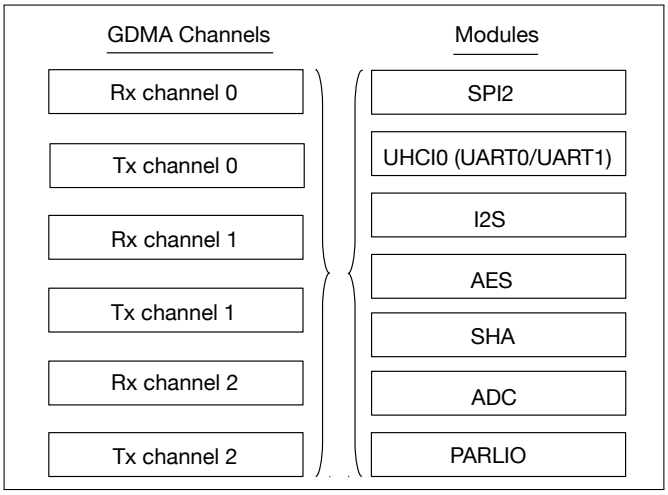


图 3-1. 具有 GDMA 功能的模块和 GDMA 通道

3.2 特性

GDMA 控制器具有以下几个特点：

- AHB 总线架构
- 数据传输以字节为单位，传输数据量可软件编程
- 支持链表
- 访问内部 RAM 时，支持 INCR burst 传输
- GDMA 能够访问的内部 RAM 最大地址空间为 324 KB（320 KB HP SRAM，4 KB LP SRAM）
- 包含 3 个 TX、3 个 RX 通道
- 任一通道支持可配置的外设选择
- 支持通道间固定优先级及轮询仲裁

3.3 架构

ESP32-H2 中所有需要进行高速数据传输的模块都具有 GDMA 功能。GDMA 控制器与 CPU 的数据总线使用相同的地址空间访问内部 RAM。图 3-2 为 GDMA 控制器基本架构图。

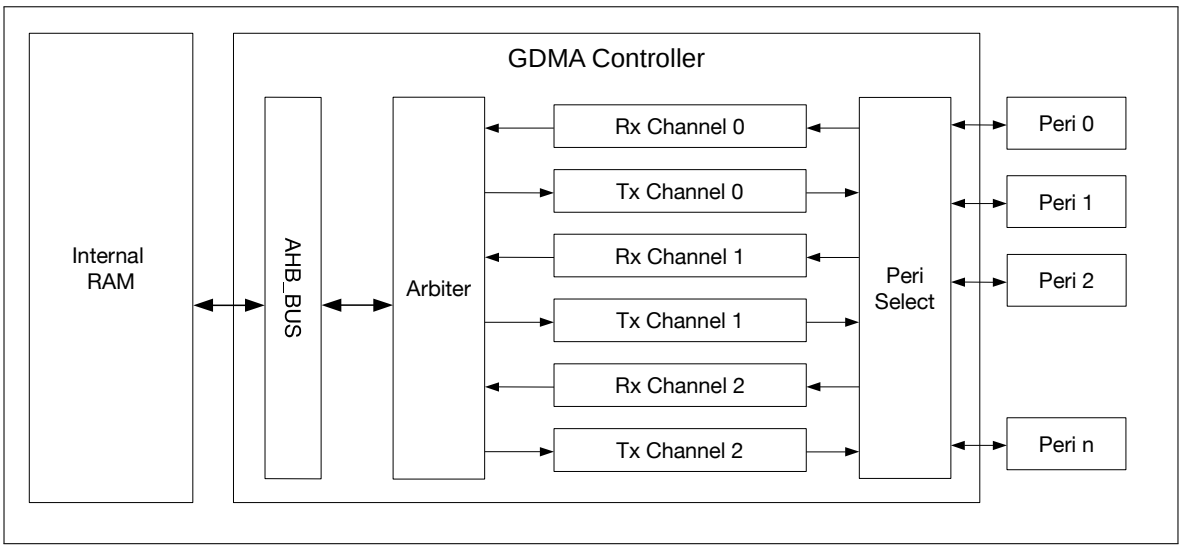


图 3-2. GDMA 控制器的架构

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

GDMA 控制器通过 AHB_BUS 将数据存入内部 RAM 或者将数据从内部 RAM 取出。在通过 AHB_BUS 传输数据之前，GDMA 采用固定优先级的仲裁机制对每个通道的读写请求进行仲裁。内部 RAM 的具体使用范围详见章节 4 系统和存储器。

软件可以通过挂载链表的方式来使用 GDMA 控制器。链表本身须存储在片内 RAM 中，包括 outlink*n* 与 inlink*n*，本文以 *n* 来表示通道号，*n*为 0 ~ 2。GDMA 从片内 RAM 中取得链表，然后根据 outlink*n* 中的内容将相应 RAM 中的数据发送出去，也可根据 inlink*n* 中的内容将接收的数据存入指定 RAM 地址空间。

3.4 功能描述

3.4.1 链表

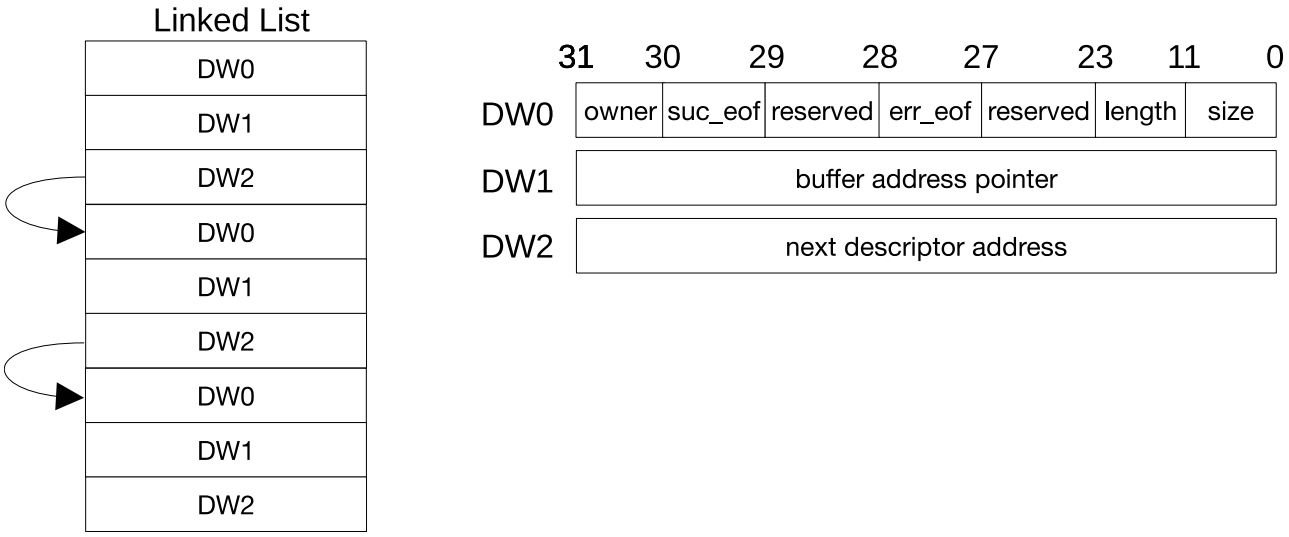


图 3-3. 链表结构图

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

- owner (DW0) [31]：表示当前描述符对应的 buffer 允许的操作者。
  0：允许的操作者为 CPU。
  1：允许的操作者为 GDMA 控制器。
  在 GDMA 使用完该描述符对应的 buffer 后，对于接收描述符，硬件默认会自动将该位清零；对于发送描述符，需要将 GDMA_OUT_AUTO_WRBACK_CH*n* 置 1，硬件才会自动将该位清零。软件也可通过置位 GDMA_OUT_LOOP_TEST_CH*n* 或 GDMA_IN_LOOP_TEST_CH*n* 来关闭硬件自动清零的功能。软件在挂载链表时需要将该位置 1。

  注意：本文以 GDMA_OUT 开头的寄存器对应 TX 通道寄存器，以 GDMA_IN 开头的寄存器对应 RX 通道寄存器。

- suc_eof (DW0) [30]：表示结束标志。
  0：当前描述符不是数据帧或包的最后一个描述符。
  1：当前描述符为数据帧或包的最后一个描述符。
  对于接收描述符，需要软件将该位写 0，硬件会在收完一帧或一个包后将该位置 1。对于发送描述符，需要软件在帧或包的最后一个描述符中的该位置 1。

- Reserved (DW0) [29]：保留。此位为无关项。

- err_eof (DW0) [28]：表示接收结束错误标志。
  0：接收数据没有错误。
  1：接收数据有错误。
  该位只用于 UHCI 或 PARLIO 利用 GDMA 接收数据。对于接收描述符，硬件在收完一帧或一个包并检测到接收数据错误会将该位置 1。

- Reserved (DW0) [27:24]：保留。

- length (DW0) [23:12]：表示当前描述符对应的 buffer 中的有效字节数。对于发送描述符，该段由软件填写，表示从 buffer 中读取数据时需要读取的字节数；对于接收描述符，该段由硬件使用完该 buffer 后或者接收到最后一个数据时自动填写，表示 buffer 中存储的有效字节数。

- size (DW0) [11:0]：表示当前描述符对应的 buffer 容量的字节数。

- buffer address pointer (DW1)：buffer 的地址。

- next descriptor address (DW2)：下一个描述符的地址。如果当前描述符为链表中最后一个描述符 (即 suc_eof = 1)，该值可以为 0。该地址必须指向片内 RAM 的地址空间。

用 GDMA 接收数据时，如果接收数据的长度小于当前描述符指定的 buffer size，那么下一个描述符对应的接收数据不会占用该 buffer 的剩余空间。

3.4.2 外设到存储及存储到外设的数据传输

GDMA 支持存储到外设及外设到存储的数据传输，分别对应 TX 及 RX 功能。TX 通道通过 outlink*n* 实现将指定存储区域中的数据搬运到外设的发送端；RX 通道通过 inlink*n* 实现将外设接收到的数据搬运到指定的存储区域。

每个 RX/TX 通道均可以被配置连接到任意一个支持 GDMA 功能的外设，表 3-1 所示为通过寄存器配置通道与其对应外设的关系。其中“Dummy-*n*”是存储到存储数据传输时可选配的寄存器值。当其中一个通道已经与某一个外设连接时，其他通道将不能配置为与该外设连接。

表 3-1. 配置寄存器与外设选择关系表

GDMA_PERI_IN_SEL_CH*n* GDMA_PERI_OUT_SEL_CH*n*	外设
0	SPI2
1	Dummy-1
2	UHCI
3	I2S
4	Dummy-4
5	Dummy-5
6	AES
7	SHA
8	ADC
9	PARLIO
10 ~ 15	Dummy-10 ~ 15
16 ~ 63	无效值

3.4.3 存储到存储数据传输

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

3.4.4 启动 GDMA

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

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

软件使用 Restart 功能时，需要重写已挂载链表的最后一个描述符，使其第三个字中的内容（即 DW2）指向新链表的首地址；然后置位GDMA_INLINK_RESTART_CH*n* 或者 GDMA_OUTLINK_RESTART_CH*n*（这两个位由硬件自动清零），如图 3-4所示，硬件会在读取已挂载链表的最后一个描述符时，获取新挂载链表的地址，从而继续处理新挂载的链表。

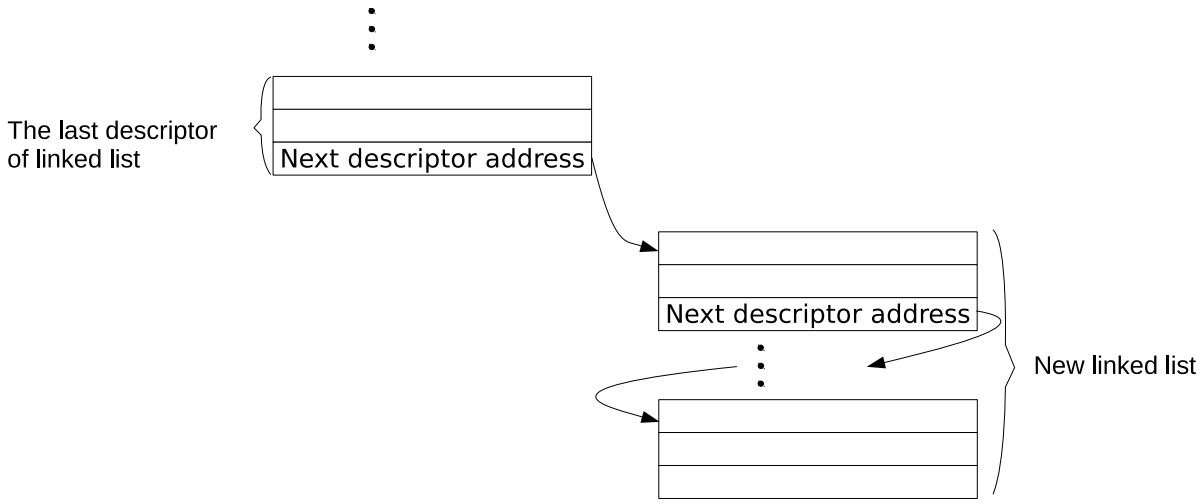


图 3-4. 链表关系图

3.4.5 读链表

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

描述符检查项包括：

- GDMA_IN_CHECK_OWNER_CH*n* 或者 GDMA_OUT_CHECK_OWNER_CH*n* 置 1 时，检查描述符的 owner 位。如果该位为 0，表示当前操作者为 CPU，则不通过检查。GDMA_IN_CHECK_OWNER_CH*n* 或者 GDMA_OUT_CHECK_OWNER_CH*n* 置 0 时，不检查描述符的 owner 位。
- 检查描述符中第二个字指示的地址是否在 0x40800000 ~ 0x4084FFFF 时（请参见本节 3.4.7）。如果在该范围内，则通过检查。否则，不通过检查。

软件在检查到通道描述符错误中断后，需要复位对应的通道，并置位 GDMA_OUTLINK_START_CH*n* 或者 GDMA_INLINK_START_CH*n* 位启动 GDMA。

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

3.4.6 数据传输结束标志

GDMA 通过 EOF 来指示一个数据帧或包传输结束。

发送数据时，置位 GDMA_OUT_TOTAL_EOF_CH*n*_INT_ENA 位使能 GDMA_OUT_TOTAL_EOF_CH*n*_INT 中断，当带有 EOF 标志的描述符对应 buffer 的数据传输完成后，GDMA 会产生该中断。

接收数据时，置位 GDMA_IN_SUC_EOF_CH*n*_INT_ENA 位使能 GDMA_IN_SUC_EOF_CH*n*_INT 中断，表示一帧或一个包数据接收完成。GDMA 还支持中断 GDMA_IN_ERR_CH*n*_EOF_INT，置位 GDMA_IN_ERR_EOF_CH*n*_INT_ENA 使能该中断，表示一个数据帧或包接收完成但该帧或包接收数据有错误。需要注意的是，只有当通道连接的外设为 UHCI 或 PARLIO 时，才支持该中断。

软件在检测到 GDMA_OUT_TOTAL_EOF_CH*n*_INT 或 GDMA_IN_SUC_EOF_CH*n*_INT 中断时，可以记录 GDMA_OUT_EOF_DES_ADDR_CH*n* 或 GDMA_IN_SUC_EOF_DES_ADDR_CH*n* 字段的值，即最后一个描述符的地址。这样，软件可以知道哪些描述符已经被使用并根据需要回收描述符。

注意：本章中提到发送链表描述符的 EOF 为 suc_eof，接收链表描述符的 EOF 可以为 suc_eof 和 err_eof。

3.4.7 访问片内 RAM

GDMA 任意 RX/TX 通道均可以访问片内 RAM，其可访问的片内地址空间为 0x40800000 ~ 0x4084FFFF。为加速数据传输速率，支持突发传输模式。置位 GDMA_IN_DATA_BURST_EN_CH*n* 使能 RX 通道突发传输模式；置位 GDMA_OUT_DATA_BURST_EN_CH*n* 使能 TX 通道突发传输模式。默认情况下，突发传输没有使能。

表 3-2. 链表描述符参数对齐要求

Inlink/Outlink	Burst Mode	Size	Length	Buffer Address Pointer
inlink	0	—	—	—
	1	字对齐	—	字对齐
outlink	0	—	—	—
	1	—	—	—

如表 3-2 所示为访问片内时，链表描述符参数配置对齐要求。

当突发模式没有被使能时，无论是发送链表描述符还是接收链表描述符，其参数 size、length 及 buffer address pointer 均没有字对齐的要求。也就是说，对于一个描述符，在可访问的片内地址空间，GDMA 可以从任意起始地址，读出配置长度的数据，长度取值范围为 1 ~ 4095；或者，将接收到的数据长度（1 ~ 4095）写入任意起始地址开始的连续地址。

当突发模式使能时，对于发送链表描述符，参数 size、length 及 buffer address pointer 均没有字对齐的要求。而对于接收链表描述符，除了参数 length，参数 size 和 buffer address pointer 均需要保持字对齐。

3.4.8 仲裁

为了确保及时响应高速低延迟的外设请求，比如 SPI 等，GDMA 在通道仲裁机制中引入固定优先级，即每个通道的优先级可配置。GDMA 支持 6（0 ~ 5）个等级的优先级。其数值越大，对应的优先级越高，请求响应越及时。当若干个通道配置为相同的优先级时，这几个通道间对请求的响应将采用轮询仲裁机制。

3.4.9 事件任务矩阵功能

在 ESP32-H2 中，GDMA 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GDMA 的 ETM 任务，或者通过 GDMA 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与 GDMA 相关的 ETM 任务和 ETM 事件。

GDMA 可接收的 ETM 任务有：

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

说明：
以上两个 ETM 任务与 CPU 配置 GDMA_INLNIK_START_CH*n*、GDMA_OUTLINK_START_CH*n* 实现的功能相同。当 GDMA_IN_ETM_EN_CH*n* 或 GDMA_OUT_ETM_EN_CH*n* 为 1 时，只有 ETM 可以配置对应方向及对应通道 GDMA 的启动，当 GDMA_IN_ETM_EN_CH*n* 或 GDMA_OUT_ETM_EN_CH*n* 为 0 时，仅可通过 CPU 使能对应的 GDMA。

GDMA 可产生的 ETM 事件有：

- GDMA_EVT_IN_DONE_CH*n*：表示通道 *n* 接收方向链表描述符指向的数据接收完成。
- GDMA_EVT_IN_SUC_EOF_CH*n*：通道 *n* 接收方向 EOF 位为 1 的链表描述符指向的数据接收完成指示。
- GDMA_EVT_IN_FIFO_EMPTY_CH*n*：通道 *n* 接收方向 FIFO 非空到空状态改变指示。
- GDMA_EVT_IN_FIFO_FULL_CH*n*：通道 *n* 接收方向 FIFO 非满到满状态改变指示。
- GDMA_EVT_OUT_DONE_CH*n*：通道 *n* 表示通道 *n* 发送方向链表描述符指向的数据发送完成。
- GDMA_EVT_OUT_SUC_EOF_CH*n*：通道 *n* 发送方向 EOF 位为 1 的链表描述符指向的数据发送完成指示。
- GDMA_EVT_OUT_TOTAL_EOF_CH*n*：通道 *n* 发送方向最后一个并且 EOF 位为 1 的链表描述指向的数据发送符完成指示。
- GDMA_EVT_OUT_FIFO_EMPTY_CH*n*：通道 *n* 发送方向 FIFO 非空到空状态改变指示。
- GDMA_EVT_OUT_FIFO_FULL_CH*n*：通道 *n* 发送方向 FIFO 非满到满状态改变指示。

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

3.5 GDMA 中断

ESP32-H2 中 GDMA 可产生以下 6 个中断信号，并将其发送给中断矩阵 *(INTMTX)*：

- GDMA_IN_CH0_INTR
- GDMA_IN_CH1_INTR
- GDMA_IN_CH2_INTR
- GDMA_OUT_CH0_INTR
- GDMA_OUT_CH1_INTR
- GDMA_OUT_CH2_INTR

以下中断源会触发 GDMA_IN_CH*n*_INTR 中断信号。

- GDMA_IN_DSCR_EMPTY_CH*n*_INT：对于接收通道*n*，当接收链表描述符指向的 buffer size 小于待接收数据长度时触发此中断。
- GDMA_IN_DSCR_ERR_CH*n*_INT：对于接收通道*n*，当接收链表描述符里有错误时触发此中断。
- GDMA_IN_ERR_EOF_CH*n*_INT：对于接收通道*n*，当接收的一个数据帧或包中有错误发生时触发此中断。(该中断只用于外设选择 UHCI（UART0/UART1）或 PARLIO 时)。
- GDMA_IN_SUC_EOF_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成，并且描述符的 suc_eof 为 1 时（即当一个接收链表对应的数据帧或包接收完成时）触发此中断。当一帧或一个包接收完成时触发此中断。
- GDMA_IN_DONE_CH*n*_INT：对于接收通道*n*，当一个接收链表描述符对应的数据接收完成时触发此中断。

以下中断源会触发 GDMA_OUT_CH*n*_INTR 中断信号。

- GDMA_OUT_TOTAL_EOF_CH*n*_INT：对于发送通道*n*，当一个链表（可包含多个链表描述符）对应的所有数据都已发送完成时触发此中断。
- GDMA_OUT_DSCR_ERR_CH*n*_INT：对于发送通道*n*，当发送链表描述符里有错误时触发此中断。

- GDMA_OUT_EOF_CH*n*_INT：对于发送通道*n*，当发送描述符的 EOF 位为 1，并且该描述符对应的数据发送完成时触发此中断。当 GDMA_OUT_EOF_MODE_CH*n*为 0 时，该描述符对应的最后一个数据进入到 GDMA TX 通道时，该中断触发；当 GDMA_OUT_EOF_MODE_CH*n*为 1 时，该描述符对应的最后一个数据从 GDMA TX 通道取出时，该中断触发。
- GDMA_OUT_DONE_CH*n*_INT：对于发送通道*n*，当一个发送链表描述符对应的数据发送完成时触发此中断。

3.6 编程流程

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

3.6.1 GDMA TX 通道配置流程

利用 GDMA 发送数据时，GDMA TX 通道的软件配置流程如下：

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

3.6.2 GDMA RX 通道配置流程

利用 GDMA 接收数据时，GDMA RX 通道的软件配置流程如下：

1. 对寄存器 GDMA_IN_RST_CH*n* 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针。
2. 挂载好接收链表，配置寄存器 GDMA_INLINK_ADDR_CH*n* 指向第一个接收链表描述符。
3. 配置 GDMA_PERI_IN_SEL_CH*n* 为对应的外设号，见表 3-1。
4. 置位 GDMA_INLINK_START_CH*n* 启动 GDMA RX 通道发送数据。
5. 配置对应的外设，并启动该外设，具体配置请参考对应的外设章节。
6. 等待 GDMA_IN_SUC_EOF_CH*n*_INT 中断，即一个数据帧或包接收完成。

3.6.3 GDMA 存储器到存储器配置流程

利用 GDMA 从存储到存储搬运数据时配置流程如下：

1. 对寄存器 GDMA_OUT_RST_CH*n* 置 1 然后置 0，复位 GDMA TX 通道状态机和 FIFO 指针。
2. 对寄存器 GDMA_IN_RST_CH*n* 置 1 然后置 0，复位 GDMA RX 通道状态机和 FIFO 指针。
3. 挂载好发送链表，配置寄存器 GDMA_OUTLINK_ADDR_CH*n* 指向第一个发送链表描述符。
4. 挂载好接收链表，配置寄存器 GDMA_INLINK_ADDR_CH*n* 指向第一个接收链表描述符。
5. 置位 GDMA_MEM_TRANS_EN_CH*n* 使能 memory-to-memory 传输功能。

6. 置位 GDMA_OUTLINK_START_CH*n* 启动 GDMA TX 通道发送数据。
7. 置位 GDMA_INLINK_START_CH*n* 启动 GDMA RX 通道发送数据。
8. 等待 GDMA_IN_SUC_EOF_CH*n*_INT 中断，即一次数据搬运完成。

3.7 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
GDMA_IN_INT_RAW_CH0_REG	接收通道 0 的原始中断状态	0x0000	R/WTC/SS
GDMA_IN_INT_ST_CH0_REG	接收通道 0 的屏蔽中断状态	0x0004	RO
GDMA_IN_INT_ENA_CH0_REG	接收通道 0 的中断使能位	0x0008	R/W
GDMA_IN_INT_CLR_CH0_REG	接收通道 0 的中断清除位	0x000C	WT
GDMA_IN_INT_RAW_CH1_REG	发送通道 1 的原始中断状态	0x0010	R/WTC/SS
GDMA_IN_INT_ST_CH1_REG	发送通道 1 的屏蔽中断状态	0x0014	RO
GDMA_IN_INT_ENA_CH1_REG	发送通道 1 的中断使能位	0x0018	R/W
GDMA_IN_INT_CLR_CH1_REG	发送通道 1 的中断清除位	0x001C	WT
GDMA_IN_INT_RAW_CH2_REG	接收通道 2 的原始中断状态	0x0020	R/WTC/SS
GDMA_IN_INT_ST_CH2_REG	接收通道 2 的屏蔽中断状态	0x0024	RO
GDMA_IN_INT_ENA_CH2_REG	接收通道 2 的中断使能位	0x0028	R/W
GDMA_IN_INT_CLR_CH2_REG	接收通道 2 的中断清除位	0x002C	WT
GDMA_OUT_INT_RAW_CH0_REG	发送通道 0 的原始中断状态	0x0030	R/WTC/SS
GDMA_OUT_INT_ST_CH0_REG	发送通道 0 的屏蔽中断状态	0x0034	RO
GDMA_OUT_INT_ENA_CH0_REG	发送通道 0 的中断使能位	0x0038	R/W
GDMA_OUT_INT_CLR_CH0_REG	发送通道 0 的中断清除位	0x003C	WT
GDMA_OUT_INT_RAW_CH1_REG	发送通道 1 的原始中断状态	0x0040	R/WTC/SS
GDMA_OUT_INT_ST_CH1_REG	发送通道 1 的屏蔽中断状态	0x0044	RO
GDMA_OUT_INT_ENA_CH1_REG	发送通道 1 的中断使能位	0x0048	R/W
GDMA_OUT_INT_CLR_CH1_REG	发送通道 1 的中断清除位	0x004C	WT
GDMA_OUT_INT_RAW_CH2_REG	发送通道 2 的原始中断状态	0x0050	R/WTC/SS
GDMA_OUT_INT_ST_CH2_REG	发送通道 2 的屏蔽中断状态	0x0054	RO
GDMA_OUT_INT_ENA_CH2_REG	发送通道 2 的中断使能位	0x0058	R/W
GDMA_OUT_INT_CLR_CH2_REG	发送通道 2 的中断清除位	0x005C	WT
调试寄存器			

名称	描述	地址	访问
GDMA_AHB_TEST_REG	保留	0x0060	R/W
配置寄存器			
GDMA_MISC_CONF_REG	杂项寄存器	0x0064	R/W
GDMA_IN_CONF0_CH0_REG	接收通道 0 的配置寄存器 0	0x0070	R/W
GDMA_IN_CONF1_CH0_REG	接收通道 0 的配置寄存器 1	0x0074	R/W
GDMA_IN_POP_CH0_REG	接收通道 0 的弹出控制寄存器	0x007C	varies
GDMA_IN_LINK_CH0_REG	接收通道 0 的链表描述符配置和控制寄存器	0x0080	varies
GDMA_OUT_CONF0_CH0_REG	发送通道 0 的配置寄存器 0	0x00D0	R/W
GDMA_OUT_CONF1_CH0_REG	发送通道 0 的配置寄存器 1	0x00D4	R/W
GDMA_OUT_PUSH_CH0_REG	接收通道 0 的推入控制寄存器	0x00DC	varies
GDMA_OUT_LINK_CH0_REG	发送通道 0 的链表描述符配置和控制寄存器	0x00E0	varies
GDMA_IN_CONF0_CH1_REG	接收通道 1 的配置寄存器 0	0x0130	R/W
GDMA_IN_CONF1_CH1_REG	接收通道 1 的配置寄存器 1	0x0134	R/W
GDMA_IN_POP_CH1_REG	接收通道 1 的弹出控制寄存器	0x013C	varies
GDMA_IN_LINK_CH1_REG	接收通道 1 的链表描述符配置和控制寄存器	0x0140	varies
GDMA_OUT_CONF0_CH1_REG	发送通道 1 的配置寄存器 0	0x0190	R/W
GDMA_OUT_CONF1_CH1_REG	发送通道 1 的配置寄存器 1	0x0194	R/W
GDMA_OUT_PUSH_CH1_REG	接收通道 1 的推入控制寄存器	0x019C	varies
GDMA_OUT_LINK_CH1_REG	发送通道 1 的链表描述符配置和控制寄存器	0x01A0	varies
GDMA_IN_CONF0_CH2_REG	接收通道 2 的配置寄存器 0	0x01F0	R/W
GDMA_IN_CONF1_CH2_REG	接收通道 2 的配置寄存器 1	0x01F4	R/W
GDMA_IN_POP_CH2_REG	接收通道 2 的弹出控制寄存器	0x01FC	varies
GDMA_IN_LINK_CH2_REG	接收通道 2 的链表描述符配置和控制寄存器	0x0200	varies
GDMA_OUT_CONF0_CH2_REG	发送通道 2 的配置寄存器 0	0x0250	R/W
GDMA_OUT_CONF1_CH2_REG	发送通道 2 的配置寄存器 1	0x0254	R/W
GDMA_OUT_PUSH_CH2_REG	接收通道 2 的推入控制寄存器	0x025C	varies
GDMA_OUT_LINK_CH2_REG	发送通道 2 的链表描述符配置和控制寄存器	0x0260	varies
版本寄存器			
GDMA_DATE_REG	版本控制寄存器	0x0068	R/W
状态寄存器			
GDMA_INFIFO_STATUS_CH0_REG	接收通道 0 的 RX FIFO 状态	0x0078	RO
GDMA_IN_STATE_CH0_REG	接收通道 0 的接收状态	0x0084	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH0_REG	接收通道 0 传输结束 (EOF) 时接收链表描述符的地址	0x0088	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH0_REG	接收通道 0 出现错误时接收链表描述符的地址	0x008C	RO
GDMA_IN_DSCR_CH0_REG	接收通道 0 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0090	RO
GDMA_IN_DSCR_BF0_CH0_REG	接收通道 0 当前已预读取的接收链表描述符所在地址	0x0094	RO
GDMA_IN_DSCR_BF1_CH0_REG	接收通道 0 前一个已预读取的接收链表描述符所在地址	0x0098	RO
GDMA_OUTFIFO_STATUS_CH0_REG	发送通道 0 的 TX FIFO 状态	0x00D8	RO

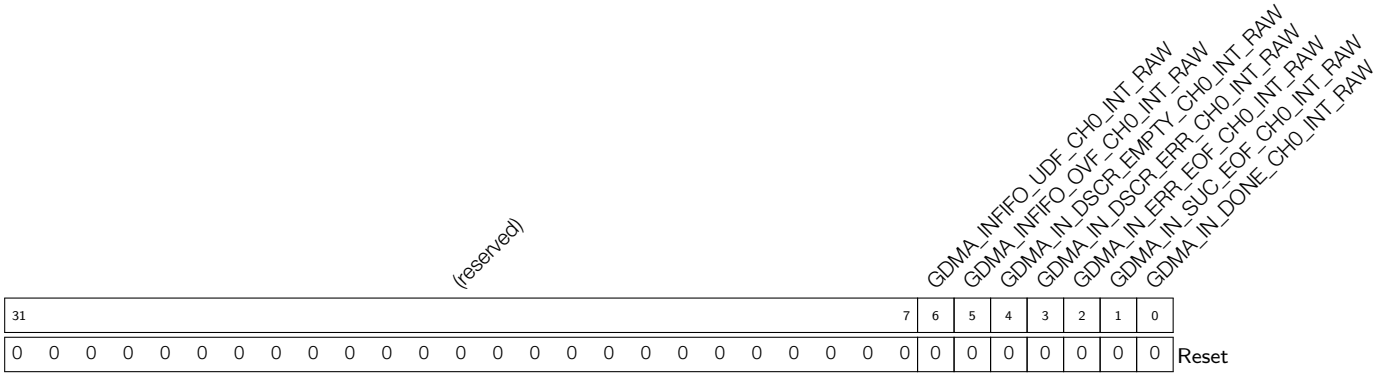
名称	描述	地址	访问
GDMA_OUT_STATE_CH0_REG	发送通道 0 的发送状态	0x00E4	RO
GDMA_OUT_EOF_DES_ADDR_CH0_REG	发送通道 0 传输结束 (EOF) 时的发送链表描述符地址	0x00E8	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH0_REG	发送通道 0 传输结束 (EOF) 时最后一个发送链表描述符的地址	0x00EC	RO
GDMA_OUT_DSCR_CH0_REG	发送通道 0 当前已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x00F0	RO
GDMA_OUT_DSCR_BF0_CH0_REG	发送通道 0 当前已预读取的发送链表描述符所在地址	0x00F4	RO
GDMA_OUT_DSCR_BF1_CH0_REG	发送通道 0 前一个已预读取的发送链表描述符所在地址	0x00F8	RO
GDMA_INFIFO_STATUS_CH1_REG	接收通道 1 的 RX FIFO 状态	0x0138	RO
GDMA_IN_STATE_CH1_REG	接收通道 1 的接收状态	0x0144	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH1_REG	接收通道 1 传输结束 (EOF) 时接收链表描述符的地址	0x0148	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH1_REG	接收通道 1 出现错误时接收链表描述符的地址	0x014C	RO
GDMA_IN_DSCR_CH1_REG	接收通道 1 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0150	RO
GDMA_IN_DSCR_BF0_CH1_REG	接收通道 1 当前已预读取的接收链表描述符所在地址	0x0154	RO
GDMA_IN_DSCR_BF1_CH1_REG	接收通道 1 前一个已预读取的接收链表描述符所在地址	0x0158	RO
GDMA_OUTFIFO_STATUS_CH1_REG	发送通道 1 的 TX FIFO 状态	0x0198	RO
GDMA_OUT_STATE_CH1_REG	发送通道 1 的发送状态	0x01A4	RO
GDMA_OUT_EOF_DES_ADDR_CH1_REG	发送通道 1 传输结束 (EOF) 时的发送链表描述符地址	0x01A8	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH1_REG	发送通道 1 传输结束 (EOF) 时最后一个发送链表描述符的地址	0x01AC	RO
GDMA_OUT_DSCR_CH1_REG	发送通道 1 当前已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x01B0	RO
GDMA_OUT_DSCR_BF0_CH1_REG	发送通道 1 当前已预读取的发送链表描述符所在地址	0x01B4	RO
GDMA_OUT_DSCR_BF1_CH1_REG	发送通道 1 前一个已预读取的发送链表描述符所在地址	0x01B8	RO
GDMA_INFIFO_STATUS_CH2_REG	接收通道 2 的 RX FIFO 状态	0x01F8	RO
GDMA_IN_STATE_CH2_REG	接收通道 2 的接收状态	0x0204	RO
GDMA_IN_SUC_EOF_DES_ADDR_CH2_REG	接收通道 2 传输结束 (EOF) 时接收链表描述符的地址	0x0208	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH2_REG	接收通道 2 出现错误时接收链表描述符的地址	0x020C	RO
GDMA_IN_DSCR_CH2_REG	接收通道 2 已预读取的接收链表描述符指向的下一个接收链表描述符地址	0x0210	RO

名称	描述	地址	访问
GDMA_IN_DSCR_BF0_CH2_REG	接收通道 2 当前已预读取的接收链表描述符所在地址	0x0214	RO
GDMA_IN_DSCR_BF1_CH2_REG	接收通道 2 前一个已预读取的接收链表描述符所在地址	0x0218	RO
GDMA_OUTFIFO_STATUS_CH2_REG	发送通道 2 的 TX FIFO 状态	0x0258	RO
GDMA_OUT_STATE_CH2_REG	发送通道 2 的发送状态	0x0264	RO
GDMA_OUT_EOF_DES_ADDR_CH2_REG	发送通道 2 传输结束 (EOF) 时的发送链表描述符地址	0x0268	RO
GDMA_OUT_EOF_BFR_DES_ADDR_CH2_REG	发送通道 2 传输结束 (EOF) 时最后一个发送链表描述符的地址	0x026C	RO
GDMA_OUT_DSCR_CH2_REG	发送通道 2 当前已预读取的发送链表描述符指向的下一个发送链表描述符地址	0x0270	RO
GDMA_OUT_DSCR_BF0_CH2_REG	发送通道 2 当前已预读取的发送链表描述符所在地址	0x0274	RO
GDMA_OUT_DSCR_BF1_CH2_REG	发送通道 2 前一个已预读取的发送链表描述符所在地址	0x0278	RO
优先级寄存器			
GDMA_IN_PRI_CH0_REG	接收通道 0 的优先级寄存器	0x009C	R/W
GDMA_OUT_PRI_CH0_REG	发送通道 0 的优先级寄存器	0x00FC	R/W
GDMA_IN_PRI_CH1_REG	接收通道 1 的优先级寄存器	0x015C	R/W
GDMA_OUT_PRI_CH1_REG	发送通道 1 的优先级寄存器	0x01BC	R/W
GDMA_IN_PRI_CH2_REG	接收通道 2 的优先级寄存器	0x021C	R/W
GDMA_OUT_PRI_CH2_REG	发送通道 2 的优先级寄存器	0x027C	R/W
外设选择寄存器			
GDMA_IN_PERI_SEL_CH0_REG	接收通道 0 的外设选择寄存器	0x00A0	R/W
GDMA_OUT_PERI_SEL_CH0_REG	发送通道 0 的外设选择寄存器	0x0100	R/W
GDMA_IN_PERI_SEL_CH1_REG	接收通道 1 的外设选择寄存器	0x0160	R/W
GDMA_OUT_PERI_SEL_CH1_REG	发送通道 1 的外设选择寄存器	0x01C0	R/W
GDMA_IN_PERI_SEL_CH2_REG	接收通道 2 的外设选择寄存器	0x0220	R/W
GDMA_OUT_PERI_SEL_CH2_REG	发送通道 2 的外设选择寄存器	0x0280	R/W

3.8 寄存器

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

Register 3.1. GDMA_IN_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0000+0x10n*)**



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

GDMA_IN_SUC_EOF_CH*n*_INT_RAW GDMA_IN_SUC_EOF_CH*n*_INT 的原始中断状态。对于 UHCI，接收链表描述符指向的最后一个字节数据接收成功、且接收通道 0 没有检测到数据错误时，该位变为 1。(R/WTC/SS)

GDMA_IN_ERR_EOF_CH*n*_INT_RAW GDMA_IN_ERR_EOF_CH*n*_INT 的原始中断状态。仅对 UHCI 或 PARLIO 有效。(R/WTC/SS)

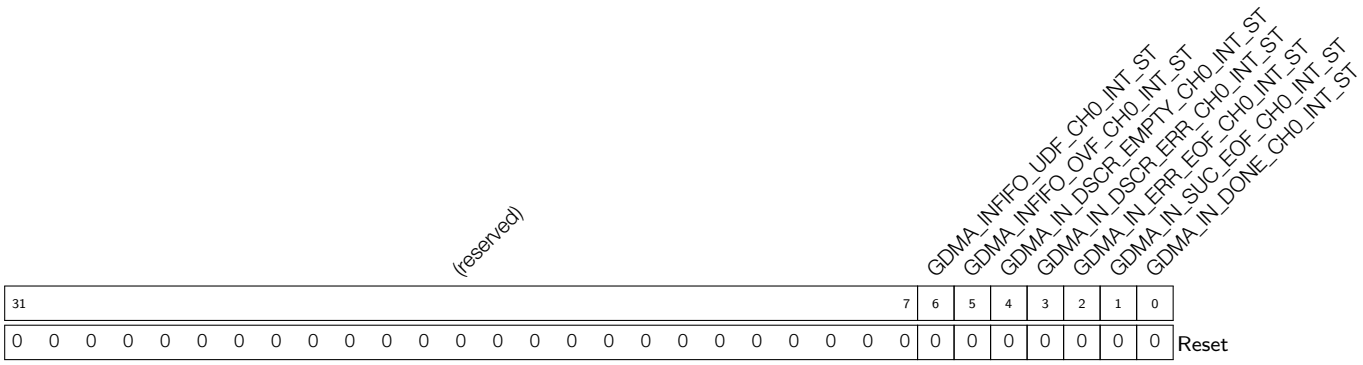
GDMA_IN_DSCR_ERR_CH*n*_INT_RAW GDMA_IN_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

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

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

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

Register 3.2. GDMA_IN_INT_ST_CH*n*_REG (*n*: 0-2) (0x0004+0x10n*)**



GDMA_IN_DONE_CH*n*_INT_ST GDMA_IN_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_IN_SUC_EOF_CH*n*_INT_ST GDMA_IN_SUC_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_IN_ERR_EOF_CH*n*_INT_ST GDMA_IN_ERR_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

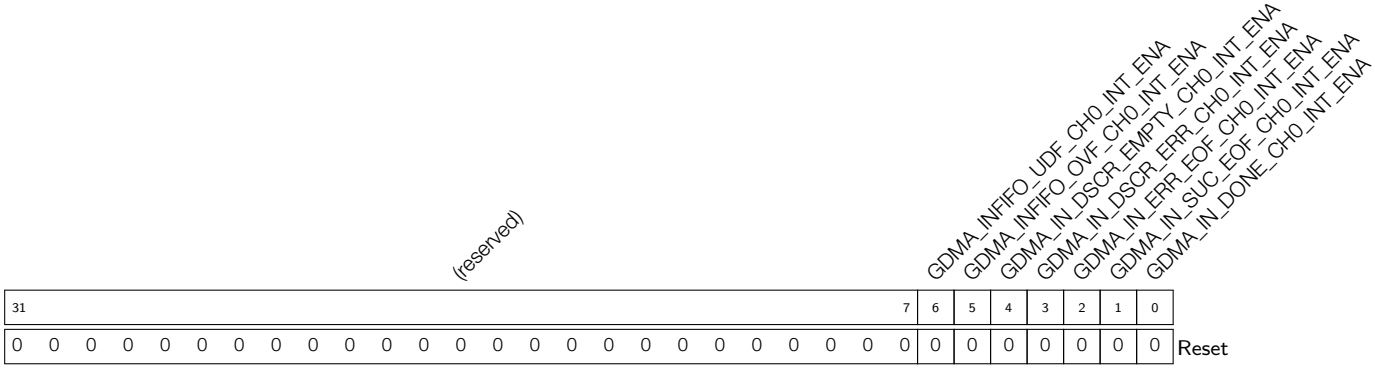
GDMA_IN_DSCR_ERR_CH*n*_INT_ST GDMA_IN_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_IN_DSCR_EMPTY_CH*n*_INT_ST GDMA_IN_DSCR_EMPTY_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_INFIFO_OVF_CH*n*_INT_ST GDMA_INFIFO_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_INFIFO_UDF_CH*n*_INT_ST GDMA_INFIFO_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

Register 3.3. GDMA_IN_INT_ENA_CH*n*_REG (*n*: 0-2) (0x0008+0x10n*)**



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

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

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

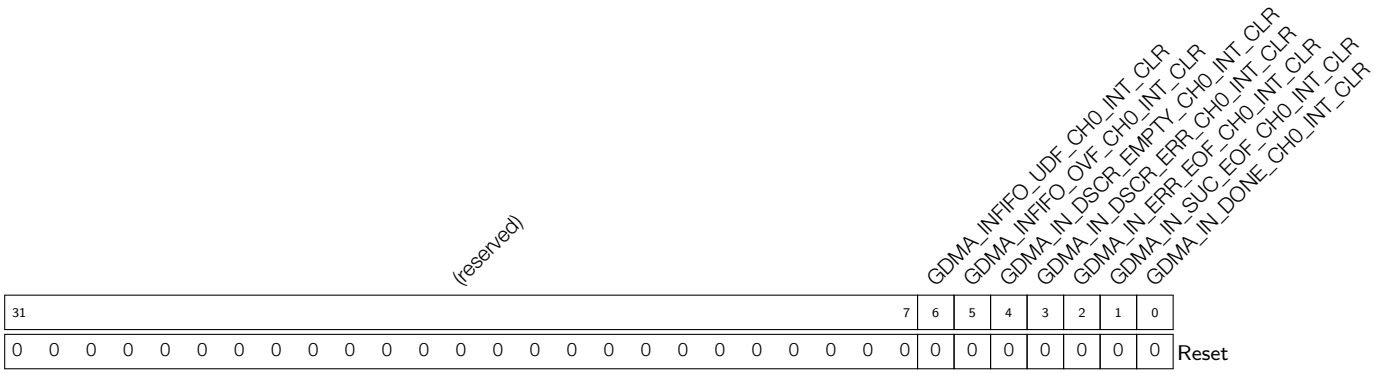
GDMA_IN_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 GDMA_IN_DSCR_ERR_CH*n*_INT。(R/W)

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

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

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

Register 3.4. GDMA_IN_INT_CLR_CH*n*_REG (*n*: 0-2) (0x000C+0x10n*)**



GDMA_IN_DONE_CH*n*_INT_CLR 写 1 清除 GDMA_IN_DONE_CH*n*_INT。(WT)

GDMA_IN_SUC_EOF_CH*n*_INT_CLR 写 1 清除 GDMA_IN_SUC_EOF_CH*n*_INT。(WT)

GDMA_IN_ERR_EOF_CH*n*_INT_CLR 写 1 清除 GDMA_IN_ERR_EOF_CH*n*_INT。(WT)

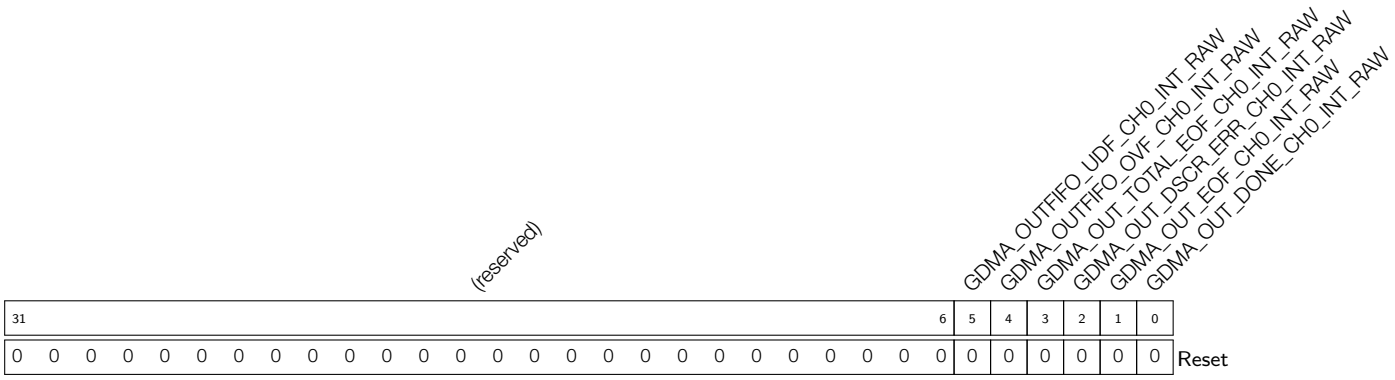
GDMA_IN_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 GDMA_IN_DSCR_ERR_CH*n*_INT。(WT)

GDMA_IN_DSCR_EMPTY_CH*n*_INT_CLR 写 1 清除 GDMA_IN_DSCR_EMPTY_CH*n*_INT。(WT)

GDMA_INFIFO_OVF_CH*n*_INT_CLR 写 1 清除 GDMA_INFIFO_OVF_CH*n*_INT。(WT)

GDMA_INFIFO_UDF_CH*n*_INT_CLR 写 1 清除 GDMA_INFIFO_UDF_CH*n*_INT。(WT)

Register 3.5. GDMA_OUT_INT_RAW_CH*n*_REG (*n*: 0-2) (0x0030+0x10n*)**



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

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

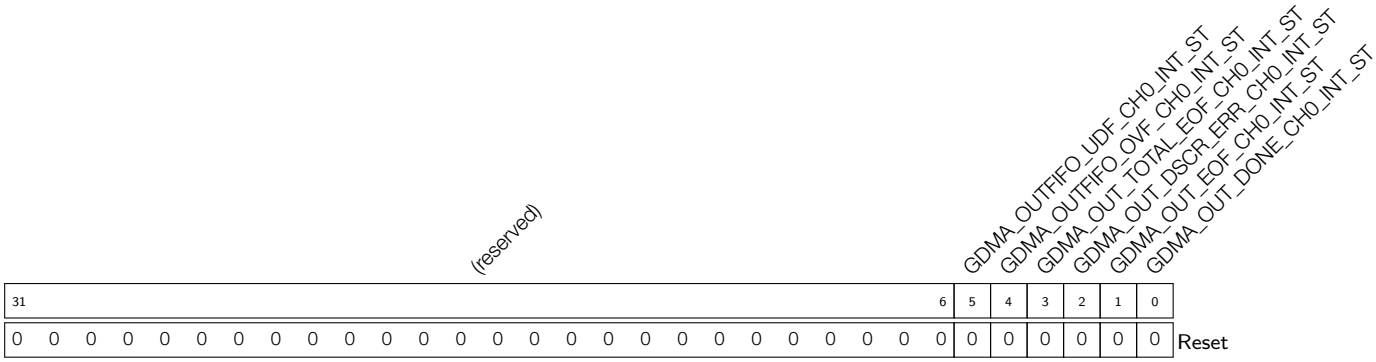
GDMA_OUT_DSCR_ERR_CH*n*_INT_RAW GDMA_OUT_DSCR_ERR_CH*n*_INT 的原始中断状态。(R/WTC/SS)

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

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

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

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



GDMA_OUT_DONE_CH*n*_INT_ST GDMA_OUT_DONE_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_OUT_EOF_CH*n*_INT_ST GDMA_OUT_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

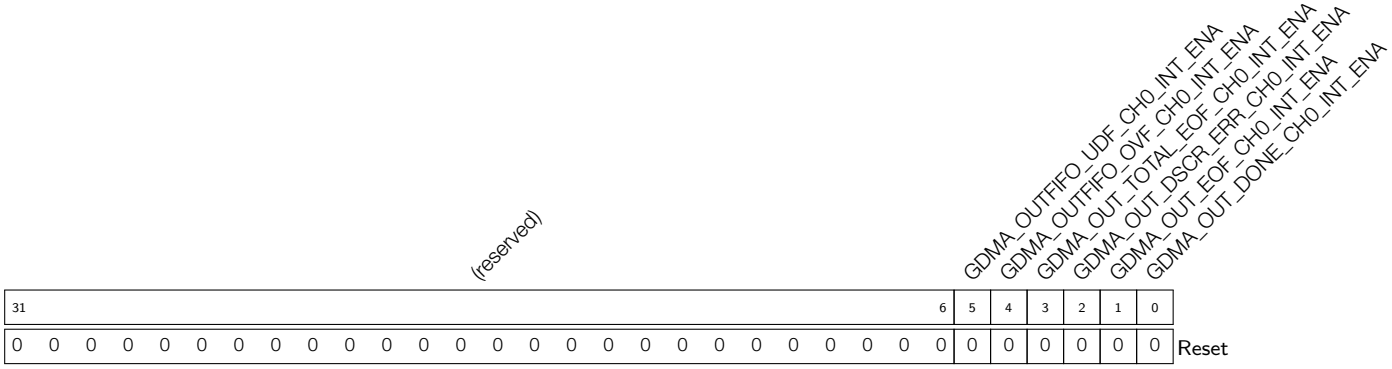
GDMA_OUT_DSCR_ERR_CH*n*_INT_ST GDMA_OUT_DSCR_ERR_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_OUT_TOTAL_EOF_CH*n*_INT_ST GDMA_OUT_TOTAL_EOF_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_OUTFIFO_OVF_CH*n*_INT_ST GDMA_OUTFIFO_OVF_CH*n*_INT 的屏蔽中断状态。(RO)

GDMA_OUTFIFO_UDF_CH*n*_INT_ST GDMA_OUTFIFO_UDF_CH*n*_INT 的屏蔽中断状态。(RO)

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



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

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

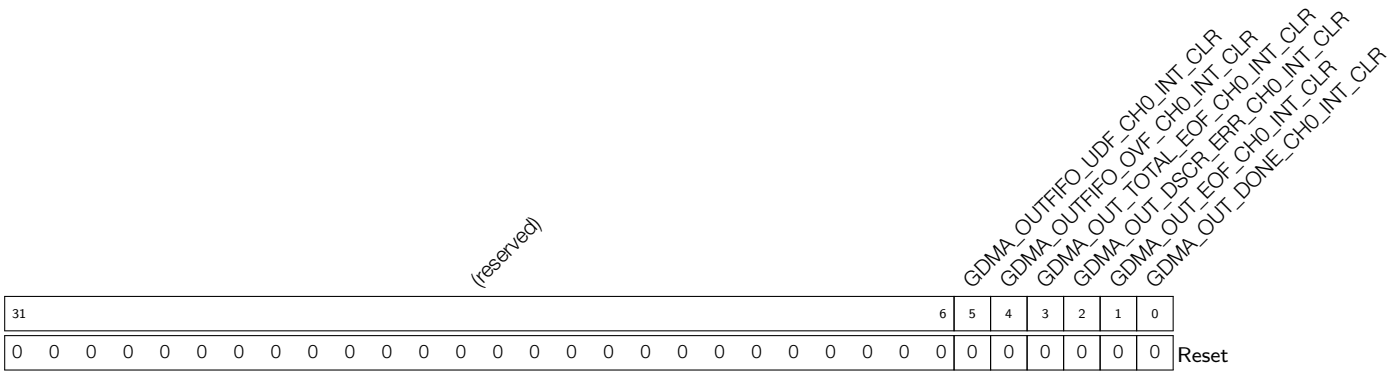
GDMA_OUT_DSCR_ERR_CH*n*_INT_ENA 写 1 使能 GDMA_OUT_DSCR_ERR_CH*n*_INT。(R/W)

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

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

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

Register 3.8. GDMA_OUT_INT_CLR_CH*n*_REG (*n*: 0-2) (0x003C+0x10n*)**



GDMA_OUT_DONE_CH*n*_INT_CLR 写 1 清除 GDMA_OUT_DONE_CH*n*_INT。(WT)

GDMA_OUT_EOF_CH*n*_INT_CLR 写 1 清除 GDMA_OUT_EOF_CH*n*_INT。(WT)

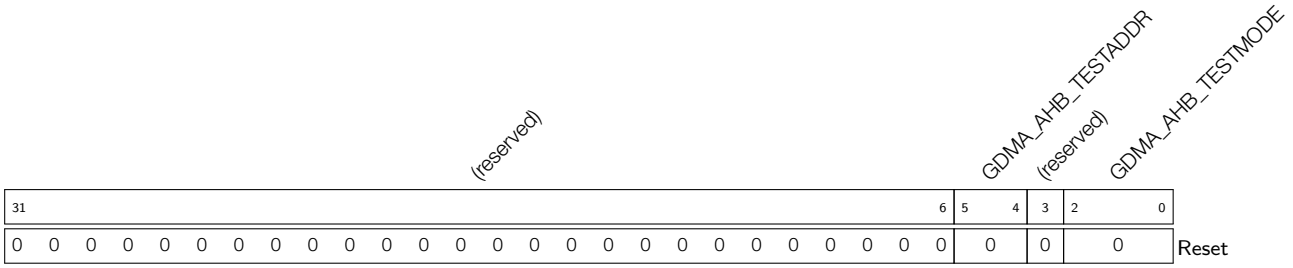
GDMA_OUT_DSCR_ERR_CH*n*_INT_CLR 写 1 清除 GDMA_OUT_DSCR_ERR_CH*n*_INT。(WT)

GDMA_OUT_TOTAL_EOF_CH*n*_INT_CLR 写 1 清除 GDMA_OUT_TOTAL_EOF_CH*n*_INT。(WT)

GDMA_OUTFIFO_OVF_CH*n*_INT_CLR 写 1 清除 GDMA_OUTFIFO_OVF_CH*n*_INT。(WT)

GDMA_OUTFIFO_UDF_CH*n*_INT_CLR 写 1 清除 GDMA_OUTFIFO_UDF_CH*n*_INT。(WT)

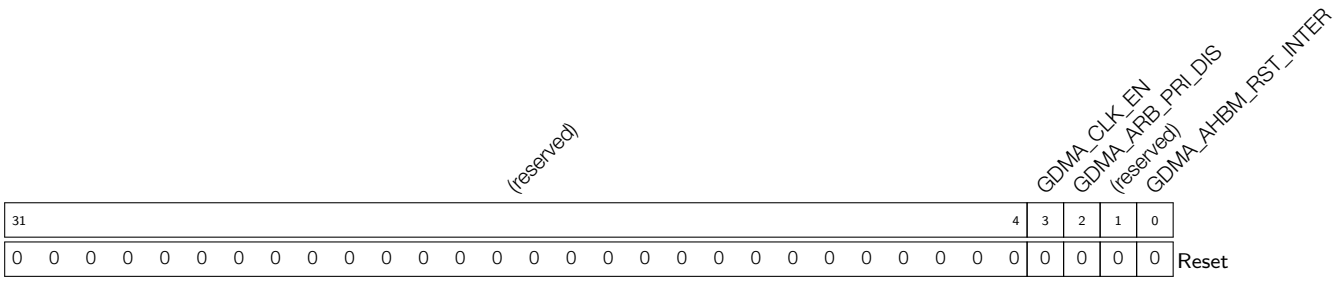
Register 3.9. GDMA_AHB_TEST_REG (0x0060)



GDMA_AHB_TESTMODE 保留。(R/W)

GDMA_AHB_TESTADDR 保留。(R/W)

Register 3.10. GDMA_MISC_CONF_REG (0x0064)



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

GDMA_ARB_PRI_DIS 配置是否关闭通道固定优先级仲裁。

0：开启

1：关闭

(R/W)

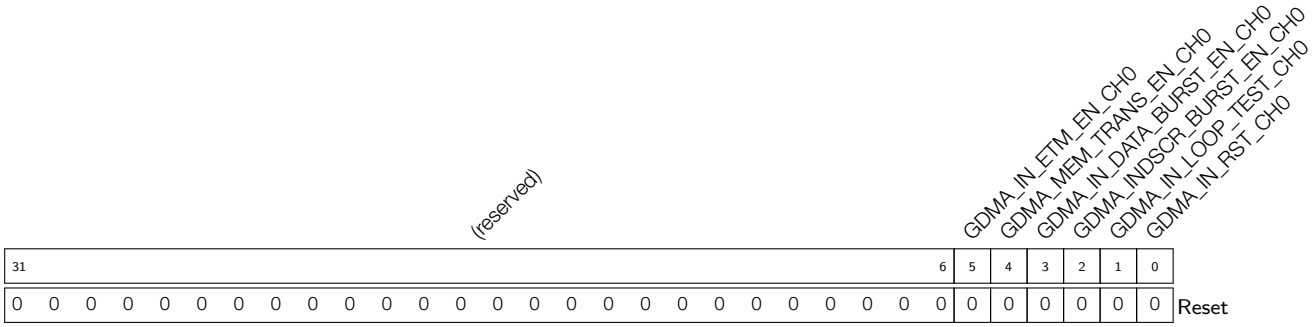
GDMA_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 3.11. GDMA_IN_CONF0_CH*n*_REG (*n*: 0-2) (0x0070+0xC0n*)**



GDMA_IN_RST_CH*n* 先写 1 再写 0 复位 GDMA 通道 0 的 RX FSM 和 RX FIFO 指针。(R/W)

GDMA_IN_LOOP_TEST_CH*n* 保留。(R/W)

GDMA_INDSCR_BURST_EN_CH*n* 配置是否开启 INCR 突发传输让接收通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

GDMA_IN_DATA_BURST_EN_CH*n* 配置是否开启接收通道 *n* 突发传输。

0：关闭

1：开启

(R/W)

GDMA_MEM_TRANS_EN_CH*n* 配置是否开启存储器到存储器数据传输。

0：关闭

1：开启

(R/W)

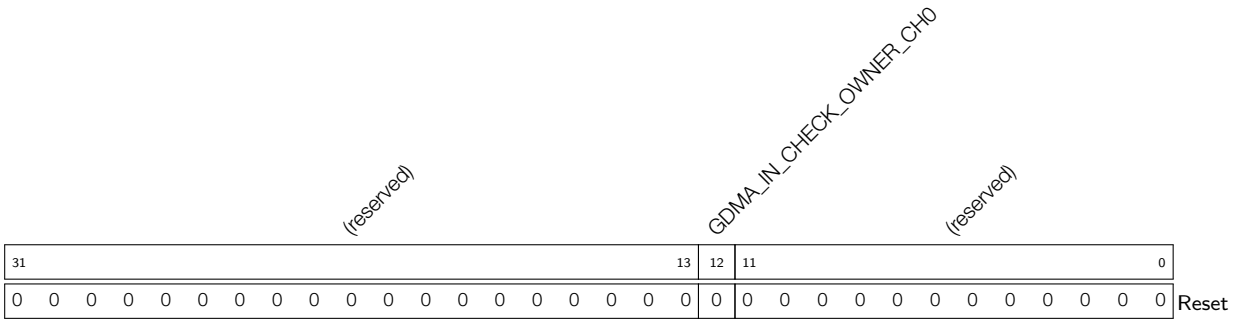
GDMA_IN_ETM_EN_CH*n* 配置是否开启接收通道 *n* 的 ETM 控制。

0：关闭

1：开启

(R/W)

Register 3.12. GDMA_IN_CONF1_CH*n*_REG (*n*: 0-2) (0x0074+0xC0n*)**



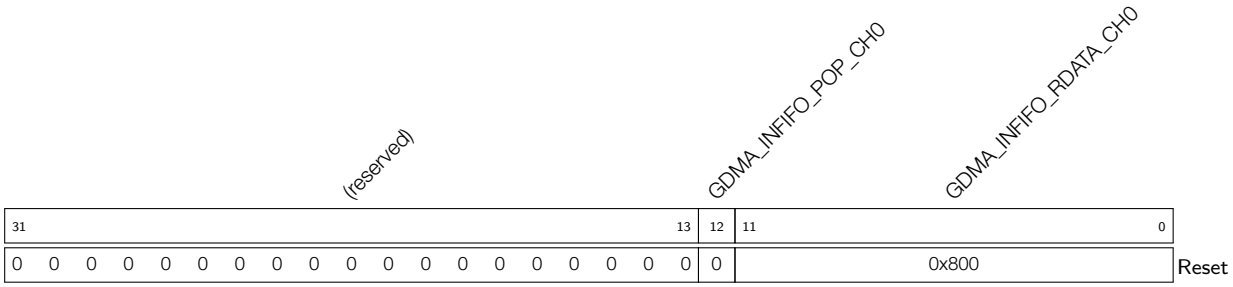
GDMA_IN_CHECK_OWNER_CH*n* 配置是否开启接收通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

Register 3.13. GDMA_IN_POP_CH*n*_REG (*n*: 0-2) (0x007C+0xC0n*)**



GDMA_INFIFO_RDATA_CH*n* 表示从 GDMA FIFO 弹出的数据。(RO)

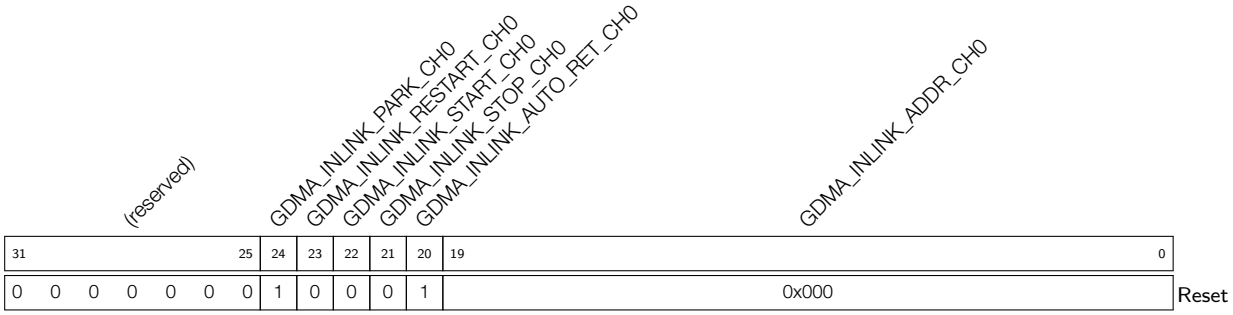
GDMA_INFIFO_POP_CH*n* 配置从 GDMA FIFO 中弹出数据。

0：无效值。没有作用

1：弹出

(WT)

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



GDMA_INLINK_ADDR_CH*n* 表示第一个接收链表描述符地址的低 20 位。(R/W)

GDMA_INLINK_AUTO_RET_CH*n* 配置是否在接收数据有错时返回到当前接收链表描述符的地址。

0：不返回

1：返回

(R/W)

GDMA_INLINK_STOP_CH*n* 配置 GDMA 的接收通道 *n* 停止接收数据。

0：无效值。没有作用

1：停止

(WT)

GDMA_INLINK_START_CH*n* 配置是否开启 GDMA 的接收通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

GDMA_INLINK_RESTART_CH*n* 配置是否重启接收通道 *n* 进行 GDMA 传输。

0：无效值。没有作用

1：重启

(WT)

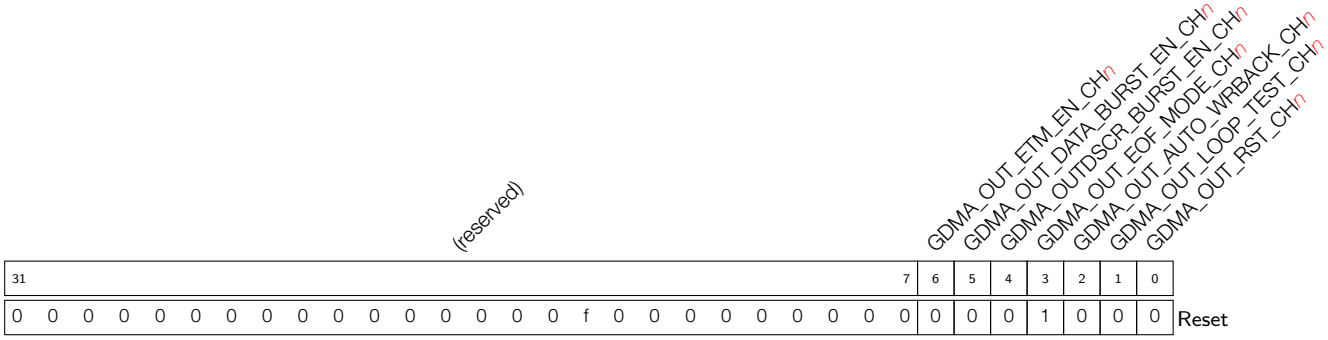
GDMA_INLINK_PARK_CH*n* 表示接收链表描述符 FSM 的状态。

0：运行

1：空闲

(RO)

Register 3.15. GDMA_OUT_CONF0_CH*n*_REG (*n*: 0-2) (0x00D0+0xC0*n)



GDMA_OUT_RST_CH*n* 配置 GDMA 通道 *n* TX FSM 和 TX FIFO 指针的复位状态。

0：复位释放

1：复位

(R/W)

GDMA_OUT_LOOP_TEST_CH*n* 保留。(R/W)

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

0：关闭

1：开启

(R/W)

GDMA_OUT_EOF_MODE_CH*n* 配置生成 EOF 标志的时间。

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

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

(R/W)

GDMA_OUTDSCR_BURST_EN_CH*n* 配置是否开启 INCR 突发传输，让发送通道 *n* 读取描述符。

0：关闭

1：开启

(R/W)

GDMA_OUT_DATA_BURST_EN_CH*n* 配置是否开启发送通道 *n* INCR 突发传输。

0：关闭

1：开启

(R/W)

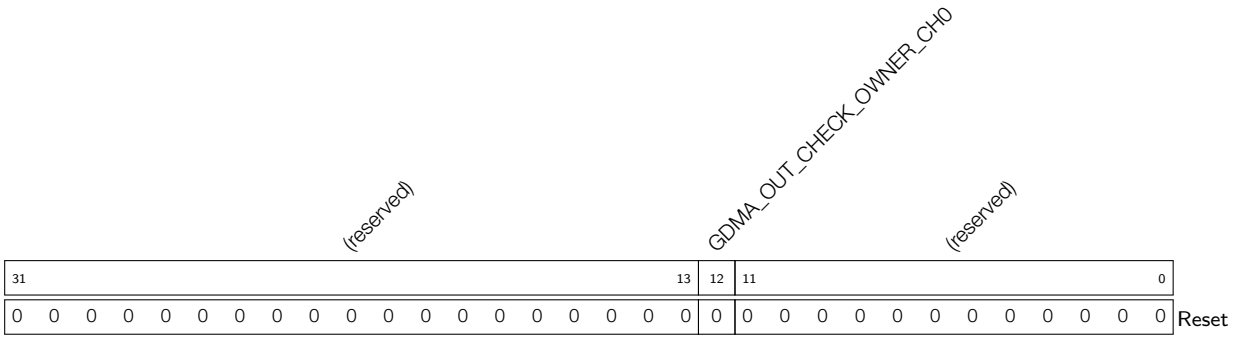
GDMA_OUT_ETM_EN_CH*n* 配置是否开启发送通道 *n* 的 ETM 控制。

0：关闭

1：开启

((R/W)

Register 3.16. GDMA_OUT_CONF1_CH*n*_REG (*n*: 0-2) (0x00D4+0xC0*n)



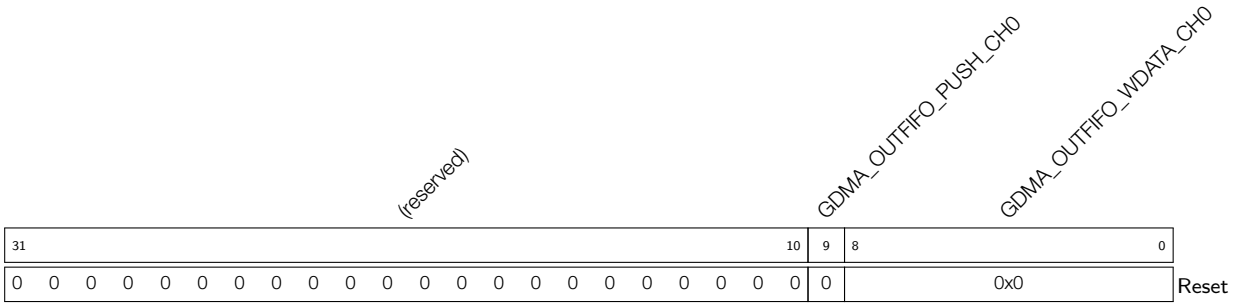
GDMA_OUT_CHECK_OWNER_CH*n* 配置是否开启发送通道 *n* 的 owner 位检查。

0：关闭

1：开启

(R/W)

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



GDMA_OUTFIFO_WDATA_CH*n* 表示需要推入 GDMA FIFO 中的数据。(R/W)

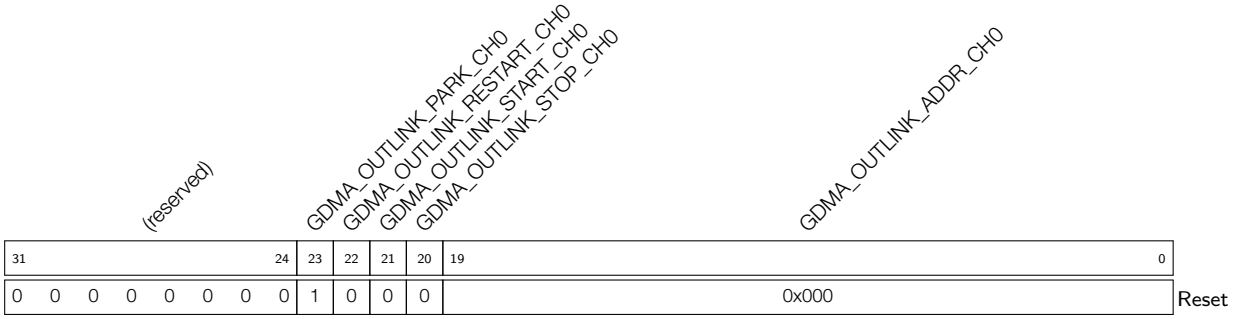
GDMA_OUTFIFO_PUSH_CH*n* 配置将数据推入 GDMA FIFO。

0：无效值。没有作用

1：推入

(WT)

Register 3.18. GDMA_OUT_LINK_CH*n*_REG (*n*: 0-2) (0x00E0+0xC0n*)**



GDMA_OUTLINK_ADDR_CH*n* 表示第一个发送链表描述符地址的低 20 位。(R/W)

GDMA_OUTLINK_STOP_CH*n* 配置 GDMA 的发送通道 *n* 停止发送数据。

0：无效值。没有作用

1：停止

(WT)

GDMA_OUTLINK_START_CH*n* 配置是否开启 GDMA 的发送通道 *n* 进行数据传输。

0：关闭

1：开启

(WT)

GDMA_OUTLINK_RESTART_CH*n* 配置重启发送通道 *n* 进行 GDMA 传输。

0：无效值。没有作用

1：重启

(WT)

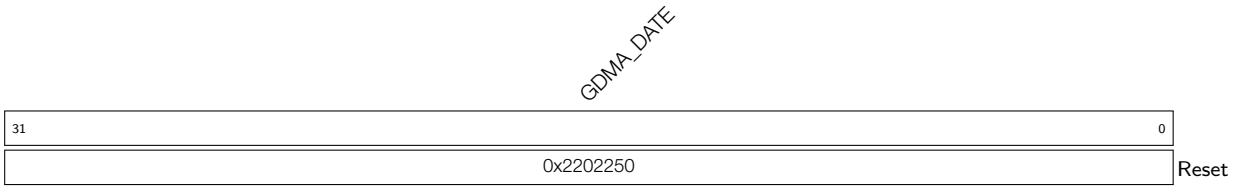
GDMA_OUTLINK_PARK_CH*n* 表示发送链表描述符 FSM 的状态。

0：运行

1：空闲

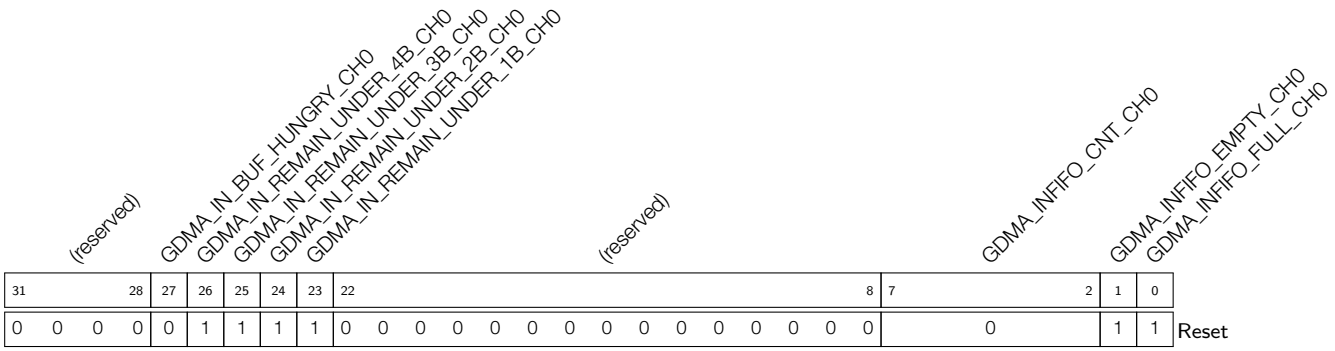
(RO)

Register 3.19. GDMA_DATE_REG (0x0068)



GDMA_DATE 版本控制寄存器。(R/W)

Register 3.20. GDMA_INFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x0078+0xC0n*)**



GDMA_INFIFO_FULL_CH*n* 表示 L1 RX FIFO 是否为满。

0：未满

1：已满

(RO)

GDMA_INFIFO_EMPTY_CH*n* 表示 L1 RX FIFO 是否为空。

0：未空

1：已空

(RO)

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

GDMA_IN_REMAIN_UNDER_1B_CH*n* 保留。(RO)

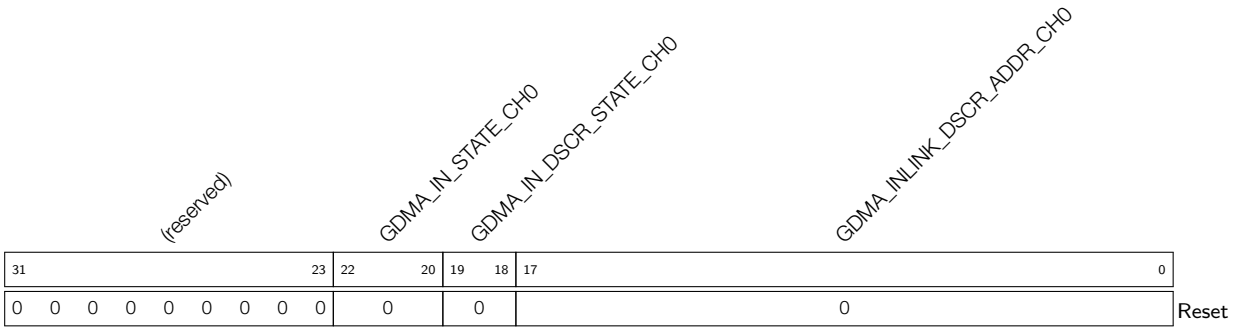
GDMA_IN_REMAIN_UNDER_2B_CH*n* 保留。(RO)

GDMA_IN_REMAIN_UNDER_3B_CH*n* 保留。(RO)

GDMA_IN_REMAIN_UNDER_4B_CH*n* 保留。(RO)

GDMA_IN_BUF_HUNGRY_CH*n* 保留。(RO)

Register 3.21. GDMA_IN_STATE_CH*n*_REG (*n*: 0-2) (0x0084+0xC0*n)

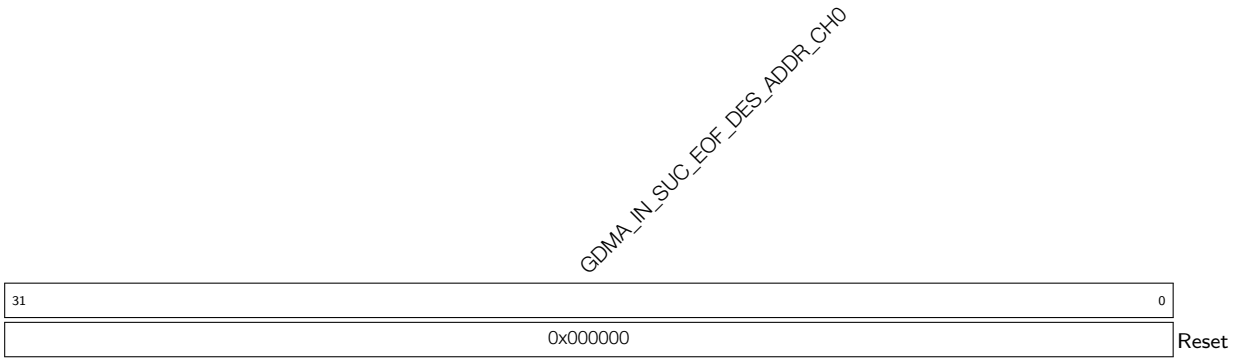


GDMA_INLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的接收链表描述符所在地址的低 18 位。如果当前处理的接收链表描述符是最后一个描述符，则该字段表示当前处理的接收链表描述符地址。(RO)

GDMA_IN_DSCR_STATE_CH*n* 保留。(RO)

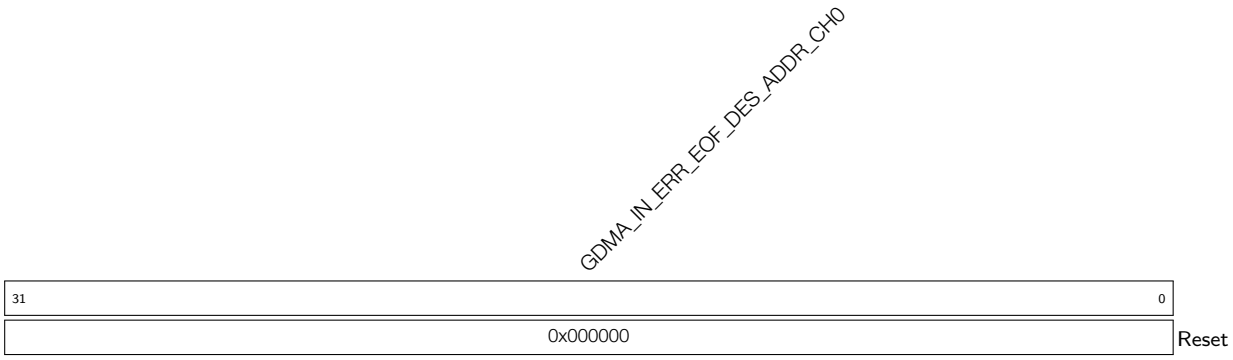
GDMA_IN_STATE_CH*n* 保留。(RO)

Register 3.22. GDMA_IN_SUC_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x0088+0xC0*n)



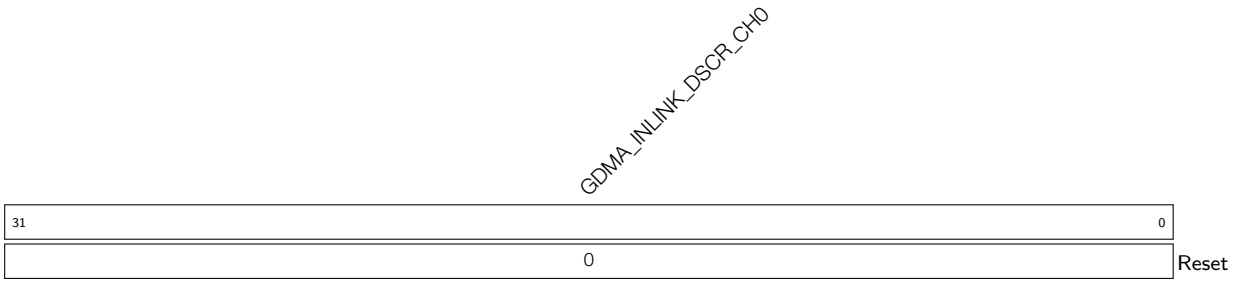
GDMA_IN_SUC_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的接收链表描述符的地址。(RO)

Register 3.23. GDMA_IN_ERR_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x008C+0xC0*n)



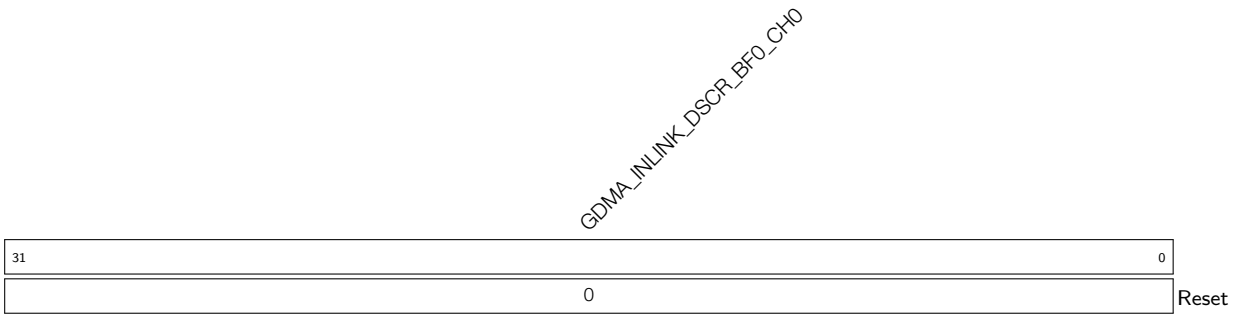
GDMA_IN_ERR_EOF_DES_ADDR_CH*n* 表示当前接收数据有错时，相应接收链表描述符的位置。仅对 UHCI 或 PARLIO 有效。(RO)

Register 3.24. GDMA_IN_DSCR_CH*n*_REG (*n*: 0-2) (0x0090+0xC0*n)



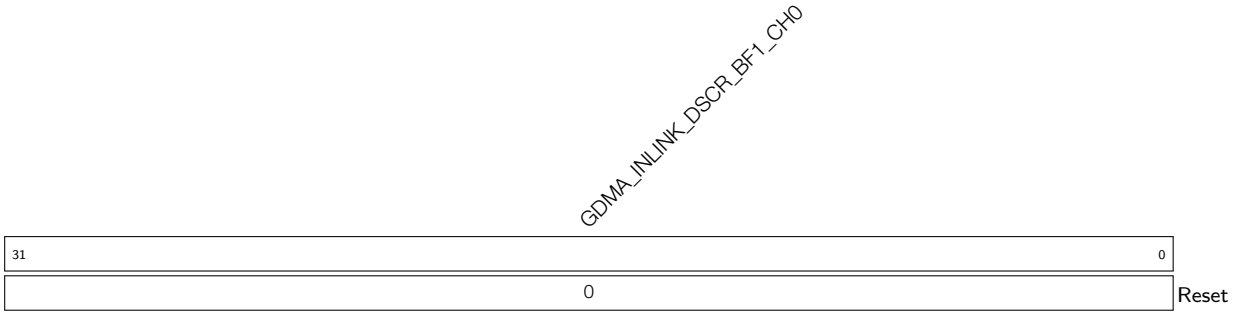
GDMA_INLINK_DSCR_CH*n* 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址 x+1。(RO)

Register 3.25. GDMA_IN_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x0094+0xC0*n)



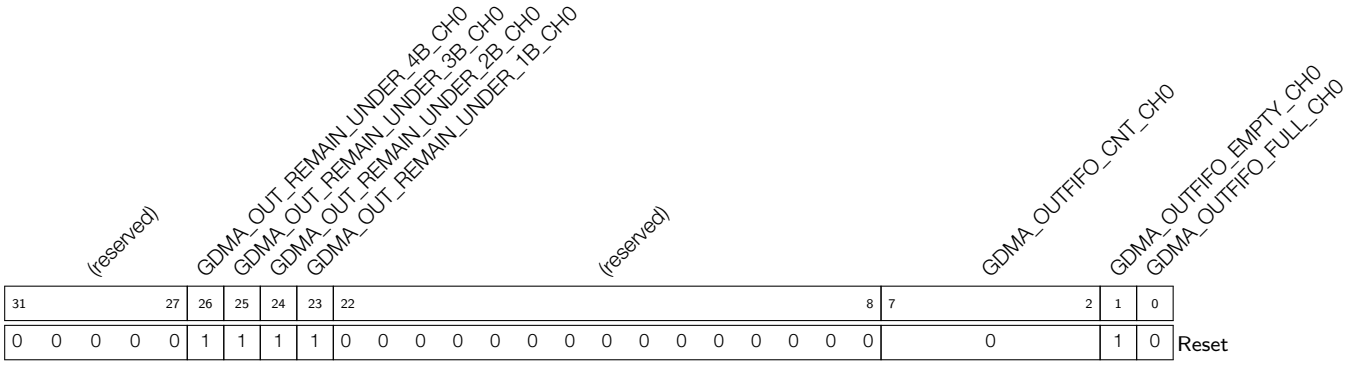
GDMA_INLINK_DSCR_BF0_CH*n* 表示当前已预读取的接收链表描述符所在地址 x。(RO)

Register 3.26. GDMA_IN_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x0098+0xC0*n)



GDMA_INLINK_DSCR_BF1_CH*n* 表示前一个已预读取的接收链表描述符所在地址 x-1。(RO)

Register 3.27. GDMA_OUTFIFO_STATUS_CH*n*_REG (*n*: 0-2) (0x00D8+0xC0*n)



GDMA_OUTFIFO_FULL_CH*n* 表示 L1 TX FIFO 是否为满。

0：未满

1：已满

(RO)

GDMA_OUTFIFO_EMPTY_CH*n* 表示 L1 TX FIFO 是否为空。

0：未空

1：已空

(RO)

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

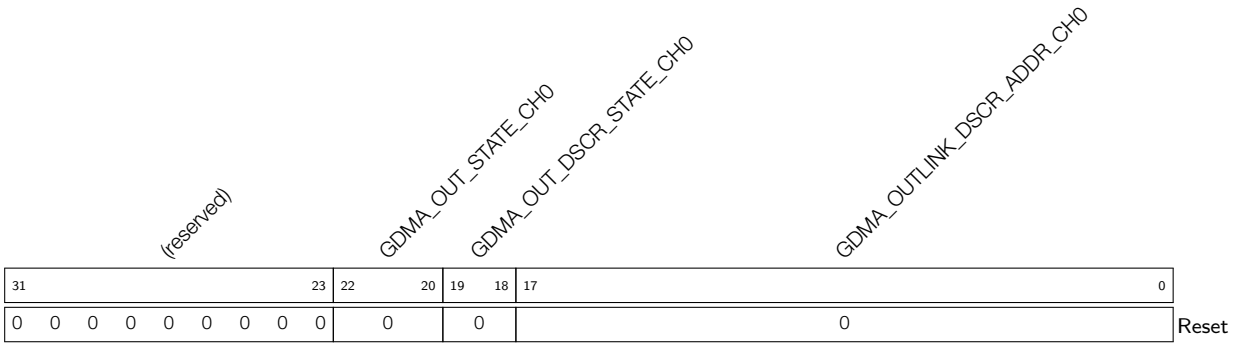
GDMA_OUT_REMAIN_UNDER_1B_CH*n* 保留。(RO)

GDMA_OUT_REMAIN_UNDER_2B_CH*n* 保留。(RO)

GDMA_OUT_REMAIN_UNDER_3B_CH*n* 保留。(RO)

GDMA_OUT_REMAIN_UNDER_4B_CH*n* 保留。(RO)

Register 3.28. GDMA_OUT_STATE_CH*n*_REG (*n*: 0-2) (0x00E4+0xC0**n*)

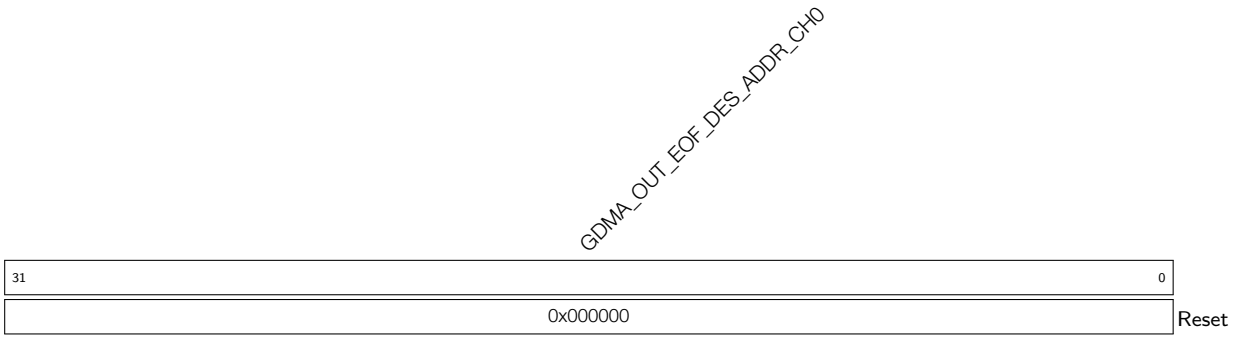


GDMA_OUTLINK_DSCR_ADDR_CH*n* 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

GDMA_OUT_DSCR_STATE_CH*n* 保留。(RO)

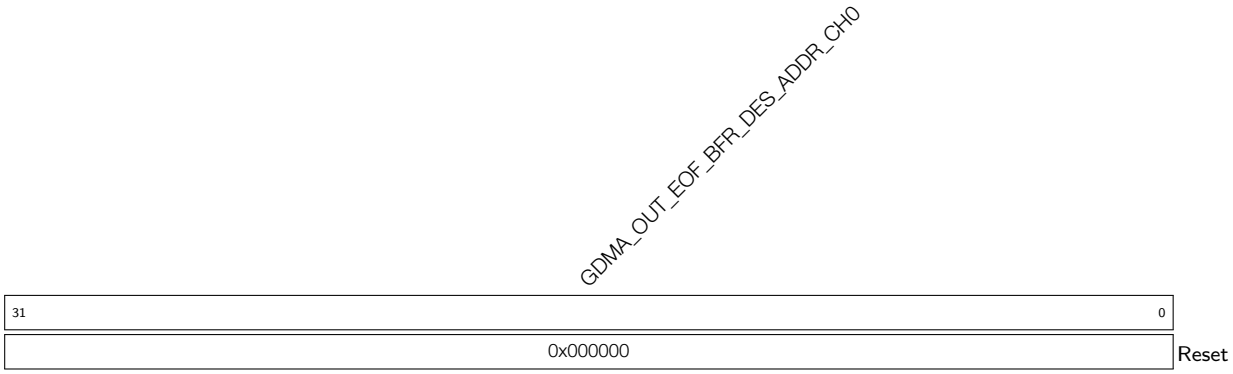
GDMA_OUT_STATE_CH*n* 保留。(RO)

Register 3.29. GDMA_OUT_EOF_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00E8+0xC0**n*)



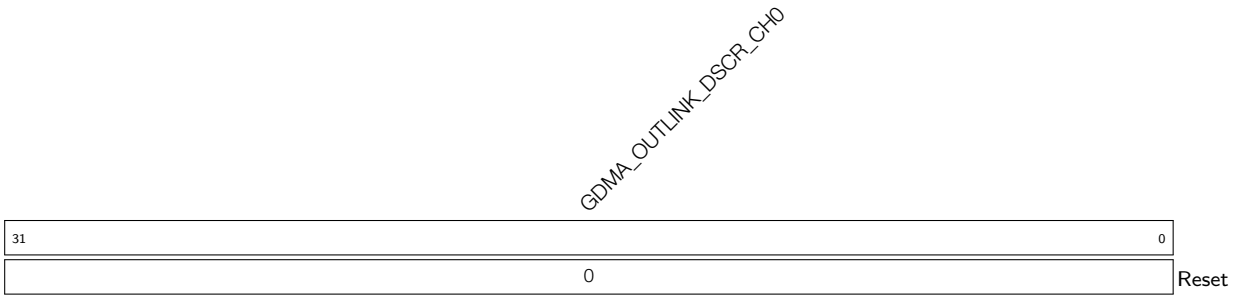
GDMA_OUT_EOF_DES_ADDR_CH*n* 表示 EOF 位为 1 的发送链表描述符的地址。(RO)

Register 3.30. GDMA_OUT_EOF_BFR_DES_ADDR_CH*n*_REG (*n*: 0-2) (0x00EC+0xC0**n*)



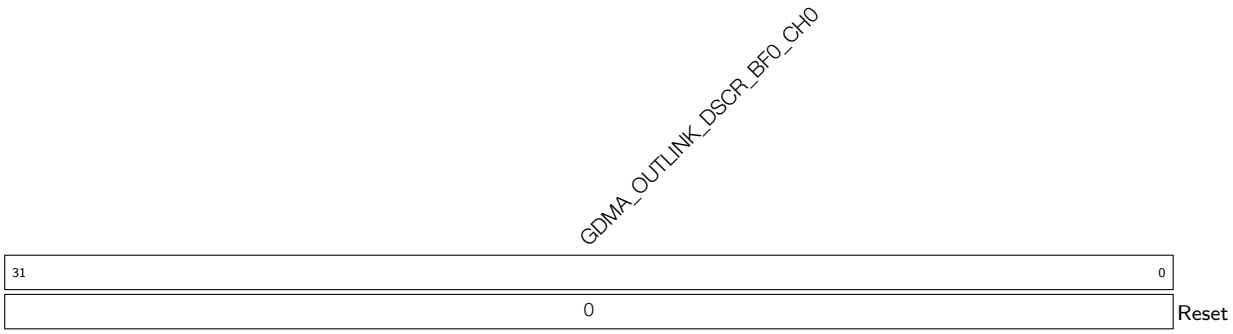
GDMA_OUT_EOF_BFR_DES_ADDR_CH*n* 表示倒数第二个发送链表描述符的地址。(RO)

Register 3.31. GDMA_OUT_DSCR_CH*n*_REG (*n*: 0-2) (0x00F0+0xC0*n*)



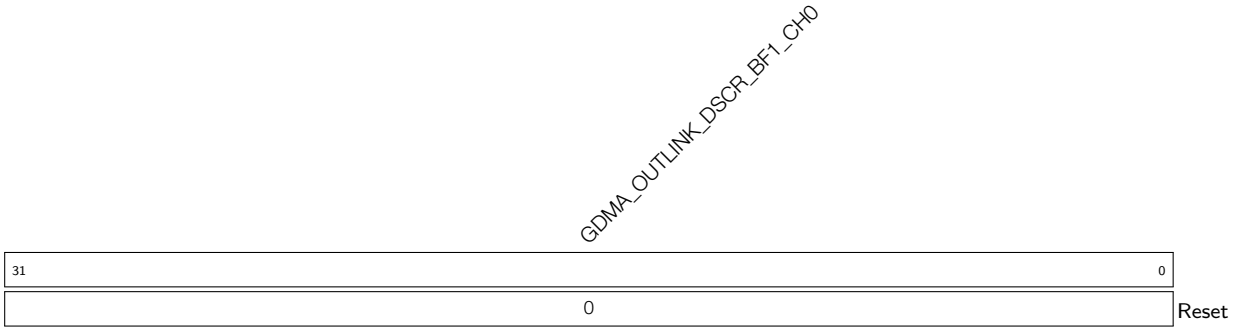
GDMA_OUTLINK_DSCR_CH*n* 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址 y+1。(RO)

Register 3.32. GDMA_OUT_DSCR_BF0_CH*n*_REG (*n*: 0-2) (0x00F4+0xC0*n*)



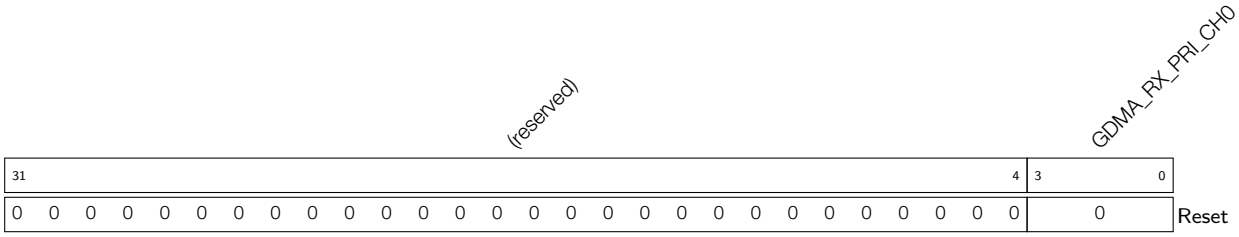
GDMA_OUTLINK_DSCR_BF0_CH*n* 表示当前已预读取的发送链表描述符所在地址 y。(RO)

Register 3.33. GDMA_OUT_DSCR_BF1_CH*n*_REG (*n*: 0-2) (0x00F8+0xC0*n*)



GDMA_OUTLINK_DSCR_BF1_CH*n* 表示前一个已预读取的发送链表描述符所在地址 y-1。(RO)

Register 3.34. GDMA_IN_PRI_CH*n*_REG (*n*: 0-2) (0x009C+0xC0*n*)

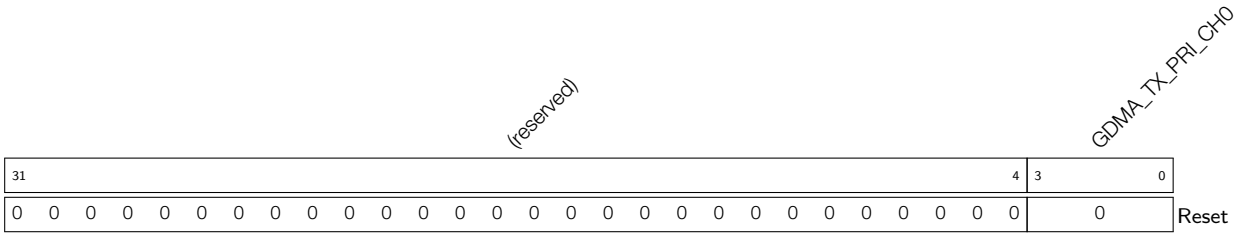


GDMA_RX_PRI_CH*n* 配置接收通道 *n* 对的优先级。

取值范围：0 ~ 9

值越大，优先级越高。(R/W)

Register 3.35. GDMA_OUT_PRI_CH*n*_REG (*n*: 0-2) (0x00FC+0xC0*n*)

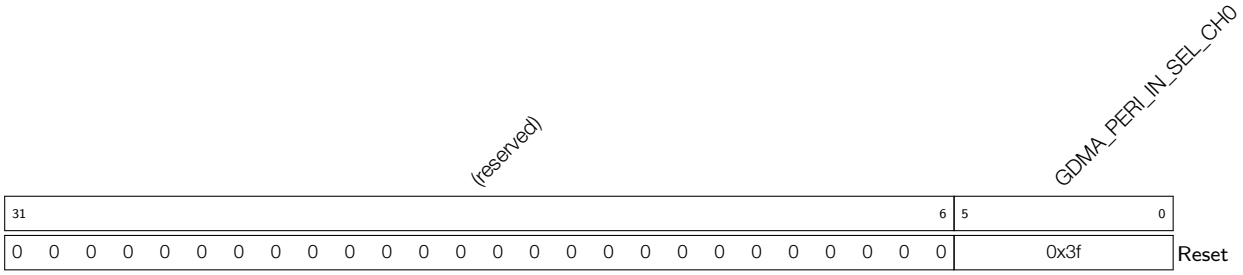


GDMA_TX_PRI_CH*n* 配置发送通道 *n* 对的优先级。

取值范围：0 ~ 9

值越大，优先级越高。(R/W)

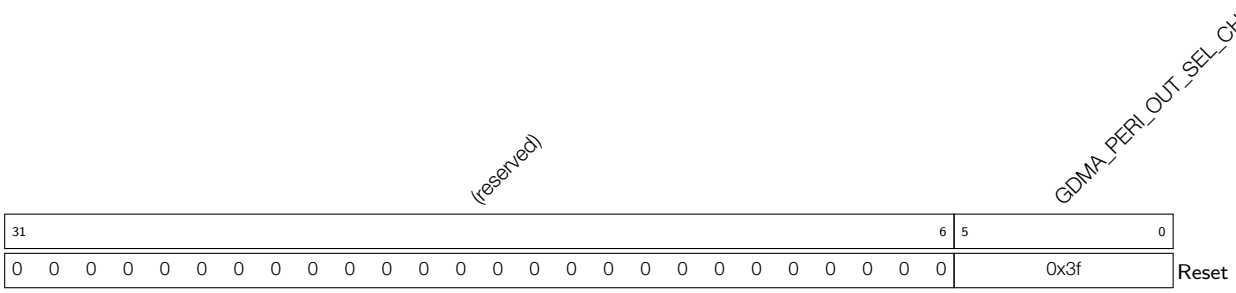
Register 3.36. GDMA_IN_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x00A0+0xC0*n)



GDMA_PERI_IN_SEL_CH*n* 配置连接接收通道 *n* 的外设。

0：SPI2
1：Dummy-1
2：UHCI
3：I2S0
4：Dummy-4
5：Dummy-5
6：AES
7：SHA
8：ADC
9：Parallel IO
10 ~ 15：Dummy-10 ~ 15
16 ~ 63：无效值
(R/W)

Register 3.37. GDMA_OUT_PERI_SEL_CH*n*_REG (*n*: 0-2) (0x0100+0xC0*n)



GDMA_PERI_OUT_SEL_CH*n* 配置连接发送通道 *n* 的外设。

0：SPI2
1：Dummy-1
2：UHCI
3：I2S0
4：Dummy-4
5：Dummy-5
6：AES
7：SHA
8：ADC
9：Parallel IO
10 ~ 15：Dummy-10 ~ 15
16 ~ 63：无效值
(R/W)

4 系统和存储器

4.1 概述

ESP32-H2 是一个超低功耗和高度集成的系统，它集成了一颗高性能 RISC-V 32 位单核处理器 (CPU)，四级流水线架构，主频高达 96 MHz。所有的内部存储器、外部存储器以及外设都分布在 CPU 的总线上。

4.2 主要特性

- **地址空间**
  - 452 KB 内部存储器空间，可被指令总线或数据总线访问
  - 832 KB 外设地址空间
  - 16 MB 外部存储器虚拟地址空间，可被指令总线或数据总线访问
  - 320 KB 内部 DMA 地址空间
- **内部存储器**
  - 128 KB 的 ROM
  - 320 KB 的 HP 存储器 (HP SRAM)
  - 4 KB 的 LP 存储器 (LP SRAM)
- **外部存储器**
  - 最大支持 16 MB 片外 flash
- **外设空间**
  - 总计 46 个模块/外设
  - 16 KB 的 Cache
  - Cache 块大小为 32 字节
- **GDMA**
  - 8 个具有 GDMA 功能的模块/外设

图 4-1 描述了系统结构与地址映射结构。

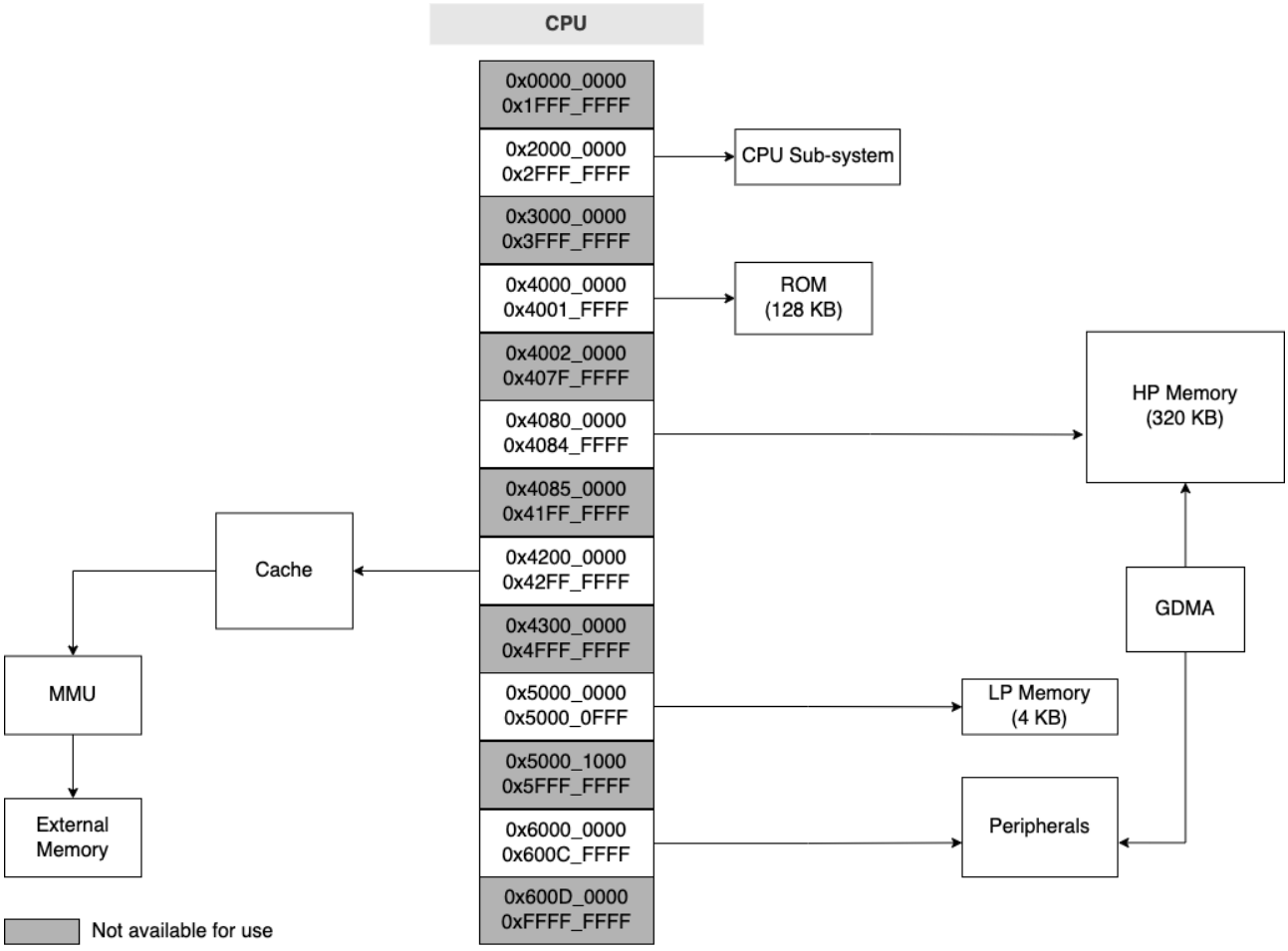


图 4-1. 系统结构与地址映射结构

说明：

- 地址空间中可用的地址范围可能大于实际可用的内存。
- 关于 CPU Sub-system，详情请参考章节 1 *ESP-RISC-V CPU*。

4.3 功能描述

4.3.1 地址映射

所有非保留地址均可通过指令总线和数据总线进行访问，即指令总线和数据总线访问的地址空间一样。

CPU 的数据总线与指令总线均为小端序。

CPU 可以通过数据总线进行单字节、双字节、四字节的数据访问。

CPU 能够：

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

表 4-1 描述了数据总线与指令总线中的各段地址所能访问的目标。

表 4-1. 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	0x1FFF_FFFF		保留
数据总线/指令总线	0x2000_0000	0x2FFF_FFFF	256 MB	CPU Sub-system
	0x3000_0000	0x3FFF_FFFF		保留
数据总线/指令总线	0x4000_0000	0x4001_FFFF	128 KB	ROM*
	0x4002_0000	0x407F_FFFF		保留
数据总线/指令总线	0x4080_0000	0x4084_FFFF	320 KB	HP SRAM*
	0x4085_0000	0x41FF_FFFF		保留
数据总线/指令总线	0x4200_0000	0x42FF_FFFF	16 MB	外部存储器
	0x4300_0000	0x4FFF_FFFF		保留
数据总线/指令总线	0x5000_0000	0x5000_0FFF	4 KB	LP SRAM*
	0x5000_1000	0x5FFF_FFFF		保留
数据总线/指令总线	0x6000_0000	0x600C_FFFF	832 KB	外设
	0x600D_0000	0xFFFF_FFFF		保留

* 所有内部存储器均接受权限管理。只有获取到访问内部存储器的访问权限，才可以执行正常的访问操作，CPU 访问内部存储器时才可以被响应。关于权限管理的更多信息，请参考章节 14 访问权限管理 (APM)。

4.3.2 内部存储器

ESP32-H2 的内部存储器包含如下三种类型：

- ROM (128 KB)：ROM 是只读存储器，不可编程。其中存放有一些系统底层软件 ROM 的代码和只读数据。
- HP SRAM (320 KB)：HP SRAM 是易失性存储器，可以快速响应 CPU 的访问请求（通常是一个时钟周期）。
- LP SRAM (4 KB)：LP SRAM 以静态 RAM (SRAM) 方式实现，因此也是易失性存储器。但是，在 Deep-sleep 模式下，存放在 LP SRAM 中的数据不会丢失。LP SRAM 可以被 CPU 访问，通常用来存放一些在睡眠模式下仍需保持的程序指令和数据。

1. ROM

ROM 的容量为 128 KB，只读。如表 4-1 所示，CPU 可以通过指令总线或数据总线地址段 0x4000_0000 ~ 0x4001_FFFF 同序访问这部分存储器。

同序访问 ROM 是指：地址（例如 0x4001_0000）可被指令总线或数据总线访问。

2. HP SRAM

HP SRAM 的容量为 320 KB，可读可写。如表 4-1 所示，CPU 可以通过数据总线或指令总线同序访问这部分存储器。

3. LP SRAM

LP SRAM 容量为 4 KB，为可读可写的 SRAM。如表 4-1 所示，CPU 可以通过数据总线和指令总线的共用地址段 0x5000_0000 ~ 0x5000_0FFF 访问这部分存储器。

4.3.3 外部存储器

ESP32-H2 支持以 SPI、Dual SPI、Quad SPI、QPI 等接口形式连接片外 flash。ESP32-H2 还支持基于 XTS-AES 算法的硬件手动加密和自动解密功能，从而保护开发者片外 flash 中的程序和数据。

4.3.3.1 外部存储器地址映射

CPU 借助高速缓存 (Cache) 来访问外部存储器。Cache 将根据内存管理单元 (Memory Management Unit, MMU) 中的信息把 CPU 的地址 (0x4200_0000 ~ 0x42FF_FFFF) 映射为访问片外存储的实地址。经过地址映射，ESP32-H2 最大支持 16 MB 的片外 flash。请注意，指令总线地址空间 (16 MB) 和数据总线地址空间 (16 MB) 是共用的。

4.3.3.2 高速缓存

如图 4-2 所示，ESP32-H2 采用一个只读的统一 Cache，为 8 路组相联，容量为 16 KB，块大小为 32 字节。

指令总线和数据总线可以同时访问该 Cache，Cache 经过仲裁对其中一个做出响应。当 Cache 缺失时，Cache 控制器会向外部存储器发起请求。

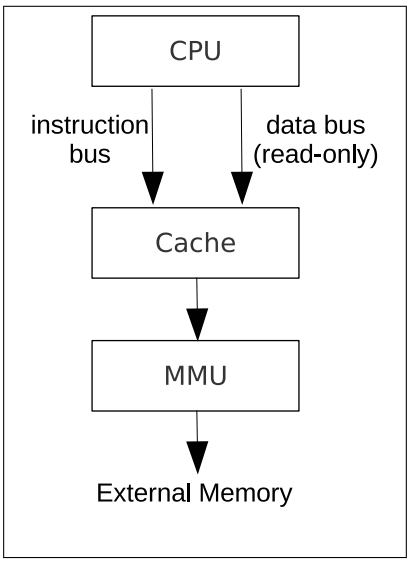


图 4-2. Cache 系统结构

4.3.3.3 Cache 操作

ESP32-H2 Cache 支持如下几种操作：

1. **失效 (Invalidate)**：该操作用于删除 Cache 中的有效数据。该操作完成后，删除的数据将仅存于外部存储器中。如果 CPU 接着去访问该数据，那么需要访问外部存储器。该操作包括两种类型：整体失效 (Invalidate-All) 和手动失效 (Manual-Invalidate)。手动失效仅对 Cache 中落入指定区域的地址对应的数据做失效处理，而整体失效会对 Cache 中的所有数据做失效处理。
2. **预取 (Preload)**：该功能用于将指令和数据提前加载到 Cache 中。预取操作的最小单位为 1 个块。预取分为手动预取 (Manual-Preload) 和自动预取 (Auto-Preload)，手动预取是指硬件按软件指定的虚地址预取一段连续的数据；自动预取是指硬件根据当前命中/缺失（取决于配置）的地址，自动地预取一段连续的数据。

3. **锁定/解锁 (Lock/Unlock)**：该操作用于保护 Cache 中的数据不被替换掉。锁定分为预锁定和手动锁定。预锁定开启时，在填充缺失数据到 Cache 时，如果该数据落在指定区域，Cache 则该数据锁定，未落入指定区域的数据不会被锁定。手动锁定开启时，Cache 检查 Cache 中的数据，并将落在指定区域的数据锁定，未落入指定区域的数据不会被锁定。当缺失发生时，Cache 会优先替换掉未被锁定的那一路的数据，因此锁定区域的数据会一直保存在 Cache 中。但当所有路都被锁定时，Cache 将进行正常替换，就像所有路都没有被锁定一样。解锁是锁定的逆操作，但解锁只有手动解锁。

   请注意，手动失效操作只对未被锁定的数据起作用。如果想对已锁定的数据执行手动失效操作，请先解锁这些数据。

4.3.4 GDMA 地址空间

ESP32-H2 中的 GDMA (General Direct Memory Access) 外设包含三个 TX 通道和三个 RX 通道，可提供直接内存访问 (Direct Memory Access, DMA) 服务，包括：

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

GDMA 可以通过与数据总线完全相同的地址读写 HP SRAM，即 GDMA 通过地址 0x4080_0000 ~ 0x4084_FFFF 访问 HP SRAM。

ESP32-H2 中共有八个外设/模块可以和 GDMA 联合工作。如图 4-3 所示，其中的八根竖线依次对应这八个具有 GDMA 功能的外设/模块，横线表示 GDMA 的某一通道（可为任意通道），竖线与横线的交点表示对应外设/模块可以访问 GDMA 的某一通道。同一行上有多个交点则表示这几个外设/模块不可以同时开启 GDMA 功能。

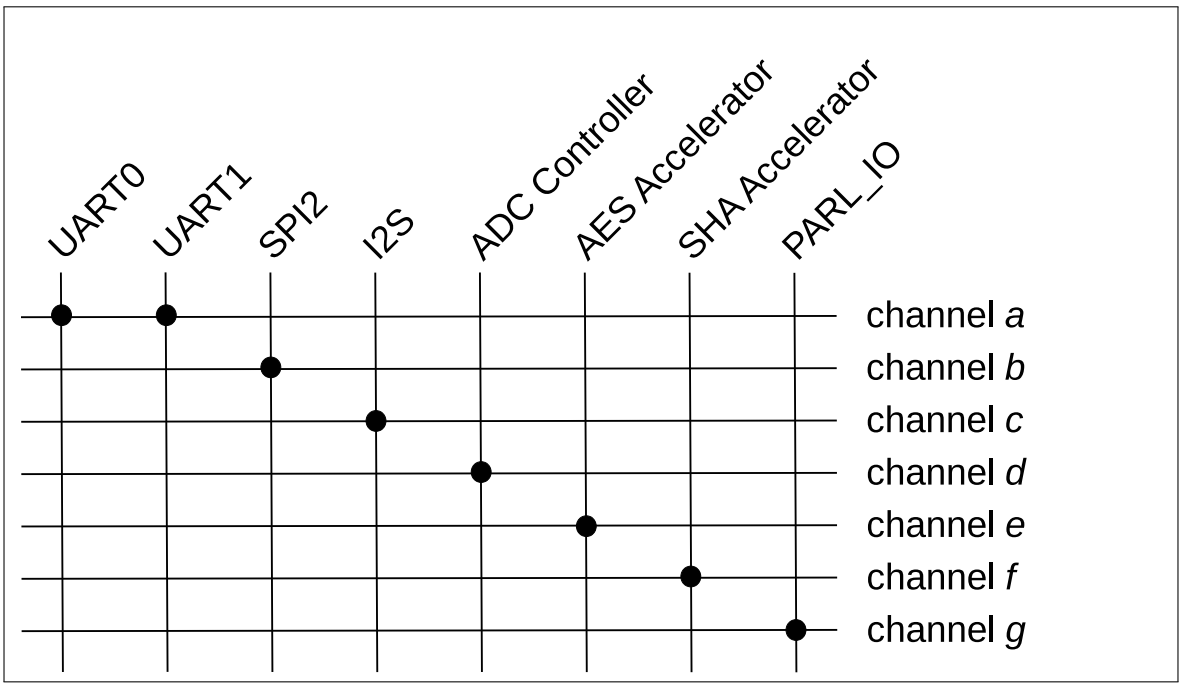


图 4-3. 具有 GDMA 功能的外设/模块

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

说明：
当使用 GDMA 访问任何存储器时，都需要获取对应的访问权限，否则访问将会失败。关于权限管理的更多信息，请参考章节 14 访问权限管理 (APM)。

4.3.5 模块/外设地址空间映射

表 4-2 详细列出了模块/外设地址空间的各段地址与其能访问到的模块/外设的映射关系。其中，“边界地址”（包括低位地址和高位地址）栏中的两列数值共同决定了对应模块/外设的地址空间。

表 4-2. 模块/外设地址空间映射表

目标	边界地址		容量 (KB)
	低位地址	高位地址	
UART 控制器 0 (UART0)	0x6000_0000	0x6000_0FFF	4
UART 控制器 1 (UART1)	0x6000_1000	0x6000_1FFF	4
片外存储器加密与解密 (XTS_AES)	0x6000_2000	0x6000_2FFF	4
保留	0x6000_3000	0x6000_3FFF	
I2C 控制器 0 (I2C0)	0x6000_4000	0x6000_4FFF	4
I2C 控制器 1 (I2C1)	0x6000_5000	0x6000_5FFF	4
UHCI 控制器 (UHCI)	0x6000_6000	0x6000_6FFF	4
红外遥控 (RMT)	0x6000_7000	0x6000_7FFF	4
LED PWM 控制器 (LEDC)	0x6000_8000	0x6000_8FFF	4
定时器组 0 (TIMG 0)	0x6000_9000	0x6000_9FFF	4
定时器组 1 (TIMG 1)	0x6000_A000	0x6000_AFFF	4
系统定时器 (SYSTIMER)	0x6000_B000	0x6000_BFFF	4
双线汽车接口 (TWAI)	0x6000_C000	0x6000_CFFF	4
I2S 控制器 (I2S)	0x6000_D000	0x6000_DFFF	4
逐次逼近型模数转换器 (SAR ADC)	0x6000_E000	0x6000_EFFF	4
USB Serial/JTAG 控制器	0x6000_F000	0x6000_FFFF	4
中断矩阵 (INTMTX)	0x6001_0000	0x6001_0FFF	4
保留	0x6001_1000	0x6001_1FFF	
脉冲计数控制器 (PCNT)	0x6001_2000	0x6001_2FFF	4
事件任务矩阵 (SOC_ETM)	0x6001_3000	0x6001_3FFF	4
电机控制器 (MCPWM)	0x6001_4000	0x6001_4FFF	4
并行 IO 控制器 (PARL_IO)	0x6001_5000	0x6001_5FFF	4
保留	0x6001_6000	0x6007_FFFF	
通用 DMA 控制器 (GDMA)	0x6008_0000	0x6008_0FFF	4
通用 SPI2 控制器 (GP-SPI2)	0x6008_1000	0x6008_1FFF	4
保留	0x6008_2000	0x6008_7FFF	
AES 加速器 (AES)	0x6008_8000	0x6008_8FFF	4
SHA 加速器 (SHA)	0x6008_9000	0x6008_9FFF	4
RSA 加速器 (RSA)	0x6008_A000	0x6008_AFFF	4
ECC 加速器 (ECC)	0x6008_B000	0x6008_BFFF	4
数字签名 (DS)	0x6008_C000	0x6008_CFFF	4

见下页

表 4-2 – 接上页

目标	边界地址		容量 (KB)
	低位地址	高位地址	
HMAC 加速器 (HMAC)	0x6008_D000	0x6008_DFFF	4
ECDSA 加速器 (ECDSA)	0x6008_E000	0x6008_EFFF	4
保留	0x6008_F000	0x6008_FFFF	
IO MUX	0x6009_0000	0x6009_0FFF	4
GPIO 交换矩阵	0x6009_1000	0x6009_1FFF	4
内存访问监控 (MEM_MONITOR)*	0x6009_2000	0x6009_2FFF	4
保留	0x6009_3000	0x6009_4FFF	
高性能系统寄存器 (HP_SYSREG)	0x6009_5000	0x6009_5FFF	4
电源/时钟/复位寄存器 (PCR)	0x6009_6000	0x6009_6FFF	4
保留	0x6009_7000	0x6009_7FFF	
可信执行环境寄存器 (TEE)*	0x6009_8000	0x6009_8FFF	4
访问权限管理 (HP_APM)*	0x6009_9000	0x6009_9FFF	4
保留	0x6009_A000	0x600A_FFFF	
电源管理单元 (PMU)	0x600B_0000	0x600B_03FF	1
低功耗时钟复位寄存器 (LP_CLKRST)	0x600B_0400	0x600B_07FF	1
eFuse 控制器	0x600B_0800	0x600B_0BFF	1
低功耗定时器 (LP_TIMER)	0x600B_0C00	0x600B_0FFF	1
低功耗常开寄存器 (LP_AON)	0x600B_1000	0x600B_13FF	1
保留	0x600B_1400	0x600B_1BFF	
低功耗看门狗定时器 (LP_WDT)	0x600B_1C00	0x600B_1FFF	1
低功耗 IO MUX (LP IO MUX)	0x600B_2000	0x600B_23FF	1
保留	0x600B_2400	0x600B_27FF	
低功耗外设 (LPPERI)	0x600B_2800	0x600B_2BFF	1
低功耗模拟外设 (LP_ANA_PERI)	0x600B_2C00	0x600B_2FFF	1
保留	0x600B_3000	0x600B_37FF	
低功耗访问权限管理 (LP_APM)*	0x600B_3800	0x600B_3BFF	1
保留	0x600B_3C00	0x600B_FFFF	
RISC-V Trace 编码器 (TRACE)	0x600C_0000	0x600C_0FFF	4
保留	0x600C_1000	0x600C_1FFF	
辅助调试 (ASSIST_DEBUG)*	0x600C_2000	0x600C_2FFF	4
保留	0x600C_3000	0x600C_4FFF	
中断优先级寄存器 (INTPRI)	0x600C_5000	0x600C_5FFF	4
保留	0x600C_6000	0x600C_FFFF	

* 该模块/外设的地址空间不连续。

5 eFuse 控制器 (EFUSE)

5.1 概述

ESP32-H2 有一块 4096 位的 eFuse 存储器用于存储用户数据或硬件参数，包括硬件模块的控制参数、校准参数、MAC 地址以及加解密模块使用的密钥等。eFuse 存储器的存储位一旦被烧写为 1，则不能再恢复为 0。用户无法直接访问 eFuse 存储空间，只能通过配置 eFuse 控制器完成对 eFuse 存储器中各个位的烧写与读取。对于有保密需求的数据，可以通过烧写对应的读保护位启用读保护使得用户无法再通过 eFuse 控制器获取 eFuse 存储器中的数据。

5.2 主要特性

- 4096 位一次性可编程存储，有 1792 个保留位供用户使用。
- 烧写保护可配置
- 读取保护可配置
- 使用多种硬件编码方式保护参数内容

5.3 功能描述

5.3.1 结构

eFuse 控制器和 eFuse 存储器共同组成了 eFuse 系统。其内部的数据通路如图 5-1 所示。

用户可以通过将待烧写的数据填入烧写寄存器中并执行烧写指令通过 eFuse 控制器完成对 eFuse 存储器内部各个位的烧写。详细烧写步骤可以参考章节 5.3.2。

用户不能直接读取 eFuse 存储器中烧写的信息内容，因此需要通过 eFuse 控制器将烧写的数据信息读取到对应的地址段的读寄存器内。如果数据在读取过程中出现了和 eFuse 存储器不一致的错误，eFuse 控制器还可以通过硬件编码机制对数据进行自动校正（详见章节 5.3.1.4 ），并将错误信息自动更新至错误报告寄存器中。详细的读取参数步骤可以参考章节 5.3.3。

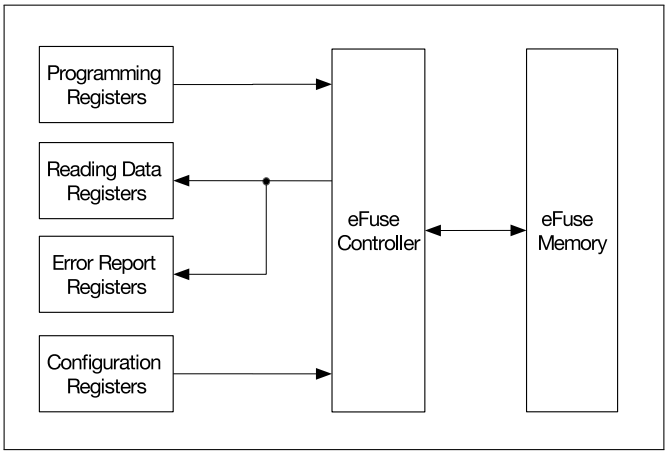


图 5-1. eFuse 系统数据通路

eFuse 存储器从结构上分成 11 个 BLOCK (BLOCK0 ~ BLOCK10)。

BLOCK0 存储了大部分供软硬件使用的控制参数。

表 5-1 列出了用户可访问（可读并可用）的所有 BLOCK0 中的参数名称、位宽、是否直接驱动硬件模块、烧写保护位、以及简要的功能描述，若需了解更多有关该参数的描述，请点击表格中对应参数的链接。

在这些参数中，**EFUSE_WR_DIS** 用于控制其他参数的烧写保护状态，**EFUSE_RD_DIS** 用于控制 BLOCK4 ~ BLOCK10 的读取保护状态。更多关于这两个参数的信息请见章节 5.3.1.2、5.3.1.3。

表 5-1. BLOCK0 参数

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_WR_DIS	32	Y	N/A	表示是否禁止 eFuse 控制器烧写对应参数
EFUSE_RD_DIS	7	Y	0	表示是否禁止用户读取 eFuse 存储器中 BLOCK4 ~ 10 的内容
EFUSE_DIS_ICACHE	1	Y	2	表示是否关闭指令 cache
EFUSE_DIS_USB_JTAG	1	Y	2	表示是否关闭 USB 模块中 USB 转 JTAG 的功能
EFUSE_POWERGLITCH_EN	1	Y	2	表示是否启用电源毛刺检测功能
EFUSE_DIS_FORCE_DOWNLOAD	1	Y	2	表示是否关闭强制芯片进入 Download 模式的功能
EFUSE_SPI_DOWNLOAD_MSPI_DIS	1	Y	17	表示在 boot_mode_download 过程中是否关闭 SPI0 控制器
EFUSE_DIS_TWAI	1	Y	2	表示是否关闭 TWAI 控制器功能
EFUSE_JTAG_SEL_ENABLE	1	Y	2	表示 EFUSE_DIS_PAD_JTAG 和 EFUSE_DIS_USB_JTAG 均配置为 0 时，是否使能通过 GPIO15 的 strapping 值选择 JTAG 的信号源
EFUSE_SOFT_DIS_JTAG	3	Y	31	表示是否软禁用 JTAG
EFUSE_DIS_PAD_JTAG	1	Y	2	表示是否硬禁用 JTAG（永久）
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	1	Y	2	表示是否禁用 flash 加密功能（SPI 启动模式除外）
EFUSE_USB_EXCHG_PINS	1	Y	30	表示是否交换 USB D+/D- 管脚
EFUSE_VDD_SPI_AS_GPIO	1	Y	30	表示是否将 VDD SPI 管脚用作普通 GPIO 管脚
EFUSE_WDT_DELAY_SEL	2	Y	3	表示启动时是否选择 RTC 看门狗超时阈值
EFUSE_SPI_BOOT_CRYPT_CNT	3	Y	4	表示是否使能 SPI boot 加解密
EFUSE_SECURE_BOOT_KEY_REVOKE0	1	N	5	表示是否撤销第一个安全启动 (Secure Boot) 密钥
EFUSE_SECURE_BOOT_KEY_REVOKE1	1	N	6	表示是否撤销第二个安全启动密钥
EFUSE_SECURE_BOOT_KEY_REVOKE2	1	N	7	表示是否撤销第三个安全启动密钥
EFUSE_KEY_PURPOSE_0	4	Y	8	表示 Key0 用途 (purpose)，见表 5-2
EFUSE_KEY_PURPOSE_1	4	Y	9	表示 Key1 用途，见表 5-2
EFUSE_KEY_PURPOSE_2	4	Y	10	表示 Key2 用途，见表 5-2
EFUSE_KEY_PURPOSE_3	4	Y	11	表示 Key3 用途，见表 5-2

见下页

GoBack

表 5-1 – 接上页

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_KEY_PURPOSE_4	4	Y	12	表示 Key4 用途，见表 5-2
EFUSE_KEY_PURPOSE_5	4	Y	13	表示 Key5 用途，见表 5-2
EFUSE_SEC_DPA_LEVEL	2	Y	14	表示防差分功耗分析 (differential power analysis, DPA) 攻击的安全级别
EFUSE_ECDSA_FORCE_USE_HARDWARE_K	1	Y	17	表示是否在 ECDSA 模块中强制使用硬件生成的随机值 K
EFUSE_CRYPT_DPA_ENABLE	1	Y	15	表示开启防 DPA 攻击功能
EFUSE_SECURE_BOOT_EN	1	N	16	表示是否使能安全启动
EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE	1	N	16	表示是否使 Secure Boot 的撤销采用激进策略
EFUSE_FLASH_TPUW	4	N	18	表示上电后 flash 等待时间
EFUSE_DIS_DOWNLOAD_MODE	1	N	18	表示是否关闭所有 download 模式
EFUSE_DIS_DIRECT_BOOT	1	N	18	表示是否关闭 direct boot 模式
EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT	1	N	18	表示是否关闭 ROM boot 过程中的 USB-Serial-JTAG 打印
EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE	1	N	18	表示是否禁用 USB-Serial-JTAG 下载模式
EFUSE_ENABLE_SECURITY_DOWNLOAD	1	N	18	表示是否使能安全下载
EFUSE_UART_PRINT_CONTROL	2	N	18	表示 UART 打印模式
EFUSE_FORCE_SEND_RESUME	1	N	18	表示是否强制 ROM 代码在 SPI 启动过程中发送恢复指令
EFUSE_SECURE_VERSION	16	N	18	表示安全版本，用于 ESP-IDF 的防回滚功能
EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE	1	N	18	表示当 Secure Boot 开启时是否关闭”fast verify on wake”
EFUSE_HYS_EN_PAD0	6	Y	19	表示是否使能 PAD0-5 的迟滞功能
EFUSE_HYS_EN_PAD1	22	Y	19	表示是否使能 PAD6-27 的迟滞功能

表 5-2 为密钥用途各个数值对应的含义。通过配置参数 EFUSE_KEY_PURPOSE_*n* 来声明 KEY*n* 用途 (*n*: 0 ~ 5)。

表 5-2. 密钥用途数值对应的含义

密钥用途数值	含义
0	指定为用户使用
1	指定为 ECDSA_KEY 使用
2	保留
3	保留
4	指定为 XTS_AES_128_KEY 使用（用于 flash/SRAM 加解密）
5	指定为 HMAC Downstream（下行）模式（JTAG 和数字签名算法）使用
6	指定为 HMAC Downstream 模式下的 JTAG 使用
7	指定为 HMAC Downstream 模式下的数字签名算法使用
8	指定为 HMAC Upstream（上行）模式使用
9	指定为 SECURE_BOOT_DIGEST0 使用（secure boot 密钥摘要）
10	指定为 SECURE_BOOT_DIGEST1 使用（secure boot 密钥摘要）
11	指定为 SECURE_BOOT_DIGEST2 使用（secure boot 密钥摘要）

表 5-3 列出了 BLOCK1 ~ BLOCK10 中存储的参数的信息。

表 5-3. BLOCK1-10 参数

块 (block)	参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	EFUSE_RD_DIS 读取保护位	描述
BLOCK1	EFUSE_MAC	48	N	20	N/A	MAC 地址
	EFUSE_MAC_EXT	16	N	20	N/A	MAC 拓展地址
	EFUSE_SYS_DATA_PART0	69	N	20	N/A	系统数据
BLOCK2	EFUSE_SYS_DATA_PART1	256	N	21	N/A	系统数据
BLOCK3	EFUSE_USR_DATA	256	N	22	N/A	用户数据
BLOCK4	EFUSE_KEY0_DATA	256	Y	23	0	KEY0 或用户数据
BLOCK5	EFUSE_KEY1_DATA	256	Y	24	1	KEY1 或用户数据
BLOCK6	EFUSE_KEY2_DATA	256	Y	25	2	KEY2 或用户数据
BLOCK7	EFUSE_KEY3_DATA	256	Y	26	3	KEY3 或用户数据
BLOCK8	EFUSE_KEY4_DATA	256	Y	27	4	KEY4 或用户数据
BLOCK9	EFUSE_KEY5_DATA	256	Y	28	5	KEY5 或用户数据
BLOCK10	EFUSE_SYS_DATA_PART2	256	N	29	6	系统数据

其中，BLOCK4 ~ 9 可用于存储 KEY0 ~ 5，表示 eFuse 中至多可以烧写 6 个 256 位的密钥。每烧写一个密钥，还需要烧写该密钥用途的数值（见表 5-2）。例如，用户将用于 HMAC Downstream 模式下的 JTAG 功能的密钥烧写到 KEY3（即 BLOCK7），还需要将密钥用途的数值 6 烧写到 EFUSE_KEY_PURPOSE_3。

说明：

请不要将 XTS-AES 密钥或 ECDSA 密钥烧写进 KEY5 BLOCK，即 BLOCK9，否则可能发生密钥无法读取的情况。建议首先将 XTS-AES 密钥和 ECDSA 密钥烧写到前几个 BLOCK 中，即 BLOCK4 ~ BLOCK8，最后一个 BLOCK，即 BLOCK9，用来烧写其他密钥。

BLOCK1 ~ BLOCK10 均采用 RS 编码方式，因此参数烧写受到一定的限制，具体请参考章节 5.3.1.4 和章节 5.3.2。

5.3.1.1 硬件模块使用参数

硬件模块使用参数是通过电路连接实现的，对于一个硬件模块使用参数而言，如果烧写的数值发生变化，那么该参数控制的外设行为将直接受到影响。这个过程无法被用户干预。硬件使用的参数为表 5-1 和 5-3 “硬件使用”一栏中标记为“Y”的参数。

5.3.1.2 EFUSE_WR_DIS

参数 EFUSE_WR_DIS 决定了 eFuse 存储器中所有的参数是否处于烧写保护状态。烧写完 EFUSE_WR_DIS 参数后，需要更新 eFuse 控制器的读寄存器以保证烧写保护状态生效。

表 5-1 以及表 5-3 中的 “EFUSE_WR_DIS 烧写保护位”列描述了各参数的烧写保护状态具体由 EFUSE_WR_DIS 的哪个位决定。

当某个参数对应的烧写保护位为 0 时，表示此参数未处于烧写保护状态，可以烧写该参数，但已经被烧写的参数不能被重复烧写。

当某个参数对应的烧写保护位为 1 时，表示此参数处于烧写保护状态，此参数的每一个位都无法被更改，未被烧写的位永远为 0，已经被烧写的位永远为 1。因此如果某个参数已经处于烧写保护状态了，则会一直处在该状态，无法再更改。

5.3.1.3 EFUSE_RD_DIS

所有参数中，只有 BLOCK4 ~ BLOCK10 的参数可以被配置为用户读取保护状态，即表 5-3 中 “EFUSE_RD_DIS 读取保护”列非 “N/A” 的参数。烧写完 EFUSE_RD_DIS 参数后，需要更新 eFuse 控制器的读寄存器以保证读取保护状态生效。

参数 EFUSE_RD_DIS 中的某个位为 0，表示此位管理的参数未处于用户读取保护状态；某个位为 1，表示此位管理的参数处于用户读取保护状态。

除 BLOCK4 ~ BLOCK10 之外，其他参数不受读取保护状态的约束，均可被用户读取。

BLOCK4 ~ BLOCK10 即使被配置处于读取保护状态，仍然可以通过设置 EFUSE_KEY_PURPOSE_*n* 供硬件加密模块在内部使用。

5.3.1.4 数据存储方式

eFuse 使用硬件编码机制保护数据，对用户不可见。

BLOCK0 除 EFUSE_WR_DIS字段外均使用 4 备份方式存储参数，即 BLOCK0 中的参数均在 eFuse 存储器中存储了 4 份。4 备份机制对用户不可见。

BLOCK0 中 EFUSE_WR_DIS 为 32 比特，其余参数为 152 比特，因此 BLOCK0 在 eFuse 存储器中共占据了 32 + 152 * 4 = 640 比特的存储空间。

BLOCK1 ~ BLOCK10 使用 Reed-Solomon 编码存储参数，编码方式为 RS(44, 32)，最多支持自动纠错 6 个字节。本文 RS (44, 32) 使用的本源多项式为 $p(x) = x^8 + x^4 + x^3 + x^2 + 1$。

如图 5-2 和 5-3 所示，移位寄存器电路对 32 字节参数进行 RS (44, 32) 编码处理，将 32 字节数据处理为 44 字节，其中：

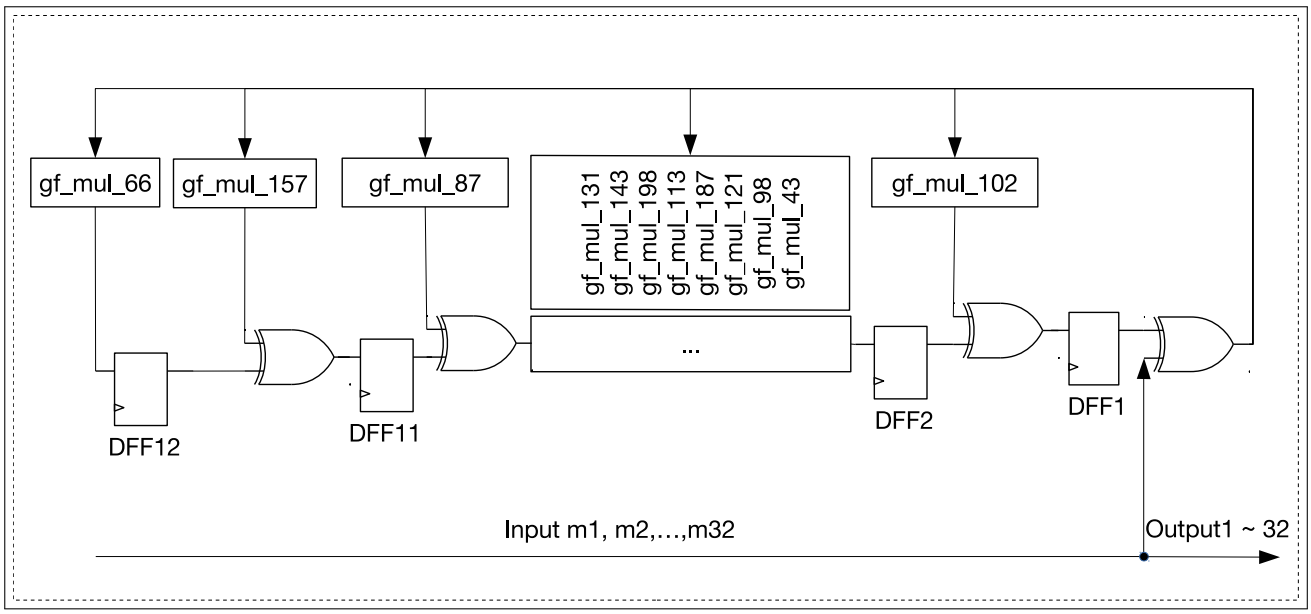


图 5-2. 移位寄存器电路图（前 32 字节）

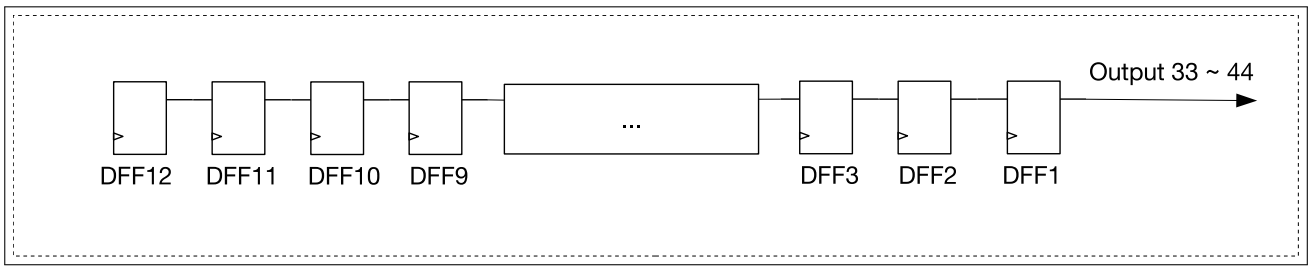


图 5-3. 移位寄存器电路图（后 12 字节）

- 字节 [0:31] 为数据本身
- 字节 [32:43] 为储存在 8 位触发器 DFF1, DFF2, ..., DFF12 中的奇偶校验字节（gf_mul_n 为 $GF(2^8)$ 域中某一字节数据与元素 α^n 相乘的结果，n 为整数）

然后，硬件将这 44 字节数据一起烧入 eFuse。eFuse 控制器会在读 eFuse 的过程中自动完成解码和纠错。

由于 RS 校验码是在整个 32 字节的 eFuse BLOCK 上生成的，因此每个 BLOCK 只能写入一次。

BLOCK1 由于数据不足 32 字节，因此不足 32 字节的部分在 RS（44，32）编码时硬件会将其视为 0，不会影响最终的编码结果。

使用 RS（44，32）编码方式的 BLOCK 中，BLOCK1 数据参数为 24 字节，RS 校验码为 12 字节，因此 BLOCK1 在 eFuse 存储器中共占据了 24 + 12 = 36 字节的存储空间。

其余 BLOCK (BLOCK2 ~ 10) 的数据参数均为 32 字节，RS 校验码为 12 字节，因此在 eFuse 存储器中共占据了（32 + 12）* 9 = 396 字节的存储空间

5.3.2 烧写参数

烧写 eFuse 参数时，需要按 BLOCK 烧写。BLOCK0 ~ BLOCK10 共用同一段地址来存储即将烧写的参数。通过配置 EFUSE_BLK_NUM 参数表明当前需要烧写的是哪一个 BLOCK。

由于读取数据的寄存器和烧写数据的寄存器在位置上存在一一对应的关系（详见表 5-4），因此用户可以通过查找读取寄存器的参数描述和位置确定待烧写数据在烧写寄存器中的具体位置。

例如，用户想将 BLOCK0 中的 EFUSE_DIS_ICACHE 烧写成 1，可以先查找 BLOCK0 读取数据的寄存器 EFUSE_RD_REPEAT_DATA0 ~ 4_REG，定位到 EFUSE_DIS_ICACHE 参数位于 EFUSE_RD_REPEAT_DATA0_REG 寄存器的第 8 个比特，因此用户可以将 EFUSE_PGM_DATA1_REG 待烧写数据的第 8 个比特写成 1，并执行烧写步骤，待烧写步骤完成后，eFuse 存储器中的对应位将被成功烧写为 1。

烧写准备

- **烧写 BLOCK0**
  1. 将寄存器域 EFUSE_BLK_NUM 配置为 0。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG 配置 BLOCK0 即将烧写的参数。EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG 以及 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中的数据不影响 BLOCK0 的烧写。
- **烧写 BLOCK1**
  1. 将寄存器域 EFUSE_BLK_NUM 配置为 1。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA5_REG 配置 BLOCK1 即将烧写的参数，EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 配置对应的 RS 校验码。
     EFUSE_PGM_DATA6_REG ~ EFUSE_PGM_DATA7_REG 中的数据不影响 BLOCK1 的烧写。软件计算 BLOCK1 的 RS 校验码时，应当视这 8 个字节的数据为 0。
- **烧写 BLOCK2 ~ 10**
  1. 将寄存器域 EFUSE_BLK_NUM 配置为烧写的 BLOCK 数值。
  2. EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 配置即将烧写的参数，EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中配置对应的 RS 校验码。

烧写流程

烧写参数的流程如下：

1. 配置 EFUSE_BLK_NUM 参数，决定烧写哪一个 BLOCK。
2. 将需要烧写的参数填写到寄存器 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中。
3. 确保 eFuse 烧写电压 VDDQ 的配置正确，具体请参考章节 5.3.4。
4. 配置寄存器 EFUSE_CONF_REG 的 EFUSE_OP_CODE 位域为 0x5A5A。
5. 配置寄存器 EFUSE_CMD_REG 的 EFUSE_PGM_CMD 位域为 1。
6. 轮询寄存器 EFUSE_CMD_REG 直到其为 0x0，或者等待 PGM_DONE（烧写完成）中断产生。识别 PGM_DONE 或 READ_DONE （读取完成）中断产生的方法详见章节 5.3.3 最后的说明。
7. 将 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中写入的参数清零。
8. 执行更新 eFuse 控制器读寄存器操作使写入的新值生效，具体请参考章节 5.3.3。

9. 检查错误寄存器内容。若读取错误寄存器内数值不为 0，需要再次执行上述步骤 1 ~ 7 重新烧写一次。

   通过该方式可以解决由于烧写不充分导致错误寄存器内数值不为 0 的问题。

   对于不同的 eFuse BLOCK，需要检查的错误寄存器如下。

   - BLOCK0：EFUSE_RD_REPEAT_ERR0_REG ~ EFUSE_RD_REPEAT_ERR4_REG
   - BLOCK1：EFUSE_MAC_SYS_ERR_NUM、EFUSE_MAC_SYS_FAIL
   - BLOCK2：EFUSE_SYS_PART1_ERR_NUM、EFUSE_SYS_PART1_FAIL
   - BLOCK3：EFUSE_USR_DATA_ERR_NUM、EFUSE_USR_DATA_FAIL
   - BLOCK4：EFUSE_KEY0_ERR_NUM、EFUSE_KEY0_FAIL
   - BLOCK5：EFUSE_KEY1_ERR_NUM、EFUSE_KEY1_FAIL
   - BLOCK6：EFUSE_KEY2_ERR_NUM、EFUSE_KEY2_FAIL
   - BLOCK7：EFUSE_KEY3_ERR_NUM、EFUSE_KEY3_FAIL
   - BLOCK8：EFUSE_KEY4_ERR_NUM、EFUSE_KEY4_FAIL
   - BLOCK9：EFUSE_KEY5_ERR_NUM、EFUSE_KEY5_FAIL
   - BLOCK10：EFUSE_SYS_PART2_ERR_NUM, EFUSE_SYS_PART2_FAIL

限制

BLOCK0 中不同的参数，甚至对于同一个参数中的不同位可以在多次烧写中分别完成。但是并不推荐这样做，而是建议尽量减少烧写次数。我们建议对于某个参数中的所有需要烧写的位都在一次烧写中完成。并且当 EFUSE_WR_DIS 的某个位管理的所有参数都烧写之后，就立即烧写 EFUSE_WR_DIS 的这个位。甚至可以在同一次烧写中既烧写 EFUSE_WR_DIS 的某个位管理的所有参数，同时也烧写 EFUSE_WR_DIS 的这个位。

BLOCK1 中数据信息在出厂时已经烧写完毕，不允许再次烧写。

BLOCK2 ~ 10 中每一个 BLOCK 都只能烧写一次，不允许重复烧写。

5.3.3 用户读取参数

用户不能直接读取 eFuse 存储器中烧写的数据。eFuse 控制器能够将烧写的数据信息读取到对应的数据读取寄存器 (以 EFUSE_RD_ 为前缀的寄存器)，用户可以通过读取这些寄存器来获取 eFuse 存储器信息。下表 5-4 列出了读取数据的寄存器名称以及对应烧写时的烧写寄存器名称。

表 5-4. 用户读取寄存器信息

BLOCK	读寄存器	烧写寄存器
0	EFUSE_RD_WR_DIS_REG	EFUSE_PGM_DATA0_REG
0	EFUSE_RD_REPEAT_DATA0 ~ 4_REG	EFUSE_PGM_DATA1 ~ 5_REG
1	EFUSE_RD_MAC_SYS_0 ~ 5_REG	EFUSE_PGM_DATA0 ~ 5_REG
2	EFUSE_RD_SYS_PART1_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_USR_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
4-9	EFUSE_RD_KEY*n*_DATA0 ~ 7_REG (*n*: 0 ~ 5)	EFUSE_PGM_DATA0 ~ 7_REG
10	EFUSE_RD_SYS_PART2_0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG

更新 eFuse 控制器的读寄存器

eFuse 控制器通过读取 eFuse 存储器来更新相应寄存器的数据。读取操作在系统复位时会执行一次，也可以根据需要由用户手动触发（例如在需要读取新烧写 eFuse 存储器中的数据内容时）。用户触发 eFuse 存储器读取操作的流程如下：

1. 配置寄存器 EFUSE_CONF_REG 的 EFUSE_OP_CODE 位域为 0x5AA5。
2. 配置寄存器 EFUSE_CMD_REG 的 EFUSE_READ_CMD 位域为 1。
3. 轮询寄存器 EFUSE_CMD_REG 直到其为 0x0，或者等待 READ_DONE 中断产生，识别 PGM_DONE 或 READ_DONE 中断产生的方法详见下方说明。
4. 用户从 eFuse 存储器中读取参数的值。

eFuse 控制器的读寄存器中的数值将一直保持到下一次执行更新 eFuse 存储器读取操作。

烧写错误检测

烧写错误记录寄存器允许用户检测 eFuse 存储器中的参数和 eFuse 控制器读取的参数是否存在不一致的错误。

EFUSE_RD_REPEAT_ERR0 ~ 3_REG 寄存器用于指示 BLOCK0 中除了 EFUSE_WR_DIS 外的其他参数的烧写是否出错（对应位为 1 代表烧写出错；为 0 代表烧写正确）。

EFUSE_RD_RS_ERR0 ~ 1_REG 寄存器记录 eFuse 控制器读 BLOCK1 ~ BLOCK10 过程中，纠错的字节数目以及 RS 解码是否失败的信息。

每次更新 eFuse 控制器读寄存器操作完成之后，上述寄存器内的数值都会被更新。

识别烧写/读取操作完成

识别烧写/读取操作完成的方法如下。位 1 对应烧写操作，位 0 对应读取操作。

- 方法 1：轮询寄存器 EFUSE_INT_RAW_REG 的位
- 方法 2:
  1. 将寄存器 EFUSE_INT_ENA_REG 的位 1/0 置 1，使 eFuse 控制器能够产生 PGM_DONE 或 READ_DONE 中断。
  2. 配置中断矩阵使 CPU 能够响应 eFuse 的中断信号，可参见 9 中断矩阵 *(INTMTX)*。
  3. 等待 PGM_DONE 或 READ_DONE 中断产生。
  4. 对寄存器 EFUSE_INT_CLR_REG 的位 1/0 置 1 以清除 PGM_DONE 或 READ_DONE 中断。

注意事项

在 eFuse 控制器执行寄存器更新操作过程中，会复用 EFUSE_PGM_DATA*n*_REG (n=0, 1, ..., 7) 寄存器的存储空间，所以在启动 eFuse 控制器更新寄存器之前，不要将有意义的数据写入上述寄存器中。
芯片启动过程中，eFuse 控制器会自动更新 eFuse 存储器数据到用户可访问的寄存器。用户可以通过读取相应的寄存器获取 eFuse 存储器内烧写的数据。因此，用户无需再驱动 eFuse 控制器执行读更新操作。

5.3.4 eFuse VDDQ 时序

eFuse 控制器工作在 20 MHz 时钟频率下，其烧写电压 VDDQ 的配置参数需要满足以下条件：

- EFUSE_DAC_NUM（烧写电压上升周期数），默认烧写电压为 2.5 V，每个上升周期增加 0.01 V，该参数对应的默认值为 255；
- EFUSE_DAC_CLK_DIV（烧写电压时钟分频系数），要求烧写电压时钟周期大于 1 μs；

- EFUSE_PWR_ON_NUM（eFuse 烧写电压上电等待时间），要求该等待时间结束后烧写电压已稳定，即要求配置数值大于 EFUSE_DAC_CLK_DIV * EFUSE_DAC_NUM；
- EFUSE_PWR_OFF_NUM（烧写电压掉电等待时间），要求该时间大于 10 μs；

表 5-5. VDDQ 默认时序参数配置

EFUSE_DAC_NUM	EFUSE_DAC_CLK_DIV	EFUSE_PWR_ON_NUM	EFUSE_PWR_OFF_NUM
0xFF	0x28	0x3000	0x190

5.3.5 中断

- PGM_DONE 中断：当 eFuse 烧写完成后，此中断被触发。如果要启动该中断信号，需将寄存器 EFUSE_INT_ENA_REG 的 EFUSE_PGM_DONE_INT_ENA 域置 1。
- READ_DONE 中断：当 eFuse 读取完成后，此中断被触发。如果要启动该中断信号，需将寄存器 EFUSE_INT_ENA_REG 的 EFUSE_READ_DONE_INT_ENA 域置 1。

5.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
烧写数据寄存器			
EFUSE_PGM_DATA0_REG	存放待烧写数据的第 0 个寄存器内容	0x0000	R/W
EFUSE_PGM_DATA1_REG	存放待烧写数据的第 1 个寄存器内容	0x0004	R/W
EFUSE_PGM_DATA2_REG	存放待烧写数据的第 2 个寄存器内容	0x0008	R/W
EFUSE_PGM_DATA3_REG	存放待烧写数据的第 3 个寄存器内容	0x000C	R/W
EFUSE_PGM_DATA4_REG	存放待烧写数据的第 4 个寄存器内容	0x0010	R/W
EFUSE_PGM_DATA5_REG	存放待烧写数据的第 5 个寄存器内容	0x0014	R/W
EFUSE_PGM_DATA6_REG	存放待烧写数据的第 6 个寄存器内容	0x0018	R/W
EFUSE_PGM_DATA7_REG	存放待烧写数据的第 7 个寄存器内容	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	存放待烧写 RS 代码的第 0 个寄存器数据	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	存放待烧写 RS 代码的第 1 个寄存器数据	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	存放待烧写 RS 代码的第 2 个寄存器数据	0x0028	R/W
读取数据寄存器			
EFUSE_RD_WR_DIS_REG	BLOCK0 的第 0 个寄存器内容	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	BLOCK0 的第 1 个寄存器内容	0x0030	RO
EFUSE_RD_REPEAT_DATA1_REG	BLOCK0 的第 2 个寄存器内容	0x0034	RO
EFUSE_RD_REPEAT_DATA2_REG	BLOCK0 的第 3 个寄存器内容	0x0038	RO
EFUSE_RD_REPEAT_DATA3_REG	BLOCK0 的第 4 个寄存器内容	0x003C	RO
EFUSE_RD_REPEAT_DATA4_REG	BLOCK0 的第 5 个寄存器内容	0x0040	RO
EFUSE_RD_MAC_SYS_0_REG	BLOCK1 的第 0 个寄存器内容	0x0044	RO
EFUSE_RD_MAC_SYS_1_REG	BLOCK1 的第 1 个寄存器内容	0x0048	RO
EFUSE_RD_MAC_SYS_2_REG	BLOCK1 的第 2 个寄存器内容	0x004C	RO
EFUSE_RD_MAC_SYS_3_REG	BLOCK1 的第 3 个寄存器内容	0x0050	RO
EFUSE_RD_MAC_SYS_4_REG	BLOCK1 的第 4 个寄存器内容	0x0054	RO
EFUSE_RD_MAC_SYS_5_REG	BLOCK1 的第 5 个寄存器内容	0x0058	RO
EFUSE_RD_SYS_PART1_DATA0_REG	BLOCK2 (system) 的第 0 个寄存器内容	0x005C	RO
EFUSE_RD_SYS_PART1_DATA1_REG	BLOCK2 (system) 的第 1 个寄存器内容	0x0060	RO
EFUSE_RD_SYS_PART1_DATA2_REG	BLOCK2 (system) 的第 2 个寄存器内容	0x0064	RO
EFUSE_RD_SYS_PART1_DATA3_REG	BLOCK2 (system) 的第 3 个寄存器内容	0x0068	RO
EFUSE_RD_SYS_PART1_DATA4_REG	BLOCK2 (system) 的第 4 个寄存器内容	0x006C	RO
EFUSE_RD_SYS_PART1_DATA5_REG	BLOCK2 (system) 的第 5 个寄存器内容	0x0070	RO
EFUSE_RD_SYS_PART1_DATA6_REG	BLOCK2 (system) 的第 6 个寄存器内容	0x0074	RO
EFUSE_RD_SYS_PART1_DATA7_REG	BLOCK2 (system) 的第 7 个寄存器内容	0x0078	RO
EFUSE_RD_USR_DATA0_REG	BLOCK3 (user) 的第 0 个寄存器内容	0x007C	RO
EFUSE_RD_USR_DATA1_REG	BLOCK3 (user) 的第 1 个寄存器内容	0x0080	RO
EFUSE_RD_USR_DATA2_REG	BLOCK3 (user) 的第 2 个寄存器内容	0x0084	RO
EFUSE_RD_USR_DATA3_REG	BLOCK3 (user) 的第 3 个寄存器内容	0x0088	RO
EFUSE_RD_USR_DATA4_REG	BLOCK3 (user) 的第 4 个寄存器内容	0x008C	RO

名称	描述	地址	访问
EFUSE_RD_USR_DATA5_REG	BLOCK3 (user) 的第 5 个寄存器内容	0x0090	RO
EFUSE_RD_USR_DATA6_REG	BLOCK3 (user) 的第 6 个寄存器内容	0x0094	RO
EFUSE_RD_USR_DATA7_REG	BLOCK3 (user) 的第 7 个寄存器内容	0x0098	RO
EFUSE_RD_KEY0_DATA0_REG	BLOCK4 (KEY0) 的第 0 个寄存器内容	0x009C	RO
EFUSE_RD_KEY0_DATA1_REG	BLOCK4 (KEY0) 的第 1 个寄存器内容	0x00A0	RO
EFUSE_RD_KEY0_DATA2_REG	BLOCK4 (KEY0) 的第 2 个寄存器内容	0x00A4	RO
EFUSE_RD_KEY0_DATA3_REG	BLOCK4 (KEY0) 的第 3 个寄存器内容	0x00A8	RO
EFUSE_RD_KEY0_DATA4_REG	BLOCK4 (KEY0) 的第 4 个寄存器内容	0x00AC	RO
EFUSE_RD_KEY0_DATA5_REG	BLOCK4 (KEY0) 的第 5 个寄存器内容	0x00B0	RO
EFUSE_RD_KEY0_DATA6_REG	BLOCK4 (KEY0) 的第 6 个寄存器内容	0x00B4	RO
EFUSE_RD_KEY0_DATA7_REG	BLOCK4 (KEY0) 的第 7 个寄存器内容	0x00B8	RO
EFUSE_RD_KEY1_DATA0_REG	BLOCK5 (KEY1) 的第 0 个寄存器内容	0x00BC	RO
EFUSE_RD_KEY1_DATA1_REG	BLOCK5 (KEY1) 的第 1 个寄存器内容	0x00C0	RO
EFUSE_RD_KEY1_DATA2_REG	BLOCK5 (KEY1) 的第 2 个寄存器内容	0x00C4	RO
EFUSE_RD_KEY1_DATA3_REG	BLOCK5 (KEY1) 的第 3 个寄存器内容	0x00C8	RO
EFUSE_RD_KEY1_DATA4_REG	BLOCK5 (KEY1) 的第 4 个寄存器内容	0x00CC	RO
EFUSE_RD_KEY1_DATA5_REG	BLOCK5 (KEY1) 的第 5 个寄存器内容	0x00D0	RO
EFUSE_RD_KEY1_DATA6_REG	BLOCK5 (KEY1) 的第 6 个寄存器内容	0x00D4	RO
EFUSE_RD_KEY1_DATA7_REG	BLOCK5 (KEY1) 的第 7 个寄存器内容	0x00D8	RO
EFUSE_RD_KEY2_DATA0_REG	BLOCK6 (KEY2) 的第 0 个寄存器内容	0x00DC	RO
EFUSE_RD_KEY2_DATA1_REG	BLOCK6 (KEY2) 的第 1 个寄存器内容	0x00E0	RO
EFUSE_RD_KEY2_DATA2_REG	BLOCK6 (KEY2) 的第 2 个寄存器内容	0x00E4	RO
EFUSE_RD_KEY2_DATA3_REG	BLOCK6 (KEY2) 的第 3 个寄存器内容	0x00E8	RO
EFUSE_RD_KEY2_DATA4_REG	BLOCK6 (KEY2) 的第 4 个寄存器内容	0x00EC	RO
EFUSE_RD_KEY2_DATA5_REG	BLOCK6 (KEY2) 的第 5 个寄存器内容	0x00F0	RO
EFUSE_RD_KEY2_DATA6_REG	BLOCK6 (KEY2) 的第 6 个寄存器内容	0x00F4	RO
EFUSE_RD_KEY2_DATA7_REG	BLOCK6 (KEY2) 的第 7 个寄存器内容	0x00F8	RO
EFUSE_RD_KEY3_DATA0_REG	BLOCK7 (KEY3) 的第 0 个寄存器内容	0x00FC	RO
EFUSE_RD_KEY3_DATA1_REG	BLOCK7 (KEY3) 的第 1 个寄存器内容	0x0100	RO
EFUSE_RD_KEY3_DATA2_REG	BLOCK7 (KEY3) 的第 2 个寄存器内容	0x0104	RO
EFUSE_RD_KEY3_DATA3_REG	BLOCK7 (KEY3) 的第 3 个寄存器内容	0x0108	RO
EFUSE_RD_KEY3_DATA4_REG	BLOCK7 (KEY3) 的第 4 个寄存器内容	0x010C	RO
EFUSE_RD_KEY3_DATA5_REG	BLOCK7 (KEY3) 的第 5 个寄存器内容	0x0110	RO
EFUSE_RD_KEY3_DATA6_REG	BLOCK7 (KEY3) 的第 6 个寄存器内容	0x0114	RO
EFUSE_RD_KEY3_DATA7_REG	BLOCK7 (KEY3) 的第 7 个寄存器内容	0x0118	RO
EFUSE_RD_KEY4_DATA0_REG	BLOCK8 (KEY4) 的第 0 个寄存器内容	0x011C	RO
EFUSE_RD_KEY4_DATA1_REG	BLOCK8 (KEY4) 的第 1 个寄存器内容	0x0120	RO
EFUSE_RD_KEY4_DATA2_REG	BLOCK8 (KEY4) 的第 2 个寄存器内容	0x0124	RO
EFUSE_RD_KEY4_DATA3_REG	BLOCK8 (KEY4) 的第 3 个寄存器内容	0x0128	RO
EFUSE_RD_KEY4_DATA4_REG	BLOCK8 (KEY4) 的第 4 个寄存器内容	0x012C	RO
EFUSE_RD_KEY4_DATA5_REG	BLOCK8 (KEY4) 的第 5 个寄存器内容	0x0130	RO
EFUSE_RD_KEY4_DATA6_REG	BLOCK8 (KEY4) 的第 6 个寄存器内容	0x0134	RO
EFUSE_RD_KEY4_DATA7_REG	BLOCK8 (KEY4) 的第 7 个寄存器内容	0x0138	RO

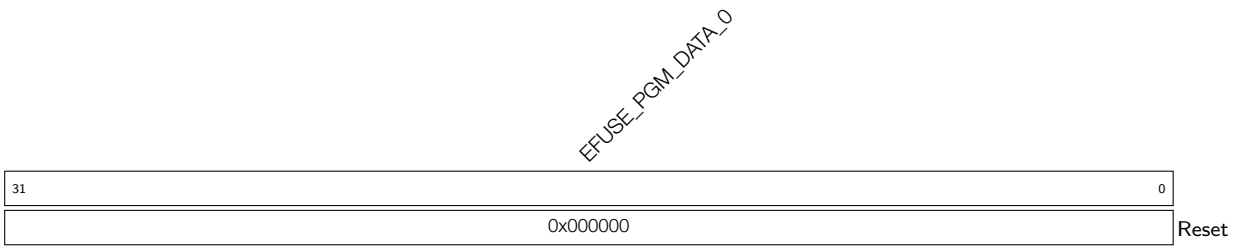
名称	描述	地址	访问
EFUSE_RD_KEY5_DATA0_REG	BLOCK9 (KEY5) 的第 0 个寄存器内容	0x013C	RO
EFUSE_RD_KEY5_DATA1_REG	BLOCK9 (KEY5) 的第 1 个寄存器内容	0x0140	RO
EFUSE_RD_KEY5_DATA2_REG	BLOCK9 (KEY5) 的第 2 个寄存器内容	0x0144	RO
EFUSE_RD_KEY5_DATA3_REG	BLOCK9 (KEY5) 的第 3 个寄存器内容	0x0148	RO
EFUSE_RD_KEY5_DATA4_REG	BLOCK9 (KEY5) 的第 4 个寄存器内容	0x014C	RO
EFUSE_RD_KEY5_DATA5_REG	BLOCK9 (KEY5) 的第 5 个寄存器内容	0x0150	RO
EFUSE_RD_KEY5_DATA6_REG	BLOCK9 (KEY5) 的第 6 个寄存器内容	0x0154	RO
EFUSE_RD_KEY5_DATA7_REG	BLOCK9 (KEY5) 的第 7 个寄存器内容	0x0158	RO
EFUSE_RD_SYS_PART2_DATA0_REG	BLOCK10 (system) 的第 0 个寄存器内容	0x015C	RO
EFUSE_RD_SYS_PART2_DATA1_REG	BLOCK10 (system) 的第 1 个寄存器内容	0x0160	RO
EFUSE_RD_SYS_PART2_DATA2_REG	BLOCK10 (system) 的第 2 个寄存器内容	0x0164	RO
EFUSE_RD_SYS_PART2_DATA3_REG	BLOCK10 (system) 的第 3 个寄存器内容	0x0168	RO
EFUSE_RD_SYS_PART2_DATA4_REG	BLOCK10 (system) 的第 4 个寄存器内容	0x016C	RO
EFUSE_RD_SYS_PART2_DATA5_REG	BLOCK10 (system) 的第 5 个寄存器内容	0x0170	RO
EFUSE_RD_SYS_PART2_DATA6_REG	BLOCK10 (system) 的第 6 个寄存器内容	0x0174	RO
EFUSE_RD_SYS_PART2_DATA7_REG	BLOCK10 (system) 的第 7 个寄存器内容	0x0178	RO
报告寄存器			
EFUSE_RD_REPEAT_ERR0_REG	BLOCK0 参数烧写错误记录第 0 个寄存器	0x017C	RO
EFUSE_RD_REPEAT_ERR1_REG	BLOCK0 参数烧写错误记录第 1 个寄存器	0x0180	RO
EFUSE_RD_REPEAT_ERR2_REG	BLOCK0 参数烧写错误记录第 2 个寄存器	0x0184	RO
EFUSE_RD_REPEAT_ERR3_REG	BLOCK0 参数烧写错误记录第 3 个寄存器	0x0188	RO
EFUSE_RD_REPEAT_ERR4_REG	BLOCK0 参数烧写错误记录第 4 个寄存器	0x0190	RO
EFUSE_RD_RS_ERR0_REG	记录 BLOCK1 ~ 10 参数烧写错误信息的第 0 个寄存器	0x01C0	RO
EFUSE_RD_RS_ERR1_REG	记录 BLOCK1 ~ 10 参数烧写错误信息的第 1 个寄存器	0x01C4	RO
配置寄存器			
EFUSE_CLK_REG	eFuse 时钟配置寄存器	0x01C8	R/W
EFUSE_CONF_REG	eFuse 运行模式配置寄存器	0x01CC	R/W
EFUSE_CMD_REG	eFuse 指令寄存器	0x01D4	varies
EFUSE_RD_TIM_CONF_REG	eFuse 读取时序参数配置寄存器	0x01EC	R/W
EFUSE_WR_TIM_CONF1_REG	eFuse 烧写时序参数第 1 个配置寄存器	0x01F0	R/W
EFUSE_WR_TIM_CONF2_REG	eFuse 烧写时序参数第 2 个配置寄存器	0x01F4	R/W
EFUSE_WR_TIM_CONF0_REG	eFuse 烧写时序参数第 0 个配置寄存器	0x01F8	varies
状态寄存器			
EFUSE_STATUS_REG	eFuse 状态寄存器	0x01D0	RO
中断寄存器			
EFUSE_INT_RAW_REG	eFuse 原始中断寄存器	0x01D8	R/ SS/ WTC
EFUSE_INT_ST_REG	eFuse 中断状态寄存器	0x01DC	RO
EFUSE_INT_ENA_REG	eFuse 中断使能寄存器	0x01E0	R/W
EFUSE_INT_CLR_REG	eFuse 中断清除寄存器	0x01E4	WO
EFUSE_DAC_CONF_REG	eFuse 烧写电压控制寄存器	0x01E8	R/W

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

5.5 寄存器

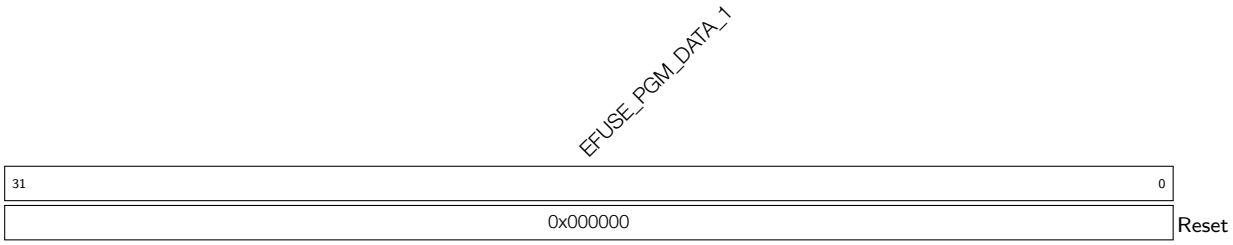
本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 5.1. EFUSE_PGM_DATA0_REG (0x0000)



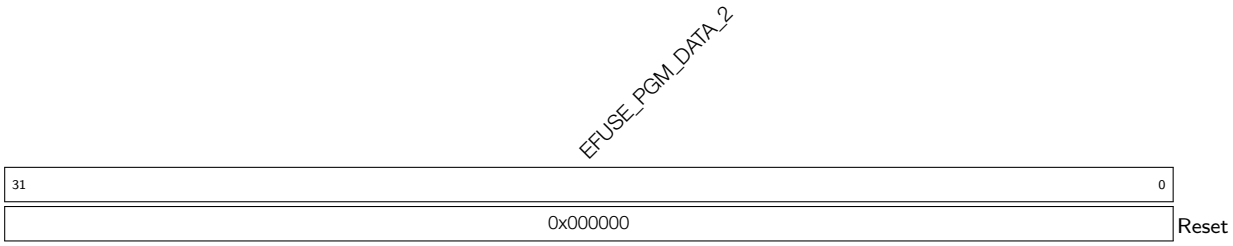
EFUSE_PGM_DATA_0 配置待烧写数据的第 0 个 32 位数据。(R/W)

Register 5.2. EFUSE_PGM_DATA1_REG (0x0004)



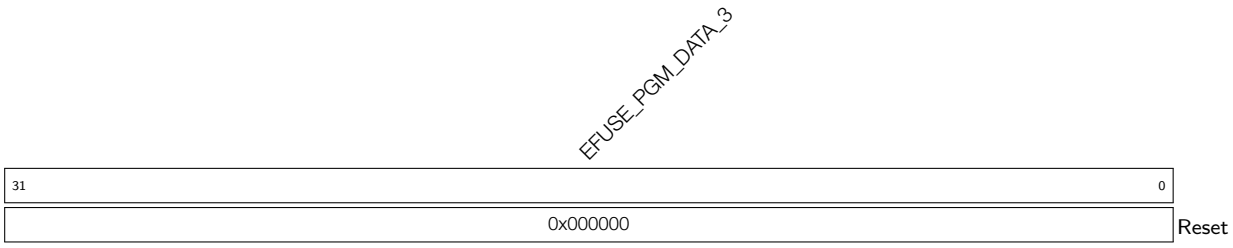
EFUSE_PGM_DATA_1 配置待烧写数据的第 1 个 32 位数据。(R/W)

Register 5.3. EFUSE_PGM_DATA2_REG (0x0008)



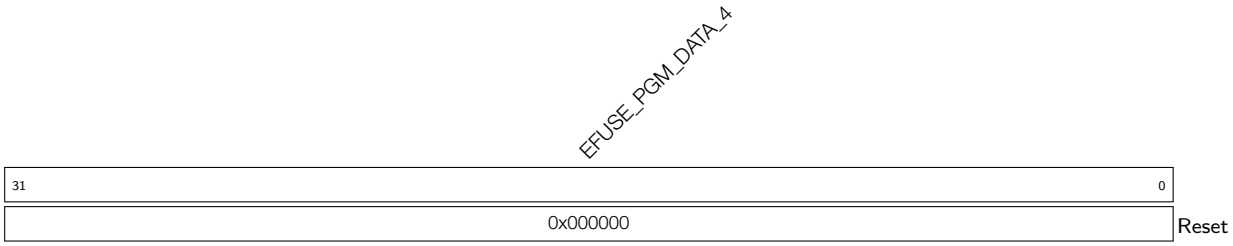
EFUSE_PGM_DATA_2 配置待烧写数据的第 2 个 32 位数据。(R/W)

Register 5.4. EFUSE_PGM_DATA3_REG (0x000C)



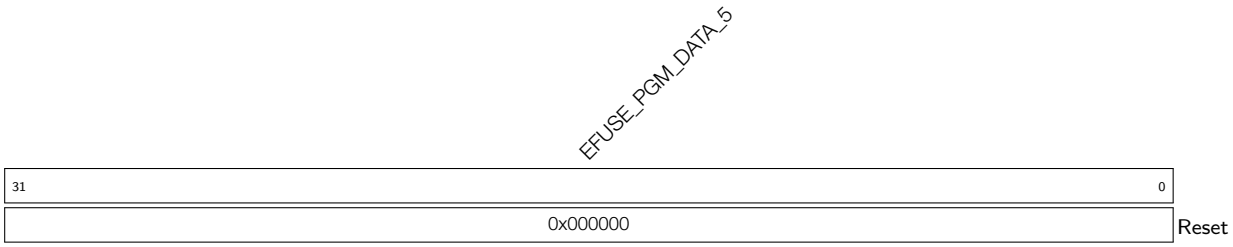
EFUSE_PGM_DATA_3 配置待烧写数据的第 3 个 32 位数据。(R/W)

Register 5.5. EFUSE_PGM_DATA4_REG (0x0010)



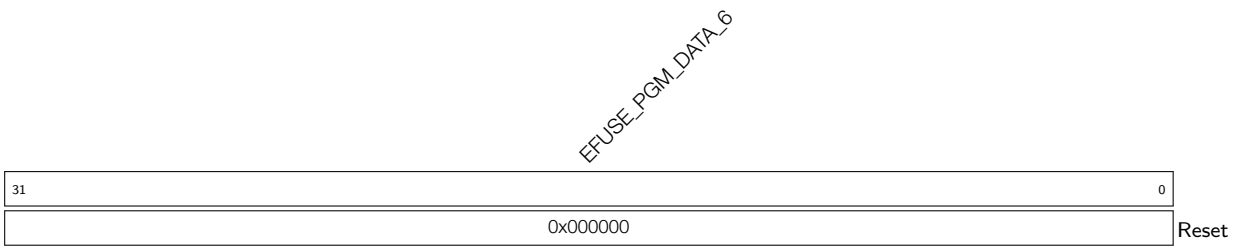
EFUSE_PGM_DATA_4 配置待烧写数据的第 4 个 32 位数据。(R/W)

Register 5.6. EFUSE_PGM_DATA5_REG (0x0014)



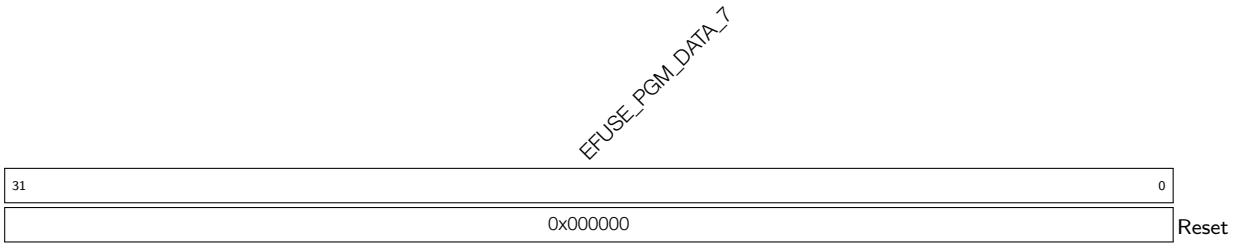
EFUSE_PGM_DATA_5 配置待烧写数据的第 5 个 32 位数据。(R/W)

Register 5.7. EFUSE_PGM_DATA6_REG (0x0018)



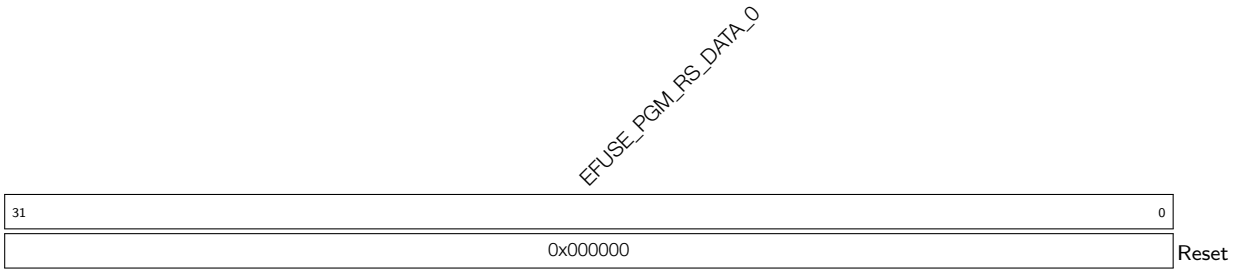
EFUSE_PGM_DATA_6 配置待烧写数据的第 6 个 32 位数据。(R/W)

Register 5.8. EFUSE_PGM_DATA7_REG (0x001C)



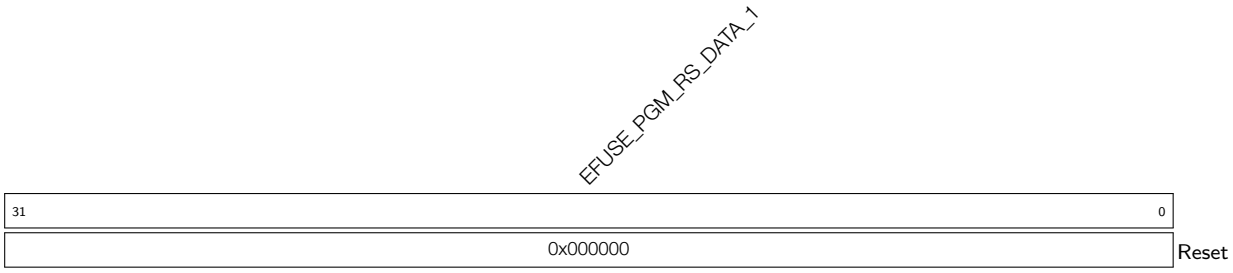
EFUSE_PGM_DATA_7 配置待烧写数据的第 7 个 32 位数据。(R/W)

Register 5.9. EFUSE_PGM_CHECK_VALUE0_REG (0x0020)



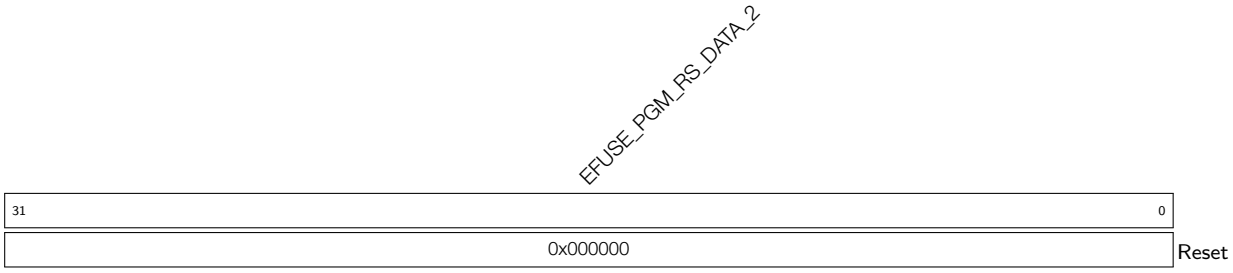
EFUSE_PGM_RS_DATA_0 配置待烧写的第 0 个 32 位 RS 码。(R/W)

Register 5.10. EFUSE_PGM_CHECK_VALUE1_REG (0x0024)



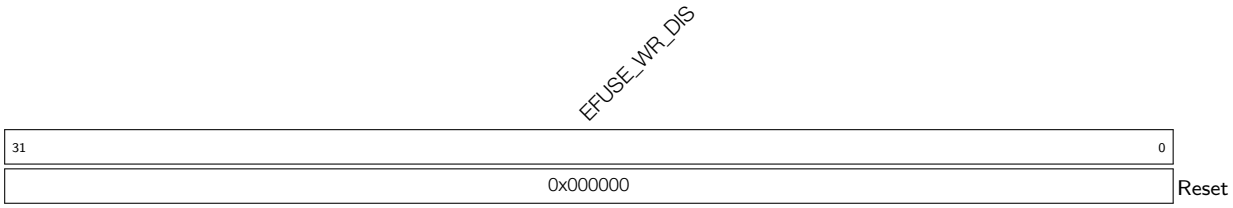
EFUSE_PGM_RS_DATA_1 配置待烧写的第 1 个 32 位 RS 码。(R/W)

Register 5.11. EFUSE_PGM_CHECK_VALUE2_REG (0x0028)



EFUSE_PGM_RS_DATA_2 配置待烧写的第 2 个 32 位 RS 码。(R/W)

Register 5.12. EFUSE_RD_WR_DIS_REG (0x002C)



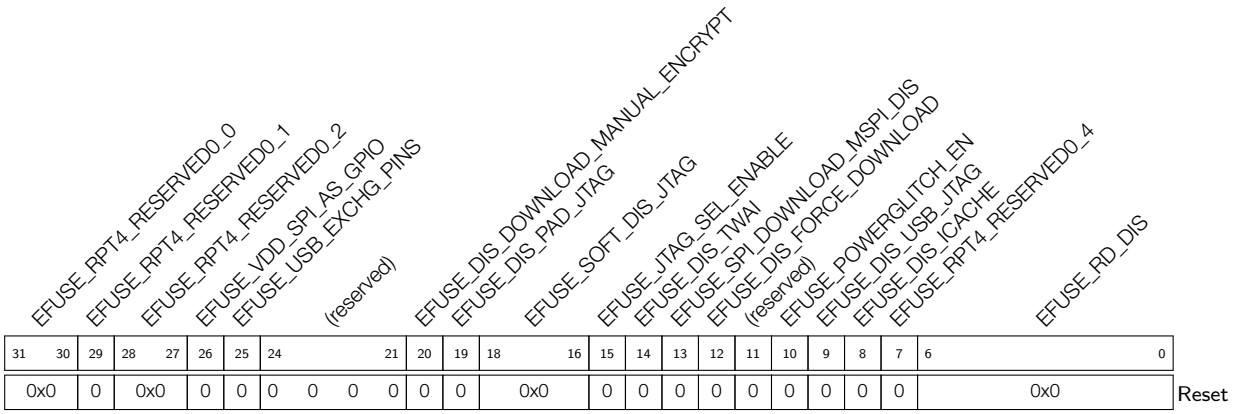
EFUSE_WR_DIS 表示是否使能 eFuse 存储器各个位的烧写。

1：不使能

0：使能

(RO)

Register 5.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS 表示是否使能 eFuse 各个块 (BLOCK4 ~ BLOCK10) 的读取。

1：不使能

0：使能

(RO)

EFUSE_RPT4_RESERVED0_4 保留。(RO)

EFUSE_DIS_ICACHE 表示是否使能指令 Cache。

1：不使能

0：使能

(RO)

EFUSE_DIS_USB_JTAG 表示是否使能 USB 转 JTAG 功能。

1：不使能

0：使能

(RO)

EFUSE_POWERGLITCH_EN 表示是否使能电源毛刺检测功能。

1：使能

0：不使能

(RO)

EFUSE_DIS_FORCE_DOWNLOAD 表示是否使能强制芯片进入 Download 模式。

1：不使能

0：使能

(RO)

见下页……

Register 5.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

接上页……

EFUSE_SPI_DOWNLOAD_MSPI_DIS 表示 boot_mode_download 过程中是否使能 SPI0 控制器。

1：不使能

0：使能

(RO)

EFUSE_DIS_TWAI 表示是否使能 TWAI 功能。

1：不使能

0：使能

(RO)

EFUSE_JTAG_SEL_ENABLE 表示 EFUSE_DIS_PAD_JTAG 和 EFUSE_DIS_USB_JTAG 均配置为 1 时，是否使能通过 GPIO25 的 strapping 值选择 JTAG 信号源。

1：使能

0：不使能

(RO)

EFUSE_SOFT_DIS_JTAG 表示是否软禁用 JTAG。可通过 HMAC 重新启动。

奇数个比特为 1：禁用

偶数个比特为 1：不禁用

(RO)

EFUSE_DIS_PAD_JTAG 表示是否硬禁用 JTAG（永久）。

1：禁用

0：不禁用

(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 表示是否使能 flash 加密（SPI 启动模式除外）。

1：禁用

0：不禁用

(RO)

EFUSE_USB_EXCHG_PINS 表示是否交换 D+ 和 D- 管脚。

1：交换

0：不交换

(RO)

EFUSE_VDD_SPI_AS_GPIO 表示是否将 VDD SPI 管脚用作普通 GPIO。

1：用作普通 GPIO

0：不用作普通 GPIO

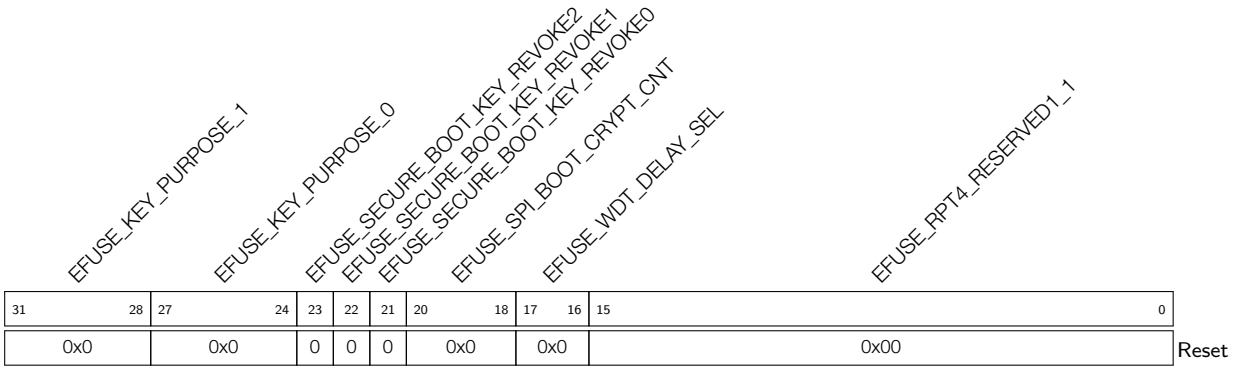
(RO)

EFUSE_RPT4_RESERVED0_2 保留。(RO)

EFUSE_RPT4_RESERVED0_1 保留。(RO)

EFUSE_RPT4_RESERVED0_0 保留。(RO)

Register 5.14. EFUSE_RD_REPEAT_DATA1_REG (0x0034)



EFUSE_RPT4_RESERVED1_1 保留。(RO)

EFUSE_WDT_DELAY_SEL 表示启动时是否选择 RTC 看门狗超时阈值。

1：选择

0：不选择

(RO)

EFUSE_SPI_BOOT_CRYPT_CNT 表示是否使能 SPI boot 加解密。

奇数个比特为 1：使能

偶数个比特为 1：不使能

(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0 表示是否撤销第一个安全启动秘钥。

1：撤销

0：不撤销

(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE1 表示是否撤销第二个安全启动秘钥。

1：撤销

0：不撤销

(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2 表示是否撤销第三个安全启动秘钥。

1：撤销

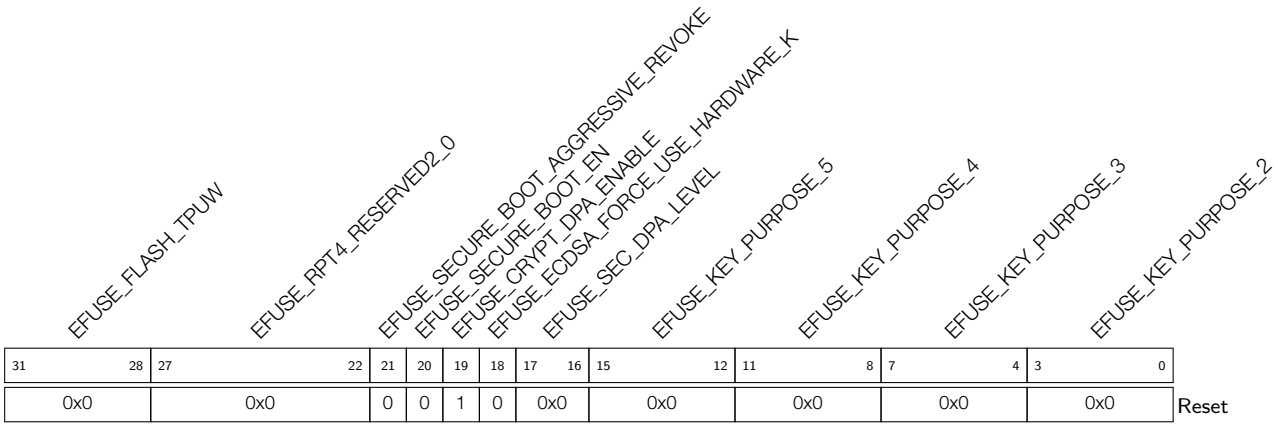
0：不撤销

(RO)

EFUSE_KEY_PURPOSE_0 表示 Key0 用途。(RO)

EFUSE_KEY_PURPOSE_1 表示 Key1 用途。(RO)

Register 5.15. EFUSE_RD_REPEAT_DATA2_REG (0x0038)



EFUSE_KEY_PURPOSE_2 表示 Key2 用途。(RO)

EFUSE_KEY_PURPOSE_3 表示 Key3 用途。(RO)

EFUSE_KEY_PURPOSE_4 表示 Key4 用途。(RO)

EFUSE_KEY_PURPOSE_5 表示 Key5 用途。(RO)

EFUSE_SEC_DPA_LEVEL 表示防 DPA 攻击的安全级别。

0：安全级别为 SEC_DPA_OFF

1/2：安全级别为 SEC_DPA_LOW

3：安全级别为 SEC_DPA_HIGH

有关详细信息，请参阅章节 15 系统寄存器 > 章节 15.2.2。

(RO)

EFUSE_ECDSA_FORCE_USE_HARDWARE_K 表示是否在 ECDSA 模块中强制使用硬件生成的随机值 K。

1：强制使用

0：不强制使用

(RO)

EFUSE_CRYPT_DPA_ENABLE 表示是否使能防 DPA 攻击功能。

1：使能

0：不使能

(RO)

EFUSE_SECURE_BOOT_EN 表示是否使能安全启动。

1：使能

0：不使能

(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE 表示是否使能安全启动的激进撤销。

1：使能

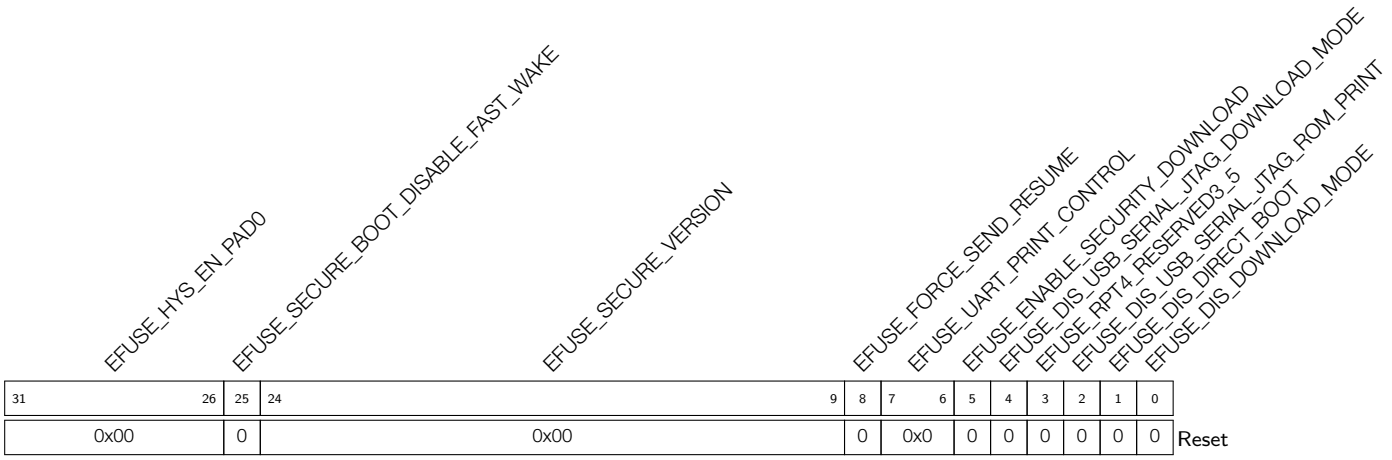
0：不使能

(RO)

EFUSE_RPT4_RESERVED2_0 保留。(RO)

EFUSE_FLASH_TPUW 表示 flash 上电等待时间。单位：ms。当值小于 15 时，等待时间为配置值。当值大于等于 15 时，等待时间固定为 30 ms。(RO)

Register 5.16. EFUSE_RD_REPEAT_DATA3_REG (0x003C)



EFUSE_DIS_DOWNLOAD_MODE 表示是否关闭所有 Download 模式。

1：关闭

0：不关闭

(RO)

EFUSE_DIS_DIRECT_BOOT 表示是否使能 direct boot 模式。

1：不使能

0：使能

(RO)

EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 表示是否使能 ROM boot 过程中的 USB-Serial-JTAG 打印。

1：不使能

0：使能

(RO)

EFUSE_RPT4_RESERVED3_5 保留。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 表示是否使能 USB-Serial-JTAG 下载功能。

1：不使能

0：使能

(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD 表示是否使能安全下载。仅支持 UART 下载，不支持读写 RAM 或寄存器，即不支持 stub 下载。

1：使能

0：不使能

(RO)

见下页……

Register 5.16. EFUSE_RD_REPEAT_DATA3_REG (0x003C)

接上页……

EFUSE_UART_PRINT_CONTROL 表示 UART 打印类型。

0：强制打印

1：GPIO8 低电平复位时使能打印

2：GPIO8 高电平复位时使能打印

3：强制关闭打印

(RO)

EFUSE_FORCE_SEND_RESUME 表示是否强制 ROM 代码在 SPI 启动过程中发送恢复指令。

1：强制

0：不强制

(RO)

EFUSE_SECURE_VERSION 表示安全版本，用于 ESP-IDF 防回滚功能。(RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE 表示安全启动使能时是否使能 FAST VERIFY ON WAKE。

1：不使能

0：使能

(RO)

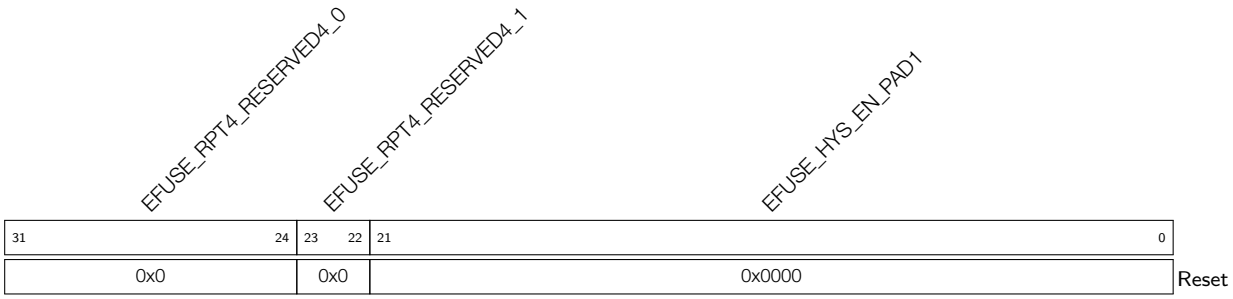
EFUSE_HYS_EN_PAD0 表示是否使能 PAD0-5 的迟滞功能。

1：使能

0：不使能

(RO)

Register 5.17. EFUSE_RD_REPEAT_DATA4_REG (0x0040)



EFUSE_RPT4_RESERVED4_0 保留。(RO)

EFUSE_RPT4_RESERVED4_1 保留。(RO)

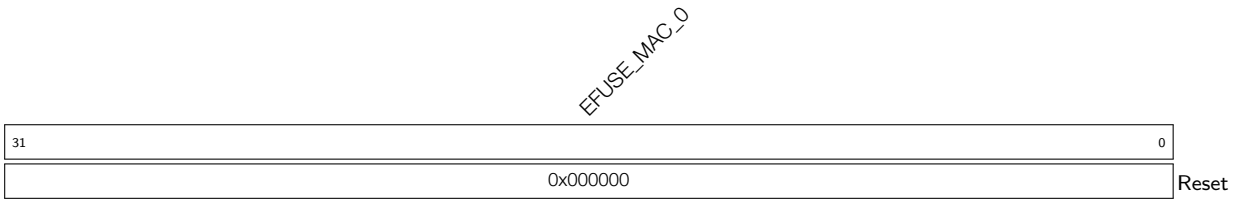
EFUSE_HYS_EN_PAD1 表示是否使能 PAD6-27 的迟滞功能。

1：使能

0：不使能

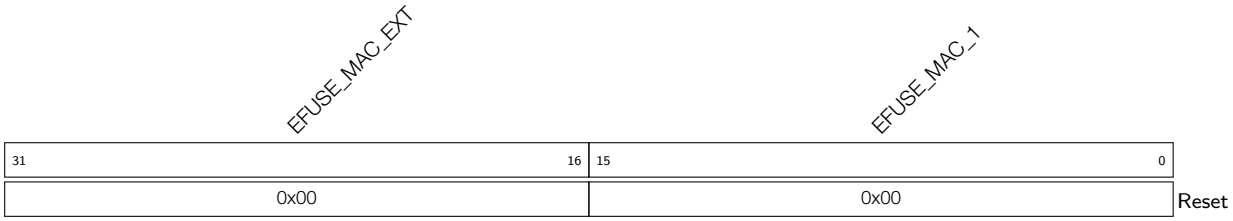
(RO)

Register 5.18. EFUSE_RD_MAC_SYS_0_REG (0x0044)



EFUSE_MAC_0 表示 MAC 地址低 32 位。(RO)

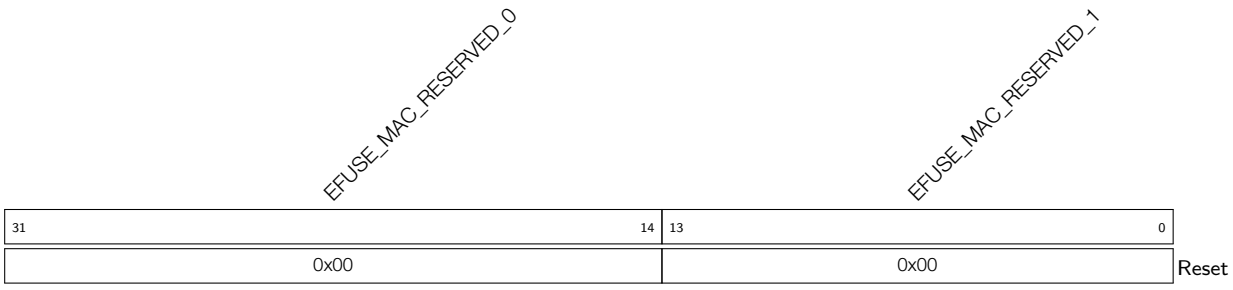
Register 5.19. EFUSE_RD_MAC_SYS_1_REG (0x0048)



EFUSE_MAC_1 表示 MAC 地址高 16 位。(RO)

EFUSE_MAC_EXT 表示 MAC 地址扩展位。(RO)

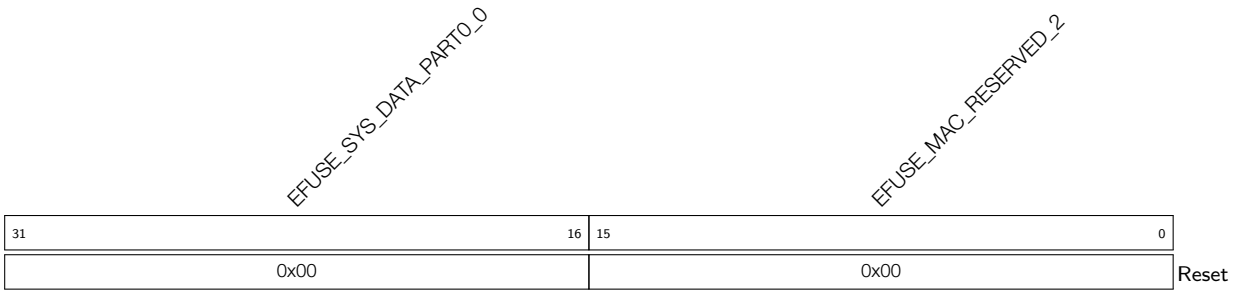
Register 5.20. EFUSE_RD_MAC_SYS_2_REG (0x004C)



EFUSE_MAC_RESERVED_0 保留。(RO)

EFUSE_MAC_RESERVED_1 保留。(RO)

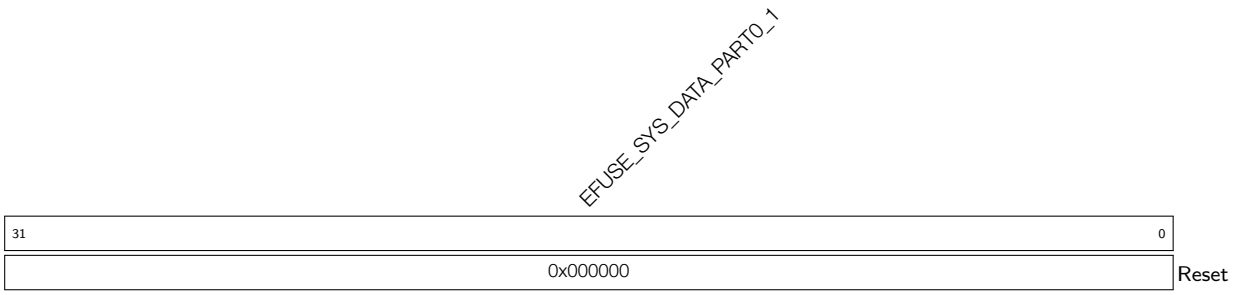
Register 5.21. EFUSE_RD_MAC_SYS_3_REG (0x0050)



EFUSE_MAC_RESERVED_2 保留。(RO)

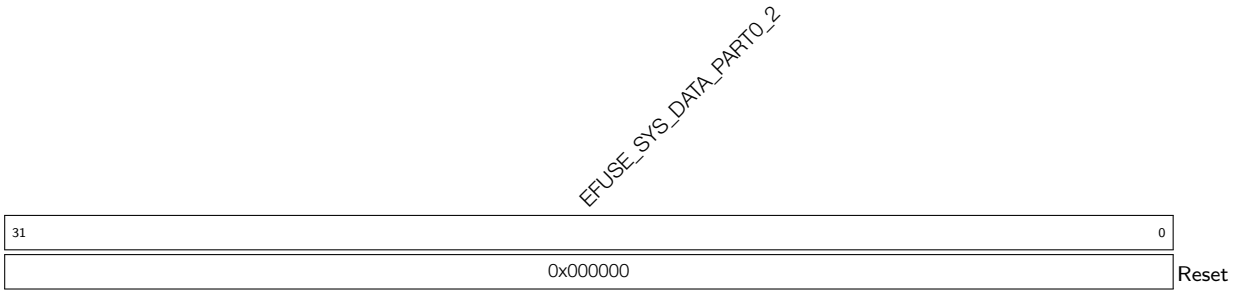
EFUSE_SYS_DATA_PART0_0 表示系统数据第 0 部分的第 1 个 16 位内容。(RO)

Register 5.22. EFUSE_RD_MAC_SYS_4_REG (0x0054)



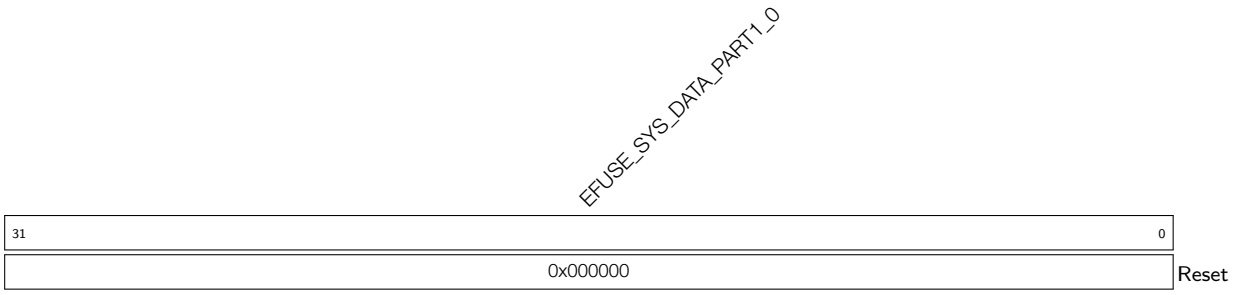
EFUSE_SYS_DATA_PART0_1 表示系统数据第 0 部分的第 1 个 32 位内容。(RO)

Register 5.23. EFUSE_RD_MAC_SYS_5_REG (0x0058)



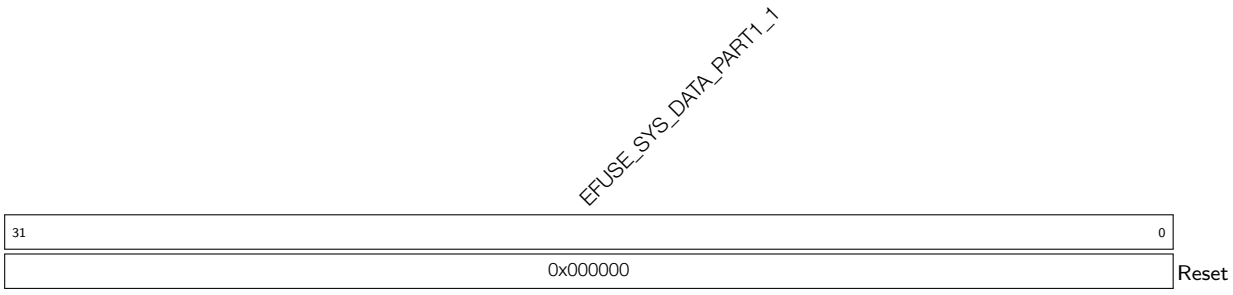
EFUSE_SYS_DATA_PART0_2 表示系统数据第 0 部分的第 2 个 32 位内容。(RO)

Register 5.24. EFUSE_RD_SYS_PART1_DATA0_REG (0x005C)



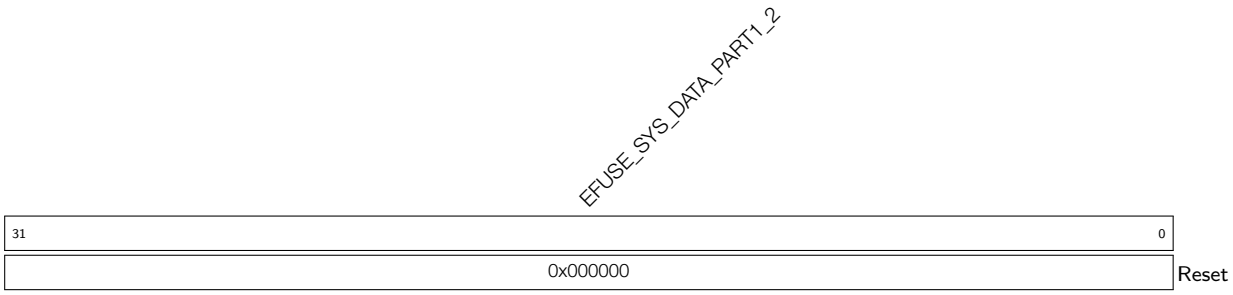
EFUSE_SYS_DATA_PART1_0 表示系统数据第 1 部分的第 0 个 32 位内容。(RO)

Register 5.25. EFUSE_RD_SYS_PART1_DATA1_REG (0x0060)



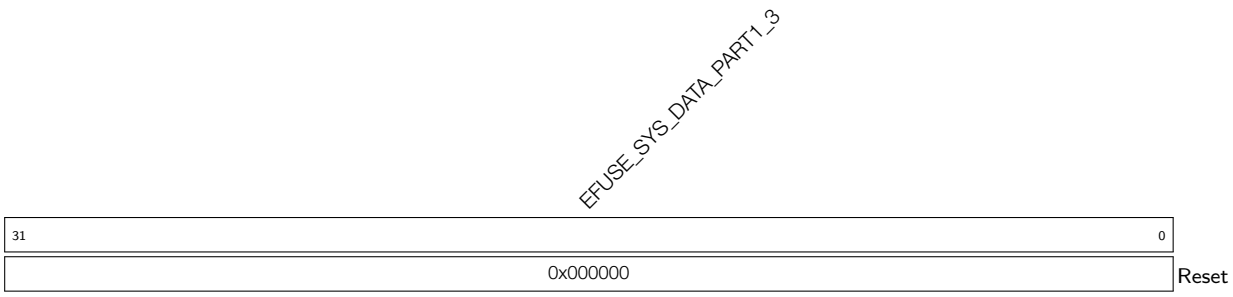
EFUSE_SYS_DATA_PART1_1 表示系统数据第 1 部分的第 1 个 32 位内容。(RO)

Register 5.26. EFUSE_RD_SYS_PART1_DATA2_REG (0x0064)



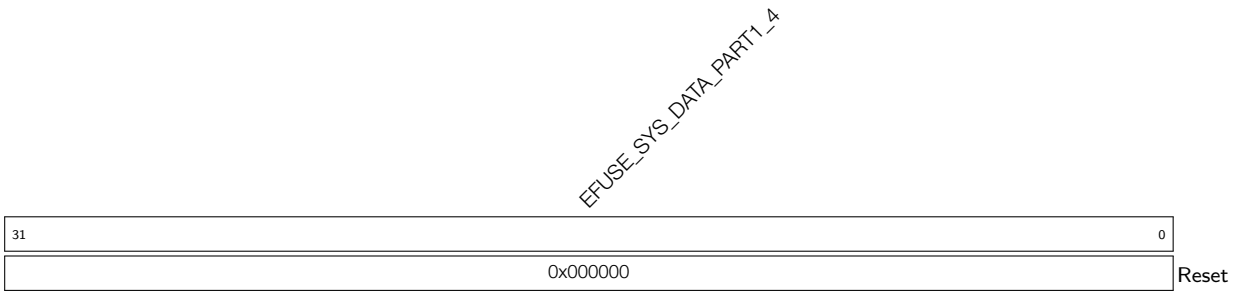
EFUSE_SYS_DATA_PART1_2 表示系统数据第 1 部分的第 2 个 32 位内容。(RO)

Register 5.27. EFUSE_RD_SYS_PART1_DATA3_REG (0x0068)



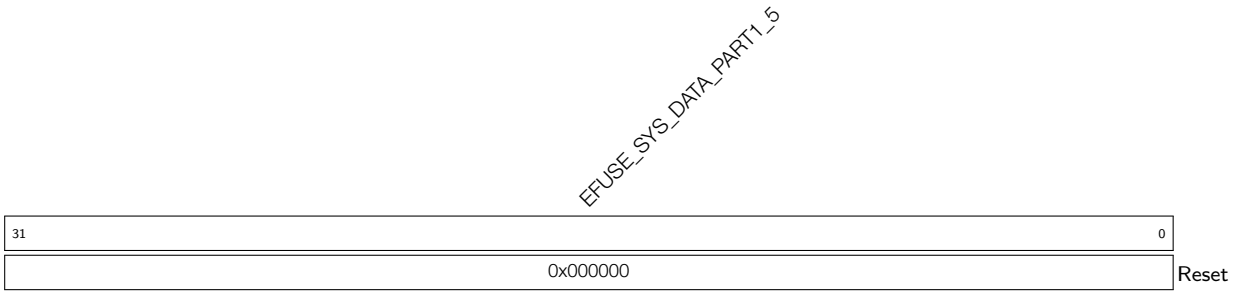
EFUSE_SYS_DATA_PART1_3 表示系统数据第 1 部分的第 3 个 32 位内容。(RO)

Register 5.28. EFUSE_RD_SYS_PART1_DATA4_REG (0x006C)



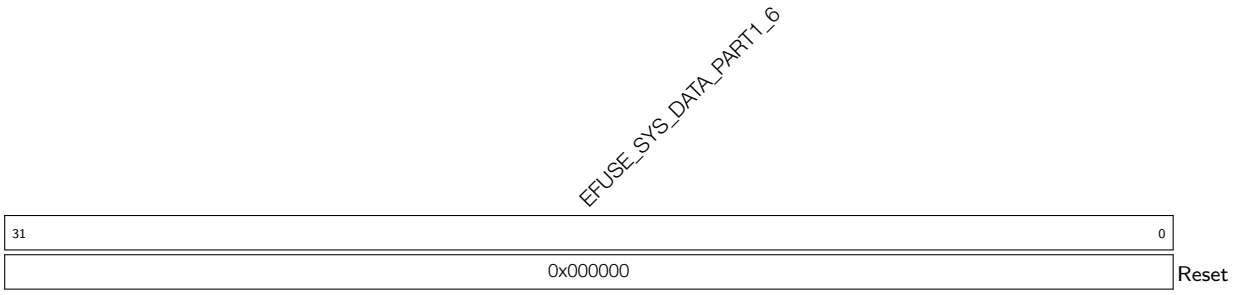
EFUSE_SYS_DATA_PART1_4 表示系统数据第 1 部分的第 4 个 32 位内容。(RO)

Register 5.29. EFUSE_RD_SYS_PART1_DATA5_REG (0x0070)



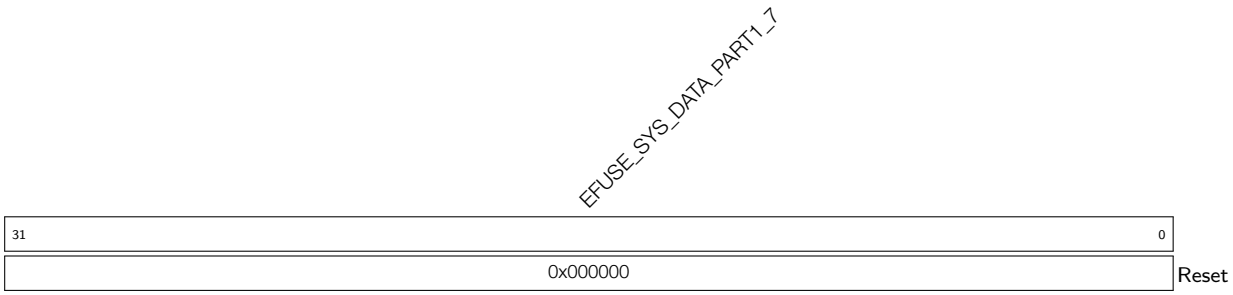
EFUSE_SYS_DATA_PART1_5 表示系统数据第 1 部分的第 5 个 32 位内容。(RO)

Register 5.30. EFUSE_RD_SYS_PART1_DATA6_REG (0x0074)



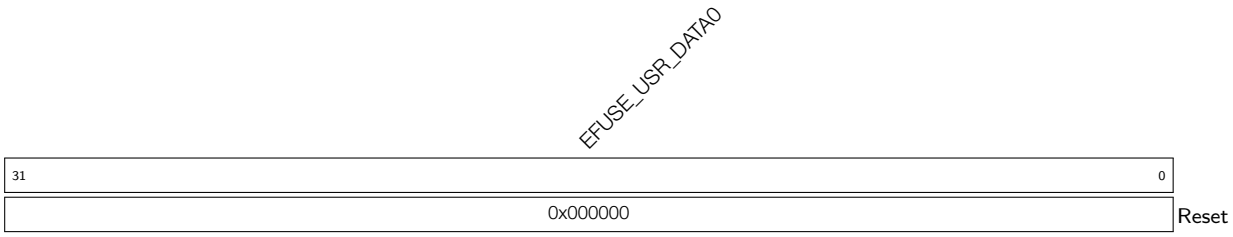
EFUSE_SYS_DATA_PART1_6 表示系统数据第 1 部分的第 6 个 32 位内容。(RO)

Register 5.31. EFUSE_RD_SYS_PART1_DATA7_REG (0x0078)



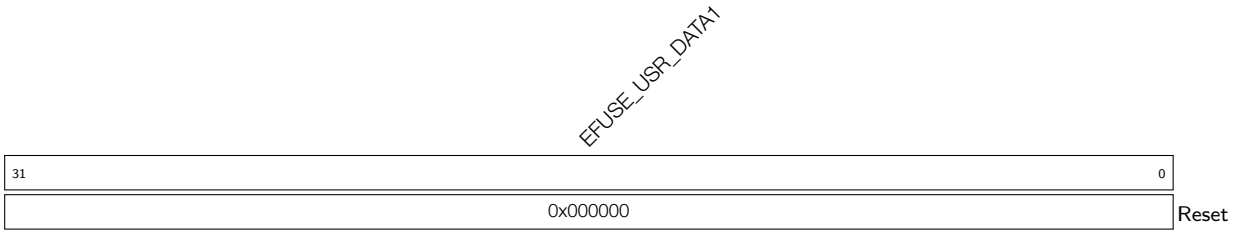
EFUSE_SYS_DATA_PART1_7 表示系统数据第 1 部分的第 7 个 32 位内容。(RO)

Register 5.32. EFUSE_RD_USR_DATA0_REG (0x007C)



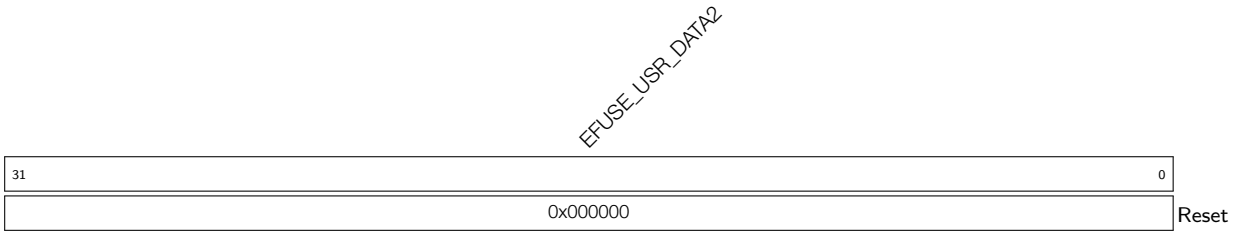
EFUSE_USR_DATA0 表示 BLOCK3 (user) 第 0 个 32 位内容。(RO)

Register 5.33. EFUSE_RD_USR_DATA1_REG (0x0080)



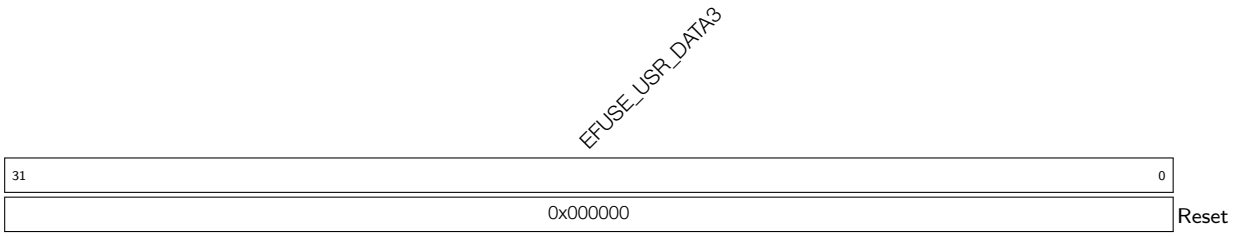
EFUSE_USR_DATA1 表示 BLOCK3 (user) 第 1 个 32 位内容。(RO)

Register 5.34. EFUSE_RD_USR_DATA2_REG (0x0084)



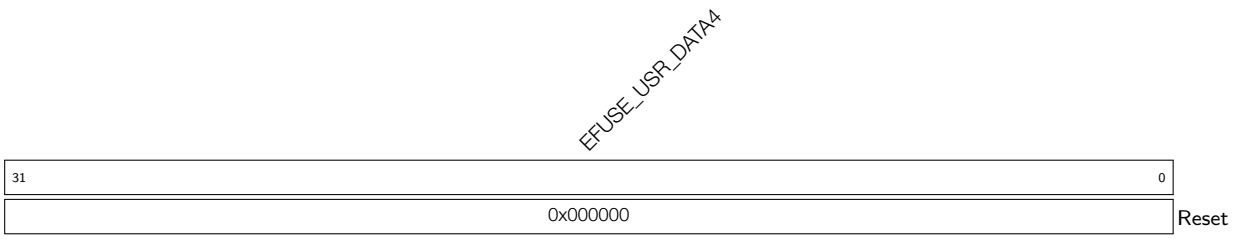
EFUSE_USR_DATA2 表示 BLOCK3 (user) 第 2 个 32 位内容。(RO)

Register 5.35. EFUSE_RD_USR_DATA3_REG (0x0088)



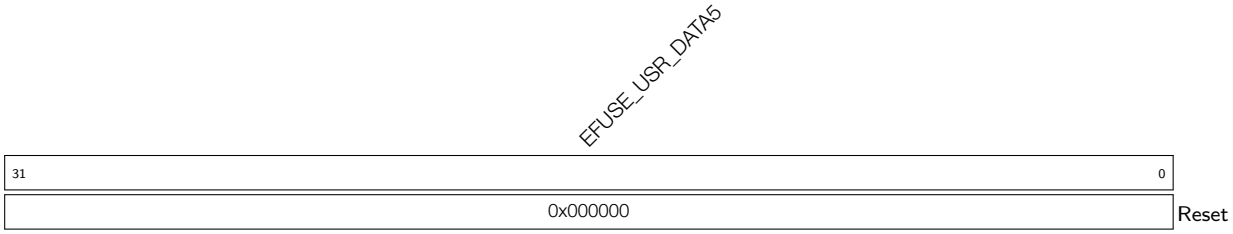
EFUSE_USR_DATA3 表示 BLOCK3 (user) 第 3 个 32 位内容。(RO)

Register 5.36. EFUSE_RD_USR_DATA4_REG (0x008C)



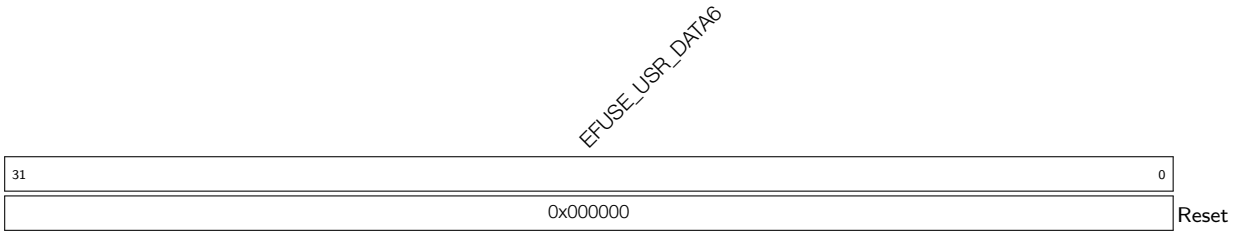
EFUSE_USR_DATA4 表示 BLOCK3 (user) 第 4 个 32 位内容。(RO)

Register 5.37. EFUSE_RD_USR_DATA5_REG (0x0090)



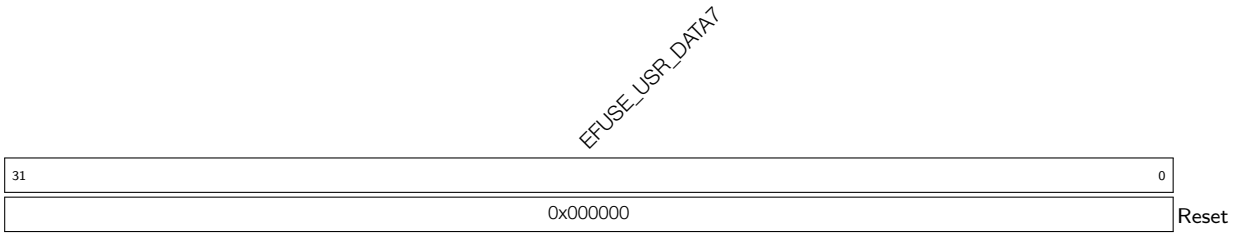
EFUSE_USR_DATA5 表示 BLOCK3 (user) 第 5 个 32 位内容。(RO)

Register 5.38. EFUSE_RD_USR_DATA6_REG (0x0094)



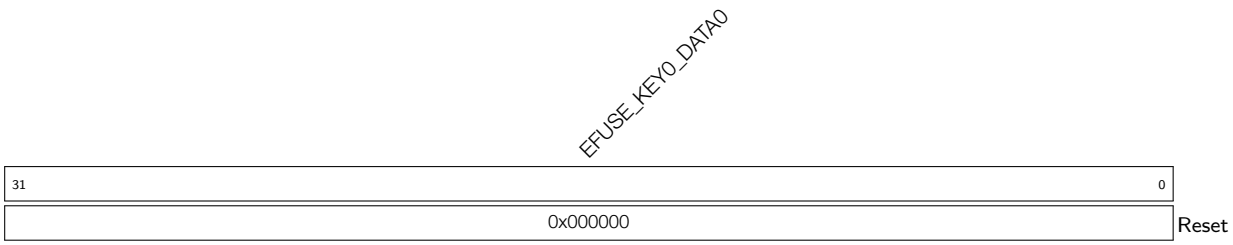
EFUSE_USR_DATA6 表示 BLOCK3 (user) 第 6 个 32 位内容。(RO)

Register 5.39. EFUSE_RD_USR_DATA7_REG (0x0098)



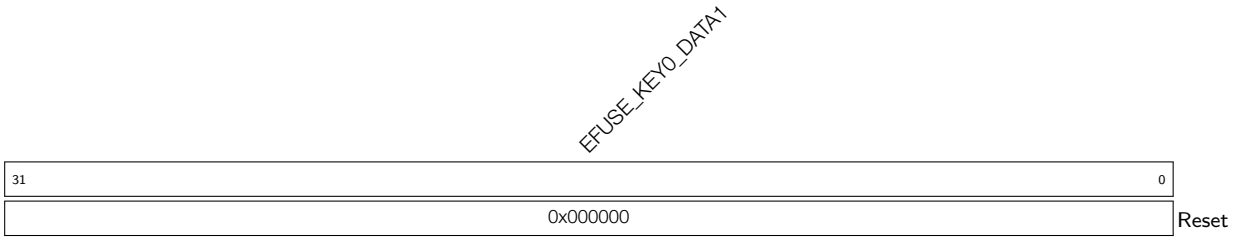
EFUSE_USR_DATA7 表示 BLOCK3 (user) 第 7 个 32 位内容。(RO)

Register 5.40. EFUSE_RD_KEY0_DATA0_REG (0x009C)



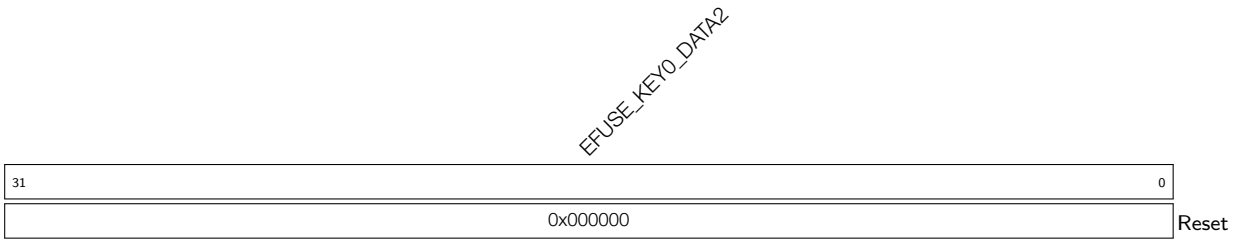
EFUSE_KEY0_DATA0 表示 KEY0 第 0 个 32 位内容。(RO)

Register 5.41. EFUSE_RD_KEY0_DATA1_REG (0x00A0)



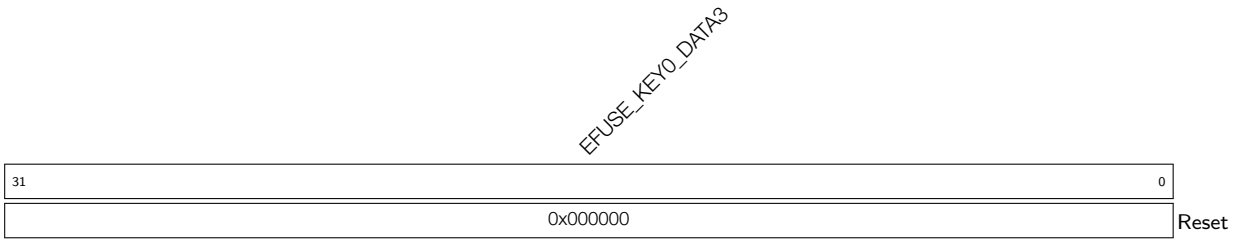
EFUSE_KEY0_DATA1 表示 KEY0 第 1 个 32 位内容。(RO)

Register 5.42. EFUSE_RD_KEY0_DATA2_REG (0x00A4)



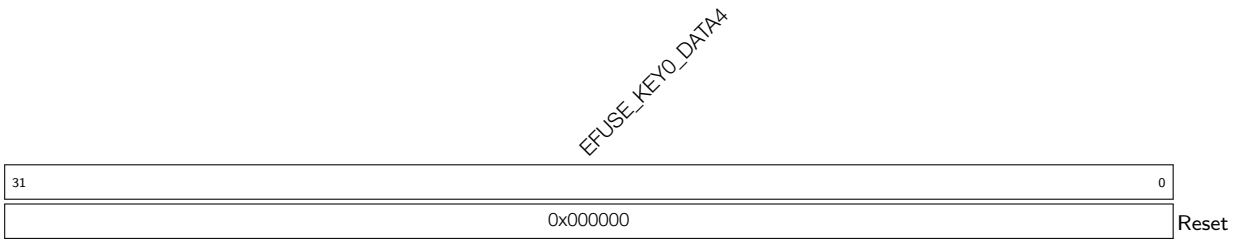
EFUSE_KEY0_DATA2 表示 KEY0 第 2 个 32 位内容。(RO)

Register 5.43. EFUSE_RD_KEY0_DATA3_REG (0x00A8)



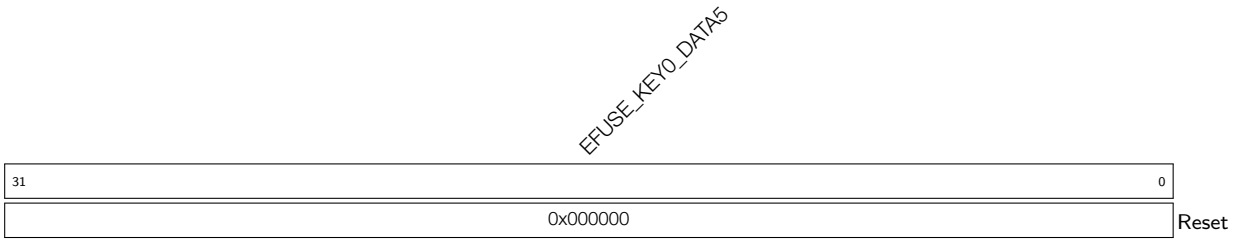
EFUSE_KEY0_DATA3 表示 KEY0 第 3 个 32 位内容。(RO)

Register 5.44. EFUSE_RD_KEY0_DATA4_REG (0x00AC)



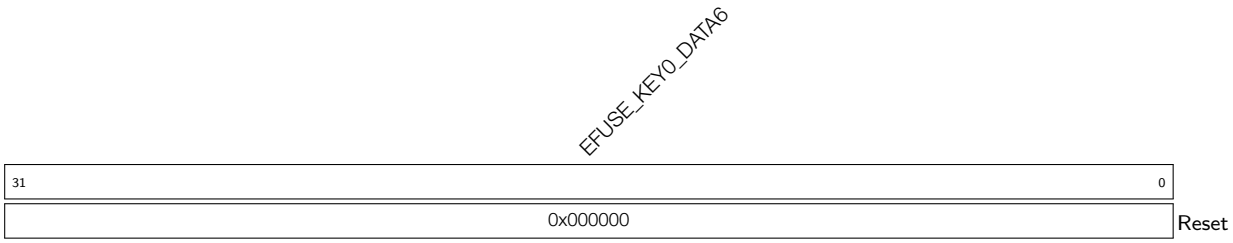
EFUSE_KEY0_DATA4 表示 KEY0 第 4 个 32 位内容。(RO)

Register 5.45. EFUSE_RD_KEY0_DATA5_REG (0x00B0)



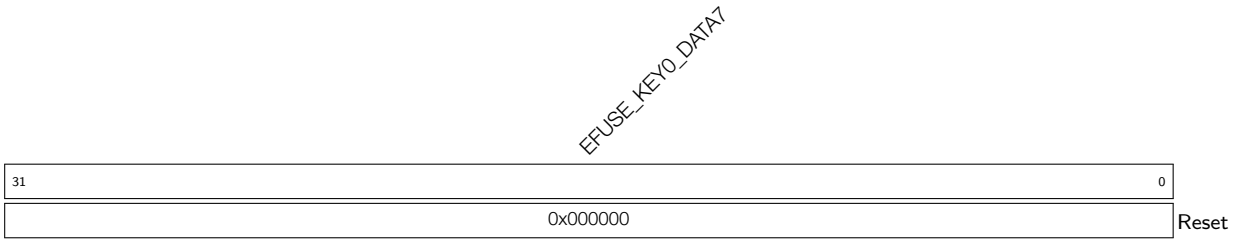
EFUSE_KEY0_DATA5 表示 KEY0 第 5 个 32 位内容。(RO)

Register 5.46. EFUSE_RD_KEY0_DATA6_REG (0x00B4)



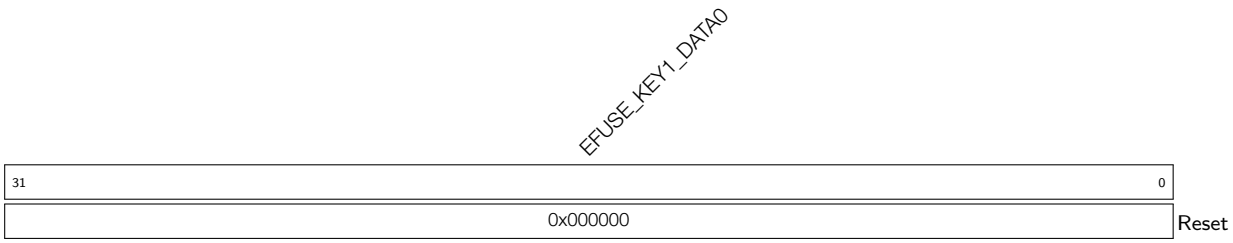
EFUSE_KEY0_DATA6 表示 KEY0 第 6 个 32 位内容。(RO)

Register 5.47. EFUSE_RD_KEY0_DATA7_REG (0x00B8)



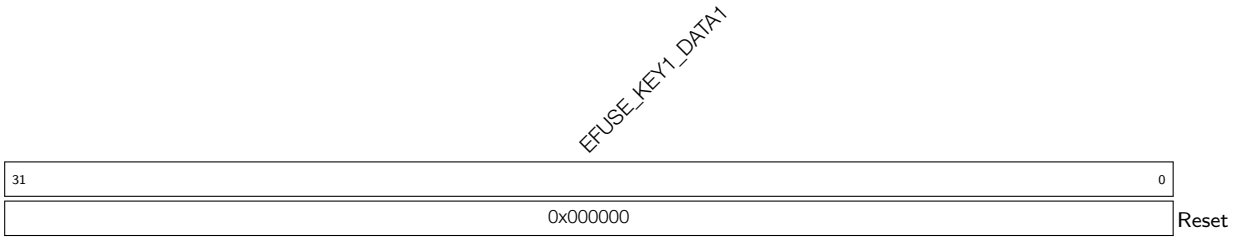
EFUSE_KEY0_DATA7 表示 KEY0 第 7 个 32 位内容。(RO)

Register 5.48. EFUSE_RD_KEY1_DATA0_REG (0x00BC)



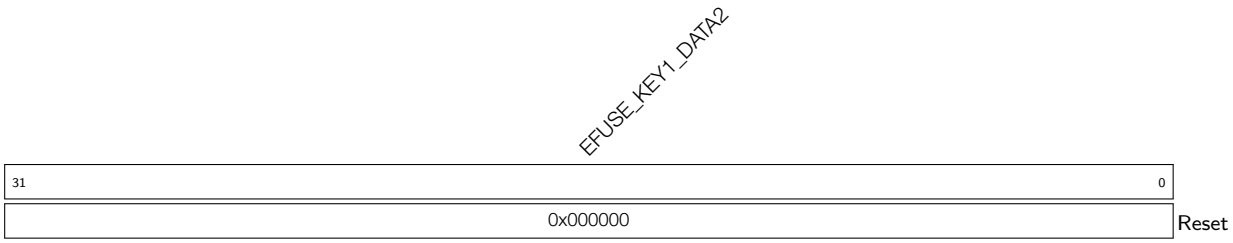
EFUSE_KEY1_DATA0 表示 KEY1 第 0 个 32 位内容。(RO)

Register 5.49. EFUSE_RD_KEY1_DATA1_REG (0x00C0)



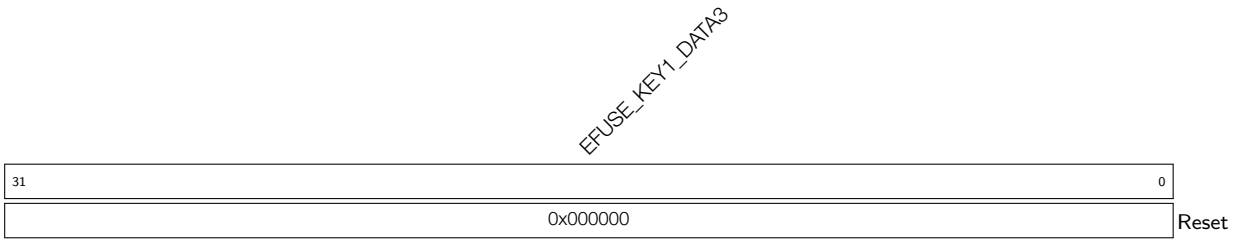
EFUSE_KEY1_DATA1 表示 KEY1 第 1 个 32 位内容。(RO)

Register 5.50. EFUSE_RD_KEY1_DATA2_REG (0x00C4)



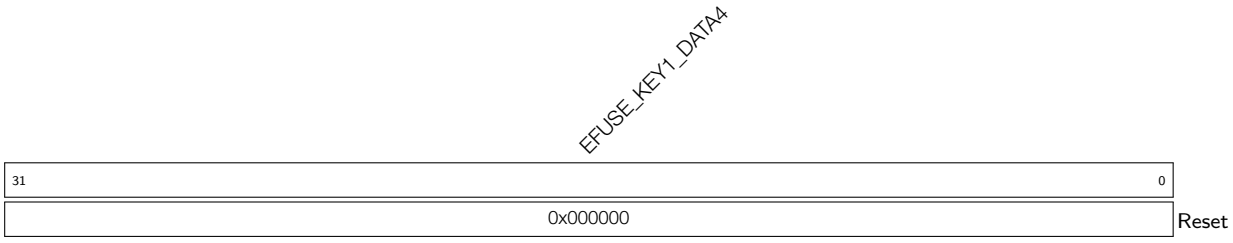
EFUSE_KEY1_DATA2 表示 KEY1 第 2 个 32 位内容。(RO)

Register 5.51. EFUSE_RD_KEY1_DATA3_REG (0x00C8)



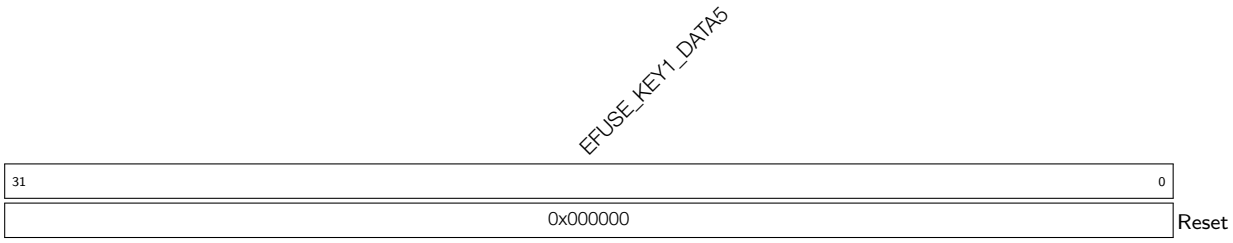
EFUSE_KEY1_DATA3 表示 KEY1 第 3 个 32 位内容。(RO)

Register 5.52. EFUSE_RD_KEY1_DATA4_REG (0x00CC)



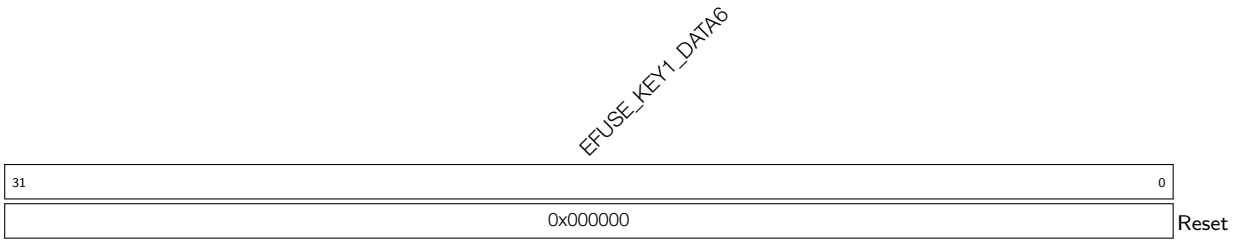
EFUSE_KEY1_DATA4 表示 KEY1 第 4 个 32 位内容。(RO)

Register 5.53. EFUSE_RD_KEY1_DATA5_REG (0x00D0)



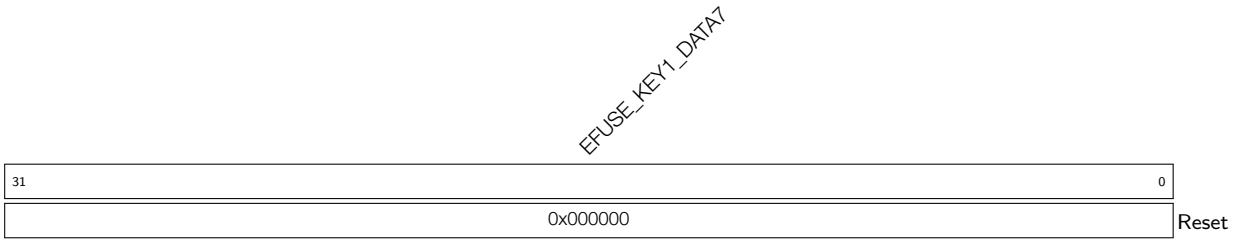
EFUSE_KEY1_DATA5 表示 KEY1 第 5 个 32 位内容。(RO)

Register 5.54. EFUSE_RD_KEY1_DATA6_REG (0x00D4)



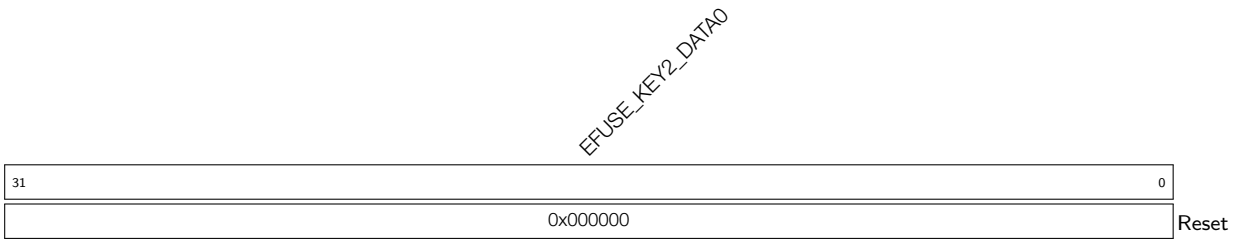
EFUSE_KEY1_DATA6 表示 KEY1 第 6 个 32 位内容。(RO)

Register 5.55. EFUSE_RD_KEY1_DATA7_REG (0x00D8)



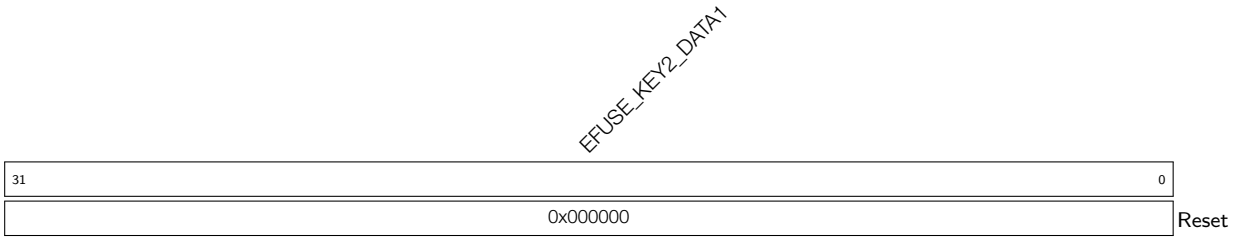
EFUSE_KEY1_DATA7 表示 KEY1 第 7 个 32 位内容。(RO)

Register 5.56. EFUSE_RD_KEY2_DATA0_REG (0x00DC)



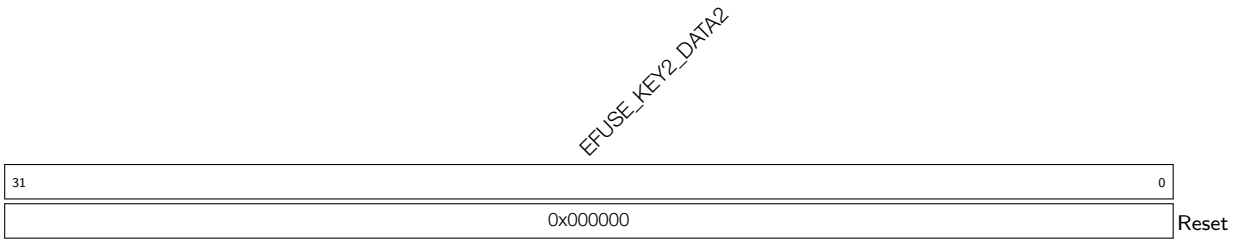
EFUSE_KEY2_DATA0 表示 KEY2 第 0 个 32 位内容。(RO)

Register 5.57. EFUSE_RD_KEY2_DATA1_REG (0x00E0)



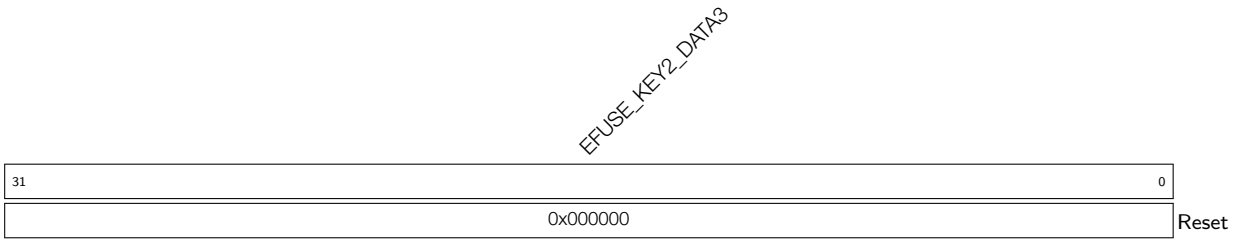
EFUSE_KEY2_DATA1 表示 KEY2 第 1 个 32 位内容。(RO)

Register 5.58. EFUSE_RD_KEY2_DATA2_REG (0x00E4)



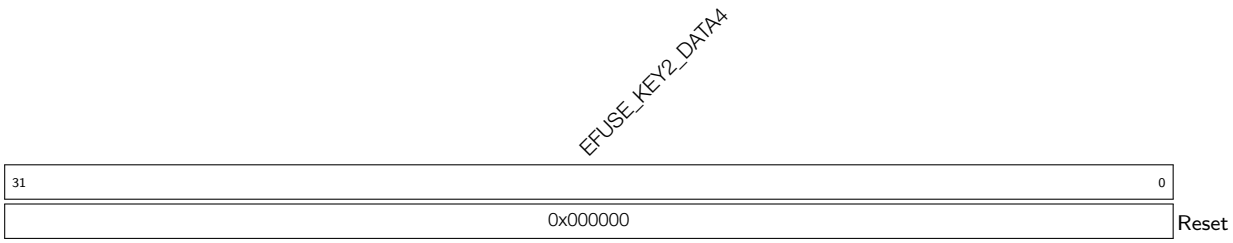
EFUSE_KEY2_DATA2 表示 KEY2 第 2 个 32 位内容。(RO)

Register 5.59. EFUSE_RD_KEY2_DATA3_REG (0x00E8)



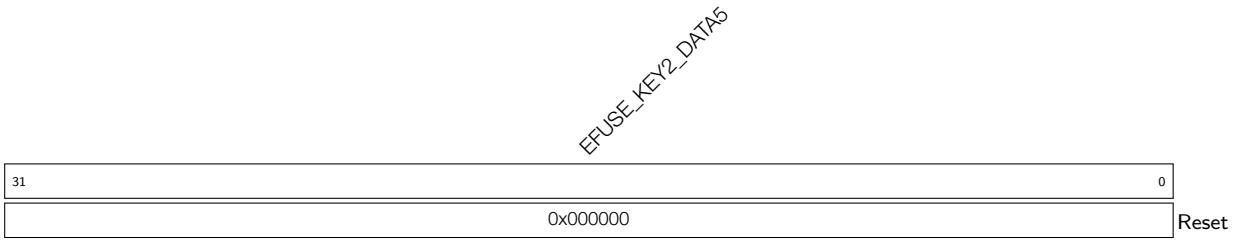
EFUSE_KEY2_DATA3 表示 KEY2 第 3 个 32 位内容。(RO)

Register 5.60. EFUSE_RD_KEY2_DATA4_REG (0x00EC)



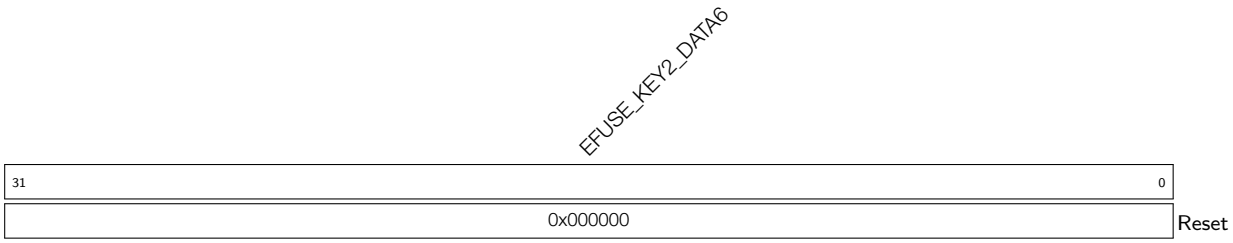
EFUSE_KEY2_DATA4 表示 KEY2 第 4 个 32 位内容。(RO)

Register 5.61. EFUSE_RD_KEY2_DATA5_REG (0x00F0)



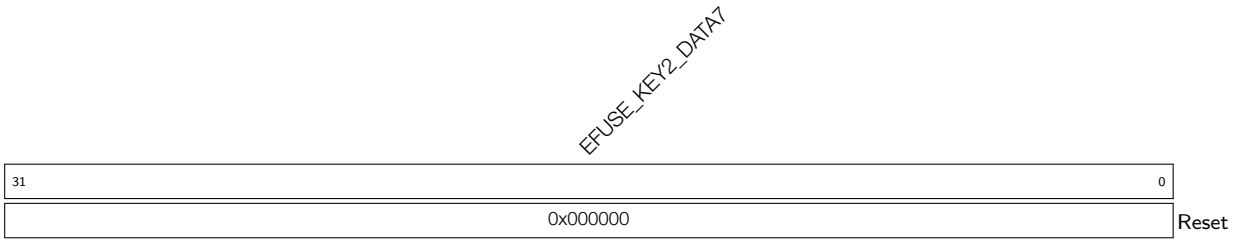
EFUSE_KEY2_DATA5 表示 KEY2 第 5 个 32 位内容。(RO)

Register 5.62. EFUSE_RD_KEY2_DATA6_REG (0x00F4)



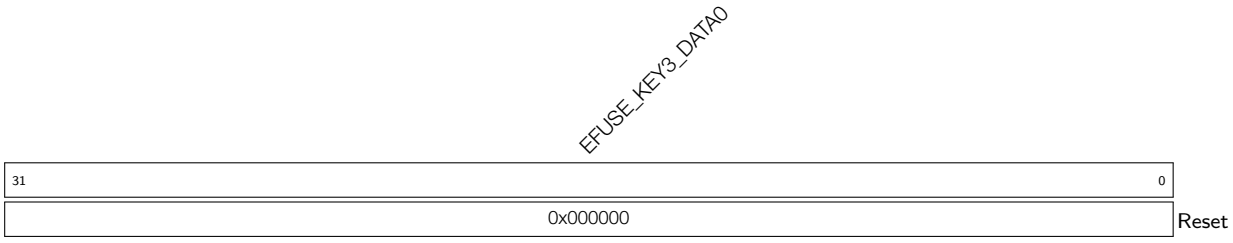
EFUSE_KEY2_DATA6 表示 KEY2 第 6 个 32 位内容。(RO)

Register 5.63. EFUSE_RD_KEY2_DATA7_REG (0x00F8)



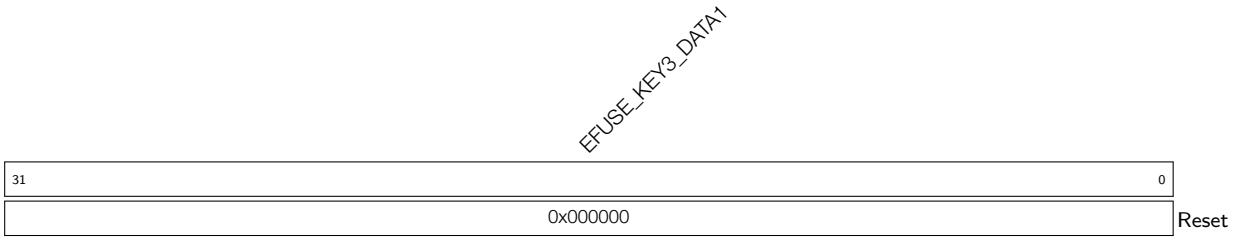
EFUSE_KEY2_DATA7 表示 KEY2 第 7 个 32 位内容。(RO)

Register 5.64. EFUSE_RD_KEY3_DATA0_REG (0x00FC)



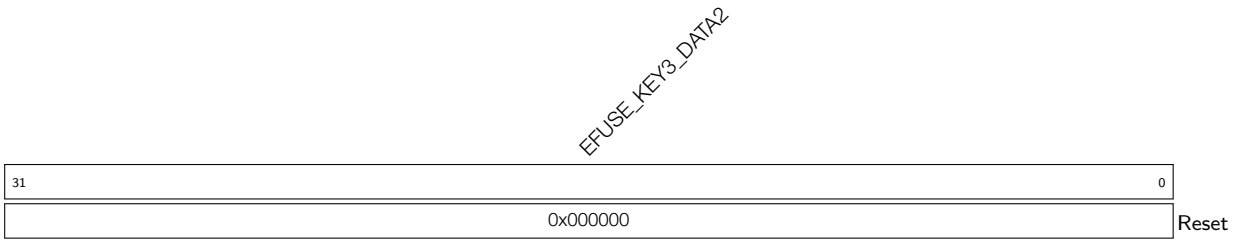
EFUSE_KEY3_DATA0 表示 KEY3 第 0 个 32 位内容。(RO)

Register 5.65. EFUSE_RD_KEY3_DATA1_REG (0x0100)



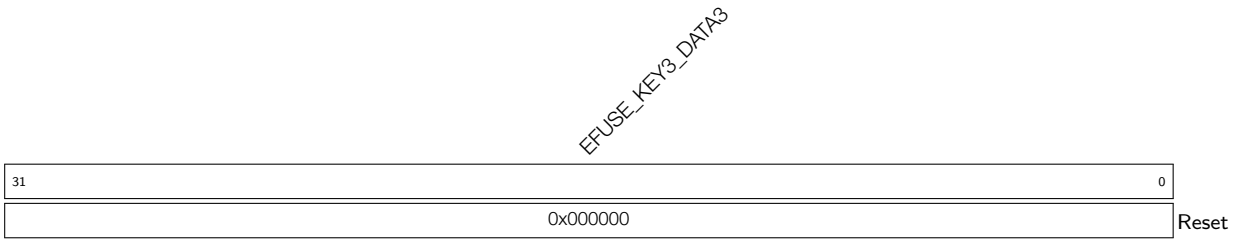
EFUSE_KEY3_DATA1 表示 KEY3 第 1 个 32 位内容。(RO)

Register 5.66. EFUSE_RD_KEY3_DATA2_REG (0x0104)



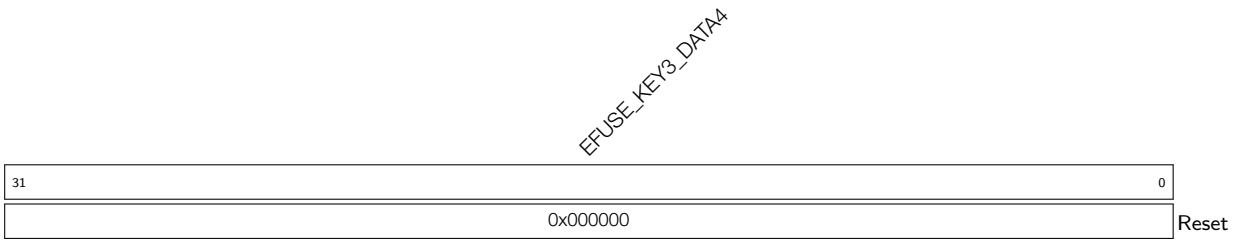
EFUSE_KEY3_DATA2 表示 KEY3 第 2 个 32 位内容。(RO)

Register 5.67. EFUSE_RD_KEY3_DATA3_REG (0x0108)



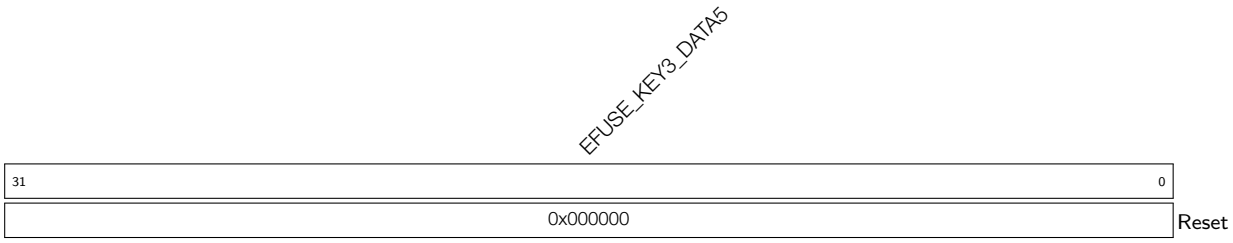
EFUSE_KEY3_DATA3 表示 KEY3 第 3 个 32 位内容。(RO)

Register 5.68. EFUSE_RD_KEY3_DATA4_REG (0x010C)



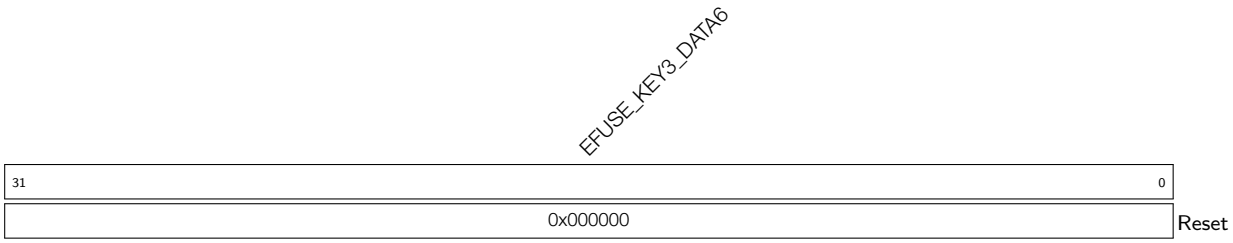
EFUSE_KEY3_DATA4 表示 KEY3 第 4 个 32 位内容。(RO)

Register 5.69. EFUSE_RD_KEY3_DATA5_REG (0x0110)



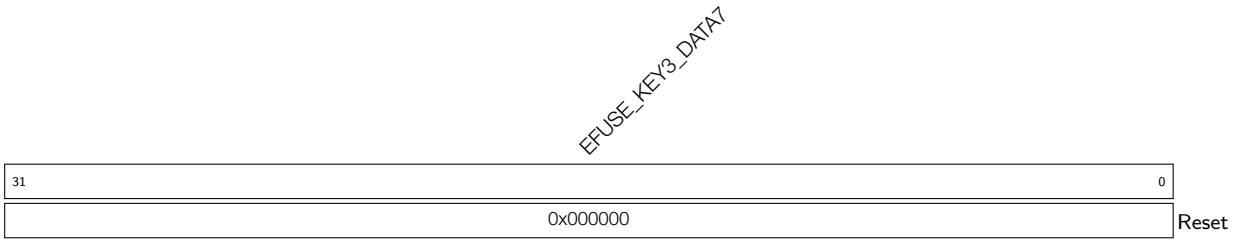
EFUSE_KEY3_DATA5 表示 KEY3 第 5 个 32 位内容。(RO)

Register 5.70. EFUSE_RD_KEY3_DATA6_REG (0x0114)



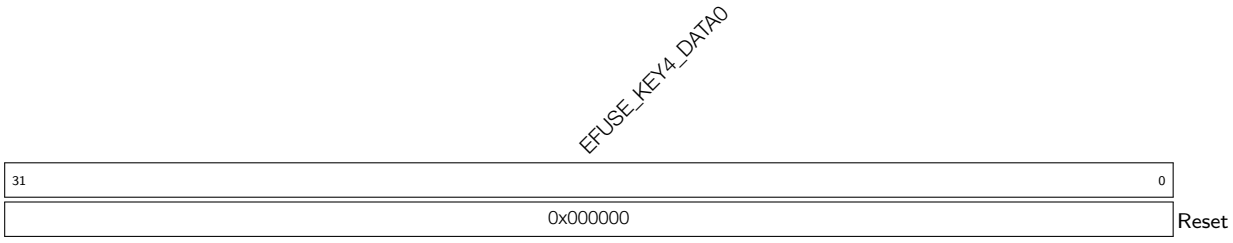
EFUSE_KEY3_DATA6 表示 KEY3 第 6 个 32 位内容。(RO)

Register 5.71. EFUSE_RD_KEY3_DATA7_REG (0x0118)



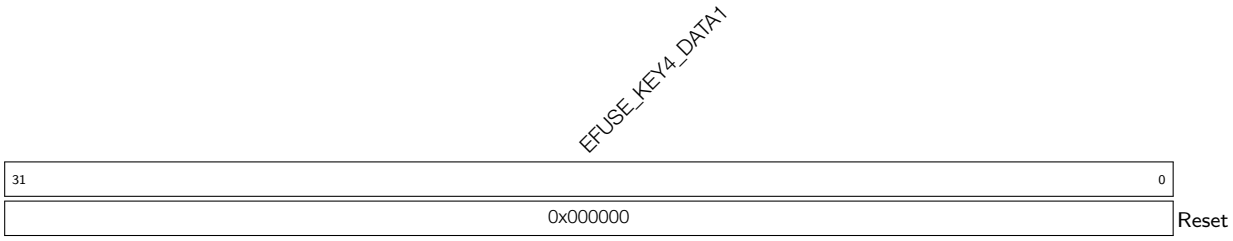
EFUSE_KEY3_DATA7 表示 KEY3 第 7 个 32 位内容。(RO)

Register 5.72. EFUSE_RD_KEY4_DATA0_REG (0x011C)



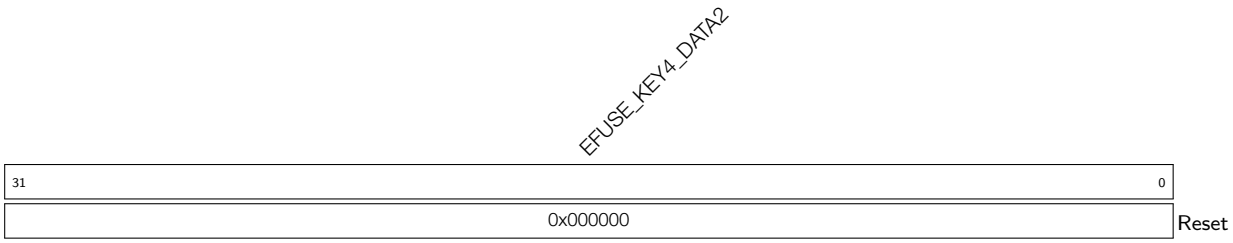
EFUSE_KEY4_DATA0 表示 KEY4 第 0 个 32 位内容。(RO)

Register 5.73. EFUSE_RD_KEY4_DATA1_REG (0x0120)



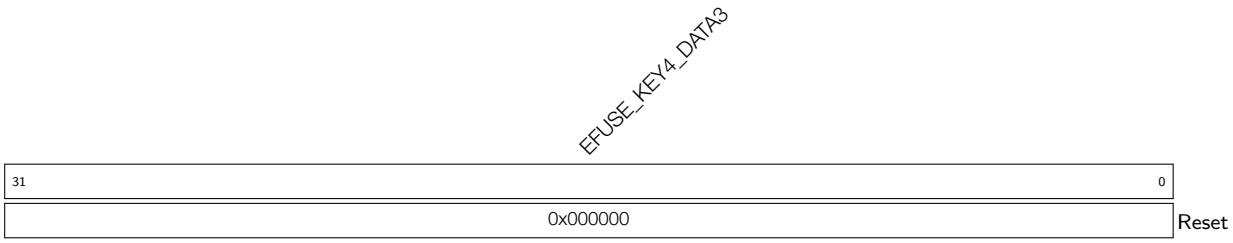
EFUSE_KEY4_DATA1 表示 KEY4 第 1 个 32 位内容。(RO)

Register 5.74. EFUSE_RD_KEY4_DATA2_REG (0x0124)



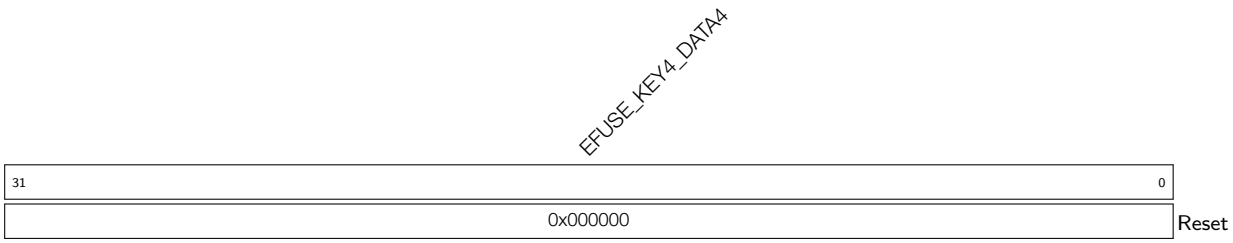
EFUSE_KEY4_DATA2 表示 KEY4 第 2 个 32 位内容。(RO)

Register 5.75. EFUSE_RD_KEY4_DATA3_REG (0x0128)



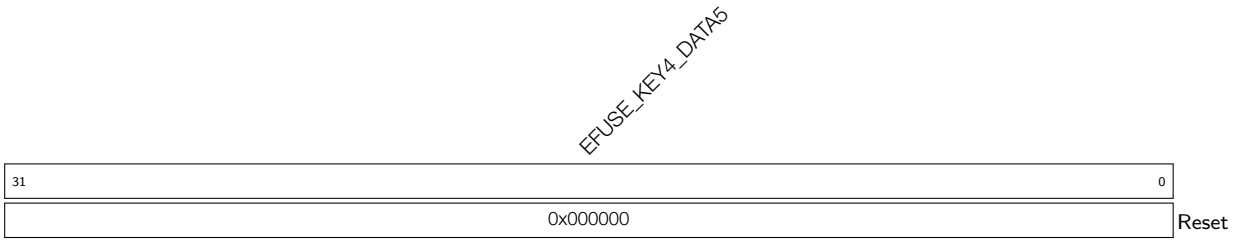
EFUSE_KEY4_DATA3 表示 KEY4 第 3 个 32 位内容。(RO)

Register 5.76. EFUSE_RD_KEY4_DATA4_REG (0x012C)



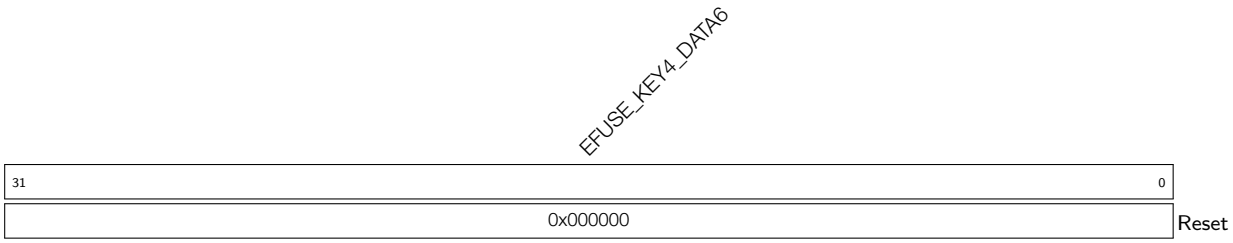
EFUSE_KEY4_DATA4 表示 KEY4 第 4 个 32 位内容。(RO)

Register 5.77. EFUSE_RD_KEY4_DATA5_REG (0x0130)



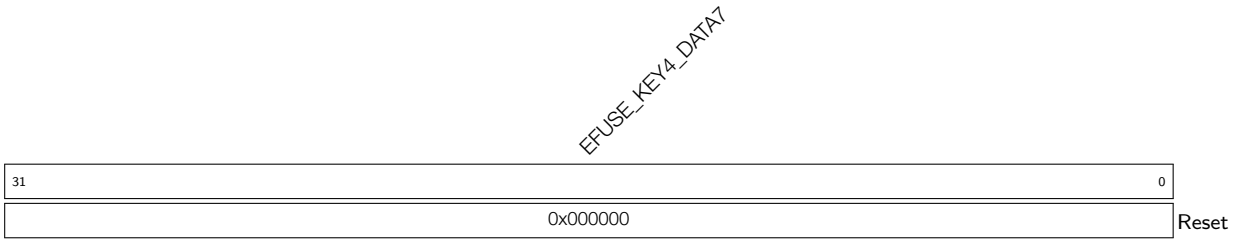
EFUSE_KEY4_DATA5 表示 KEY4 第 5 个 32 位内容。(RO)

Register 5.78. EFUSE_RD_KEY4_DATA6_REG (0x0134)



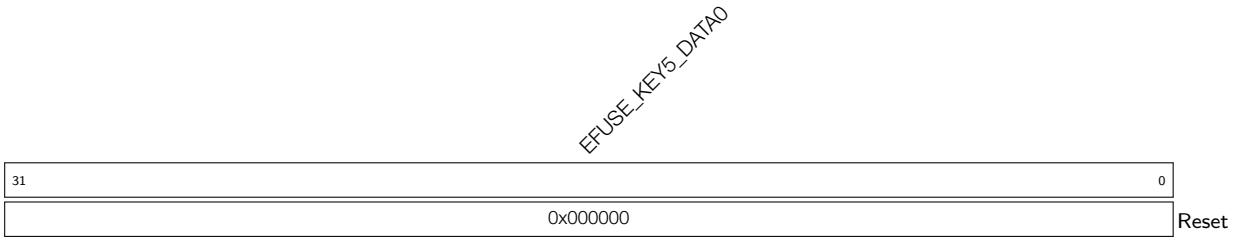
EFUSE_KEY4_DATA6 表示 KEY4 第 6 个 32 位内容。(RO)

Register 5.79. EFUSE_RD_KEY4_DATA7_REG (0x0138)



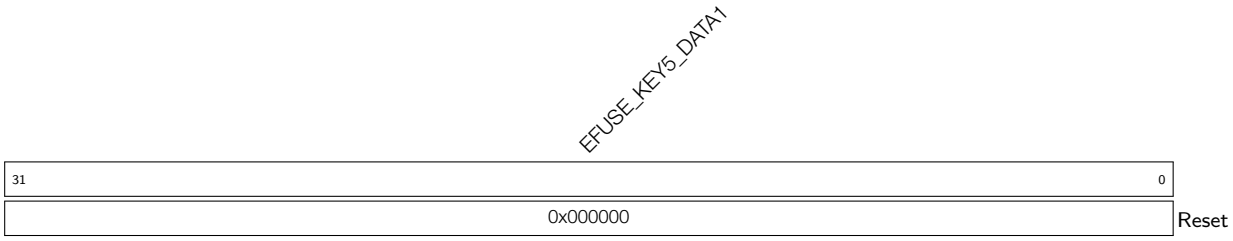
EFUSE_KEY4_DATA7 表示 KEY4 第 7 个 32 位内容。(RO)

Register 5.80. EFUSE_RD_KEY5_DATA0_REG (0x013C)



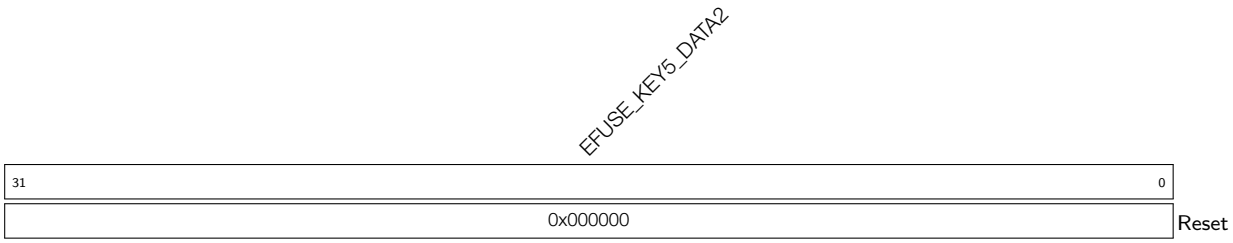
EFUSE_KEY5_DATA0 表示 KEY5 第 0 个 32 位内容。(RO)

Register 5.81. EFUSE_RD_KEY5_DATA1_REG (0x0140)



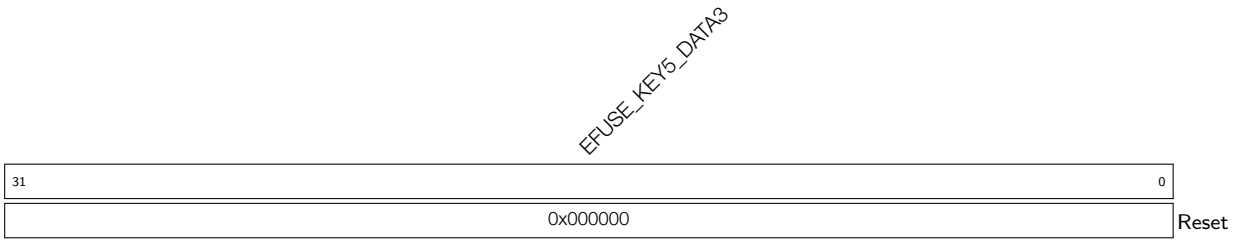
EFUSE_KEY5_DATA1 表示 KEY5 第 1 个 32 位内容。(RO)

Register 5.82. EFUSE_RD_KEY5_DATA2_REG (0x0144)



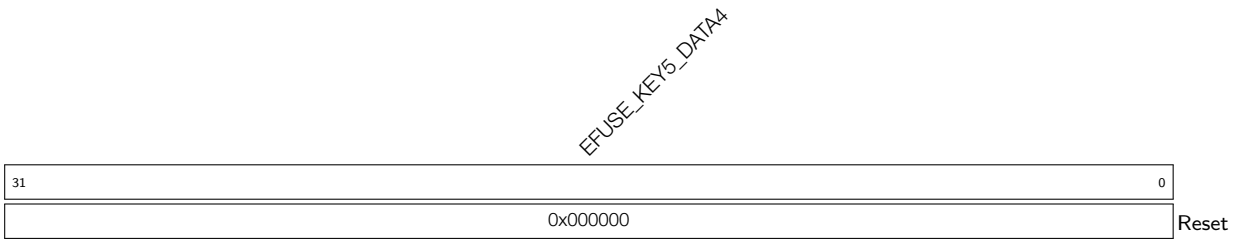
EFUSE_KEY5_DATA2 表示 KEY5 第 2 个 32 位内容。(RO)

Register 5.83. EFUSE_RD_KEY5_DATA3_REG (0x0148)



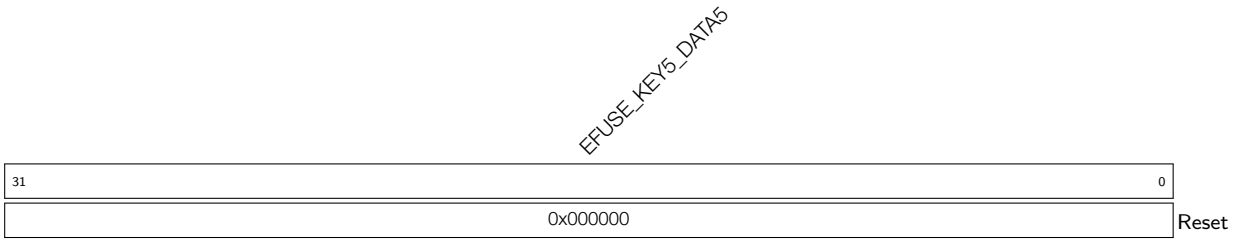
EFUSE_KEY5_DATA3 表示 KEY5 第 3 个 32 位内容。(RO)

Register 5.84. EFUSE_RD_KEY5_DATA4_REG (0x014C)



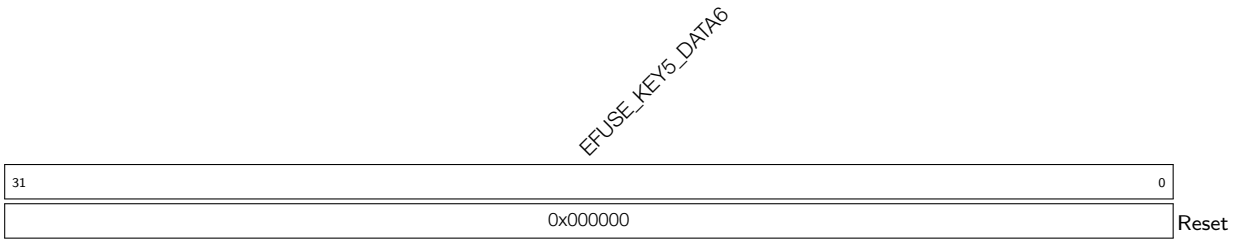
EFUSE_KEY5_DATA4 表示 KEY5 第 4 个 32 位内容。(RO)

Register 5.85. EFUSE_RD_KEY5_DATA5_REG (0x0150)



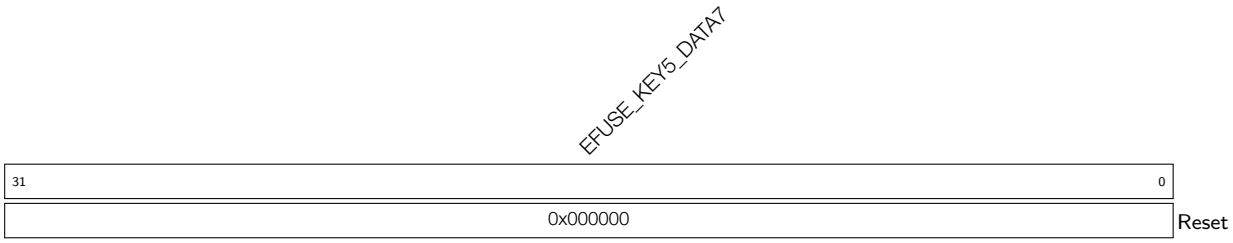
EFUSE_KEY5_DATA5 表示 KEY5 第 5 个 32 位内容。(RO)

Register 5.86. EFUSE_RD_KEY5_DATA6_REG (0x0154)



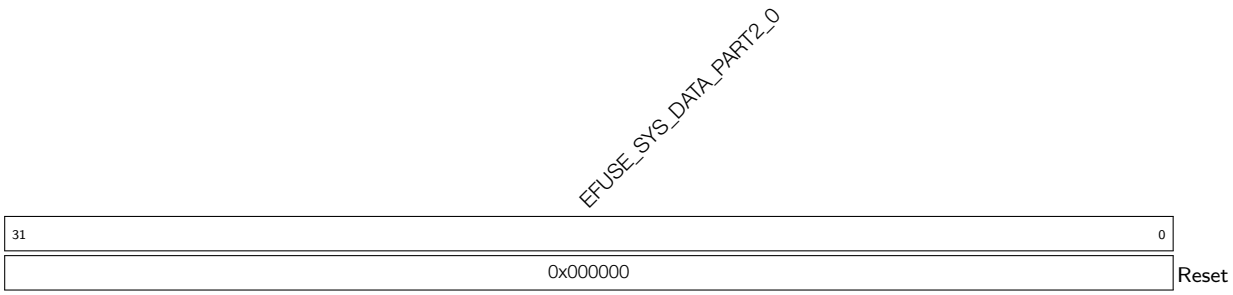
EFUSE_KEY5_DATA6 表示 KEY5 第 6 个 32 位内容。(RO)

Register 5.87. EFUSE_RD_KEY5_DATA7_REG (0x0158)



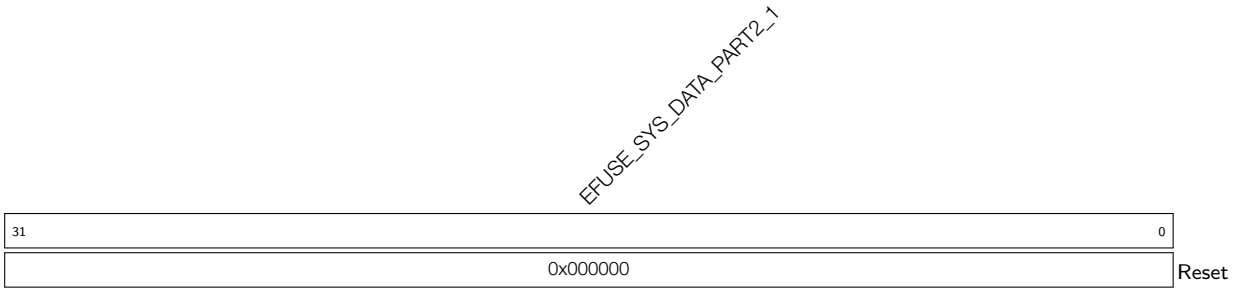
EFUSE_KEY5_DATA7 表示 KEY5 第 7 个 32 位内容。(RO)

Register 5.88. EFUSE_RD_SYS_PART2_DATA0_REG (0x015C)



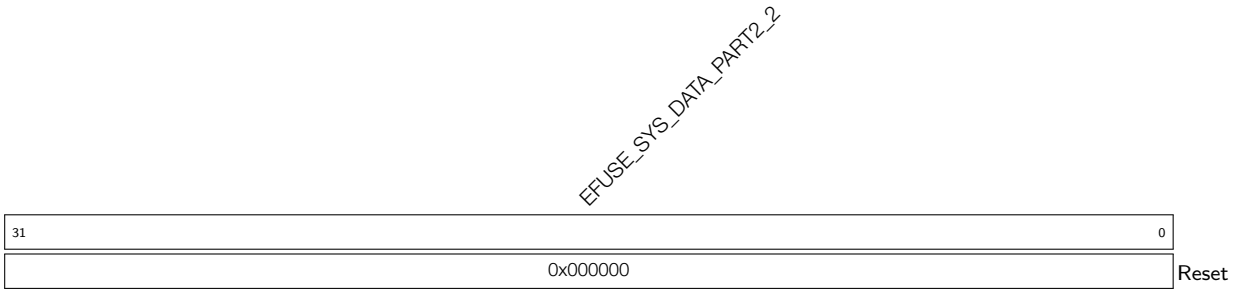
EFUSE_SYS_DATA_PART2_0 表示系统数据第 2 部分的第 0 个 32 位内容。(RO)

Register 5.89. EFUSE_RD_SYS_PART2_DATA1_REG (0x0160)



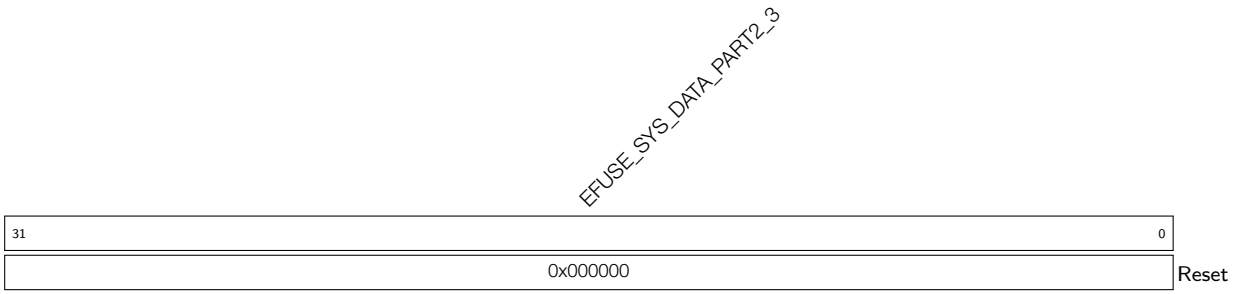
EFUSE_SYS_DATA_PART2_1 表示系统数据第 2 部分的第 1 个 32 位内容。(RO)

Register 5.90. EFUSE_RD_SYS_PART2_DATA2_REG (0x0164)



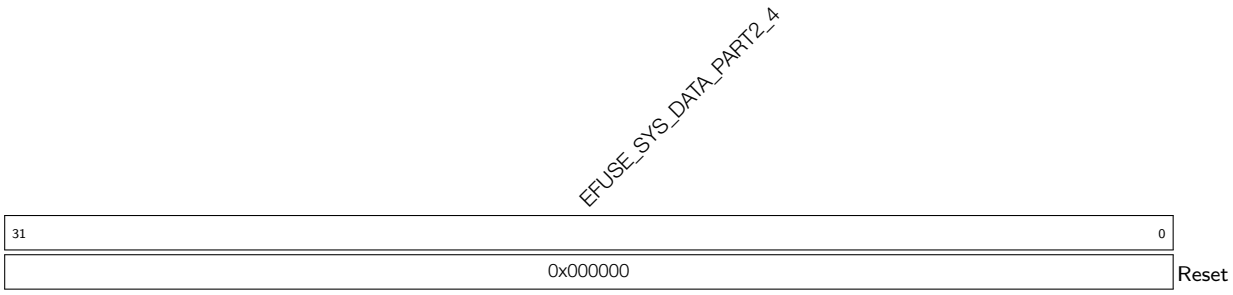
EFUSE_SYS_DATA_PART2_2 表示系统数据第 2 部分的第 2 个 32 位内容。(RO)

Register 5.91. EFUSE_RD_SYS_PART2_DATA3_REG (0x0168)



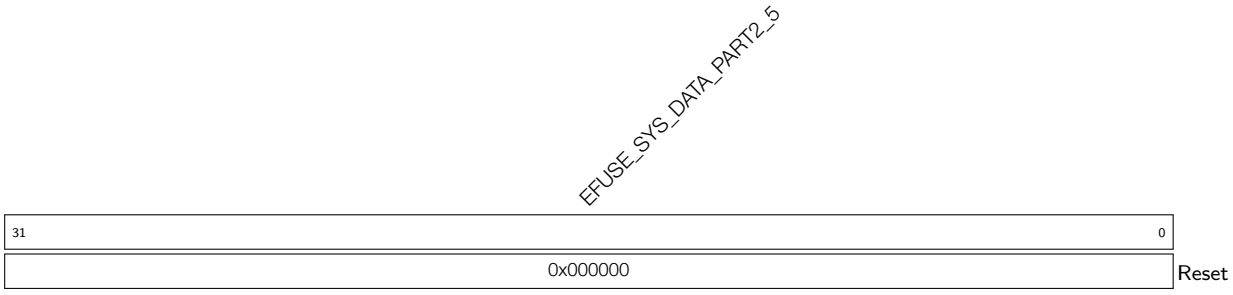
EFUSE_SYS_DATA_PART2_3 表示系统数据第 2 部分的第 3 个 32 位内容。(RO)

Register 5.92. EFUSE_RD_SYS_PART2_DATA4_REG (0x016C)



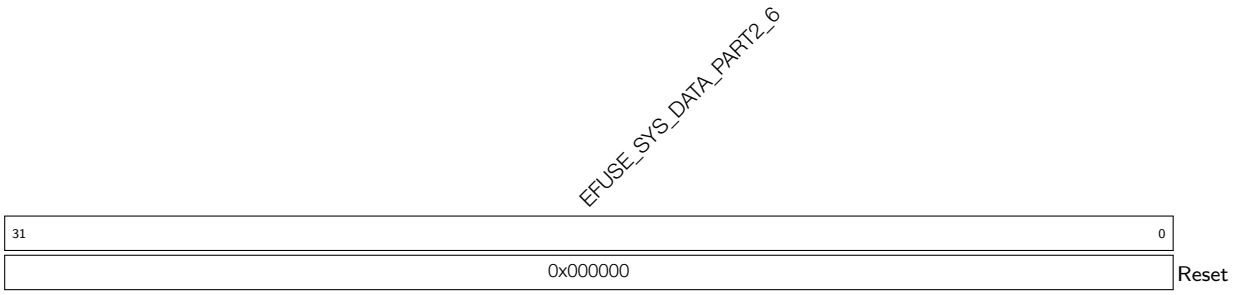
EFUSE_SYS_DATA_PART2_4 表示系统数据第 2 部分的第 4 个 32 位内容。(RO)

Register 5.93. EFUSE_RD_SYS_PART2_DATA5_REG (0x0170)



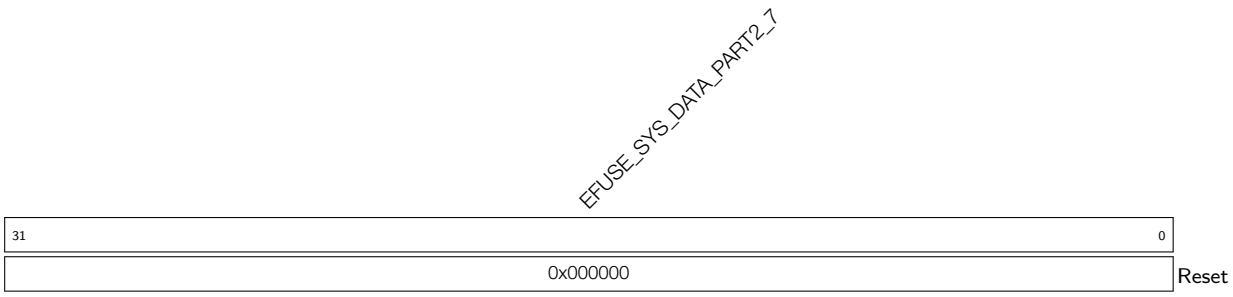
EFUSE_SYS_DATA_PART2_5 表示系统数据第 2 部分的第 5 个 32 位内容。(RO)

Register 5.94. EFUSE_RD_SYS_PART2_DATA6_REG (0x0174)



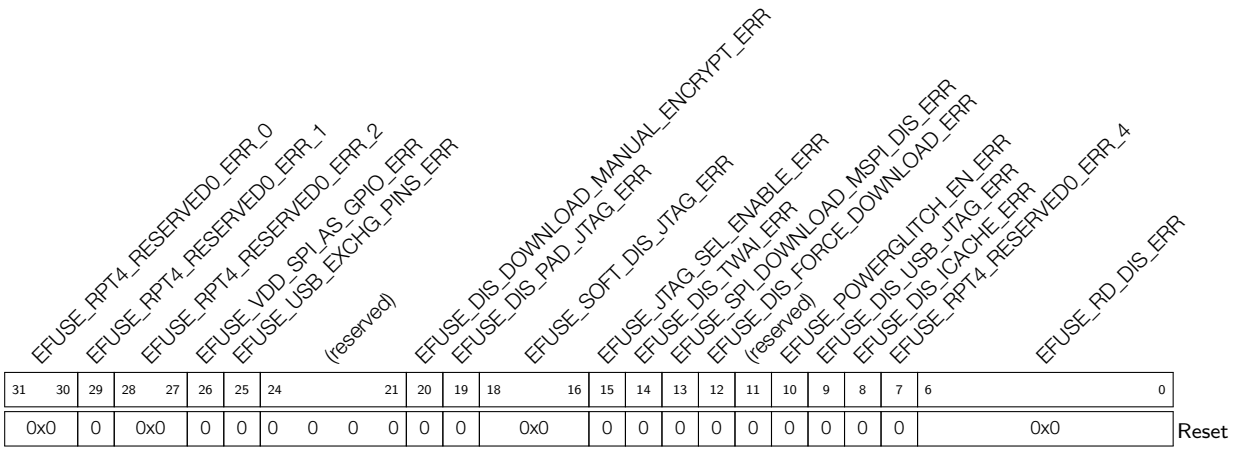
EFUSE_SYS_DATA_PART2_6 表示系统数据第 2 部分的第 6 个 32 位内容。(RO)

Register 5.95. EFUSE_RD_SYS_PART2_DATA7_REG (0x0178)



EFUSE_SYS_DATA_PART2_7 表示系统数据第 2 部分的第 7 个 32 位内容。(RO)

Register 5.96. EFUSE_RD_REPEAT_ERR0_REG (0x017C)



EFUSE_RD_DIS_ERR 若该域中的任何位为 1，则表示 RD_DIS 烧写错误。(RO)

EFUSE_RPT4_RESERVED0_ERR_4 保留。(RO)

EFUSE_DIS_ICACHE_ERR 若该位为 1，则表示 DIS_ICACHE 烧写错误。(RO)

EFUSE_DIS_USB_JTAG_ERR 若该位为 1，则表示 DIS_USB_JTAG 烧写错误。(RO)

EFUSE_POWERGLITCH_EN_ERR 若该位为 1，则表示 POWERGLITCH_EN 烧写错误。(RO)

EFUSE_DIS_FORCE_DOWNLOAD_ERR 若该位为 1，则表示 DIS_FORCE_DOWNLOAD 烧写错误。(RO)

EFUSE_SPI_DOWNLOAD_MSPI_DIS_ERR 若该位为 1，则表示 SPI_DOWNLOAD_MSPI_DIS 烧写错误。(RO)

EFUSE_DIS_TWAI_ERR 若该位为 1，则表示 DIS_TWAI 烧写错误。(RO)

EFUSE_JTAG_SEL_ENABLE_ERR 若该位为 1，则表示 JTAG_SEL_ENABLE 烧写错误。(RO)

EFUSE_SOFT_DIS_JTAG_ERR 若该域中的任何位为 1，则表示 SOFT_DIS_JTAG 烧写错误。(RO)

EFUSE_DIS_PAD_JTAG_ERR 若该位为 1，则表示 DIS_PAD_JTAG 烧写错误。(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR 若该位为 1，则表示 DIS_DOWNLOAD_MANUAL_ENCRYPT 烧写错误。(RO)

EFUSE_USB_EXCHG_PINS_ERR 若该位为 1，则表示 USB_EXCHG_PINS 烧写错误。(RO)

见下页……

Register 5.96. EFUSE_RD_REPEAT_ERR0_REG (0x0080)

接上页……

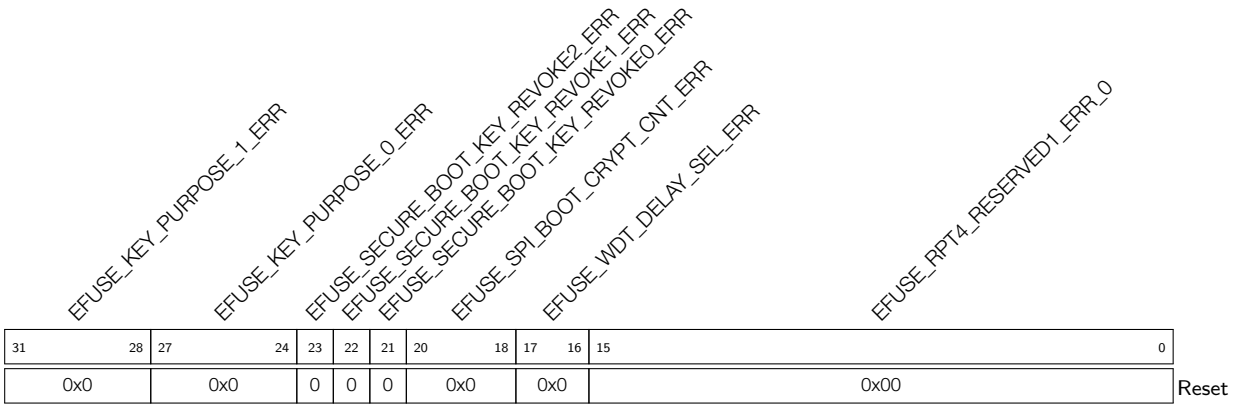
EFUSE_VDD_SPI_AS_GPIO_ERR 若该位为 1，则表示 VDD_SPI_AS_GPIO 烧写错误。(RO)

EFUSE_RPT4_RESERVED0_ERR_2 保留。(RO)

EFUSE_RPT4_RESERVED0_ERR_1 保留。(RO)

EFUSE_RPT4_RESERVED0_ERR_0 保留。(RO)

Register 5.97. EFUSE_RD_REPEAT_ERR1_REG (0x0180)



EFUSE_RPT4_RESERVED1_ERR_0 保留。(RO)

EFUSE_WDT_DELAY_SEL_ERR 若该域中的任何位为 1，则表示 WDT_DELAY_SEL 烧写错误。(RO)

EFUSE_SPI_BOOT_CRYPT_CNT_ERR 若该域中的任何位为 1，则表示 SPI_BOOT_CRYPT_CNT 烧写错误。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE0_ERR 若该位为 1，则表示 SECURE_BOOT_KEY_REVOKE0 烧写错误。(RO)

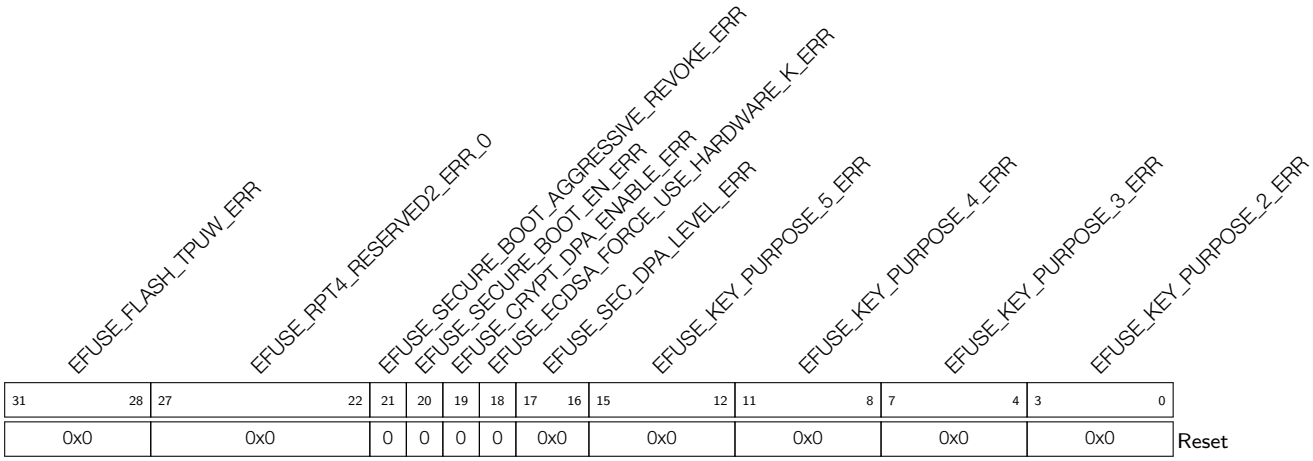
EFUSE_SECURE_BOOT_KEY_REVOKE1_ERR 若该位为 1，则表示 SECURE_BOOT_KEY_REVOKE1 烧写错误。(RO)

EFUSE_SECURE_BOOT_KEY_REVOKE2_ERR 若该位为 1，则表示 SECURE_BOOT_KEY_REVOKE2 烧写错误。(RO)

EFUSE_KEY_PURPOSE_0_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_0 烧写错误。(RO)

EFUSE_KEY_PURPOSE_1_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_1 烧写错误。(RO)

Register 5.98. EFUSE_RD_REPEAT_ERR2_REG (0x0184)



EFUSE_KEY_PURPOSE_2_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_2 烧写错误。(RO)

EFUSE_KEY_PURPOSE_3_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_3 烧写错误。(RO)

EFUSE_KEY_PURPOSE_4_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_4 烧写错误。(RO)

EFUSE_KEY_PURPOSE_5_ERR 若该域中的任何位为 1，则表示 KEY_PURPOSE_5 烧写错误。(RO)

EFUSE_SEC_DPA_LEVEL_ERR 若该域中的任何位为 1，则表示 SEC_DPA_LEVEL 烧写错误。(RO)

EFUSE_ECDSA_FORCE_USE_HARDWARE_K_ERR 若该位为 1，则表示 ECDSA_FORCE_USE_HARDWARE_K 烧写错误。(RO)

EFUSE_CRYPT_DPA_ENABLE_ERR 若该位为 1，则表示 CRYPT_DPA_ENABLE 烧写错误。(RO)

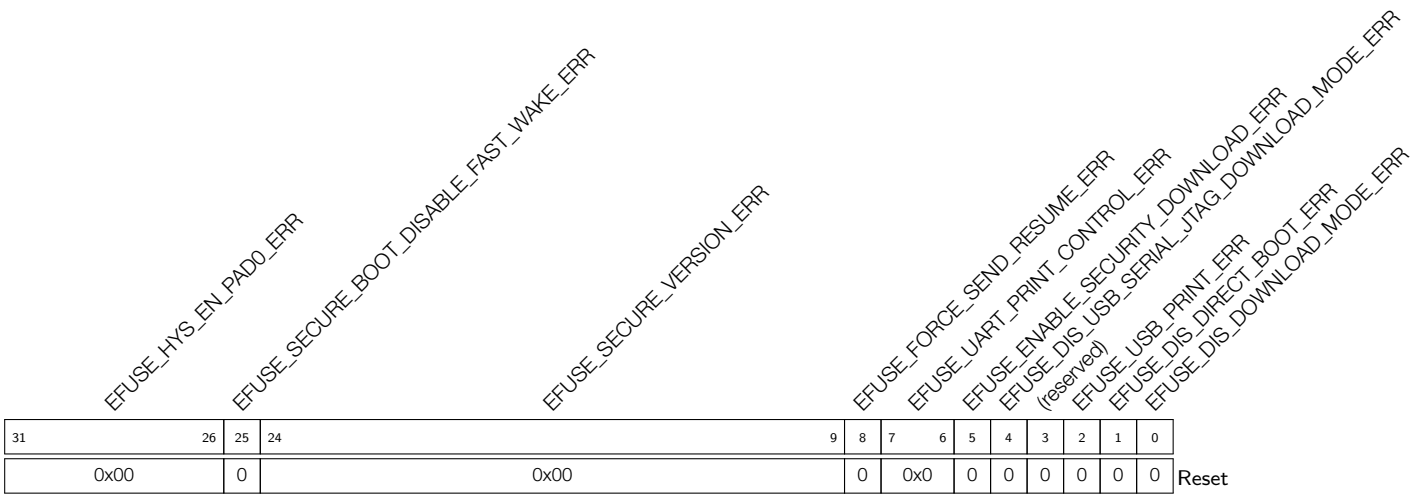
EFUSE_SECURE_BOOT_EN_ERR 若该位为 1，则表示 SECURE_BOOT_EN 烧写错误。(RO)

EFUSE_SECURE_BOOT_AGGRESSIVE_REVOKE_ERR 若该位为 1，则表示 SECURE_BOOT_AGGRESSIVE_REVOKE 烧写错误。(RO)

EFUSE_RPT4_RESERVED2_ERR_0 保留。(RO)

EFUSE_FLASH_TPUW_ERR 若该域中的任何位为 1，则表示 FLASH_TPUW 烧写错误。(RO)

Register 5.99. EFUSE_RD_REPEAT_ERR3_REG (0x0188)



EFUSE_DIS_DOWNLOAD_MODE_ERR 若该位为 1，则表示 DIS_DOWNLOAD_MODE 烧写错误。(RO)

EFUSE_DIS_DIRECT_BOOT_ERR 若该位为 1，则表示 DIS_DIRECT_BOOT 烧写错误。(RO)

EFUSE_USB_PRINT_ERR 若该位为 1，则表示 UART_PRINT_CHANNEL 烧写错误。(RO)

EFUSE_DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE_ERR 若该位为 1，则表示 DIS_USB_SERIAL_JTAG_DOWNLOAD_MODE 烧写错误。(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR 若该位为 1，则表示 ENABLE_SECURITY_DOWNLOAD 烧写错误。(RO)

EFUSE_UART_PRINT_CONTROL_ERR 若该域中的任何位为 1，则表示 UART_PRINT_CONTROL 烧写错误。(RO)

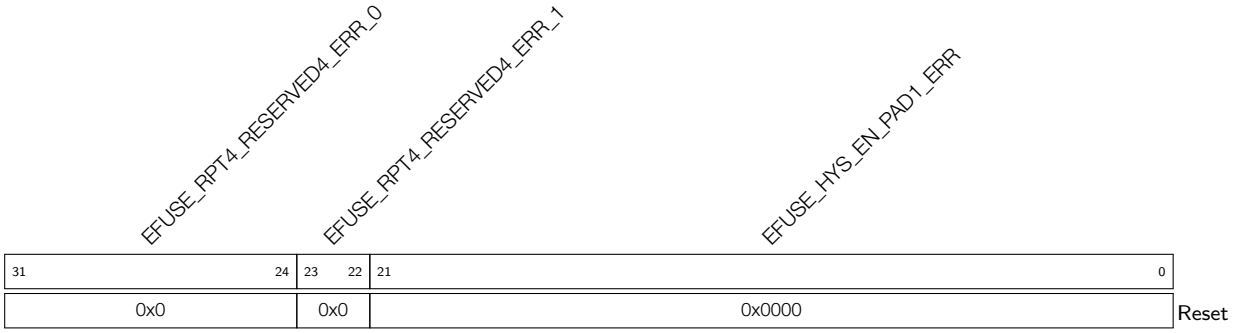
EFUSE_FORCE_SEND_RESUME_ERR 若该位为 1，则表示 FORCE_SEND_RESUME 烧写错误。(RO)

EFUSE_SECURE_VERSION_ERR 若该域中的任何位为 1，则表示 SECURE_VERSION 烧写错误。(RO)

EFUSE_SECURE_BOOT_DISABLE_FAST_WAKE_ERR 若该位为 1，则表示 SECURE_BOOT_DISABLE_FAST_WAKE 烧写错误。(RO)

EFUSE_HYS_EN_PAD0_ERR 若该域中的任何位为 1，则表示 HYS_EN_PAD0 烧写错误。(RO)

Register 5.100. EFUSE_RD_REPEAT_ERR4_REG (0x0190)

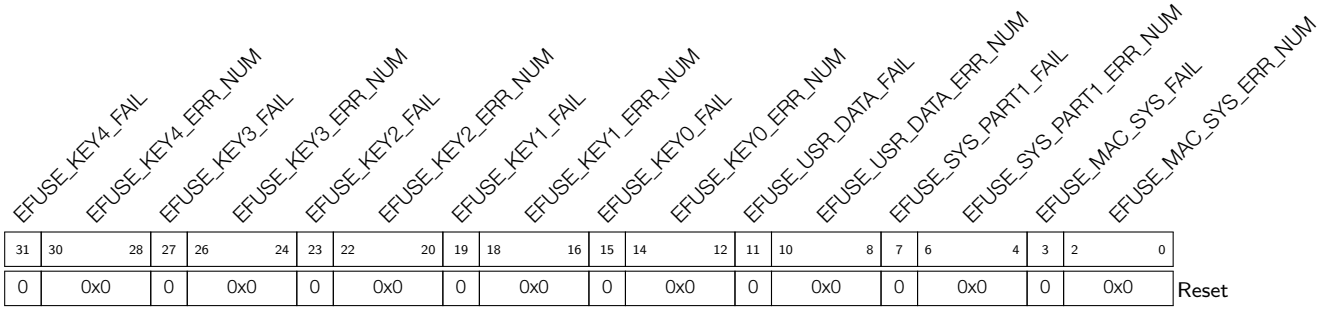


EFUSE_HYS_EN_PAD1_ERR 若该域中的任何位为 1，则表示 HYS_EN_PAD1 烧写错误。(RO)

EFUSE_RPT4_RESERVED4_ERR_1 保留。(RO)

EFUSE_RPT4_RESERVED4_ERR_0 保留。(RO)

Register 5.101. EFUSE_RD_RS_ERR0_REG (0x01C0)



EFUSE_MAC_SYS_ERR_NUM 表示错误字节数。(RO)

EFUSE_MAC_SYS_FAIL 表示 MAC_SYS 的数据是否烧写失败。

0：无烧写错误，MAC_SYS 的数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

EFUSE_SYS_PART1_ERR_NUM 表示错误字节数。(RO)

EFUSE_SYS_PART1_FAIL 表示第 1 部分的系统数据是否烧写失败。

0：无烧写错误，第 1 部分的系统数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

EFUSE_USR_DATA_ERR_NUM 表示错误字节数。(RO)

EFUSE_USR_DATA_FAIL 表示用户数据烧写是否失败。

0：无烧写错误，用户数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

EFUSE_KEY0_ERR_NUM 表示错误字节数。(RO)

EFUSE_KEY0_FAIL 表示 key0 数据是否烧写失败。

0：无烧写错误，key0 数据是可靠的

1：key0 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_KEY1_ERR_NUM 表示错误字节数。(RO)

EFUSE_KEY1_FAIL 表示 key1 数据是否烧写失败。

0：无烧写错误，key1 数据是可靠的

1：key1 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_KEY2_ERR_NUM 表示错误字节数。(RO)

EFUSE_KEY2_FAIL 表示 key2 数据是否烧写失败。

0：无烧写错误，key2 数据是可靠的

1：key2 数据烧写失败，错误字节数超过 6

(RO)

见下页……

Register 5.101. EFUSE_RD_RS_ERR0_REG (0x01C0)

接上页……

EFUSE_KEY3_ERR_NUM 表示错误字节数。(RO)

EFUSE_KEY3_FAIL 表示 key3 数据是否烧写失败。

0：无烧写错误，key3 数据是可靠的

1：key3 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_KEY4_ERR_NUM 表示错误字节数。(RO)

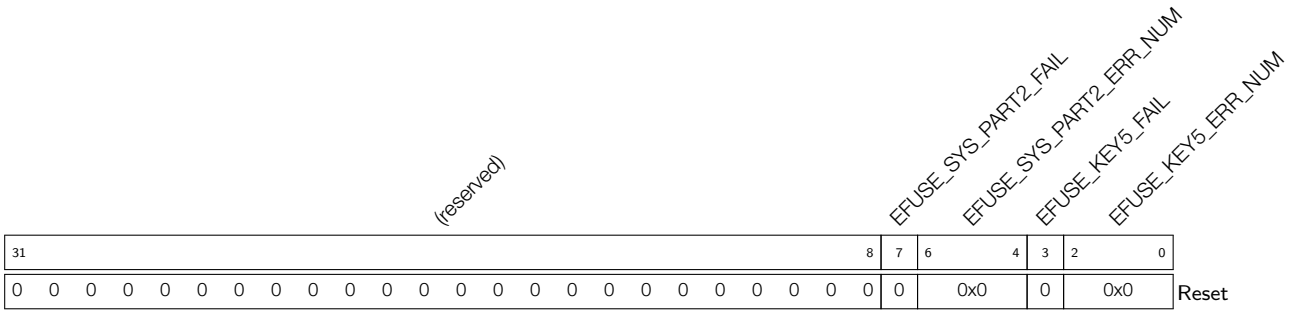
EFUSE_KEY4_FAIL 表示 key4 数据是否烧写失败。

0：无烧写错误，key4 数据是可靠的

1：key4 数据烧写失败，错误字节数超过 6

(RO)

Register 5.102. EFUSE_RD_RS_ERR1_REG (0x01C4)



EFUSE_KEY5_ERR_NUM 表示错误字节数。(RO)

EFUSE_KEY5_FAIL 表示 key5 数据是否烧写失败。

0：无烧写错误，key5 数据是可靠的

1：key5 数据烧写失败，错误字节数超过 6

(RO)

EFUSE_SYS_PART2_ERR_NUM 表示错误字节数。(RO)

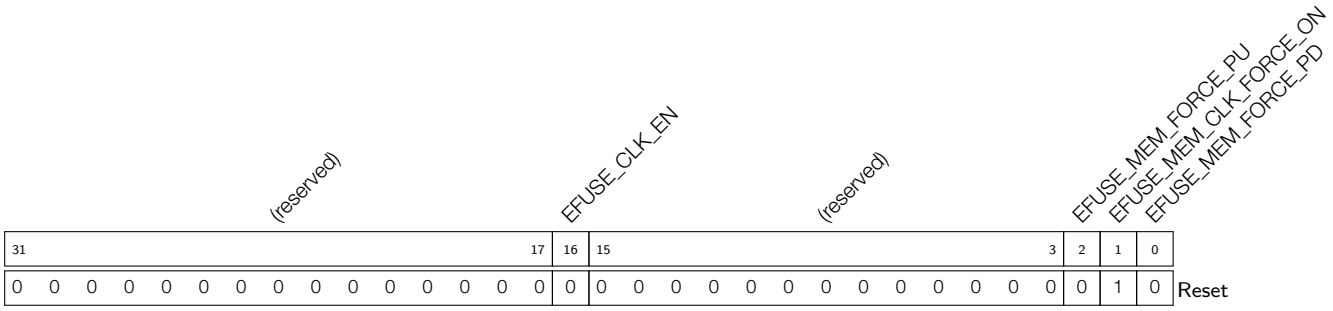
EFUSE_SYS_PART2_FAIL 表示第 2 部分的系统数据是否烧写失败。

0：无烧写错误，第 2 部分的系统数据是可靠的

1：用户数据烧写失败，错误字节数超过 6

(RO)

Register 5.103. EFUSE_CLK_REG (0x01C8)



EFUSE_MEM_FORCE_PD 配置是否强制 eFuse SRAM 进入节能模式。

1：强制

0：没有作用

(R/W)

EFUSE_MEM_CLK_FORCE_ON 配置是否强制激活 eFuse SRAM 时钟信号。

1：强制激活

0：没有作用

(R/W)

EFUSE_MEM_FORCE_PU 配置是否强制 eFuse SRAM 进入工作模式。

1：强制

0：没有作用

(R/W)

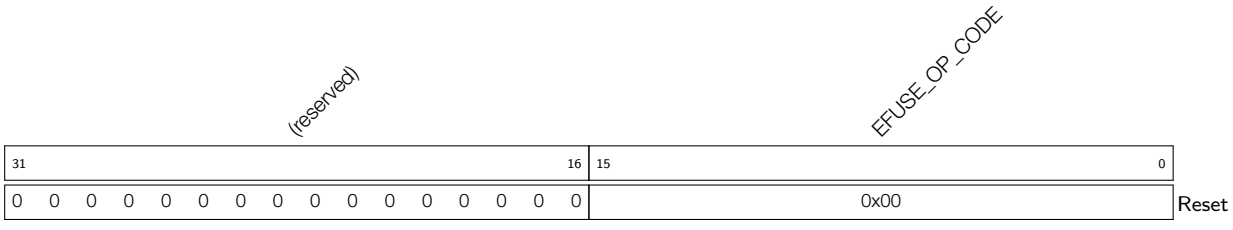
EFUSE_CLK_EN 配置是否强制使能 eFuse 寄存器配置时钟信号。

1：强制使能

0：时钟仅在读写寄存器时开启

(R/W)

Register 5.104. EFUSE_CONF_REG (0x01CC)



EFUSE_OP_CODE 配置操作指令类型。

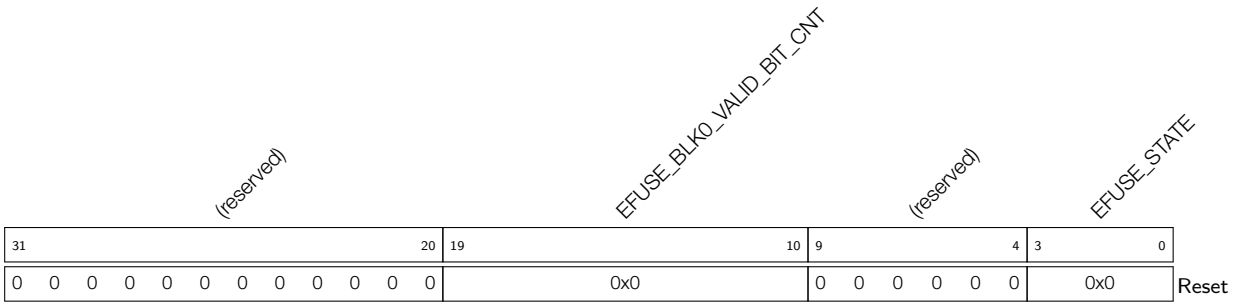
0x5A5A：烧写操作指令

0x5AA5：读操作指令

其他值：没有作用

(R/W)

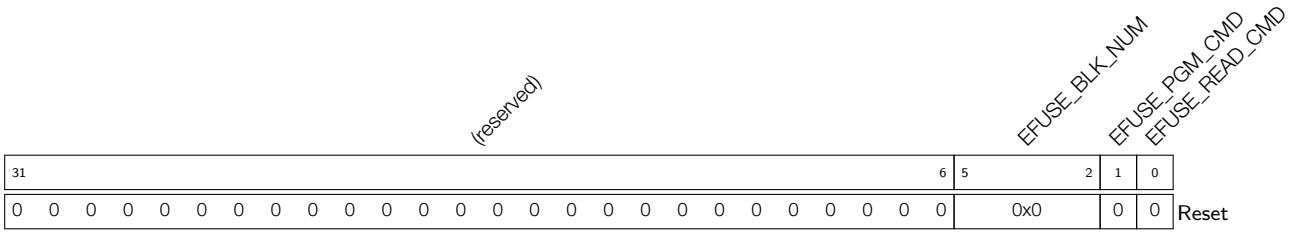
Register 5.105. EFUSE_STATUS_REG (0x01D0)



EFUSE_STATE 表示 eFuse 状态机的状态。(RO)

EFUSE_BLK0_VALID_BIT_CNT 表示块中有效位的数量。(RO)

Register 5.106. EFUSE_CMD_REG (0x01D4)



EFUSE_READ_CMD 配置是否发送读指令。

1：发送

0：没有作用

(R/W/SC)

EFUSE_PGM_CMD 配置是否发送烧写指令。

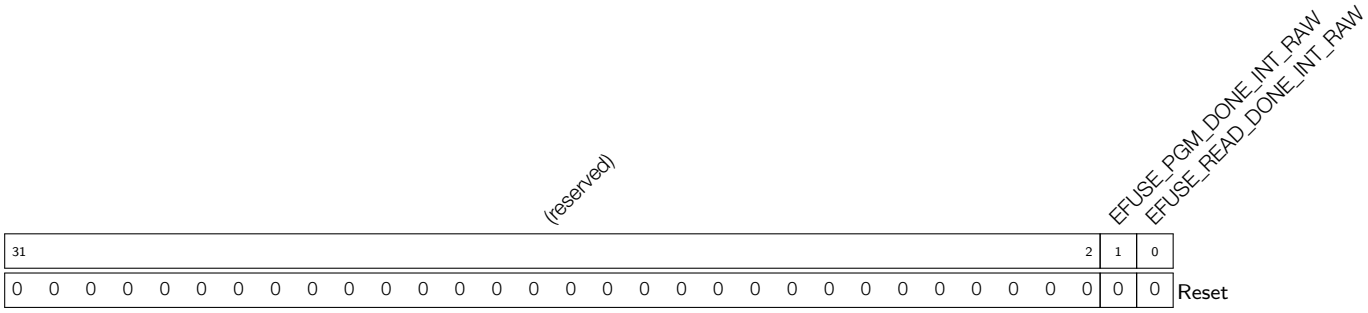
1：发送

0：没有作用

(R/W/SC)

EFUSE_BLK_NUM 表示待烧写块的序号。值 0 ~ 10 分别对应 BLOCK0 ~ BLOCK10。(R/W)

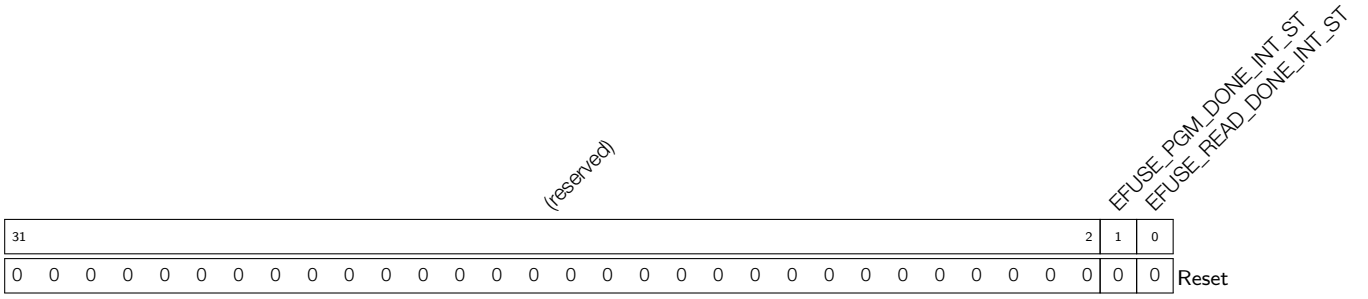
Register 5.107. EFUSE_INT_RAW_REG (0x01D8)



EFUSE_READ_DONE_INT_RAW 读取完成的原始中断状态。(R/SS/WTC)

EFUSE_PGM_DONE_INT_RAW 烧写完成的原始中断状态。(R/SS/WTC)

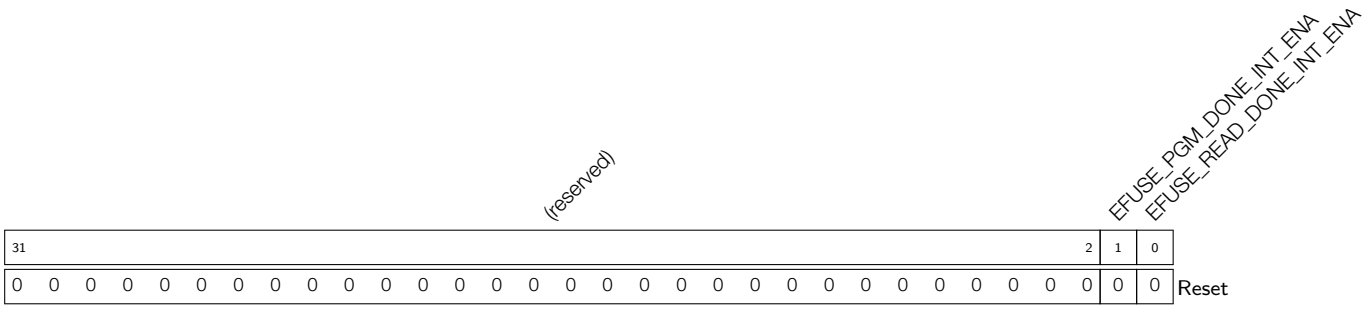
Register 5.108. EFUSE_INT_ST_REG (0x01DC)



EFUSE_READ_DONE_INT_ST 读取完成的屏蔽中断状态。(RO)

EFUSE_PGM_DONE_INT_ST 烧写完成的屏蔽中断状态。(RO)

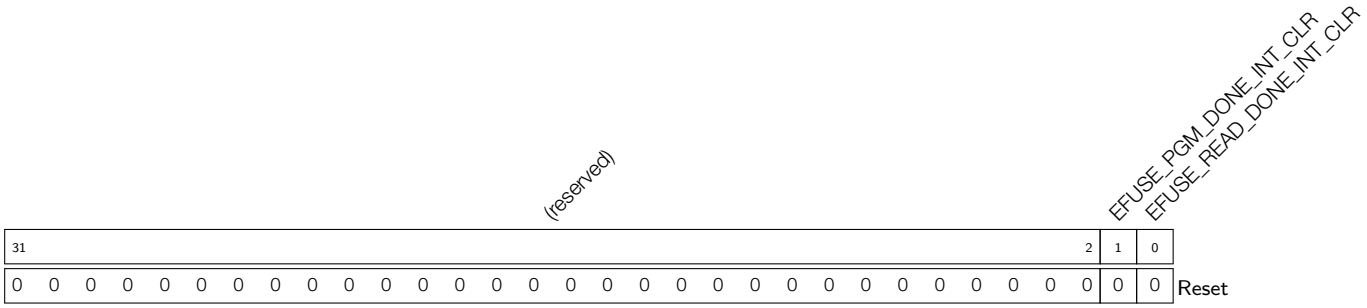
Register 5.109. EFUSE_INT_ENA_REG (0x01E0)



EFUSE_READ_DONE_INT_ENA 写 1 使能读取完成的中断。(R/W)

EFUSE_PGM_DONE_INT_ENA 写 1 使能烧写完成的中断。(R/W)

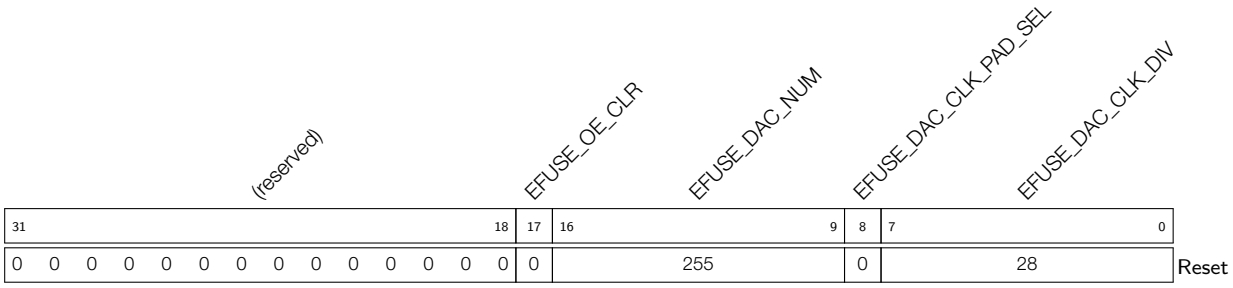
Register 5.110. EFUSE_INT_CLR_REG (0x01E4)



EFUSE_READ_DONE_INT_CLR 写 1 清除读取完成的中断。(WO)

EFUSE_PGM_DONE_INT_CLR 写 1 清除烧写完成的中断。(WO)

Register 5.111. EFUSE_DAC_CONF_REG (0x01E8)



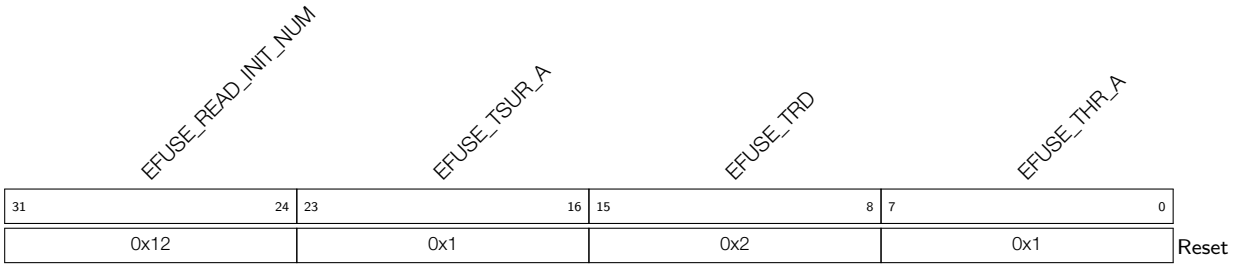
EFUSE_DAC_CLK_DIV 配置烧写电压的爬升时钟分频系数。(R/W)

EFUSE_DAC_CLK_PAD_SEL 无关项。(R/W)

EFUSE_DAC_NUM 配置烧写电压上升周期，单位：EFUSE_DAC_CLK_DIV 分频后的时钟频率。(R/W)

EFUSE_OE_CLR 降低烧写电压的供电能力。(R/W)

Register 5.112. EFUSE_RD_TIM_CONF_REG (0x01EC)



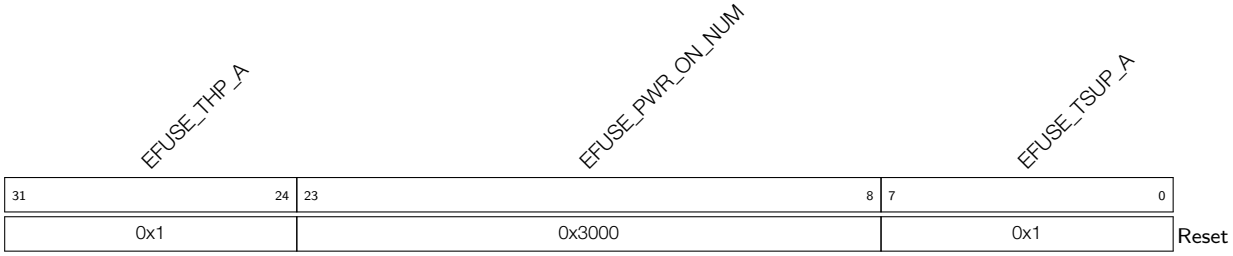
EFUSE_THR_A 配置读操作的保持时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_TRD 配置读操作的时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_TSUR_A 配置读操作的设置时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_READ_INIT_NUM 配置读取 eFuse 存储器的等待时间，单位：一个 eFuse 核心时钟周期。(R/W)

Register 5.113. EFUSE_WR_TIM_CONF1_REG (0x01F0)

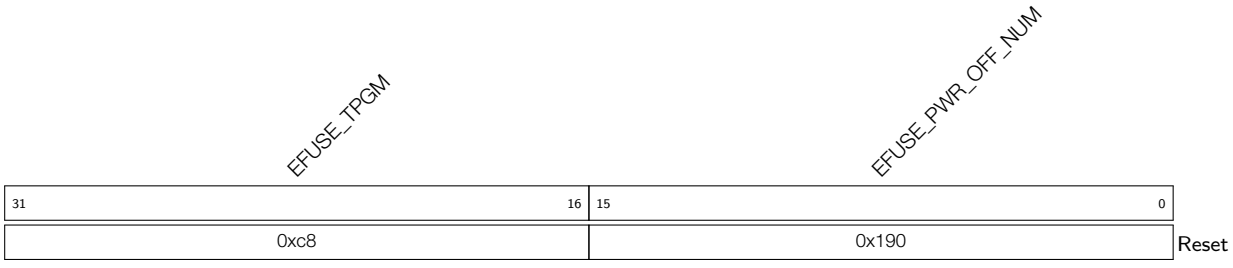


EFUSE_TSUP_A 配置烧写的设置时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_PWR_ON_NUM 配置 VDDQ 的上电时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_THP_A 配置烧写的保持时间，单位：一个 eFuse 核心时钟周期。(R/W)

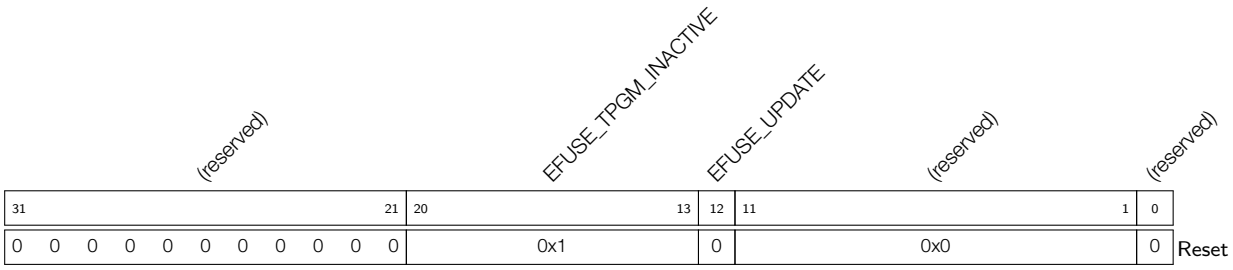
Register 5.114. EFUSE_WR_TIM_CONF2_REG (0x01F4)



EFUSE_PWR_OFF_NUM 配置 VDDQ 的掉电时间，单位：一个 eFuse 核心时钟周期。(R/W)

EFUSE_TPGM 配置活跃状态的烧写时间，单位：一个 eFuse 核心时钟周期。(R/W)

Register 5.115. EFUSE_WR_TIM_CONF0_REG (0x01F8)



EFUSE_UPDATE 配置是否更新多位寄存器信号。

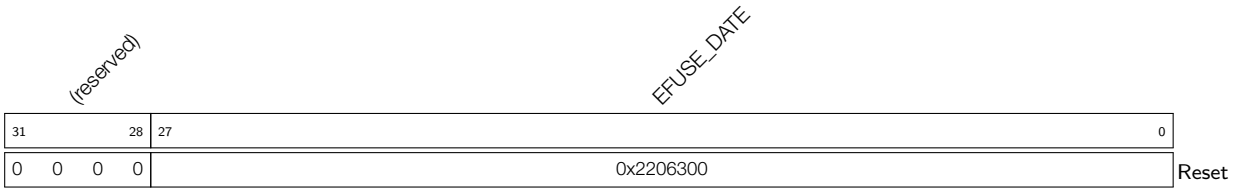
1：更新

0：没有作用

(WT)

EFUSE_TPGM_INACTIVE 配置非活跃状态的烧写时间，单位：一个 eFuse 核心时钟周期。(R/W)

Register 5.116. EFUSE_DATE_REG (0x01FC)



EFUSE_DATE 版本控制寄存器。(R/W)

6 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)

6.1 概述

ESP32-H2 芯片有 19 个通用输入输出管脚 (GPIO Pin)。每个管脚都可用作一个通用 IO，或连接一个内部的外设信号。利用 GPIO 交换矩阵和 IO MUX，可配置外设模块的输入信号来源于任何的 IO 管脚，并且外设模块的输出信号也可连接到任意 IO 管脚。这些模块共同组成了芯片的 IO 控制。

注意：这 19 个物理 GPIO 管脚的编号为：GPIO0 ~ GPIO5、GPIO8 ~ GPIO14、GPIO22 ~ GPIO27。

6.2 主要特性

GPIO 交换矩阵具有如下特性：

- GPIO 交换矩阵是外设输入输出信号和 GPIO 管脚之间的全交换矩阵；
- 78 个外设输入信号可以选择任意一个 GPIO 管脚的输入信号；
- 每个 GPIO 管脚的输出信号可以来自 99 个外设输出信号的任意一个；
- 支持输入信号经 GPIO SYNC 模块同步至 IO MUX 的运行时钟。关于 IO MUX 的运行时钟，详情请见章节 7 复位和时钟；
- 支持 GPIO 滤波器对输入信号进行滤波；
- 支持毛刺滤波器对输入信号进行二次滤波；
- 支持 Sigma Delta 调制输出 (SDM)；
- 支持 GPIO 简单输入输出。

IO MUX 具有如下特性：

- 支持 SPI、JTAG、UART 等信号旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以此类信号可以直接通过 IO MUX 输入和输出。
- 为每个 GPIO 管脚提供一个寄存器 IO_MUX_GPIO*n*_REG，每个管脚可配置成：
  - GPIO 功能，连接 GPIO 交换矩阵；
  - 直连功能，旁路 GPIO 交换矩阵。

6.3 结构概览

图 6-1 所示为 GPIO 交换矩阵和 IO MUX 将信号引入外设和引出至管脚的具体过程。

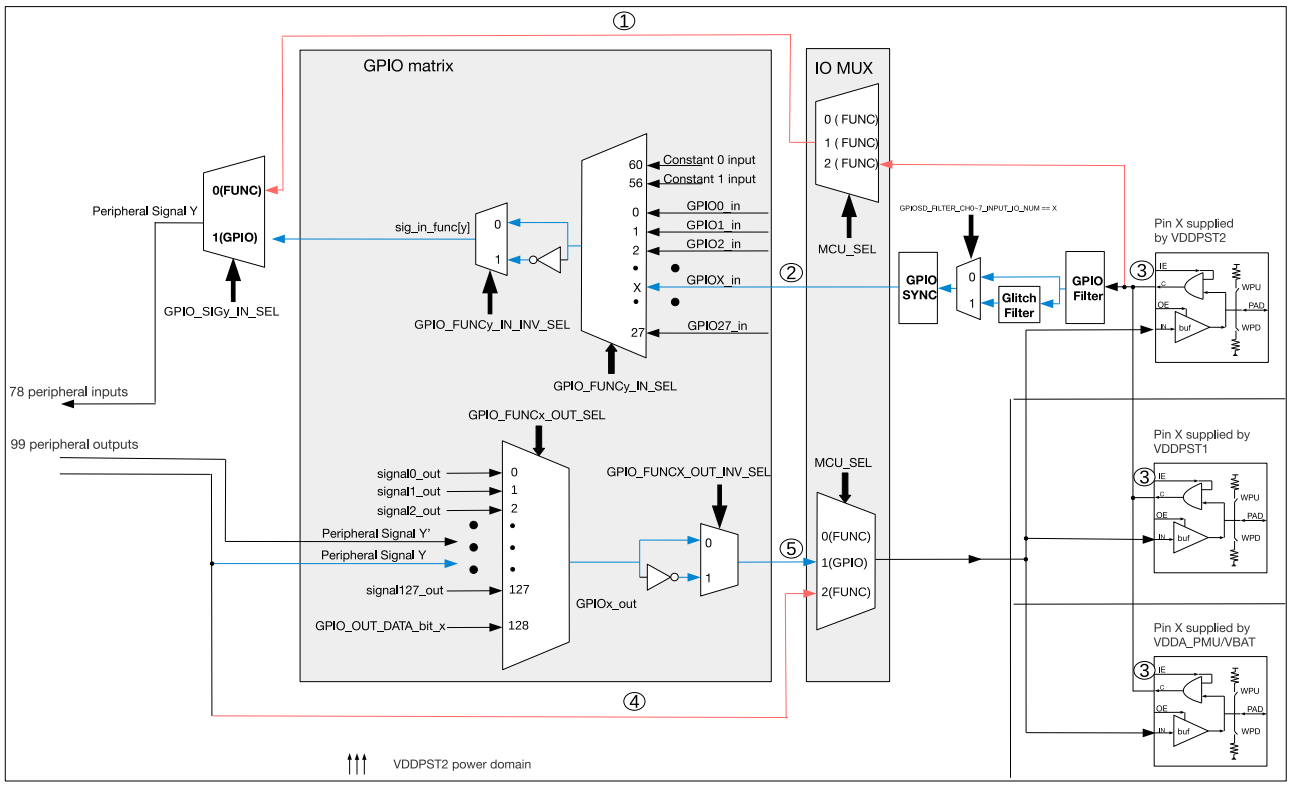


图 6-1. IO MUX 和 GPIO 交换矩阵框图

1. 仅有部分输入信号可以直接通过 IO MUX 直连外设，这些输入信号在表 6-2 “信号可经由 IO MUX 直接输入”一栏中被标为“yes”。剩余其他信号只能通过 GPIO 交换矩阵连接至外设；
2. ESP32-H2 共有 19 个 GPIO 管脚，因此从 GPIO SYNC 进入到 GPIO 交换矩阵的输入共有 19 个。注意：
3. GPIO 管脚由 IE、OE、WPU 和 WPD 信号控制；
4. 仅有部分输出信号可通过 IO MUX 直连管脚，这些输出信号在表 6-2 “信号可经由 IO MUX 直接输出”一栏中被标为“yes”。剩余其他信号只能通过 GPIO 交换矩阵连接至管脚；
5. 从 GPIO 交换矩阵到 IO MUX 的输出共有 19 个，对应 GPIO *X*：0 ~ 5、8 ~ 14、22 ~ 27。

图 6-2 展示了芯片焊盘 (PAD) 的内部结构，即芯片逻辑与 GPIO 管脚之间的电气接口。19 个 GPIO 管脚均采用这一结构，且由 IE、OE、WPU 和 WPD 信号控制。

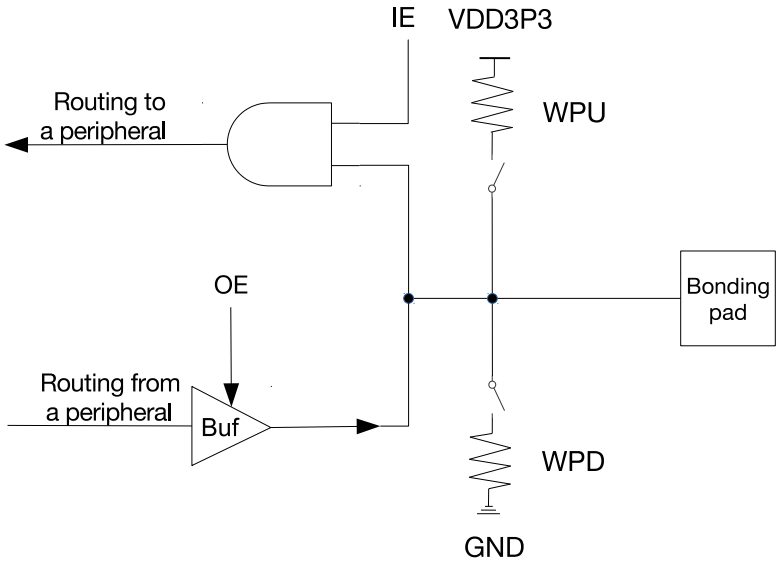


图 6-2. 焊盘内部结构

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

6.4 通过 GPIO 交换矩阵的外设输入

6.4.1 概述

为实现通过 GPIO 交换矩阵接收外部输入信号，需要配置 GPIO 交换矩阵从 19 个 GPIO (0 ~ 5、8 ~ 14、22 ~ 27) 中获取外部输入信号，见交换矩阵表格 6-2。并需要配置外设输入选择通过 GPIO 交换矩阵接收输入信号。

如图 6-1 所示，使用 GPIO 交换矩阵将信号从管脚输入时，所有外部输入信号均来源于 GPIO 管脚，然后经 GPIO 滤波器进行滤波，见章节 6.4.3 中的步骤 2。

毛刺滤波器硬件可从 GPIO 滤波器输出的信号中选择 8 个信号进行滤波，其他没被选中的信号则直接进入 GPIO SYNC 硬件，见章节 6.4.3 中的步骤 3。

所有信号经 GPIO 滤波器硬件或毛刺滤波器硬件滤波后，均将被 GPIO SYNC 硬件同步至 IO MUX 的运行时钟，然后进入 GPIO 交换矩阵，见章节 6.4.2。外部输入信号也可经 IO MUX 直接输入至外设，但信号无法进行滤波及时钟同步操作。

6.4.2 信号同步

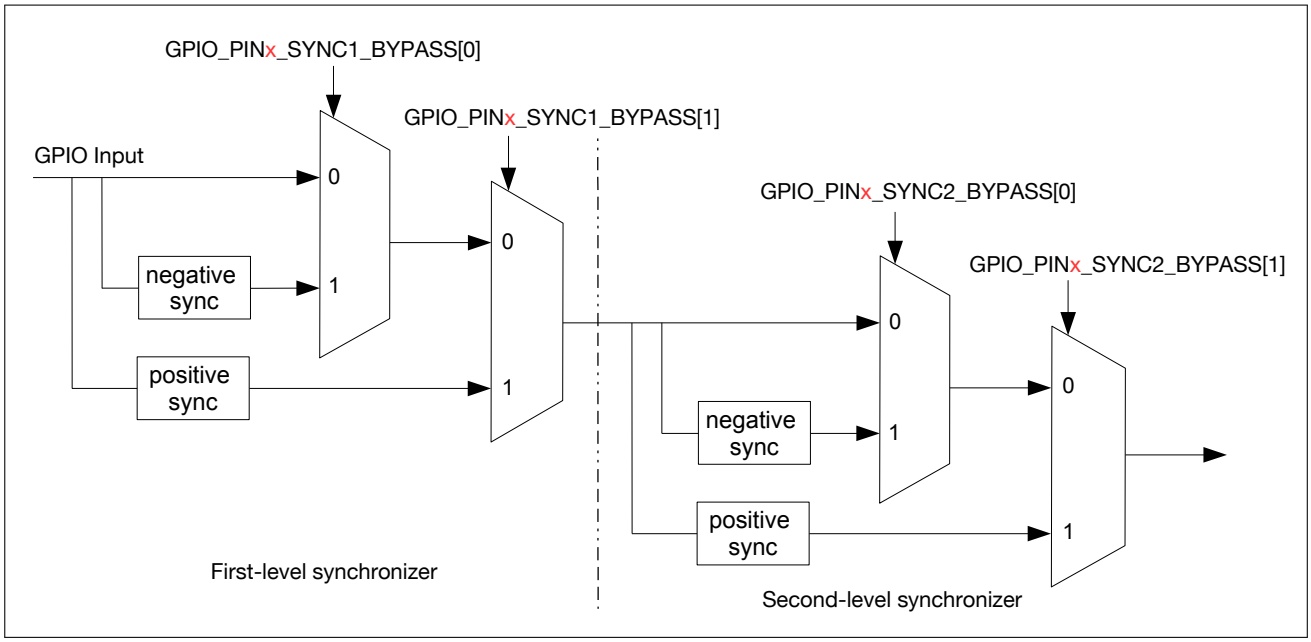


图 6-3. GPIO 输入经 IO MUX 运行时钟上升沿或下降沿同步

GPIO SYNC 模块的功能如图 6-3 所示。其中，negative sync 为 GPIO 输入经过 IO MUX 的运行时钟的下降沿同步，positive sync 为 GPIO 输入经过 IO MUX 的运行时钟上升沿同步。

同步器 (synchronizer) 默认关闭同步功能，即 GPIO_PIN*x*_SYNC1/2_BYPASS[1:0] = 0。但如果一个异步外设信号连接到管脚时，该信号应通过两级同步器（即图中的 first-level synchronizer 和 second-level synchronizer）进行同步，以减小亚稳态产生的概率。更多信息，见下一章节中的步骤 4。

6.4.3 功能描述

把某个外设输入信号 *Y* 绑定到某个 GPIO 管脚 *X*[1] 的配置过程如下：

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

   注意： 并不是所有外设信号都有有效的 GPIO_SIG*y*_IN_SEL 位，即有些外设信号只能通过 GPIO 交换矩阵接收外部输入信号。
2. 可选：置位 IO_MUX_GPIO*x*_FILTER_EN 使能 GPIO 管脚的输入信号滤波功能，如图 6-4 所示。只有当输入信号的有效宽度大于两个时钟周期时，输入信号才会被采样。否则，输入信号将会被滤掉。

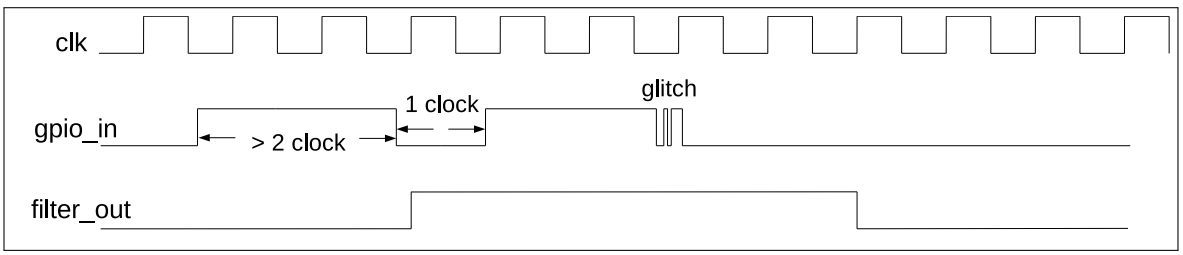


图 6-4. GPIO 输入信号滤波时序图

3. 毛刺滤波器硬件支持八个通道，每个通道可从 GPIO 滤波器硬件输出的 19 个信号 (0 ~ 5、8 ~ 14、22 ~ 27) 中选择一个信号，进行二次滤波。该毛刺滤波器硬件可用于对慢速信号进行滤波。如需使能该功能，见如下步骤：

   - 配置 GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM 为 *m*。此处 *n* 为通道编号，取值范围为：0 ~ 7；*m* 为 GPIO 管脚编号，取值范围为：0 ~ 5、8 ~ 14、22 ~ 27。
   - 配置 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 为 **VALUE1**，配置 GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 为 **VALUE2**。在 **VALUE1** + 1 个周期内，如果有 **VALUE2** + 1 个输入信号与当前输出信号值不一致，则毛刺滤波器硬件会将输出信号反转。用户可将 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 和 GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 同时设置为 **VALUE3**，则仅有有效宽度大于 **VALUE3** + 1 个时钟周期的信号会被采样。
   - 置位 GPIO_EXT_FILTER_CH*n*_EN 使能通道 *n*。

   相关示例见图 6-5，其中 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 配置为 3，GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 配置为 2。则在 T1 之前的四个时钟周期内，输出信号值 (signal_out) 一直为 “0”；输入信号值 (signal_in) 有三个时钟周期为 “1”，则输出信号在 T1 之后反转为 “1”。

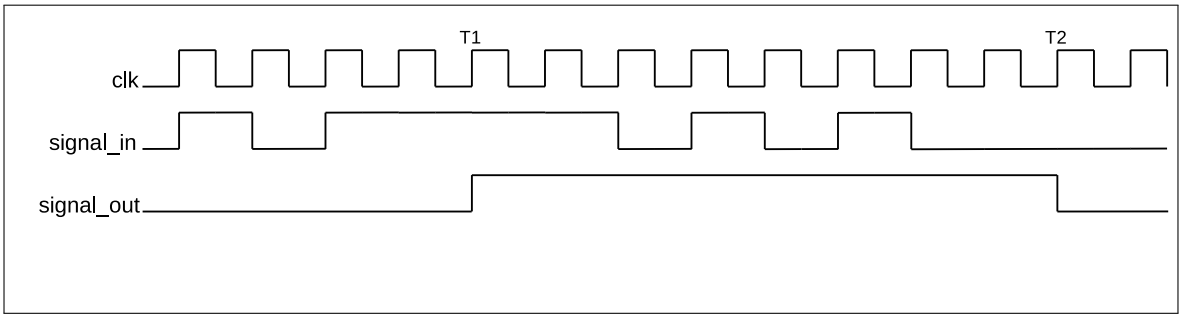


图 6-5. 毛刺滤波器时序示例

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

   - 如图 6-3 所示，配置 GPIO_PIN*x*_SYNC1_BYPASS 使能输入信号在第一级同步中为上升沿或下降沿同步。
   - 如图 6-3 所示，配置 GPIO_PIN*x*_SYNC2_BYPASS 使能输入信号在第二级同步中为上升沿或下降沿同步。

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

   - 置位 IO_MUX_GPIO*x*_FUN_IE 使能输入[2]。
   - 置位或清零 IO_MUX_GPIO*x*_FUN_WPU 和 IO_MUX_GPIO*x*_FUN_WPD，使能或关闭内部上拉/下拉电阻。

例如，要把 I2S MCLK 输入信号 [3]（I2S_MCLK_in，信号索引号 12）绑定到 GPIO3，请按照以下步骤操作。注意，GPIO3 也叫做 MTDO 管脚。

1. 置位 GPIO_FUNC12_IN_SEL_CFG_REG 寄存器的 GPIO_SIG12_IN_SEL 位，使能通过 GPIO 交换矩阵接收外部输入信号；
2. 配置 GPIO_FUNC12_IN_SEL_CFG_REG 寄存器中的 GPIO_FUNC12_IN_SEL 为 3，即选择管脚 GPIO3；
3. 置位 IO_MUX_GPIO3_REG 寄存器中 IO_MUX_GPIO3_FUN_IE 位使能管脚输入。

说明：

1. 同一个输入管脚可以同时绑定多个输入信号；
2. 置位 GPIO_FUNC*y*_IN_INV_SEL 可以把输入信号取反；
3. 无需将输入信号绑定到一个 GPIO 管脚也可以使外设读取恒低或恒高电平的输入值。实现方式为选择特定的 GPIO_FUNC*y*_IN_SEL 输入值而不是一个 GPIO 序号：
   - 设置 GPIO_FUNC*y*_IN_SEL 为 0x3C，则输入信号恒为 0；
   - 设置 GPIO_FUNC*y*_IN_SEL 为 0x38，则输入信号恒为 1。

6.4.4 简单 GPIO 输入

GPIO 交换矩阵也可用于简单 GPIO 输入，即任意 GPIO 管脚的值均可随时读取，而无需将 GPIO 管脚输入绑定到某个外设信号。其中，每个 GPIO 管脚的输入值保存在 GPIO_IN_REG 寄存器中。

配置简单 GPIO 输入，具体过程如下：

- 配置 GPIO 管脚 *X* 对应的 IO_MUX_GPIO*x*_REG 中 IO_MUX_GPIO*x*_FUN_IE，使能管脚输入；
- 读取 GPIO_IN_REG[*x*]，即可实现简单 GPIO 输入。

6.5 通过 GPIO 交换矩阵的外设输出

6.5.1 概述

为实现通过 GPIO 交换矩阵输出外设信号，需要配置 GPIO 交换矩阵将外设信号（即在表 6-2 中“输出信号”一栏所列出的信号）输出到 19 个 GPIO (0 ~ 5、8 ~ 14、22 ~ 27) 管脚。

输出信号从外设输出到 GPIO 交换矩阵，然后到达 IO MUX。IO MUX 必须设置相应管脚为 GPIO 功能，这样输出 GPIO 信号就能连接到相应管脚。

说明：

表 6-2 中输出索引号为 97 ~ 100 的外设信号没有连接至外设，可配置为从一个 GPIO 管脚输出后，直接由另一个 GPIO 管脚输入（索引号：97 ~ 100）。

6.5.2 功能描述

如图 6-1 所示，对于信号输出，99 个输出信号（即在表 6-2 中“输出信号”列的所有信号）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 GPIO 管脚。

输出外设信号 *Y* 到某一 GPIO 管脚 *X*[1] 的步骤如下：

1. 在 GPIO 交换矩阵中配置 GPIO 管脚 *X* 的 GPIO_FUNC*x*_OUT_SEL_CFG_REG 寄存器和 GPIO_ENABLE_REG[*x*] 寄存器。推荐使用相应 W1TS（写 1 置位）和 W1TC（写 1 清零）寄存器来更新 GPIO_ENABLE_REG 寄存器中的值：
   - 设置 GPIO_FUNC*x*_OUT_SEL_CFG_REG 寄存器的 GPIO_FUNC*x*_OUT_SEL 字段为外设输出信号 Y 的索引号 (Y)。
   - 要将信号强制使能为输出模式，需要将 GPIO 管脚 *X* 对应的 GPIO_FUNC*x*_OUT_SEL_CFG_REG 寄存器中 GPIO_FUNC*x*_OEN_SEL 字段置位；同时需要将 GPIO_ENABLE_W1TS_REG 中的相应位置位。或者，将 GPIO_FUNC*x*_OEN_SEL 清零，即选择采用外设的输出使能信号，此时输出使能信号

由内部逻辑功能决定。比如，表 6-2 中 “GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号” 一栏的 SPIQ_oe 信号。

- 置位 GPIO_ENABLE_W1TC_REG 中相应位可以关闭 GPIO 管脚的输出。

2. 要选择以开漏方式输出，可以设置 GPIO 管脚 *X* 的 GPIO_PIN*x*_REG 寄存器中 GPIO_PIN*x*_PAD_DRIVER 位。
3. 配置 IO MUX 寄存器来选择经由 GPIO 交换矩阵输出信号。配置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_REG 的过程如下：
   - 配置 GPIO 管脚 *X* 的 IO_MUX_GPIO*x*_MCU_SEL 为所需的管脚功能。此处选择数值 1，即功能 1（GPIO 功能），适用于所有管脚。
   - 设置 IO_MUX_GPIO*x*_FUN_DRV 字段为特定的输出强度值 (0 ~ 3)，值越大，输出驱动能力越强：
     - 0: ~5 mA
     - 1: ~10 mA
     - 2: ~20 mA（默认值）
     - 3: ~40 mA
   - 在开漏模式下，通过置位/清零 IO_MUX_GPIO*x*_FUN_WPU 和 IO_MUX_GPIO*x*_FUN_WPD 使能或关闭上拉/下拉电阻。

说明：

1. 某一个外设的输出信号可以同时从多个管脚输出；
2. 置位 GPIO_FUNC*x*_OUT_INV_SEL 可以把输出的信号取反。

6.5.3 简单 GPIO 输出

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

- 设置 GPIO 交换矩阵 GPIO_FUNC*n*_OUT_SEL 寄存器为特定的外设索引值 128 (0x80)；
- 设置 GPIO_OUT_REG 寄存器中相应位的值为期望 GPIO 输出的值。

说明：

- GPIO_OUT_REG[0] ~ GPIO_OUT_REG[27] 对应 GPIO0 ~ GPIO27，GPIO_OUT_REG[28] ~ GPIO_OUT_REG[31] 无效。
- 推荐使用 GPIO_OUT_W1TS/GPIO_OUT_W1TC 来置位/清零 GPIO_OUT_REG。

6.5.4 Sigma Delta 调制输出 (SDM)

6.5.4.1 功能描述

99 个外设输出信号中有四个信号（在表 6-2 中索引为：83 ~ 86）支持 1-bit 二阶 SDM 调制输出。上述四个信号通道默认输出使能。Sigma Delta 调制器可实现输出可配占空比的 PDM（脉冲密度调制）信号。二阶 SDM 调制的转换公式如下：

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

E(z) 为量化误差，X(z) 为输入。
Sigma Delta 调制器内部支持对 IO MUX 运行时钟的 1 ~ 256 倍分频：

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

分频后的时钟周期为调制器输出单位脉冲的周期。

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

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

PDM 信号占空比计算公式为：

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

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

6.5.4.2 配置方法

SDM 的配置方法如下：

- 将 SDM 输出经 GPIO 交换矩阵连接至相应管脚，见 6.5.2 章节；
- 置位 GPIO_EXT_FUNCTION_CLK_EN，使能 SDM 时钟；
- 配置 GPIO_EXT_SD*n*_PRESCALE 设置时钟分频系数；
- 配置 GPIO_EXT_SD*n*_IN 设置 SDM 输出信号的占空比。

6.6 IO MUX 的直接输入输出功能

6.6.1 概述

SPI、JTAG 等信号可以旁路 GPIO 交换矩阵以实现更好的高频数字特性。所以此类信号可以直接通过 IO MUX 输入和输出。

这样比使用 GPIO 交换矩阵的灵活度要低，即每个 GPIO 管脚的 IO MUX 寄存器只有较少的功能选择，但可以实现更好的高频数字特性。

6.6.2 功能描述

对于外设输入信号，旁路 GPIO 交换矩阵必须配置两个字段：

1. GPIO 管脚的 IO_MUX_GPIO*n*_MCU_SEL 必须设置为相应的管脚功能，章节 6.13 列出了管脚功能。

2. 清零 GPIO_SIG*n*_IN_SEL，直接将输入信号连接到外设。

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

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

6.7 GPIO 管脚的模拟功能

ESP32-H2 部分 GPIO 管脚具有模拟功能。用于模拟功能时，请确保已按照下述方法关闭了上拉电阻和下拉电阻：

- 设置 IO_MUX_GPIO*n*_MCU_SEL 为 1，同时清零 IO_MUX_GPIO*n*_FUN_IE、IO_MUX_GPIO*n*_FUN_WPU、IO_MUX_GPIO*n*_FUN_WPD；
- 置位 GPIO_ENABLE_W1TC*[n]*，清除输出使能。

表 6-4 列出了 ESP32-H2 管脚的模拟功能。

6.8 Light-sleep 模式管脚功能

当 ESP32-H2 处于 Light-sleep 模式时管脚可以有不同的功能。如果某一 GPIO 管脚的 IO_MUX_GPIO*n*_REG 寄存器中 IO_MUX_GPIO*n*_SLP_SEL 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管脚。

表 6-1. IO MUX Light-sleep 管脚功能控制寄存器

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

说明：
如果 IO_MUX_GPIO*n*_SLP_SEL 置为 0，则芯片在正常工作和 Light-sleep 模式下，管脚的功能一样。此时，具体的输出使能配置请参考 6.5.2 章节。

6.9 GPIO 管脚的 Hold 特性

每个 GPIO 管脚（包括 LP 管脚 GPIO8 ~ GPIO14）都有单独的 Hold 功能，由 LP 寄存器控制。管脚的 Hold 功能被置上后，管脚在置上 Hold 那一刻的状态被强制保持，无论内部信号如何变化、如何修改 IO MUX 配置或者 GPIO 配置，都不会改变管脚的状态。应用如果希望在看门狗超时触发内核复位或者 Deep-sleep 触发内核复位时管脚的状态不被改变，就需要提前把 Hold 置上。

具体配置如下：

- 数字管脚 (GPIO0 ~ GPIO5、GPIO22 ~ GPIO27)：
  - 若要在 Deep-sleep 中保持 GPIO*n* 管脚输入输出的状态值，需要在掉电之前将寄存器 LP_AON_GPIO_HOLD0_REG[*n*] 位置 1；在芯片被唤醒后，若要关闭 GPIO*n* 的 Hold 功能，可将寄存器 LP_AON_GPIO_HOLD0_REG[*n*] 位设置为 0。
  - 同时，也可将 PMU_TIE_HIGH_HP_PAD_HOLD_ALL 置位，用以保持所有数字管脚的状态值；在芯片被唤醒后，也可将 PMU_TIE_LOW_HP_PAD_HOLD_ALL 设置为 1，用以关闭所有数字管脚的 Hold 功能。
- LP 管脚 (GPIO8 ~ GPIO14)：
  - LP 管脚的输入输出值由 LP_AON_GPIO_HOLD0_REG[*n*]、PMU_TIE_HIGH_LP_PAD_HOLD_ALL 和 PMU_TIE_LOW_LP_PAD_HOLD_ALL 共同控制。用户可置位 LP_AON_GPIO_HOLD0_REG[*n*] 来保持 GPIO*n* 的状态值，可将 LP_AON_GPIO_HOLD0_REG[*n*] 设置为 0 以关闭 GPIO*n* 的 Hold 功能。
  - 同时，用户可置位 PMU_TIE_HIGH_LP_PAD_HOLD_ALL 来保持所有 LP 管脚的状态值，也可置位 PMU_TIE_LOW_LP_PAD_HOLD_ALL 关闭所有 LP 管脚的 Hold 功能。
  - 将 LP 管脚 Hold 为输入状态时，LP 管脚可以作为唤醒源将芯片从 Deep-sleep 中唤醒。

6.10 GPIO 管脚的迟滞特性

每个 GPIO 管脚都有迟滞功能。当未使能迟滞功能时，如图 6-6 所示，芯片焊盘（PAD）上输入给芯片内部的信号（C）的电平翻转只有一个阈值（Vt，约为 1.7 V），当 PAD 的电压高于 Vt 时，C 的电平为高，否则为低。但如果 PAD 上的信号有噪声时，它可能会影响 C 上的信号。

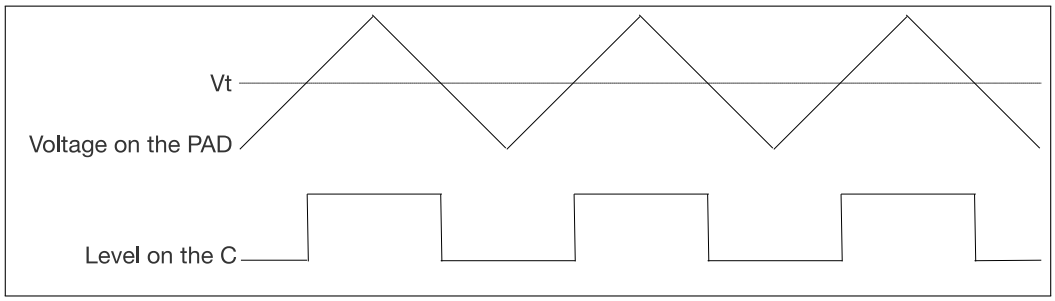


图 6-6. 未使能迟滞功能时芯片焊盘上的电平转换示例

当使能迟滞功能时，如图 6-7 所示，C 的电平翻转有两个阈值，分别为高电平阈值（Vth，约为 1.7 V）和低电平阈值（Vtl，约为 1.4 V）。当 PAD 的电压由低到高时，电压高于 Vth 时，C 的电平为高；当 PAD 的电压由高到低时，电压低于 Vtl 时，C 的电平为低；当 PAD 的电压在 Vth 和 Vtl 之间时，C 的电平不发生变化。迟滞功能能减小噪声影响，起到抗干扰的作用，同时能减小 C 的电平翻转时间。

参考如下配置使能迟滞功能：

- 当 IO_MUX_GPIO*n*_HYS_SEL 为 0 时（*n* 的取值范围是 0 ~ 5、8 ~ 14、22 ~ 27，分别对应 GPIO0 ~ GPIO5、GPIO8 ~ GPIO14、GPIO22 ~ GPIO27）：
  - 当 EFUSE_HYS_EN_PAD0[0:5] 的对应比特位为 1 时，则开启 GPIO0 ~ GPIO5 的迟滞功能；当 EFUSE_HYS_EN_PAD0[0:5] 的对应比特位为 0 时，则关闭 GPIO0 ~ GPIO5 的迟滞功能。

  - 当 EFUSE_HYS_EN_PAD1[2:8] 和 EFUSE_HYS_EN_PAD1[16:21] 的对应比特位为 1 时，则开启 GPIO8 ~ GPIO14 和 GPIO22 ~ GPIO27 的迟滞功能；当 EFUSE_HYS_EN_PAD1[2:8] 和 EFUSE_HYS_EN_PAD1[16:21] 的对应比特位设置为 0 时，则关闭 GPIO8 ~ GPIO14 和 GPIO22 ~ GPIO27 的迟滞功能。

- 当 IO_MUX_GPIO*n*_HYS_SEL 为 1 时（*n* 的取值范围是 0 ~ 5、8 ~ 14、22 ~ 27，分别对应 GPIO0 ~ GPIO5、GPIO8 ~ GPIO14、GPIO22 ~ GPIO27）：
  - 设置 IO_MUX_GPIO*n*_HYS_EN 为 1，可开启 GPIO*n* 的迟滞功能
  - 设置 IO_MUX_GPIO*n*_HYS_EN 为 0，可关闭 GPIO*n* 的迟滞功能

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

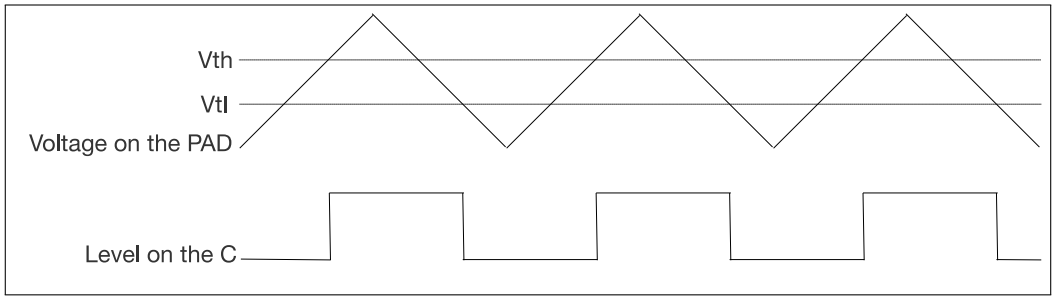


图 6-7. 使能迟滞功能时芯片焊盘上的电平转换示例

6.11 GPIO 管脚供电和电源管理

6.11.1 GPIO 管脚供电

GPIO 管脚供电请参考《*ESP32-H2 规格书*》中管脚定义章节。所有管脚均可用于将芯片从 Light-sleep 中唤醒，但仅有 LP 管脚 (GPIO8 ~ GPIO14) 可用于将芯片从 Deep-sleep 唤醒。

6.11.2 电源管理

ESP32-H2 的管脚可分为如下三种不同的电源域。

- VDDPST1：部分数字 GPIO 和部分 LP GPIO 的输入电源
- VDDPST2：部分数字 GPIO 的输入电源
- VDDA_PMU/VBAT：GPIO 管脚中 GPIO12、XTAL_32K_P 和 XTAL_32K_N 的输入电源

6.12 外设信号列表

表 6-2 列出了所有经由 GPIO 交换矩阵的外设输入输出信号。

请注意 GPIO_FUNC*n*_OEN_SEL 位的配置：

- GPIO_FUNC*n*_OEN_SEL = 1，则寄存器 GPIO_ENABLE_REG 中的相应位 *n* 将用于控制信号输出使能。
  - GPIO_ENABLE_REG = 0：输出关闭；
  - GPIO_ENABLE_REG = 1：输出使能；

- GPIO_FUNC*n*_OEN_SEL = 0，则输出信号的使能由外设控制，例如表 6-2 中 “GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号” 一栏的 SPIQ_oe。注意，使能信号 SPIQ_oe 可设置为 1 (1’d1) 或 0 (1’d0)，具体由外设的配置决定。如果 “GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号” 一栏中为 1’d1，则表示 GPIO_FUNC*n*_OEN_SEL 已清零，输出信号默认始终使能。

说明：

信号连续编号，但并非所有信号均有效。

- 表 6-2 “输入信号” 一栏中有名字的信号均为有效输入信号；
- 表 6-2 “输出信号” 一栏中有名字的信号均为有效输出信号。

表 6-2. GPIO 交换矩阵外设信号

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNCn_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
0	ext_adc_start	0	no	ledc_ls_sig_out0	1'd1	no
1	-	-	-	ledc_ls_sig_out1	1'd1	no
2	-	-	-	ledc_ls_sig_out2	1'd1	no
3	-	-	-	ledc_ls_sig_out3	1'd1	no
4	-	-	-	ledc_ls_sig_out4	1'd1	no
5	-	-	-	ledc_ls_sig_out5	1'd1	no
6	U0RXD_in	0	yes	U0TXD_out	1'd1	yes
7	U0CTS_in	0	no	U0RTS_out	1'd1	no
8	U0DSR_in	0	no	U0DTR_out	1'd1	no
9	U1RXD_in	1	no	U1TXD_out	1'd1	no
10	U1CTS_in	0	no	U1RTS_out	1'd1	no
11	U1DSR_in	0	no	U1DTR_out	1'd1	no
12	I2S_MCLK_in	0	no	I2S_MCLK_out	1'd1	no
13	I2SO_BCK_in	0	no	I2SO_BCK_out	1'd1	no
14	I2SO_WS_in	0	no	I2SO_WS_out	1'd1	no
15	I2SI_SD_in	0	no	I2SO_SD_out	1'd1	no
16	I2SI_BCK_in	0	no	I2SI_BCK_out	1'd1	no
17	I2SI_WS_in	0	no	I2SI_WS_out	1'd1	no
18	-	-	-	I2SO_SD1_out	1'd1	no
19	usb_jtag_tdo_bridge	0	no	usb_jtag_trst	1'd1	no
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-
24	-	-	-	-	-	-
25	-	-	-	-	-	-

GoBack

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
26	-	-	-	-	-	-
27	-	-	-	-	-	-
28	cpu_gpio_in0	0	no	cpu_gpio_out0	cpu_gpio_out_oen0	no
29	cpu_gpio_in1	0	no	cpu_gpio_out1	cpu_gpio_out_oen1	no
30	cpu_gpio_in2	0	no	cpu_gpio_out2	cpu_gpio_out_oen2	no
31	cpu_gpio_in3	0	no	cpu_gpio_out3	cpu_gpio_out_oen3	no
32	cpu_gpio_in4	0	no	cpu_gpio_out4	cpu_gpio_out_oen4	no
33	cpu_gpio_in5	0	no	cpu_gpio_out5	cpu_gpio_out_oen5	no
34	cpu_gpio_in6	0	no	cpu_gpio_out6	cpu_gpio_out_oen6	no
35	cpu_gpio_in7	0	no	cpu_gpio_out7	cpu_gpio_out_oen7	no
36	-	-	-	-	-	-
37	-	-	-	-	-	-
38	-	-	-	-	-	-
39	-	-	-	-	-	-
40	-	-	-	-	-	-
41	-	-	-	-	-	-
42	-	-	-	-	-	-
43	-	-	-	-	-	-
44	-	-	-	-	-	-
45	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
46	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
47	parl_rx_data0	0	no	parl_tx_data0	1'd1	no
48	parl_rx_data1	0	no	parl_tx_data1	1'd1	no
49	parl_rx_data2	0	no	parl_tx_data2	1'd1	no
50	parl_rx_data3	0	no	parl_tx_data3	1'd1	no
51	parl_rx_data4	0	no	parl_tx_data4	1'd1	no
52	parl_rx_data5	0	no	parl_tx_data5	1'd1	no
53	parl_rx_data6	0	no	parl_tx_data6	1'd1	no

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
54	parl_rx_data7	0	no	parl_tx_data7	1'd1	no
55	I2CEXT1_SCL_in	1	no	I2CEXT1_SCL_out	I2CEXT1_SCL_oe	no
56	I2CEXT1_SDA_in	1	no	I2CEXT1_SDA_out	I2CEXT1_SDA_oe	no
57		-	-	cte_ant0	1'd1	no
58		-	-	cte_ant1	1'd1	no
59		-	-	cte_ant2	1'd1	no
60		-	-	cte_ant3	1'd1	no
61		-	-	cte_ant4	1'd1	no
62		-	-	cte_ant5	1'd1	no
63	FSPICLK_in	0	yes	FSPICLK_out_mux	FSPICLK_oe	yes
64	FSPIQ_in	0	yes	FSPIQ_out	FSPIQ_oe	yes
65	FSPID_in	0	yes	FSPID_out	FSPID_oe	yes
66	FSPIHD_in	0	yes	FSPIHD_out	FSPIHD_oe	yes
67	FSPIWP_in	0	yes	FSPIWP_out	FSPIWP_oe	yes
68	FSPICS0_in	0	yes	FSPICS0_out	FSPICS0_oe	yes
69	parl_rx_clk_in	0	no	parl_rx_clk_out	1'd1	no
70	parl_tx_clk_in	0	no	parl_tx_clk_out	1'd1	no
71	rmt_sig_in0	0	no	rmt_sig_out0	1'd1	no
72	rmt_sig_in1	0	no	rmt_sig_out1	1'd1	no
73	twai0_rx	1	no	twai0_tx	1'd1	no
74	-	-	-	twai0_bus_off_on	1'd1	no
75	-	-	-	twai0_clkout	1'd1	no
76	-	-	-	twai0_standby	1'd1	no
77		-	-	cte_ant6	1'd1	no
78	-	-	-	cte_ant7	1'd1	no
79	-	-	-	cte_ant8	1'd1	no
80	-	-	-	cte_ant9	1'd1	no
81	-	-	-	-	-	-

GoBack

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
82	-	-	-	-	-	-
83	-	-	-	gpio_sd0_out	1'd1	no
84	-	-	-	gpio_sd1_out	1'd1	no
85	-	-	-	gpio_sd2_out	1'd1	no
86	-	-	-	gpio_sd3_out	1'd1	no
87	pwm0_sync0_in	0	no	pwm0_out0a	1'd1	no
88	pwm0_sync1_in	0	no	pwm0_out0b	1'd1	no
89	pwm0_sync2_in	0	no	pwm0_out1a	1'd1	no
90	pwm0_f0_in	0	no	pwm0_out1b	1'd1	no
91	pwm0_f1_in	0	no	pwm0_out2a	1'd1	no
92	pwm0_f2_in	0	no	pwm0_out2b	1'd1	no
93	pwm0_cap0_in	0	no	-	-	-
94	pwm0_cap1_in	0	no	-	-	-
95	pwm0_cap2_in	0	no	-	-	-
96	-	-	-	-	-	-
97	sig_in_func_97	0	no	sig_in_func97	1'd1	no
98	sig_in_func_98	0	no	sig_in_func98	1'd1	no
99	sig_in_func_99	0	no	sig_in_func99	1'd1	no
100	sig_in_func_100	0	no	sig_in_func100	1'd1	no
101	pcnt_sig_ch0_in0	0	no	FSPICS1_out	FSPICS1_oe	yes
102	pcnt_sig_ch1_in0	0	no	FSPICS2_out	FSPICS2_oe	yes
103	pcnt_ctrl_ch0_in0	0	no	FSPICS3_out	FSPICS3_oe	yes
104	pcnt_ctrl_ch1_in0	0	no	FSPICS4_out	FSPICS4_oe	yes
105	pcnt_sig_ch0_in1	0	no	FSPICS5_out	FSPICS5_oe	yes
106	pcnt_sig_ch1_in1	0	no	cte_ant10	1'd1	no
107	pcnt_ctrl_ch0_in1	0	no	cte_ant11	1'd1	no
108	pcnt_ctrl_ch1_in1	0	no	cte_ant12	1'd1	no
109	pcnt_sig_ch0_in2	0	no	cte_ant13	1'd1	no

GoBack

信号索引	输入信号	默认值	信号可经由 IO MUX 直接输入	输出信号	GPIO_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
110	pcnt_sig_ch1_in2	0	no	cte_ant14	1'd1	no
111	pcnt_ctrl_ch0_in2	0	no	cte_ant15	1'd1	no
112	pcnt_ctrl_ch1_in2	0	no	-	-	-
113	pcnt_sig_ch0_in3	0	no	-	-	-
114	pcnt_sig_ch1_in3	0	no	SPICLK_out_mux	SPICLK_oe	yes
115	pcnt_ctrl_ch0_in3	0	no	SPICS0_out	SPICS0_oe	yes
116	pcnt_ctrl_ch1_in3	0	no	SPICS1_out	SPICS1_oe	no
117	-	-	-	-	-	-
118	-	-	-	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe	yes
122	SPID_in	0	yes	SPID_out	SPID_oe	yes
123	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe	yes
124	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe	yes
125	-	-	-	CLK_OUT_out1	1'd1	no
126	-	-	-	CLK_OUT_out2	1'd1	no
127	-	-	-	CLK_OUT_out3	1'd1	no

6.13 IO MUX 管脚功能列表

表 6-3 列出了所有 GPIO 管脚的 IO MUX 功能。

表 6-3. IO MUX 管脚功能

GPIO	管脚名称	功能 0	功能 1	功能 2	功能 3	驱动强度	复位	说明
0	GPIO0	GPIO0	GPIO0	FSPIQ	—	2	0	—
1	GPIO1	GPIO1	GPIO1	FSPICS0	—	2	0	—
2	MTMS	MTMS	GPIO2	FSPIWP	—	2	1	—
3	MTDO	MTDO	GPIO3	FSPIHD	—	2	1	—
4	MTCK	MTCK	GPIO4	FSPICLK	—	2	1*	—
5	MTDI	MTDI	GPIO5	FSPID	—	2	1	—
8	GPIO8	GPIO8	GPIO8	—	—	2	1	R
9	GPIO9	GPIO9	GPIO9	—	—	2	3	R
10	GPIO10	GPIO10	GPIO10	—	—	2	0	R
11	GPIO11	GPIO11	GPIO11	—	—	2	0	R
12	GPIO12	GPIO12	GPIO12	—	—	2	0	R
13	XTAL_32K_P	GPIO13	GPIO13	—	—	2	0	R
14	XTAL_32K_N	GPIO14	GPIO14	—	—	2	0	R
22	GPIO22	GPIO22	GPIO22	—	—	2	0	—
23	U0RXD	U0RXD	GPIO23	FSPICS1	—	2	3	—
24	U0TXD	U0TXD	GPIO24	FSPICS2	—	2	4	—
25	GPIO25	GPIO25	GPIO25	FSPICS3	—	2	1	—
26	GPIO26	GPIO26	GPIO26	FSPICS4	—	3	1	USB
27	GPIO27	GPIO27	GPIO27	FSPICS5	—	3	3*	USB

驱动强度

“驱动强度” 一栏所示为每个管脚复位后的默认驱动强度。

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

复位

“复位” 一栏所示为每个管脚复位后的默认配置。

- **0** - IE = 0（输入关闭）
- **1** - IE = 1（输入使能）
- **2** - IE = 1，WPD = 1（输入使能，下拉电阻使能）
- **3** - IE = 1，WPU = 1 （输入使能，上拉电阻使能）
- **4** - OE = 1，WPU = 1 （输出使能，上拉电阻使能）

- **1*** - 如果 EFUSE_DIS_PAD_JTAG = 1，则 MTCK 管脚复位后浮空，即 IE = 1。如果 EFUSE_DIS_PAD_JTAG = 0，则 MTCK 管脚连接内部上拉电阻，即 IE = 1，WPU = 1。
- **3*** - IE = 1，WPU = 0，GPIO27 的 USB 上拉默认值为 1，因此，其上拉电阻使能，具体见以下**说明**。

GPIO 输入模式

GPIO 的输入功能可配置为迟滞或普通模式：

- 迟滞模式：在迟滞模式下，GPIO 输入高低电平的切换阈值与电平切换方向有关。具体来说，从高电平切换到低电平的电压阈值略低于从低电平切换到高电平的电压阈值，详见章节 6.10。
- 普通模式：不使能 GPIO 管脚的迟滞功能，GPIO 输入的高低电平与电平切换方向无关，即从高电平切换至低电平和从低电平切换至高电平的电压阈值一致。

说明

- **R** - LP 管脚，部分具有模拟功能，见表 6-4。
- **USB** - USB 上拉电阻使能
  - USB 管脚（GPIO26 和 GPIO27）默认开启 USB 功能，此时管脚是否上拉由 USB 是否上拉决定。USB 上拉由 USB_SERIAL_JTAG_DP/DM_PULLUP 控制，USB 上拉电阻的具体阻值可通过 USB_SERIAL_JTAG_PULLUP_VALUE 位控制，详见《ESP32-H2 技术参考手册》> 章节 *USB 串口/JTAG 控制器*。
  - USB 管脚关闭 USB 功能时，用作普通 GPIO，默认禁用管脚内部弱上/下拉电阻，可通过 IO_MUX_GPIO*n*_MCU_WPU/WPD 配置。

6.14 IO MUX 管脚模拟功能列表

表 6-4 列出了具有模拟功能的 IO MUX 管脚。

表 6-4. IO MUX 管脚的模拟功能

GPIO No.[1]	管脚名称	模拟功能 0	模拟功能 1
1	GPIO1	-	ADC1_CH0
2	MTMS	-	ADC1_CH1
3	MTDO	-	ADC1_CH2
4	MTCK	-	ADC1_CH3
5	MTDI	-	ADC1_CH4
10	GPIO10	ZCD0[1]	-
11	GPIO11	ZCD1[1]	-
13	XTAL_32K_P	XTAL_32K_P	-
14	XTAL_32K_N	XTAL_32K_N	-
26	GPIO26	USB_D-	-
27	GPIO27	USB_D+	-

[1] ZCD0 和 ZCD1 是模拟 PAD 电压比较功能，详情见小节 6.15。

6.15 模拟 PAD 电压比较功能

GPIO10 和 GPIO11 PAD 具有模拟 PAD 电压比较功能，GPIO_EXT_XPD_COMP 设置为 1 以使能该功能。使能模拟 PAD 电压比较功能后，当 GPIO11 PAD 上的电压高于参考电压时，表示比较结果的 PAD_COMP_OUT 信号为高电平，否则为低电平。

设置 GPIO_EXT_MODE_COMP 的值为：

- 0：参考电压为（GPIO_EXT_DREF_COMP * VDDPST2）/ 10。
- 1：参考电压为 GPIO10 PAD 上的电压。

PAD_COMP_OUT 信号会同步至 IO MUX 的运行时钟，得到 PAD_COMP_OUT_sync 信号，从而产生中断源 GPIO_EXT_PAD_COMP_INT。

设置 GPIO_EXT_ZERO_DET_MODE 的值为：

- 0：关闭中断源产生。
- 1，2：保留位。
- 3：使能中断源为 PAD_COMP_OUT_sync 信号的任意沿。

同时，产生一次中断源后，将在 GPIO_EXT_ZERO_DET_FILTER_CNT 个 IO MUX 运行时钟周期内屏蔽新的中断源。

6.16 事件任务矩阵功能

在 ESP32-H2 中，GPIO 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 GPIO 的 ETM 任务，或者通过 GPIO 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与 GPIO 相关的 ETM 任务和 ETM 事件。

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

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

将任务通道 *x* 配置到 GPIO*y* 的示例如下：

- 配置 IO_MUX_GPIO*y*_MCU_SEL 的值为 1，使得 GPIO*y* 选择表 6-3 所示的 IO MUX 管脚功能 1；
- 配置 GPIO_ENABLE_REG[*y*] 的值为 1；
- 配置 GPIO_EXT_ETM_TASK_GPIO*y*_SEL 为 *x*；
- 置位 GPIO_EXT_ETM_TASK_GPIO*y*_EN，使能 GPIO*y* 受到 ETM 任务通道 *x* 控制。

说明：

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

GPIO 有八个事件通道，每个事件通道可产生的 ETM 事件有：

- GPIO_EVT_CH*x*_RISE_EDGE，表示对应 GPIO 滤波器（见图 6-1）输出信号出现上升沿
- GPIO_EVT_CH*x*_FALL_EDGE，表示对应 GPIO 滤波器（见图 6-1）输出信号出现下降沿
- GPIO_EVT_CH*x*_ANY_EDGE，表示对应 GPIO 滤波器（见图 6-1）输出信号发生翻转

事件通道具体配置如下：

- 置位 GPIO_EXT_ETM_CH*x*_EVENT_EN 使能事件通道 *x* (0 ~ 7)；
- 配置 GPIO_EXT_ETM_CH*x*_EVENT_SEL 为 *y* (0 ~ 5、8 ~ 14、22 ~ 27)，即选择 19 个 GPIO 中的一个 GPIO。

说明：
一个 GPIO 可被一个或多个事件通道选择。

在具体应用中，GPIO 的 ETM 事件可以用来触发 GPIO 的 ETM 任务，例如，事件通道 0 选择 GPIO0，GPIO1 选择任务通道 0，并且 GPIO_EVT_CH*0*_RISE_EDGE 事件用于触发 GPIO_TASK_CH*0*_TOGGLE 任务，当通过 GPIO0 向芯片输入一个方波信号时，芯片通过 GPIO1 输出一个二分频的方波信号。

6.17 寄存器列表

6.17.1 GPIO 交换矩阵寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

ESP32-H2 可配置使用 19 个 GPIO 管脚，即 GPIO0 ~ GPIO5、GPIO8 ~ GPIO14、GPIO22 ~ GPIO27 管脚，因此：

- **配置寄存器**只可以配置 GPIO0 ~ GPIO5、GPIO8 ~ GPIO14 和 GPIO22 ~ GPIO27；
- **管脚配置寄存器**只可以使用 GPIO_PIN0_REG ~ GPIO_PIN5_REG、GPIO_PIN8_REG ~ GPIO_PIN14_REG 和 GPIO_PIN22_REG ~ GPIO_PIN27_REG 寄存器；
- **输入配置寄存器**只可以配置选择 GPIO0 ~ GPIO5、GPIO8 ~ GPIO14 和 GPIO22 ~ GPIO27；
- **输出配置寄存器**只可以使用 GPIO_FUNC0_OUT_SEL_CFG_REG ~ GPIO_FUNC5_OUT_SEL_CFG_REG、GPIO_FUNC8_OUT_SEL_CFG_REG ~ GPIO_FUNC14_OUT_SEL_CFG_REG 和 GPIO_PIN22_OUT_SEL_CFG_REG ~ GPIO_PIN27_OUT_SEL_CFG_REG 寄存器。

名称	描述	地址	访问
配置寄存器			
GPIO_OUT_REG	GPIO 输出寄存器	0x0004	R/ W/ SC/ WTC
GPIO_OUT_W1TS_REG	GPIO 输出置位寄存器	0x0008	WT
GPIO_OUT_W1TC_REG	GPIO 输出清除寄存器	0x000C	WT
GPIO_ENABLE_REG	GPIO 输出使能寄存器	0x0020	R/W/WTC
GPIO_ENABLE_W1TS_REG	GPIO 输出使能置位寄存器	0x0024	WT
GPIO_ENABLE_W1TC_REG	GPIO 输出使能清除寄存器	0x0028	WT
GPIO_STRAP_REG	Strapping 管脚寄存器	0x0038	RO
GPIO_IN_REG	GPIO 输入寄存器	0x003C	RO
中断状态寄存器			
GPIO_STATUS_REG	GPIO 中断状态寄存器	0x0044	R/W/WTC
GPIO_STATUS_W1TS_REG	GPIO 中断状态置位寄存器	0x0048	WT
GPIO_STATUS_W1TC_REG	GPIO 中断状态清除寄存器	0x004C	WT
GPIO_PCPU_INT_REG	GPIO 的 CPU 中断状态寄存器	0x005C	RO
GPIO_STATUS_NEXT_REG	GPIO 中断源寄存器	0x014C	RO
管脚配置寄存器			
GPIO_PIN0_REG	GPIO0 管脚配置寄存器	0x0074	R/W
GPIO_PIN1_REG	GPIO1 管脚配置寄存器	0x0078	R/W
GPIO_PIN2_REG	GPIO2 管脚配置寄存器	0x007C	R/W
...	...	...	...
GPIO_PIN25_REG	GPIO25 管脚配置寄存器	0x00D8	R/W
GPIO_PIN26_REG	GPIO26 管脚配置寄存器	0x00DC	R/W
GPIO_PIN27_REG	GPIO27 管脚配置寄存器	0x00E0	R/W
输入配置寄存器			
GPIO_FUNC0_IN_SEL_CFG_REG	输入信号 0 的配置寄存器	0x0154	R/W

名称	描述	地址	访问
GPIO_FUNC1_IN_SEL_CFG_REG	输入信号 1 的配置寄存器	0x0158	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	输入信号 2 的配置寄存器	0x015C	R/W
...	...	...	...
GPIO_FUNC125_IN_SEL_CFG_REG	输入信号 125 的配置寄存器	0x0348	R/W
GPIO_FUNC126_IN_SEL_CFG_REG	输入信号 126 的配置寄存器	0x034C	R/W
GPIO_FUNC127_IN_SEL_CFG_REG	输入信号 127 的配置寄存器	0x0350	R/W
输出配置寄存器			
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 输出配置寄存器	0x0554	varies
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 输出配置寄存器	0x0558	varies
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 输出配置寄存器	0x055C	varies
...	...	...	...
GPIO_FUNC25_OUT_SEL_CFG_REG	GPIO25 输出配置寄存器	0x05B8	varies
GPIO_FUNC26_OUT_SEL_CFG_REG	GPIO26 输出配置寄存器	0x05BC	varies
GPIO_FUNC27_OUT_SEL_CFG_REG	GPIO27 输出配置寄存器	0x05C0	varies
版本寄存器			
GPIO_DATE_REG	版本控制寄存器	0x06FC	R/W
时钟门控寄存器			
GPIO_CLOCK_GATE_REG	GPIO 时钟门控寄存器	0x062C	R/W

6.17.2 IO MUX 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

ESP32-H2 可配置使用 19 个 GPIO，即 GPIO0 ~ GPIO5、GPIO8 ~ GPIO14 和 GPIO22 ~ GPIO27 管脚，因此**配置寄存器**不可以配置 IO_MUX_GPIO6_REG ~ IO_MUX_GPIO7_REG 寄存器和 IO_MUX_GPIO15_REG ~ IO_MUX_GPIO21_REG 寄存器。

名称	描述	地址	访问
配置寄存器			
IO_MUX_PIN_CTRL_REG	时钟输出配置寄存器	0x0000	R/W
IO_MUX_GPIO0_REG	GPIO0 的 IO MUX 配置寄存器	0x0004	R/W
IO_MUX_GPIO1_REG	GPIO1 的 IO MUX 配置寄存器	0x0008	R/W
IO_MUX_GPIO2_REG	GPIO2 的 IO MUX 配置寄存器	0x000C	R/W
...	...	...	...
IO_MUX_GPIO25_REG	GPIO25 的 IO MUX 配置寄存器	0x0068	R/W
IO_MUX_GPIO26_REG	GPIO26 的 IO MUX 配置寄存器	0x006C	R/W
IO_MUX_GPIO27_REG	GPIO27 的 IO MUX 配置寄存器	0x0070	R/W
版本寄存器			
IO_MUX_DATE_REG	版本控制寄存器	0x00FC	R/W

6.17.3 GPIO_EXT 寄存器列表

GPIO_EXT 寄存器包括 SDM 寄存器、毛刺滤波器寄存器、过零检测寄存器和 ETM 寄存器。

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
SDM 配置寄存器			
GPIO_EXT_SIGMADELTA0_REG	SDM 通道 0 的占空比配置寄存器	0x0000	R/W
GPIO_EXT_SIGMADELTA1_REG	SDM 通道 1 的占空比配置寄存器	0x0004	R/W
GPIO_EXT_SIGMADELTA2_REG	SDM 通道 2 的占空比配置寄存器	0x0008	R/W
GPIO_EXT_SIGMADELTA3_REG	SDM 通道 3 的占空比配置寄存器	0x000C	R/W
GPIO_EXT_SIGMADELTA_MISC_REG	MISC 寄存器	0x0024	R/W
毛刺滤波器配置寄存器			
GPIO_EXT_GLITCH_FILTER_CH0_REG	毛刺滤波器通道 0 的配置寄存器	0x0030	R/W
GPIO_EXT_GLITCH_FILTER_CH1_REG	毛刺滤波器通道 1 的配置寄存器	0x0034	R/W
GPIO_EXT_GLITCH_FILTER_CH2_REG	毛刺滤波器通道 2 的配置寄存器	0x0038	R/W
GPIO_EXT_GLITCH_FILTER_CH3_REG	毛刺滤波器通道 3 的配置寄存器	0x003C	R/W
GPIO_EXT_GLITCH_FILTER_CH4_REG	毛刺滤波器通道 4 的配置寄存器	0x0040	R/W
GPIO_EXT_GLITCH_FILTER_CH5_REG	毛刺滤波器通道 5 的配置寄存器	0x0044	R/W
GPIO_EXT_GLITCH_FILTER_CH6_REG	毛刺滤波器通道 6 的配置寄存器	0x0048	R/W
GPIO_EXT_GLITCH_FILTER_CH7_REG	毛刺滤波器通道 7 的配置寄存器	0x004C	R/W
ETM 配置寄存器			
GPIO_EXT_ETM_EVENT_CH0_CFG_REG	ETM 通道 0 的配置寄存器	0x0060	R/W
GPIO_EXT_ETM_EVENT_CH1_CFG_REG	ETM 通道 1 的配置寄存器	0x0064	R/W
GPIO_EXT_ETM_EVENT_CH2_CFG_REG	ETM 通道 2 的配置寄存器	0x0068	R/W
GPIO_EXT_ETM_EVENT_CH3_CFG_REG	ETM 通道 3 的配置寄存器	0x006C	R/W
GPIO_EXT_ETM_EVENT_CH4_CFG_REG	ETM 通道 4 的配置寄存器	0x0070	R/W
GPIO_EXT_ETM_EVENT_CH5_CFG_REG	ETM 通道 5 的配置寄存器	0x0074	R/W
GPIO_EXT_ETM_EVENT_CH6_CFG_REG	ETM 通道 6 的配置寄存器	0x0078	R/W
GPIO_EXT_ETM_EVENT_CH7_CFG_REG	ETM 通道 7 的配置寄存器	0x007C	R/W
GPIO_EXT_ETM_TASK_P0_CFG_REG	ETM 的 GPIO 选择配置寄存器 0	0x00A0	R/W
GPIO_EXT_ETM_TASK_P1_CFG_REG	ETM 的 GPIO 选择配置寄存器 1	0x00A4	R/W
GPIO_EXT_ETM_TASK_P2_CFG_REG	ETM 的 GPIO 选择配置寄存器 2	0x00A8	R/W
GPIO_EXT_ETM_TASK_P3_CFG_REG	ETM 的 GPIO 选择配置寄存器 3	0x00AC	R/W
GPIO_EXT_ETM_TASK_P4_CFG_REG	ETM 的 GPIO 选择配置寄存器 4	0x00B0	R/W
GPIO_EXT_ETM_TASK_P5_CFG_REG	ETM 的 GPIO 选择配置寄存器 5	0x00B4	R/W
GPIO_EXT_ETM_TASK_P6_CFG_REG	ETM 的 GPIO 选择配置寄存器 6	0x00B8	R/W
过零检测配置寄存器			
GPIO_EXT_PAD_COMP_CONFIG_REG	过零检测的配置寄存器	0x0028	R/W
GPIO_EXT_PAD_COMP_FILTER_REG	过零检测的中断源屏蔽时间配置寄存器	0x002C	R/W
中断状态寄存器			
GPIO_EXT_INT_RAW_REG	原始中断寄存器	0x00E0	varies
GPIO_EXT_INT_ST_REG	中断使能寄存器	0x00E4	RO
GPIO_EXT_INT_ENA_REG	中断状态寄存器	0x00E8	R/W
GPIO_EXT_INT_CLR_REG	中断清除寄存器	0x00EC	WT
版本寄存器			

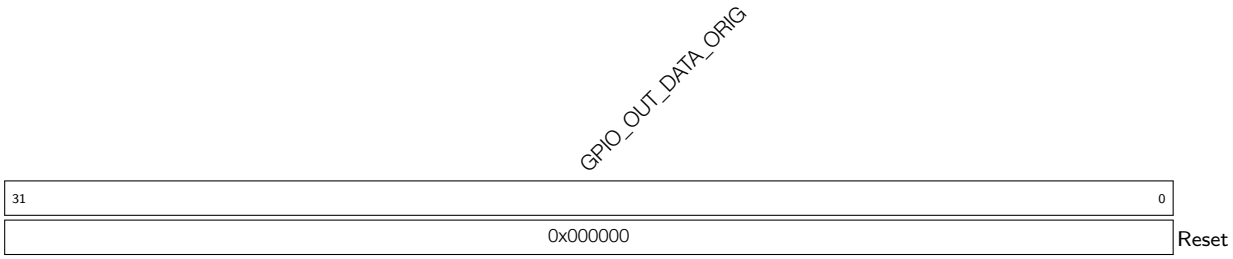
名称	描述	地址	访问
GPIO_EXT_VERSION_REG	版本控制寄存器	0x00FC	R/W

6.18 寄存器

6.18.1 GPIO 交换矩阵寄存器

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

Register 6.1. GPIO_OUT_REG (0x0004)



GPIO_OUT_DATA_ORIG 配置简单 GPIO 输出模式下 GPIO0 ~ GPIO27 的输出值。

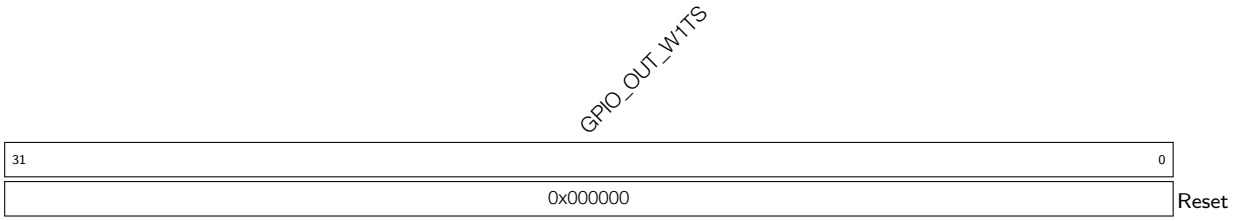
0：低电平

1：高电平

bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27 的输出值。bit28 ~ bit31 无效。

(R/W/SC/WTC)

Register 6.2. GPIO_OUT_W1TS_REG (0x0008)



GPIO_OUT_W1TS 配置是否置位 GPIO0 ~ GPIO27 输出寄存器 GPIO_OUT_REG。

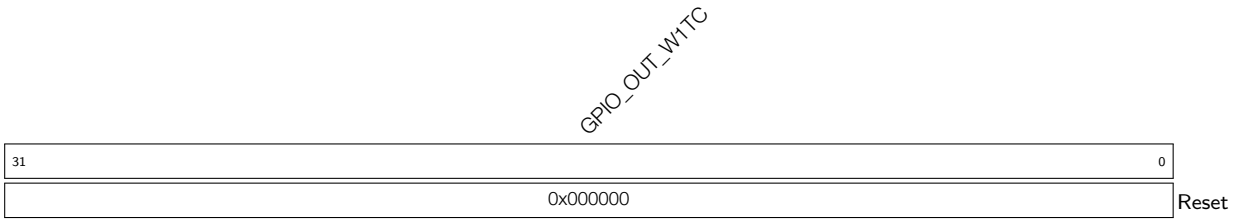
0：不置位

1：GPIO_OUT_REG 中相应位也将置 1

bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。推荐使用此寄存器来置位 GPIO_OUT_REG。

(WT)

Register 6.3. GPIO_OUT_W1TC_REG (0x000C)



GPIO_OUT_W1TC 配置是否清除 GPIO0 ~ GPIO27 输出寄存器 GPIO_OUT_REG。

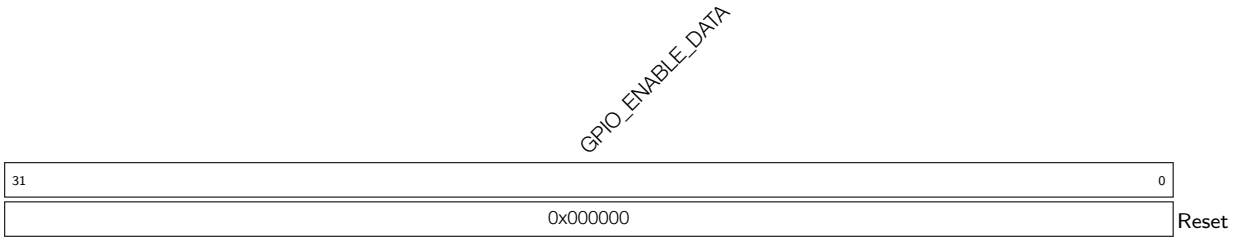
0：不清除

1：GPIO_OUT_REG 中相应位将被清除

bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。推荐使用该寄存器来清除 GPIO_OUT_REG。

(WT)

Register 6.4. GPIO_ENABLE_REG (0x0020)



GPIO_ENABLE_DATA 配置是否使能 GPIO0 ~ GPIO27 输出。

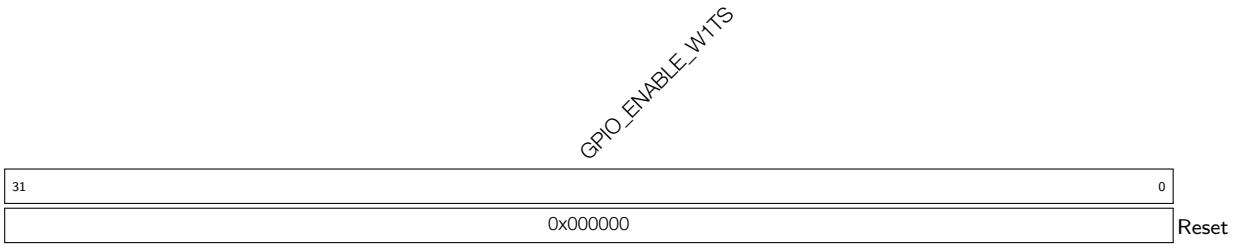
0：不使能

1：使能

bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。

(R/W/WTC)

Register 6.5. GPIO_ENABLE_W1TS_REG (0x0024)



GPIO_ENABLE_W1TS 配置是否使能 GPIO0 ~ GPIO27 的输出使能寄存器 GPIO_ENABLE_REG。

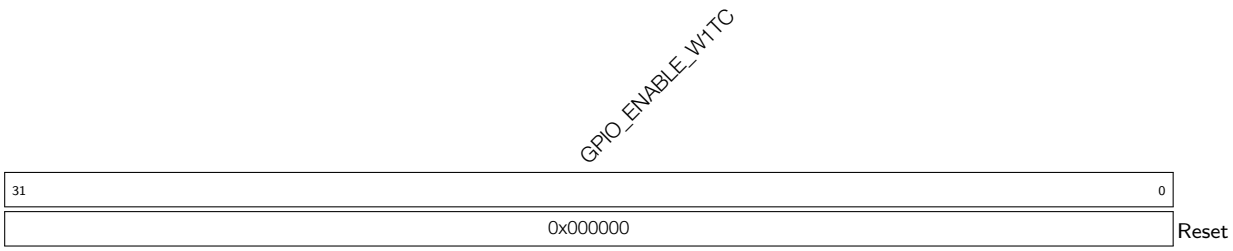
0：不使能

1：GPIO_ENABLE_REG 中相应位也将置 1

bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。推荐使用该寄存器来置位 GPIO_ENABLE_REG。

(WT)

Register 6.6. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC 配置是否清除 GPIO0 ~ GPIO27 输出使能寄存器 GPIO_ENABLE_REG。

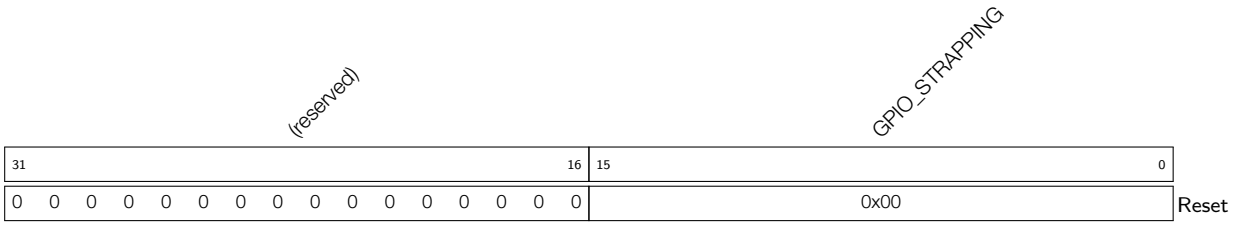
0：不清除

1：GPIO_ENABLE_REG 中相应位将被清除

bit0 ~ bit27 分别对应 GPIO0 ~ 27。bit28 ~ bit31 无效。推荐使用此寄存器清除 GPIO_ENABLE_REG。

(WT)

Register 6.7. GPIO_STRAP_REG (0x0038)

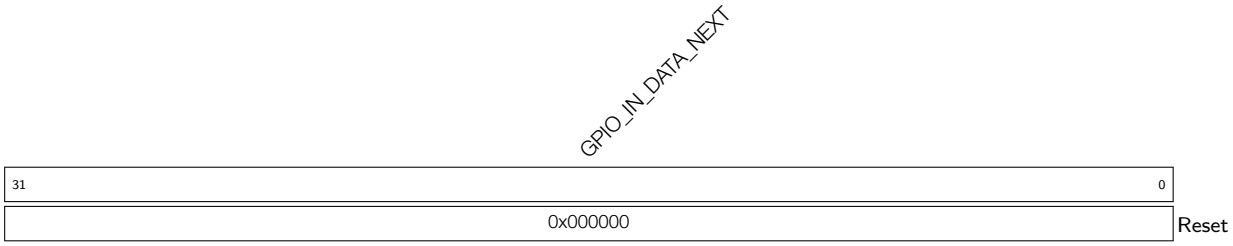


GPIO_STRAPPING 表示 GPIO Strapping 管脚的值。

- bit0：表示 GPIO2 管脚的值（仅在使用 SPI Download Boot 模式时，GPIO2 才用作 Strapping 管脚）
- bit1：表示 GPIO3 管脚的值（仅在使用 SPI Download Boot 模式时，GPIO3 才用作 Strapping 管脚）
- bit2：表示 GPIO8 管脚的值（该值同时会受到 EFUSE_DIS_FORCE_DOWNLOAD 和 LP_AON_FORCE_DOWNLOAD_BOOT 的影响）
- bit3：表示 GPIO9 管脚的值（该值同时会受到 EFUSE_DIS_FORCE_DOWNLOAD 和 LP_AON_FORCE_DOWNLOAD_BOOT 的影响）
- bit4：表示 GPIO25 管脚的值
- bit5 ~ bit15：无效

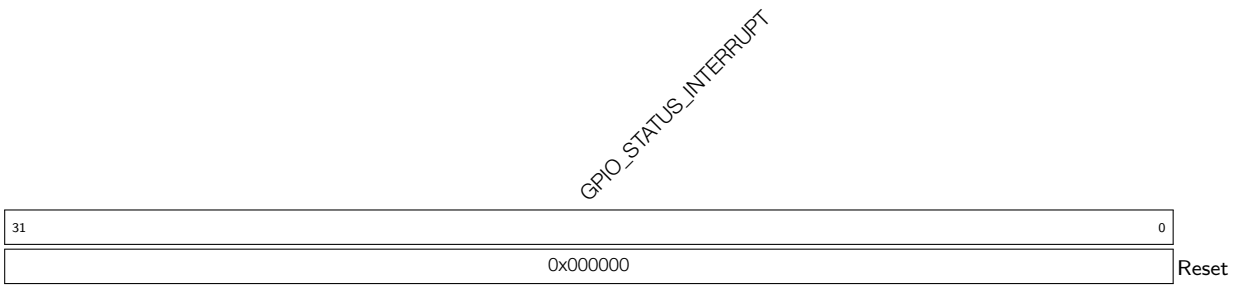
关于 GPIO Strapping 管脚详细信息，请参考 8 芯片 *Boot* 控制 章节。
(RO)

Register 6.8. GPIO_IN_REG (0x003C)



GPIO_IN_DATA_NEXT 表示 GPIO0 ~ GPIO27 的输入值。每一位均代表一个管脚的输入值：
0：低电平
1：高电平
bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27，bit28 ~ bit31 无效。
(RO)

Register 6.9. GPIO_STATUS_REG (0x0044)

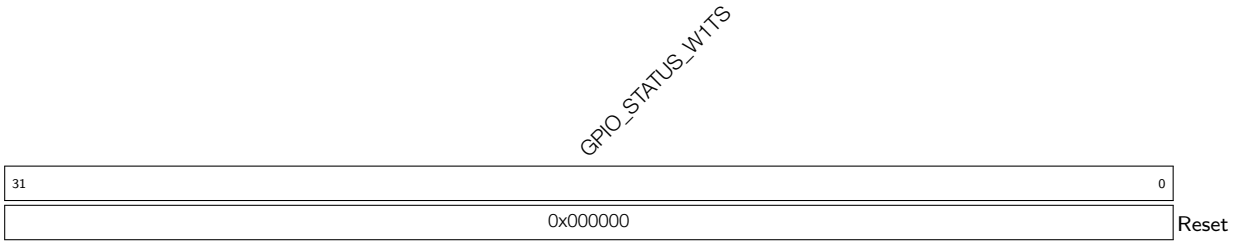


GPIO_STATUS_INTERRUPT GPIO0 ~ GPIO27 的中断状态寄存器，并且软件可配置该寄存器的值。

- bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。
- 每一位可查询对应 GPIO 的中断状态：
  - 0：表示对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 0
  - 1：表示对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所选择的中断，或软件配置该位寄存器的值为 1

(R/W/WTC)

Register 6.10. GPIO_STATUS_W1TS_REG (0x0048)

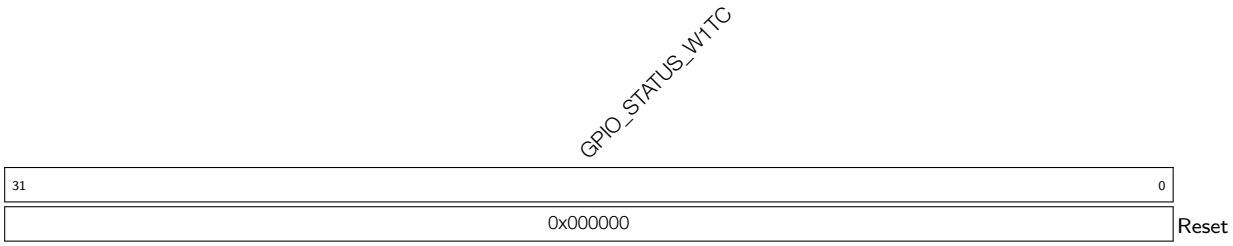


GPIO_STATUS_W1TS 配置是否置位 GPIO0 ~ GPIO27 的中断状态寄存器 GPIO_STATUS_INTERRUPT。

- bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。
- 每一位置 1，则 GPIO_STATUS_INTERRUPT 中相应位也将置 1。
- 推荐使用此寄存器置位 GPIO_STATUS_INTERRUPT。

(WT)

Register 6.11. GPIO_STATUS_W1TC_REG (0x004C)

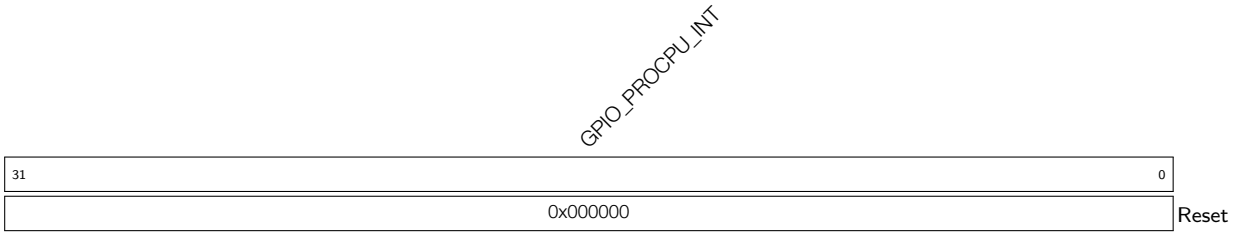


GPIO_STATUS_W1TC 配置是否清除 GPIO0 ~ GPIO27 的中断状态寄存器 GPIO_STATUS_INTERRUPT。

- bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。
- 每一位置 1，则 GPIO_STATUS_INTERRUPT 中相应位将被清除。
- 推荐使用此寄存器来清除 GPIO_STATUS_INTERRUPT。

(WT)

Register 6.12. GPIO_PCPU_INT_REG (0x005C)

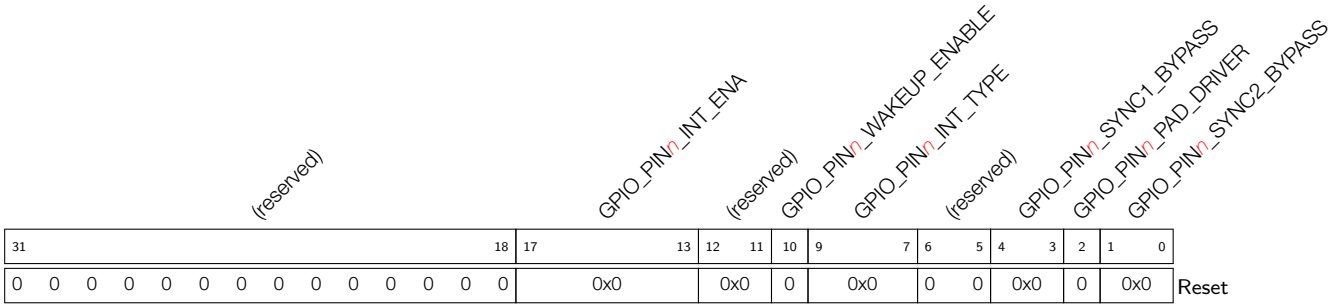


GPIO_PROCPU_INT 表示 GPIO0 ~ GPIO27 CPU 中断状态。

- bit0 ~ bit27 分别对应 GPIO0 ~ GPIO27。bit28 ~ bit31 无效。
- 如果 GPIO_PIN*n*_REG 中 bit13 有效，即使能 CPU 中断，则此寄存器所示的中断状态应与 GPIO_STATUS_REG 中相应位的中断状态一致，否则为 0。
  - 0：表示未使能 CPU 中断，或对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所选择的中断
  - 1：表示使能 CPU 中断后，对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所选择的中断

(RO)

Register 6.13. GPIO_PIN*n*_REG (*n*: 0-27) (0x0074+4n*)**



GPIO_PIN*n*_SYNC2_BYPASS 配置是否使能 GPIO 输入数据在第二级同步中 IO MUX 运行时钟上升沿同步或下降沿同步。

0：不进行同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

GPIO_PIN*n*_PAD_DRIVER 配置选择管脚的输出驱动模式。

0：正常输出

1：开漏输出

(R/W)

GPIO_PIN*n*_SYNC1_BYPASS 配置是否使能 GPIO 输入数据在第一级同步中 IO MUX 运行时钟上升沿同步或下降沿同步。

0：不进行同步

1：在下降沿进行同步

2：在上升沿进行同步

3：在上升沿进行同步

(R/W)

GPIO_PIN*n*_INT_TYPE 配置 GPIO 中断类型。

0：关闭 GPIO 中断

1：上升沿触发

2：下降沿触发

3：任一沿触发

4：低电平触发

5：高电平触发

(R/W)

GPIO_PIN*n*_WAKEUP_ENABLE 配置是否使能 GPIO 唤醒功能。

0：不使能

1：使能

该功能仅能将 CPU 从 Light-sleep 模式唤醒。

(R/W)

见下页

Register 6.13. GPIO_PIN*n*_REG (*n*: 0-27) (0x0074+4n*)**

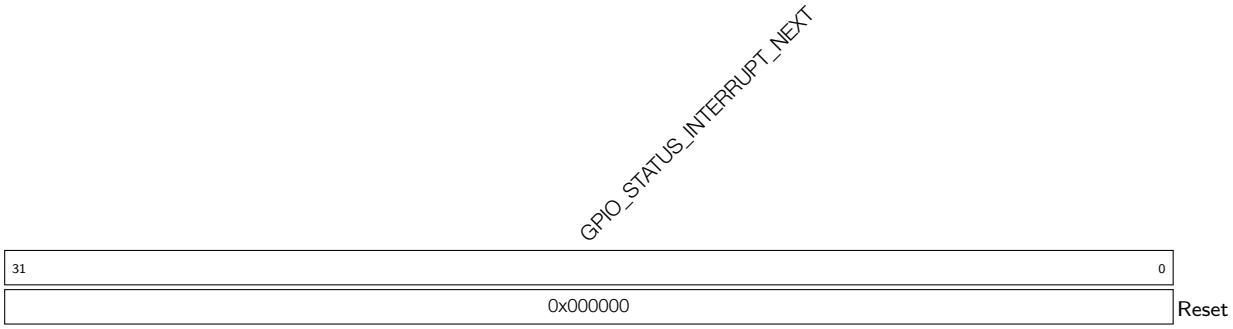
接上页

GPIO_PIN*n*_INT_ENA 配置是否使能 CPU 中断或 CPU 非屏蔽中断。

- bit13：配置是否使能 CPU 中断：
  0：不使能
  1：使能
- bit14：配置是否使能 CPU 非屏蔽中断：
  0：不使能
  1：使能
- bit15 ~ bit17：无效

(R/W)

Register 6.14. GPIO_STATUS_NEXT_REG (0x014C)

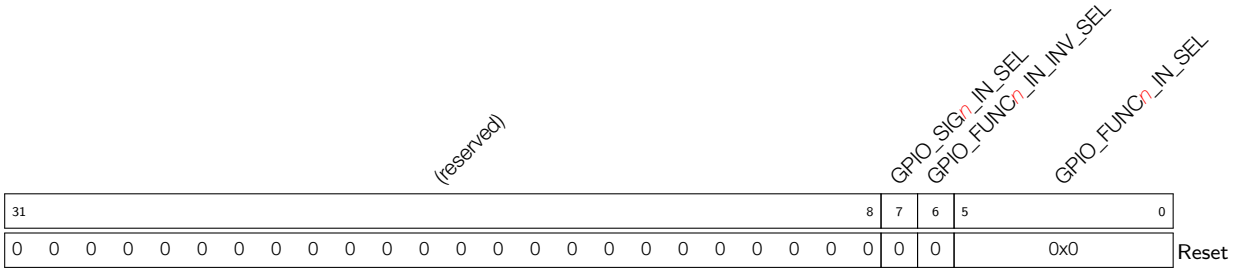


GPIO_STATUS_INTERRUPT_NEXT 表示 GPIO0 ~ GPIO27 的中断源信号状态。
bit0 ~ bit27 分别对应 GPIO0 ~ 27。bit28 ~ bit31 无效，每个 bit 表示：

- 0：表示对应 GPIO 未产生 GPIO_PIN*n*_INT_TYPE 所选择的中断
- 1：表示对应 GPIO 产生了 GPIO_PIN*n*_INT_TYPE 所选择的中断

上述中断可以为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。
(RO)

Register 6.15. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-127) (0x0154+4n*)**



GPIO_FUNC*n*_IN_SEL 配置从 28 个 GPIO 中选择一个管脚连接输入信号 *n*。

0：选择 GPIO0

1：选择 GPIO1

.....

26：选择 GPIO26

27：选择 GPIO27

或者

0x38：选择恒高电平输入

0x3C：选择恒低电平输入

(R/W)

GPIO_FUNC*n*_IN_INV_SEL 配置是否反转输入值。

0：不反转

1：反转

(R/W)

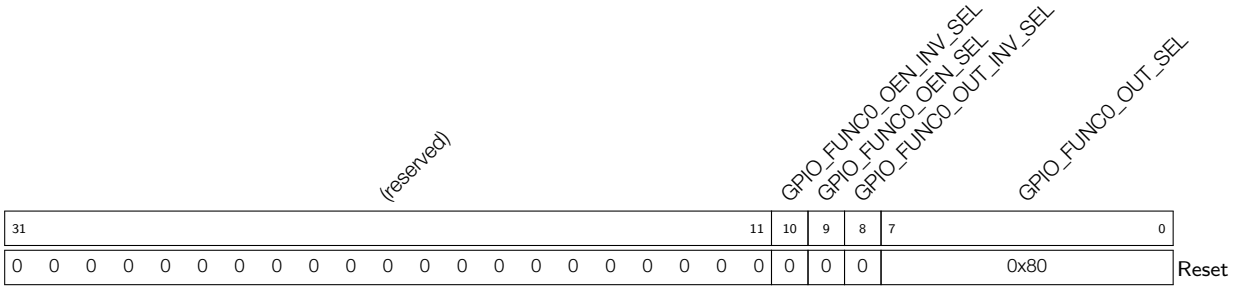
GPIO_SIG*n*_IN_SEL 配置是否通过 GPIO 交换矩阵连接信号。

0：旁路 GPIO 交换矩阵，即直接通过 IO MUX 连接信号与外设

1：通过 GPIO 交换矩阵连接信号与外设

(R/W)

Register 6.16. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-27) (0x0554+4n*)**



GPIO_FUNC*n*_OUT_SEL 配置从 128 个外设信号中选择一个信号 *Y* (0 <= *Y* < 128) 输出至 GPIO*n*。

0：选择信号 0

1：选择信号 1

.....

126：选择信号 126

127：选择信号 127

或者

128：GPIO_OUT_REG 和 GPIO_ENABLE_REG 中的 bit*n* 将用作输出值和输出使能信号。

信号列表见表 6-2。

(R/W/SC)

GPIO_FUNC*n*_OUT_INV_SEL 配置是否反转输出值。

0：不反转

1：反转

(R/W/SC)

GPIO_FUNC*n*_OEN_SEL 配置选择输出使能信号。

0：使用外设的输出使能信号

1：强制使用 GPIO_ENABLE_REG 的 bit*n* 用作输出使能信号

(R/W)

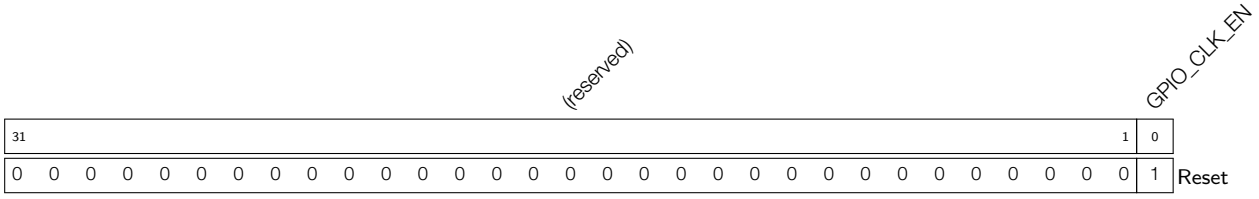
GPIO_FUNC*n*_OEN_INV_SEL 配置是否反转输出使能信号。

0：不反转

1：反转

(R/W)

Register 6.17. GPIO_CLOCK_GATE_REG (0x062C)



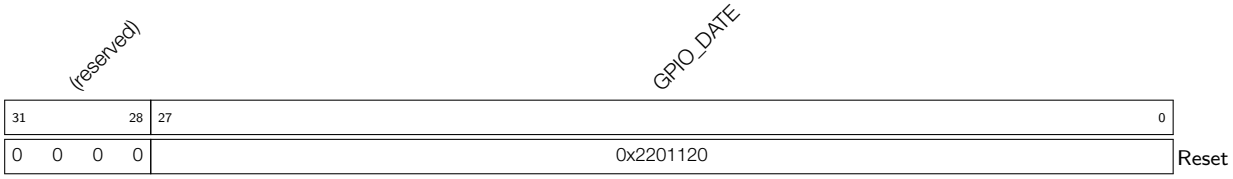
GPIO_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能，则时钟自由运转。

(R/W)

Register 6.18. GPIO_DATE_REG (0x06FC)

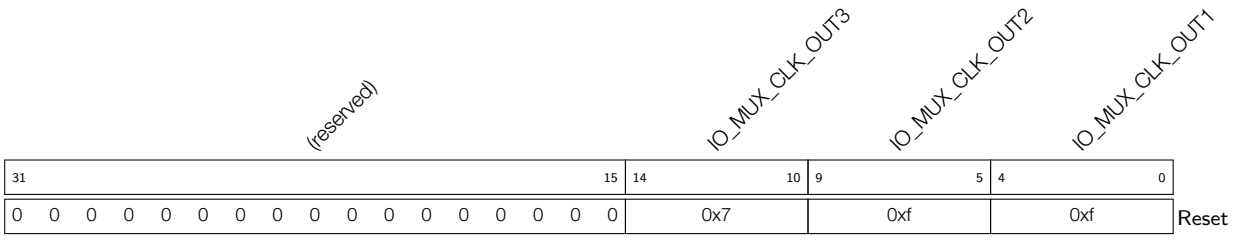


GPIO_DATE 版本控制寄存器。(R/W)

6.18.2 IO MUX 寄存器

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

Register 6.19. IO_MUX_PIN_CTRL_REG (0x0000)



IO_MUX_CLK_OUT*x* 配置 I2S 的输出时钟。

0x0：配置 I2S 外设时钟输出到 CLK_OUT_out*x*

有关 CLK_OUT_out*x* 的信息，见表 6-2。

(R/W)

Register 6.20. IO_MUX_GPIO*n*_REG (*n*: 0-27) (0x0004+4n*)**

IO_MUX_GPIO*n*_MCU_OE 配置是否使能睡眠模式下 GPIO*n* 的输出。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_SLP_SEL 配置是否使能 GPIO*n* 进入睡眠模式。

0：不进入

1：进入

(R/W)

IO_MUX_GPIO*n*_MCU_WPD 配置是否使能睡眠模式下 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_WPU 配置是否使能睡眠模式下 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_IE 配置是否使能睡眠模式下 GPIO*n* 的输入。

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_MCU_DRV 配置睡眠模式下 GPIO*n* 的驱动强度。

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO*n*_FUN_WPD 配置是否使能 GPIO*n* 的下拉电阻。

0：不使能

1：使能

(R/W)

见下页

Register 6.20. IO_MUX_GPIO*n*_REG (*n*: 0-27) (0x0004+4n*)**

接上页

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

0：不使能

1：使能

(R/W)

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

0：不使能

1：使能

(R/W)

IO_MUX_GPIO*n*_FUN_DRV 配置 GPIO*n* 的驱动强度

0: ~5 mA

1: ~10 mA

2: ~20 mA

3: ~40 mA

(R/W)

IO_MUX_GPIO*n*_MCU_SEL 配置选择 IO MUX 功能。

0：选择功能 0

1：选择功能 1

.....

(R/W)

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

0：不使能

1：使能

(R/W)

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

0：不使能

1：使能

(R/W)

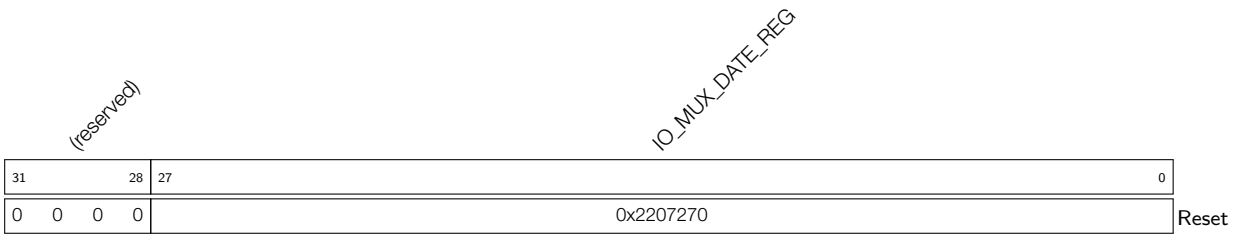
IO_MUX_GPIO*n*_HYS_SEL 配置选择 GPIO*n* 迟滞功能的使能信号。

0：使用 eFuse 的输出使能信号

1：使用 IO_MUX_GPIO*n*_HYS_EN 用作输出使能信号

(R/W)

Register 6.21. IO_MUX_DATE_REG (0x00FC)

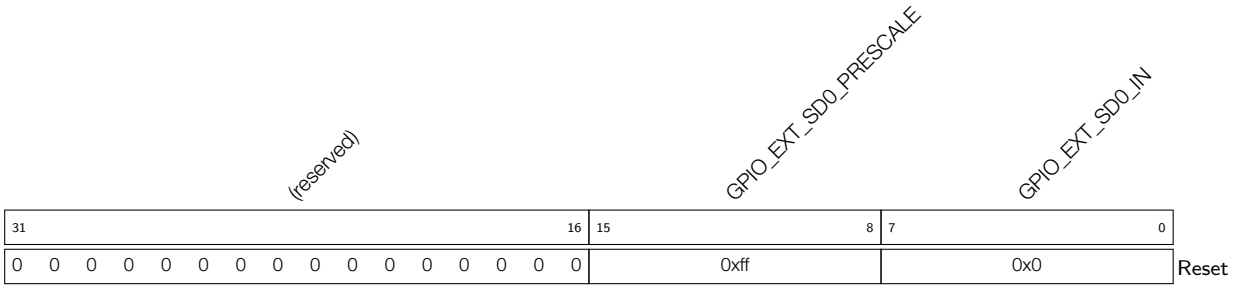


IO_MUX_DATE_REG 版本控制寄存器。(R/W)

6.18.3 GPIO_EXT 寄存器

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

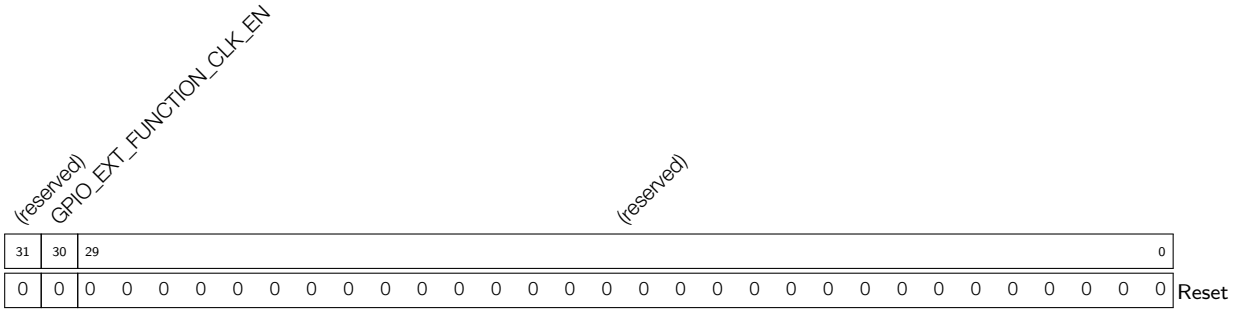
Register 6.22. GPIO_EXT_SIGMADELTA*n*_REG (*n*: 0-3) (0x0000+0x4n*)**



GPIO_EXT_SD0_IN 配置 SDM 输出信号的占空比。(R/W)

GPIO_EXT_SD0_PRESCALE 配置 IO MUX 运行时钟的分频系数。(R/W)

Register 6.23. GPIO_EXT_SIGMADELTA_MISC_REG (0x0024)



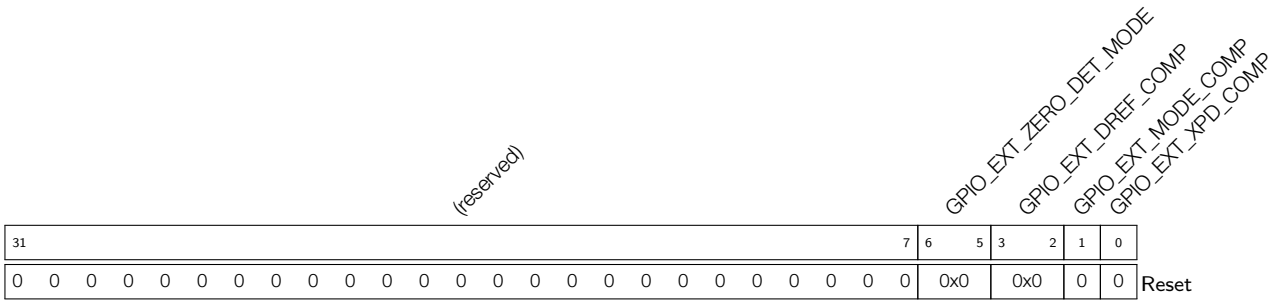
GPIO_EXT_FUNCTION_CLK_EN 配置是否使能 SDM 时钟。

0：不使能

1：使能

(R/W)

Register 6.24. GPIO_EXT_PAD_COMP_CONFIG_REG (0x0028)



GPIO_EXT_ZERO_DET_MODE 配置选择模拟 PAD 电压比较功能的中断源产生模式。

0：关闭中断源产生

1：保留位

2：保留位

3：使能中断源为 PAD_COMP_OUT_sync 信号的任意沿

(R/W)

GPIO_EXT_DREF_COMP 配置选择模拟 PAD 电压比较功能的内部参考电压。

0：内部参考电压为 0 * VDDPST2

1：内部参考电压为 0.1 * VDDPST2

.....

6：内部参考电压为 0.6 * VDDPST2

7：内部参考电压为 0.7 * VDDPST2

(R/W)

GPIO_EXT_MODE_COMP 配置选择模拟 PAD 电压比较功能的参考电压。

0：参考电压为内部参考电压

1：参考电压为 GPIO10 PAD 上的电压

(R/W)

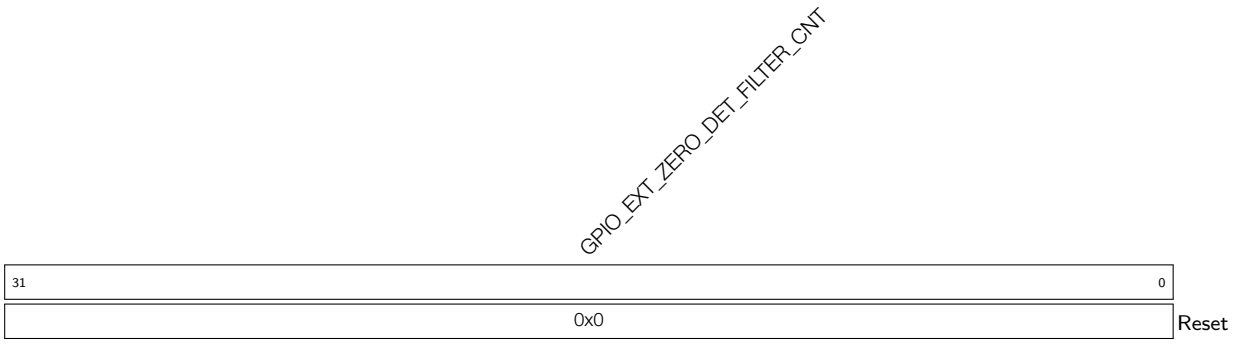
GPIO_EXT_XPD_COMP 配置是否使能模拟 PAD 电压比较功能。

0：不使能

1：使能

(R/W)

Register 6.25. GPIO_EXT_PAD_COMP_FILTER_REG (0x002C)

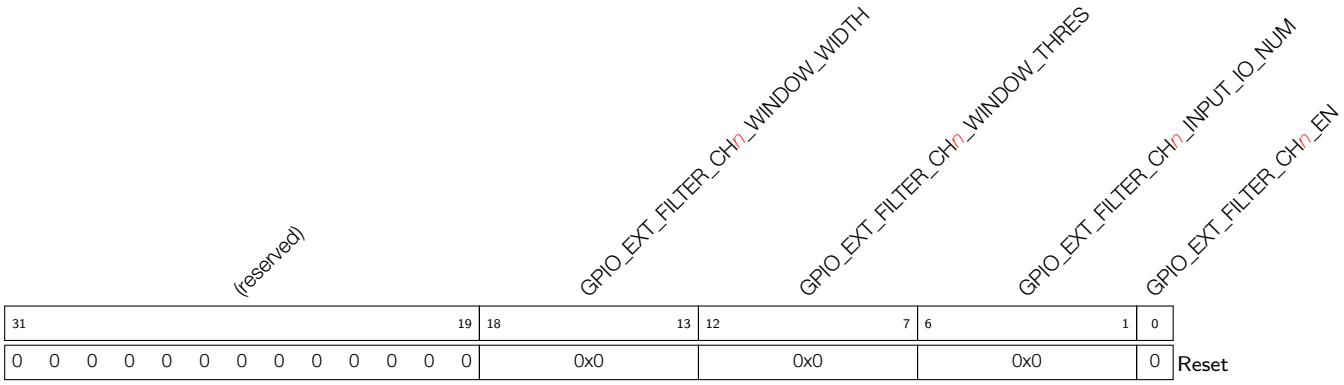


GPIO_EXT_ZERO_DET_FILTER_CNT 配置模拟 PAD 电压比较功能的新中断源屏蔽周期。

单位：IO MUX 运行时钟的一个周期

(R/W)

Register 6.26. GPIO_EXT_GLITCH_FILTER_CH*n*_REG (*n*: 0-7) (0x0030+0x4n*)**



GPIO_EXT_FILTER_CH*n*_EN 配置是否使能毛刺滤波器的通道 *n*。

0：不使能

1：使能

(R/W)

GPIO_EXT_FILTER_CH*n*_INPUT_IO_NUM 配置选择毛刺滤波器的 GPIO 输入。

0：选择 GPIO0

1：选择 GPIO1

.....

26：选择 GPIO26

27：选择 GPIO27

(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_THRES 配置毛刺滤波器的窗口阈值。窗口阈值应小于或等于 GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH。

单位：IO MUX 运行时钟的一个周期

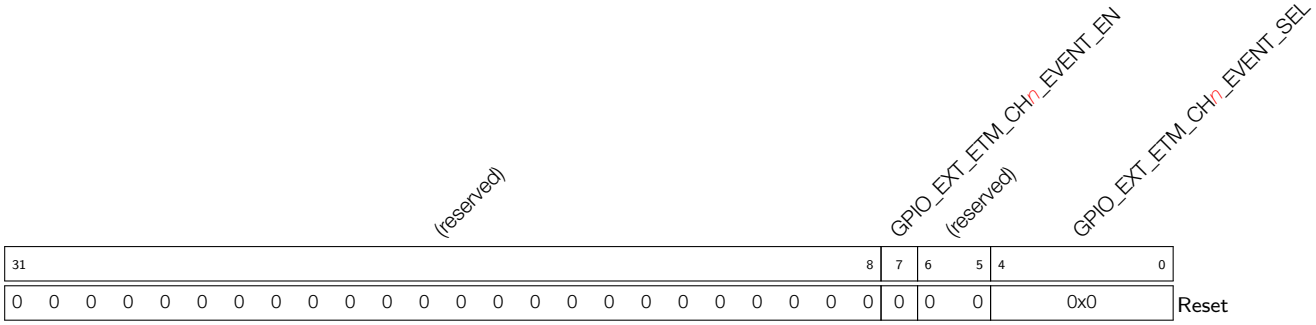
(R/W)

GPIO_EXT_FILTER_CH*n*_WINDOW_WIDTH 配置毛刺滤波器的窗口宽度。窗口宽度有效值为 0 +~+62，63 为保留值，不可使用。

单位：IO MUX 运行时钟的一个周期

(R/W)

Register 6.27. GPIO_EXT_ETM_EVENT_CH*n*_CFG_REG (*n*: 0-7) (0x0060+0x4n*)**



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

0：选择 GPIO0

1：选择 GPIO1

.....

26：选择 GPIO26

27：选择 GPIO27

(R/W)

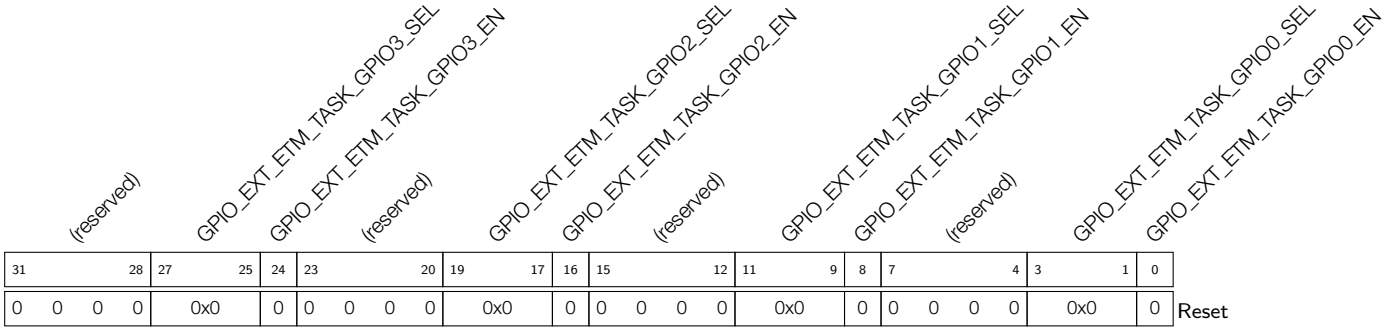
GPIO_EXT_ETM_CH*n*_EVENT_EN 配置是否使能 ETM 事件发送。

0：不使能

1：使能

(R/W)

Register 6.28. GPIO_EXT_ETM_TASK_P0_CFG_REG (0x00A0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 0 ~ 3) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 0 ~ 3) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 6.29. GPIO_EXT_ETM_TASK_P1_CFG_REG (0x00A4)

GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 4 ~ 7) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 4 ~ 7) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

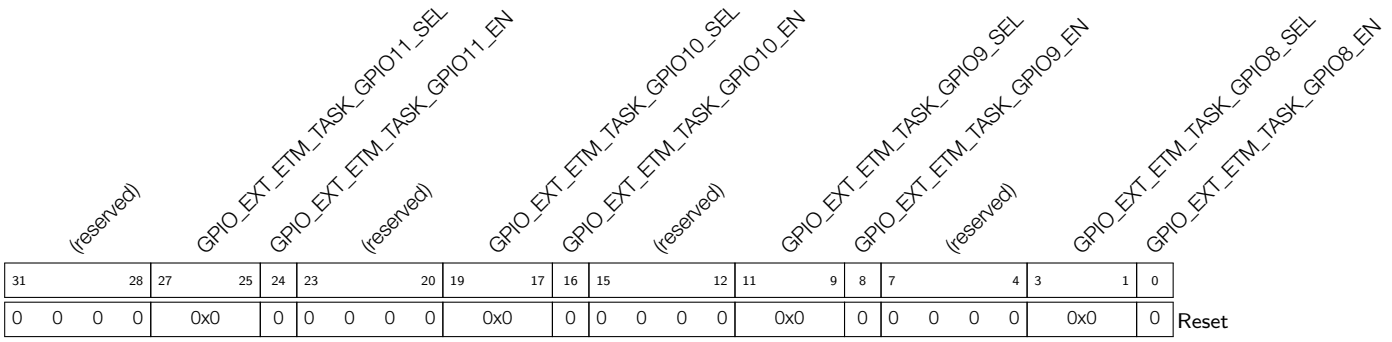
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 6.30. GPIO_EXT_ETM_TASK_P2_CFG_REG (0x00A8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 8 ~ 11) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 8 ~ 11) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 6.31. GPIO_EXT_ETM_TASK_P3_CFG_REG (0x00AC)

GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 12 ~ 15) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 12 ~ 15) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

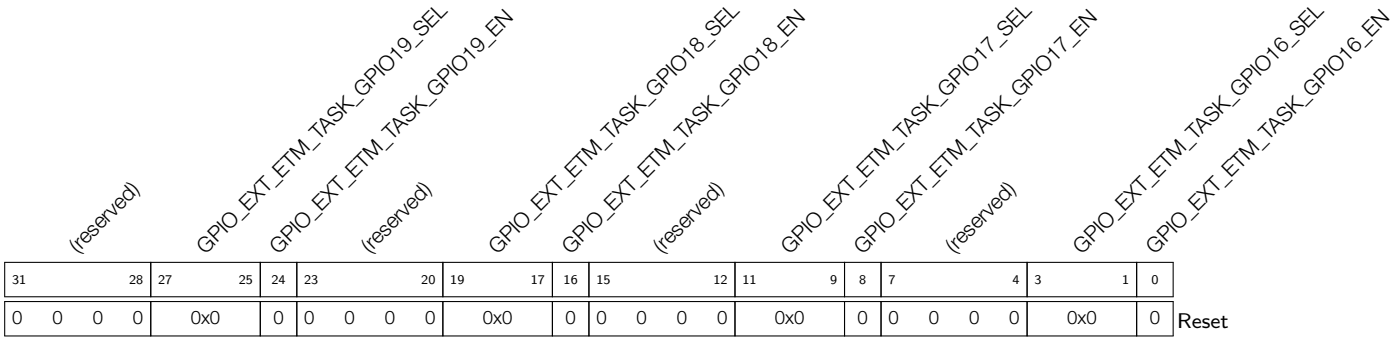
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 6.32. GPIO_EXT_ETM_TASK_P4_CFG_REG (0x00B0)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 16 ~ 19) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 16 ~ 19) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 6.33. GPIO_EXT_ETM_TASK_P5_CFG_REG (0x00B4)

GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 20 ~ 23) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 20 ~ 23) 配置选择 GPIO*n* 的 ETM 任务通道。

0：选择通道 0

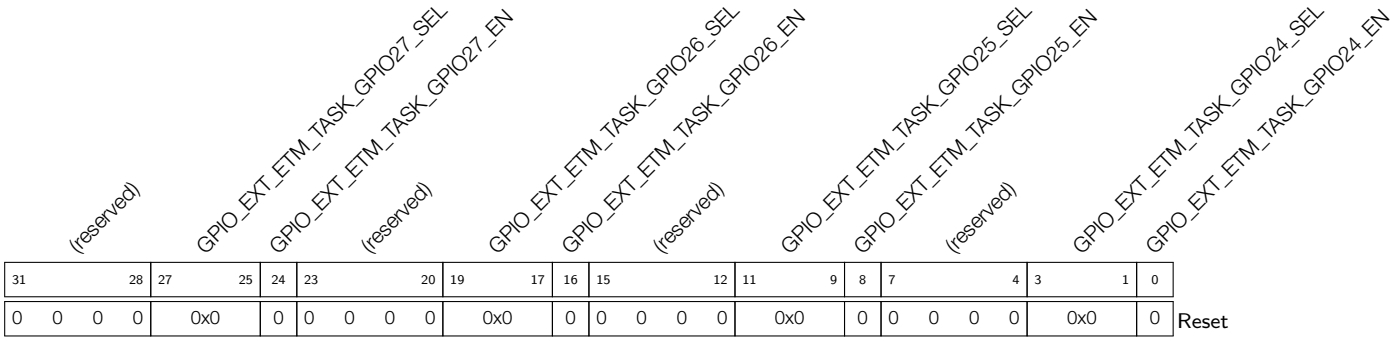
1：选择通道 1

.....

7：选择通道 7

(R/W)

Register 6.34. GPIO_EXT_ETM_TASK_P6_CFG_REG (0x00B8)



GPIO_EXT_ETM_TASK_GPIO*n*_EN (*n* = 24 ~ 27) 配置是否使能 GPIO*n* 响应 ETM 任务。

0：不使能

1：使能

(R/W)

GPIO_EXT_ETM_TASK_GPIO*n*_SEL (*n* = 24 ~ 27) 配置选择 GPIO *n* 的 ETM 任务通道。

0：选择通道 0

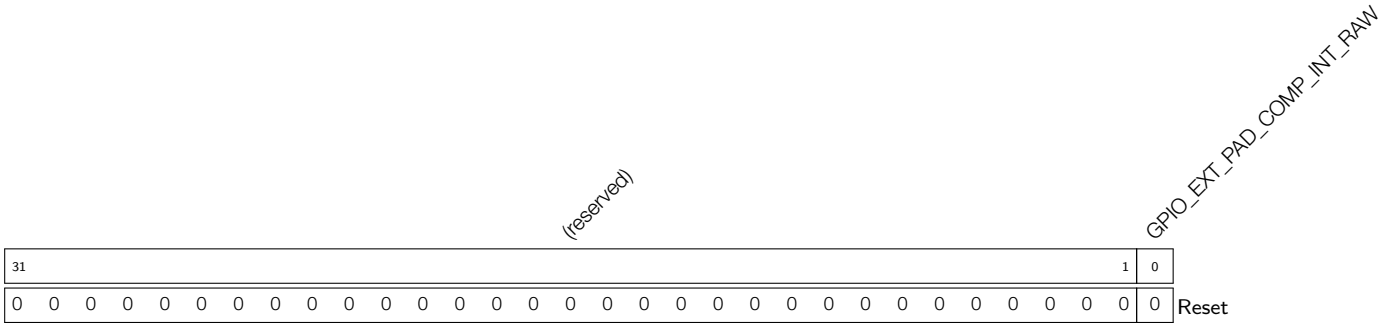
1：选择通道 1

.....

7：选择通道 7

(R/W)

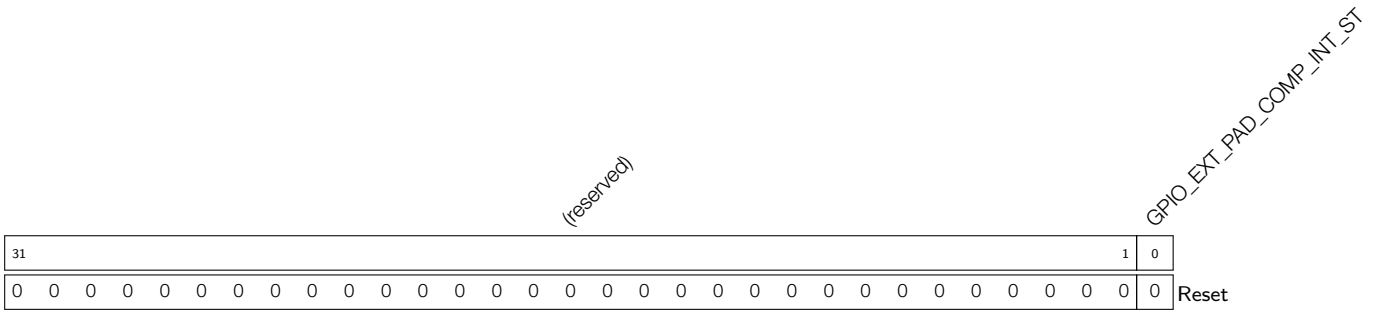
Register 6.35. GPIO_EXT_INT_RAW_REG (0x00E0)



GPIO_EXT_PAD_COMP_INT_RAW GPIO_EXT_PAD_COMP_INT 的原始中断位。

(RO/WTC/SS)

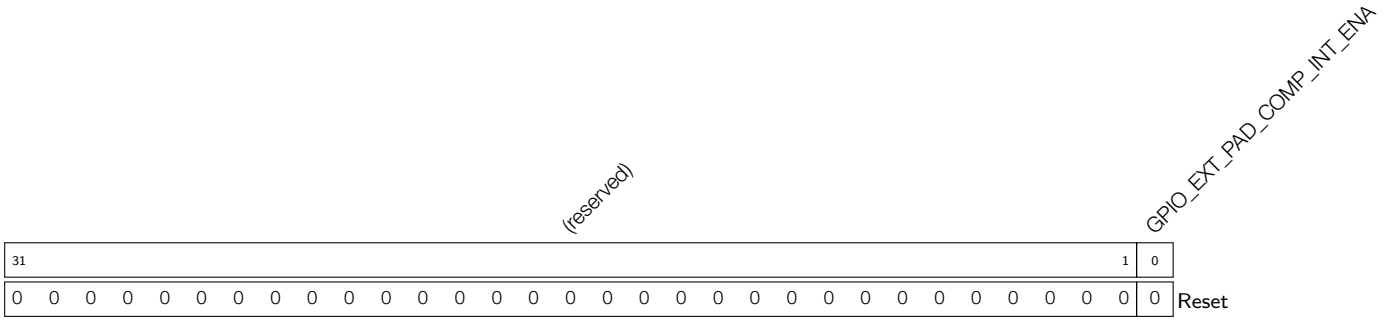
Register 6.36. GPIO_EXT_INT_ST_REG (0x00E4)



GPIO_EXT_PAD_COMP_INT_ST GPIO_EXT_PAD_COMP_INT 的中断状态位。

(RO)

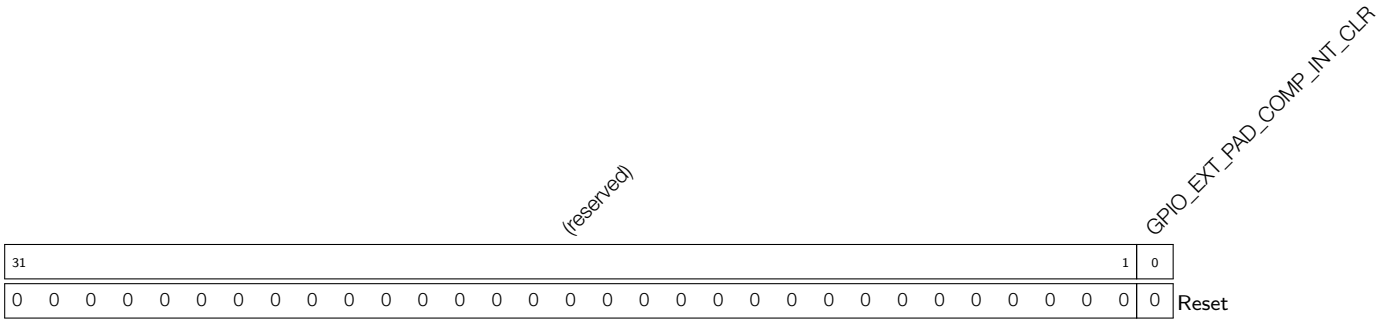
Register 6.37. GPIO_EXT_INT_ENA_REG (0x00E8)



GPIO_EXT_PAD_COMP_INT_ENA GPIO_EXT_PAD_COMP_INT 的中断使能位。

(RW)

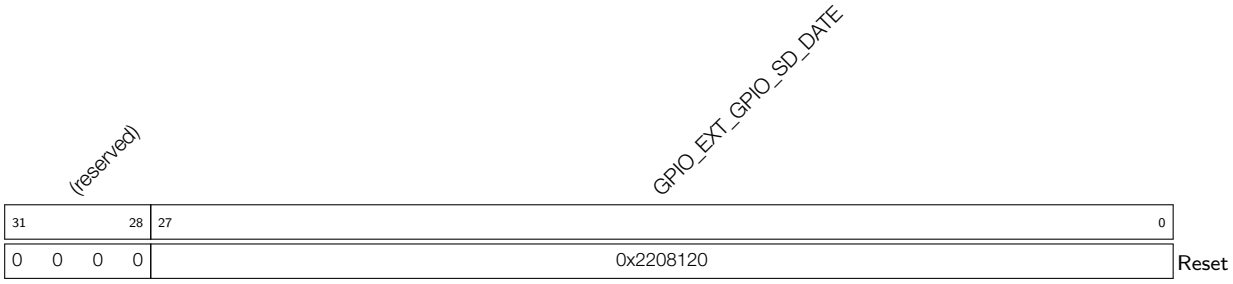
Register 6.38. GPIO_EXT_INT_CLR_REG (0x00EC)



GPIO_EXT_PAD_COMP_INT_CLR GPIO_EXT_PAD_COMP_INT 的中断清除位。

(WT)

Register 6.39. GPIO_EXT_VERSION_REG (0x00FC)



GPIO_EXT_GPIO_SD_DATE 版本控制寄存器。

(R/W)

7 复位和时钟

7.1 复位

7.1.1 概述

ESP32-H2 提供四种级别的复位方式，分别是 CPU 复位 (CPU reset)、内核复位 (Core reset)、系统复位 (System reset) 和芯片复位 (Chip reset)。除芯片复位外，其他复位方式不影响片上内存存储的数据。图 7-1 展示了整个芯片系统的结构以及四种复位等级。

7.1.2 结构图

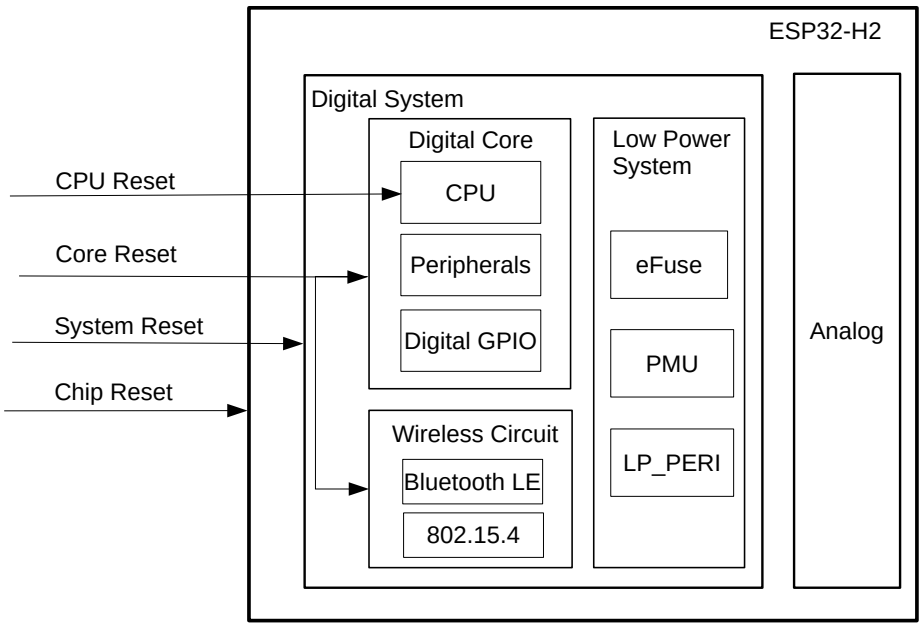


图 7-1. 四种复位等级

ESP32-H2 的数字系统分为两部分：高性能系统 (High Performance System, HP system) 和 低功耗系统 (Low Power System, LP system)。其中高性能系统包含数字内核 (Digital Core) 、无线电路 (Wireless Circuit) 和 HP SRAM，低功耗系统中仅包含一些低功耗外设和 LP SRAM。详细信息可参考图 7-1（HP SRAM 和 LP SRAM 不会复位，因此未在图中标注）。

7.1.3 特性

- 支持四种复位等级：
  - CPU 复位：复位 CPU 核。复位释放后，程序将从 CPU Reset Vector (0x40000000) 开始执行；
  - 内核复位：复位除低功耗系统以外的其他数字系统，包括 CPU、外设、数字 GPIO、Bluetooth® LE、802.15.4；
  - 系统复位：复位包括低功耗系统在内的整个数字系统；

  - 芯片复位：复位整个芯片。

- 支持软件复位和硬件复位：
  - 软件复位：CPU 配置相关寄存器可触发软件复位，见章节 2 低功耗管理 *(RTC_CNTL) [to be added later]*；
  - 硬件复位：硬件复位直接由硬件电路触发。

7.1.4 功能描述

上述任一复位发生时，CPU 将立刻复位。复位释放后，CPU 可通过读取 LP_CLKRST_RESET_CAUSE 获取复位源。

表 7-1 列出了从上述寄存器中可能读出的复位源及其触发的复位等级。

表 7-1. 复位源

编码	复位源	复位等级	说明
0x01	芯片复位[1]	芯片复位	—
0x0F	欠压系统复位	芯片复位或系统复位	欠压检测器触发的复位[2]
0x10	RWDT 系统复位	系统复位	详见章节 13 看门狗定时器 *(WDT)*
0x12	超级看门狗复位	系统复位	详见章节 13 看门狗定时器 *(WDT)*
0x03	软件系统复位	内核复位	配置 LP_AON_HPSYS_SW_RESET 触发
0x05	Deep-sleep 复位	内核复位	详见章节 2 低功耗管理 *(RTC_CNTL) [to be added later]*
0x07	MWDT0 内核复位	内核复位	详见章节 13 看门狗定时器 *(WDT)*
0x08	MWDT1 内核复位	内核复位	详见章节 13 看门狗定时器 *(WDT)*
0x09	RWDT 内核复位	内核复位	详见章节 13 看门狗定时器 *(WDT)*
0x14	eFuse 复位	内核复位	eFuse CRC 校验错误触发复位
0x15	USB (UART) 复位	内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 Serial 接口将触发此复位，详见章节 30 *USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*
0x16	USB (JTAG) 复位	内核复位	外部 USB 主机发送特定命令给 USB Serial/JTAG 控制器的 JTAG 接口将触发此复位，详见章节 30 *USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*
0x0B	MWDT0 CPU 复位	CPU 复位	详见章节 13 看门狗定时器 *(WDT)*
0x0C	软件 CPU 复位	CPU 复位	配置 LP_AON_CPU_CORE0_SW_RESET 触发
0x0D	RWDT CPU 复位	CPU 复位	详见章节 13 看门狗定时器 *(WDT)*
0x11	MWDT1 CPU 复位	CPU 复位	详见章节 13 看门狗定时器 *(WDT)*
0x17	电源毛刺复位	系统复位	电源受到电源毛刺攻击时触发此复位
0x18	JTAG CPU 复位	CPU 复位	接收 “JDB 复位 CPU” 指令时触发

[1] 芯片复位的触发源包括以下两项：
- 芯片上电触发芯片复位
- 欠压检测器触发芯片复位

[2] 欠压检测器在检测到欠压状态时，将根据寄存器配置，选择触发系统复位或者芯片复位，详见章节 2 低功耗管理 *(RTC_CNTL) [to be added later]*。

7.1.5 外设复位

外设可单独复位，也可通过配置内核复位、系统复位、芯片复位连带复位。在解复位后，需要读取各外设的解复位完成指示寄存器，此类寄存器为 1 表示对应外设解复位成功，否则外设可能无法正常工作。

ESP32-H2 外设复位寄存器已整合到 PCR (POWER/CLOCK/RESET) 模块，可通过章节 7.4寄存器列表 查看各外设对应的复位寄存器以及解复位成功指示寄存器。

7.2 时钟

7.2.1 概述

ESP32-H2 的时钟主要来源于振荡器 (oscillator, OSC)、RC 振荡电路和 PLL 时钟生成电路。上述时钟源产生的时钟经时钟分频器或时钟选择器等时钟模块的处理，使得大部分功能模块可以根据不同功耗和性能需求来获取及选择对应频率的工作时钟。图 7-2 为系统时钟结构。

7.2.2 结构图

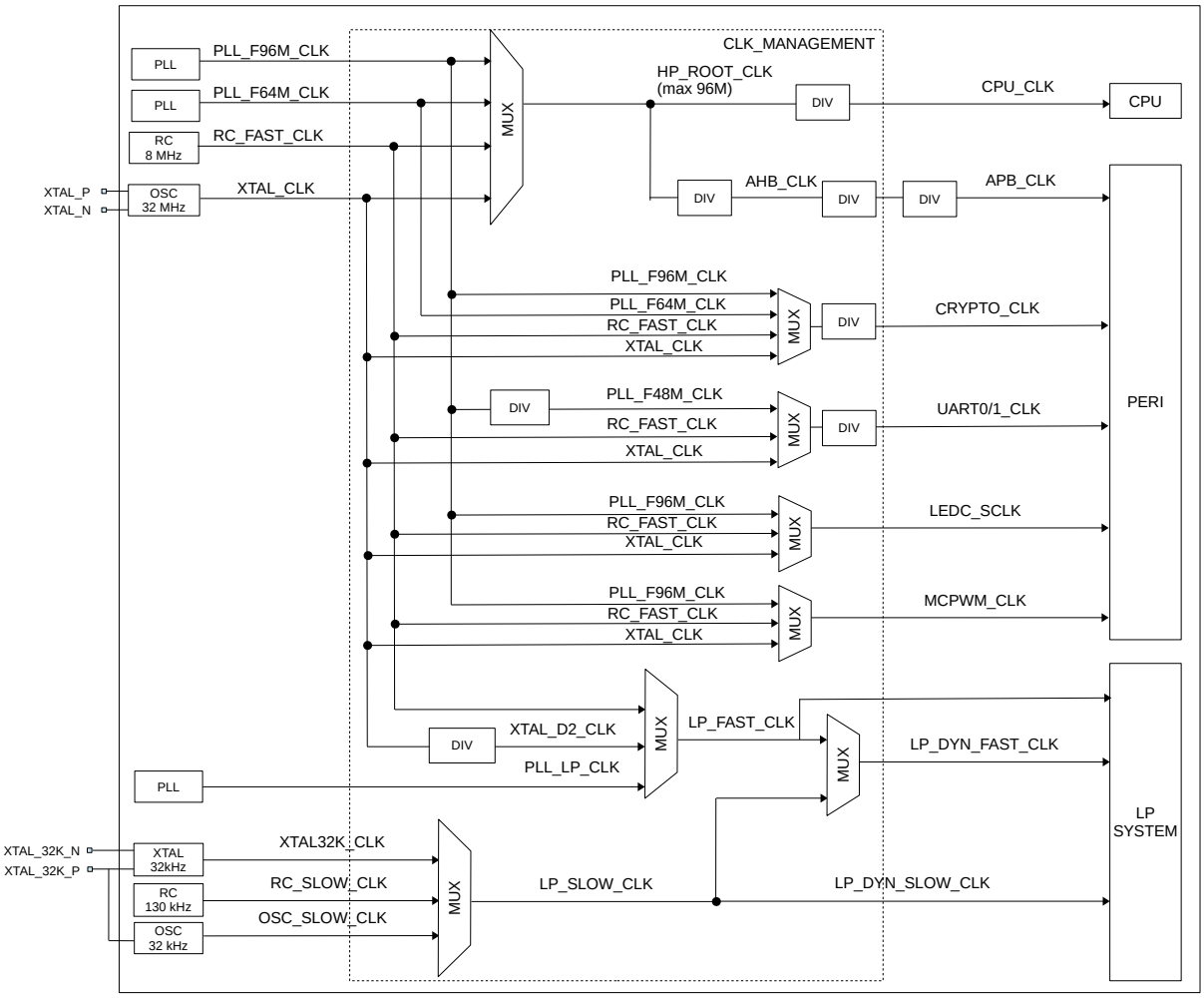


图 7-2. 系统时钟

7.2.3 特性

ESP32-H2 的时钟根据频率不同，可分为：

- 高性能时钟，主要为 CPU 和数字外设提供工作时钟
  - PLL_F96M_CLK：96 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）
  - PLL_F64M_CLK：64 MHz 内部 PLL 时钟（参考时钟是 XTAL_CLK）
  - XTAL_CLK：32 MHz 外部晶振时钟
- 低功耗时钟，主要为低功耗系统以及部分处于低功耗模式的外设提供工作时钟
  - XTAL32K_CLK：32 kHz 外部晶振时钟
  - RC_FAST_CLK：内置快速 RC 振荡器时钟，频率可调节（通常为 8 MHz）
  - RC_SLOW_CLK：内置慢速 RC 振荡器，频率可调节（通常为 130 kHz）
  - OSC_SLOW_CLK：外置低速时钟，通常频率为 32 kHz，通过 XTAL_32K_P 输入，在配置好这个 GPIO 的状态后需要配置 Hold 功能，具体请参考章节 6 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* > 6.9 *GPIO 管脚的 Hold 特性*
  - PLL_LP_CLK：内部 PLL 时钟，参考时钟为 XTAL32K_CLK

7.2.4 功能描述

7.2.4.1 高性能系统时钟

如图 7-2 所示，CPU_CLK 为 CPU 主时钟。CPU 主频最高可以达到 96 MHz，并且能够在超低频下工作（通常为 2 MHz），以减少功耗。CPU_CLK 与 AHB_CLK、APB_CLK 共用一个时钟源。用户可配置 PCR_SOC_CLK_SEL 选择 XTAL_CLK、PLL_96M_CLK、PLL_64M_CLK 或 RC_FAST_CLK 作为 CPU_CLK 的时钟源，具体请参考表 7-2 和表 7-3。默认状态下，CPU 的时钟源为 XTAL_CLK，且分频系数为 1 分频，即 32 MHz。

表 7-2. CPU_CLK 时钟源选择

PCR_SOC_CLK_SEL	时钟源
0	XTAL_CLK
1	PLL_F96M_CLK
2	RC_FAST_CLK
3	PLL_F64M_CLK

表 7-3. CPU_CLK、AHB_CLK 和 HP_ROOT_CLK 的时钟频率

时钟名称	时钟源	时钟频率
HP_ROOT_CLK	PLL_F96M_CLK	96 MHz
	PLL_F64M_CLK	64 MHz
	XTAL_CLK	32 MHz
	RC_FAST_CLK	8 MHz
CPU_CLK[1]	HP_ROOT_CLK	$f_{HP\_ROOT\_CLK}$ / (PCR_CPU_DIV_NUM + 1)
AHB_CLK[2]	HP_ROOT_CLK	$f_{HP\_ROOT\_CLK}$ / (PCR_AHB_DIV_NUM + 1)

[1] CPU_CLK 频率必须大于等于 AHB_CLK 频率，并且必须为 AHB_CLK 频率的整数倍

[2] AHB_CLK 时钟频率不能超过 32 MHz

说明：

配置 HP_ROOT_CLK 的时钟源选择寄存器或者 CPU_CLK、AHB_CLK 的分频寄存器后，需要额外配置 PCR_BUS_CLOCK_UPDATE 寄存器将上述寄存器配置生效，可通过读取 PCR_BUS_CLOCK_UPDATE 寄存器来判断上述配置是否成功生效。

如图 7-2 所示，APB_CLK 是在 AHB_CLK 基础上经过两级分频得到的。其中第一级分频为恒定分频，即始终按照分频系数（PCR_APB_DIV_NUM + 1）的值进行分频。第二级分频为自动降频，如果芯片中的主机没有访问外设寄存器的请求时，会以分频系数（APB_DECREASE_DIV_NUM + 1）进行二次分频产生 APB_CLK，从而达到节约功耗的目的。但当芯片中的主机发起了访问外设寄存器的请求时，APB_CLK 将恢复至一级分频后的频率。

注意，使用自动降频功能将降低芯片性能，用户可将 APB_DECREASE_DIV_NUM 设置为 0 以禁用此功能。默认情况下，该功能禁用。

7.2.4.2 低功耗系统时钟

低功耗系统能够在大多数时钟源关闭的状态下工作。低功耗系统时钟包括 LP_SLOW_CLK 时钟和 LP_FAST_CLK 时钟。

LP_SLOW_CLK 和 LP_FAST_CLK 的时钟源为低频时钟，其中：

- LP_SLOW_CLK 时钟有三种可能的时钟源：
  - RC_SLOW_CLK
  - XTAL32K_CLK
  - OSC_SLOW_CLK
- LP_FAST_CLK 时钟有三种可能的时钟源：
  - XTAL_CLK 的 2 分频时钟 XTAL_D2_CLK (16 MHz)
  - RC_FAST_CLK
  - PLL_LP_CLK

LP_DYN_SLOW_CLK 时钟源为 LP_SLOW_CLK，频率同时钟源频率。

LP_DYN_FAST_CLK 根据芯片电源的模式（详见章节 2 *低功耗管理 (RTC_CNTL) [to be added later]*）选择时钟源：

- 芯片电源为 Active、Modem-sleep 模式时始终选择 LP_FAST_CLK 作为时钟源
- 芯片电源为 Light-sleep、Deep-sleep 模式时选择 LP_SLOW_CLK 作为时钟源

7.2.4.3 外设时钟

表 7-4、表 7-5、表 7-6、和表 7-7 分别列出了接入各个外设时钟的衍生高性能/低功耗时钟源、接入高性能系统各个外设的时钟以及接入低功耗系统各个外设的时钟。

表 7-4. 衍生高性能时钟源

衍生时钟	源时钟			衍生时钟	源时钟					衍生时钟									源时钟
	XTAL_CLK 32 MHz	PLL_CLK PLL_F96M_CLK 96 MHz	PLL_CLK PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz	RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/ 32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
PLL_F48M_CLK		Y																	
HP_ROOT_CLK	Y	Y	Y		Y														
CRYPTO_CLK	Y	Y	Y		Y														
APB_CLK										Y									
AHB_CLK										Y									
CPU_CLK										Y									

表 7-5. 高性能系统外设时钟

外设	源时钟			衍生时钟	源时钟					衍生时钟									源时钟
	XTAL_CLK 32 MHz	PLL_CLK PLL_F96M_CLK 96 MHz	PLL_CLK PLL_F64M_CLK 64 MHz	PLL_F48M_CLK 48 MHz	RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/ 32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
定时器组 (TIMG)	Y			Y	Y														
主系统看门狗定时器 (MWDT)	Y			Y	Y														
I2S 控制器 (I2S)	Y	Y	Y																I2S_MCLK_in
UART 控制器 (UART)	Y			Y	Y														
红外遥控 (RMT)	Y				Y														
电机控制脉宽调制器 (MCPWM)	Y	Y			Y														
I2C 控制器 (I2C)	Y				Y														
SPI2	Y			Y	Y														
SAR ADC	Y	Y			Y														
USB 串口/ JTAG 控制器 (USB_SERIAL_JTAG)				Y															
双线汽车接口 (TWAI)	Y				Y														
LED PWM 控制器 (LEDC)	Y	Y			Y														
系统定时器 (SYSTIMER)	Y				Y														
并行 IO 控制器 (PARL_IO)	Y	Y			Y														parl_rx_clk_in parl_tx_clk_in
IO MUX	Y			Y	Y														
AES 加速器 (AES) SHA 加速器 (SHA) RSA 加速器 (RSA) ECC 加速器 (ECC) 数字签名算法 (DSA) HMAC 加速器 (HMAC) 椭圆曲线数字签名算法 (ECDSA)											Y								
中断矩阵 (INTMTX)													Y						
脉冲计数控制器 (PCNT)												Y							
事件任务矩阵 (SOC_ETM)													Y						
通用 DMA 控制器 (GDMA)													Y						
UHCI													Y						
辅助调试 (ASSIST_DEBUG, MEM_MONITOR)														Y					
RISC-V 追踪编码器 (TRACE)													Y	Y					
中断优先级寄存器 (INTPRI)														Y					

表 7-6. 衍生低功耗时钟源

衍生时钟	源时钟				衍生时钟	源时钟					衍生时钟									源时钟
	XTAL_CLK 32 MHz	PLL_CLK			PLL_F48M_CLK 48 MHz	RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/ 32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz																	
LP_DYN_FAST_CLK							Y	Y	Y										Y	
LP_DYN_SLOW_CLK							Y	Y	Y											
XTAL_D2_CLK	Y																			
LP_FAST_CLK						Y				Y								Y		

表 7-7. 低功耗系统外设时钟

外设	源时钟				衍生时钟	源时钟					衍生时钟									源时钟
	XTAL_CLK 32 MHz	PLL_CLK		PLL_F48M_CLK 48 MHz		RC_FAST_CLK 8 MHz	RC_SLOW_CLK 130 kHz	OSC_SLOW_CLK 32 kHz	XTAL32K_CLK 32 kHz	PLL_LP_CLK 8 MHz	HP_ROOT_CLK 96 MHz/64 MHz/ 32 MHz/8 MHz	CRYPTO_CLK	APB_CLK	AHB_CLK	CPU_CLK	LP_DYN_ FAST_CLK	LP_DYN_ SLOW_CLK	XTAL_D2_CLK 16 MHz	LP_FAST_CLK	Clock from IO
		PLL_F96M_CLK 96 MHz	PLL_F64M_CLK 64 MHz																	
eFuse 控制器 (eFuse)																		Y		
RTC 看门狗定时器 (RWDT)																Y	Y			
RTC 定时器 (RTC Timer)																Y	Y			
欠压检测器 (Brownout Detector)																Y				
电源管理单元 (PMU)																Y	Y		Y	

PLL_F96M_CLK 时钟

PLL_F96M_CLK 的频率是 96 MHz。PLL_F48M_CLK (48 MHz) 为 PLL_F96M_CLK 分频所得。

CRYPTO_CLK 时钟

如表 7-4 所示，可选择 XTAL_CLK、PLL_96M_CLK、PLL_64M_CLK 或 RC_FAST_CLK 作为 CRYPTO_CLK 的时钟源，CRYPTO_CLK 时钟频率最大为 96 MHz。

为了防止 DPA (Differential Power Analysis) 攻击加解密外设，加解密功能时钟采用随机分频策略。根据随机分频范围划分为三个安全等级。可通过配置 HP_SYSTEM_SEC_DPA_CONF_REG 选择安全等级。当 HP_SYSTEM_SEC_DPA_CFG_SEL 为 1 时，安全等级由 eFuse 中的配置信息 (EFUSE_SEC_DPA_LEVEL) 决定，否则由 HP_SYSTEM_SEC_DPA_LEVEL 的值决定。

LED_PWM 时钟

LEDC 模块能将 PLL_F96M_CLK、RC_FAST_CLK 以及 XTAL_CLK 作为时钟源使用，即在 APB_CLK 关闭的时候，LEDC 也可工作。换而言之，当系统处于低功耗模式时，其它外设都将停止工作（APB_CLK 关闭），但是 LEDC 仍然可以通过 RC_FAST_CLK 来正常工作。

7.3 配置流程

7.3.1 高性能系统时钟配置

当配置 HP_ROOT_CLK 的时钟选源 PCR_SOC_CLK_SEL ，或 CPU_CLK 时钟分频 PCR_CPU_DIV_NUM ，或 AHB_CLK 时钟分频 PCR_AHB_DIV_NUM 时，在配置完上述寄存器后，需要将 PCR_BUS_CLOCK_UPDATE 寄存器写 1 来生效上述配置。可通过读 PCR_BUS_CLOCK_UPDATE 寄存器判断上述配置是否更新完成，当为 0 时表示配置更新完成，为 1 表示未更新完成。

7.3.2 低功耗系统时钟配置

可配置 LP_CLKRST_SLOW_CLK_SEL 选择 LP_SLOW_CLK 的时钟源。

可配置 LP_CLKRST_FAST_CLK_SEL 选择 LP_FAST_CLK 的时钟源。

7.3.3 外设时钟复位配置

大部分外设模块的时钟都可以分为总线时钟和功能时钟。外设模块的总线时钟主要是用于配置外设模块的寄存器。功能时钟主要是模块工作所使用的时钟，例如 UART 的参考时钟。大部分模块的功能时钟可以选择多个时钟源。在门控寄存器的描述中会说明该寄存器属于总线（AHB_CLK、APB_CLK）时钟门控寄存器还是功能时钟门控寄存器。

ESP32-H2 的总线时钟开关、功能时钟的开关以及时钟源选择和时钟分频的配置寄存器均放在了 PCR 模块中。更多信息，详见 7.4 寄存器列表。功能模块不工作时，可以配置 PCR 模块中对应寄存器关闭功能时钟，关闭模块功能时钟不会影响系统其他部分。

下文以 I2C 时钟配置为例：

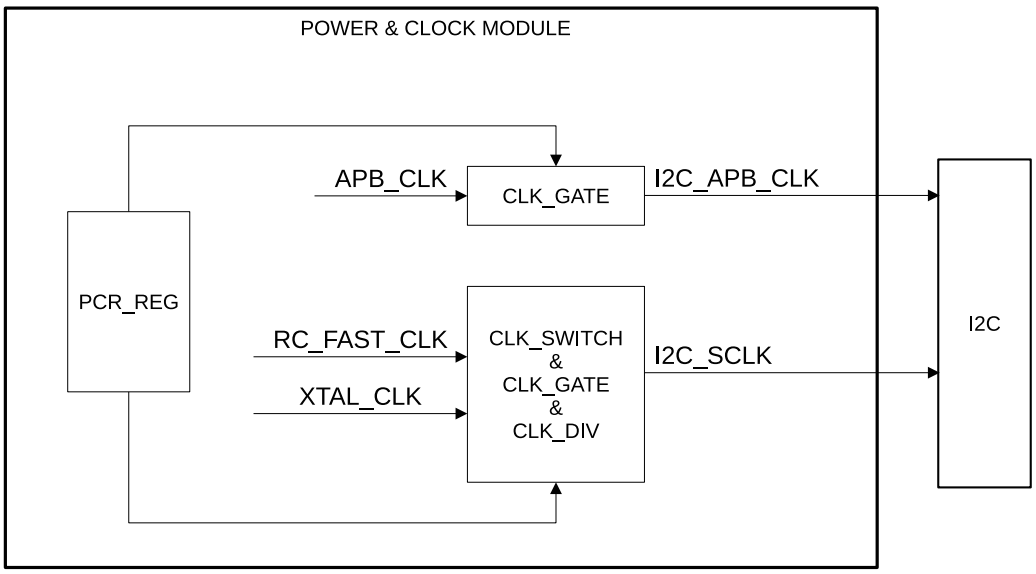


图 7-3. 时钟配置示例

图 7-3 是 I2C 的时钟结构图，其他模块的时钟结构类似。其中 CLK_SWITCH 的功能是选择一个时钟输出，CLK_GATE 的功能是打开/关闭时钟。

对于要求低功耗的场景，当模块不工作时，除了关闭功能时钟，还可以关闭模块的总线时钟来进一步降低功耗。注意，如果先关闭模块总线时钟，那么模块的功能时钟还是有可能继续工作，所以推荐先关闭功能时钟再关闭总线时钟，打开时钟时推荐先打开总线时钟再打开功能时钟。

说明：

本章节中所有分频寄存器配置的值为实际分频值减去 1。

7.4 寄存器列表

7.4.1 PCR 模块寄存器列表

本小节的所有地址均为相对于电源/时钟/复位寄存器（PCR）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PCR_UART0_CONF_REG	UART0 配置寄存器	0x0000	varies
PCR_UART0_SCLK_CONF_REG	UART0 时钟配置寄存器	0x0004	R/W
PCR_UART0_PD_CTRL_REG	UART0 电源控制寄存器	0x0008	R/W
PCR_UART1_CONF_REG	UART1 配置寄存器	0x000C	varies
PCR_UART1_SCLK_CONF_REG	UART1 时钟配置寄存器	0x0010	R/W
PCR_UART1_PD_CTRL_REG	UART1 电源控制寄存器	0x0014	R/W
PCR_I2C0_CONF_REG	I2C 配置寄存器	0x0020	varies
PCR_I2C0_SCLK_CONF_REG	I2C 时钟配置寄存器	0x0024	R/W
PCR_I2C1_CONF_REG	I2C 配置寄存器	0x0028	varies
PCR_I2C1_SCLK_CONF_REG	I2C 时钟配置寄存器	0x002C	R/W
PCR_UHCI_CONF_REG	UHCI 配置寄存器	0x0030	varies
PCR_RMT_CONF_REG	RMT 配置寄存器	0x0034	varies
PCR_RMT_SCLK_CONF_REG	RMT 时钟配置寄存器	0x0038	R/W
PCR_LEDC_CONF_REG	LEDC 配置寄存器	0x003C	varies
PCR_LEDC_SCLK_CONF_REG	LEDC 时钟配置寄存器	0x0040	R/W
PCR_TIMERGROUP0_CONF_REG	定时器组 0 配置寄存器	0x0044	varies
PCR_TIMERGROUP0_TIMER_CLK_CONF_REG	定时器组 0 通用定时器时钟配置寄存器	0x0048	R/W
PCR_TIMERGROUP0_WDT_CLK_CONF_REG	定时器组 0 WDT 时钟配置寄存器	0x004C	R/W
PCR_TIMERGROUP1_CONF_REG	定时器组 1 配置寄存器	0x0050	varies
PCR_TIMERGROUP1_TIMER_CLK_CONF_REG	定时器组 1 通用定时器时钟配置寄存器	0x0054	R/W
PCR_TIMERGROUP1_WDT_CLK_CONF_REG	定时器组 1 WDT 时钟配置寄存器	0x0058	R/W
PCR_SYSTIMER_CONF_REG	系统定时器配置寄存器	0x005C	varies
PCR_SYSTIMER_FUNC_CLK_CONF_REG	系统定时器功能时钟配置寄存器	0x0060	R/W
PCR_TWAI0_CONF_REG	TWAI0 配置寄存器	0x0064	varies
PCR_TWAI0_FUNC_CLK_CONF_REG	TWAI0 功能时钟配置寄存器	0x0068	R/W
PCR_I2S_CONF_REG	I2S 配置寄存器	0x006C	varies
PCR_I2S_TX_CLKM_CONF_REG	I2S TX 时钟配置寄存器	0x0070	R/W
PCR_I2S_TX_CLKM_DIV_CONF_REG	I2S TX 时钟分频系数配置寄存器	0x0074	R/W
PCR_I2S_RX_CLKM_CONF_REG	I2S RX 时钟配置寄存器	0x0078	R/W
PCR_I2S_RX_CLKM_DIV_CONF_REG	I2S RX 时钟分频系数配置寄存器	0x007C	R/W
PCR_SARADC_CONF_REG	SAR ADC 配置寄存器	0x0080	R/W
PCR_SARADC_CLKM_CONF_REG	SAR ADC 时钟配置寄存器	0x0084	R/W
PCR_TSENS_CLK_CONF_REG	温度传感器时钟配置寄存器	0x0088	R/W
PCR_USB_DEVICE_CONF_REG	USB Serial/JTAG 配置寄存器	0x008C	varies
PCR_INTMTX_CONF_REG	中断矩阵配置寄存器	0x0090	varies

名称	描述	地址	访问
PCR_PCNT_CONF_REG	PCNT 配置寄存器	0x0094	varies
PCR_ETM_CONF_REG	ETM 配置寄存器	0x0098	varies
PCR_PWM_CONF_REG	MCPWM 配置寄存器	0x009C	varies
PCR_PWM_CLK_CONF_REG	MCPWM 时钟配置寄存器	0x00A0	R/W
PCR_PARL_IO_CONF_REG	并行 IO 配置寄存器	0x00A4	varies
PCR_PARL_CLK_RX_CONF_REG	并行 IO RX 时钟配置寄存器	0x00A8	R/W
PCR_PARL_CLK_TX_CONF_REG	并行 IO TX 配置寄存器	0x00AC	R/W
PCR_GDMA_CONF_REG	GDMA 配置寄存器	0x00B8	R/W
PCR_SPI2_CONF_REG	SPI2 配置寄存器	0x00BC	varies
PCR_SPI2_CLKM_CONF_REG	SPI2 时钟配置寄存器	0x00C0	R/W
PCR_AES_CONF_REG	AES 配置寄存器	0x00C4	varies
PCR_SHA_CONF_REG	SHA 配置寄存器	0x00C8	varies
PCR_RSA_CONF_REG	RSA 配置寄存器	0x00CC	varies
PCR_RSA_PD_CTRL_REG	RSA 电源控制寄存器	0x00D0	R/W
PCR_ECC_CONF_REG	ECC 配置寄存器	0x00D4	varies
PCR_ECC_PD_CTRL_REG	ECC 电源控制寄存器	0x00D8	R/W
PCR_DS_CONF_REG	DS 配置寄存器	0x00DC	varies
PCR_HMAC_CONF_REG	HMAC 配置寄存器	0x00E0	varies
PCR_ECDSA_CONF_REG	ECDSA 配置寄存器	0x00E4	varies
PCR_IOMUX_CONF_REG	IO MUX 配置寄存器	0x00E8	R/W
PCR_IOMUX_CLK_CONF_REG	IO MUX 时钟配置寄存器	0x00EC	R/W
PCR_MEM_MONITOR_CONF_REG	内存访问监控配置寄存器	0x00F0	varies
PCR_TRACE_CONF_REG	RISC-V 追踪编码器配置寄存器	0x00F8	R/W
PCR_ASSIST_CONF_REG	辅助调试模块配置寄存器	0x00FC	R/W
PCR_CACHE_CONF_REG	Cache 配置寄存器	0x0100	R/W
PCR_TIMEOUT_CONF_REG	超时保护模块配置寄存器	0x0108	R/W
PCR_SYSCLK_CONF_REG	系统时钟配置寄存器	0x010C	varies
PCR_CPU_WAITI_CONF_REG	CPU 等待中断模式时钟门控配置寄存器	0x0110	R/W
PCR_CPU_FREQ_CONF_REG	CPU_CLK 频率配置寄存器	0x0114	R/W
PCR_AHB_FREQ_CONF_REG	AHB_CLK 频率配置寄存器	0x0118	R/W
PCR_APB_FREQ_CONF_REG	APB_CLK 频率配置寄存器	0x011C	R/W
PCR_PLL_DIV_CLK_EN_REG	PLL 分频时钟门控配置寄存器	0x0124	R/W
PCR_CTRL_TICK_CONF_REG	时钟分频系数配置寄存器	0x012C	R/W
PCR_CTRL_32K_CONF_REG	32 kHz 时钟配置寄存器	0x0130	R/W
PCR_SRAM_POWER_CONF_0_REG	HP SRAM/ROM 配置寄存器	0x0134	R/W
PCR_SRAM_POWER_CONF_1_REG	HP SRAM/ROM 配置寄存器	0x0138	R/W
PCR_SEC_CONF_REG	片外存储器加密与解密模块时钟源配置寄存器	0x013C	R/W
PCR_BUS_CLK_UPDATE_REG	高性能系统时钟源更新使能寄存器	0x0148	R/ W/ WTC
PCR_SAR_CLK_DIV_REG	SAR ADC 时钟分频系数配置寄存器	0x014C	R/W
频率数据寄存器			

名称	描述	地址	访问
PCR_SYSCLK_FREQ_QUERY_0_REG	系统时钟频率轮询寄存器 0	0x0120	HRO
版本寄存器			
PCR_DATE_REG	版本控制寄存器	0x0FFC	R/W

7.4.2 低功耗系统时钟寄存器列表

本小节前缀为 LPPERI 的寄存器（最后两个）地址为相对于低功耗外设寄存器（LPPERI）基地址的地址偏移量（相对地址），前缀为 LP_AON 的寄存器（倒数第三、四个）地址为相对于低功耗常开寄存器（LP_AON）基地址的地址偏移量（相对地址），其余所有地址均为相对于低功耗时钟复位寄存器（LP_CLKRST）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

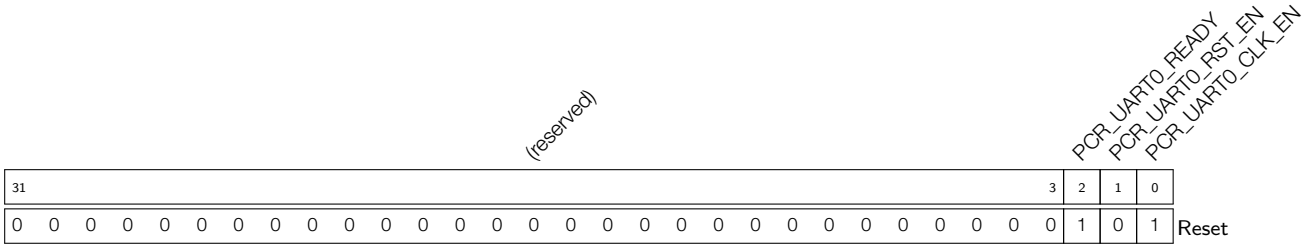
名称	描述	地址	访问
配置寄存器			
LP_CLKRST_LP_CLK_CONF_REG	LP 系统时钟源配置寄存器	0x0000	R/W
LP_CLKRST_LP_CLK_PO_EN_REG	管脚时钟信号门控配置寄存器	0x0004	R/W
LP_CLKRST_LP_CLK_EN_REG	LP 时钟源门控配置寄存器	0x0008	R/W
LP_CLKRST_LP_RST_EN_REG	LP 外设软件复位配置寄存器	0x000C	R/W
LP_CLKRST_RESET_CAUSE_REG	复位原因状态寄存器	0x0010	varies
LP_CLKRST_CPU_RESET_REG	CPU 复位配置寄存器	0x0014	R/W
LP_CLKRST_FOSC_CNTL_REG	RC_FAST_CLK 时钟频率配置寄存器	0x0018	R/W
LP_CLKRST_CLK_TO_HP_REG	HP 系统时钟信号门控配置寄存器	0x0020	R/W
LP_CLKRST_LPMEM_FORCE_REG	LP 存储器强制开启时钟门控配置寄存器	0x0024	R/W
LP_CLKRST_LPPERI_REG	LP 外设时钟配置寄存器	0x0028	R/W
LP_CLKRST_XTAL32K_REG	XTAL32K 配置寄存器	0x002C	R/W
LP_CLKRST_DATE_REG	版本控制寄存器	0x03FC	R/W
LP_AON_SYS_CFG_REG	软件系统复位寄存器	0x0034	WT
LP_AON_CPUCORE0_CFG_REG	软件 CPU 复位寄存器	0x0038	WT
LPPERI_CLK_EN_REG	eFuse 时钟门控配置寄存器	0x0000	R/W
LPPERI_RESET_EN_REG	eFuse 复位配置寄存器	0x0004	R/W

7.5 寄存器

7.5.1 PCR 模块寄存器

本小节的所有地址均为相对于电源/时钟/复位寄存器（PCR）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 7.1. PCR_UART0_CONF_REG (0x0000)



PCR_UART0_CLK_EN 配置是否使能 UART0 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_UART0_RST_EN 配置是否复位 UART0 模块。

0：不复位

1：复位

(R/W)

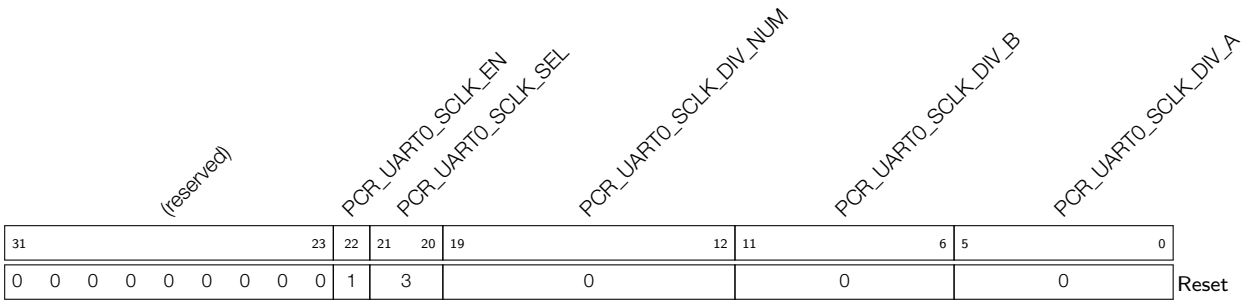
PCR_UART0_READY 表示 UART0 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.2. PCR_UART0_SCLK_CONF_REG (0x0004)



PCR_UART0_SCLK_DIV_A 配置 UART0 功能时钟分频系数小数部分的分母。(R/W)

PCR_UART0_SCLK_DIV_B 配置 UART0 功能时钟分频系数小数部分的分子。(R/W)

PCR_UART0_SCLK_DIV_NUM 配置 UART0 功能时钟分频系数的整数部分。(R/W)

PCR_UART0_SCLK_SEL 配置 UART0 时钟源。

0：不选择任何时钟

1：选择 PLL_F80M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3（默认）：选择 XTAL_CLK 时钟

(R/W)

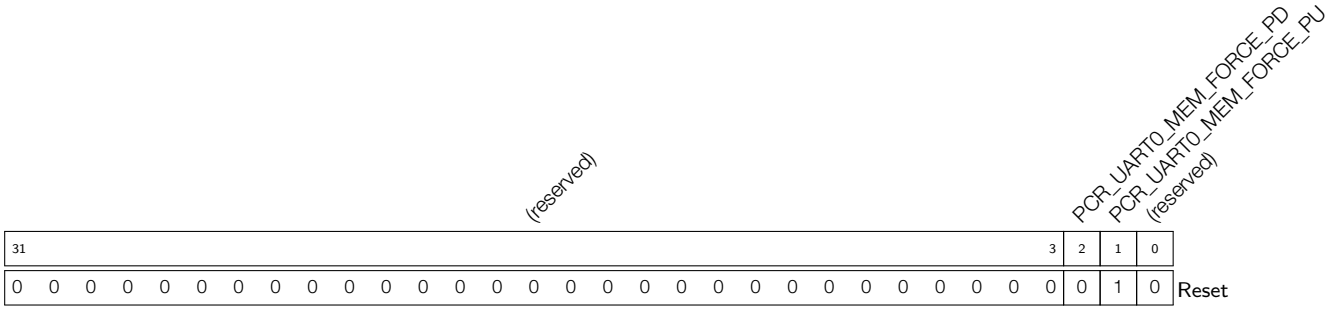
PCR_UART0_SCLK_EN 配置是否使能 UART0 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.3. PCR_UART0_PD_CTRL_REG (0x0008)



PCR_UART0_MEM_FORCE_PU 配置是否强制 UART0 存储器上电。

0：不强制 UART0 存储器上电

1：强制 UART0 存储器上电

(R/W)

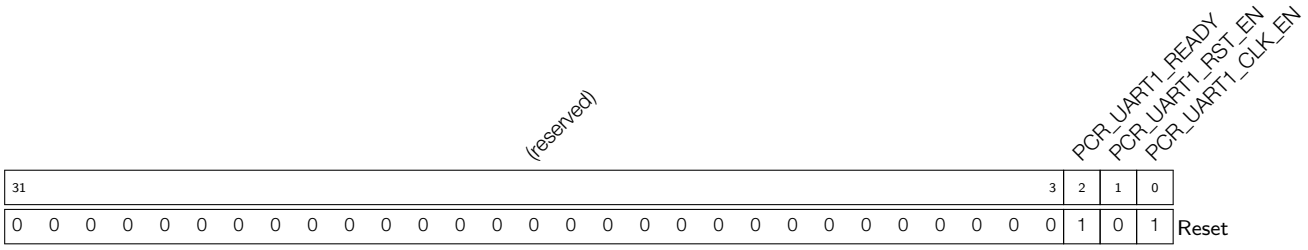
PCR_UART0_MEM_FORCE_PD 配置是否强制 UART0 存储器掉电。

0：不强制 UART0 存储器掉电

1：强制 UART0 存储器掉电

(R/W)

Register 7.4. PCR_UART1_CONF_REG (0x000C)



PCR_UART1_CLK_EN 配置是否使能 UART1 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_UART1_RST_EN 配置是否复位 UART1 模块。

0：不复位

1：复位

(R/W)

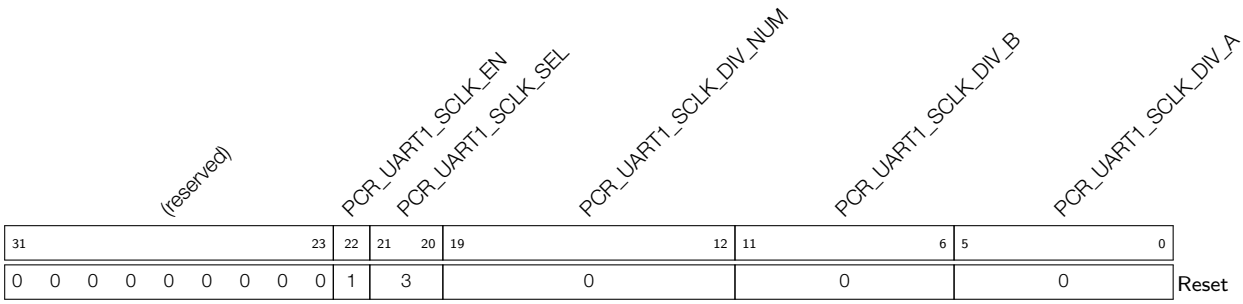
PCR_UART1_READY 表示 UART1 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.5. PCR_UART1_SCLK_CONF_REG (0x0010)



PCR_UART1_SCLK_DIV_A 配置 UART1 功能时钟分频系数小数部分的分母。(R/W)

PCR_UART1_SCLK_DIV_B 配置 UART1 功能时钟分频系数小数部分的分子。(R/W)

PCR_UART1_SCLK_DIV_NUM 配置 UART1 功能时钟分频系数的整数部分。(R/W)

PCR_UART1_SCLK_SEL 配置选择 UART1 时钟源。

0：不选择任何时钟

1：选择 PLL_F80M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3（默认）：选择 XTAL_CLK 时钟

(R/W)

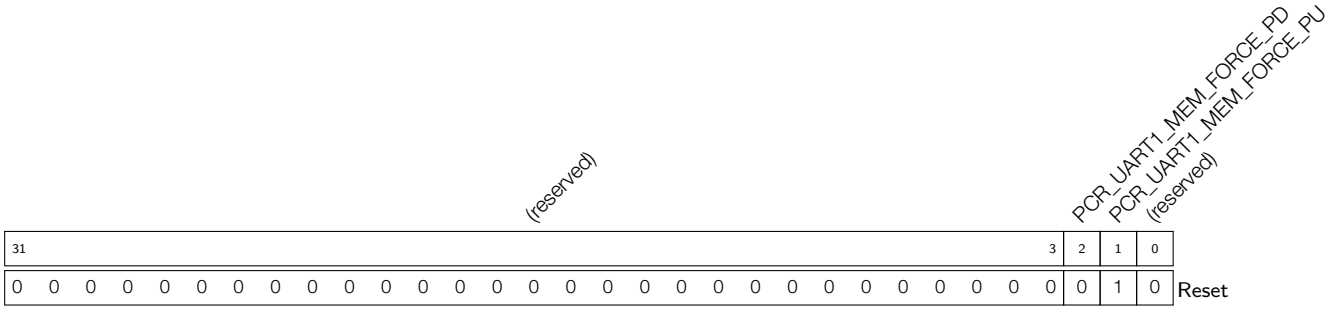
PCR_UART1_SCLK_EN 配置是否使能 UART1 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.6. PCR_UART1_PD_CTRL_REG (0x0014)



PCR_UART1_MEM_FORCE_PU 配置是否强制 UART1 存储器上电。

0：不强制 UART1 存储器上电

1：强制 UART1 存储器上电

(R/W)

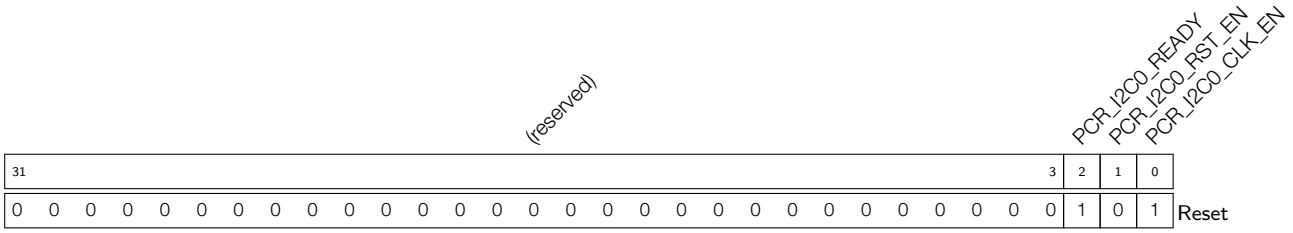
PCR_UART1_MEM_FORCE_PD 配置是否强制 UART1 存储器掉电。

0：不强制 UART1 存储器掉电

1：强制 UART1 存储器掉电

(R/W)

Register 7.7. PCR_I2C0_CONF_REG (0x0020)



PCR_I2C0_CLK_EN 配置是否使能 I2C0 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_I2C0_RST_EN 配置是否复位 I2C0 模块。

0：不复位

1：复位

(R/W)

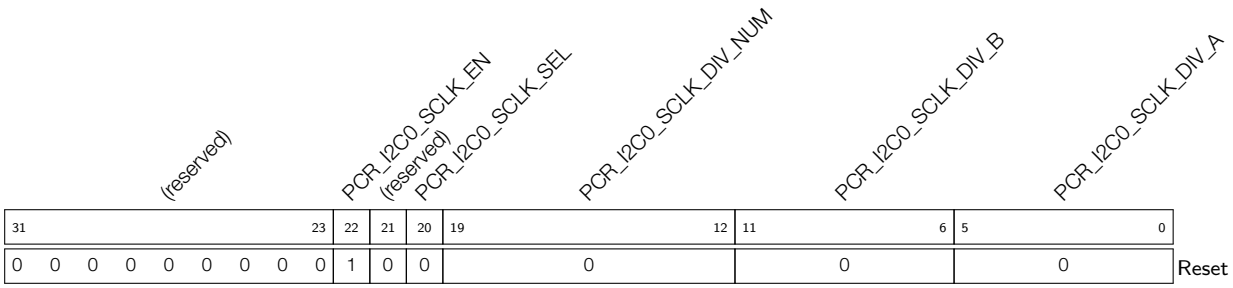
PCR_I2C0_READY 表示 I2C0 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.8. PCR_I2C0_SCLK_CONF_REG (0x0024)



PCR_I2C0_SCLK_DIV_A 配置 I2C0 功能时钟分频系数小数部分的分母。(R/W)

PCR_I2C0_SCLK_DIV_B 配置 I2C0 功能时钟分频系数小数部分的分子。(R/W)

PCR_I2C0_SCLK_DIV_NUM 配置 I2C0 功能时钟分频系数的整数部分。(R/W)

PCR_I2C0_SCLK_SEL 配置选择 I2C0 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

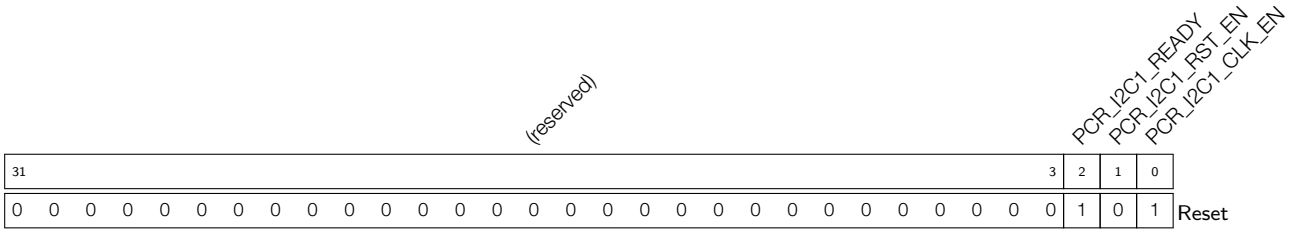
PCR_I2C0_SCLK_EN 配置是否使能 I2C0 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.9. PCR_I2C1_CONF_REG (0x0028)



PCR_I2C1_CLK_EN 配置是否使能 I2C1 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_I2C1_RST_EN 配置是否复位 I2C1 模块。

0：不复位

1：复位

(R/W)

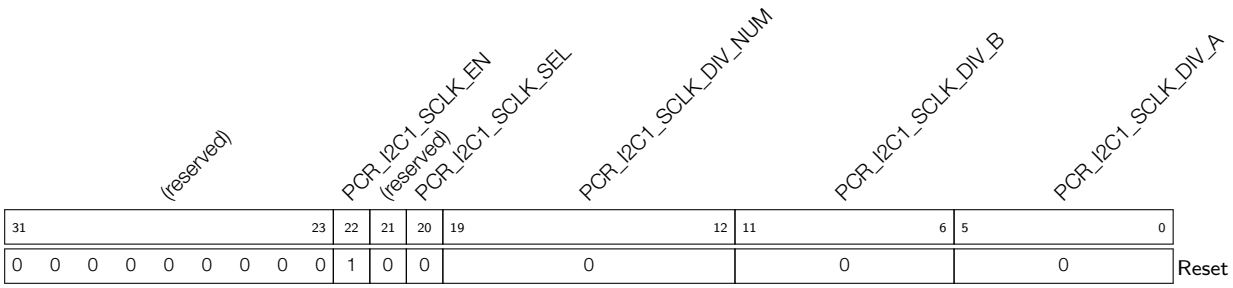
PCR_I2C1_READY 表示 I2C1 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.10. PCR_I2C1_SCLK_CONF_REG (0x002C)



PCR_I2C1_SCLK_DIV_A 配置 I2C1 功能时钟分频系数小数部分的分母。(R/W)

PCR_I2C1_SCLK_DIV_B 配置 I2C1 功能时钟分频系数小数部分的分子。(R/W)

PCR_I2C1_SCLK_DIV_NUM 配置 I2C1 功能时钟分频系数的整数部分。(R/W)

PCR_I2C1_SCLK_SEL 配置选择 I2C1 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

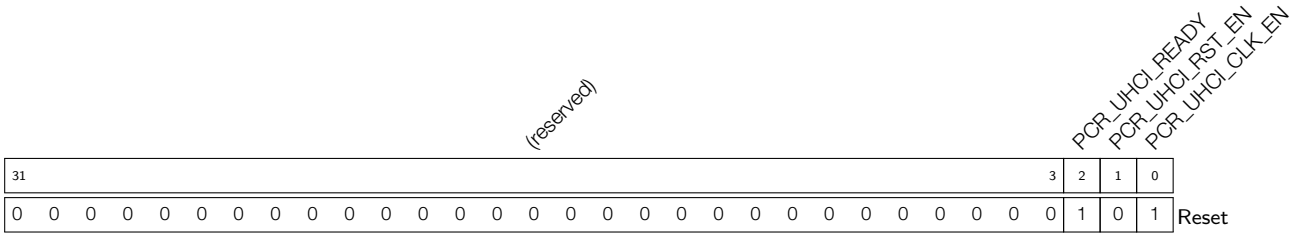
PCR_I2C1_SCLK_EN 配置是否使能 I2C1 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.11. PCR_UHCI_CONF_REG (0x0030)



PCR_UHCI_CLK_EN 配置是否使能 UHCI 时钟。

0：不使能

1：使能

(R/W)

PCR_UHCI_RST_EN 配置是否复位 UHCI 模块。

0：不复位

1：复位

(R/W)

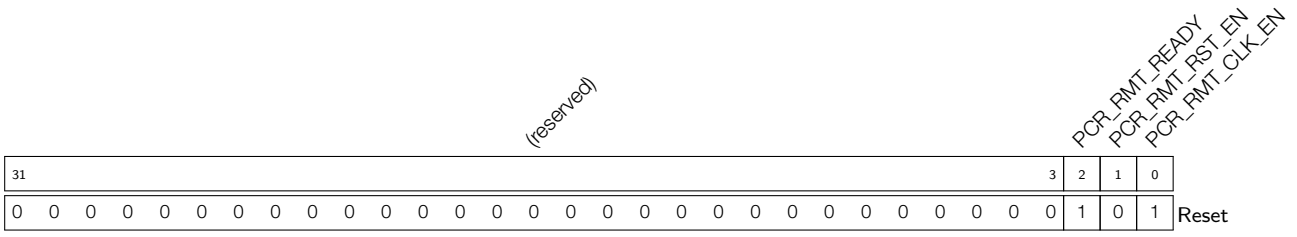
PCR_UHCI_READY 表示 UHCI 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.12. PCR_RMT_CONF_REG (0x0034)



PCR_RMT_CLK_EN 配置是否使能 RMT APB_CLK。

0：使能

1：不使能

(R/W)

PCR_RMT_RST_EN 配置是否复位 RMT 模块。

0：不复位

1：复位

(R/W)

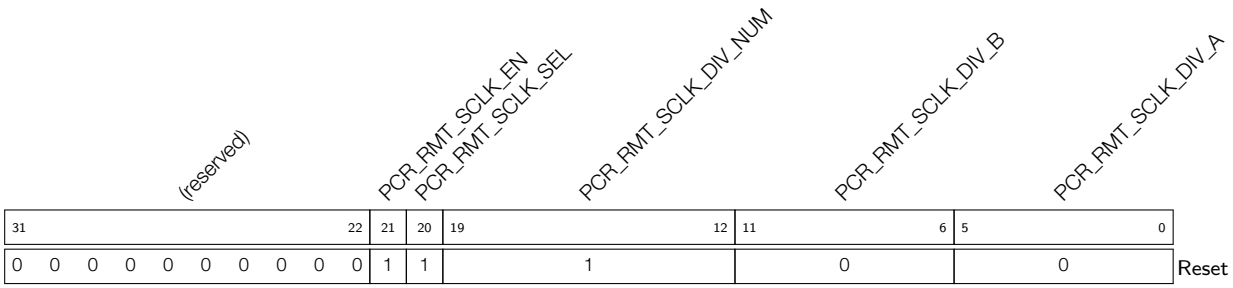
PCR_RMT_READY 表示 RMT 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.13. PCR_RMT_SCLK_CONF_REG (0x0038)



PCR_RMT_SCLK_DIV_A 配置 RMT 功能时钟分频系数小数部分的分母。(R/W)

PCR_RMT_SCLK_DIV_B 配置 RMT 功能时钟分频系数小数部分的分子。(R/W)

PCR_RMT_SCLK_DIV_NUM 配置 RMT 功能时钟分频系数的整数部分。(R/W)

PCR_RMT_SCLK_SEL 配置选择 RMT 时钟源。

0：选择 XTAL_CLK 时钟

1（默认）：选择 RC_FAST_CLK 时钟

(R/W)

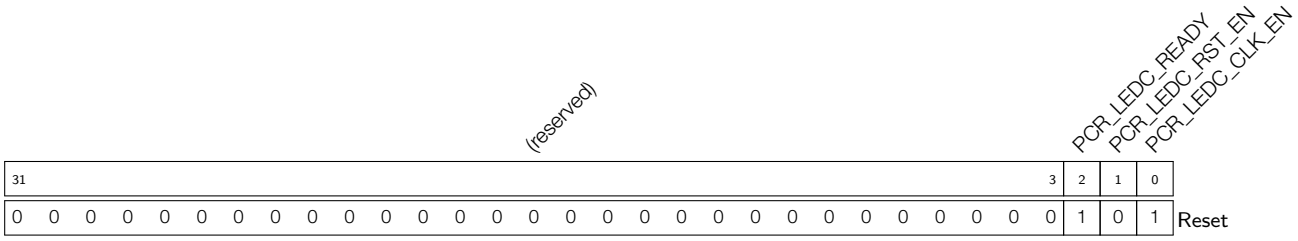
PCR_RMT_SCLK_EN 配置是否使能 RMT 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.14. PCR_LEDC_CONF_REG (0x003C)



PCR_LEDC_CLK_EN 配置是否使能 LEDC APB_CLK。

0：不使能

1：使能

(R/W)

PCR_LEDC_RST_EN 配置是否复位 LEDC 模块。

0：不复位

1：复位

(R/W)

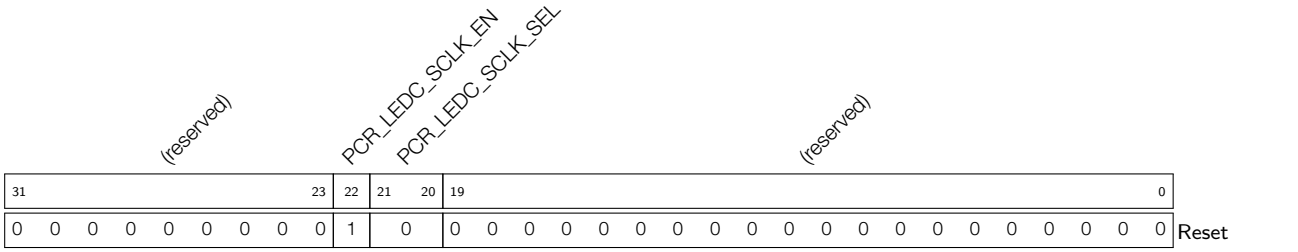
PCR_LEDC_READY 表示 LEDC 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.15. PCR_LEDC_SCLK_CONF_REG (0x0040)



PCR_LEDC_SCLK_SEL 配置选择 LEDC 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F96M_CLK 时钟

3：不选择任何时钟

(R/W)

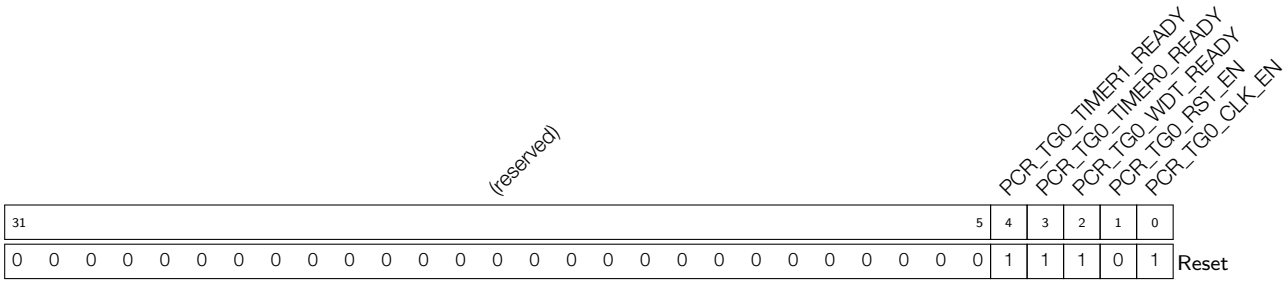
PCR_LEDC_SCLK_EN 配置是否使能 LEDC 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.16. PCR_TIMERGROUP0_CONF_REG (0x0044)



PCR_TG0_CLK_EN 配置是否使能定时器组 0 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TG0_RST_EN 配置是否复位定时器组 0 模块。

0：不复位

1：复位

(R/W)

PCR_TG0_WDT_READY 表示定时器组 0 中的 WDT 是否解复位完成。

0：未完成

1：完成

(RO)

PCR_TG0_TIMER0_READY 表示定时器组 0 中的定时器 0 是否解复位完成。

0：未完成

1：完成

(RO)

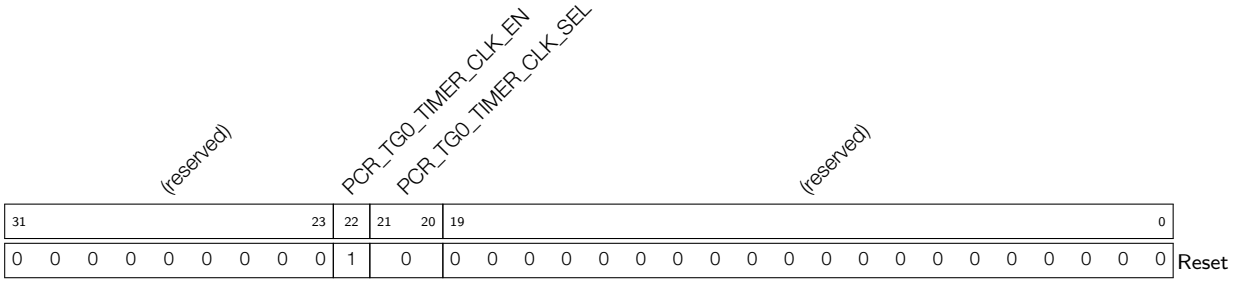
PCR_TG0_TIMER1_READY 表示定时器组 0 中的定时器 1 是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.17. PCR_TIMERGROUP0_TIMER_CLK_CONF_REG (0x0048)



PCR_TG0_TIMER_CLK_SEL 配置选择定时器组 0 通用定时器的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F48M_CLK 时钟

3：不选择任何时钟

(R/W)

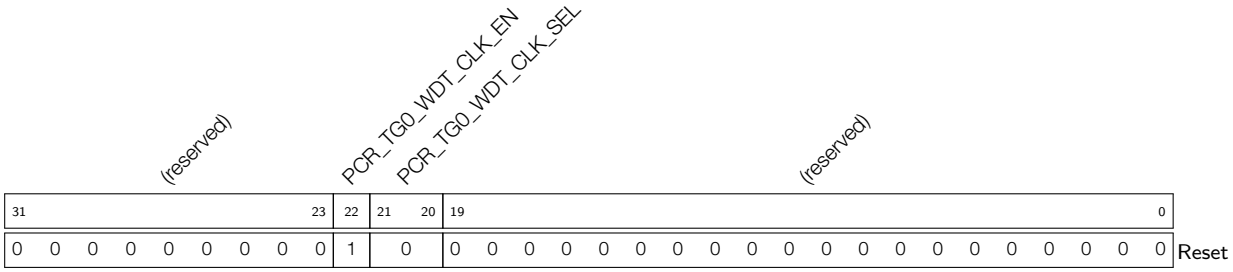
PCR_TG0_TIMER_CLK_EN 配置是否使能定时器组 0 通用定时器时钟。

0：不使能

1：使能

(R/W)

Register 7.18. PCR_TIMERGROUP0_WDT_CLK_CONF_REG (0x004C)



PCR_TG0_WDT_CLK_SEL 配置选择定时器组 0 中 WDT 的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F48M_CLK 时钟

3：不选择任何时钟

(R/W)

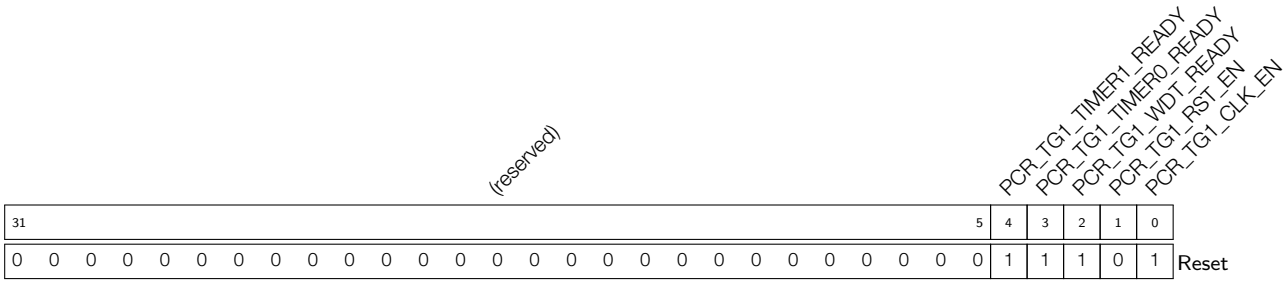
PCR_TG0_WDT_CLK_EN 配置是否使能定时器组 0 中 WDT 的时钟。

0：不使能

1：使能

(R/W)

Register 7.19. PCR_TIMERGROUP1_CONF_REG (0x0050)



PCR_TG1_CLK_EN 配置是否使能定时器组 1 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TG1_RST_EN 配置是否复位定时器组 1 模块。

0：不复位

1：复位

(R/W)

PCR_TG1_WDT_READY 表示定时器组 1 中的看门狗定时器是否解复位完成。

0：未完成

1：完成

(RO)

PCR_TG1_TIMER0_READY 表示定时器组 1 中的定时器 0 是否解复位完成。

0：未完成

1：完成

(RO)

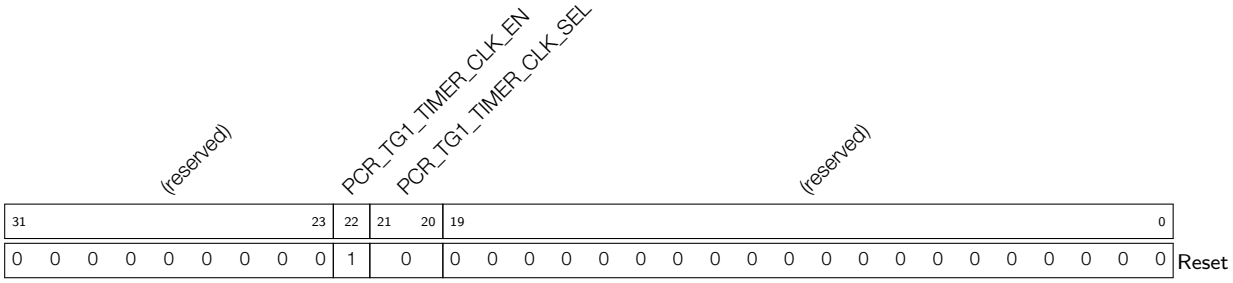
PCR_TG1_TIMER1_READY 表示定时器组 1 中的定时器 1 是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.20. PCR_TIMERGROUP1_TIMER_CLK_CONF_REG (0x0054)



PCR_TG1_TIMER_CLK_SEL 配置选择定时器组 1 通用定时器的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F48M_CLK 时钟

3：不选择任何时钟

(R/W)

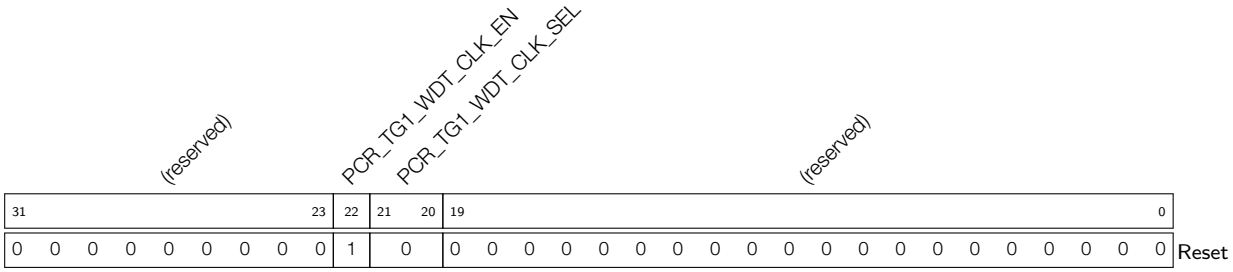
PCR_TG1_TIMER_CLK_EN 配置是否使能定时器组 1 通用定时器的时钟。

0：不使能

1：使能

(R/W)

Register 7.21. PCR_TIMERGROUP1_WDT_CLK_CONF_REG (0x0058)



PCR_TG1_WDT_CLK_SEL 配置选择定时器组 1 WDT 的时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F48M_CLK 时钟

3：不选择任何时钟

(R/W)

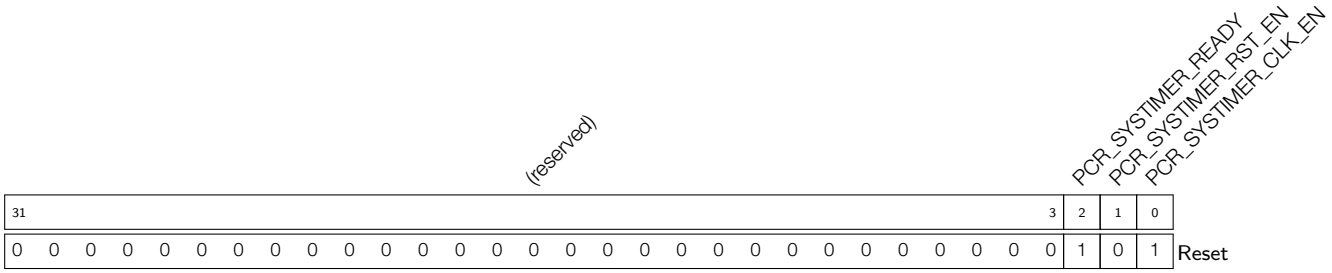
PCR_TG1_WDT_CLK_EN 配置是否使能定时器组 1 WDT 的时钟。

0：不使能

1：使能

(R/W)

Register 7.22. PCR_SYSTIMER_CONF_REG (0x005C)



PCR_SYSTIMER_CLK_EN 配置是否使能系统定时器 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_SYSTIMER_RST_EN 配置是否复位系统定时器模块。

0：不复位

1：复位

(R/W)

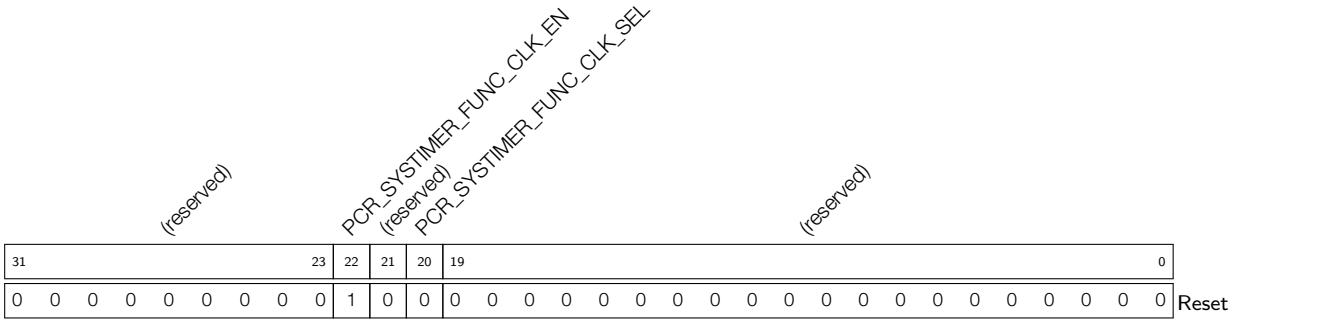
PCR_SYSTIMER_READY 表示系统定时器模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.23. PCR_SYSTIMER_FUNC_CLK_CONF_REG (0x0060)



PCR_SYSTIMER_FUNC_CLK_SEL 配置选择系统定时器时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

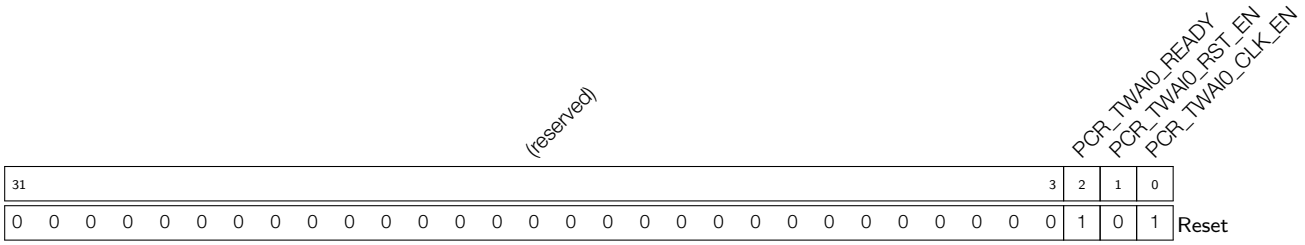
PCR_SYSTIMER_FUNC_CLK_EN 配置是否使能系统定时器功能时钟。

0：不使能

1：使能

(R/W)

Register 7.24. PCR_TWAI0_CONF_REG (0x0064)



PCR_TWAI0_CLK_EN 配置是否使能 TWAI0 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_TWAI0_RST_EN 配置是否复位 TWAI0 模块。

0：不复位

1：复位

(R/W)

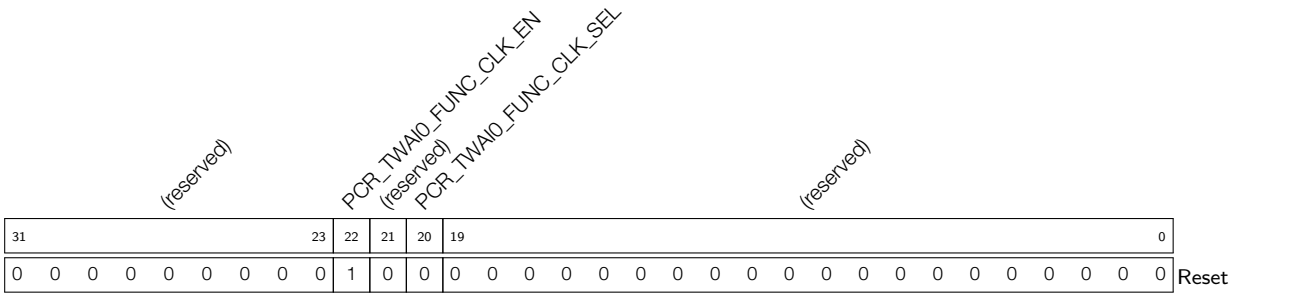
PCR_TWAI0_READY 表示 TWAI0 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.25. PCR_TWAI0_FUNC_CLK_CONF_REG (0x0068)



PCR_TWAI0_FUNC_CLK_SEL 配置选择 TWAI0 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

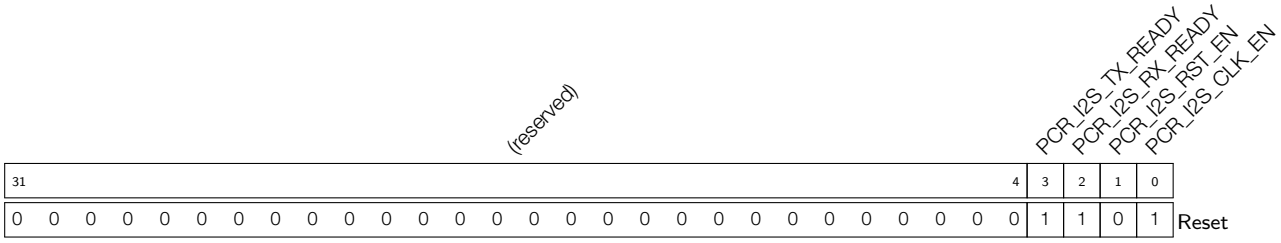
PCR_TWAI0_FUNC_CLK_EN 配置是否使能 TWAI0 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.26. PCR_I2S_CONF_REG (0x006C)



PCR_I2S_CLK_EN 配置是否使能 I2S APB_CLK。

0：不使能

1：使能

(R/W)

PCR_I2S_RST_EN 配置是否复位 I2S 模块。

0：不复位

1：复位

(R/W)

PCR_I2S_RX_READY 表示 I2S RX 模块是否解复位完成。

0：未完成

1：完成

(RO)

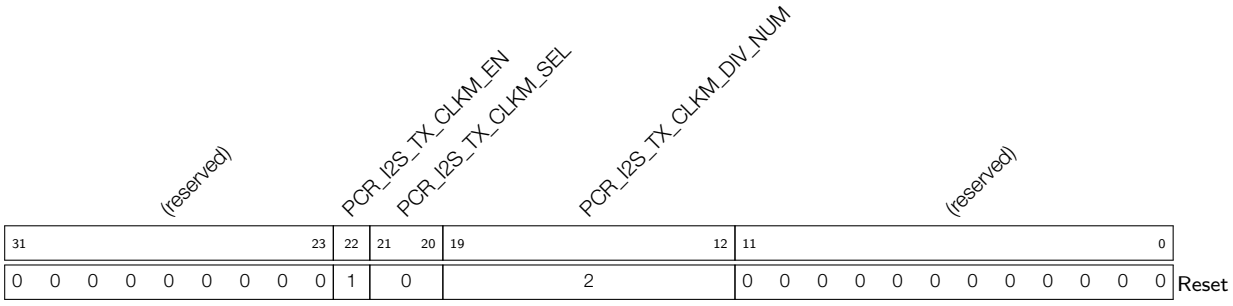
PCR_I2S_TX_READY 表示 I2S TX 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.27. PCR_I2S_TX_CLKM_CONF_REG (0x0070)



PCR_I2S_TX_CLKM_DIV_NUM 配置 I2S TX 时钟分频系数的整数部分。

(R/W)

PCR_I2S_TX_CLKM_SEL 配置选择 I2S TX 模块的时钟源。

0：选择 XTAL_CLK 时钟

1：选择 PLL_F96M_CLK 时钟

2：选择 PLL_F64M_CLK 时钟

3：选择 I2S_MCLK_in 时钟

(R/W)

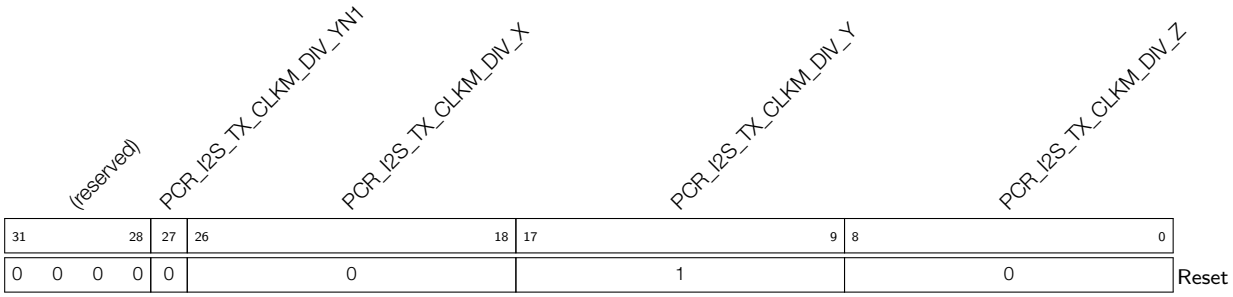
PCR_I2S_TX_CLKM_EN 配置是否使能 I2S TX 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.28. PCR_I2S_TX_CLKM_DIV_CONF_REG (0x0074)



PCR_I2S_TX_CLKM_DIV_Z b <= a/2 时，I2S_TX_CLKM_DIV_Z 的值为 b。b > a/2 时，I2S_TX_CLKM_DIV_Z 的值为 a - b。(R/W)

PCR_I2S_TX_CLKM_DIV_Y b <= a/2 时，I2S_TX_CLKM_DIV_Y 的值为 a%b。b > a/2 时，I2S_TX_CLKM_DIV_Y 的值为 a%(a - b)。(R/W)

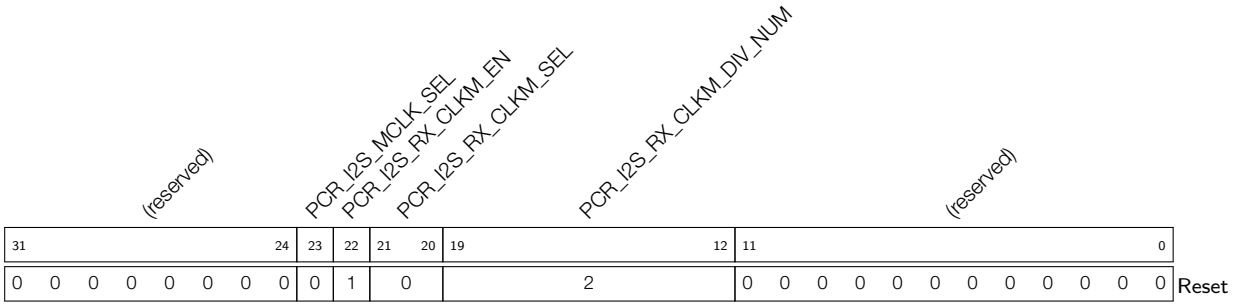
PCR_I2S_TX_CLKM_DIV_X b <= a/2 时，I2S_TX_CLKM_DIV_X 的值为 floor(a/b) - 1。b > a/2，I2S_TX_CLKM_DIV_X 的值为 floor(a/(a - b)) - 1。(R/W)

PCR_I2S_TX_CLKM_DIV_YN1 b <= a/2 时，I2S_TX_CLKM_DIV_YN1 的值为 0。b > a/2 时，I2S_TX_CLKM_DIV_YN1 的值为 1。(R/W)

说明：

上文所述的 “a” 和 “b” 分别为小数分频的分母部分和分子部分。更多信息，见章节 *I2S 控制器 (I2S)* > 章节 28.6。

Register 7.29. PCR_I2S_RX_CLKM_CONF_REG (0x0078)



PCR_I2S_RX_CLKM_DIV_NUM 配置 I2S 时钟的整数分频系数。(R/W)

PCR_I2S_RX_CLKM_SEL 配置选择 I2S RX 模块的时钟源。

0：选择 XTAL_CLK 时钟

1：选择 PLL_F96M_CLK 时钟

2：选择 PLL_F64M_CLK 时钟

3：选择 I2S_MCLK_in 时钟

(R/W)

PCR_I2S_RX_CLKM_EN 配置是否使能 I2S RX 功能时钟。

0：不使能

1：使能

(R/W)

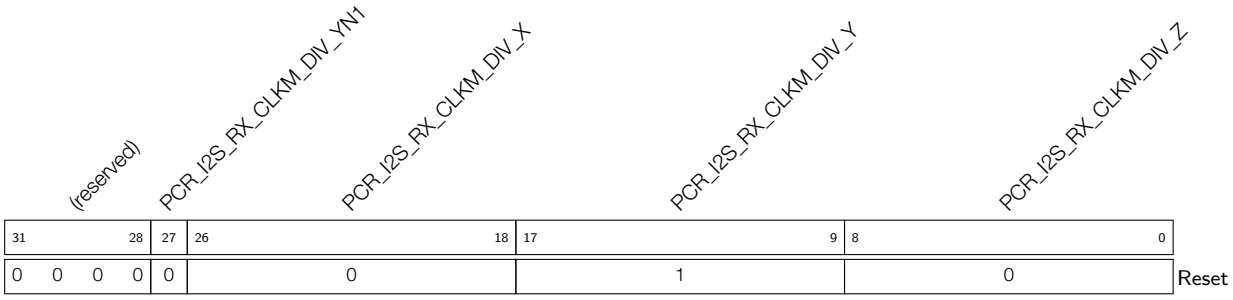
PCR_I2S_MCLK_SEL 配置选择主时钟。

0（默认）：选择 I2S_RX_CLK

1：选择 I2S_TX_CLK

(R/W)

Register 7.30. PCR_I2S_RX_CLKM_DIV_CONF_REG (0x007C)



PCR_I2S_RX_CLKM_DIV_Z b <= a/2 时，I2S_RX_CLKM_DIV_Z 的值为 b。b > a/2 时，I2S_RX_CLKM_DIV_Z 的值为 a - b。

(R/W)

PCR_I2S_RX_CLKM_DIV_Y b <= a/2 时，I2S_RX_CLKM_DIV_Y 的值为 a%b。b > a/2 时，I2S_RX_CLKM_DIV_Y 的值为 a%(a-b)。

(R/W)

PCR_I2S_RX_CLKM_DIV_X b <= a/2 时，I2S_RX_CLKM_DIV_X 的值为 floor(a/b) - 1。b > a/2，I2S_RX_CLKM_DIV_X 的值为 floor(a/(a - b)) - 1。

(R/W)

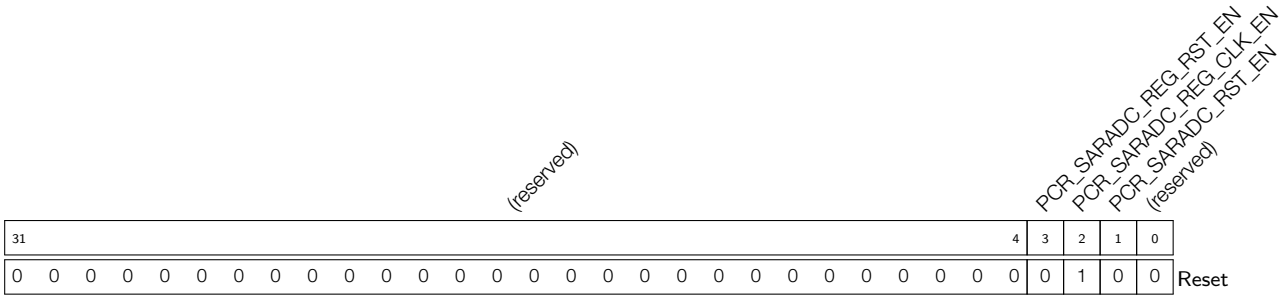
PCR_I2S_RX_CLKM_DIV_YN1 b <= a/2 时，I2S_RX_CLKM_DIV_YN1 的值为 0。b > a/2 时，I2S_RX_CLKM_DIV_YN1 的值为 1。

(R/W)

说明：

上文所述的 “a” 和 “b” 分别为小数分频的分母部分和分子部分。更多信息，见章节 28.6 。

Register 7.31. PCR_SARADC_CONF_REG (0x0080)



PCR_SARADC_RST_EN 配置是否复位 SAR ADC 功能寄存器。

0：不复位

1：复位

(R/W)

PCR_SARADC_REG_CLK_EN 配置是否使能 SAR ADC APB_CLK。

0：不使能

1：使能

(R/W)

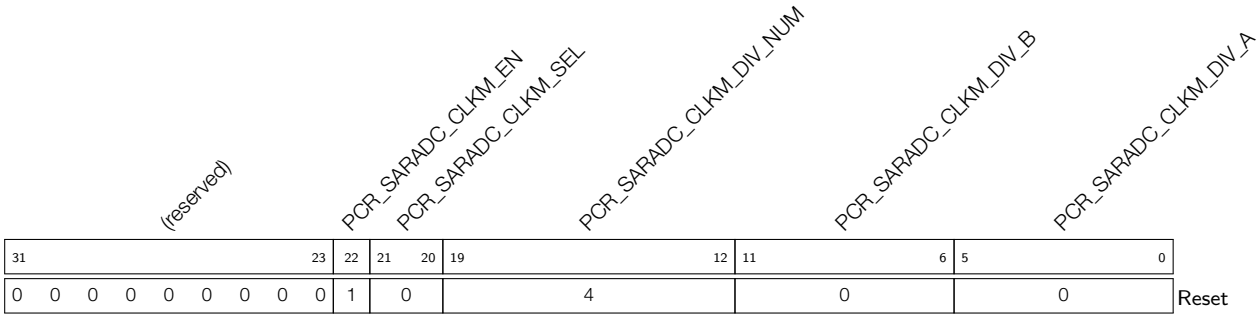
PCR_SARADC_REG_RST_EN 配置是否复位 SAR ADC 模块的 APB 寄存器。

0：不复位

1：复位

(R/W)

Register 7.32. PCR_SARADC_CLKM_CONF_REG (0x0084)



PCR_SARADC_CLKM_DIV_A 配置 SAR ADC 功能时钟分频系数小数部分的分母。(R/W)

PCR_SARADC_CLKM_DIV_B 配置 SAR ADC 功能时钟分频系数小数部分的分子。(R/W)

PCR_SARADC_CLKM_DIV_NUM 配置 SAR ADC 功能时钟分频系数的整数部分。(R/W)

PCR_SARADC_CLKM_SEL 配置选择 SAR ADC 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 PLL_F96M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3：不选择任何时钟

(R/W)

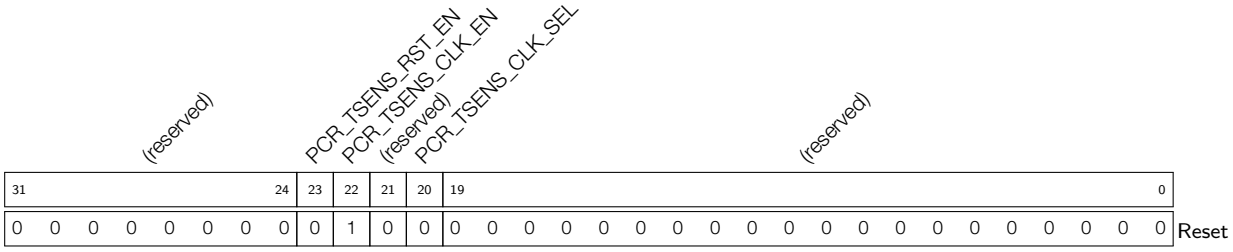
PCR_SARADC_CLKM_EN 配置是否使能 SAR ADC 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.33. PCR_TSENS_CLK_CONF_REG (0x0088)



PCR_TSENS_CLK_SEL 配置选择温度传感器时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

(R/W)

PCR_TSENS_CLK_EN 配置是否使能温度传感器时钟。

0：不使能

1：使能

(R/W)

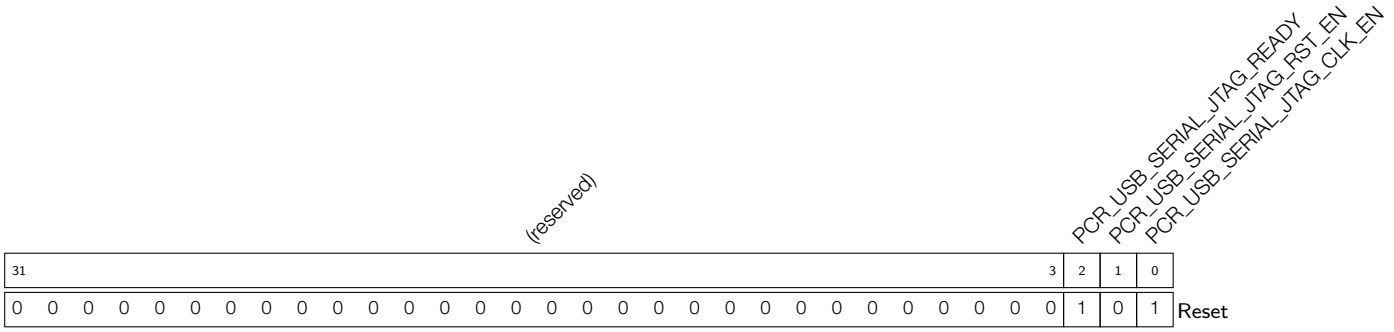
PCR_TSENS_RST_EN 配置是否复位温度传感器模块。

0：不复位

1：复位

(R/W)

Register 7.34. PCR_USB_SERIAL_JTAG_CONF_REG (0x008C)



PCR_USB_SERIAL_JTAG_CLK_EN 配置是否使能 USB Serial/JTAG 时钟。

0：不使能

1：使能

(R/W)

PCR_USB_SERIAL_JTAG_RST_EN 配置是否复位 USB Serial/JTAG 模块。

0：不复位

1：复位

(R/W)

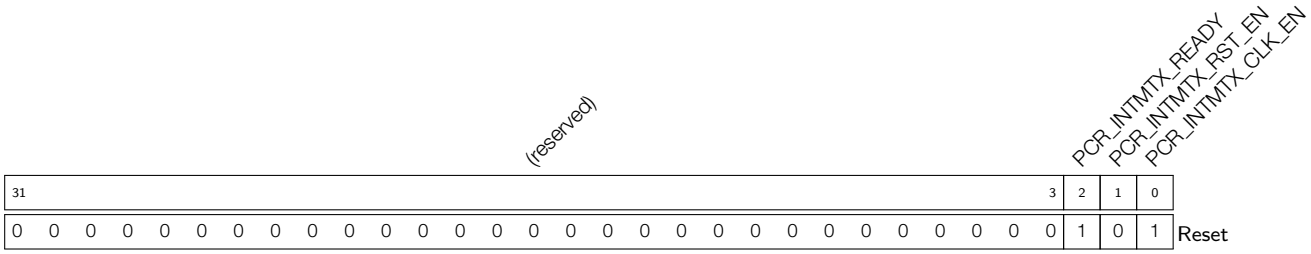
PCR_USB_SERIAL_JTAG_READY 表示 USB Serial/JTAG 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.35. PCR_INTMTX_CONF_REG (0x0090)



PCR_INTMTX_CLK_EN 配置是否使能中断矩阵时钟。

0：不使能

1：使能

(R/W)

PCR_INTMTX_RST_EN 配置是否复位中断矩阵模块。

0：不复位

1：复位

(R/W)

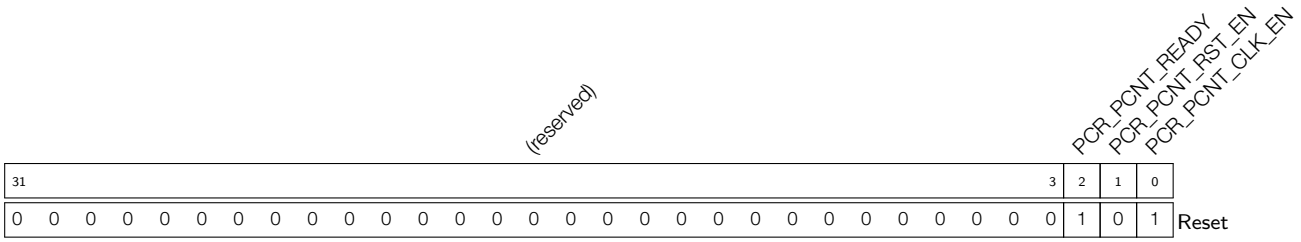
PCR_INTMTX_READY 表示中断矩阵模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.36. PCR_PCNT_CONF_REG (0x0094)



PCR_PCNT_CLK_EN 配置是否使能 PCNT 时钟。

0：不使能

1：使能

(R/W)

PCR_PCNT_RST_EN 配置是否复位 PCNT 模块。

0：不复位

1：复位

(R/W)

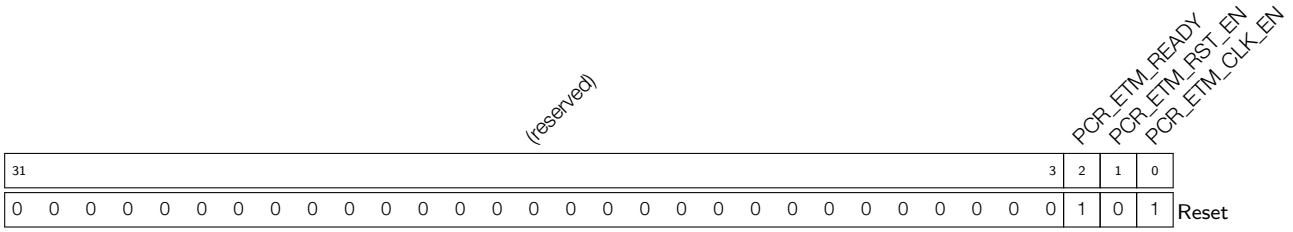
PCR_PCNT_READY 表示 PCNT 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.37. PCR_ETM_CONF_REG (0x0098)



PCR_ETM_CLK_EN 配置是否使能 ETM 时钟。

0：不使能

1：使能

(R/W)

PCR_ETM_RST_EN 配置是否复位 ETM 模块。

0：不复位

1：复位

(R/W)

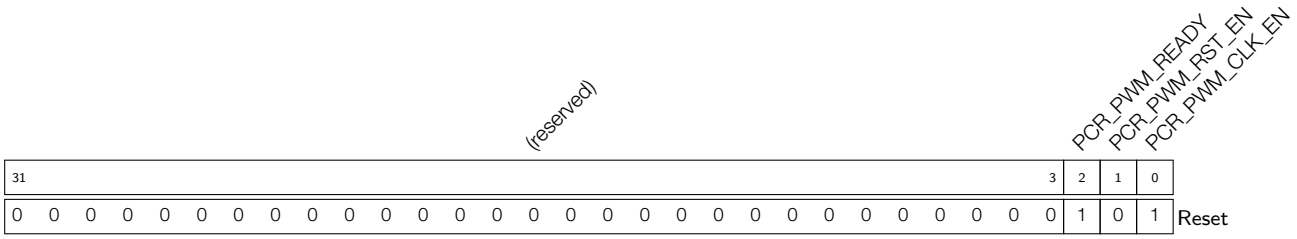
PCR_ETM_READY 表示 ETM 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.38. PCR_PWM_CONF_REG (0x009C)



PCR_PWM_CLK_EN 配置是否使能 MCPWM 总线时钟。

0：不使能

1：使能

(R/W)

PCR_PWM_RST_EN 配置是否复位 MCPWM 模块。

0：不复位

1：复位

(R/W)

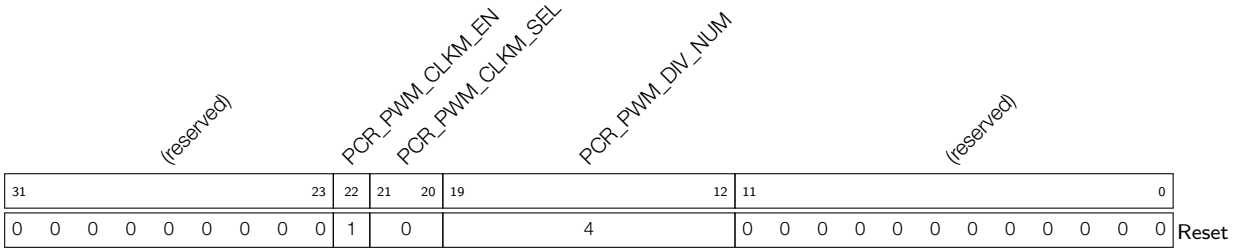
PCR_PWM_READY 表示 MCPWM 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.39. PCR_PWM_CLK_CONF_REG (0x00A0)



PCR_PWM_DIV_NUM 配置 MCPWM 功能时钟分频系数的整数部分。(R/W)

PCR_PWM_CLKM_SEL 配置选择 MCPWM 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F96M_CLK 时钟

3：不选择任何时钟

(R/W)

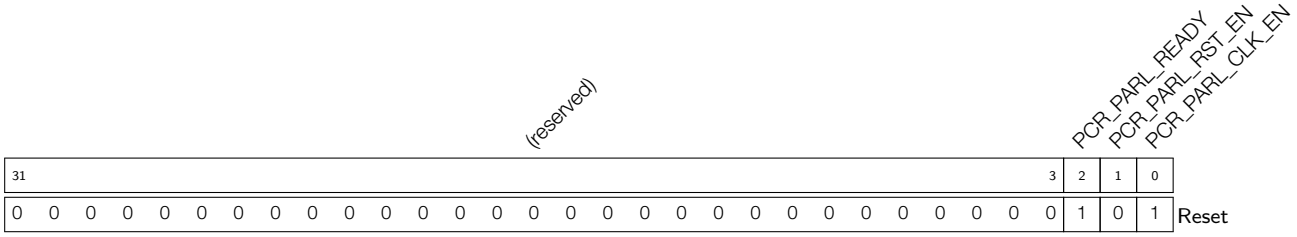
PCR_PWM_CLKM_EN 配置是否使能 MCPWM 的功能时钟。

0：不使能

1：使能

(R/W)

Register 7.40. PCR_PARL_IO_CONF_REG (0x00A4)



PCR_PARL_CLK_EN 配置是否使能并行 IO 控制器 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_PARL_RST_EN 配置是否复位并行 IO 控制器 APB 寄存器。

0：不复位

1：复位

(R/W)

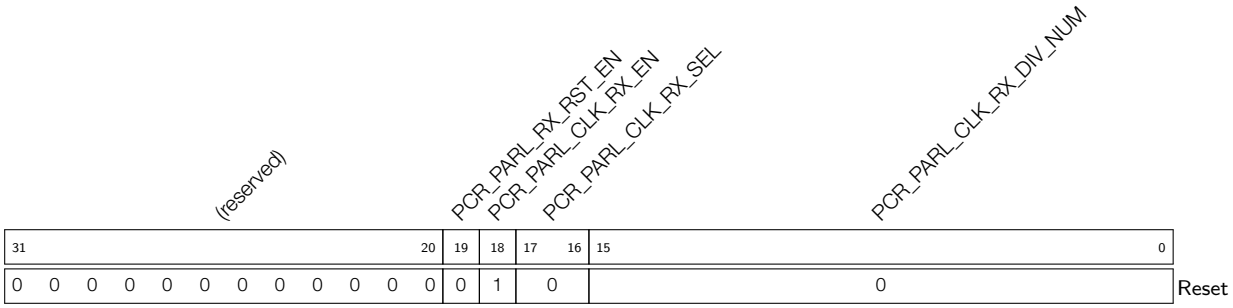
PCR_PARL_READY 表示并行 IO 控制器模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.41. PCR_PARL_CLK_RX_CONF_REG (0x00A8)



PCR_PARL_CLK_RX_DIV_NUM 配置并行 IO 控制器 RX 时钟的整数分频系数。(R/W)

PCR_PARL_CLK_RX_SEL 配置并行 IO 控制器 RX 选择时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 PLL_F96M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3：使用来自管脚的时钟

(R/W)

PCR_PARL_CLK_RX_EN 配置是否使能并行 IO 控制器 RX 时钟。

0：不使能

1：使能

(R/W)

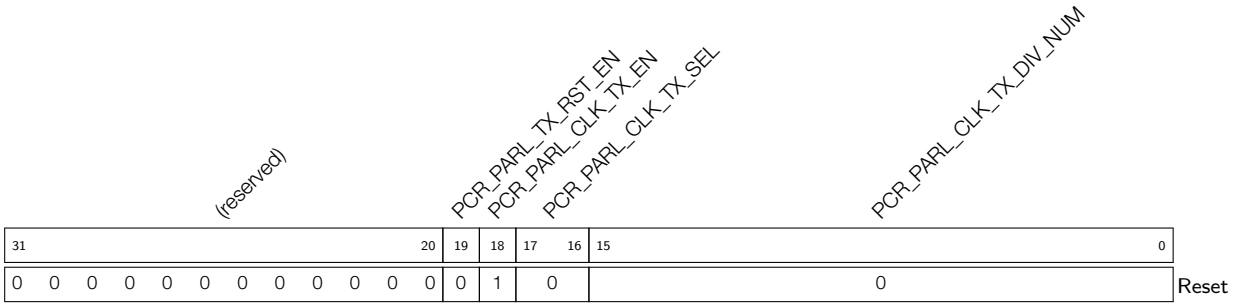
PCR_PARL_RX_RST_EN 配置是否复位并行 IO 控制器 RX 模块。

0：不复位

1：复位

(R/W)

Register 7.42. PCR_PARL_CLK_TX_CONF_REG (0x00AC)



PCR_PARL_CLK_TX_DIV_NUM 配置并行 IO 控制器 TX 时钟的整数分频系数。(R/W)

PCR_PARL_CLK_TX_SEL 配置选择并行 IO 控制器 TX 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 PLL_F96M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3：使用来自管脚的时钟

(R/W)

PCR_PARL_CLK_TX_EN 配置是否使能并行 IO 控制器 TX 时钟。

0：不使能

1：使能

(R/W)

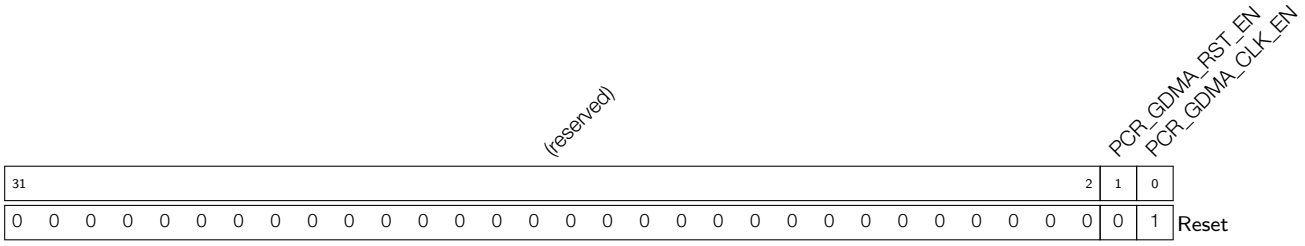
PCR_PARL_TX_RST_EN 配置是否复位并行 IO 控制器 TX 模块。

0：不复位

1：复位

(R/W)

Register 7.43. PCR_GDMA_CONF_REG (0x00B8)



PCR_GDMA_CLK_EN 配置是否使能 GDMA 时钟。

0：不使能

1：使能

(R/W)

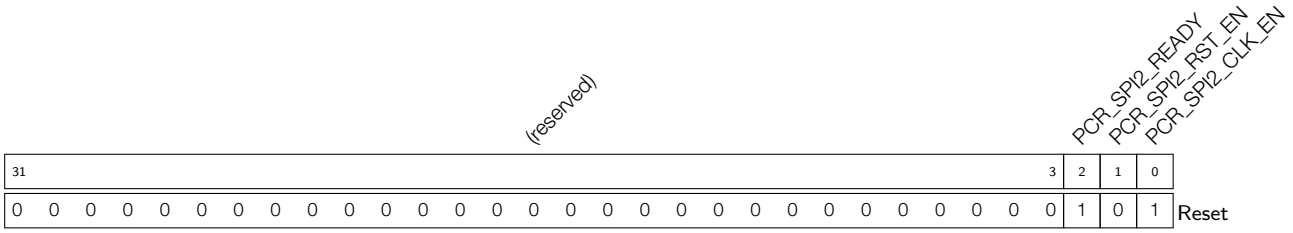
PCR_GDMA_RST_EN 配置是否复位 GDMA 模块。

0：不复位

1：复位

(R/W)

Register 7.44. PCR_SPI2_CONF_REG (0x00BC)



PCR_SPI2_CLK_EN 配置是否使能 SPI2 APB_CLK。

0：不使能

1：使能

(R/W)

PCR_SPI2_RST_EN 配置是否复位 SPI2 模块。

0：不复位

1：复位

(R/W)

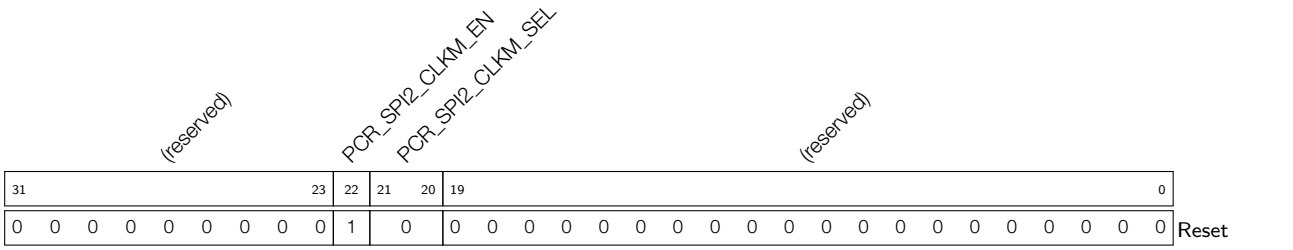
PCR_SPI2_READY 表示 SPI2 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.45. PCR_SPI2_CLKM_CONF_REG (0x00C0)



PCR_SPI2_CLKM_SEL 配置选择 SPI2 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 PLL_F48M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3：不选择任何时钟

(R/W)

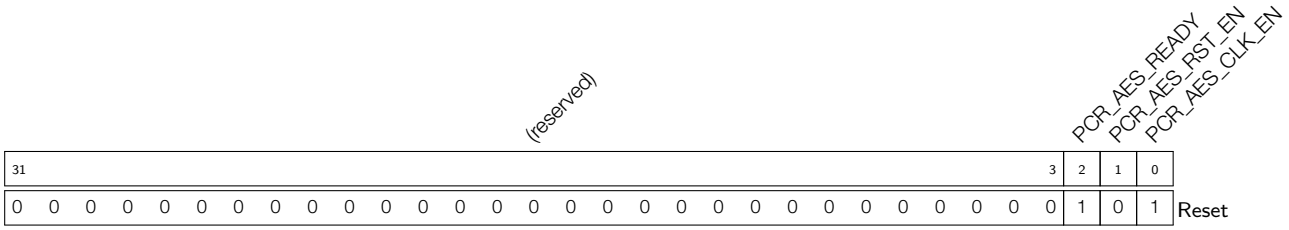
PCR_SPI2_CLKM_EN 配置是否使能 SPI2 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.46. PCR_AES_CONF_REG (0x00C4)



PCR_AES_CLK_EN 配置是否使能 AES 时钟。

0：不使能

1：使能

(R/W)

PCR_AES_RST_EN 配置是否复位 AES 模块。

0：不复位

1：复位

(R/W)

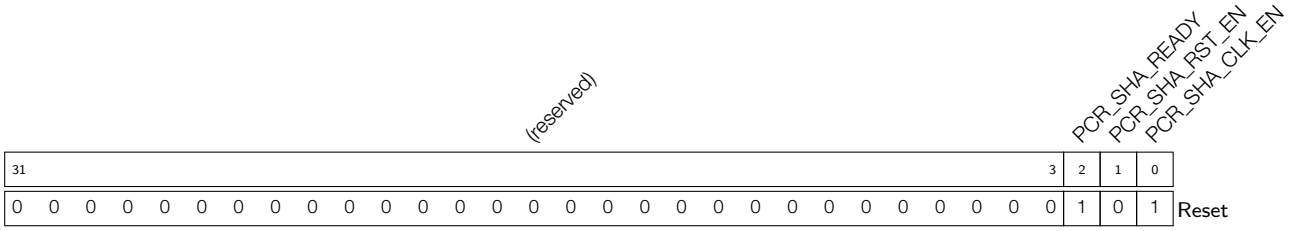
PCR_AES_READY 表示 AES 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.47. PCR_SHA_CONF_REG (0x00C8)



PCR_SHA_CLK_EN 配置是否使能 SHA 时钟。

0：不使能

1：使能

(R/W)

PCR_SHA_RST_EN 配置是否复位 SHA 模块。

0：不复位

1：复位

(R/W)

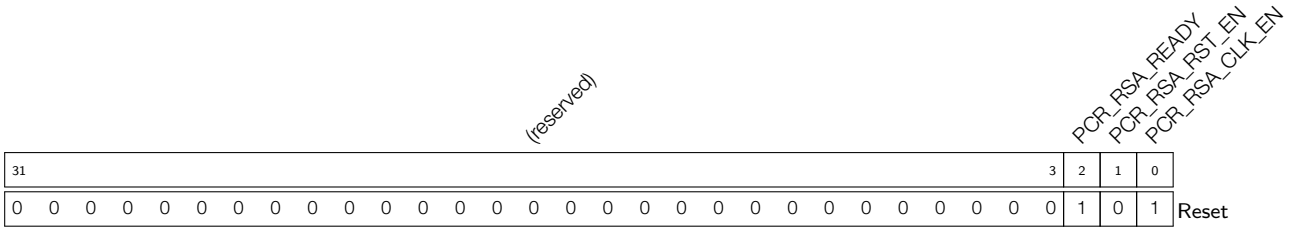
PCR_SHA_READY 表示 SHA 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.48. PCR_RSA_CONF_REG (0x00CC)



PCR_RSA_CLK_EN 配置是否使能 RSA 时钟。

0：不使能

1：使能

(R/W)

PCR_RSA_RST_EN 配置是否复位 RSA 模块。

0：不复位

1：复位

(R/W)

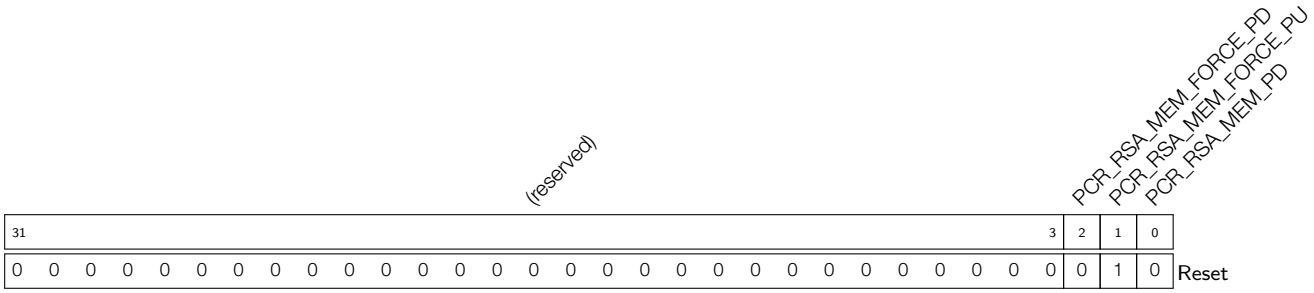
PCR_RSA_READY 表示 RSA 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.49. PCR_RSA_PD_CTRL_REG (0x00D0)



PCR_RSA_MEM_PD 配置是否使 RSA 内部存储器掉电。

0：不掉电

1：强制掉电

(R/W)

PCR_RSA_MEM_FORCE_PU 配置是否强制 RSA 内部存储器上电。

0：不强制上电

1：强制上电

(R/W)

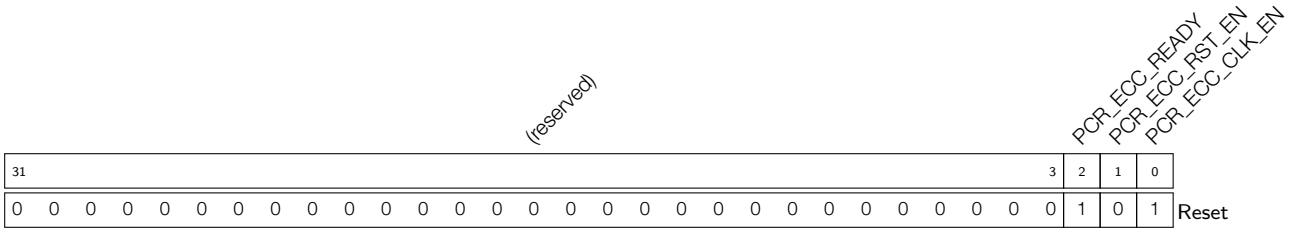
PCR_RSA_MEM_FORCE_PD 配置是否强制 RSA 内部存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

Register 7.50. PCR_ECC_CONF_REG (0x00D4)



PCR_ECC_CLK_EN 配置是否使能 ECC 时钟。

0：不使能

1：使能

(R/W)

PCR_ECC_RST_EN 配置是否复位 ECC 模块。

0：不复位

1：复位

(R/W)

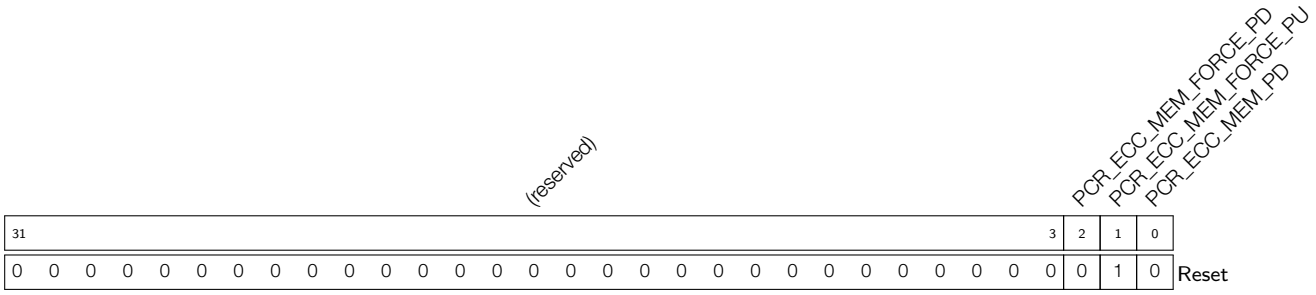
PCR_ECC_READY 表示 ECC 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.51. PCR_ECC_PD_CTRL_REG (0x00D8)



PCR_ECC_MEM_PD 配置是否使 ECC 内部存储器掉电。

0：不掉电

1：掉电

(R/W)

PCR_ECC_MEM_FORCE_PU 配置是否强制 ECC 内部存储器上电。

0：不强制上电

1：强制上电

(R/W)

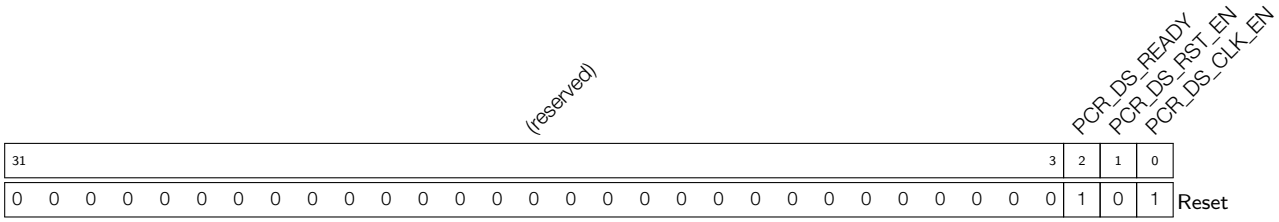
PCR_ECC_MEM_FORCE_PD 配置是否强制 ECC 内部存储器掉电。

0：不强制掉电

1：强制掉电

(R/W)

Register 7.52. PCR_DS_CONF_REG (0x00DC)



PCR_DS_CLK_EN 配置是否使能 DS 时钟。

0：不使能

1：使能

(R/W)

PCR_DS_RST_EN 配置是否复位 DS 模块。

0：不复位

1：复位

(R/W)

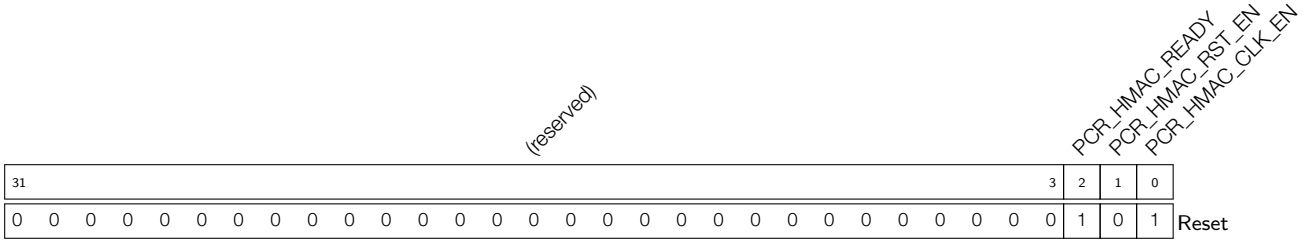
PCR_DS_READY 表示 DS 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.53. PCR_HMAC_CONF_REG (0x00E0)



PCR_HMAC_CLK_EN 配置是否使能 HMAC 时钟。

0：不使能

1：使能

(R/W)

PCR_HMAC_RST_EN 配置是否复位 HMAC 模块。

0：不复位

1：复位

(R/W)

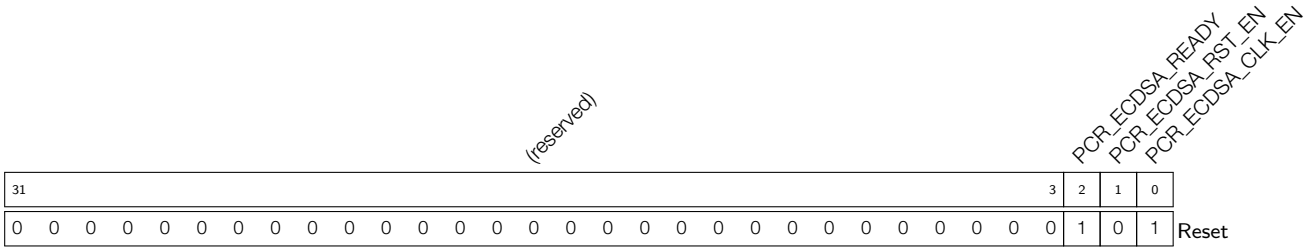
PCR_HMAC_READY 表示 HMAC 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.54. PCR_ECDSA_CONF_REG (0x00E4)



PCR_ECDSA_CLK_EN 配置是否使能 ECDSA 时钟。

0：不使能

1：使能

(R/W)

PCR_ECDSA_RST_EN 配置是否复位 ECDSA 模块。

0：不复位

1：复位

(R/W)

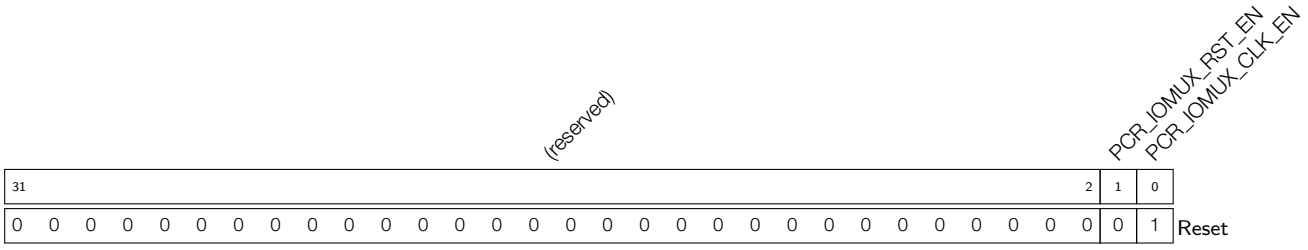
PCR_ECDSA_READY 表示 ECDSA 模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.55. PCR_IOMUX_CONF_REG (0x00E8)



PCR_IOMUX_CLK_EN 配置是否使能 IO MUX APB_CLK。

0：不使能

1：使能

(R/W)

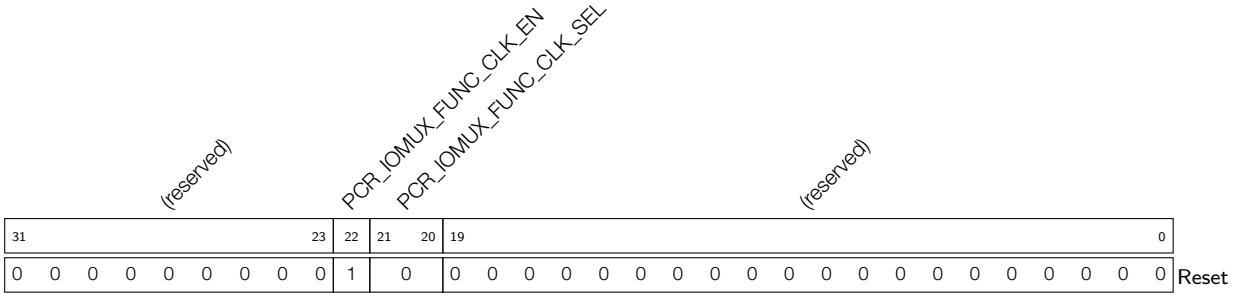
PCR_IOMUX_RST_EN 配置是否复位 IO MUX 模块。

0：不复位

1：复位

(R/W)

Register 7.56. PCR_IOMUX_CLK_CONF_REG (0x00EC)



PCR_IOMUX_FUNC_CLK_SEL 配置选择 IO MUX 时钟源。

0（默认）：选择 XTAL_CLK 时钟

1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F48M_CLK 时钟

3：不选择任何时钟

(R/W)

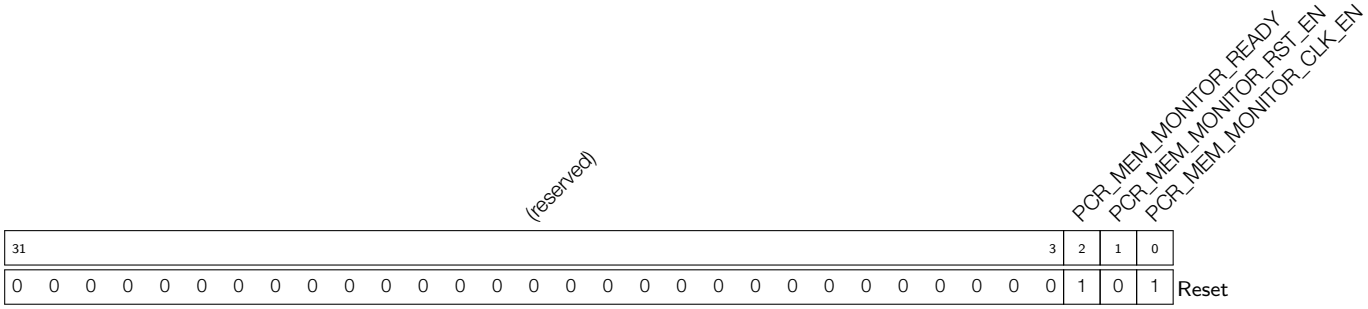
PCR_IOMUX_FUNC_CLK_EN 配置是否使能 IO MUX 功能时钟。

0：不使能

1：使能

(R/W)

Register 7.57. PCR_MEM_MONITOR_CONF_REG (0x00F0)



PCR_MEM_MONITOR_CLK_EN 配置是否使能内存访问监控模块时钟。

0：不使能

1：使能

(R/W)

PCR_MEM_MONITOR_RST_EN 配置是否复位内存访问监控模块。

0：不复位

1：复位

(R/W)

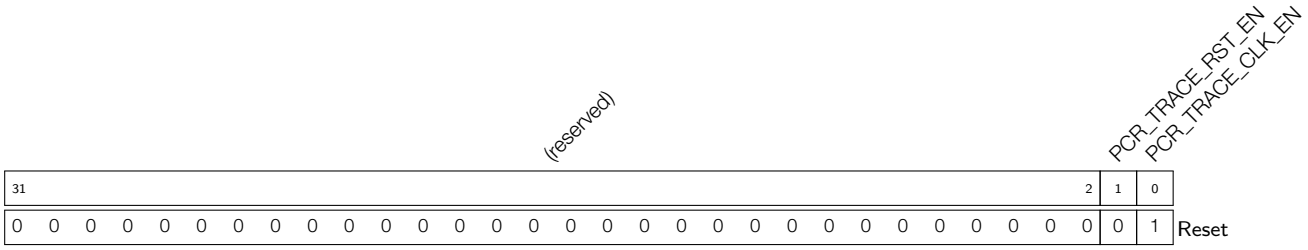
PCR_MEM_MONITOR_READY 表示内存访问监控模块是否解复位完成。

0：未完成

1：完成

(RO)

Register 7.58. PCR_TRACE_CONF_REG (0x00F8)



PCR_TRACE_CLK_EN 配置是否使能 RISC-V 追踪编码器时钟。

0：不使能

1：使能

(R/W)

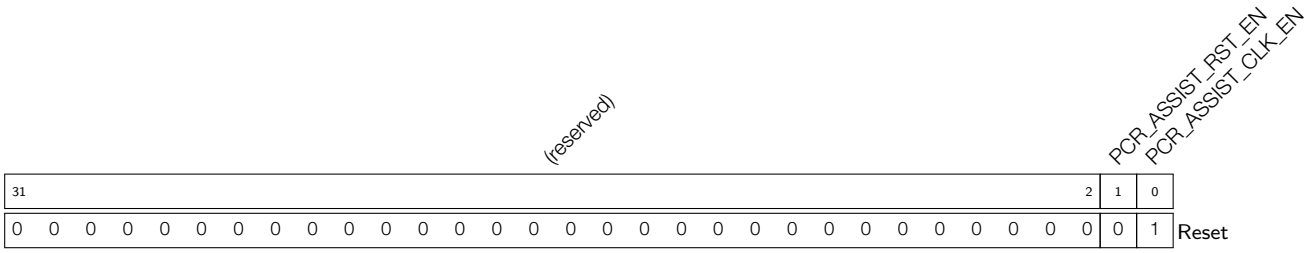
PCR_TRACE_RST_EN 配置是否复位 RISC-V 追踪编码器模块。

0：不复位

1：复位

(R/W)

Register 7.59. PCR_ASSIST_CONF_REG (0x00FC)



PCR_ASSIST_CLK_EN 配置是否使能辅助调试模块时钟。

0：不使能

1：使能

(R/W)

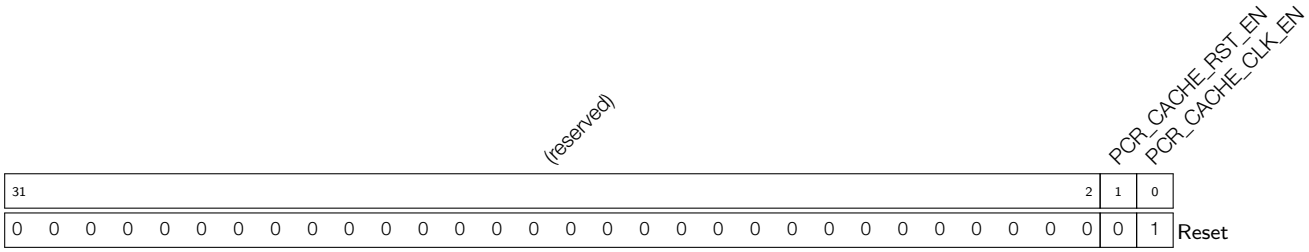
PCR_ASSIST_RST_EN 配置是否复位辅助调试模块。

0：不复位

1：复位

(R/W)

Register 7.60. PCR_CACHE_CONF_REG (0x0100)



PCR_CACHE_CLK_EN 配置是否使能 Cache 时钟。

0：不使能

1：使能

(R/W)

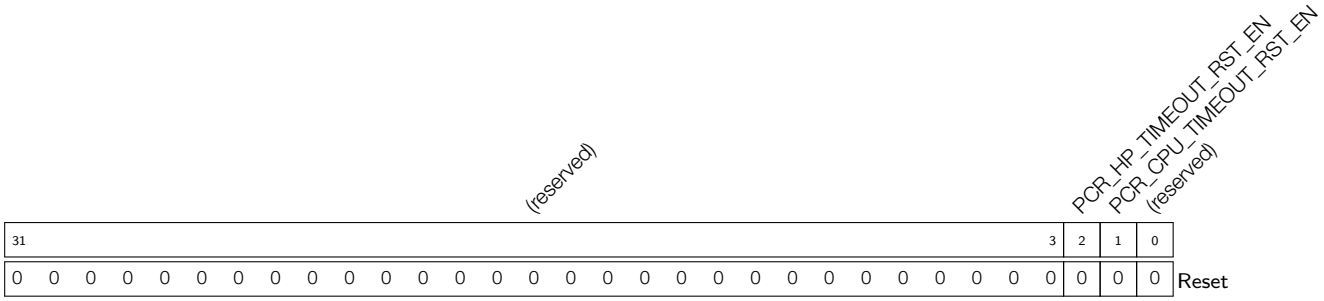
PCR_CACHE_RST_EN 配置是否复位 Cache 模块。

0：不复位

1：复位

(R/W)

Register 7.61. PCR_TIMEOUT_CONF_REG (0x0108)



PCR_CPU_TIMEOUT_RST_EN 配置是否复位 CPU 外设超时保护模块。

0：不复位

1：复位

(R/W)

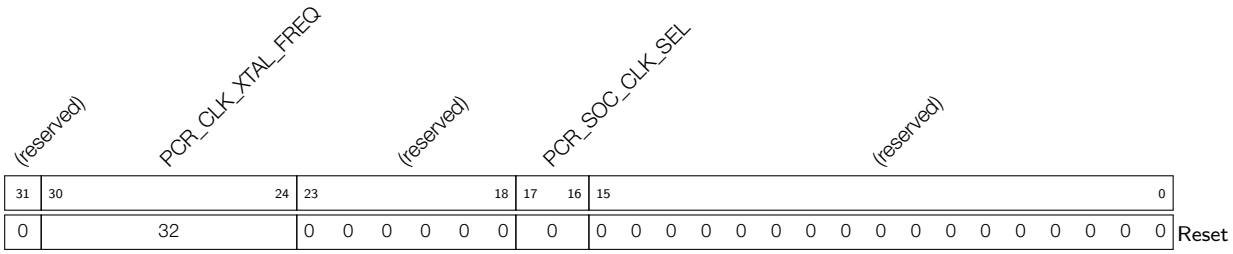
PCR_HP_TIMEOUT_RST_EN 配置是否复位 HP 外设超时保护模块。

0：不复位

1：复位

(R/W)

Register 7.62. PCR_SYSCLK_CONF_REG (0x010C)



PCR_SOC_CLK_SEL 配置选择 HP_ROOT_CLK 时钟源。

0：选择 XTAL_CLK 时钟

1：选择 PLL_F96M_CLK 时钟

2：选择 RC_FAST_CLK 时钟

3：选择 PLL_F64M_CLK 时钟

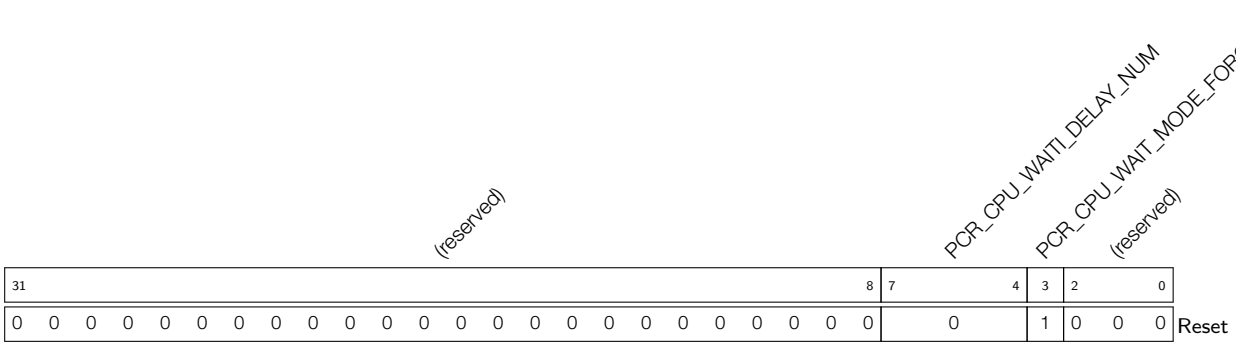
(R/W)

PCR_CLK_XTAL_FREQ 表示 XTAL_CLK 时钟频率。

单位：MHz。

(RO)

Register 7.63. PCR_CPU_WAITI_CONF_REG (0x0110)



PCR_CPU_WAIT_MODE_FORCE_ON 配置是否强制打开 CPU 等待中断模式下的门控时钟。

0：不强制使能

1：强制使能

通常情况下，CPU 执行 WFI (wait for interrupt) 指令后会进入等待中断模式。在此模式下 CPU 的时钟门控一直处于关闭状态，直到中断产生，因此可降低功耗。若此位置 1，CPU 的门控时钟会被强制打开，不受 WFI 指令的影响。

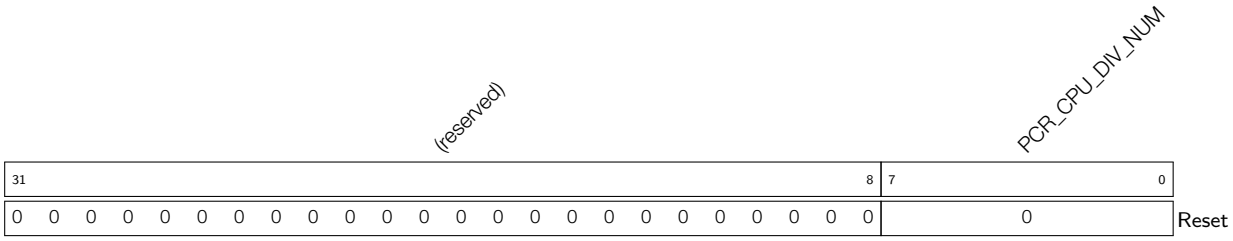
(R/W)

PCR_CPU_WAITI_DELAY_NUM 配置 CPU 在收到 WFI 指令后进入 CPU 等待中断模式后，关闭 CPU 的门控时钟需要的等待周期。

单位：CPU_CLK 周期。

(R/W)

Register 7.64. PCR_CPU_FREQ_CONF_REG (0x0114)

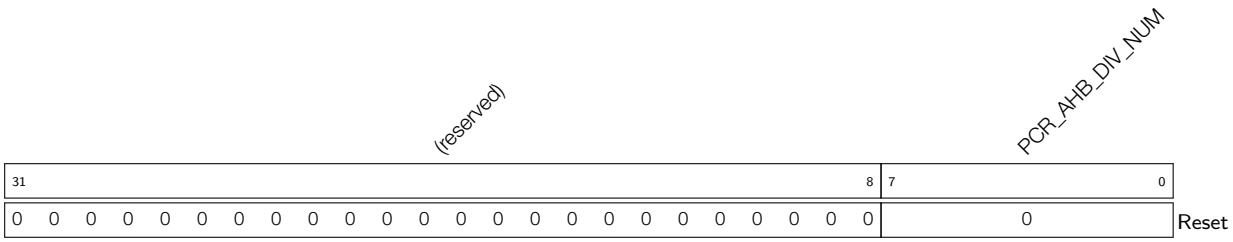


PCR_CPU_DIV_NUM 配置 HP_ROOT_CLK 生成 CPU_CLK 时使用的分频系数。

该字段需要与 PCR_AHB_DIV_NUM 搭配使用。

(R/W)

Register 7.65. PCR_AHB_FREQ_CONF_REG (0x0118)

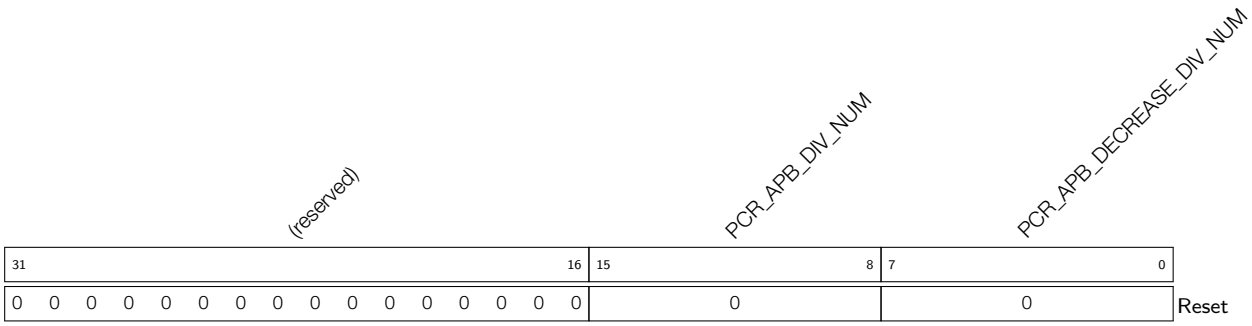


PCR_AHB_DIV_NUM 配置 HP_ROOT_CLK 生成 AHB_CLK 时使用的分频系数。

该字段需要与 PCR_CPU_DIV_NUM 搭配使用。

(R/W)

Register 7.66. PCR_APB_FREQ_CONF_REG (0x011C)



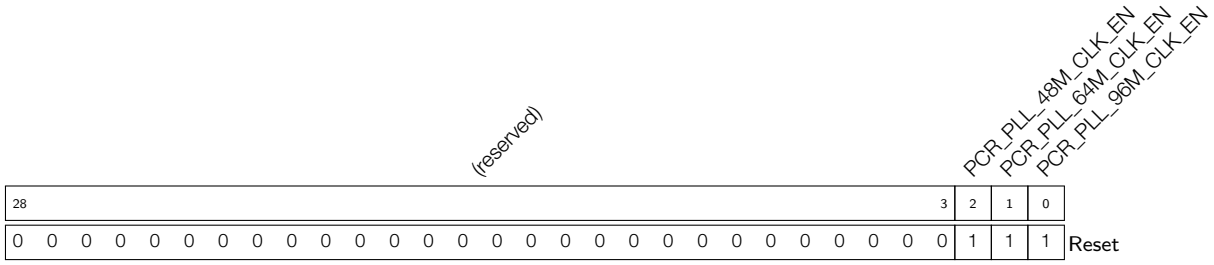
PCR_APB_DECREASE_DIV_NUM 配置 AHB_CLK 生成 APB_CLK 时使用的第二级分频系数。

(R/W)

PCR_APB_DIV_NUM 配置 AHB_CLK 生成 APB_CLK 时使用的第一级分频系数。

(R/W)

Register 7.67. PCR_PLL_DIV_CLK_EN_REG (0x0124)



PCR_PLL_96M_CLK_EN 配置是否使能 PLL_F96M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

PCR_PLL_64M_CLK_EN 配置是否使能 PLL_F64M_CLK 时钟。

0：不使能

1（默认）：使能

(R/W)

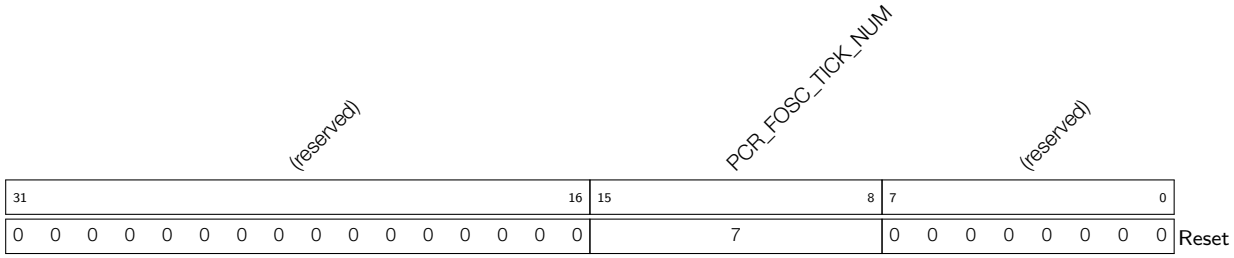
PCR_PLL_48M_CLK_EN 配置是否使能 48 MHz 时钟。该时钟由 PLL_F96M_CLK 经 2 分频生成。

0：不使能

1（默认）：使能

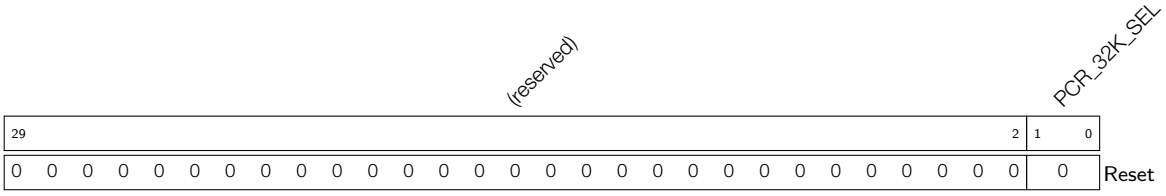
(R/W)

Register 7.68. PCR_CTRL_TICK_CONF_REG (0x012C)



PCR_FOSC_TICK_NUM 配置待进入校准模块 RC_FAST_CLK 的分频系数。(R/W)

Register 7.69. PCR_CTRL_32K_CONF_REG (0x0130)



PCR_32K_SEL 为 TIMER_GROUP 配置选择一个 32 kHz 时钟。

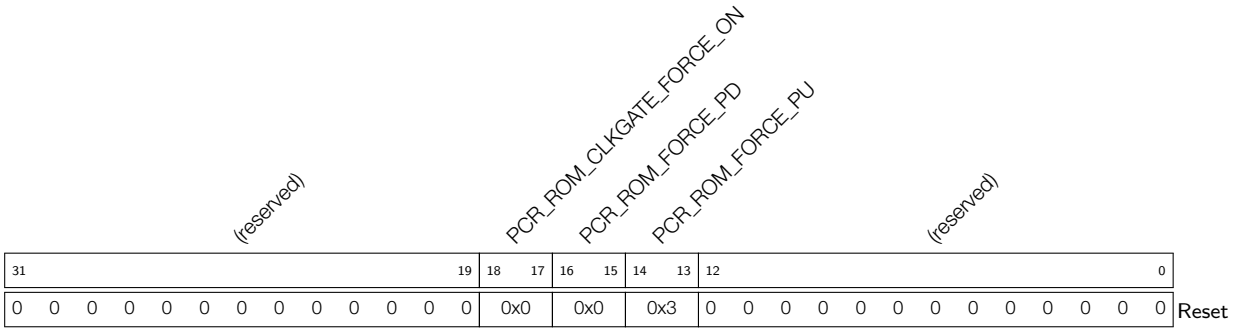
0：无效值，没有作用

1：选择 XTAL32K_CLK 时钟

2/3：选择 OSC_SLOW_CLK 时钟

(R/W)

Register 7.70. PCR_SRAM_POWER_CONF_0_REG (0x0134)



PCR_ROM_FORCE_PU 配置是否强制 ROM 上电。

0：不强制上电

1：强制上电

(R/W)

PCR_ROM_FORCE_PD 配置是否强制 ROM 掉电。

0：不强制掉电

1：强制掉电

(R/W)

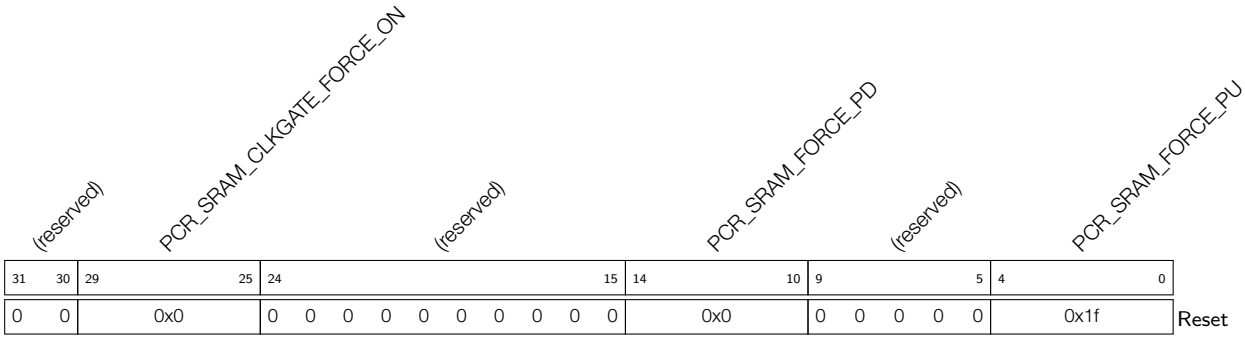
PCR_ROM_CLKGATE_FORCE_ON 配置在访问 ROM 时是否强制打开时钟并绕过时钟门控。

0：访问 ROM 时使用时钟门控

1：访问 ROM 时强制打开时钟并绕过时钟门控

(R/W)

Register 7.71. PCR_SRAM_POWER_CONF_1_REG (0x0138)



PCR_SRAM_FORCE_PU 配置是否强制 SRAM 上电。

0：不强制上电

1：强制上电

(R/W)

PCR_SRAM_FORCE_PD 配置是否强制 SRAM 掉电。

0：不强制掉电

1：强制掉电

(R/W)

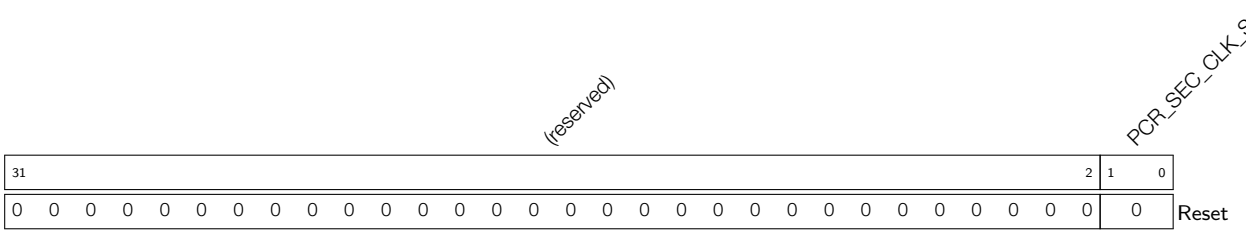
PCR_SRAM_CLKGATE_FORCE_ON 配置在访问 SRAM 时是否强制打开时钟并绕过时钟门控。

0：访问 SRAM 时使用时钟门控

1：访问 SRAM 时强制打开时钟并绕过时钟门控

(R/W)

Register 7.72. PCR_SEC_CONF_REG (0x013C)



PCR_SEC_CLK_SEL 配置选择片外存储器加密与解密模块的时钟源。

0（默认）：选择 XTAL_CLK 时钟

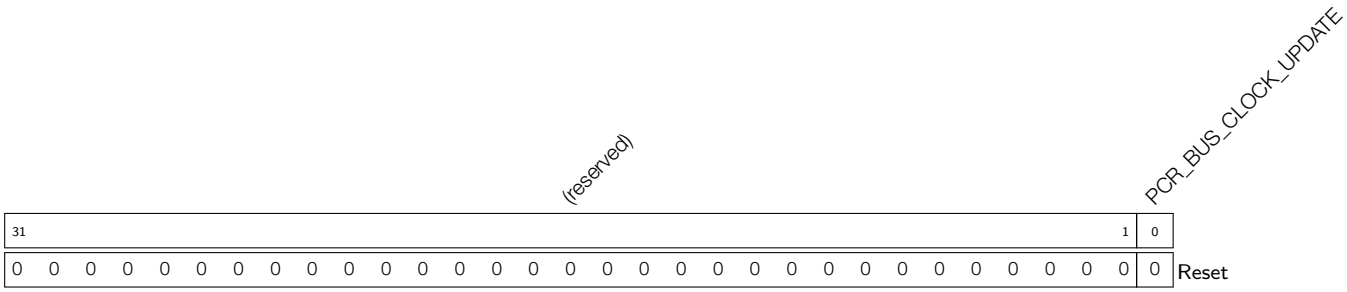
1：选择 RC_FAST_CLK 时钟

2：选择 PLL_F64M_CLK 时钟

3：选择 PLL_F96M_CLK 时钟

(R/W)

Register 7.73. PCR_BUS_CLK_UPDATE_REG (0x0148)



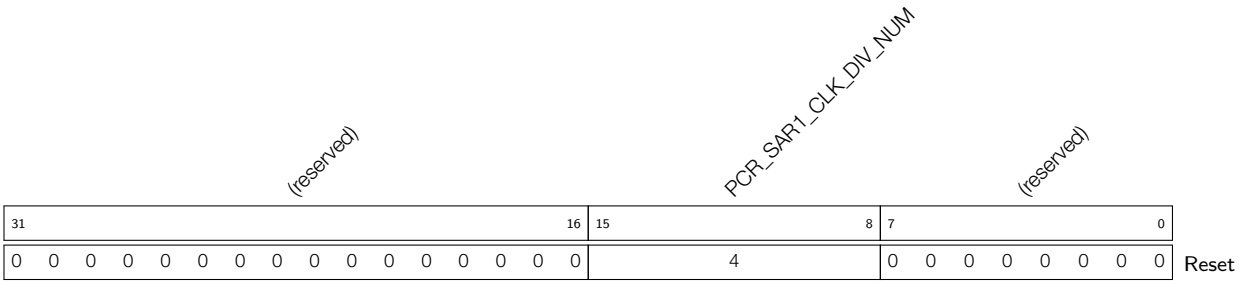
PCR_BUS_CLOCK_UPDATE 配置是否更新 CPU_CLK 分频、AHB_CLK 分频、HP_ROOT_CLK 时钟源选择的配置。

0：不更新配置

1：更新配置

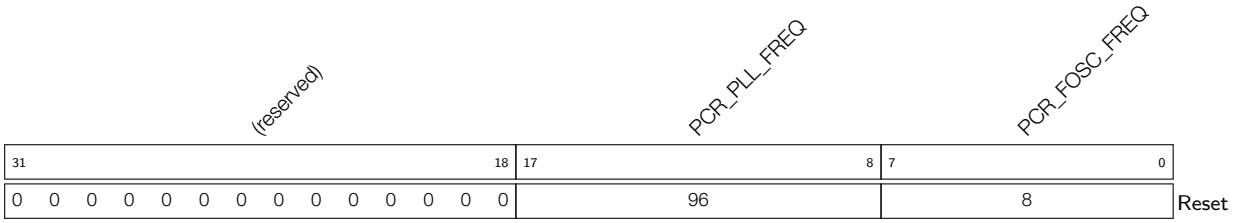
此位在更新完成后自动清零。(R/W/WTC)

Register 7.74. PCR_SAR_CLK_DIV_REG (0x014C)



PCR_SAR1_CLK_DIV_NUM SAR ADC 采样逻辑时钟的分频系数，用来产生 ADC 模拟电路控制信号。(R/W)

Register 7.75. PCR_SYSCLK_FREQ_QUERY_0_REG (0x0120)



PCR_FOSC_FREQ 表示 RC_FAST_CLK 时钟频率。

单位：MHz。

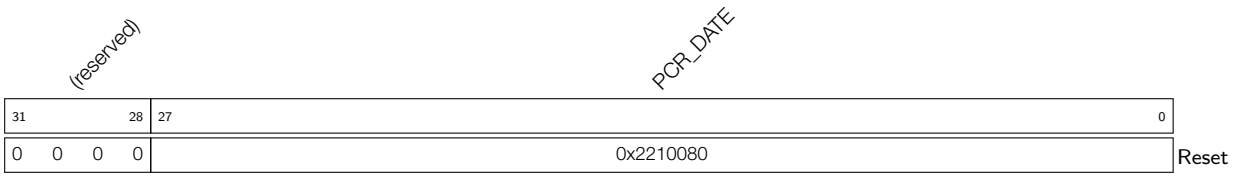
(HRO)

PCR_PLL_FREQ 表示 PLL_F96M_CLK 时钟频率。

单位：MHz。

(HRO)

Register 7.76. PCR_DATE_REG (0x0FFC)

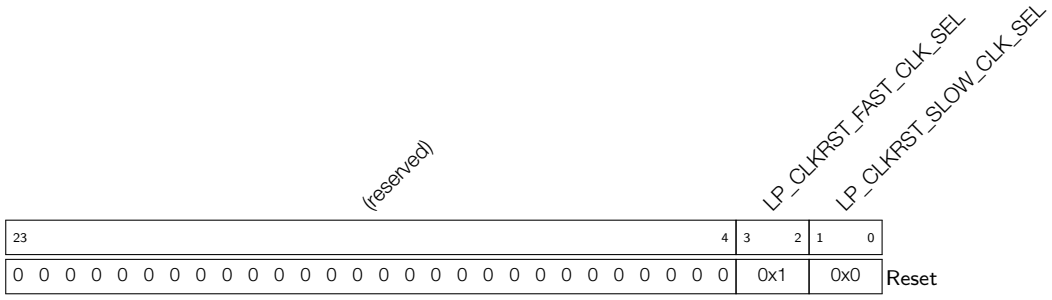


PCR_DATE 版本控制寄存器。(R/W)

7.5.2 低功耗系统时钟寄存器

本小节前缀为 LPPERI 的寄存器（最后两个）地址为相对于低功耗外设寄存器（LPPERI）基地址的地址偏移量（相对地址），前缀为 LP_AON 的寄存器（倒数第三、四个）地址为相对于低功耗常开寄存器（LP_AON）基地址的地址偏移量（相对地址），其余所有地址均为相对于低功耗时钟复位寄存器（LP_CLKRST）基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 7.77. LP_CLKRST_LP_CLK_CONF_REG (0x0000)



LP_CLKRST_SLOW_CLK_SEL 配置 LP_SLOW_CLK 的时钟源。

0：RC_SLOW_CLK

1：XTAL32K_CLK

2：无效值。没有作用

3：OSC_SLOW_CLK

(R/W)

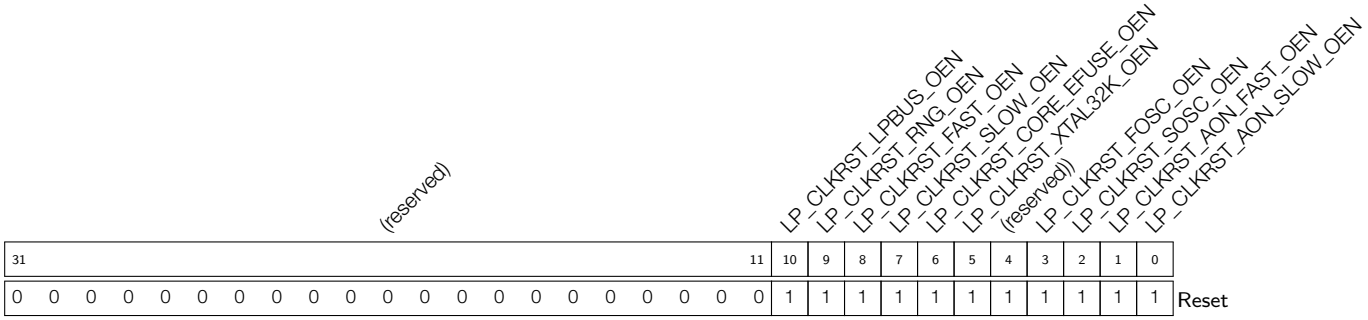
LP_CLKRST_FAST_CLK_SEL 配置 LP_FAST_CLK 的时钟源。

0：RC_FAST_CLK

1：XTAL_D2_CLK

(R/W)

Register 7.78. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)



LP_CLKRST_AON_SLOW_OEN 配置是否使能 LP_DYN_SLOW_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_AON_FAST_OEN 配置是否使能 LP_DYN_FAST_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_SOSC_OEN 配置是否使能 OSC_SLOW_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_FOSC_OEN 配置是否使能 RC_FAST_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_XTAL32K_OEN 配置是否使能 XTAL32K_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

见下页...

Register 7.78. LP_CLKRST_LP_CLK_PO_EN_REG (0x0004)

接上页...

LP_CLKRST_CORE_EFUSE_OEN 配置是否使能 EFUSE_CTRL 时钟输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_SLOW_OEN 配置是否使能 LP_SLOW_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_FAST_OEN 配置是否使能 LP_FAST_CLK 输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

LP_CLKRST_RNG_OEN 配置是否使能 RNG 时钟输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

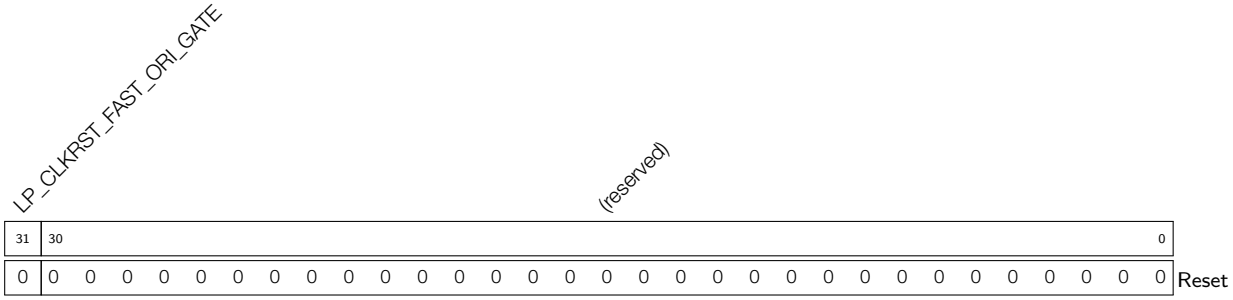
LP_CLKRST_LPBUS_OEN 配置是否使能 LP 总线时钟输出到 pad 的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

Register 7.79. LP_CLKRST_LP_CLK_EN_REG (0x0008)



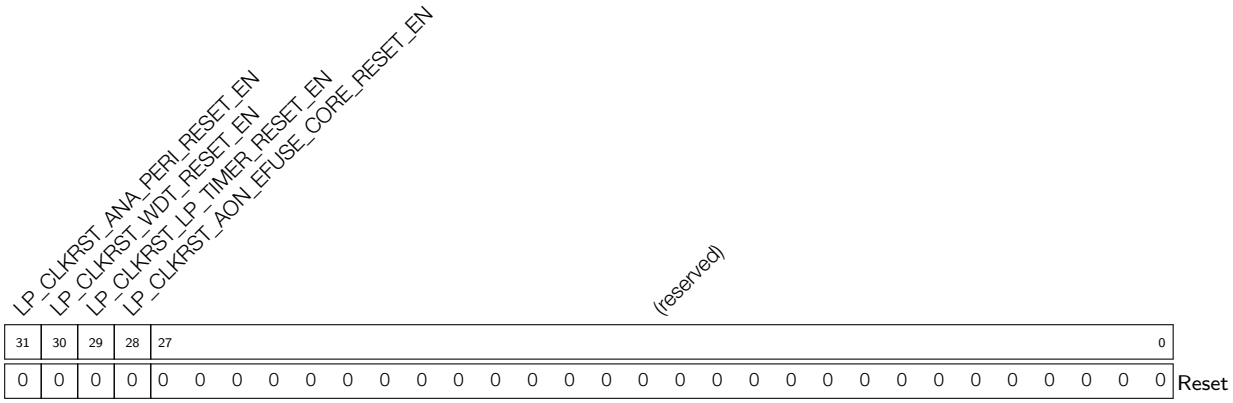
LP_CLKRST_FAST_ORI_GATE 配置 LP_FAST_CLK 的时钟门控。

0：无效。时钟门控由硬件 FSM 控制

1：强制时钟通过门控

(R/W)

Register 7.80. LP_CLKRST_LP_RST_EN_REG (0x000C)



LP_CLKRST_AON_EFUSE_CORE_RESET_EN 配置复位 EFUSE_CTRL 常开部分。

0：无效

1：复位

(R/W)

LP_CLKRST_LP_TIMER_RESET_EN 配置复位 RTC 定时器。

0：无效

1：复位

(R/W)

LP_CLKRST_WDT_RESET_EN 配置复位 RWDT 和超级看门狗定时器。

0：无效

1：复位

(R/W)

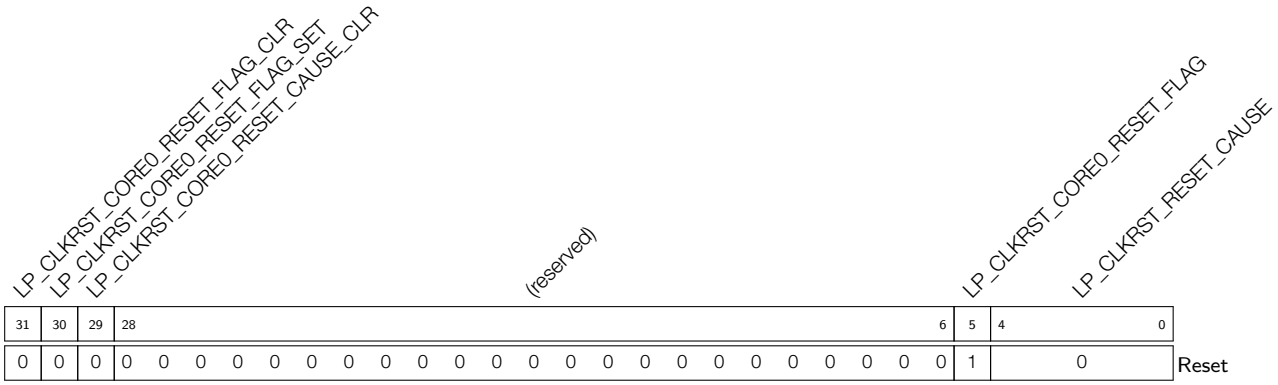
LP_CLKRST_ANA_PERI_RESET_EN 配置复位模拟外设，包括欠压检测器。

0：无效

1：复位

(R/W)

Register 7.81. LP_CLKRST_RESET_CAUSE_REG (0x0010)



LP_CLKRST_RESET_CAUSE 表示复位原因。(RO)

LP_CLKRST_CORE0_RESET_FLAG 表示发生的复位是否为 LP_CLKRST_RESET_CAUSE 中记录的复位。

0：非法复位

1：已记录的复位

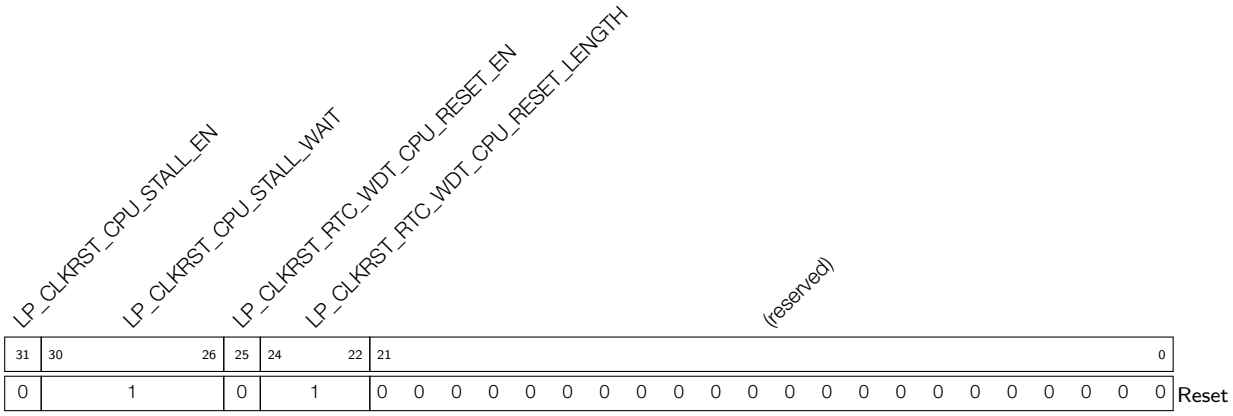
(RO)

LP_CLKRST_CORE0_RESET_CAUSE_CLR 写 1 清除 LP_CLKRST_RESET_CAUSE。(WT)

LP_CLKRST_CORE0_RESET_FLAG_SET 写 1 置位 LP_CLKRST_CORE0_RESET_FLAG。(WT)

LP_CLKRST_CORE0_RESET_FLAG_CLR 写 1 清除 LP_CLKRST_CORE0_RESET_FLAG。(WT)

Register 7.82. LP_CLKRST_CPU_RESET_REG (0x0014)



LP_CLKRST_RTC_WDT_CPU_RESET_LENGTH 配置 RWDT 复位 CPU 的时间长度。

单位：LP_DYN_FAST_CLK 周期。

(R/W)

LP_CLKRST_RTC_WDT_CPU_RESET_EN 配置 RWDT 是否能复位 CPU。

0：RWDT 超时时无法复位 CPU

1：RWDT 超时时复位 CPU

(R/W)

LP_CLKRST_CPU_STALL_WAIT 配置 CPU 停顿状态 (CPU Stall) 到复位之间的时间间隔。

单位：LP_DYN_FAST_CLK 周期。

(R/W)

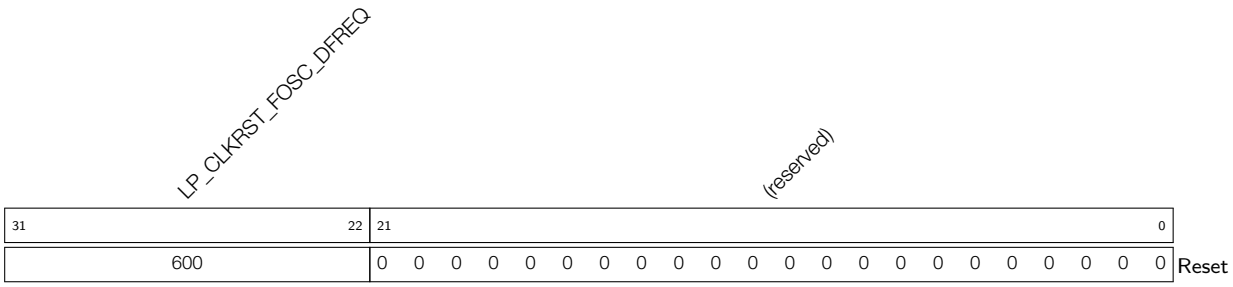
LP_CLKRST_CPU_STALL_EN 配置在 RWDT 和软件复位 CPU 之前是否进入 CPU 停顿状态。

0：否

1：是

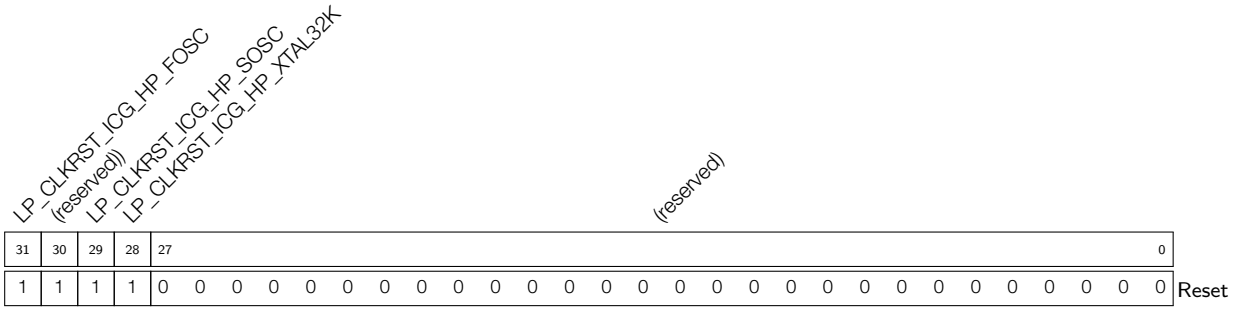
(R/W)

Register 7.83. LP_CLKRST_FOSC_CNTL_REG (0x0018)



LP_CLKRST_FOSC_DFREQ 配置 RC_FAST_CLK 频率，值越大，时钟周期越大。(R/W)

Register 7.84. LP_CLKRST_CLK_TO_HP_REG (0x0020)



LP_CLKRST_ICG_HP_XTAL32K 配置 XTAL32K_CLK 到 HP System 的时钟门控。

0：时钟无法到达 HP System

1：时钟能够到达 HP System

(R/W)

LP_CLKRST_ICG_HP_SOSC 配置 RC_SLOW_CLK 到 HP System 的时钟门控。

0：时钟无法到达 HP System

1：时钟能够到达 HP System

(R/W)

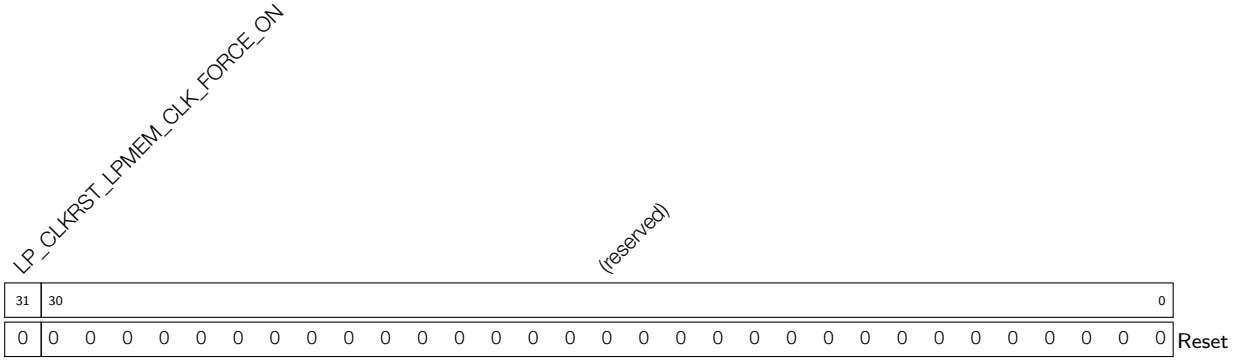
LP_CLKRST_ICG_HP_FOSC 配置 RC_FAST_CLK 到 HP System 的时钟门控。

0：时钟无法到达 HP System

1：时钟能够到达 HP System

(R/W)

Register 7.85. LP_CLKRST_LPMEM_FORCE_REG (0x0024)



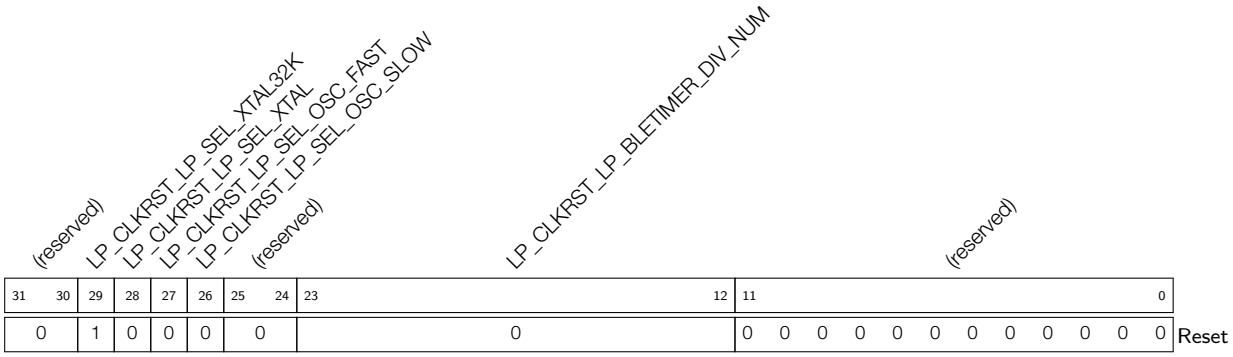
LP_CLKRST_LPMEM_CLK_FORCE_ON 配置强制打开 LP 存储器的时钟门控。

0：无效。时钟门控由硬件 FSM 控制

1：强制打开时钟门控

(R/W)

Register 7.86. LP_CLKRST_LPPERI_REG (0x0028)



LP_CLKRST_LP_BLETIMER_DIV_NUM 配置 RTC BLE 定时器的时钟分频器的分频系数, RTC BLE 定时器的工作时钟频率 = 源时钟频率 / ((LP_CLKRST_LP_BLETIMER_DIV_NUM - 1) / 2) (R/W)

LP_CLKRST_LP_SEL_OSC_SLOW 配置 RTC BLE 定时器的时钟源

1: 选择 RC_SLOW_CLK 为 RTC BLE 定时器的源时钟

0 : 禁止 RC_SLOW_CLK 为 RTC BLE 定时器的源时钟

(R/W)

LP_CLKRST_LP_SEL_OSC_FAST 配置 RTC BLE 定时器的时钟源

1: 选择 RC_FAST_CLK 为 RTC BLE 定时器的源时钟

0: 禁止 RC_FAST_CLK 为 RTC BLE 定时器的源时钟 (R/W)

LP_CLKRST_LP_SEL_XTAL 配置 RTC BLE 定时器的时钟源

1: 选择 XTAL_CLK 为 RTC BLE 定时器的源时钟

0: 禁止 XTAL_CLK 为 RTC BLE 定时器的源时钟

(R/W)

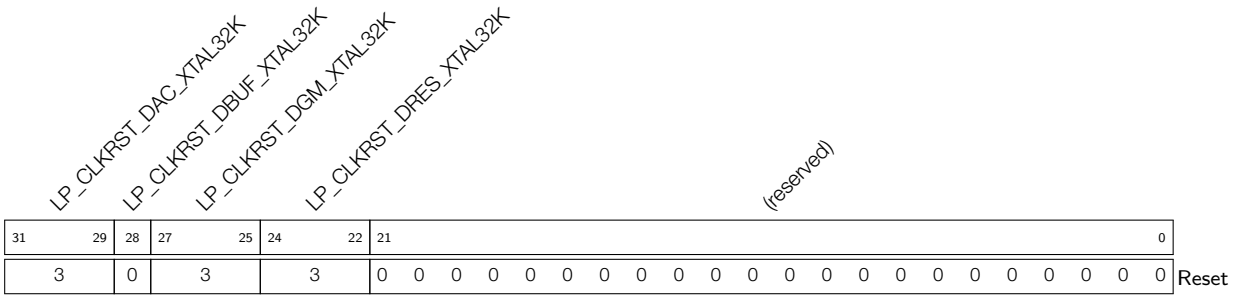
LP_CLKRST_LP_SEL_XTAL32K 配置 RTC BLE 定时器的时钟源

1: 选择 XTAL32K_CLK 为 RTC BLE 定时器的源时钟

0: 禁止 XTAL32K_CLK 为 RTC BLE 定时器的源时钟

(R/W)

Register 7.87. LP_CLKRST_XTAL32K_REG (0x002C)



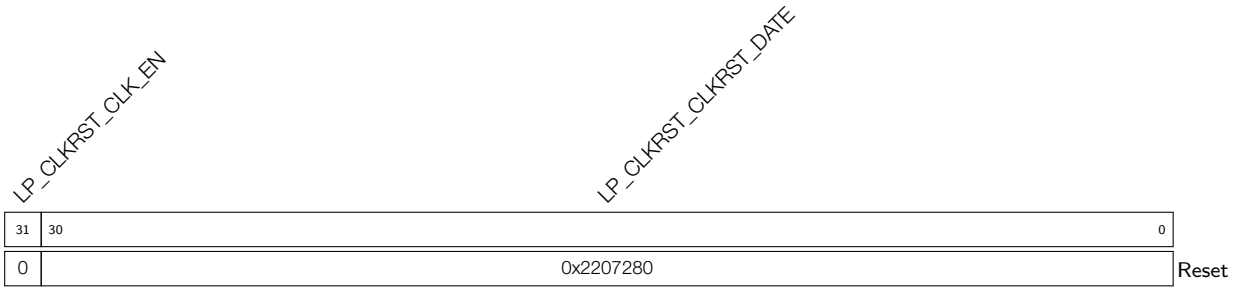
LP_CLKRST_DRES_XTAL32K 配置 DRES。(R/W)

LP_CLKRST_DGM_XTAL32K 配置 DGM。(R/W)

LP_CLKRST_DBUF_XTAL32K 配置 DBUF。(R/W)

LP_CLKRST_DAC_XTAL32K 配置 DAC。(R/W)

Register 7.88. LP_CLKRST_DATE_REG (0x03FC)



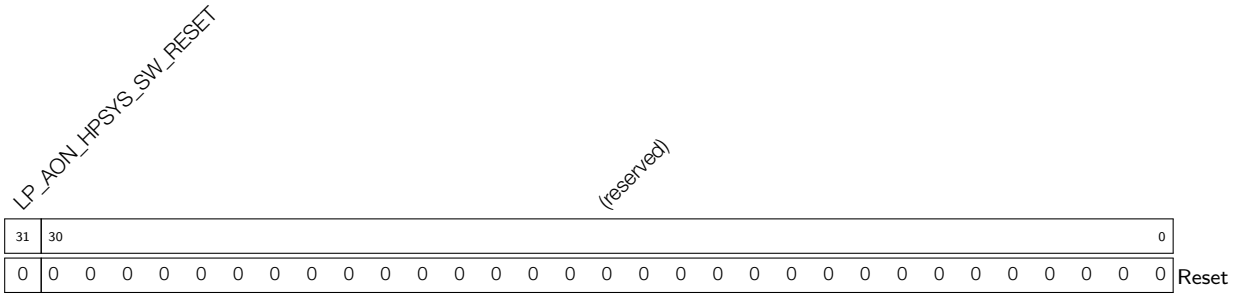
LP_CLKRST_CLKRST_DATE 版本控制寄存器。(R/W)

LP_CLKRST_CLK_EN 配置强制开启寄存器的配置的时钟。0：无效值

1：强制开启

(R/W)

Register 7.89. LP_AON_SYS_CFG_REG (0x0034)



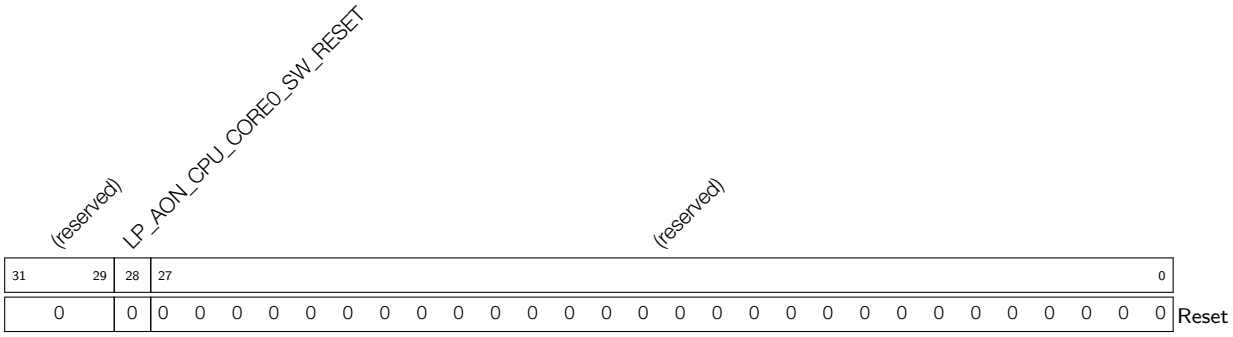
LP_AON_HPSYS_SW_RESET 配置系统的软件复位。

0：不复位

1：复位

(WT)

Register 7.90. LP_AON_CPUCORE0_CFG_REG (0x0038)



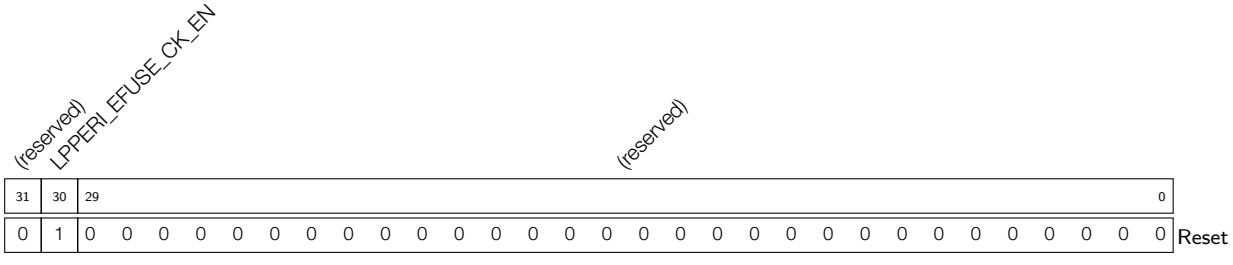
LP_AON_CPU_CORE0_SW_RESET 配置 CPU 的软件复位。

0：不复位

1：复位

(WT)

Register 7.91. LPPERI_CLK_EN_REG (0x0000)



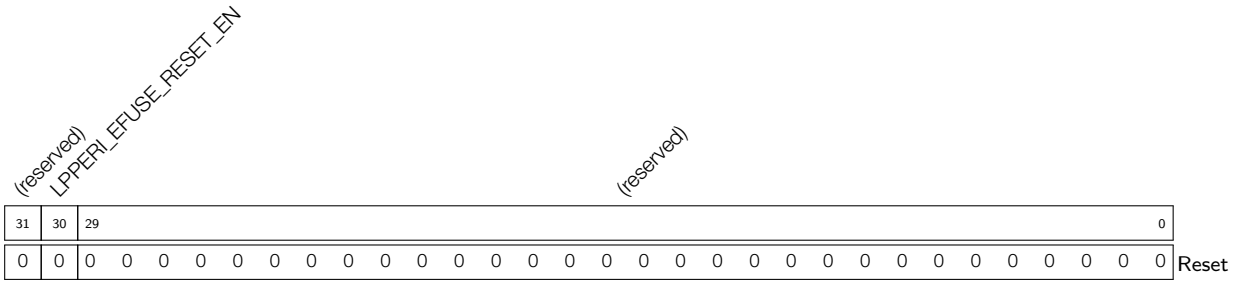
LPPERI_EFUSE_CK_EN 配置是否使能 eFuse 控制器模块的时钟门控。

0：禁止时钟信号通过

1：使能时钟信号通过

(R/W)

Register 7.92. LPPERI_RESET_EN_REG (0x0004)



LPPERI_EFUSE_RESET_EN 配置 eFuse 控制器模块的软件复位。

0：不复位

1：复位

(R/W)

8 芯片 Boot 控制

8.1 概述

芯片启动过程以及芯片的某些功能是在上电或硬件复位时由 Strapping 管脚和一些 eFuse 来确定的，具体包括以下功能：

- 控制 Boot 模式
- 控制 ROM 日志输出到 UART0/USB
- 控制 JTAG 信号源

ESP32-H2 共有三个 Strapping 管脚：

- GPIO8
- GPIO9
- GPIO25

在上电复位和欠压复位过程中（请参考章节 7 复位和时钟），硬件将采样 Strapping 管脚电平存储到锁存器中，并一直保持到芯片掉电或下一次芯片复位。Strapping 管脚的锁存值可以通过软件从寄存器 GPIO_STRAPPING 中读取。

8.2 功能描述

本小节主要介绍芯片复位时的功能以及控制该功能使用到的 Strapping 管脚和 eFuse 组合模式。

注意：
请使用本章节所介绍的组合，其它组合可能会导致不可控结果。

8.2.1 默认配置

GPIO9 默认连接内部上拉电阻。如果这一管脚没有外部连接或者连接的外部线路处于高阻抗状态，内部弱上拉将决定这一管脚输入电平的默认值，如表 8-1 所示。

表 8-1. 管脚默认上拉/下拉

管脚	默认值
GPIO8	浮空
GPIO9	上拉
GPIO25	浮空

如需改变 Strapping 管脚的默认值，用户可以应用外部下拉/上拉电阻，或者应用主机 MCU 的 GPIO 来控制 ESP32-H2 上电复位时的 Strapping 管脚电平。复位释放后，Strapping 管脚和普通管脚功能相同。

8.2.2 Boot 模式控制

复位释放后，GPIO9、GPIO8、GPIO3 和 GPIO2 共同控制 Boot 模式。表 8-2 列出了 GPIO9、GPIO8、GPIO3 和 GPIO2 的 Strapping 值及其对应的系统启动模式。

表 8-2. 系统启动模式

启动模式	GPIO9	GPIO8	GPIO3	GPIO2
SPI Boot 模式	1	x[1]	x	x
Joint Download Boot 模式[2]	0	1	x	x
SPI Download Boot 模式[3]	0	0	0	1
无效组合[4]	0	0	x	0

[1] x：任何取值均不会对结果有影响，因此可忽略。

[2] Joint Download Boot 模式：Joint Download Boot 模式下支持以下下载方式：

- USB-Serial-JTAG Download Boot
- UART Download Boot

[3] SPI Download Boot 模式：只有使用 SPI Download Boot 模式时才需要预留 GPIO3 和 GPIO2。GPIO3 和 GPIO2 默认浮空，在复位时处于高阻抗状态。

[4] 无效组合：该组合会触发意外行为，应当避免。

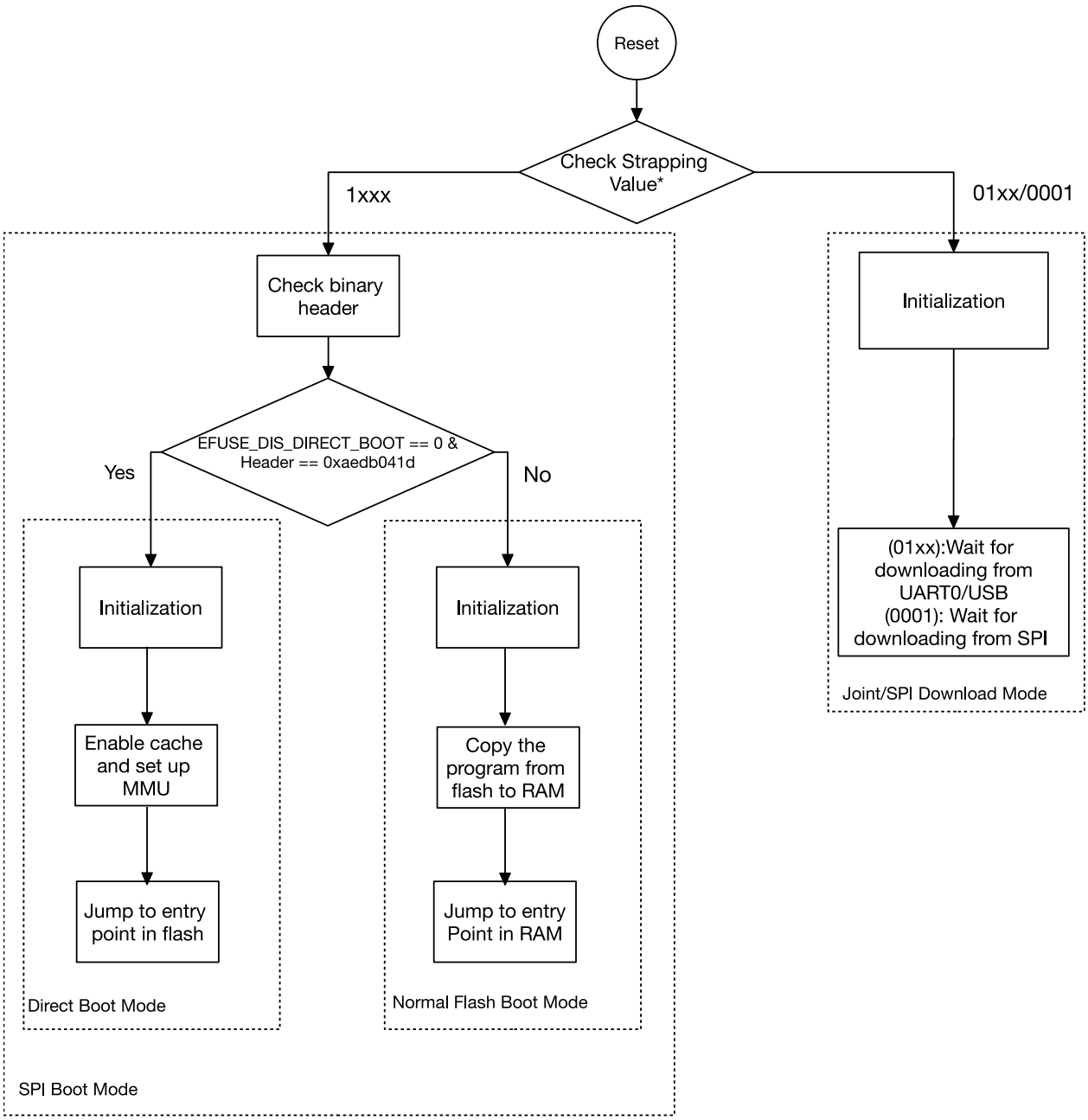
在 SPI Boot 模式下，ROM 引导加载程序通过从 SPI flash 中读取程序来启动系统。SPI Boot 模式可进一步细分为以下两种启动方式：

- 常规 flash 启动方式：支持安全启动。ROM 引导加载程序将程序从 flash 加载到 SRAM，并执行。在大多数实际应用场景中，上述执行的程序多为二级引导程序，该二级引导程序将启动最终的应用程序。
- 直接启动方式：不支持安全启动，程序直接从 flash 中运行。如需使能这一启动方式，请确保下载至 flash 的 bin 文件其前两个字为 0xaedb041d。详细的启动流程，见图 8-1。

在 Joint Download Boot 模式下，用户可通过 UART0 或 USB 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并运行 SRAM 中的程序。

在 SPI Download Boot 模式下，用户可通过 SPI 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并运行 SRAM 中的程序。

芯片启动的具体流程见图 8-1。



* **注意**：图中的"Strapping Value" "1xxx" 和"01xx/0001" 是 GPIO9、GPIO8、GPIO3 和 GPIO2 的组合值，见表 8-2。

图 8-1. 芯片启动流程

下面几个 eFuse 可用于控制启动模式的具体行为：

- EFUSE_DIS_FORCE_DOWNLOAD
  - 如果此 eFuse 设置为 0（默认），软件可通过设置 LP_AON_FORCE_DOWNLOAD_BOOT，触发 CPU 复位，将芯片启动模式强制从 SPI Boot 模式切换至 Joint Download Boot 模式。这种情况下，硬件会将寄存器 GPIO_STRAPPING[3:2] 的值从"1x" 覆盖为"01"；
  - 如果此 eFuse 设置为 1，则禁用 LP_AON_FORCE_DOWNLOAD_BOOT，寄存器 GPIO_STRAPPING 的值不受影响。
- EFUSE_DIS_DOWNLOAD_MODE

如果此 eFuse 设置为 1，则禁用 Joint Download Boot 模式，寄存器 GPIO_STRAPPING 的值则不受 LP_AON_FORCE_DOWNLOAD_BOOT 的影响。

- EFUSE_ENABLE_SECURITY_DOWNLOAD

  如果此 eFuse 设置为 1，则在 Joint Download Boot 模式下，只允许读取、写入和擦除明文 flash，不支持 SRAM 或寄存器操作。如已禁用 Joint Download Boot 模式，请忽略此 eFuse。

- EFUSE_DIS_DIRECT_BOOT

  如果此 eFuse 设置为 1，则禁用 Direct Boot 模式。

USB Serial/JTAG 控制器可将芯片从 SPI Boot 模式强制切换到 Joint Download Boot 模式，或从 Joint Download Boot 模式强制切换到 SPI Boot 模式。更多信息，请参考章节 30 *USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)*。

8.2.3 ROM 代码日志打印控制

系统在 SPI flash 启动模式下的 ROM 引导阶段，GPIO8、LP_AON_STORE4_REG[0] 与 EFUSE_UART_PRINT_CONTROL 一起控制 ROM 代码日志打印。

表 8-3. ROM 代码日志打印控制

Register[1]	eFuse[2]	GPIO8	ROM 代码日志打印
0	0(0b00)	x[3]	启动过程中，ROM 代码日志始终打印至 UART0
	1(0b01)	0	启动过程中使能打印
		1	启动过程中关闭打印
	2(0b10)	0	启动过程中关闭打印
		1	启动过程中使能打印
	3(0b11)	x	启动过程中关闭打印
1	x	x	启动过程中关闭打印

[1] Register：LP_AON_STORE4_REG[0]

[2] eFuse: EFUSE_UART_PRINT_CONTROL

[3] x: 任何取值均不会对结果有影响，因此可忽略。

ROM 代码日志上电默认打印至 UART0 和 USB Serial/JTAG 控制器，可通过置位 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 禁用 USB Serial/JTAG 控制器打印。

注意：在 EFUSE_DIS_USB_SERIAL_JTAG_ROM_PRINT 设置为 0，即可打印至 USB 的情况下，如果 USB Serial/JTAG 控制器已被禁用，则 ROM 代码将无法打印到 USB Serial/JTAG 控制器。

8.2.4 JTAG 信号源控制

GPIO25 与 EFUSE_DIS_PAD_JTAG、EFUSE_DIS_USB_JTAG 和 EFUSE_JTAG_SEL_ENABLE 一起控制 JTAG 信号源，见表 8-4。

表 8-4. JTAG 信号源控制

eFuse 1[a]	eFuse 2[b]	eFuse 3[c]	GPIO25	JTAG 信号源
0	0	0	x[d]	JTAG 信号来自 USB Serial/JTAG 控制器
		1	0	JTAG 信号来自相应管脚[e]
			1	JTAG 信号来自 USB Serial/JTAG 控制器
0	1	x	x	JTAG 信号来自相应管脚[e]
1	0	x	x	JTAG 信号来自 USB Serial/JTAG 控制器
1	1	x	x	JTAG 被禁用

[a] eFuse 1: EFUSE_DIS_PAD_JTAG

[b] eFuse 2: EFUSE_DIS_USB_JTAG

[c] eFuse 3: EFUSE_JTAG_SEL_ENABLE

[d] x：任何取值均不会对结果有影响，因此可忽略

[e] JTAG 管脚：MTDI、MTCK、MTMS 和 MTDO

9 中断矩阵 (INTMTX)

9.1 概述

ESP32-H2 中断矩阵将任一或多个外部中断源映射到 ESP-RISC-V CPU 的任一外部中断上。ESP32-H2 有 65 个外部中断源，但 CPU 只支持 28 个外部中断。因此，将这些外部中断源映射至 CPU 中断必须使用中断矩阵。

说明：
本章节只涉及将外部中断源映射到 CPU 中断，关于中断配置、中断向量表、中断服务程序推荐处理机制请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。

9.2 ESP32-H2 中断术语

下列有关中断的术语是在《ESP32-H2 技术参考手册》的语境下定义的，旨在帮助读者更好地理解本文档。

9.2.1 中断

中断指的是在特定事件或条件下，CPU 中止当前执行去处理优先级更高的任务。中断机制使 CPU 能够迅速响应特定的事件。

《ESP32-H2 技术参考手册》里涉及的“中断”是个广义的概念，可以泛指中断信号或中断源。

9.2.2 中断信号/中断源

中断信号和中断源只是定义的角度不同，实际意思相同。

从外设的角度来看，中断信号是由外设的内部中断源产生的，并发送到中断矩阵。

从中断矩阵的角度来看，它接收来自外设发送的中断信号，并将其视为中断源。然后，中断矩阵将 CPU 外部中断信号发送到 CPU。

从 CPU 的角度来看，来自中断矩阵的中断信号又变成中断源，与核心本地中断源一起发送给 CPU 中断控制器。

9.2.3 ESP32-H2 中断流

图 9-1 所示为 ESP32-H2 的中断流。

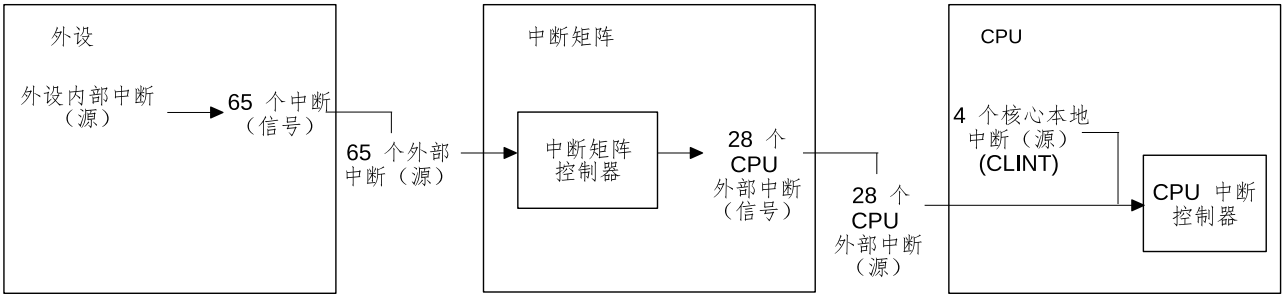


图 9-1. ESP32-H2 的中断流

9.3 特性

ESP32-H2 的中断矩阵模块具有如下特性：

- 接收 65 个外部中断源作为输入
- 生成 28 个 CPU 的外部中断作为输出
- 支持查询外部中断源当前的中断状态
- 支持将多个中断源映射到单个 CPU 中断（即共享中断）

9.4 结构概览

中断矩阵的结构如图 9-2 所示。

用户通过 中断矩阵寄存器 配置外部中断源和 CPU 中断的映射关系。图 9-2 中的中断矩阵控制器负责映射，并将每个外部中断源的中断状态给到中断矩阵寄存器里的中断状态寄存器。

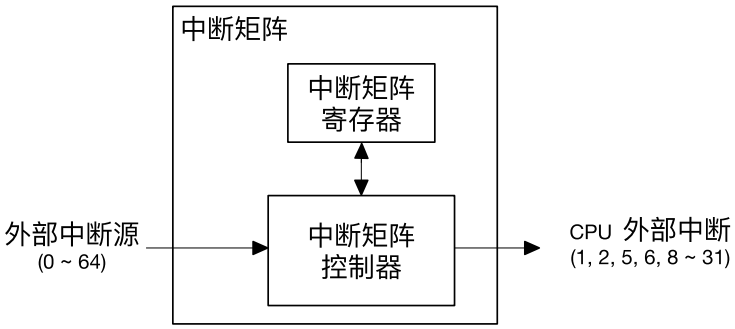


图 9-2. 中断矩阵结构图

9.5 功能描述

9.5.1 外部中断源

ESP32-H2 共有 65 个外部中断源。表 9-1 列出了所有外部中断源及其对应的中断源映射寄存器与中断状态寄存器。

- “No.”：表示外部中断源序号，范围：0 ~ 64
- “章节”：详细描述外部中断源的章节
- “中断源”：外部中断源名称
- “中断源映射寄存器”：用于将外部中断源分配至 CPU 外部中断
- “中断状态寄存器”：用于读取中断源的中断状态
  - “中断状态寄存器 - 位”：表示在中断状态寄存器中的比特位置，用于记录相应中断源的状态
  - “中断状态寄存器 - 名称”：表示中断状态寄存器的名称

表 9-1. CPU 外部中断源映射寄存器、外部中断状态寄存器、外部中断源

No.	章节	中断源	中断源映射寄存器	中断状态寄存器	
				位	名称
0	低功耗管理 (RTC_CNTL) [to be added later]	PMU_INTR	INTMTX_CORE0_PMU_INTR_MAP_REG	0	INTMTX_CORE0_INT_STATUS_0_REG
1	eFuse 控制器 (EFUSE)	EFUSE_INTR	INTMTX_CORE0_EFUSE_INTR_MAP_REG	1	
2	低功耗管理 (RTC_CNTL) [to be added later]	LP_RTC_TIMER_INTR	INTMTX_CORE0_LP_RTC_TIMER_INTR_MAP_REG	2	
3	n/a	保留	保留	3	
4	低功耗管理 (RTC_CNTL) [to be added later]	LP_WDT_INTR	INTMTX_CORE0_LP_WDT_INTR_MAP_REG	4	
5	系统寄存器	LP_PERI_TIMEOUT_INTR	INTMTX_CORE0_LP_PERI_TIMEOUT_INTR_MAP_REG	5	
6	访问权限管理 (APM)	LP_APM_M0_INTR	INTMTX_CORE0_LP_APM_M0_INTR_MAP_REG	6	
7	ESP-RISC-V CPU	CPU_INTR_FROM_CPU_0	INTMTX_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG	7	
8	ESP-RISC-V CPU	CPU_INTR_FROM_CPU_1	INTMTX_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG	8	
9	ESP-RISC-V CPU	CPU_INTR_FROM_CPU_2	INTMTX_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG	9	
10	ESP-RISC-V CPU	CPU_INTR_FROM_CPU_3	INTMTX_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG	10	
11	辅助调试 (ASSIST_DEBUG、MEM_MONITOR)	ASSIST_DEBUG_INTR	INTMTX_CORE0_ASSIST_DEBUG_INTR_MAP_REG	11	
12	ESP-RISC-V CPU	TRACE_INTR	INTMTX_CORE0_TRACE_INTR_MAP_REG	12	
13	Cache [to be added later]	CACHE_INTR	INTMTX_CORE0_CACHE_INTR_MAP_REG	13	
14	系统寄存器	CPU_PERI_TIMEOUT_INTR	INTMTX_CORE0_CPU_PERI_TIMEOUT_INTR_MAP_REG	14	
15	n/a	保留	保留	15	
16	n/a	保留	保留	16	
17	n/a	保留	保留	17	
18	n/a	保留	保留	18	
19	n/a	保留	保留	19	
20	n/a	保留	保留	20	
21	n/a	保留	保留	21	
22	IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)	GPIO_INTERRUPT_PRO	INTMTX_CORE0_GPIO_INTERRUPT_PRO_MAP_REG	22	
23	IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)	保留	保留	23	
24	低功耗管理 (RTC_CNTL) [to be added later]	PAU_INTR	INTMTX_CORE0_PAU_INTR_MAP_REG	24	
25	系统寄存器	HP_PERI_TIMEOUT_INTR	INTMTX_CORE0_HP_PERI_TIMEOUT_INTR_MAP_REG	25	
26	访问权限管理 (APM)	HP_APM_M0_INTR	INTMTX_CORE0_HP_APM_M0_INTR_MAP_REG	26	
27	访问权限管理 (APM)	HP_APM_M1_INTR	INTMTX_CORE0_HP_APM_M1_INTR_MAP_REG	27	
28	访问权限管理 (APM)	HP_APM_M2_INTR	INTMTX_CORE0_HP_APM_M2_INTR_MAP_REG	28	
29	访问权限管理 (APM)	HP_APM_M3_INTR	INTMTX_CORE0_HP_APM_M3_INTR_MAP_REG	29	
30	n/a	保留	保留	30	
31	I2S 控制器 (I2S)	I2S_INTR	INTMTX_CORE0_I2S_INTR_MAP_REG	31	
32	UART 控制器 (UART)	UHCI0_INTR	INTMTX_CORE0_UHCI0_INTR_MAP_REG	0	
33	UART 控制器 (UART)	UART0_INTR	INTMTX_CORE0_UART0_INTR_MAP_REG	1	
34	UART 控制器 (UART)	UART1_INTR	INTMTX_CORE0_UART1_INTR_MAP_REG	2	
35	LED PWM 控制器 (LEDC)	LEDC_INTR	INTMTX_CORE0_LEDC_INTR_MAP_REG	3	
36	双线汽车接口 (TWAI)	TWAI0_INTR	INTMTX_CORE0_TWAI0_INTR_MAP_REG	4	

GoBack

No.	章节	中断源	中断源映射寄存器	中断状态寄存器	
				位	名称
37	USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)	USB_SERIAL_JTAG_INTR	INTMTX_CORE0_USB_INTR_MAP_REG	5	
38	红外遥控 (RMT)	RMT_INTR	INTMTX_CORE0_RMT_INTR_MAP_REG	6	
39	I2C 控制器 (I2C)	I2C_EXT0_INTR	INTMTX_CORE0_I2C_EXT0_INTR_MAP_REG	7	
40	I2C 控制器 (I2C)	I2C_EXT1_INTR	INTMTX_CORE0_I2C_EXT1_INTR_MAP_REG	8	
41	定时器组 (TIMG)	TG0_T0_INTR	INTMTX_CORE0_TG0_T0_INTR_MAP_REG	9	
42	定时器组 (TIMG)	TG0_WDT_INTR	INTMTX_CORE0_TG0_WDT_INTR_MAP_REG	10	
43	定时器组 (TIMG)	TG1_T0_INTR	INTMTX_CORE0_TG1_T0_INTR_MAP_REG	11	
44	定时器组 (TIMG)	TG1_WDT_INTR	INTMTX_CORE0_TG1_WDT_INTR_MAP_REG	12	
45	系统定时器 (SYSTIMER)	SYSTIMER_TARGET0_INTR	INTMTX_CORE0_SYSTIMER_TARGET0_INTR_MAP_REG	13	
46	系统定时器 (SYSTIMER)	SYSTIMER_TARGET1_INTR	INTMTX_CORE0_SYSTIMER_TARGET1_INTR_MAP_REG	14	
47	系统定时器 (SYSTIMER)	SYSTIMER_TARGET2_INTR	INTMTX_CORE0_SYSTIMER_TARGET2_INTR_MAP_REG	15	
48	SAR ADC 转换器与温度传感器	APB_ADC_INTR	INTMTX_CORE0_APB_ADC_INTR_MAP_REG	16	
49	电机控制脉宽调制器 (MCPWM)	PWM_INTR	INTMTX_CORE0_PWM_INTR_MAP_REG	17	
50	脉冲计数控制器 (PCNT)	PCNT_INTR	INTMTX_CORE0_PCNT_INTR_MAP_REG	18	
51	并行 IO 控制器 (PARL_IO)	PARL_IO_TX_INTR	INTMTX_CORE0_PARL_IO_TX_INTR_MAP_REG	19	
52	并行 IO 控制器 (PARL_IO)	PARL_IO_RX_INTR	INTMTX_CORE0_PARL_IO_RX_INTR_MAP_REG	20	
53	通用 DMA 控制器 (GDMA)	GDMA_IN_CH0_INTR	INTMTX_CORE0_DMA_IN_CH0_INTR_MAP_REG	21	
54	通用 DMA 控制器 (GDMA)	GDMA_IN_CH1_INTR	INTMTX_CORE0_DMA_IN_CH1_INTR_MAP_REG	22	
55	通用 DMA 控制器 (GDMA)	GDMA_IN_CH2_INTR	INTMTX_CORE0_DMA_IN_CH2_INTR_MAP_REG	23	
56	通用 DMA 控制器 (GDMA)	GDMA_OUT_CH0_INTR	INTMTX_CORE0_DMA_OUT_CH0_INTR_MAP_REG	24	
57	通用 DMA 控制器 (GDMA)	GDMA_OUT_CH1_INTR	INTMTX_CORE0_DMA_OUT_CH1_INTR_MAP_REG	25	
58	通用 DMA 控制器 (GDMA)	GDMA_OUT_CH2_INTR	INTMTX_CORE0_DMA_OUT_CH2_INTR_MAP_REG	26	
59	SPI 控制器 (SPI)	GPSPI2_INTR	INTMTX_CORE0_GPSPI2_INTR_MAP_REG	27	
60	AES 加速器 (AES)	AES_INTR	INTMTX_CORE0_AES_INTR_MAP_REG	28	
61	SHA 加速器 (SHA)	SHA_INTR	INTMTX_CORE0_SHA_INTR_MAP_REG	29	
62	RSA 加速器 (RSA)	RSA_INTR	INTMTX_CORE0_RSA_INTR_MAP_REG	30	
63	ECC 加速器 (ECC)	ECC_INTR	INTMTX_CORE0_ECC_INTR_MAP_REG	31	
64	椭圆曲线数字签名算法 (ECDSA)	ECDSA_INTR	INTMTX_CORE0_ECDSA_INTR_MAP_REG	0	INTMTX_CORE0_INT_STATUS_2_REG

9.5.2 CPU 中断

ESP32-H2 采用非 RISC-V 标准规范中断机制。CPU 共有 32 个中断号 (0 ~ 31)，其中 0、3、4、7 号中断仅在 CPU 内部使用，其余 28 个中断 (中断号为 1、2、5、6、8 ~ 31) 可供中断矩阵使用。每个中断：

- 优先级可设置为 1 ~ 15（数字越大优先级越高）；
- 可配置为电平触发或者边沿触发；
- 可通过设置中断阈值屏蔽低优先级的中断。

说明：
CPU 中断的具体功能及配置流程见章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。CPU 中断的配置寄存器请见本章节的 9.6.2 中断优先级寄存器列表。

9.5.3 分配外部中断源至 CPU 外部中断

在本小节中，我们将使用以下术语描述中断矩阵相关操作：

- *SOURCE*：代表某个外部中断源，详见表 9-1。
- INTMTX_CORE0_*SOURCE*_INTR_MAP_REG：*SOURCE* 的中断源映射寄存器。
- Num_P：CPU 中断编号，范围：1、2、5、6、8 ~ 31。
- Interrupt_P：表示中断号为 Num_P 的 CPU 中断。

9.5.3.1 分配一个外部中断源 SOURCE 至 CPU 外部中断

将 *SOURCE* 对应的寄存器 INTMTX_CORE0_*SOURCE*_INTR_MAP_REG 配成 Num_P，即可将该中断源分配至序号为 Num_P 的 CPU 中断 (Interrupt_P)。

9.5.3.2 分配多个外部中断源 SOURCE 至 CPU 外部中断

将各个中断源对应的寄存器 INTMTX_CORE0_*SOURCE*_INTR_MAP_REG 均配置成相同的 Num_P，即可将多个 *SOURCE* 分配至同一 CPU 外部中断 Interrupt_P。上述任一外设中断均会触发 CPU 外部中断 Interrupt_P。待中断触发后，CPU 需查询中断状态寄存器，判断产生中断的外设。更多信息，见章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。

9.5.3.3 关闭 CPU 外部中断源 SOURCE

将中断源对应的寄存器 INTMTX_CORE0_*SOURCE*_INTR_MAP_REG 配置成 0，即可关闭外部中断源。

9.5.4 查询外部中断源 SOURCE 当前的中断状态

配置开启 *SOURCE* 后，读取寄存器 INTMTX_CORE0_INT_STATUS_*n*_REG（只读）中特定位的值可以获取 *SOURCE* 当前的中断状态。寄存器 INTMTX_CORE0_INT_STATUS_*n*_REG 与 *SOURCE* 的对应关系如表 9-1 所示。

9.6 寄存器列表

9.6.1 中断矩阵寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
外设中断源映射寄存器			
INTMTX_CORE0_PMU_INTR_MAP_REG	PMU_INTR 映射寄存器	0x0000	R/W
INTMTX_CORE0_EFUSE_INTR_MAP_REG	EFUSE_INTR 映射寄存器	0x0004	R/W
INTMTX_CORE0_LP_RTC_TIMER_INTR_MAP_REG	LP_RTC_TIMER_INTR 映射寄存器	0x0008	R/W
INTMTX_CORE0_LP_WDT_INTR_MAP_REG	LP_WDT_INTR 映射寄存器	0x0010	R/W
INTMTX_CORE0_LP_PERI_TIMEOUT_INTR_MAP_REG	LP_PERI_TIMEOUT_INTR 映射寄存器	0x0014	R/W
INTMTX_CORE0_LP_APM_M0_INTR_MAP_REG	LP_APM_M0_INTR 映射寄存器	0x0018	R/W
INTMTX_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 映射寄存器	0x001C	R/W
INTMTX_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 映射寄存器	0x0020	R/W
INTMTX_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 映射寄存器	0x0024	R/W
INTMTX_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 映射寄存器	0x0028	R/W
INTMTX_CORE0_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR 映射寄存器	0x002C	R/W
INTMTX_CORE0_TRACE_INTR_MAP_REG	TRACE_INTR 映射寄存器	0x0030	R/W
INTMTX_CORE0_CACHE_INTR_MAP_REG	CACHE_INTR 映射寄存器	0x0034	R/W
INTMTX_CORE0_CPU_PERI_TIMEOUT_INTR_MAP_REG	CPU_PERI_TIMEOUT_INTR 映射寄存器	0x0038	R/W
INTMTX_CORE0_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO 映射寄存器	0x0058	R/W
INTMTX_CORE0_PAU_INTR_MAP_REG	PAU_INTR 映射寄存器	0x0060	R/W
INTMTX_CORE0_HP_PERI_TIMEOUT_INTR_MAP_REG	HP_PERI_TIMEOUT_INTR 映射寄存器	0x0064	R/W
INTMTX_CORE0_HP_APM_M0_INTR_MAP_REG	HP_APM_M0_INTR 映射寄存器	0x0068	R/W
INTMTX_CORE0_HP_APM_M1_INTR_MAP_REG	HP_APM_M1_INTR 映射寄存器	0x006C	R/W
INTMTX_CORE0_HP_APM_M2_INTR_MAP_REG	HP_APM_M2_INTR 映射寄存器	0x0070	R/W
INTMTX_CORE0_HP_APM_M3_INTR_MAP_REG	HP_APM_M3_INTR 映射寄存器	0x0074	R/W
INTMTX_CORE0_I2S_INTR_MAP_REG	I2S_INTR 映射寄存器	0x007C	R/W

名称	描述	地址	访问
INTMTX_CORE0_UHCI0_INTR_MAP_REG	UHCI0_INTR 映射寄存器	0x0080	R/W
INTMTX_CORE0_UART0_INTR_MAP_REG	UART0_INTR 映射寄存器	0x0084	R/W
INTMTX_CORE0_UART1_INTR_MAP_REG	UART1_INTR 映射寄存器	0x0088	R/W
INTMTX_CORE0_LEDC_INTR_MAP_REG	LEDC_INTR 映射寄存器	0x008C	R/W
INTMTX_CORE0_TWAI0_INTR_MAP_REG	TWAI0_INTR 映射寄存器	0x0090	R/W
INTMTX_CORE0_USB_INTR_MAP_REG	USB_SERIAL_JTAG_INTR 映射寄存器	0x0094	R/W
INTMTX_CORE0_RMT_INTR_MAP_REG	RMT_INTR 映射寄存器	0x0098	R/W
INTMTX_CORE0_I2C_EXT0_INTR_MAP_REG	I2C_EXT0_INTR 映射寄存器	0x009C	R/W
INTMTX_CORE0_I2C_EXT1_INTR_MAP_REG	I2C_EXT1_INTR 映射寄存器	0x00A0	R/W
INTMTX_CORE0_TG0_T0_INTR_MAP_REG	TG0_T0_INTR 映射寄存器	0x00A4	R/W
INTMTX_CORE0_TG0_WDT_INTR_MAP_REG	TG0_WDT_INTR 映射寄存器	0x00A8	R/W
INTMTX_CORE0_TG1_T0_INTR_MAP_REG	TG1_T0_INTR 映射寄存器	0x00AC	R/W
INTMTX_CORE0_TG1_WDT_INTR_MAP_REG	TG1_WDT_INTR 映射寄存器	0x00B0	R/W
INTMTX_CORE0_SYSTIMER_TARGET0_INTR_MAP_REG	SYSTIMER_TARGET0_INTR 映射寄存器	0x00B4	R/W
INTMTX_CORE0_SYSTIMER_TARGET1_INTR_MAP_REG	SYSTIMER_TARGET1_INTR 映射寄存器	0x00B8	R/W
INTMTX_CORE0_SYSTIMER_TARGET2_INTR_MAP_REG	SYSTIMER_TARGET2_INTR 映射寄存器	0x00BC	R/W
INTMTX_CORE0_APB_ADC_INTR_MAP_REG	APB_ADC_INTR 映射寄存器	0x00C0	R/W
INTMTX_CORE0_PWM_INTR_MAP_REG	PWM_INTR 映射寄存器	0x00C4	R/W
INTMTX_CORE0_PCNT_INTR_MAP_REG	PCNT_INTR 映射寄存器	0x00C8	R/W
INTMTX_CORE0_PARL_IO_TX_INTR_MAP_REG	PARL_IO_TX_INTR 映射寄存器	0x00CC	R/W
INTMTX_CORE0_PARL_IO_RX_INTR_MAP_REG	PARL_IO_RX_INTR 映射寄存器	0x00D0	R/W
INTMTX_CORE0_DMA_IN_CH0_INTR_MAP_REG	DMA_IN_CH0_INTR 映射寄存器	0x00D4	R/W
INTMTX_CORE0_DMA_IN_CH1_INTR_MAP_REG	DMA_IN_CH1_INTR 映射寄存器	0x00D8	R/W
INTMTX_CORE0_DMA_IN_CH2_INTR_MAP_REG	DMA_IN_CH2_INTR 映射寄存器	0x00DC	R/W
INTMTX_CORE0_DMA_OUT_CH0_INTR_MAP_REG	DMA_OUT_CH0_INTR 映射寄存器	0x00E0	R/W
INTMTX_CORE0_DMA_OUT_CH1_INTR_MAP_REG	DMA_OUT_CH1_INTR 映射寄存器	0x00E4	R/W
INTMTX_CORE0_DMA_OUT_CH2_INTR_MAP_REG	DMA_OUT_CH2_INTR 映射寄存器	0x00E8	R/W
INTMTX_CORE0_GPSPI2_INTR_MAP_REG	GPSPI2_INTR 映射寄存器	0x00EC	R/W
INTMTX_CORE0_AES_INTR_MAP_REG	AES_INTR 映射寄存器	0x00F0	R/W

名称	描述	地址	访问
INTMTX_CORE0_SHA_INTR_MAP_REG	SHA_INTR 映射寄存器	0x00F4	R/W
INTMTX_CORE0_RSA_INTR_MAP_REG	RSA_INTR 映射寄存器	0x00F8	R/W
INTMTX_CORE0_ECC_INTR_MAP_REG	ECC_INTR 映射寄存器	0x00FC	R/W
INTMTX_CORE0_ECDSA_INTR_MAP_REG	ECDSA_INTR 映射寄存器	0x0100	R/W
中断状态寄存器			
INTMTX_CORE0_INT_STATUS_0_REG	中断源 0 ~ 31 的状态寄存器	0x0104	RO
INTMTX_CORE0_INT_STATUS_1_REG	中断源 32 ~ 63 的状态寄存器	0x0108	RO
INTMTX_CORE0_INT_STATUS_2_REG	中断源 64 的状态寄存器	0x010C	RO
版本控制寄存器			
INTMTX_CORE0_INTERRUPT_REG_DATE_REG	版本控制寄存器	0x07FC	R/W

9.6.2 中断优先级寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
INTPRI_CORE0_CPU_INT_ENABLE_REG	CPU 中断使能配置寄存器	0x0000	R/W
INTPRI_CORE0_CPU_INT_TYPE_REG	CPU 中断类型配置寄存器	0x0004	R/W
INTPRI_CORE0_CPU_INT_EIP_STATUS_REG	CPU 中断阻塞状态寄存器	0x0008	RO
INTPRI_CORE0_CPU_INT_PRI_0_REG	CPU 中断优先级配置寄存器 0	0x000C	R/W
INTPRI_CORE0_CPU_INT_PRI_1_REG	CPU 中断优先级配置寄存器 1	0x0010	R/W
INTPRI_CORE0_CPU_INT_PRI_2_REG	CPU 中断优先级配置寄存器 2	0x0014	R/W
INTPRI_CORE0_CPU_INT_PRI_3_REG	CPU 中断优先级配置寄存器 3	0x0018	R/W
INTPRI_CORE0_CPU_INT_PRI_4_REG	CPU 中断优先级配置寄存器 4	0x001C	R/W
INTPRI_CORE0_CPU_INT_PRI_5_REG	CPU 中断优先级配置寄存器 5	0x0020	R/W
INTPRI_CORE0_CPU_INT_PRI_6_REG	CPU 中断优先级配置寄存器 6	0x0024	R/W
INTPRI_CORE0_CPU_INT_PRI_7_REG	CPU 中断优先级配置寄存器 7	0x0028	R/W

名称	描述	地址	访问
INTPRI_CORE0_CPU_INT_PRI_8_REG	CPU 中断优先级配置寄存器 8	0x002C	R/W
INTPRI_CORE0_CPU_INT_PRI_9_REG	CPU 中断优先级配置寄存器 9	0x0030	R/W
INTPRI_CORE0_CPU_INT_PRI_10_REG	CPU 中断优先级配置寄存器 10	0x0034	R/W
INTPRI_CORE0_CPU_INT_PRI_11_REG	CPU 中断优先级配置寄存器 11	0x0038	R/W
INTPRI_CORE0_CPU_INT_PRI_12_REG	CPU 中断优先级配置寄存器 12	0x003C	R/W
INTPRI_CORE0_CPU_INT_PRI_13_REG	CPU 中断优先级配置寄存器 13	0x0040	R/W
INTPRI_CORE0_CPU_INT_PRI_14_REG	CPU 中断优先级配置寄存器 14	0x0044	R/W
INTPRI_CORE0_CPU_INT_PRI_15_REG	CPU 中断优先级配置寄存器 15	0x0048	R/W
INTPRI_CORE0_CPU_INT_PRI_16_REG	CPU 中断优先级配置寄存器 16	0x004C	R/W
INTPRI_CORE0_CPU_INT_PRI_17_REG	CPU 中断优先级配置寄存器 17	0x0050	R/W
INTPRI_CORE0_CPU_INT_PRI_18_REG	CPU 中断优先级配置寄存器 18	0x0054	R/W
INTPRI_CORE0_CPU_INT_PRI_19_REG	CPU 中断优先级配置寄存器 19	0x0058	R/W
INTPRI_CORE0_CPU_INT_PRI_20_REG	CPU 中断优先级配置寄存器 20	0x005C	R/W
INTPRI_CORE0_CPU_INT_PRI_21_REG	CPU 中断优先级配置寄存器 21	0x0060	R/W
INTPRI_CORE0_CPU_INT_PRI_22_REG	CPU 中断优先级配置寄存器 22	0x0064	R/W
INTPRI_CORE0_CPU_INT_PRI_23_REG	CPU 中断优先级配置寄存器 23	0x0068	R/W
INTPRI_CORE0_CPU_INT_PRI_24_REG	CPU 中断优先级配置寄存器 24	0x006C	R/W
INTPRI_CORE0_CPU_INT_PRI_25_REG	CPU 中断优先级配置寄存器 25	0x0070	R/W
INTPRI_CORE0_CPU_INT_PRI_26_REG	CPU 中断优先级配置寄存器 26	0x0074	R/W
INTPRI_CORE0_CPU_INT_PRI_27_REG	CPU 中断优先级配置寄存器 27	0x0078	R/W
INTPRI_CORE0_CPU_INT_PRI_28_REG	CPU 中断优先级配置寄存器 28	0x007C	R/W
INTPRI_CORE0_CPU_INT_PRI_29_REG	CPU 中断优先级配置寄存器 29	0x0080	R/W
INTPRI_CORE0_CPU_INT_PRI_30_REG	CPU 中断优先级配置寄存器 30	0x0084	R/W
INTPRI_CORE0_CPU_INT_PRI_31_REG	CPU 中断优先级配置寄存器 31	0x0088	R/W
INTPRI_CORE0_CPU_INT_THRESH_REG	CPU 中断阈值配置寄存器	0x008C	R/W
INTPRI_CORE0_CPU_INT_CLEAR_REG	CPU 中断清零寄存器	0x00A8	R/W
中断寄存器			
INTPRI_CPU_INTR_FROM_CPU_0_REG	CPU_INTR_FROM_CPU_0 中断配置寄存器	0x0090	R/W
INTPRI_CPU_INTR_FROM_CPU_1_REG	CPU_INTR_FROM_CPU_1 中断配置寄存器	0x0094	R/W

GoBack

名称	描述	地址	访问
INTPRI_CPU_INTR_FROM_CPU_2_REG	CPU_INTR_FROM_CPU_2 中断配置寄存器	0x0098	R/W
INTPRI_CPU_INTR_FROM_CPU_3_REG	CPU_INTR_FROM_CPU_3 中断配置寄存器	0x009C	R/W
版本寄存器			
INTPRI_DATE_REG	版本控制寄存器	0x00A0	R/W

GoBack

9.7 寄存器

9.7.1 中断矩阵寄存器

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

Register 9.1. INTMTX_CORE0_PMU_INTR_MAP_REG (0x0000)

Register 9.2. INTMTX_CORE0_EFUSE_INTR_MAP_REG (0x0004)

Register 9.3. INTMTX_CORE0_LP_RTC_TIMER_INTR_MAP_REG (0x0008)

Register 9.4. INTMTX_CORE0_LP_WDT_INTR_MAP_REG (0x0010)

Register 9.5. INTMTX_CORE0_LP_PERI_TIMEOUT_INTR_MAP_REG (0x0014)

Register 9.6. INTMTX_CORE0_LP_APM_M0_INTR_MAP_REG (0x0018)

Register 9.7. INTMTX_CORE0_CPU_INTR_FROM_CPU_0_MAP_REG (0x001C)

Register 9.8. INTMTX_CORE0_CPU_INTR_FROM_CPU_1_MAP_REG (0x0020)

Register 9.9. INTMTX_CORE0_CPU_INTR_FROM_CPU_2_MAP_REG (0x0024)

Register 9.10. INTMTX_CORE0_CPU_INTR_FROM_CPU_3_MAP_REG (0x0028)

Register 9.11. INTMTX_CORE0_ASSIST_DEBUG_INTR_MAP_REG (0x002C)

Register 9.12. INTMTX_CORE0_TRACE_INTR_MAP_REG (0x0030)

Register 9.13. INTMTX_CORE0_CACHE_INTR_MAP_REG (0x0034)

Register 9.14. INTMTX_CORE0_CPU_PERI_TIMEOUT_INTR_MAP_REG (0x0038)

Register 9.15. INTMTX_CORE0_GPIO_INTERRUPT_PRO_MAP_REG (0x0058)

Register 9.16. INTMTX_CORE0_PAU_INTR_MAP_REG (0x0060)

Register 9.17. INTMTX_CORE0_HP_PERI_TIMEOUT_INTR_MAP_REG (0x0064)

Register 9.18. INTMTX_CORE0_HP_APM_M0_INTR_MAP_REG (0x0068)

Register 9.19. INTMTX_CORE0_HP_APM_M1_INTR_MAP_REG (0x006C)

Register 9.20. INTMTX_CORE0_HP_APM_M2_INTR_MAP_REG (0x0070)

Register 9.21. INTMTX_CORE0_HP_APM_M3_INTR_MAP_REG (0x0074)

Register 9.22. INTMTX_CORE0_I2S_INTR_MAP_REG (0x007C)

Register 9.23. INTMTX_CORE0_UHCI0_INTR_MAP_REG (0x0080)

Register 9.24. INTMTX_CORE0_UART0_INTR_MAP_REG (0x0084)

Register 9.25. INTMTX_CORE0_UART1_INTR_MAP_REG (0x0088)

Register 9.26. INTMTX_CORE0_LEDC_INTR_MAP_REG (0x008C)

Register 9.27. INTMTX_CORE0_TWAI0_INTR_MAP_REG (0x0090)

Register 9.28. INTMTX_CORE0_USB_INTR_MAP_REG (0x0094)

Register 9.29. INTMTX_CORE0_RMT_INTR_MAP_REG (0x0098)

Register 9.30. INTMTX_CORE0_I2C_EXT0_INTR_MAP_REG (0x009C)

Register 9.31. INTMTX_CORE0_I2C_EXT1_INTR_MAP_REG (0x00A0)

Register 9.32. INTMTX_CORE0_TG0_T0_INTR_MAP_REG (0x00A4)

Register 9.33. INTMTX_CORE0_TG0_WDT_INTR_MAP_REG (0x00A8)

Register 9.34. INTMTX_CORE0_TG1_T0_INTR_MAP_REG (0x00AC)

Register 9.35. INTMTX_CORE0_TG1_WDT_INTR_MAP_REG (0x00B0)

Register 9.36. INTMTX_CORE0_SYSTIMER_TARGET0_INTR_MAP_REG (0x00B4)

Register 9.37. INTMTX_CORE0_SYSTIMER_TARGET1_INTR_MAP_REG (0x00B8)

Register 9.38. INTMTX_CORE0_SYSTIMER_TARGET2_INTR_MAP_REG (0x00BC)

Register 9.39. INTMTX_CORE0_APB_ADC_INTR_MAP_REG (0x00C0)

Register 9.40. INTMTX_CORE0_PWM_INTR_MAP_REG (0x00C4)

Register 9.41. INTMTX_CORE0_PCNT_INTR_MAP_REG (0x00C8)

Register 9.42. INTMTX_CORE0_PARL_IO_TX_INTR_MAP_REG (0x00CC)

Register 9.43. INTMTX_CORE0_PARL_IO_RX_INTR_MAP_REG (0x00D0)

Register 9.44. INTMTX_CORE0_DMA_IN_CH0_INTR_MAP_REG (0x00D4)

Register 9.45. INTMTX_CORE0_DMA_IN_CH1_INTR_MAP_REG (0x00D8)

Register 9.46. INTMTX_CORE0_DMA_IN_CH2_INTR_MAP_REG (0x00DC)

Register 9.47. INTMTX_CORE0_DMA_OUT_CH0_INTR_MAP_REG (0x00E0)

Register 9.48. INTMTX_CORE0_DMA_OUT_CH1_INTR_MAP_REG (0x00E4)

Register 9.49. INTMTX_CORE0_DMA_OUT_CH2_INTR_MAP_REG (0x00E8)

Register 9.50. INTMTX_CORE0_GPSPI2_INTR_MAP_REG (0x00EC)

Register 9.51. INTMTX_CORE0_AES_INTR_MAP_REG (0x00F0)

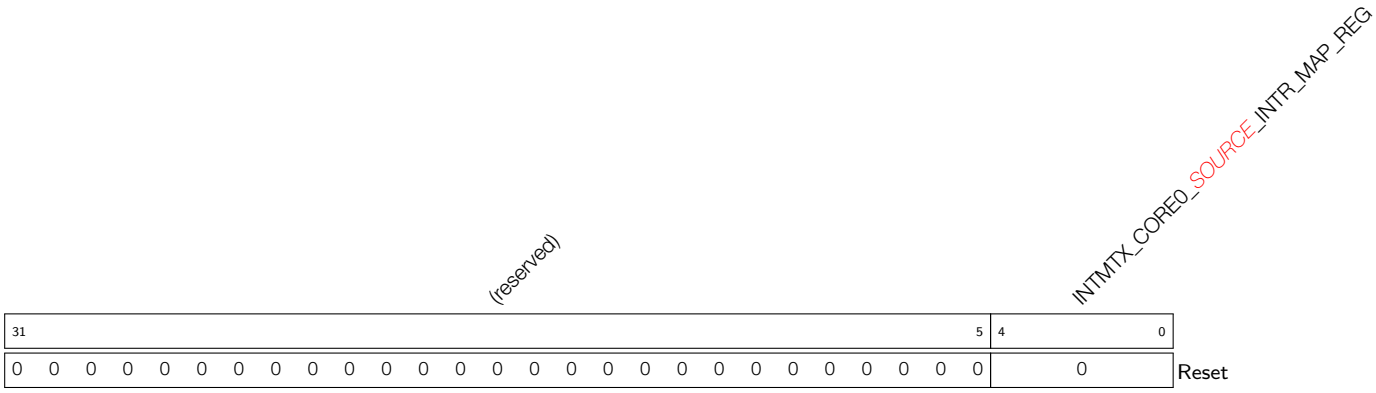
Register 9.52. INTMTX_CORE0_SHA_INTR_MAP_REG (0x00F4)

Register 9.53. INTMTX_CORE0_RSA_INTR_MAP_REG (0x00F8)

Register 9.54. INTMTX_CORE0_ECC_INTR_MAP_REG (0x00FC)

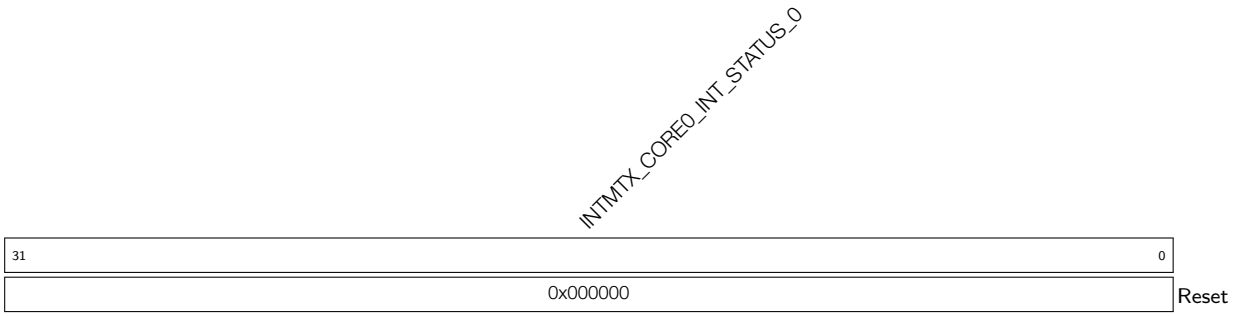
Register 9.55. INTMTX_CORE0_ECDSA_INTR_MAP_REG (0x0100)

Register 9.56. INTMTX_CORE0_*SOURCE*_INTR_MAP_REG (0x0000 - 0x0100)



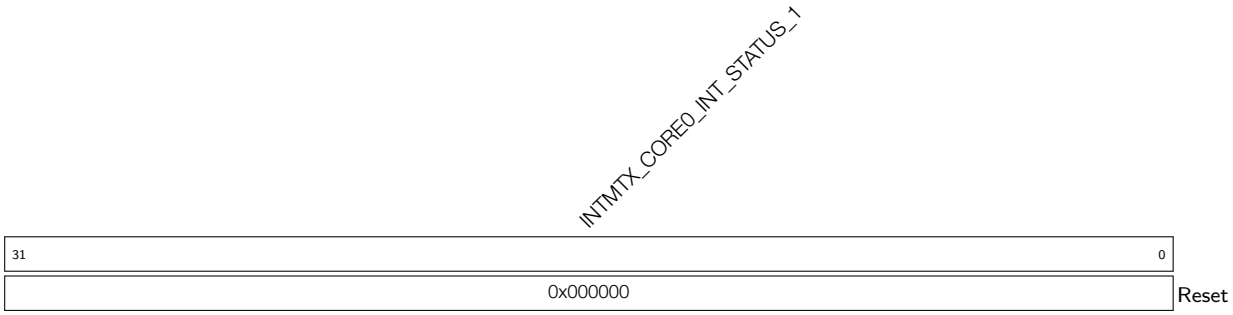
INTMTX_CORE0_*SOURCE*_INTR_MAP 将中断源 *SOURCE* 映射至 CPU 外部中断。中断源 *SOURCE* 见表 9-1。(R/W)

Register 9.57. INTMTX_CORE0_INT_STATUS_0_REG (0x0104)



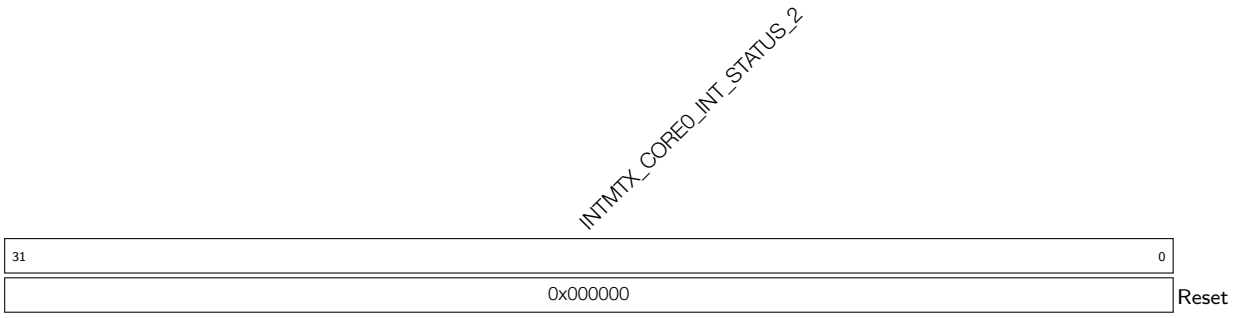
INTMTX_CORE0_INT_STATUS_0 中断源 0 ~ 31 的状态寄存器。(RO)

Register 9.58. INTMTX_CORE0_INT_STATUS_1_REG (0x0108)



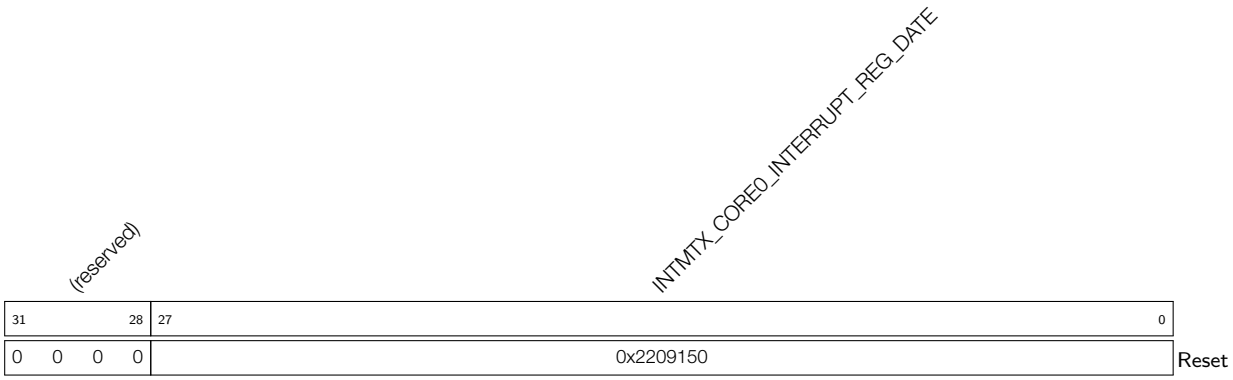
INTMTX_CORE0_INT_STATUS_1 中断源 32 ~ 63 的状态寄存器。(RO)

Register 9.59. INTMTX_CORE0_INT_STATUS_2_REG (0x010C)



INTMTX_CORE0_INT_STATUS_2 中断源 64 的状态寄存器。(RO)

Register 9.60. INTMTX_CORE0_INTERRUPT_REG_DATE_REG (0x07FC)

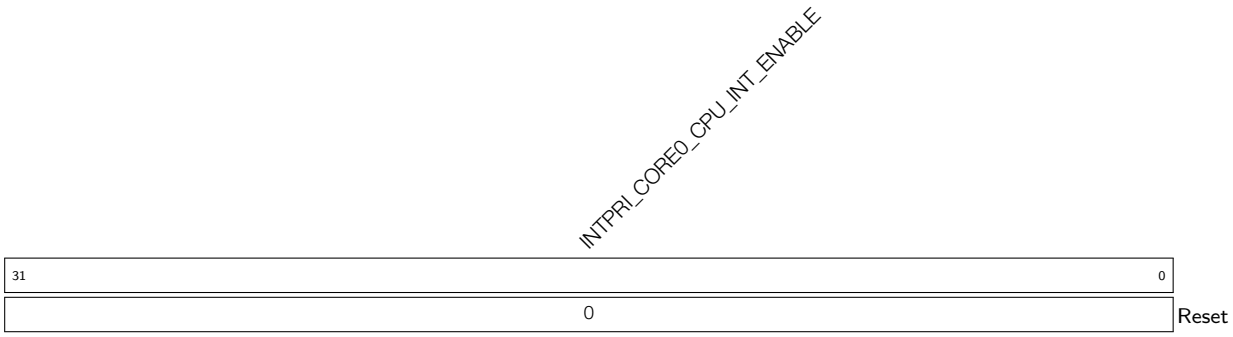


INTMTX_CORE0_INTERRUPT_REG_DATE 版本控制寄存器。(R/W)

9.7.2 中断优先级寄存器

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

Register 9.61. INTPRI_CORE0_CPU_INT_ENABLE_REG (0x0000)



INTPRI_CORE0_CPU_INT_ENABLE 配置是否使能对应 CPU 中断。

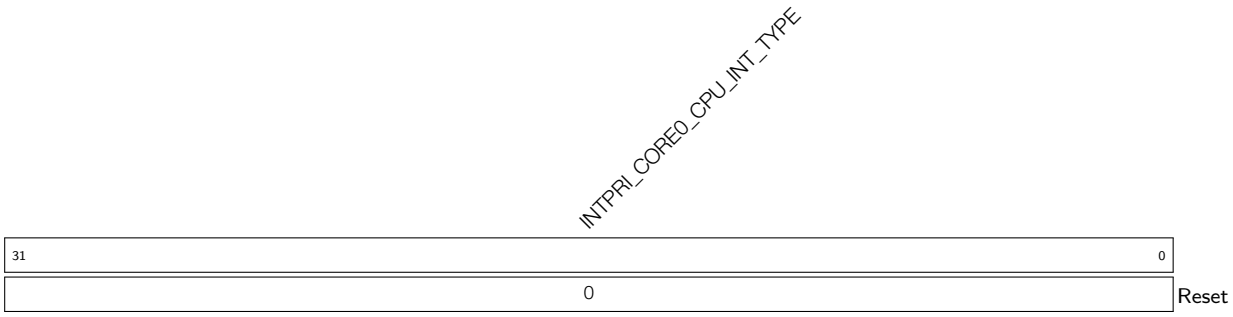
0：不使能

1：使能

更多信息，请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。

(R/W)

Register 9.62. INTPRI_CORE0_CPU_INT_TYPE_REG (0x0004)



INTPRI_CORE0_CPU_INT_TYPE 配置 CPU 中断类型。

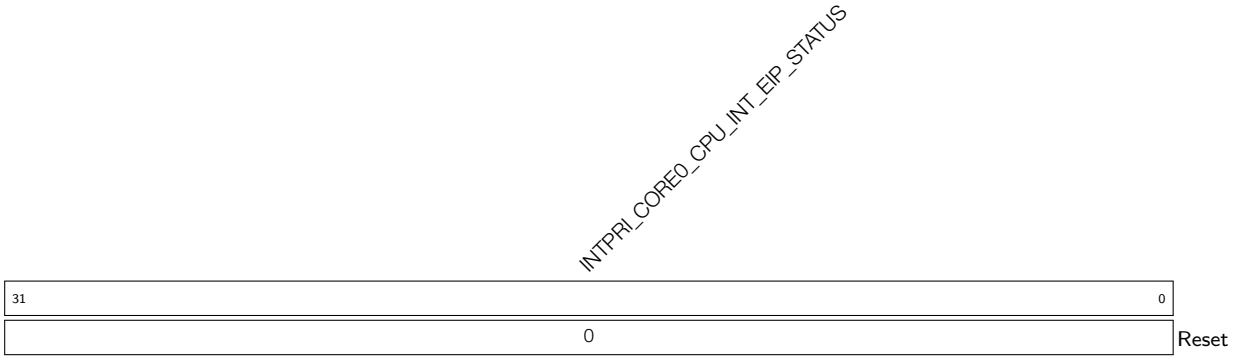
0：电平触发

1：边沿触发

更多信息，请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。

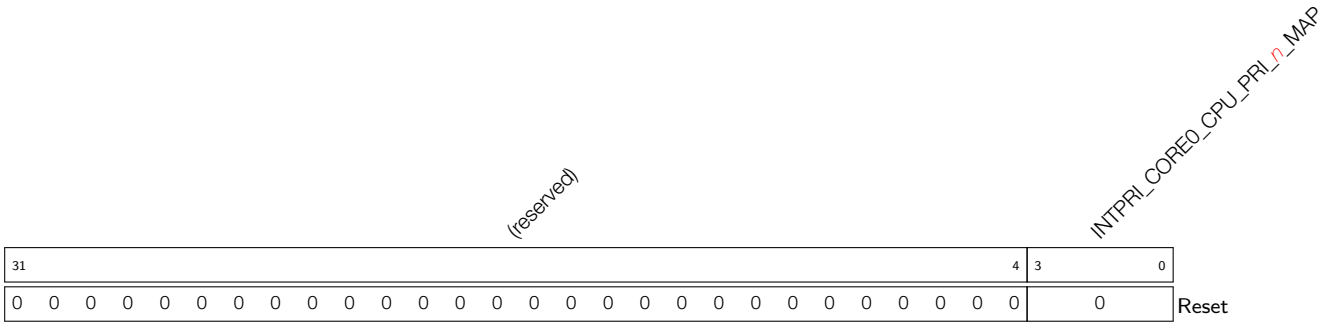
(R/W)

Register 9.63. INTPRI_CORE0_CPU_INT_EIP_STATUS_REG (0x0008)



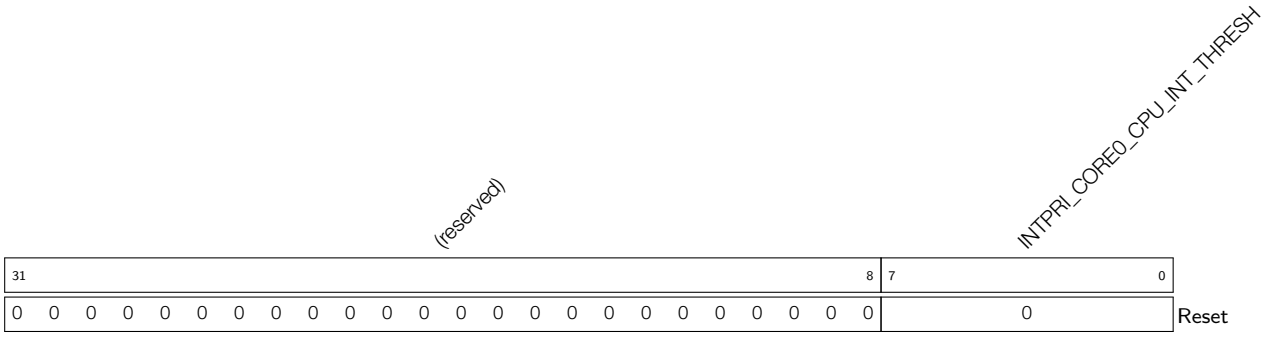
INTPRI_CORE0_CPU_INT_EIP_STATUS 表示 CPU 中断的阻塞状态。更多信息，请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。(RO)

Register 9.64. INTPRI_CORE0_CPU_INT_PRI_*n*_REG (*n*: 0-31)(0x000C+0x4**n*)



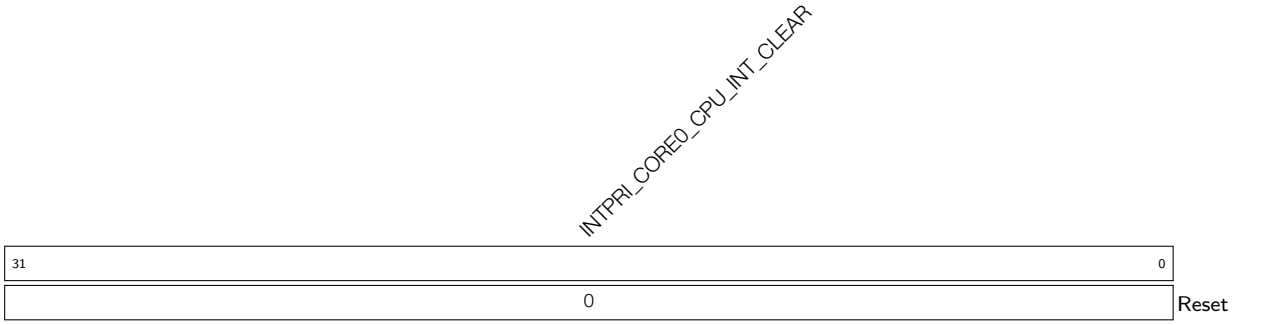
INTPRI_CORE0_CPU_PRI_*n*_MAP 配置 CPU 中断 *n* 的优先级。可配置为 1 ~ 15。数字越大，优先级越高。更多信息，请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。(R/W)

Register 9.65. INTPRI_CORE0_CPU_INT_THRESH_REG (0x008C)



INTPRI_CORE0_CPU_INT_THRESH 配置 CPU 中断阈值。仅当中断的优先级等于或高于该阈值，CPU 才会响应该中断。更多信息，请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。(R/W)

Register 9.66. INTPRI_CORE0_CPU_INT_CLEAR_REG (0x00A8)



INTPRI_CORE0_CPU_INT_CLEAR 配置是否清除对应的 CPU 中断。

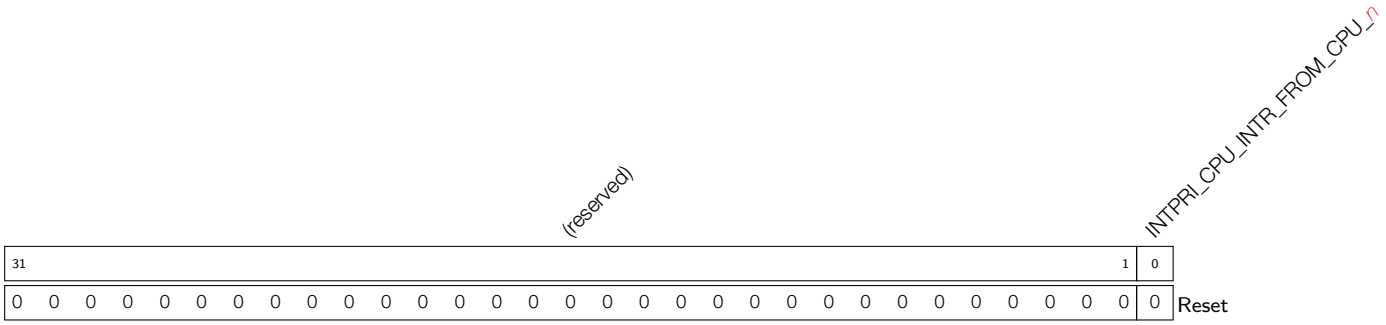
0：无效

1：清除

更多信息，请参考章节 1 *ESP-RISC-V CPU* > 1.6中断控制器。

(R/W)

Register 9.67. INTPRI_CPU_INTR_FROM_CPU_*n*_REG (*n*: 0-3)(0x0090+0x4n*)**



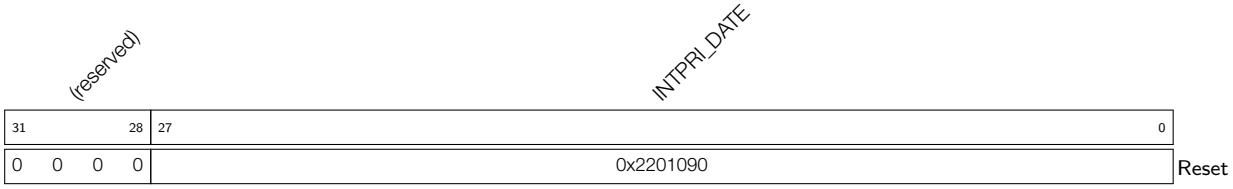
INTPRI_CPU_INTR_FROM_CPU_*n* 配置是否产生来自 CPU_INTR_FROM_CPU_*n* 的中断。

0：不产生来自 CPU_INTR_FROM_CPU_*n* 的中断

1：产生来自 CPU_INTR_FROM_CPU_*n* 的中断

(R/W)

Register 9.68. INTPRI_DATE_REG (0x00A0)



INTPRI_DATE 版本控制寄存器。(R/W)

10 事件任务矩阵 (SOC_ETM)

10.1 概述

事件任务矩阵 (ETM) 外设包含 50 个可配置通道。每个通道可以将任意指定外设的事件映射到任意指定外设的任务，从而触发外设执行指定任务，无需 CPU 干预。

10.2 特性

事件任务矩阵具有以下特性：

- 支持从多个外设接收 122 种事件
- 支持为多个外设生成 113 种任务
- 拥有 50 个可独立配置的 ETM 通道
- 每个通道接收到的事件可以是所有事件中的任意一个，每个通道接收到的事件可以映射到任意的任务上输出
- 每个 ETM 通道都可以独立使能。当通道未使能时，它不会响应所配置的事件，也不会生成要映射到的任务
- 能够产生事件、接收任务的外设有：GPIO、LED PWM、通用定时器、RTC 定时器、系统定时器、MCPWM、温度传感器、ADC、I2S、GDMA 和 PMU

由于 50 个 ETM 通道的功能和操作都是相同的。以下部分将 ETM 通道统称为通道*n*（其中 *n* 范围从 0 到 49）。

10.3 功能描述

10.3.1 架构

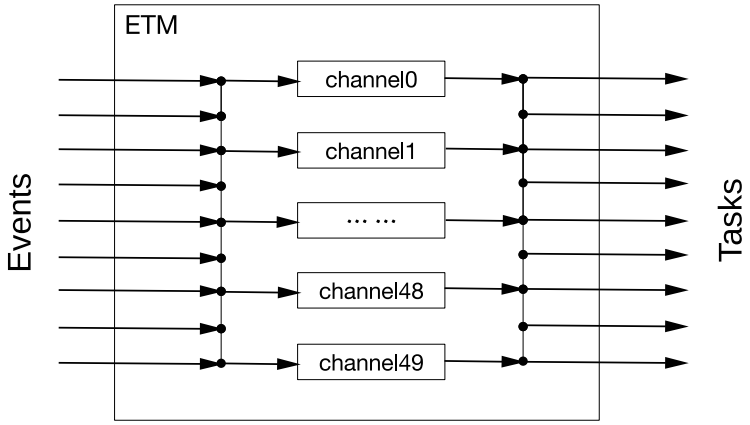


图 10-1. 事件任务矩阵架构

图 10-1 展示了事件任务矩阵的架构。

事件任务矩阵有 50 个独立通道，每个通道可以选择所有事件中的任意一个作为该通道的输入（有关配置方法，参考 10.3.2 章节），并且可以选择映射事件到所有任务中的任何一个（有关配置方法，参考 10.3.3 章节）。每个通道都有独立的使能配置位（有关配置方法，参考 10.3.5 章节）。

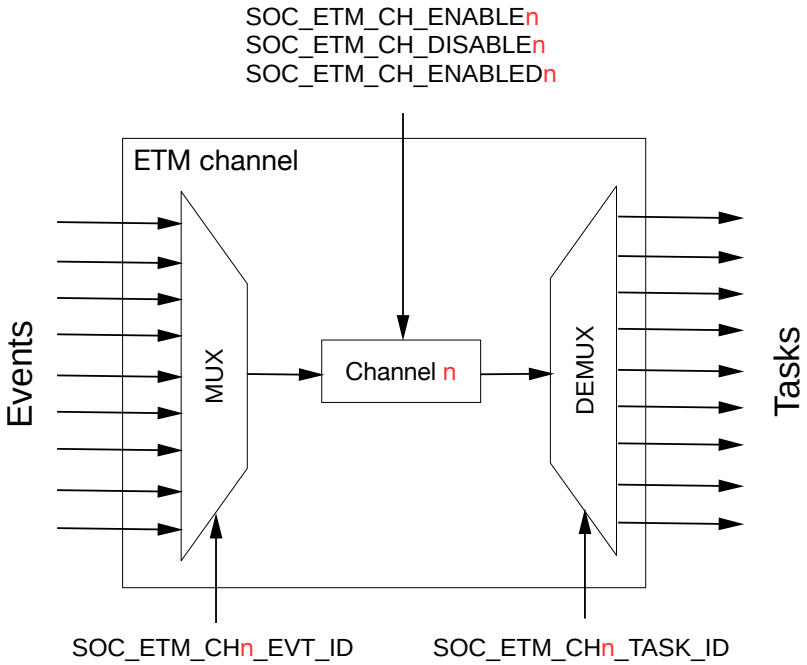


图 10-2. 事件任务矩阵通道*n*架构

图 10-2 展示了单个事件任务矩阵通道的结构，其中 SOC_ETM_CH*n*_EVT_ID 字段配置 MUX 选择所有事件之一作为通道*n* 的输入，SOC_ETM_CH*n*_TASK_ID 字段配置 DEMUX 以映射通道*n* 所选择的事件到所有任务之一。字段 SOC_ETM_CH_ENABLE*n* 和 SOC_ETM_CH_DISABLE*n* 用于使能或关闭通道*n*，字段 SOC_ETM_CH_ENABLED*n* 用于指示通道*n* 的开启或关闭状态。

10.3.2 事件

每个事件任务矩阵通道可以选择所有事件之一作为其接收的事件。每个通道都有字段 SOC_ETM_CH*n*_EVT_ID，用于指定选择哪个事件。字段 SOC_ETM_CH*n*_EVT_ID 的配置值与所选择的事件的关系如表 10-1 所示。

表 10-1. 事件任务矩阵通道*n*可选事件

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
1	GPIO_EVT_CH0_RISE_EDGE	GPIO
2	GPIO_EVT_CH1_RISE_EDGE	
3	GPIO_EVT_CH2_RISE_EDGE	
4	GPIO_EVT_CH3_RISE_EDGE	
5	GPIO_EVT_CH4_RISE_EDGE	
6	GPIO_EVT_CH5_RISE_EDGE	
7	GPIO_EVT_CH6_RISE_EDGE	
8	GPIO_EVT_CH7_RISE_EDGE	
9	GPIO_EVT_CH0_FALL_EDGE	
10	GPIO_EVT_CH1_FALL_EDGE	
11	GPIO_EVT_CH2_FALL_EDGE	
12	GPIO_EVT_CH3_FALL_EDGE	
13	GPIO_EVT_CH4_FALL_EDGE	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
14	GPIO_EVT_CH5_FALL_EDGE	
15	GPIO_EVT_CH6_FALL_EDGE	
16	GPIO_EVT_CH7_FALL_EDGE	
17	GPIO_EVT_CH0_ANY_EDGE	
18	GPIO_EVT_CH1_ANY_EDGE	
19	GPIO_EVT_CH2_ANY_EDGE	
20	GPIO_EVT_CH3_ANY_EDGE	
21	GPIO_EVT_CH4_ANY_EDGE	
22	GPIO_EVT_CH5_ANY_EDGE	
23	GPIO_EVT_CH6_ANY_EDGE	
24	GPIO_EVT_CH7_ANY_EDGE	
25	LEDC_EVT_DUTY_CHNG_END_CH0	LED PWM 控制器 (LEDC)
26	LEDC_EVT_DUTY_CHNG_END_CH1	
27	LEDC_EVT_DUTY_CHNG_END_CH2	
28	LEDC_EVT_DUTY_CHNG_END_CH3	
29	LEDC_EVT_DUTY_CHNG_END_CH4	
30	LEDC_EVT_DUTY_CHNG_END_CH5	
31	LEDC_EVT_OVF_CNT_PLS_CH0	
32	LEDC_EVT_OVF_CNT_PLS_CH1	
33	LEDC_EVT_OVF_CNT_PLS_CH2	
34	LEDC_EVT_OVF_CNT_PLS_CH3	
35	LEDC_EVT_OVF_CNT_PLS_CH4	
36	LEDC_EVT_OVF_CNT_PLS_CH5	
37	LEDC_EVT_TIME_OVF_TIMER0	
38	LEDC_EVT_TIME_OVF_TIMER1	
39	LEDC_EVT_TIME_OVF_TIMER2	
40	LEDC_EVT_TIME_OVF_TIMER3	
41	LEDC_EVT_TIMER0_CMP	
42	LEDC_EVT_TIMER1_CMP	
43	LEDC_EVT_TIMER2_CMP	
44	LEDC_EVT_TIMER3_CMP	
48	TIMER0_EVT_CNT_CMP_TIMER0	通用定时器 0
49	TIMER1_EVT_CNT_CMP_TIMER0	通用定时器 1
50	SYSTIMER_EVT_CNT_CMP0	系统定时器 (SYSTIMER)
51	SYSTIMER_EVT_CNT_CMP1	
52	SYSTIMER_EVT_CNT_CMP2	
58	MCPWM_EVT_TIMER0_STOP	电机控制脉宽调制器 (MCPWM)
59	MCPWM_EVT_TIMER1_STOP	
60	MCPWM_EVT_TIMER2_STOP	
61	MCPWM_EVT_TIMER0_TEZ	
62	MCPWM_EVT_TIMER1_TEZ	
63	MCPWM_EVT_TIMER2_TEZ	
64	MCPWM_EVT_TIMER0_TEP	
65	MCPWM_EVT_TIMER1_TEP	
66	MCPWM_EVT_TIMER2_TEP	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
67	MCPWM_EVT_OP0_TEA	
68	MCPWM_EVT_OP1_TEA	
69	MCPWM_EVT_OP2_TEA	
70	MCPWM_EVT_OP0_TEB	
71	MCPWM_EVT_OP1_TEB	
72	MCPWM_EVT_OP2_TEB	
73	MCPWM_EVT_F0	
74	MCPWM_EVT_F1	
75	MCPWM_EVT_F2	
76	MCPWM_EVT_F0_CLR	
77	MCPWM_EVT_F1_CLR	
78	MCPWM_EVT_F2_CLR	
79	MCPWM_EVT_TZ0_CBC	
80	MCPWM_EVT_TZ1_CBC	
81	MCPWM_EVT_TZ2_CBC	
82	MCPWM_EVT_TZ0_OST	
83	MCPWM_EVT_TZ1_OST	
84	MCPWM_EVT_TZ2_OST	
85	MCPWM_EVT_CAP0	
86	MCPWM_EVT_CAP1	
87	MCPWM_EVT_CAP2	
88	ADC_EVT_CONV_CMPLT0	ADC
89	ADC_EVT_EQ_ABOVE_THRESH0	
90	ADC_EVT_EQ_ABOVE_THRESH1	
91	ADC_EVT_EQ_BELOW_THRESH0	
92	ADC_EVT_EQ_BELOW_THRESH1	
94	ADC_EVT_STOPPED0	
95	ADC_EVT_STARTED0	
110	TMPSNSR_EVT_OVER_LIMIT	温度传感器
126	I2S_EVT_RX_DONE	I2S 控制器 (I2S)
127	I2S_EVT_TX_DONE	
128	I2S_EVT_X_WORDS_RECEIVED	
129	I2S_EVT_X_WORDS_SENT	
135	RTC_EVT_TICK	RTC 定时器
136	RTC_EVT_OVF	
137	RTC_EVT_CMP	
138	GDMA_EVT_IN_DONE_CH0	通用 DMA 控制器 (GDMA)
139	GDMA_EVT_IN_DONE_CH1	
140	GDMA_EVT_IN_DONE_CH2	
141	GDMA_EVT_IN_SUC_EOF_CH0	
142	GDMA_EVT_IN_SUC_EOF_CH1	
143	GDMA_EVT_IN_SUC_EOF_CH2	
144	GDMA_EVT_IN_FIFO_EMPTY_CH0	
145	GDMA_EVT_IN_FIFO_EMPTY_CH1	
146	GDMA_EVT_IN_FIFO_EMPTY_CH2	

SOC_ETM_CH*n*_EVT_ID	所选事件	产生该事件的外设
147	GDMA_EVT_IN_FIFO_FULL_CH0	
148	GDMA_EVT_IN_FIFO_FULL_CH1	
149	GDMA_EVT_IN_FIFO_FULL_CH2	
150	GDMA_EVT_OUT_DONE_CH0	
151	GDMA_EVT_OUT_DONE_CH1	
152	GDMA_EVT_OUT_DONE_CH2	
153	GDMA_EVT_OUT_EOF_CH0	
154	GDMA_EVT_OUT_EOF_CH1	
155	GDMA_EVT_OUT_EOF_CH2	
156	GDMA_EVT_OUT_TOTAL_EOF_CH0	
157	GDMA_EVT_OUT_TOTAL_EOF_CH1	
158	GDMA_EVT_OUT_TOTAL_EOF_CH2	
159	GDMA_EVT_OUT_FIFO_EMPTY_CH0	
160	GDMA_EVT_OUT_FIFO_EMPTY_CH1	
161	GDMA_EVT_OUT_FIFO_EMPTY_CH2	
162	GDMA_EVT_OUT_FIFO_FULL_CH0	
163	GDMA_EVT_OUT_FIFO_FULL_CH1	
164	GDMA_EVT_OUT_FIFO_FULL_CH2	
165	PMU_EVT_SLEEP_WEEKUP	PMU

其中，每个事件都是对应外设产生的一个脉冲信号。当事件脉冲信号有效时，视为接收到相应的事件。

对于事件更详细的描述，请参考产生该事件外设的章节。

10.3.3 任务

每个事件任务矩阵通道可以选择将本通道的事件映射到所有任务中的任何一个。每个通道都有字段 SOC_ETM_CH*n*_TASK_ID，用于选择要映射到的任务。字段 SOC_ETM_CH*n*_TASK_ID 的配置值与要映射到的任务的关系如表 10-2 所示。

表 10-2. 事件任务矩阵通道*n*可映射任务

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
1	GPIO_TASK_CH0_SET	GPIO
2	GPIO_TASK_CH1_SET	
3	GPIO_TASK_CH2_SET	
4	GPIO_TASK_CH3_SET	
5	GPIO_TASK_CH4_SET	
6	GPIO_TASK_CH5_SET	
7	GPIO_TASK_CH6_SET	
8	GPIO_TASK_CH7_SET	
9	GPIO_TASK_CH0_CLEAR	
10	GPIO_TASK_CH1_CLEAR	
11	GPIO_TASK_CH2_CLEAR	
12	GPIO_TASK_CH3_CLEAR	
13	GPIO_TASK_CH4_CLEAR	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
14	GPIO_TASK_CH5_CLEAR	
15	GPIO_TASK_CH6_CLEAR	
16	GPIO_TASK_CH7_CLEAR	
17	GPIO_TASK_CH0_TOGGLE	
18	GPIO_TASK_CH1_TOGGLE	
19	GPIO_TASK_CH2_TOGGLE	
20	GPIO_TASK_CH3_TOGGLE	
21	GPIO_TASK_CH4_TOGGLE	
22	GPIO_TASK_CH5_TOGGLE	
23	GPIO_TASK_CH6_TOGGLE	
24	GPIO_TASK_CH7_TOGGLE	
25	LEDC_TASK_TIMER0_RES_UPDATE	LED PWM 控制器 (LEDC)
26	LEDC_TASK_TIMER1_RES_UPDATE	
27	LEDC_TASK_TIMER2_RES_UPDATE	
28	LEDC_TASK_TIMER3_RES_UPDATE	
31	LEDC_TASK_DUTY_SCALE_UPDATE_CH0	
32	LEDC_TASK_DUTY_SCALE_UPDATE_CH1	
33	LEDC_TASK_DUTY_SCALE_UPDATE_CH2	
34	LEDC_TASK_DUTY_SCALE_UPDATE_CH3	
35	LEDC_TASK_DUTY_SCALE_UPDATE_CH4	
36	LEDC_TASK_DUTY_SCALE_UPDATE_CH5	
37	LEDC_TASK_TIMER0_CAP	
38	LEDC_TASK_TIMER1_CAP	
39	LEDC_TASK_TIMER2_CAP	
40	LEDC_TASK_TIMER3_CAP	
41	LEDC_TASK_SIG_OUT_DIS_CH0	
42	LEDC_TASK_SIG_OUT_DIS_CH1	
43	LEDC_TASK_SIG_OUT_DIS_CH2	
44	LEDC_TASK_SIG_OUT_DIS_CH3	
45	LEDC_TASK_SIG_OUT_DIS_CH4	
46	LEDC_TASK_SIG_OUT_DIS_CH5	
47	LEDC_TASK_OVF_CNT_RST_CH0	
48	LEDC_TASK_OVF_CNT_RST_CH1	
49	LEDC_TASK_OVF_CNT_RST_CH2	
50	LEDC_TASK_OVF_CNT_RST_CH3	
51	LEDC_TASK_OVF_CNT_RST_CH4	
52	LEDC_TASK_OVF_CNT_RST_CH5	
53	LEDC_TASK_TIMER0_RST	
54	LEDC_TASK_TIMER1_RST	
55	LEDC_TASK_TIMER2_RST	
56	LEDC_TASK_TIMER3_RST	
57	LEDC_TASK_TIMER0_RESUME	
58	LEDC_TASK_TIMER1_RESUME	
59	LEDC_TASK_TIMER2_RESUME	
60	LEDC_TASK_TIMER3_RESUME	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
61	LEDC_TASK_TIMER0_PAUSE	
62	LEDC_TASK_TIMER1_PAUSE	
63	LEDC_TASK_TIMER2_PAUSE	
64	LEDC_TASK_TIMER3_PAUSE	
65	LEDC_TASK_GAMMA_RESTART_CH0	
66	LEDC_TASK_GAMMA_RESTART_CH1	
67	LEDC_TASK_GAMMA_RESTART_CH2	
68	LEDC_TASK_GAMMA_RESTART_CH3	
69	LEDC_TASK_GAMMA_RESTART_CH4	
70	LEDC_TASK_GAMMA_RESTART_CH5	
71	LEDC_TASK_GAMMA_PAUSE_CH0	
72	LEDC_TASK_GAMMA_PAUSE_CH1	
73	LEDC_TASK_GAMMA_PAUSE_CH2	
74	LEDC_TASK_GAMMA_PAUSE_CH3	
75	LEDC_TASK_GAMMA_PAUSE_CH4	
76	LEDC_TASK_GAMMA_PAUSE_CH5	
77	LEDC_TASK_GAMMA_RESUME_CH0	
78	LEDC_TASK_GAMMA_RESUME_CH1	
79	LEDC_TASK_GAMMA_RESUME_CH2	
80	LEDC_TASK_GAMMA_RESUME_CH3	
81	LEDC_TASK_GAMMA_RESUME_CH4	
82	LEDC_TASK_GAMMA_RESUME_CH5	
88	TIMER0_TASK_CNT_START_TIMER0	通用定时器 0
90	TIMER0_TASK_ALARM_START_TIMER0	
92	TIMER0_TASK_CNT_STOP_TIMER0	
94	TIMER0_TASK_CNT_RELOAD_TIMER0	
96	TIMER0_TASK_CNT_CAP_TIMER0	
89	TIMER1_TASK_CNT_START_TIMER0	通用定时器 1
91	TIMER1_TASK_ALARM_START_TIMER0	
93	TIMER1_TASK_CNT_STOP_TIMER0	
95	TIMER1_TASK_CNT_RELOAD_TIMER0	
97	TIMER1_TASK_CNT_CAP_TIMER0	
102	MCPWM_TASK_CMPR0_A_UP	电机控制脉宽调制器 (MCPWM)
103	MCPWM_TASK_CMPR1_A_UP	
104	MCPWM_TASK_CMPR2_A_UP	
105	MCPWM_TASK_CMPR0_B_UP	
106	MCPWM_TASK_CMPR1_B_UP	
107	MCPWM_TASK_CMPR2_B_UP	
108	MCPWM_TASK_GEN_STOP	
109	MCPWM_TASK_TIMER0_SYN	
110	MCPWM_TASK_TIMER1_SYN	
111	MCPWM_TASK_TIMER2_SYN	
112	MCPWM_TASK_TIMER0_PERIOD_UP	
113	MCPWM_TASK_TIMER1_PERIOD_UP	
114	MCPWM_TASK_TIMER2_PERIOD_UP	

SOC_ETM_CH*n*_TASK_ID	所映射任务	接收该任务的外设
115	MCPWM_TASK_TZ0_OST	
116	MCPWM_TASK_TZ1_OST	
117	MCPWM_TASK_TZ2_OST	
118	MCPWM_TASK_CLR0_OST	
119	MCPWM_TASK_CLR1_OST	
120	MCPWM_TASK_CLR2_OST	
121	MCPWM_TASK_CAP0	
122	MCPWM_TASK_CAP1	
123	MCPWM_TASK_CAP2	
124	ADC_TASK_SAMPLE0	ADC
125	ADC_TASK_SAMPLE1	
126	ADC_TASK_START0	
127	ADC_TASK_STOP0	
135	TMPSNSR_TASK_START_SAMPLE	温度传感器
136	TMPSNSR_TASK_STOP_SAMPLE	
148	I2S_TASK_START_RX	I2S 控制器 (I2S)
149	I2S_TASK_START_TX	
150	I2S_TASK_STOP_RX	
151	I2S_TASK_STOP_TX	
159	GDMA_TASK_IN_START_CH0	通用 DMA 控制器 (GDMA)
160	GDMA_TASK_IN_START_CH1	
161	GDMA_TASK_IN_START_CH2	
162	GDMA_TASK_OUT_START_CH0	
163	GDMA_TASK_OUT_START_CH1	
164	GDMA_TASK_OUT_START_CH2	
165	PMU_TASK_SLEEP_REQ	PMU

一个通道接收到一个有效事件脉冲信号时，会产生映射任务脉冲信号。

对于任务更详细的描述，请参考接收该任务外设的章节。

来自多个不同通道的事件可以选择映射到同一个任务（例如，可以将多个通道的字段 SOC_ETM_CH*n*_TASK_ID 都配置为相同的值，而字段 SOC_ETM_CH*n*_EVT_ID 可以配置为相同或不同的值）。在这种情况下，当任意一个通道接收到的事件有效时，指定的任务就会被生成。当多个通道接收到的事件同时有效时，指定的任务只会产生一次。

10.3.4 时序考虑因素

ETM 外设接收的事件、发送的任务和 ETM 通道之间驱动时钟的结构如图 10-3 所示。

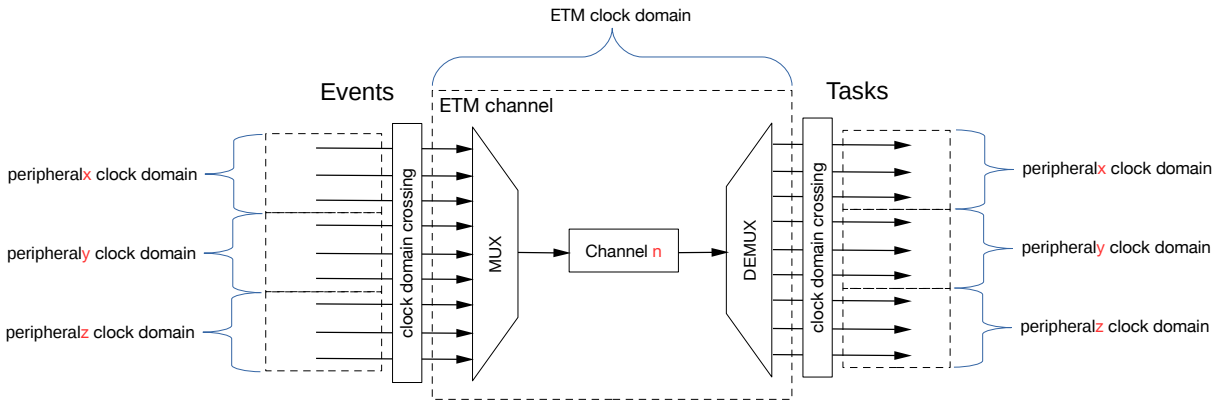


图 10-3. 事件任务矩阵时钟结构

ETM 运行在 AHB_CLK 时钟域下（详见章节 7 复位和时钟），每个事件对应一个由相应外设在其时钟域中产生的脉冲信号，每个任务被 ETM 映射为对应外设时钟域下的脉冲信号。由于产生事件的外设、事件任务矩阵和接收任务的外设不一定由同个时钟驱动，需要同步处理，所以对两个连续事件之间的时间间隔有要求，否则会丢失事件。为了让事件任务矩阵成功接收到每个事件，对于每个可以产生事件的外设，其产生的两个连续事件脉冲之间的时间间隔必须大于一个事件任务矩阵的时钟周期，即 $ceil(\frac{peripheral\_clock\_frequenty}{ETM\_clock\_frequenty})$ 个外设时钟周期。

例如，假设外设 A 在 96 MHz 时钟域（PLL_F96M_CLK）生成事件 1，ETM 在 32 MHz 时钟域（AHB_CLK）运行。要成功接收每个事件 1，两个连续事件 1 之间的时间间隔必须大于三个外设 A 的时钟周期（即一个 ETM 时钟周期）。

同样的，为了使事件任务矩阵成功地将接收到的事件（即同步到事件任务矩阵时钟域的事件）映射为外设的任务，事件任务矩阵时钟域中的两个连续事件脉冲之间的时间间隔必须大于一个外设的时钟周期，即 $ceil(\frac{ETM\_clock\_frequenty}{peripheral\_clock\_frequenty})$ 个事件任务矩阵时钟周期。

例如，假设外设 B 在 8 MHz 时钟域（RC_FAST_CLK）接收任务 1，ETM 在 32 MHz 时钟域（AHB_CLK）运行。要成功将每个接收到的事件映射到任务 1，两个连续事件之间的时间间隔必须大于四个 ETM 时钟周期（即一个外设 B 时钟周期）。

因此，要将外设 A 产生的两个连续事件完整映射到外设 B 的两个任务中，这两个事件之间的时间间隔必须大于 $ceil(\frac{peripheral\_A\_clock\_frequenty}{ETM\_clock\_frequenty}) * ceil(\frac{ETM\_clock\_frequenty}{peripheral\_B\_clock\_frequenty})$ 个外设 A 时钟周期。

例如，假设外设 A 在 96 MHz 时钟域（PLL_F96M_CLK）生成事件 1，外设 B 在 8 MHz 时钟域（RC_FAST_CLK）接收任务 1，ETM 在 32 MHz 时钟域（AHB_CLK）运行。要成功将每个事件 1（外设 A 生成）映射到任务 1（外设 B 接收），两个连续事件 1 之间的时间间隔必须大于 $3 * 4 = 12$ 个外设 A 时钟周期。

10.3.5 通道控制

事件任务矩阵的每个通道都可以独立配置为使能或关闭。当通道*n* 配置为使能时，如果该通道接收到所配置的事件（字段 SOC_ETM_CH*n*_EVT_ID），该事件将映射到所配置的任务上（字段 SOC_ETM_CH*n*_TASK_ID）。当通道*n* 配置为不使能时，即使该通道接收到所配置的事件（字段SOC_ETM_CH*n*_EVT_ID），也不会产生任何任务。

使能事件任务矩阵通道*n* 的配置过程如下：

1. 向字段 SOC_ETM_CH_ENABLE*n* 写 1

2. 读取字段 SOC_ETM_CH_ENABLED*n* 的值，如果该字段的值为 1，则表示通道*n* 已使能，否则通道*n* 未使能

关闭事件任务矩阵通道*n* 的配置过程如下：

1. 向字段 SOC_ETM_CH_DISABLE*n* 写 1
2. 读取字段 SOC_ETM_CH_ENABLED*n* 的值，如果该字段的值为 0，则表示通道*n* 已经关闭，否则通道*n* 使能

（如果将字段 SOC_ETM_CH*n*_EVT_ID 的值配置为 0 或将字段 SOC_ETM_CH*n*_TASK_ID 的值配置为 0，事件任务矩阵通道*n* 也将被关闭。）

事件任务矩阵通道*n* 的完整配置过程如下：

1. 向字段 PCR_ETM_CLK_EN 写 1，开启 ETM 的时钟
2. 配置字段 SOC_ETM_CH*n*_EVT_ID 的值，选择通道*n* 所要接收的事件
3. 配置字段 SOC_ETM_CH*n*_TASK_ID 的值，选择通道*n* 所接收到的事件要映射到的任务
4. 向字段 SOC_ETM_CH_ENABLE*n* 写 1
5. 当通道*n* 不再需要将配置的事件映射到配置的任务时，向字段SOC_ETM_CH_DISABLE*n*写 1 来关闭通道*n*。之后，如果要为通道*n* 配置新的事件和任务的映射关系，则重复步骤 2-4，如果不做任何配置，则通道*n* 保持停止运行状态
6. 若需复位 ETM，可通过向 PCR_ETM_RST_EN 字段先写 0 再写 1 来实现。PCR_ETM_READY 变为 1 时复位完成

10.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
SOC_ETM_CH_ENA_AD0_REG	通道状态寄存器	0x0000	R/WTC/SS
SOC_ETM_CH_ENA_AD0_SET_REG	通道使能寄存器	0x0004	WT
SOC_ETM_CH_ENA_AD0_CLR_REG	通道关闭寄存器	0x0008	WT
SOC_ETM_CH_ENA_AD1_REG	通道状态寄存器	0x000C	R/WTC/SS
SOC_ETM_CH_ENA_AD1_SET_REG	通道使能寄存器	0x0010	WT
SOC_ETM_CH_ENA_AD1_CLR_REG	通道关闭寄存器	0x0014	WT
SOC_ETM_CH0_EVT_ID_REG	通道 0 事件 ID 寄存器	0x0018	R/W
SOC_ETM_CH0_TASK_ID_REG	通道 0 任务 ID 寄存器	0x001C	R/W
SOC_ETM_CH1_EVT_ID_REG	通道 1 事件 ID 寄存器	0x0020	R/W
SOC_ETM_CH1_TASK_ID_REG	通道 1 任务 ID 寄存器	0x0024	R/W
SOC_ETM_CH2_EVT_ID_REG	通道 2 事件 ID 寄存器	0x0028	R/W
SOC_ETM_CH2_TASK_ID_REG	通道 2 任务 ID 寄存器	0x002C	R/W
SOC_ETM_CH3_EVT_ID_REG	通道 3 事件 ID 寄存器	0x0030	R/W
SOC_ETM_CH3_TASK_ID_REG	通道 3 任务 ID 寄存器	0x0034	R/W
SOC_ETM_CH4_EVT_ID_REG	通道 4 事件 ID 寄存器	0x0038	R/W
SOC_ETM_CH4_TASK_ID_REG	通道 4 任务 ID 寄存器	0x003C	R/W
SOC_ETM_CH5_EVT_ID_REG	通道 5 事件 ID 寄存器	0x0040	R/W
SOC_ETM_CH5_TASK_ID_REG	通道 5 任务 ID 寄存器	0x0044	R/W
SOC_ETM_CH6_EVT_ID_REG	通道 6 事件 ID 寄存器	0x0048	R/W
SOC_ETM_CH6_TASK_ID_REG	通道 6 任务 ID 寄存器	0x004C	R/W
SOC_ETM_CH7_EVT_ID_REG	通道 7 事件 ID 寄存器	0x0050	R/W
SOC_ETM_CH7_TASK_ID_REG	通道 7 任务 ID 寄存器	0x0054	R/W
SOC_ETM_CH8_EVT_ID_REG	通道 8 事件 ID 寄存器	0x0058	R/W
SOC_ETM_CH8_TASK_ID_REG	通道 8 任务 ID 寄存器	0x005C	R/W
SOC_ETM_CH9_EVT_ID_REG	通道 9 事件 ID 寄存器	0x0060	R/W
SOC_ETM_CH9_TASK_ID_REG	通道 9 任务 ID 寄存器	0x0064	R/W
SOC_ETM_CH10_EVT_ID_REG	通道 10 事件 ID 寄存器	0x0068	R/W
SOC_ETM_CH10_TASK_ID_REG	通道 10 任务 ID 寄存器	0x006C	R/W
SOC_ETM_CH11_EVT_ID_REG	通道 11 事件 ID 寄存器	0x0070	R/W
SOC_ETM_CH11_TASK_ID_REG	通道 11 任务 ID 寄存器	0x0074	R/W
SOC_ETM_CH12_EVT_ID_REG	通道 12 事件 ID 寄存器	0x0078	R/W
SOC_ETM_CH12_TASK_ID_REG	通道 12 任务 ID 寄存器	0x007C	R/W
SOC_ETM_CH13_EVT_ID_REG	通道 13 事件 ID 寄存器	0x0080	R/W
SOC_ETM_CH13_TASK_ID_REG	通道 13 任务 ID 寄存器	0x0084	R/W
SOC_ETM_CH14_EVT_ID_REG	通道 14 事件 ID 寄存器	0x0088	R/W
SOC_ETM_CH14_TASK_ID_REG	通道 14 任务 ID 寄存器	0x008C	R/W
SOC_ETM_CH15_EVT_ID_REG	通道 15 事件 ID 寄存器	0x0090	R/W

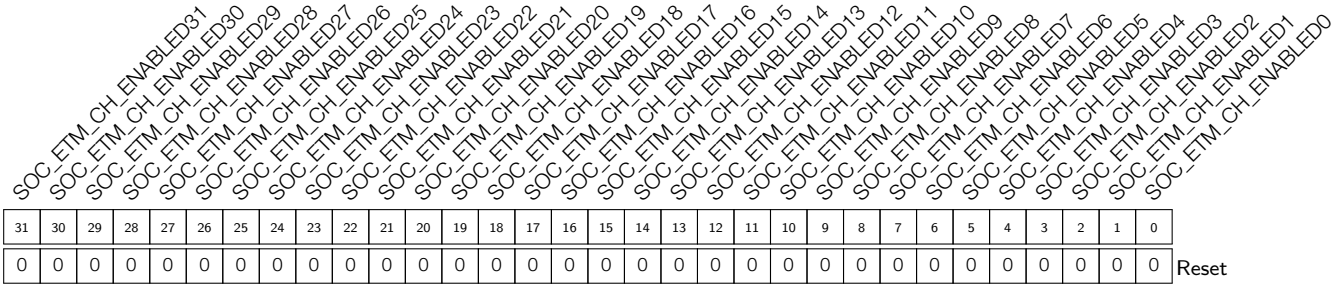
名称	描述	地址	访问
SOC_ETM_CH15_TASK_ID_REG	通道 15 任务 ID 寄存器	0x0094	R/W
SOC_ETM_CH16_EVT_ID_REG	通道 16 事件 ID 寄存器	0x0098	R/W
SOC_ETM_CH16_TASK_ID_REG	通道 16 任务 ID 寄存器	0x009C	R/W
SOC_ETM_CH17_EVT_ID_REG	通道 17 事件 ID 寄存器	0x00A0	R/W
SOC_ETM_CH17_TASK_ID_REG	通道 17 任务 ID 寄存器	0x00A4	R/W
SOC_ETM_CH18_EVT_ID_REG	通道 18 事件 ID 寄存器	0x00A8	R/W
SOC_ETM_CH18_TASK_ID_REG	通道 18 任务 ID 寄存器	0x00AC	R/W
SOC_ETM_CH19_EVT_ID_REG	通道 19 事件 ID 寄存器	0x00B0	R/W
SOC_ETM_CH19_TASK_ID_REG	通道 19 任务 ID 寄存器	0x00B4	R/W
SOC_ETM_CH20_EVT_ID_REG	通道 20 事件 ID 寄存器	0x00B8	R/W
SOC_ETM_CH20_TASK_ID_REG	通道 20 任务 ID 寄存器	0x00BC	R/W
SOC_ETM_CH21_EVT_ID_REG	通道 21 事件 ID 寄存器	0x00C0	R/W
SOC_ETM_CH21_TASK_ID_REG	通道 21 任务 ID 寄存器	0x00C4	R/W
SOC_ETM_CH22_EVT_ID_REG	通道 22 事件 ID 寄存器	0x00C8	R/W
SOC_ETM_CH22_TASK_ID_REG	通道 22 任务 ID 寄存器	0x00CC	R/W
SOC_ETM_CH23_EVT_ID_REG	通道 23 事件 ID 寄存器	0x00D0	R/W
SOC_ETM_CH23_TASK_ID_REG	通道 23 任务 ID 寄存器	0x00D4	R/W
SOC_ETM_CH24_EVT_ID_REG	通道 24 事件 ID 寄存器	0x00D8	R/W
SOC_ETM_CH24_TASK_ID_REG	通道 24 任务 ID 寄存器	0x00DC	R/W
SOC_ETM_CH25_EVT_ID_REG	通道 25 事件 ID 寄存器	0x00E0	R/W
SOC_ETM_CH25_TASK_ID_REG	通道 25 任务 ID 寄存器	0x00E4	R/W
SOC_ETM_CH26_EVT_ID_REG	通道 26 事件 ID 寄存器	0x00E8	R/W
SOC_ETM_CH26_TASK_ID_REG	通道 26 任务 ID 寄存器	0x00EC	R/W
SOC_ETM_CH27_EVT_ID_REG	通道 27 事件 ID 寄存器	0x00F0	R/W
SOC_ETM_CH27_TASK_ID_REG	通道 27 任务 ID 寄存器	0x00F4	R/W
SOC_ETM_CH28_EVT_ID_REG	通道 28 事件 ID 寄存器	0x00F8	R/W
SOC_ETM_CH28_TASK_ID_REG	通道 28 任务 ID 寄存器	0x00FC	R/W
SOC_ETM_CH29_EVT_ID_REG	通道 29 事件 ID 寄存器	0x0100	R/W
SOC_ETM_CH29_TASK_ID_REG	通道 29 任务 ID 寄存器	0x0104	R/W
SOC_ETM_CH30_EVT_ID_REG	通道 30 事件 ID 寄存器	0x0108	R/W
SOC_ETM_CH30_TASK_ID_REG	通道 30 任务 ID 寄存器	0x010C	R/W
SOC_ETM_CH31_EVT_ID_REG	通道 31 事件 ID 寄存器	0x0110	R/W
SOC_ETM_CH31_TASK_ID_REG	通道 31 任务 ID 寄存器	0x0114	R/W
SOC_ETM_CH32_EVT_ID_REG	通道 32 事件 ID 寄存器	0x0118	R/W
SOC_ETM_CH32_TASK_ID_REG	通道 32 任务 ID 寄存器	0x011C	R/W
SOC_ETM_CH33_EVT_ID_REG	通道 33 事件 ID 寄存器	0x0120	R/W
SOC_ETM_CH33_TASK_ID_REG	通道 33 任务 ID 寄存器	0x0124	R/W
SOC_ETM_CH34_EVT_ID_REG	通道 34 事件 ID 寄存器	0x0128	R/W
SOC_ETM_CH34_TASK_ID_REG	通道 34 任务 ID 寄存器	0x012C	R/W
SOC_ETM_CH35_EVT_ID_REG	通道 35 事件 ID 寄存器	0x0130	R/W
SOC_ETM_CH35_TASK_ID_REG	通道 35 任务 ID 寄存器	0x0134	R/W
SOC_ETM_CH36_EVT_ID_REG	通道 36 事件 ID 寄存器	0x0138	R/W
SOC_ETM_CH36_TASK_ID_REG	通道 36 任务 ID 寄存器	0x013C	R/W

名称	描述	地址	访问
SOC_ETM_CH37_EVT_ID_REG	通道 37 事件 ID 寄存器	0x0140	R/W
SOC_ETM_CH37_TASK_ID_REG	通道 37 任务 ID 寄存器	0x0144	R/W
SOC_ETM_CH38_EVT_ID_REG	通道 38 事件 ID 寄存器	0x0148	R/W
SOC_ETM_CH38_TASK_ID_REG	通道 38 任务 ID 寄存器	0x014C	R/W
SOC_ETM_CH39_EVT_ID_REG	通道 39 事件 ID 寄存器	0x0150	R/W
SOC_ETM_CH39_TASK_ID_REG	通道 39 任务 ID 寄存器	0x0154	R/W
SOC_ETM_CH40_EVT_ID_REG	通道 40 事件 ID 寄存器	0x0158	R/W
SOC_ETM_CH40_TASK_ID_REG	通道 40 任务 ID 寄存器	0x015C	R/W
SOC_ETM_CH41_EVT_ID_REG	通道 41 事件 ID 寄存器	0x0160	R/W
SOC_ETM_CH41_TASK_ID_REG	通道 41 任务 ID 寄存器	0x0164	R/W
SOC_ETM_CH42_EVT_ID_REG	通道 42 事件 ID 寄存器	0x0168	R/W
SOC_ETM_CH42_TASK_ID_REG	通道 42 任务 ID 寄存器	0x016C	R/W
SOC_ETM_CH43_EVT_ID_REG	通道 43 事件 ID 寄存器	0x0170	R/W
SOC_ETM_CH43_TASK_ID_REG	通道 43 任务 ID 寄存器	0x0174	R/W
SOC_ETM_CH44_EVT_ID_REG	通道 44 事件 ID 寄存器	0x0178	R/W
SOC_ETM_CH44_TASK_ID_REG	通道 44 任务 ID 寄存器	0x017C	R/W
SOC_ETM_CH45_EVT_ID_REG	通道 45 事件 ID 寄存器	0x0180	R/W
SOC_ETM_CH45_TASK_ID_REG	通道 45 任务 ID 寄存器	0x0184	R/W
SOC_ETM_CH46_EVT_ID_REG	通道 46 事件 ID 寄存器	0x0188	R/W
SOC_ETM_CH46_TASK_ID_REG	通道 46 任务 ID 寄存器	0x018C	R/W
SOC_ETM_CH47_EVT_ID_REG	通道 47 事件 ID 寄存器	0x0190	R/W
SOC_ETM_CH47_TASK_ID_REG	通道 47 任务 ID 寄存器	0x0194	R/W
SOC_ETM_CH48_EVT_ID_REG	通道 48 事件 ID 寄存器	0x0198	R/W
SOC_ETM_CH48_TASK_ID_REG	通道 48 任务 ID 寄存器	0x019C	R/W
SOC_ETM_CH49_EVT_ID_REG	通道 49 事件 ID 寄存器	0x01A0	R/W
SOC_ETM_CH49_TASK_ID_REG	通道 49 任务 ID 寄存器	0x01A4	R/W
SOC_ETM_CLK_EN_REG	ETM 时钟使能寄存器	0x01A8	R/W
版本寄存器			
SOC_ETM_DATE_REG	版本控制寄存器	0x01AC	R/W

10.5 寄存器

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

Register 10.1. SOC_ETM_CH_ENA_AD0_REG (0x0000)



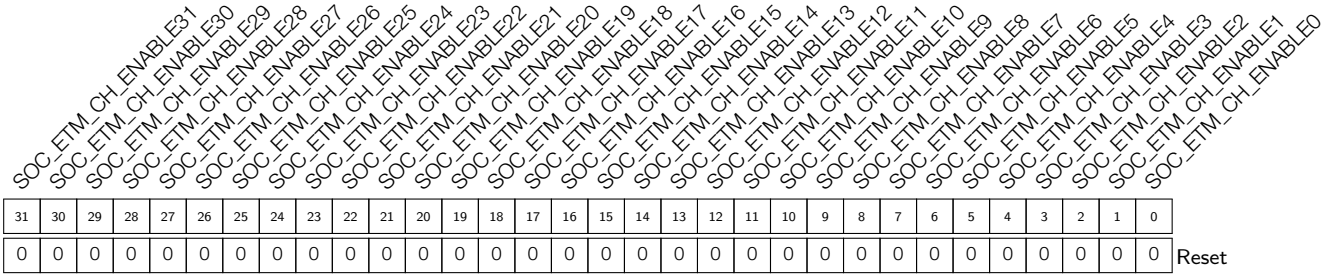
SOC_ETM_CH_ENABLED*n* (*n*: 0-31) 表示通道*n*的状态。

0：已关闭

1：已使能

(R/WTC/SS)

Register 10.2. SOC_ETM_CH_ENA_AD0_SET_REG (0x0004)



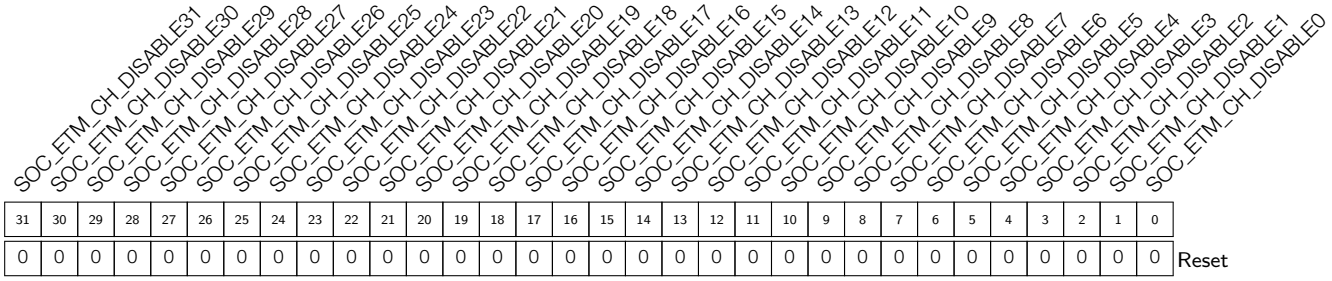
SOC_ETM_CH_ENABLE*n* (*n*: 0-31) 配置使能通道*n*。

0：无效值，没有作用

1：使能

(WT)

Register 10.3. SOC_ETM_CH_ENA_AD0_CLR_REG (0x0008)



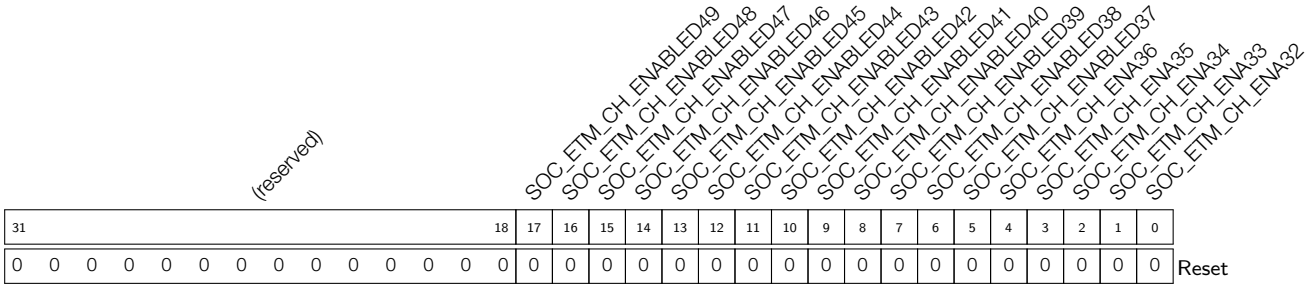
SOC_ETM_CH_DISABLE*n* (*n*: 0-31) 配置关闭通道*n*。

0：无效值，没有作用

1：关闭

(WT)

Register 10.4. SOC_ETM_CH_ENA_AD1_REG (0x000C)



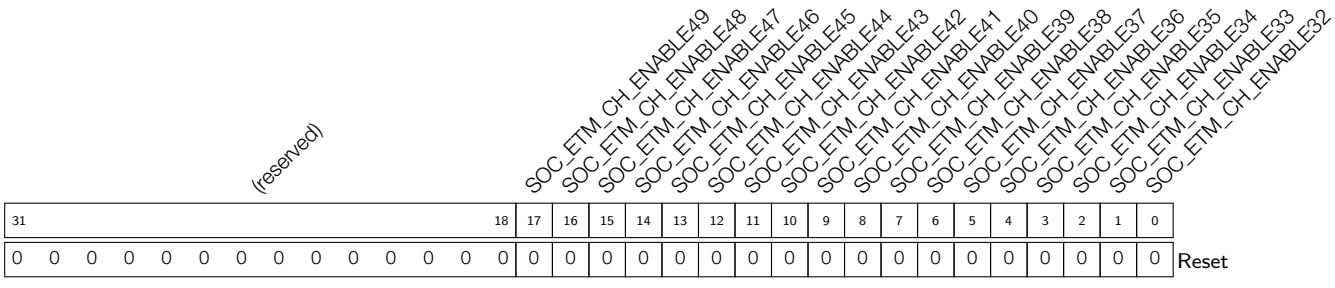
SOC_ETM_CH_ENABLED*n* (*n*: 32-49) 表示通道*n*的状态。

0：已关闭

1：已使能

(R/WTC/SS)

Register 10.5. SOC_ETM_CH_ENA_AD1_SET_REG (0x0010)



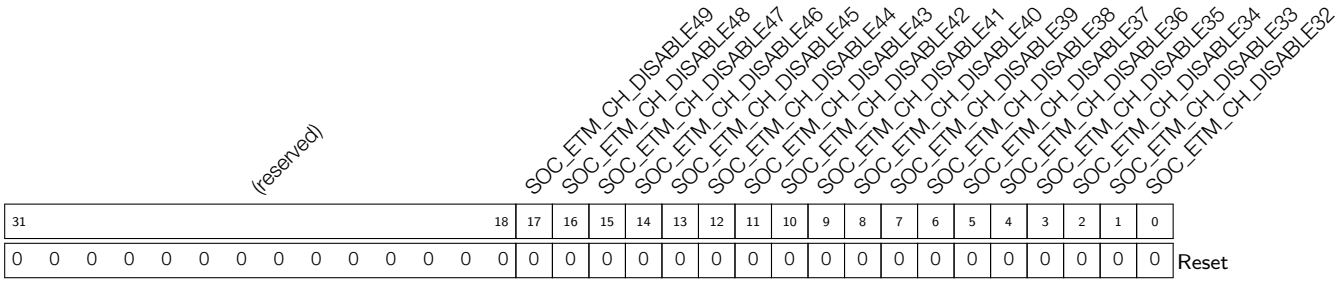
SOC_ETM_CH_ENABLE*n* (*n*: 32-49) 配置使能通道*n*。

0：无效值，没有作用

1：使能

(WT)

Register 10.6. SOC_ETM_CH_ENA_AD1_CLR_REG (0x0014)



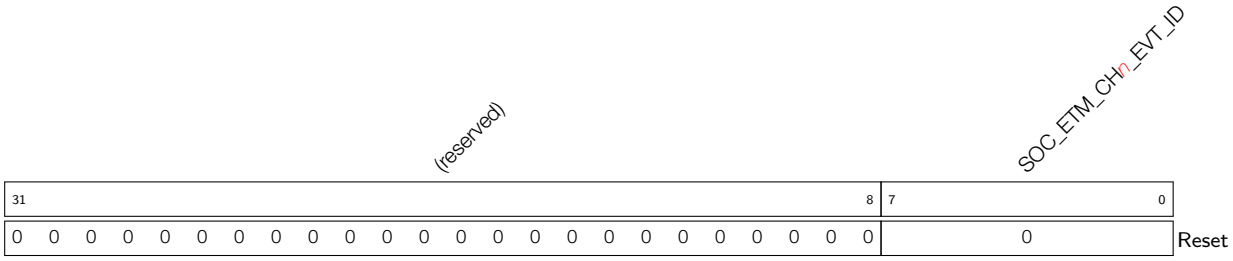
SOC_ETM_CH_DISABLE*n* (*n*: 32-49) 配置关闭通道*n*。

0：无效值，没有作用

1：关闭

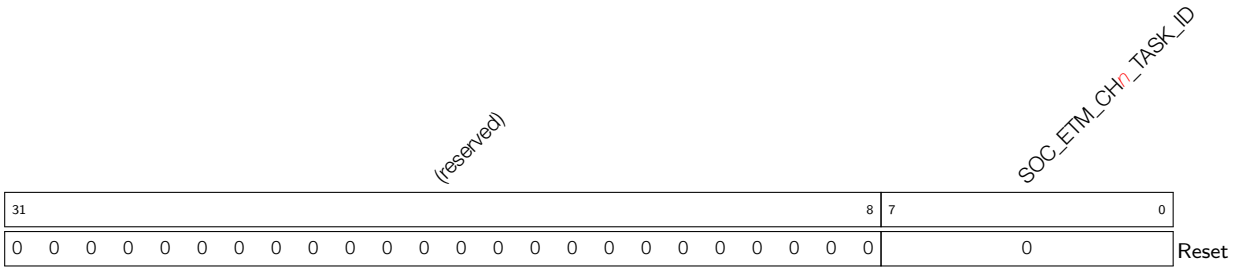
(WT)

Register 10.7. SOC_ETM_CH*n*_EVT_ID_REG (*n*: 0-49) (0x0018+0x8*n)



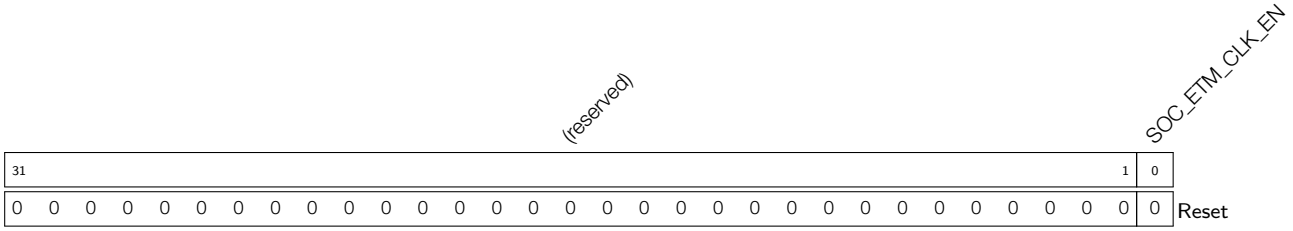
SOC_ETM_CH*n*_EVT_ID (*n*: 0-49) 配置通道*n*事件 ID，详见表 10-1。(R/W)

Register 10.8. SOC_ETM_CH*n*_TASK_ID_REG (*n*: 0-49) (0x001C+0x8*n)



SOC_ETM_CH*n*_TASK_ID (*n*: 0-49) 配置通道*n*任务 ID，详见表 10-2。(R/W)

Register 10.9. SOC_ETM_CLK_EN_REG (0x01A8)



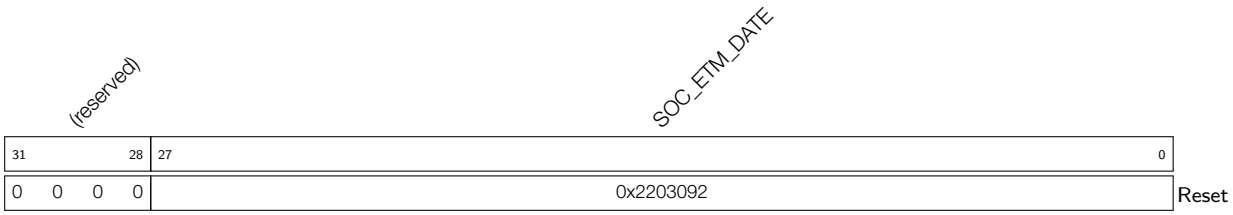
SOC_ETM_CLK_EN 配置寄存器时钟门控。

0：仅在应用写寄存器时开启时钟

1：强制为寄存器开启时钟门控

(R/W)

Register 10.10. SOC_ETM_DATE_REG (0x01AC)



SOC_ETM_DATE 版本控制寄存器。(R/W)

11 系统定时器 (SYSTIMER)

11.1 概述

ESP32-H2 芯片内置 52 位系统定时器。该定时器可用于生成操作系统所需的滴答定时中断，也可以用作普通的定时器生成周期或单次延时中断。在 RTC 定时器的协助下，系统定时器可在芯片从 Deep-sleep 或 Light-sleep 唤醒后补偿睡眠时间。

系统定时器内置两个计数器（UNIT0 和 UNIT1）以及三个比较器（COMP0、COMP1 和 COMP2）。比较器用于监控计数器的计数值是否达到报警值。定时器的功能块图见图 11-1。

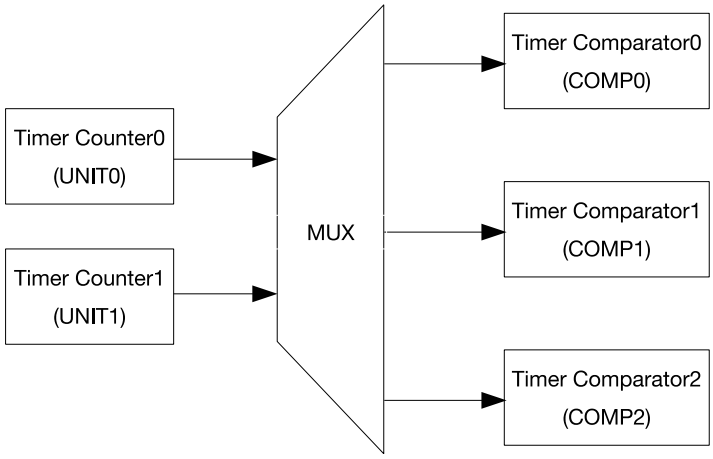


图 11-1. 系统定时器结构图

11.2 主要特性

系统定时器具有如下特性：

- 集成两个 52 位计数器和三个 52 位比较器
- 支持软件访问由 APB_CLK 时钟驱动的寄存器
- 支持 CNT_CLK 时钟计数，两次计数周期的平均频率为 16 MHz
- 配置 XTAL_CLK (32 MHz) 或者 RC_FAST_CLK 作为 CNT_CLK 时钟源
- 支持 52 位报警值 (t) 和 26 位报警周期 (δt)
- 支持两种报警模式：
  - 单次报警模式：根据设定的目标报警值 (t)，生成一次性报警
  - 周期报警模式：根据设定的报警周期 (δt)，生成周期性报警
- 支持根据设置的报警值 (t) 或报警周期 (δt)，通过三个比较器生成三个独立中断
- 支持在从 Deep-sleep 或 Light-sleep 唤醒之后，系统定时器通过软件加载 RTC 定时器记录的睡眠时间，并进行补偿
- CPU 处于停止状态或处于在线调试状态时，系统定时器可选择停止运行或继续运行

- 支持事件任务矩阵 (ETM) 事件报警

11.3 时钟源选择

计数器和比较器使用 XTAL_CLK 或者 RC_FAST_CLK 作为时钟源。可通过配置寄存器 PCR_SYSTIMER_FUNC_CLK_CONF_REG 的 PCR_SYSTIMER_FUNC_CLK_SEL 字段来选择时钟源。XTAL_CLK 经 2 分频后，在一个计数周期生成频率为 $f_{XTAL\_CLK}/2$ 的时钟信号，即 16 MHz，如图 11-2 所示。每个 CNT_CLK 时钟周期，计数递增 1/16 μs，即 16 个周期递增 1 μs。如果时钟源选择为 RC_FAST_CLK，则时钟 RC_FAST_CLK 不会分频，直接接到 SYSTIMER 控制器。

APB_CLK 为配置寄存器等软件操作提供时钟信号。更多有关 APB_CLK 的信息，见章节 7 复位和时钟。

用户可使用系统寄存器的两个相关位来控制系统定时器，具体如下：

- 置位寄存器 PCR_SYSTIMER_CONF_REG 中 PCR_SYSTIMER_CLK_EN 位，使能系统定时器的 APB_CLK 信号。
- 置位寄存器 PCR_SYSTIMER_CONF_REG 中 PCR_SYSTIMER_RST_EN 位，复位系统定时器。

注意，复位后，系统定时器的寄存器将恢复到默认值。更多信息可参考章节 7 复位和时钟。

11.4 功能描述

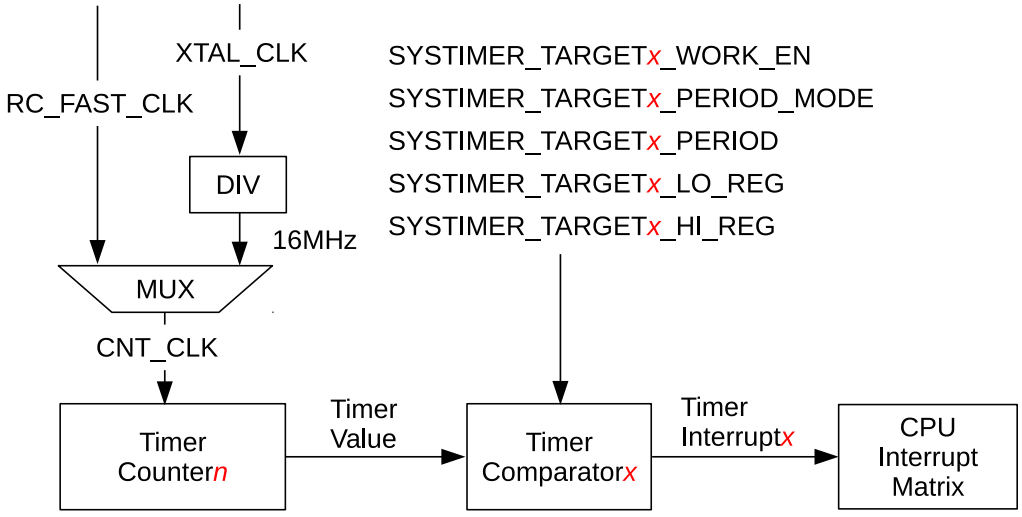


图 11-2. 系统定时器生成报警

图 11-2 展示了系统定时器生成报警的过程。过程中需要用到一个计数器和一个比较器，比较器将根据比较结果，生成报警中断。

11.4.1 计数器

系统定时器集成两个 52 位计数器，用 UNIT*n* 表示，*n* 可以取 0 或 1。计数器使用 16 MHz CNT_CLK 作为计数时钟。用户可通过配置寄存器 SYSTIMER_CONF_REG 中的两个位来控制计数器 UNIT*n*，具体如下：

- SYSTIMER_TIMER_UNIT*n*_WORK_EN：置位此位，使能计数器 UNIT*n*。
- SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN：置位此位，CPU 停止运行后，计数器 UNIT*n* 也将停止工作。CPU 恢复运行后，计数器重新开始工作。

假设 CPU 当前状态为停止工作，UNIT*n* 的具体配置见下表：

表 11-1. UNIT*n* 配置控制位

SYSTIMER_TIMER_UNIT*n*_WORK_EN	SYSTIMER_TIMER_UNIT*n*_CORE0_STALL_EN	计数器 UNIT*n*
0	x *	未处于工作状态。
1	1	暂停计数，但在 CPU 恢复运行后继续计数。
1	0	不受影响，照常计数。

* x：无关项

计数器 UNIT*n* 处于工作状态时，计数值按计数周期递增。UNIT*n* 停止工作时，计数值将保持不变，不再递增。

计数起始值的低 32 位和高 20 位分别从 SYSTIMER_TIMER_UNIT*n*_LOAD_LO 和 SYSTIMER_TIMER_UNIT*n*_LOAD_HI 中装载。置位 SYSTIMER_TIMER_UNIT*n*_LOAD 将触发重装载事件，当前计数起始值立即更新。如果计数器 UNIT*n* 处于工作状态，则从新装载的计数值开始计数。

置位 SYSTIMER_TIMER_UNIT*n*_UPDATE 将触发更新事件。当前计数值的低 32 位和高 20 位被锁存至寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 后，SYSTIMER_TIMER_UNIT*n*_VALUE_VALID 会被置起。在下次更新事件发生前，SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 寄存器中的值保持不变。

11.4.2 比较器和报警

系统定时器有三个 52 位比较器，用 COMP*x* 表示，其中 *x* 可以取 0、1、2。比较器可根据设置的不同报警值 (t) 或报警周期 (δt)，触发不同的中断。

用户可配置寄存器 SYSTIMER_TARGET*x*_PERIOD_MODE，选择比较器 COMP*x* 生成报警的模式：

- 1：周期报警模式
- 0：单次报警模式

选择周期报警模式时，寄存器 SYSTIMER_TARGET*x*_PERIOD 中的值为报警周期 (δt)。假设当前计数值为 t1，经过一段时间，当计数值达到 t1 + δt 时，将触发一次报警中断。再经过一段时间，当计数值达到 t1 + 2*δt 时，将再一次触发报警中断，以此类推。通过上述方式即可实现周期性报警。

选择单次报警模式时，SYSTIMER_TIMER_TARGET*x*_LO 和 SYSTIMER_TIMER_TARGET*x*_HI 分别提供报警值 (t) 的低 32 位和高 20 位。假设当前计数值为 t2 (t2 <= t)，经过一段时间，当计数到报警值 (t) 时，触发一次报警。与周期报警模式不同，单次报警模式仅生成一次报警中断。

用户可通过寄存器 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择用于与 COMP*x* 进行比较的计数器值，然后生成报警：

- 1：与计数器 UNIT*1* 的计数值进行比较
- 0：与计数器 UNIT*0* 的计数值进行比较

置位 SYSTIMER_TARGET*x*_WORK_EN，COMP*x* 开始进行比较：

- 在单次报警模式下，COMP*x* 将比较计数器中的实际计数值与寄存器中设置的报警值 (t)；
- 在周期报警模式下，COMP*x* 将比较计数器中的实际计数值与 t1 + n*δt (n = 1,2,3...)。

在单次报警模式下，当实际计数值等于报警值 (t)，或在周期报警模式下，实际计数值等于 t1 + n*δt (n = 1,2,3...) 时，将触发一次报警中断。但如果设定的报警值 (t) 小于当前计数值，即报警值 (t) 已成为过去，在当前计数值超过设定的报警值 (t) 一定范围 (0 ~ 2^{51} -1) 时，也将立即触发中断。无论是在单次报警模式还是在周期报警模式下，都可以从 SYSTIMER_TARGET*x*_LO_RO 和 SYSTIMER_TARGET*x*_HI_RO 中读取实际报警值的低 32 位和高 20 位。当前计数值 t_c、报警值 t_t 和触发报警的关系如下表所示：

表 11-2. 报警触发条件

t_c 与 t_t 的关系	触发条件
t_c- t_t <= 0	当 t_c = t_t 时，触发报警
0 <= t_c - t_t < 2^{51} - 1 (t_c < 2^{51} 且 t_t < 2^{51}； 或 t_c >= 2^{51} 且 t_t >= 2^{51})	立即触发报警
t_c - t_t >= 2^{51} - 1	t_c 达到最大值 52'hfffffffffffff 后溢出，然后从 0 开始计数，计数达到 t_t 时触发报警

11.4.3 事件任务矩阵

在 ESP32-H2 中，系统定时器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发系统定时器的 ETM 任务，或者通过系统定时器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与系统定时器相关的 ETM 任务和 ETM 事件。

系统定时器可产生的 ETM 事件有：

- SYSTIMER_EVT_CNT_CMP*x*：表示 COMP*x* 产生的报警事件。

当 SYSTIMER_ETM_EN 置 1 时，对应的报警事件会传到 ETM 模块。

11.4.4 同步操作

不同于计数器和比较器工作时使用的时钟 CNT_CLK，软件操作的时钟为 APB_CLK。时钟频率不同，因此需要对部分配置寄存器进行同步。完整的同步过程包括下面两个步骤：

1. 通过软件向配置寄存器字段写入合适的值，见表 11-3 第一列。
2. 通过软件置位相应的同步使能位，开始同步操作，见表 11-3 第二列。

表 11-3. 同步操作

需要同步的字段	同步使能位
SYSTIMER_TIMER_UNIT*n*_LOAD_LO SYSTIMER_TIMER_UNIT*n*_LOAD_HI	SYSTIMER_TIMER_UNIT*n*_LOAD
SYSTIMER_TARGET*x*_PERIOD SYSTIMER_TIMER_TARGET*x*_HI SYSTIMER_TIMER_TARGET*x*_LO	SYSTIMER_TIMER_COMP*x*_LOAD

由于与计数器的相关状态使用的时钟并非 APB_CLK，时钟频率不同，因此，读取状态寄存器时也需要进行同步。完整的同步步骤如下：

1. 软件将 1 写入更新寄存器 SYSTIMER_TIMER_UNIT*n*_UPDATE 中。
2. 软件读取相应位 SYSTIMER_TIMER_UNIT*n*_VALUE_VALID 为有效，即表示同步完成。

3. 软件读取相应的状态寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO。

11.4.5 中断

上述三个比较器均有一个对应的报警中断，即 SYSTIMER_TARGET*x*_INT 中断，该中断为电平类型中断。比较器开始报警时，中断信号拉高。中断信号将一直保持高电平，直至软件清除中断。用户可置位 SYSTIMER_TARGET*x*_INT_ENA 使能中断。

11.5 编程示例

注意，在配置 COMP*x* 和 UNIT*n* 过程中，需保证对应的 COMP 和 UNIT 处于工作状态。

11.5.1 读取当前计数器的值

1. 置位 SYSTIMER_TIMER_UNIT*n*_UPDATE，将计数器 UNIT*n* 的值更新至寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO；
2. 轮询 SYSTIMER_TIMER_UNIT*n*_VALUE_VALID，直至其值为 1。之后，用户可从寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 和 SYSTIMER_TIMER_UNIT*n*_VALUE_LO 中读取计数器的值；
3. 读取寄存器 SYSTIMER_TIMER_UNIT*n*_VALUE_LO（低 32 位）和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI（高 20 位）。

11.5.2 在单次报警模式下配置一次性报警

1. 设置 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择与 COMP*x* 进行比较的计数器（UNIT*0* 或 UNIT*1*）。
2. 读取当前计数器的值，步骤见章节 11.5.1。读取的当前值可用于计算步骤 4 中的报警值 (t)。
3. 清除 SYSTIMER_TARGET*x*_PERIOD_MODE，使能单次报警模式。
4. 设置报警值 (t)，并将报警值 (t) 的低 32 位和高 20 位分别写入 SYSTIMER_TIMER_TARGET*x*_LO 和 SYSTIMER_TIMER_TARGET*x*_HI。
5. 置位 SYSTIMER_TIMER_COMP*x*_LOAD，同步报警值 (t)，即将报警值 (t) 装载至比较器 COMP*x*。
6. 置位 SYSTIMER_TARGET*x*_WORK_EN 使能选择的比较器 COMP*x*；比较器 COMP*x* 开始比较计数值与报警值 (t)。
7. 置位 SYSTIMER_TARGET*x*_INT_ENA，使能中断。Unit*n* 达到报警值 (t) 时，触发一次报警中断 SYSTIMER_TARGET*x*_INT。

11.5.3 在周期报警模式下配置周期性报警

1. 设置 SYSTIMER_TARGET*x*_TIMER_UNIT_SEL 选择与 COMP*x* 进行比较的计数器。
2. 将报警周期 (δt) 写入 SYSTIMER_TARGET*x*_PERIOD。
3. 置位 SYSTIMER_TIMER_COMP*x*_LOAD 同步报警周期值，即将 (δt) 装载至比较器 COMP*x*。
4. 先清除再置位 SYSTIMER_TARGET*x*_PERIOD_MODE 将 COMP*x* 配置为周期报警模式。
5. 置位 SYSTIMER_TARGET*x*_WORK_EN 使能选择的比较器 COMP*x*；比较器 COMP*x* 开始将计数值与计数初始值 + n*δt (n = 1, 2, 3...) 进行比较。

6. 置位 SYSTIMER_TARGET*x*_INT_ENA，使能中断。Unit*n* 计数达到计数初始值 + n*δt (n = 1, 2, 3...) 时，触发一次 SYSTIMER_TARGET*x*_INT 中断。

11.5.4 唤醒后时间补偿

1. 在芯片进入 Deep-sleep 或 Light-sleep 模式之前，用户需配置 RTC 定时器，精确记录睡眠时间，见章节 2 *低功耗管理 (RTC_CNTL) [to be added later]*。
2. 从 Deep-sleep 或 Light-sleep 模式唤醒后，读取 RTC 定时器记录的睡眠时间。
3. 读取当前系统定时器的计数值，见章节 11.5.1。
4. 将 RTC 记录的以 RTC_SLOW_CLK 周期为单位的睡眠时间，转换成以 CNT_CLK (16 MHz) 周期为单位的睡眠时间。例如，如果 RTC_SLOW_CLK 频率为 32 kHz，则 RTC 定时器记录的时间乘以 500。
5. 将 RTC 定时器转换后的值加到系统定时器的当前计数值：
   - 将计算所得值的低 32 位写入 SYSTIMER_TIMER_UNIT*n*_LOAD_LO，高 20 位写入 SYSTIMER_TIMER_UNIT*n*_LOAD_HI。
   - 置位 SYSTIMER_TIMER_UNIT*n*_LOAD，将新的定时器值装载到系统定时器。至此，系统定时器更新完成。

11.6 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
时钟控制寄存器			
SYSTIMER_CONF_REG	配置系统定时器时钟	0x0000	R/W
UNIT*0* 控制和配置寄存器			
SYSTIMER_UNIT0_OP_REG	读取 UNT0 的值到相应寄存器	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	待装载至 UNIT0 的值，高 20 位	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	待装载至 UNIT0 的值，低 32 位	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 的读值，高 20 位	0x0040	RO
SYSTIMER_UNIT0_VALUE_LO_REG	UNIT0 的读值，低 32 位	0x0044	RO
SYSTIMER_UNIT0_LOAD_REG	计数器 UNIT0 的装载同步寄存器	0x005C	WT
UNIT*1* 控制和配置寄存器			
SYSTIMER_UNIT1_OP_REG	读取计数器 UNIT1 的值	0x0008	varies
SYSTIMER_UNIT1_LOAD_HI_REG	待装载至计数器 UNIT1 的值，高 20 位	0x0014	R/W
SYSTIMER_UNIT1_LOAD_LO_REG	待装载至计数器 UNIT1 的值，低 32 位	0x0018	R/W
SYSTIMER_UNIT1_VALUE_HI_REG	计数器 UNIT1 的读值，高 20 位	0x0048	RO
SYSTIMER_UNIT1_VALUE_LO_REG	计数器 UNIT1 的读值，低 32 位	0x004C	RO
SYSTIMER_UNIT1_LOAD_REG	计数器 UNIT1 的装载同步寄存器	0x0060	WT
比较器 COMP*0* 的控制和配置寄存器			
SYSTIMER_TARGET0_HI_REG	待装载至比较器 COMP0 的报警值，高 20 位	0x001C	R/W
SYSTIMER_TARGET0_LO_REG	待装载至比较器 COMP0 的报警值，低 32 位	0x0020	R/W
SYSTIMER_TARGET0_CONF_REG	配置比较器 COMP0 的报警模式	0x0034	R/W
SYSTIMER_COMP0_LOAD_REG	比较器 COMP0 的装载同步寄存器	0x0050	WT
比较器 COMP*1* 的控制和配置寄存器			
SYSTIMER_TARGET1_HI_REG	待装载至比较器 COMP1 的报警值，高 20 位	0x0024	R/W
SYSTIMER_TARGET1_LO_REG	待装载至比较器 COMP1 的报警值，低 32 位	0x0028	R/W
SYSTIMER_TARGET1_CONF_REG	配置比较器 COMP1 的报警模式	0x0038	R/W
SYSTIMER_COMP1_LOAD_REG	比较器 COMP1 的装载同步寄存器	0x0054	WT
比较器 COMP*2* 的控制和配置寄存器			
SYSTIMER_TARGET2_HI_REG	待装载至比较器 COMP2 的报警值，高 20 位	0x002C	R/W
SYSTIMER_TARGET2_LO_REG	待装载至比较器 COMP2 的报警值，低 32 位	0x0030	R/W
SYSTIMER_TARGET2_CONF_REG	配置比较器 COMP2 的报警模式	0x003C	R/W
SYSTIMER_COMP2_LOAD_REG	比较器 COMP2 的装载同步寄存器	0x0058	WT
中断寄存器			
SYSTIMER_INT_ENA_REG	系统定时器的中断使能寄存器	0x0064	R/W
SYSTIMER_INT_RAW_REG	系统定时器的原始中断寄存器	0x0068	R/WTC/SS
SYSTIMER_INT_CLR_REG	系统定时器的中断清除寄存器	0x006C	WT
SYSTIMER_INT_ST_REG	系统定时器的中断状态寄存器	0x0070	RO

名称	描述	地址	访问
COMP*0* 状态寄存器			
SYSTIMER_REAL_TARGET0_LO_REG	COMP0 的实际报警值，低 32 位	0x0074	RO
SYSTIMER_REAL_TARGET0_HI_REG	COMP0 的实际报警值，高 20 位	0x0078	RO
COMP*1* 状态寄存器			
SYSTIMER_REAL_TARGET1_LO_REG	COMP1 的实际报警值，低 32 位	0x007C	RO
SYSTIMER_REAL_TARGET1_HI_REG	COMP1 的实际报警值，高 20 位	0x0080	RO
COMP*2* 状态寄存器			
SYSTIMER_REAL_TARGET2_LO_REG	COMP2 的实际报警值，低 32 位	0x0084	RO
SYSTIMER_REAL_TARGET2_HI_REG	COMP2 的实际报警值，高 20 位	0x0088	RO
版本寄存器			
SYSTIMER_DATE_REG	版本控制寄存器	0x00FC	R/W

11.7 寄存器

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

Register 11.1. SYSTIMER_CONF_REG (0x0000)

Bit	31	30	29	28	27	26	25	24	23	22	21 ... 2	1	0
Field	SYSTIMER_CLK_EN	SYSTIMER_TIMER_UNIT0_WORK_EN	SYSTIMER_TIMER_UNIT1_WORK_EN	SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN	SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN	SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN	SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN	SYSTIMER_TARGET0_WORK_EN	SYSTIMER_TARGET1_WORK_EN	SYSTIMER_TARGET2_WORK_EN	(reserved)	SYSTIMER_ETM_EN	(reserved)
Reset	0	1	0	0	0	1	1	0	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0

SYSTIMER_ETM_EN 配置是否使能生成 ETM 事件。

0：无效

1：使能生成 ETM 事件

(R/W)

SYSTIMER_TARGET2_WORK_EN 配置是否使能 COMP2。

0：不使能

1：使能 COMP2

(R/W)

SYSTIMER_TARGET1_WORK_EN 配置是否使能 COMP1。具体配置值请见 SYSTIMER_TARGET2_WORK_EN。(R/W)

SYSTIMER_TARGET0_WORK_EN 配置是否使能 COMP0。具体配置值请见 SYSTIMER_TARGET2_WORK_EN。(R/W)

SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT1 是否也将停止工作。

0：无效

1：计数器 UNIT1 停止工作

(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT1 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

SYSTIMER_TIMER_UNIT0_CORE1_STALL_EN 配置如果 CPU1 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN 配置如果 CPU0 停止工作，计数器 UNIT0 是否也将停止工作。具体配置值请见 SYSTIMER_TIMER_UNIT1_CORE1_STALL_EN。(R/W)

见下页...

Register 11.1. SYSTIMER_CONF_REG (0x0000)

... 接上页

SYSTIMER_TIMER_UNIT1_WORK_EN 配置是否使能计数器 UNIT1。

0：无效

1：使能计数器 UNIT1

(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN 配置是否使能计数器 UNIT0。

0：无效

1：使能计数器 UNIT0

(R/W)

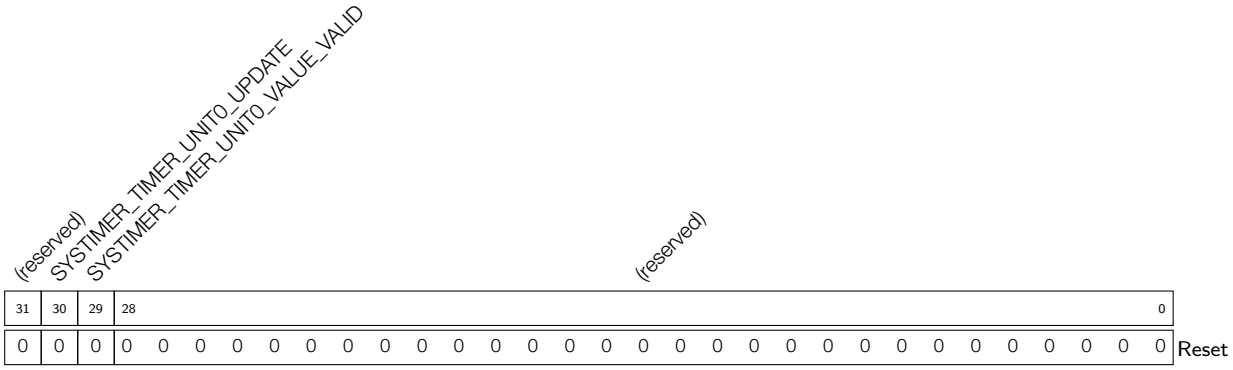
SYSTIMER_CLK_EN 配置寄存器时钟门控。

0：只在读写寄存器时打开所需时钟

1：持续开启读写寄存器的时钟

(R/W)

Register 11.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID 表示计数器 UNIT0 的值已同步读取至寄存器，且读值有效。(R/SS/WTC)

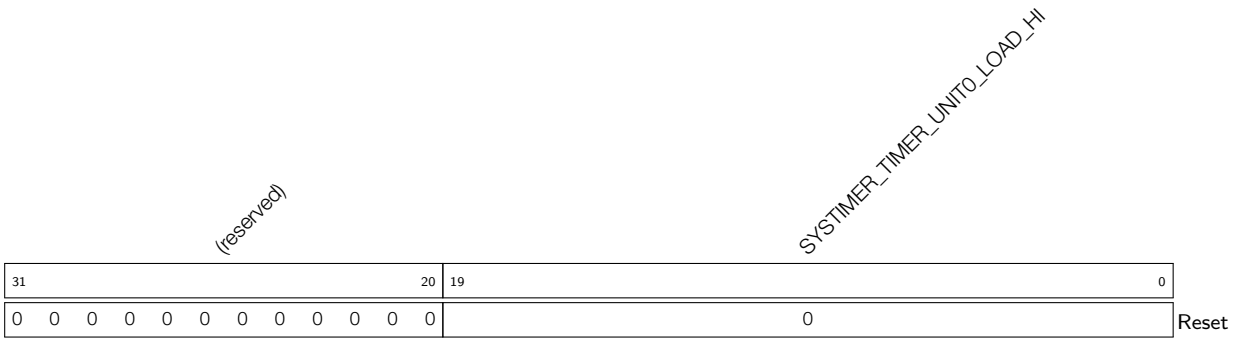
SYSTIMER_TIMER_UNIT0_UPDATE 配置是否更新计数器 UNIT0，即读取计数器 UNIT0 的值到寄存器 SYSTIMER_TIMER_UNIT0_VALUE_HI 和 SYSTIMER_TIMER_UNIT0_VALUE_LO。

0：无效

1：更新计数器 UNIT0

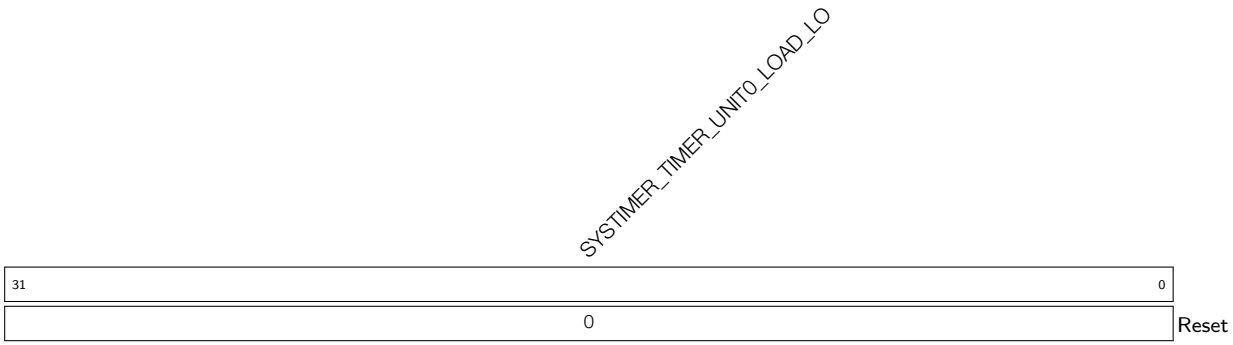
(WT)

Register 11.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



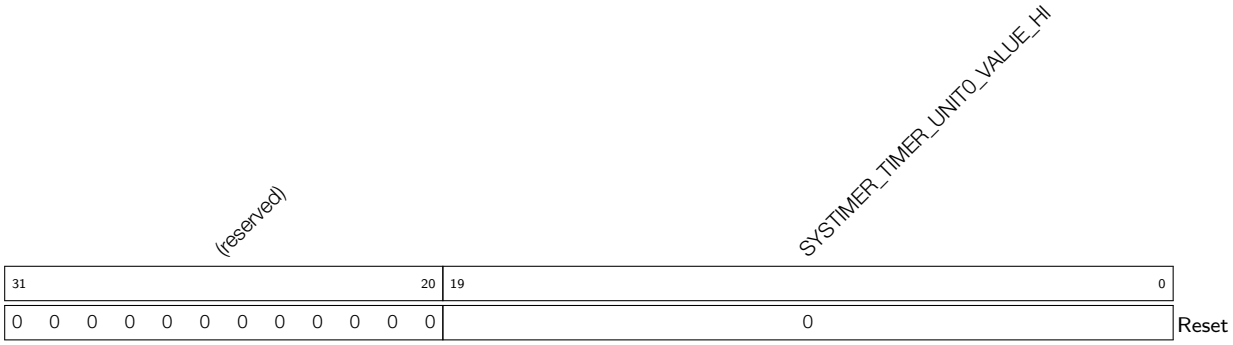
SYSTIMER_TIMER_UNIT0_LOAD_HI 配置待装载至计数器 UNIT0 的值，高 20 位。(R/W)

Register 11.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



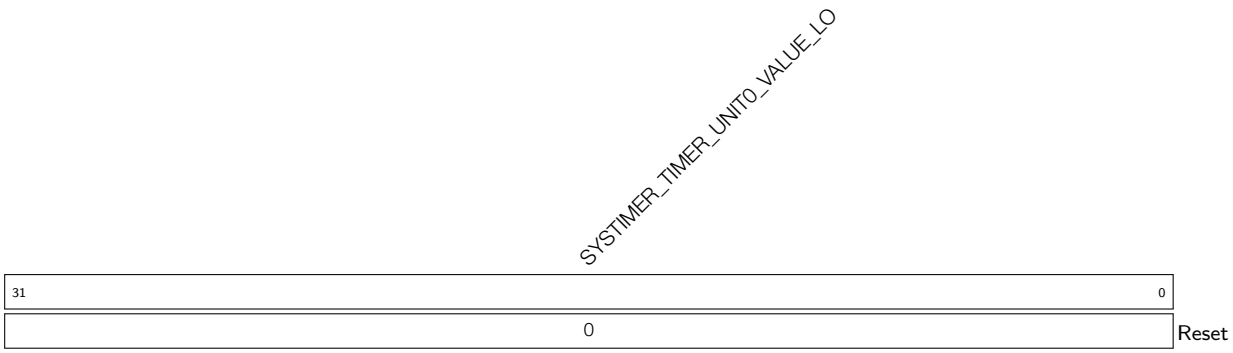
SYSTIMER_TIMER_UNIT0_LOAD_LO 配置待装载至计数器 UNIT0 的值，低 32 位。(R/W)

Register 11.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



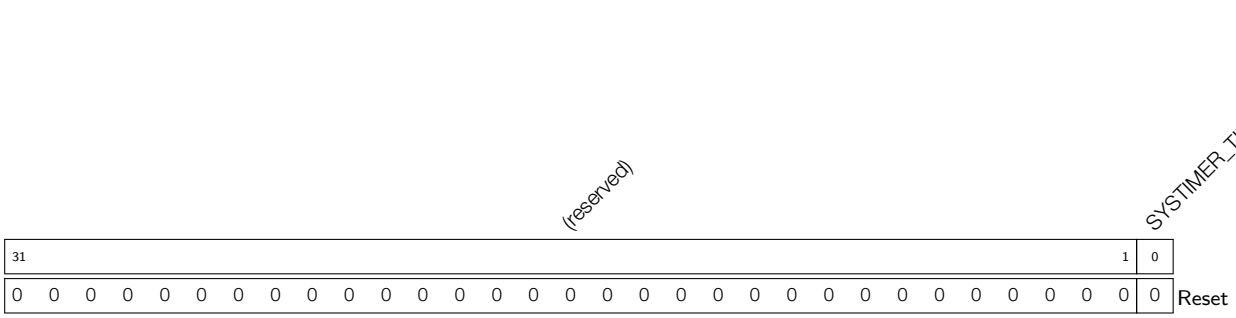
SYSTIMER_TIMER_UNIT0_VALUE_HI 表示计数器 UNIT0 的读数，高 20 位。(RO)

Register 11.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



SYSTIMER_TIMER_UNIT0_VALUE_LO 表示计数器 UNIT0 的读数，低 32 位。(RO)

Register 11.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



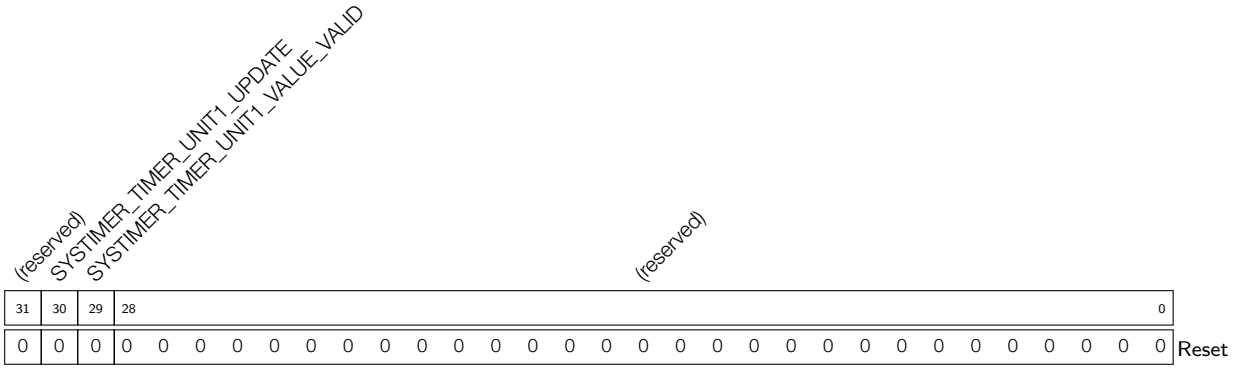
SYSTIMER_TIMER_UNIT0_LOAD 配置是否重新装载计数器 UNIT0，即是否重新装载寄存器 SYSTIMER_TIMER_UNIT0_LOAD_HI 和 SYSTIMER_TIMER_UNIT0_LOAD_LO 的值到计数器 UNIT0。

0：无效

1：重新装载计数器 UNIT0

(WT)

Register 11.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID 表示计数器 UNIT1 的值已同步读取至寄存器，且读值有效。(R/SS/WTC)

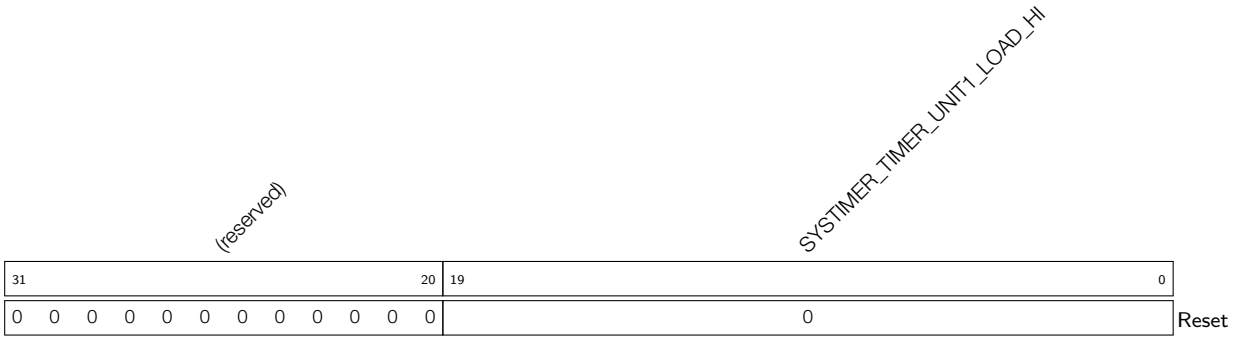
SYSTIMER_TIMER_UNIT1_UPDATE 配置是否更新计数器 UNIT1，即将计数器 UNIT1 的值读取到寄存器 SYSTIMER_TIMER_UNIT1_VALUE_HI 和 SYSTIMER_TIMER_UNIT1_VALUE_LO。

0：无效

1：更新计数器 UNIT1

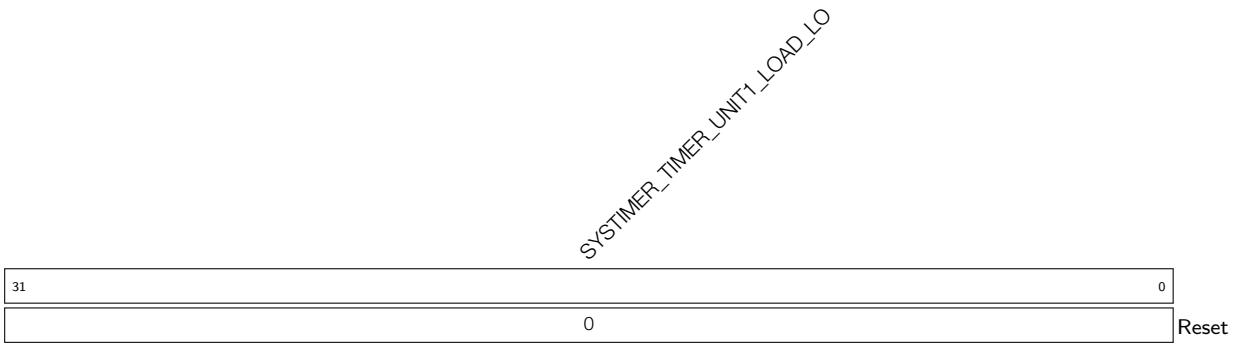
(WT)

Register 11.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



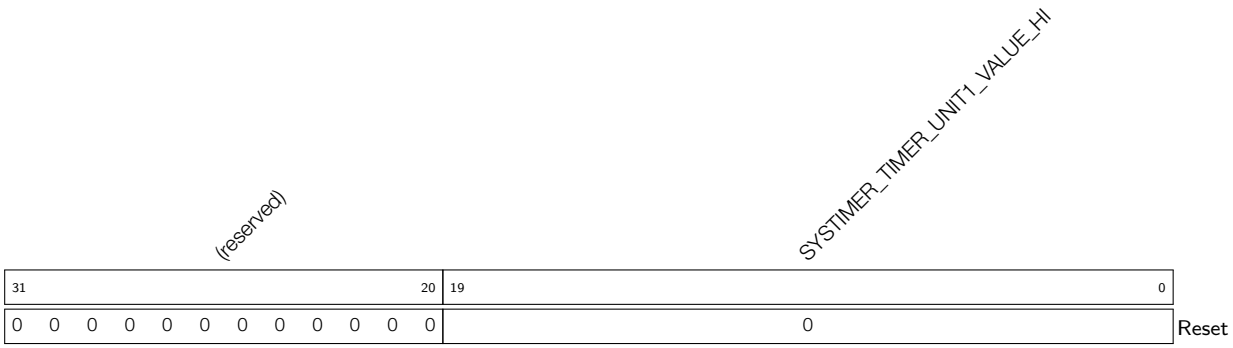
SYSTIMER_TIMER_UNIT1_LOAD_HI 配置待装载至计数器 UNIT1 的值，高 20 位。(R/W)

Register 11.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



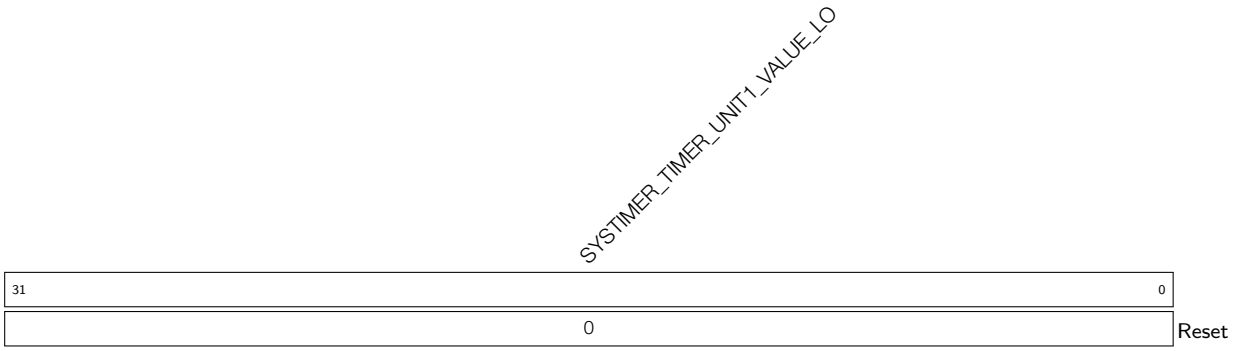
SYSTIMER_TIMER_UNIT1_LOAD_LO 配置待装载至计数器 UNIT1 的值，低 32 位。(R/W)

Register 11.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



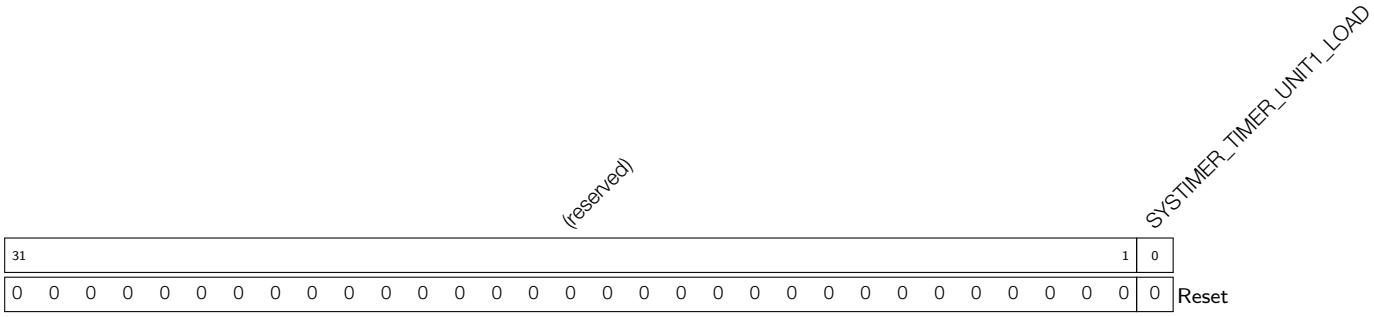
SYSTIMER_TIMER_UNIT1_VALUE_HI 表示计数器 UNIT1 的读数，高 20 位。(RO)

Register 11.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



SYSTIMER_TIMER_UNIT1_VALUE_LO 表示计数器 UNIT1 的读数，低 32 位。(RO)

Register 11.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



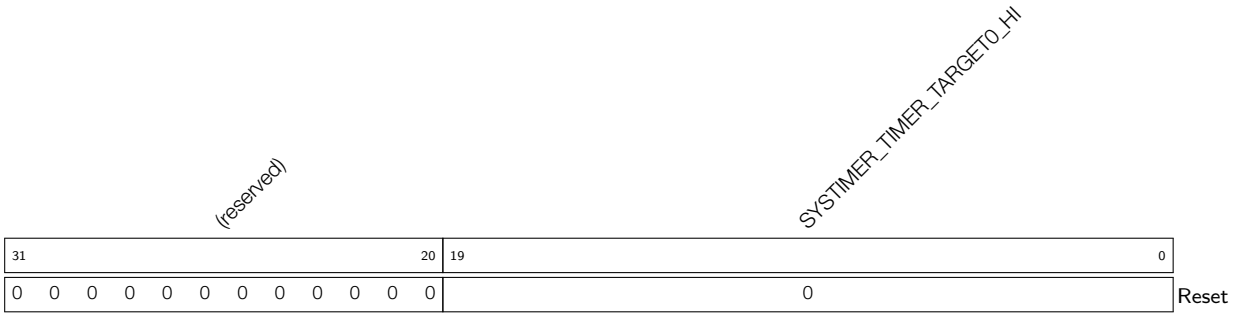
SYSTIMER_TIMER_UNIT1_LOAD 配置是否重新装载计数器 UNIT1，即是否重新装载寄存器 SYSTIMER_TIMER_UNIT1_LOAD_HI 和 SYSTIMER_TIMER_UNIT1_LOAD_LO 的值到计数器 UNIT1。

0：无效

1：重新装载计数器 UNIT1

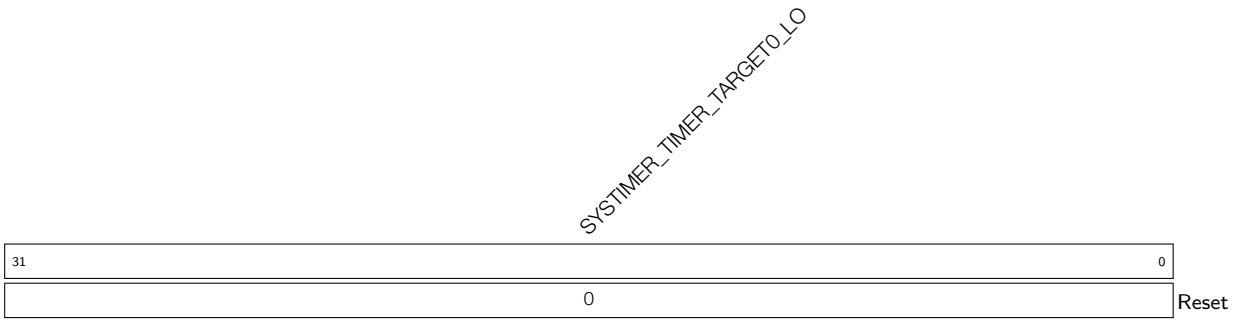
(WT)

Register 11.14. SYSTIMER_TARGET0_HI_REG (0x001C)



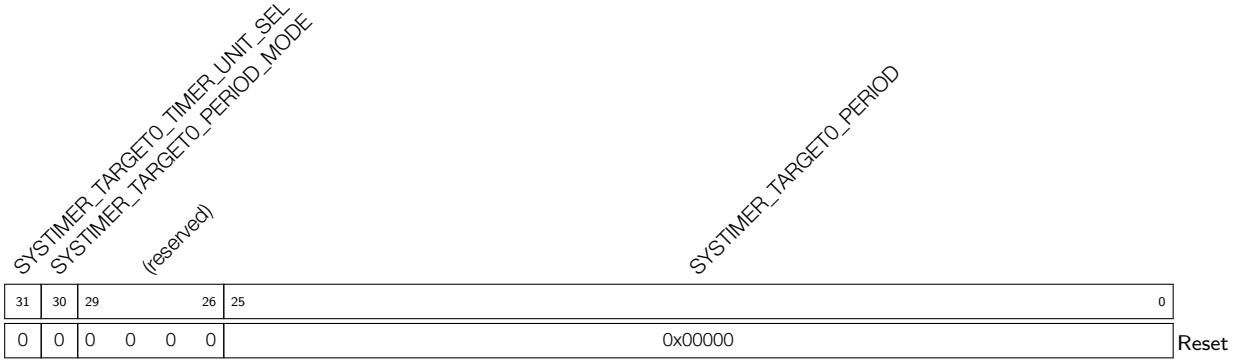
SYSTIMER_TIMER_TARGET0_HI 配置待装载至 COMP0 的报警值，高 20 位。(R/W)

Register 11.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO 配置待装载至 COMP0 的报警值，低 32 位。(R/W)

Register 11.16. SYSTIMER_TARGET0_CONF_REG (0x0034)



SYSTIMER_TARGET0_PERIOD 配置待装载至 COMP0 的报警周期。(R/W)

SYSTIMER_TARGET0_PERIOD_MODE 选择比较器 COMP0 生成报警的模式。

0：单次报警模式

1：周期报警模式

(R/W)

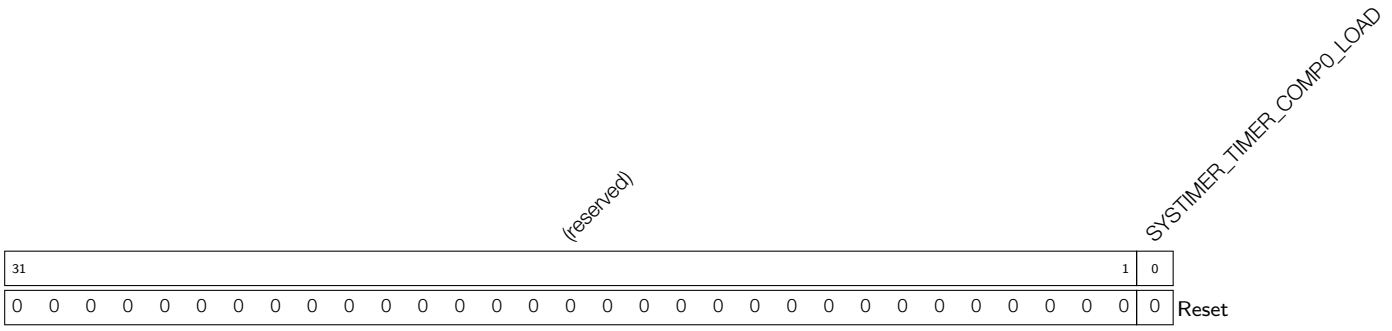
SYSTIMER_TARGET0_TIMER_UNIT_SEL 选择要与 COMP0 比较的计数器。

0：与计数器 UNIT*0* 的计数值进行比较

1：与计数器 UNIT*1* 的计数值进行比较

(R/W)

Register 11.17. SYSTIMER_COMP0_LOAD_REG (0x0050)



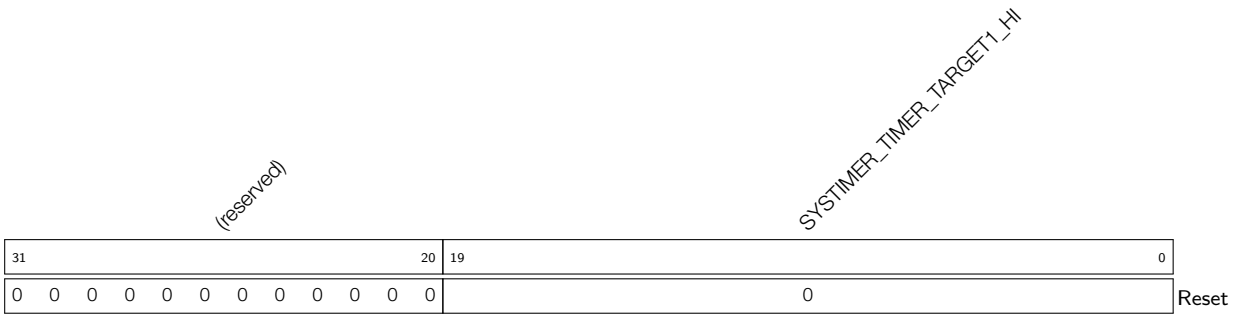
SYSTIMER_TIMER_COMP0_LOAD 配置是否使能比较器 COMP0 同步，即是否重新装载报警值或报警周期到 COMP0。

0：无效

1：使能比较器 COMP0 同步

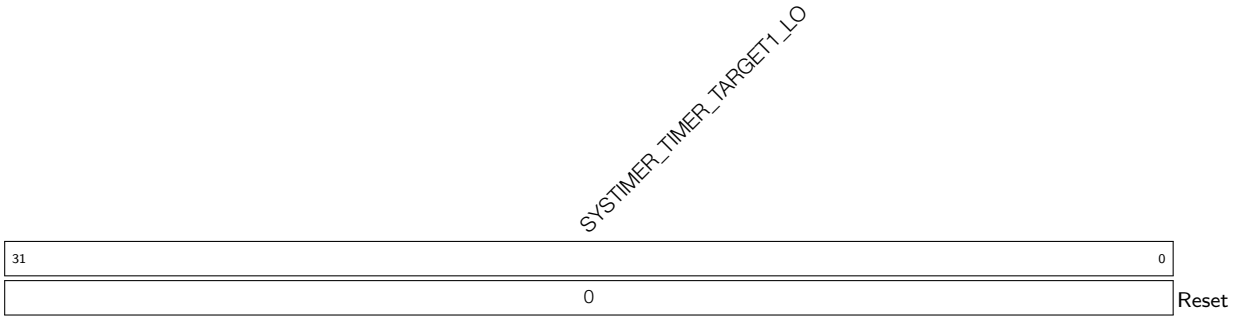
(WT)

Register 11.18. SYSTIMER_TARGET1_HI_REG (0x0024)



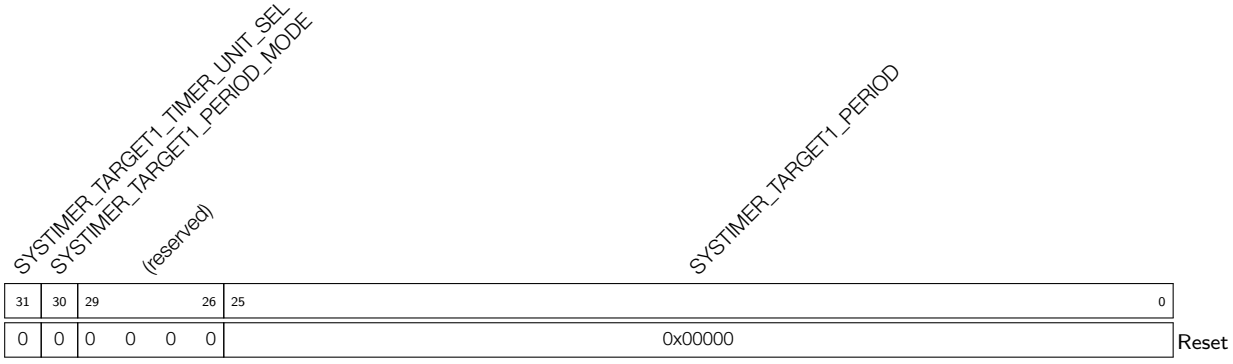
SYSTIMER_TIMER_TARGET1_HI 配置待装载至 COMP1 的报警值，高 20 位。(R/W)

Register 11.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO 配置待装载至 COMP1 的报警值，低 32 位。(R/W)

Register 11.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

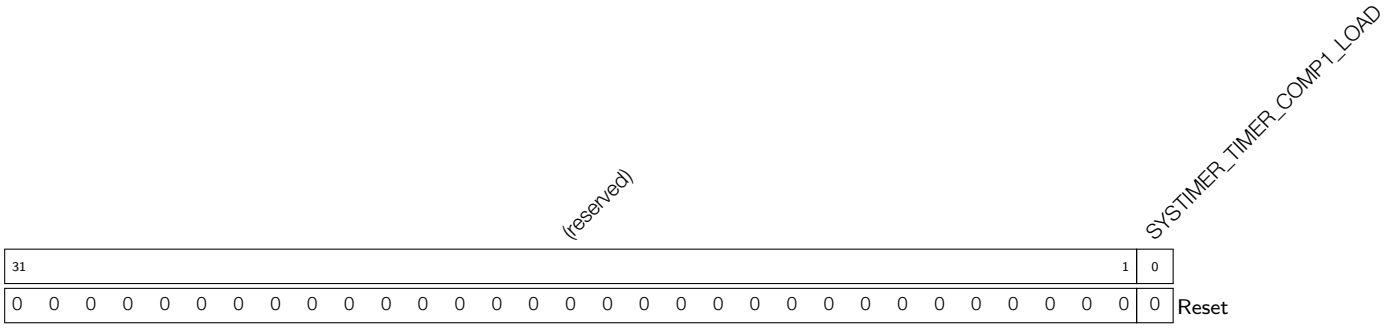


SYSTIMER_TARGET1_PERIOD 配置待装载至 COMP1 的报警周期。(R/W)

SYSTIMER_TARGET1_PERIOD_MODE 选择比较器 COMP1 生成报警的模式。具体配置值请见 SYSTIMER_TARGET0_PERIOD_MODE。(R/W)

SYSTIMER_TARGET1_TIMER_UNIT_SEL 选择要与 COMP1 比较的计数器。具体配置值请见 SYSTIMER_TARGET0_TIMER_UNIT_SEL。(R/W)

Register 11.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



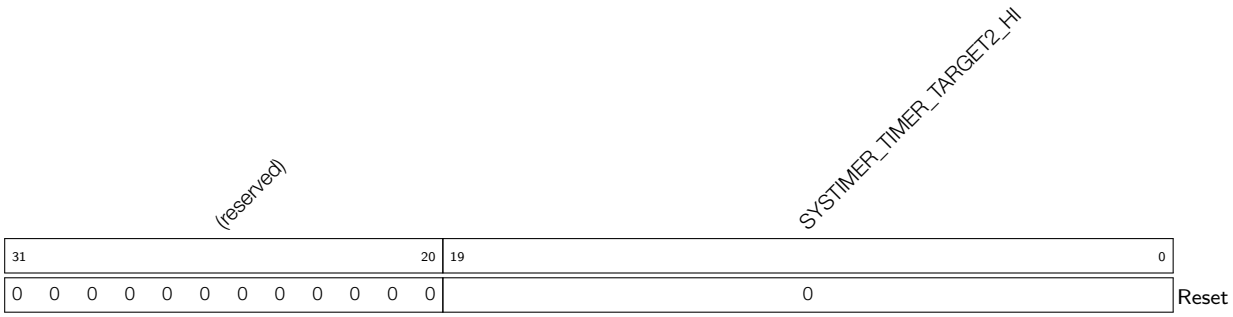
SYSTIMER_TIMER_COMP1_LOAD 配置是否使能比较器 COMP1 同步，即是否重新装载报警值或报警周期到 COMP1。

0：无效

1：使能比较器 COMP1 同步

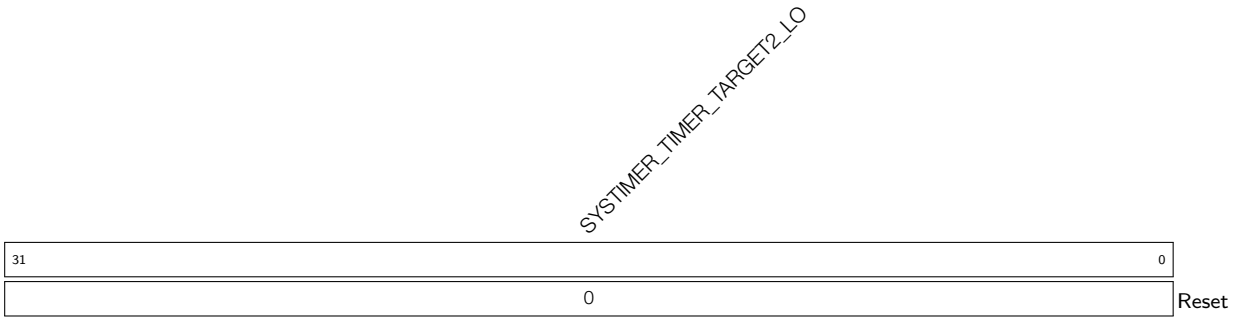
(WT)

Register 11.22. SYSTIMER_TARGET2_HI_REG (0x002C)



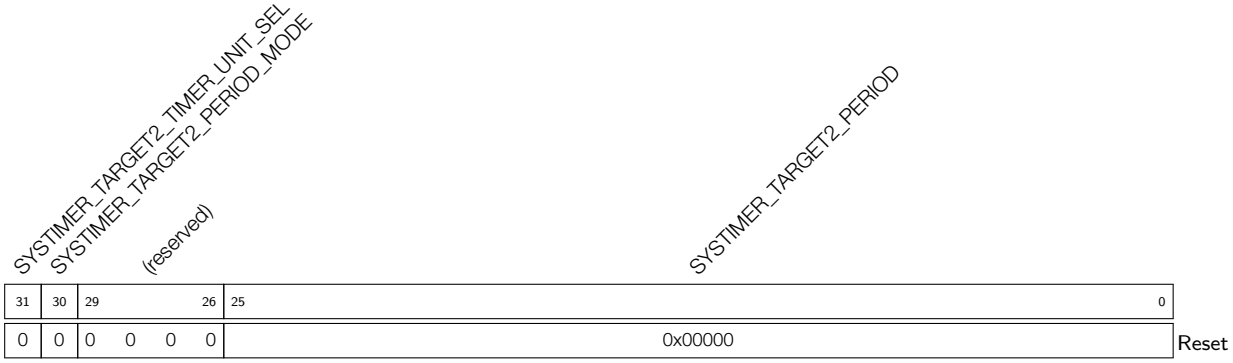
SYSTIMER_TIMER_TARGET2_HI 配置待装载至比较器 COMP2 的报警值，高 20 位。(R/W)

Register 11.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO 配置待装载至比较器 COMP2 的报警值，低 32 位。(R/W)

Register 11.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

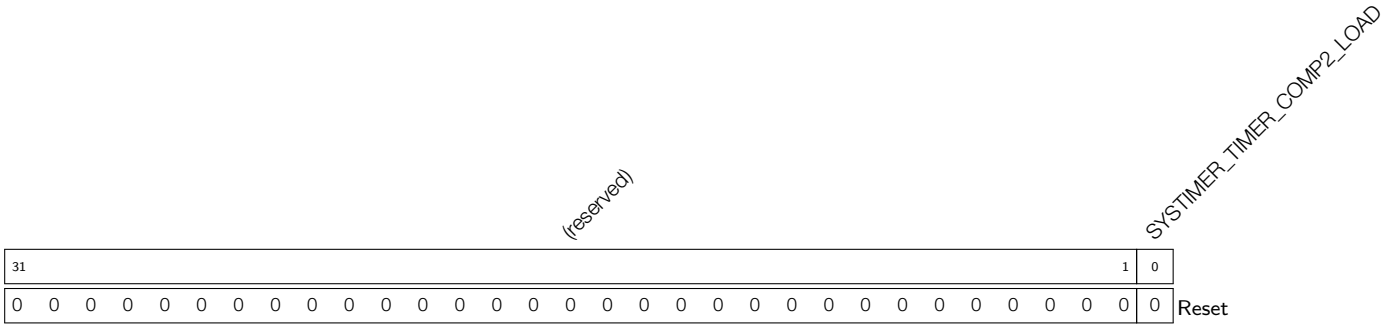


SYSTIMER_TARGET2_PERIOD 配置待装载至 COMP2 的报警周期。(R/W)

SYSTIMER_TARGET2_PERIOD_MODE 选择比较器 COMP2 生成报警的模式。具体配置值请见 SYSTIMER_TARGET0_PERIOD_MODE。(R/W)

SYSTIMER_TARGET2_TIMER_UNIT_SEL 选择要与 COMP2 比较的计数器。具体配置值请见 SYSTIMER_TARGET0_TIMER_UNIT_SEL。(R/W)

Register 11.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



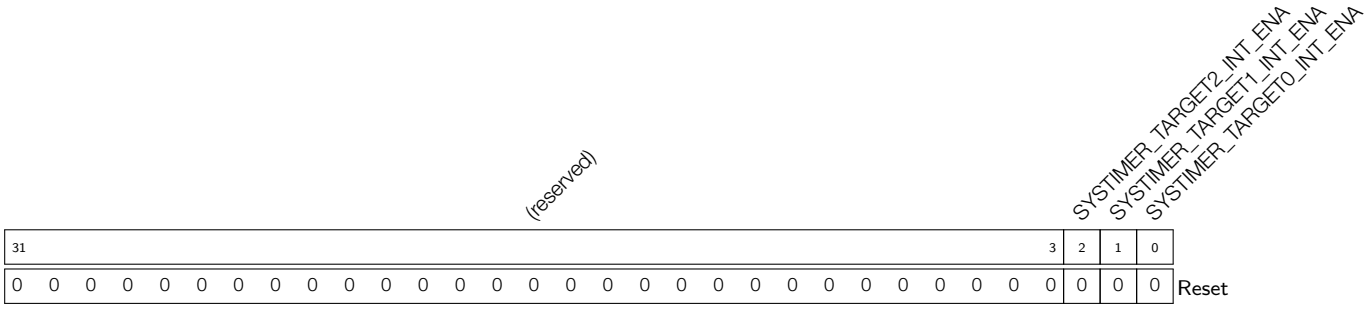
SYSTIMER_TIMER_COMP2_LOAD 配置是否使能比较器 COMP2 同步，即是否重新装载报警值或报警周期到 COMP2。

0：无效

1：使能比较器 COMP2 同步

(WT)

Register 11.26. SYSTIMER_INT_ENA_REG (0x0064)

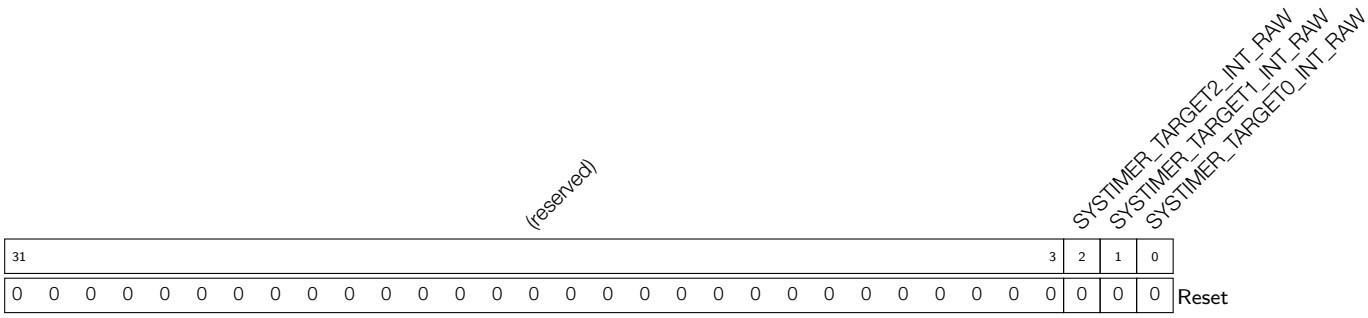


SYSTIMER_TARGET0_INT_ENA 写 1 使能 SYSTIMER_TARGET0_INT。(R/W)

SYSTIMER_TARGET1_INT_ENA 写 1 使能 SYSTIMER_TARGET1_INT。(R/W)

SYSTIMER_TARGET2_INT_ENA 写 1 使能 SYSTIMER_TARGET2_INT。(R/W)

Register 11.27. SYSTIMER_INT_RAW_REG (0x0068)

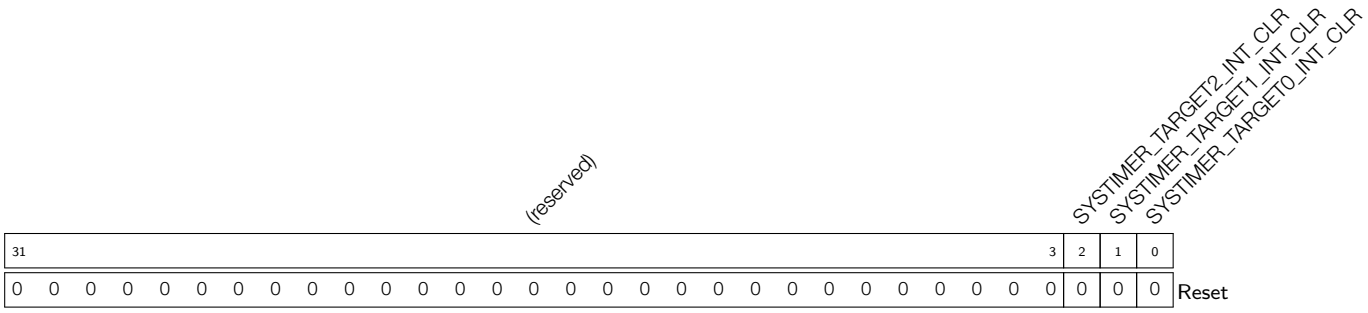


SYSTIMER_TARGET0_INT_RAW SYSTIMER_TARGET0_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW SYSTIMER_TARGET1_INT 的原始中断状态。(R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW SYSTIMER_TARGET2_INT 的原始中断状态。(R/WTC/SS)

Register 11.28. SYSTIMER_INT_CLR_REG (0x006C)

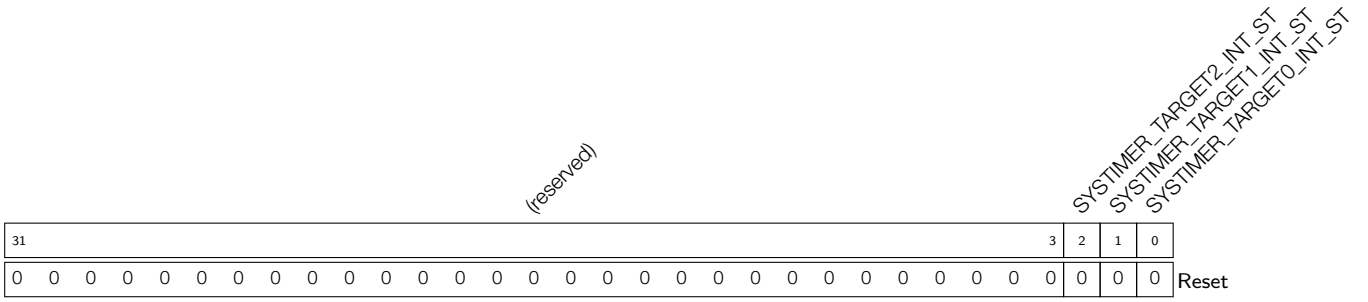


SYSTIMER_TARGET0_INT_CLR 写 1 清除 SYSTIMER_TARGET0_INT。(WT)

SYSTIMER_TARGET1_INT_CLR 写 1 清除 SYSTIMER_TARGET1_INT。(WT)

SYSTIMER_TARGET2_INT_CLR 写 1 清除 SYSTIMER_TARGET2_INT。(WT)

Register 11.29. SYSTIMER_INT_ST_REG (0x0070)

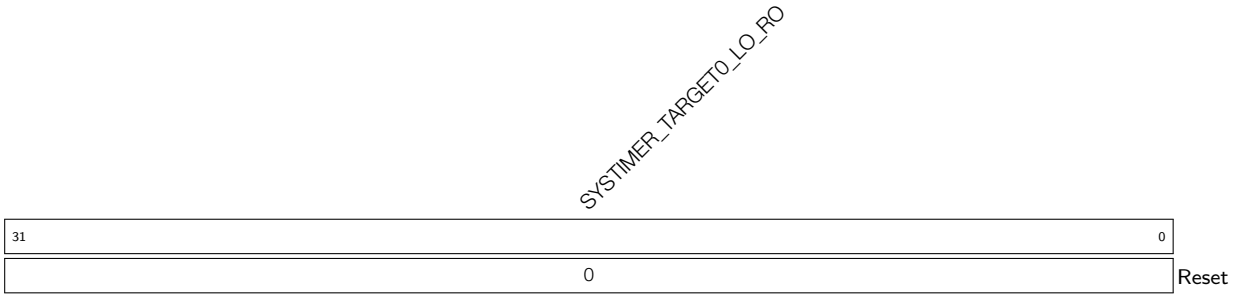


SYSTIMER_TARGET0_INT_ST SYSTIMER_TARGET0_INT 的中断状态。(RO)

SYSTIMER_TARGET1_INT_ST SYSTIMER_TARGET1_INT 的中断状态。(RO)

SYSTIMER_TARGET2_INT_ST SYSTIMER_TARGET2_INT 的中断状态。(RO)

Register 11.30. SYSTIMER_REAL_TARGET0_LO_REG (0x0074)

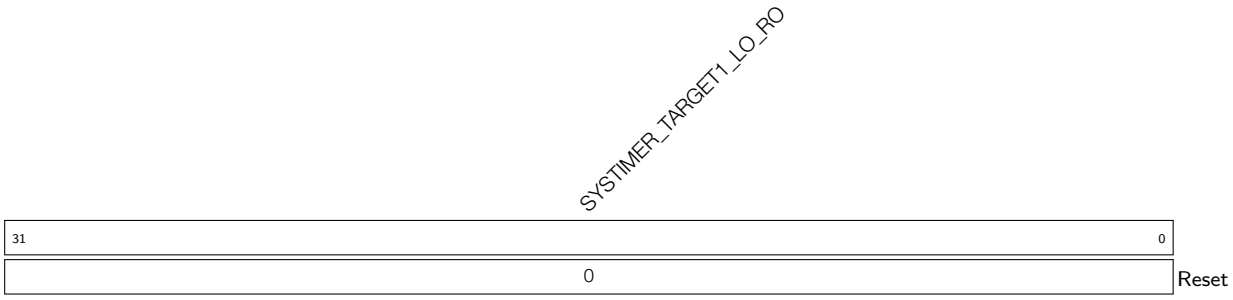


SYSTIMER_TARGET0_LO_RO 表示 COMP0 的实际报警值，低 32 位。(RO)

Register 11.31. SYSTIMER_REAL_TARGET0_HI_REG (0x0078)

SYSTIMER_TARGET0_HI_RO 表示 COMP0 的实际报警值，高 20 位。(RO)

Register 11.32. SYSTIMER_REAL_TARGET1_LO_REG (0x007C)

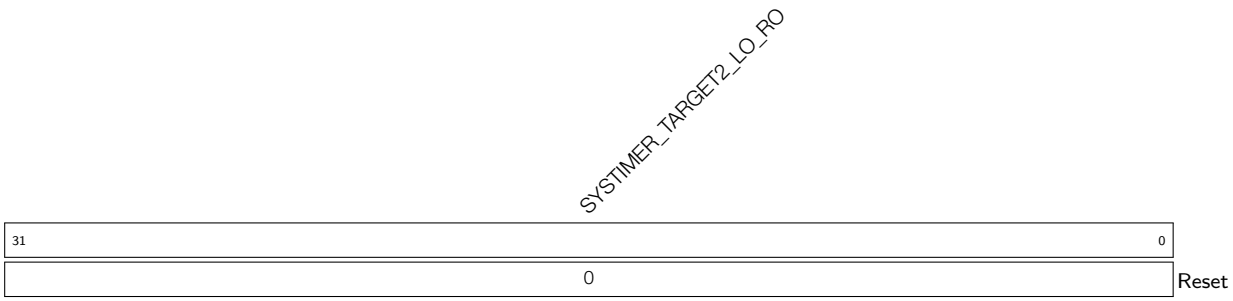


SYSTIMER_TARGET1_LO_RO 表示 COMP1 的实际报警值，低 32 位。(RO)

Register 11.33. SYSTIMER_REAL_TARGET1_HI_REG (0x0080)

SYSTIMER_TARGET1_HI_RO 表示 COMP1 的实际报警值，高 20 位。(RO)

Register 11.34. SYSTIMER_REAL_TARGET2_LO_REG (0x0084)

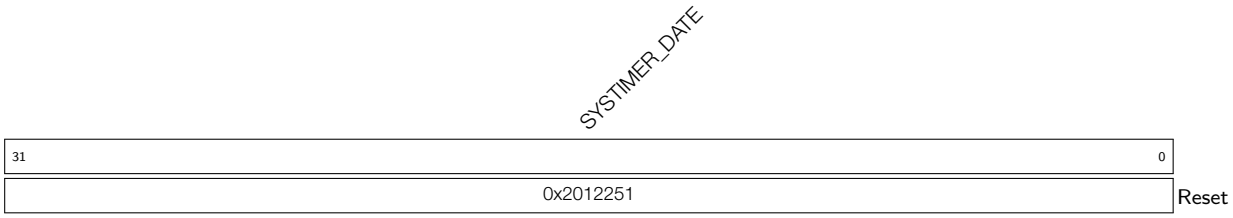


SYSTIMER_TARGET2_LO_RO 表示 COMP2 的实际报警值，低 32 位。(RO)

Register 11.35. SYSTIMER_REAL_TARGET2_HI_REG (0x0088)

SYSTIMER_TARGET2_HI_RO 表示 COMP2 的实际报警值，高 20 位。(RO)

Register 11.36. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE 版本控制寄存器。(R/W)

12 定时器组 (TIMG)

12.1 概述

通用定时器可用于准确设定时间间隔、在一定间隔后触发（周期或非周期的）中断或充当硬件时钟。如图 12-1 所示，ESP32-H2 包含两个定时器组，即定时器组 0 (TIMG*0*) 和定时器组 1 (TIMG*1*)。每个定时器组有一个通用定时器（下文用 T0 表示）和一个主系统看门狗定时器。通用定时器基于 16 位预分频器和 54 位可自动重新加载的可逆计数器。

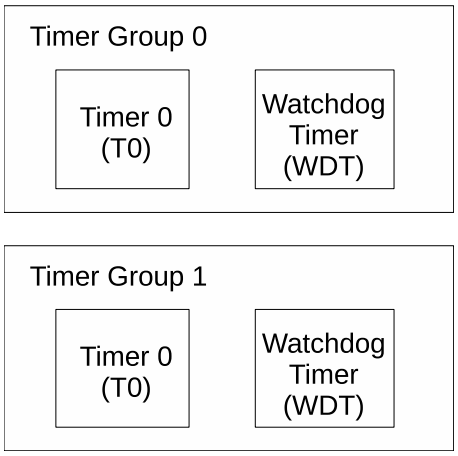


图 12-1. 定时器组概览

本章虽然包含主系统看门狗定时器的寄存器描述，但其功能描述请参阅章节 13 *看门狗定时器 (WDT)*。本章中“定时器”指代通用定时器。

12.2 主要特性

定时器具有如下功能：

- 54 位时基计数器，可配置成递增或递减
- 三个时钟源：XTAL_CLK 或 RC_FAST_CLK 或 PLL_F48M_CLK 时钟
- 16 位时钟预分频器，分频系数为 2 到 65536
- 可读取时基计数器的实时值
- 暂停和恢复时基计数器
- 可配置的报警产生机制
- 计数器值重新加载——报警时自动重新加载或软件控制的即时重新加载
- RTC 慢速时钟 RTC_SLOW_CLK 频率计算
- 电平触发中断
- 支持多个 ETM 任务和事件

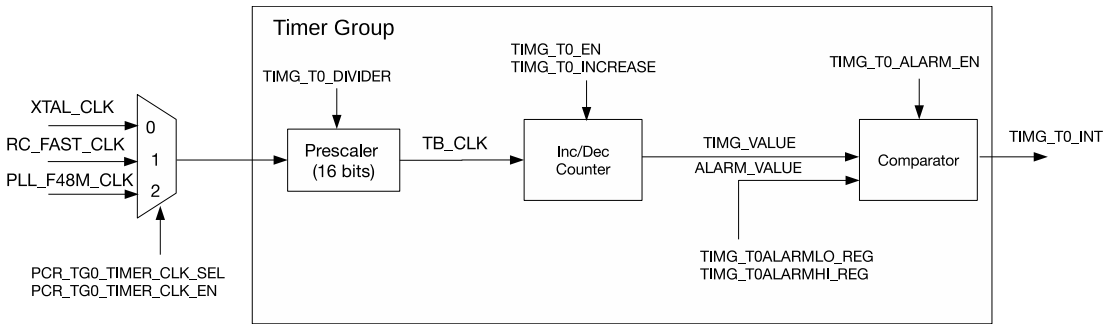


图 12-2. 定时器组架构

12.3 功能描述

图 12-2 为定时器组的 T0。T0 包含一个 16 位整数预分频器、一个时基计数器和一个用于产生警报的比较器。

12.3.1 16 位预分频器与时钟选择器

以 TIMG*0* 中的 T0 定时器为例：

- 该定时器可通过配置寄存器 PCR_TIMERGROUP0_TIMER_CLK_CONF_REG 的 PCR_TG0_TIMER_CLK_SEL 字段来选择时钟源。该字段为 0 时，选择 XTAL_CLK 为时钟源；为 1 时，选择 RC_FAST_CLK 为时钟源；为 2 时，选择 PLL_F48M_CLK 为时钟源。
- 要开启所选时钟，需配置 PCR_TIMERGROUP0_TIMER_CLK_CONF_REG 寄存器的 PCR_TG0_TIMER_CLK_EN 字段为 1，要关闭时钟需清零该字段。所选时钟经 16 位预分频器分频，产生时基计数器使用的时基计数器时钟 (TB_CLK)。16 位预分频器的分频系数可通过 TIMG*0*_T0_DIVIDER 字段配置。

TIMG*0*_T0_DIVIDER 字段可配置为 0 ~ 65535 之间的任意值，分频系数的范围为 2 ~ 65536。具体来说：

- 当 TIMG*0*_T0_DIVIDER 字段置 0 时：分频系数为 65536。
- 当 TIMG*0*_T0_DIVIDER 字段置 1 时：分频系数为 2。
- 当 TIMG*0*_T0_DIVIDER 字段置 2 时：分频系数仍为 2。
- 当 TIMG*0*_T0_DIVIDER 字段置 3 ~ 65535 时：分频系数为 3 ~ 65535。

要更改 16 位预分频器，需先重新配置 TIMG*0*_T0_DIVIDER 字段，再将 TIMG*0*_T0_DIVCNT_RST 置 1，同时需要关闭定时器（即 TIMG*0*_T0_EN 必须清零），否则会造成不可预知的结果。

12.3.2 54 位时基计数器

54 位时基计数器基于 TB_CLK，可通过 TIMG*n*_T0_INCREASE 字段配置为递增或递减。时基计数器可通过置位或清零 TIMG*n*_T0_EN 字段使能或关闭。使能时，时基计数器的值会在每个 TB_CLK 周期递增或递减。关闭时，时基计数器暂停计数。注意，无论TIMG*n*_T0_EN 是否置位，TIMG*n*_T0_INCREASE 字段都可以更改，时基计数器可立即改变计数方向。

时基计数器 54 位定时器的当前值必须被锁入两个寄存器，才能被 CPU 读取（因为 CPU 为 32 位）。向 TIMG*n*_T0UPDATE_REG 写入任意值时，54 位定时器的值开始被锁入寄存器 TIMG*n*_T0LO_REG 和 TIMG*n*_T0HI_REG，两个寄存器分别锁存低 32 位和高 22 位。当 TIMG0*n*_T0UPDATE_REG 被硬件清零，表明

锁存操作已经完成，可以从这两个寄存器中读取当前计数值。在 TIMG*n*_T0UPDATE_REG 被写入新值之前，保持寄存器 TIMG*n*_T0LO_REG 和 TIMG*n*_T0HI_REG 的值不变，以供 32 位的 CPU 读值。

12.3.3 报警产生

定时器可配置为在当前值与报警值相同时触发报警。报警会产生中断，同时（可选择）让定时器的当前值自动重新加载（详见第 12.3.4节）。

54 位报警值可在 TIMG*n*_T0ALARMLO_REG 和 TIMG*n*_T0ALARMHI_REG 配置，两者分别代表报警值的低 32 位和高 22 位。但是，只有置位 TIMG*n*_T0_ALARM_EN 字段使能报警功能后，配置的报警值才会生效。为解决报警使能“过晚”（即报警使能时，定时器的值已过报警值），出现以下情况时硬件会立即触发报警：

- 可逆计数器向上计数时，定时器的当前值高于报警值（在一定范围内）
- 可逆计数器向下计数时，定时器的当前值低于报警值（在一定范围内）

表 12-1 和表 12-2 说明了定时器当前值、报警值与报警触发的关系。假设定时器当前值和报警值如下：

- TIMG_VALUE = {TIMG*n*_T0HI_REG, TIMG*n*_T0LO_REG}
- ALARM_VALUE = {TIMG*n*_T0ALARMHI_REG, TIMG*n*_T0ALARMLO_REG}

表 12-1. 可逆计数器向上计数时的报警场景

场景	范围	报警
1	ALARM_VALUE − TIMG_VALUE $> 2^{53}$	触发
2	$0 <$ ALARM_VALUE − TIMG_VALUE $\leq 2^{53}$	可逆计数器向上计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
3	$0 \leq$ TIMG_VALUE − ALARM_VALUE $< 2^{53}$	触发
4	TIMG_VALUE − ALARM_VALUE $\geq 2^{53}$	可逆计数器向上计数达到最大值时，重新开始从 0 向上计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

表 12-2. 可逆计数器向下计数时的报警场景

场景	范围	报警
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	触发
6	$0 <$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	可逆计数器向下计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
7	$0 \leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	触发
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	可逆计数器向下计数达到最小值时，重新开始从最大值向下计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

报警时，TIMG*n*_T0_ALARM_EN 字段自动清零，在下次置位 TIMG*n*_T0_ALARM_EN 前不会再次报警。

12.3.4 定时器重新加载

定时器重新加载是指将定时器的低 32 位和高 22 位分别更新为 TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI 字段存储的重新加载值。但是，把重新加载值写入 TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI 字段不会改变

定时器的当前值。写入的重新加载值会被定时器忽视，直到重新加载事件被触发。重新加载事件可由软件即时重新加载或报警时自动重新加载触发。

CPU 在寄存器 TIMG*n*_T0LOAD_REG 写任意值会触发软件即时重新加载，定时器的当前值会立即改变。若此时 TIMG*n*_T0_EN 是置位状态，定时器会继续从新数值开始递增或递减计数。若此时 TIMG*n*_T0_ALARM_EN置位，仍会根据表 12-1 或表 12-2 中的关系在对应时刻产生报警。若 TIMG*n*_T0_EN 是清零状态，定时器将保持当前值，直至计数重新使能。

报警时，自动重新加载功能可让定时器在报警时重新加载，从重新加载值开始继续递增或递减计数。该功能通常用于周期性报警时重置定时器的值。要使能报警时自动重新加载，需将 TIMG*n*_T0_AUTORELOAD 字段置 1。如未使能该功能，报警后定时器的值会在过报警值后继续递增或递减。

12.3.5 事件任务矩阵功能

在 ESP32-H2 中，定时器组支持 ETM 功能，即可以通过任意外设的 ETM 事件触发定时器组的 ETM 任务，或者通过定时器组的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与定时器组相关的 ETM 任务和 ETM 事件。

定时器组可接收的 ETM 任务有：

- TIMER*n*_TASK_CNT_START_TIMER0 (*n*:0-1)：触发时启用时基计数器。
- TIMER*n*_TASK_CNT_STOP_TIMER0 (*n*:0-1)：触发时关闭时基计数器。

> **说明：**
> 以上两个 ETM 任务与 APB 配置 TIMG*n*_T0_EN 实现的功能相同。当这些操作同时发生时，各操作优先级由高到低如下：
>
> 1. TIMER*n*_TASK_CNT_START_TIMER0：触发时启用时基计数器
> 2. TIMER*n*_TASK_CNT_STOP_TIMER0：触发时关闭时基计数器
> 3. APB 方式配置 TIMG*n*_T0_EN：启用或关闭时基计数器

- TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1)：触发时启用报警产生。

> **说明：**
> 通过 APB 方式配置 TIMG*n*_ALARM_EN 以及硬件事件触发也可以启用报警产生功能。当这些操作同时发生时，各个操作的优先级从高到低依次为：
>
> 1. TIMER*n*_TASK_ALARM_START_TIMER0：触发时启用报警产生
> 2. 警报事件：触发时关闭警报生成
> 3. APB 方式配置 TIMG*n*_T0_ALARM_EN：启用/关闭报警产生

- TIMER*n*_TASK_CNT_CAP_TIMER0 (*n*:0-1)：触发时会将当前计数器的值更新至 TIMG*n*_T0LO_REG 和 TIMG*n*_T0HI_REG 寄存器。
- TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1)：触发时会用存储在 TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI 中的重新加载值覆盖当前计数器值。

定时器组可产生的 ETM 事件有：

- TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1)：表示 TIMG*n* 的 T0 定时器中断事件。

只有当 TIMG*n*_ETM_EN 设置为 1 时，ETM 任务和事件才会生效。

在具体应用中，定时器组的 ETM 事件可以用来触发定时器组的 ETM 任务。例如，TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) 事件可以触发 TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) 任务，从而实现周期性报警。具体配置步骤请参考 12.4.4 通过 ETM 设置定时器用于周期性报警。

12.3.6 RTC 慢速时钟 (RTC_SLOW_CLK) 频率计算

定时器组可以以 XTAL_CLK 为基准时钟，计算 RTC 慢速时钟的三个慢速时钟源 RC_SLOW_CLK、RC_FAST_DIV_CLK 和 XTAL32K_CLK 的实际频率。仅 TIMG*0* 支持该功能，计算方式如下：

1. 通过周期性或单次计算的方式启动频率计算模块（详见章节 12.4.5）；
2. 在接收到计算开始的信号后，两个分别工作在 XTAL_CLK 以及 RTC_SLOW_CLK 的计数器同时开始计数，当 RTC_SLOW_CLK 的计数器达到设定的计算周期 C0 时，同时停止两个计数器；
3. 通过 XTAL_CLK 的计数器值 C1 即可计算 RTC_SLOW_CLK 的时钟频率：$f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

12.3.7 中断

每个定时器都有一根连接至 CPU 的中断线。因此，每个定时器组有两根中断线。定时器每次产生的电平中断必须由 CPU 清除。

电平中断在报警后（或看门狗定时器阶段超时）触发。报警（或阶段超时）后，电平中断会一直被拉高，直至手动清除中断。要使能定时器的中断，TIMG*n*_T0_INT_ENA 需置 1。

每个定时器组的中断受一组寄存器控制。每个定时器在下列寄存器中都有对应的位：

- TIMG*n*_T0_INT_RAW：报警时置 1。该位在写值到对应的 TIMG*n*_T0_INT_CLR 位后才会被清零。
- TIMG*n*_WDT_INT_RAW：阶段超时时置 1。该位在写值到对应的 TIMG*n*_WDT_INT_CLR 位后才会被清零。
- TIMG*n*_T0_INT_ST：反映每个定时器中断的状态，通过用 TIMG*n*_T0_INT_ENA 屏蔽 TIMG*n*_T0_INT_RAW 位来生成。
- TIMG*n*_WDT_INT_ST：反映每个看门狗定时器中断的状态，通过用 TIMG*n*_WDT_INT_ENA 屏蔽 TIMG*n*_WDT_INT_RAW 位来生成。
- TIMG*n*_T0_INT_ENA：用于使能或屏蔽组内定时器的中断状态位。
- TIMG*n*_WDT_INT_ENA：用于使能或屏蔽组内看门狗定时器的中断状态位。
- TIMG*n*_T0_INT_CLR：置 1 此位清除定时器中断，定时器在 TIMG*n*_T0_INT_RAW 和 TIMG*n*_T0_INT_ST 的对应位会清零。注意，下一个中断产生前，必须清除定时器中断。
- TIMG*n*_WDT_INT_CLR：置 1 此位清除定时器中断，看门狗定时器在 TIMG*n*_WDT_INT_RAW 和 TIMG*n*_WDT_INT_ST 的对应位会清零。注意，下一个中断产生前，必须清除看门狗定时器中断。

12.4 配置与使用

12.4.1 定时器用作简单时钟

1. 配置时基计数器。
    - 置位或清除 PCR_TG0_TIMER_CLK_SEL 字段选择时钟源。
    - 置位 TIMG*n*_T0_DIVIDER 配置 16 位预分频器。

- 置位或清除 TIMG*n*_T0_INCREASE 配置定时器方向。
- 在 TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI 上写初始值设置定时器的初始值，然后在 TIMG*n*_T0LOAD_REG 上写任意值将初始值重新加载进定时器。

2. 置位 TIMG*n*_T0_EN 开启定时器。
3. 获得定时器的当前值。
   - 在 TIMG*n*_T0UPDATE_REG 上写任意值锁存定时器的当前值。
   - 等待硬件将 TIMG*n*_T0UPDATE_REG 清 0。
   - 从 TIMG*n*_T0LO_REG 和 TIMG*n*_T0HI_REG 读取锁存的定时器值。

12.4.2 定时器用于单次报警

1. 按照第 12.4.1 节的第 1 步配置时基计数器。
2. 配置报警。
   - 置位 TIMG*n*_T0ALARMLO_REG 和 TIMG*n*_T0ALARMHI_REG 配置报警值。
   - 置位 TIMG*n*_T0_INT_ENA 使能中断。
3. 清零 TIMG*n*_T0_AUTORELOAD 关闭自动重新加载。
4. 置位 TIMG*n*_T0_ALARM_EN 开启报警。
5. 处理报警中断。
   - 置位定时器在 TIMG*n*_T0_INT_CLR 的对应位清除中断。
   - 清零 TIMG*n*_T0_EN 关闭定时器。

12.4.3 通过 APB 设置定时器用于周期性报警

1. 按照第 12.4.1 节的第 1 步配置时基计数器。
2. 按照第 12.4.2 节的第 2 步配置报警。
3. 置位 TIMG*n*_T0_AUTORELOAD 使能自动重新加载，将重新加载值写入 TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI。
4. 置位 TIMG*n*_T0_ALARM_EN 开启报警。
5. 处理报警中断（每次报警时重复）。
   - 置位定时器在 TIMG*n*_T0_INT_CLR 的对应位清除中断。
   - 如下一次报警需要新的报警值和重新加载值（即每次都有不同的报警间隔），则应根据需要重新配置 TIMG*n*_T0ALARMLO_REG、TIMG*n*_T0ALARMHI_REG、TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI 。否则，上述寄存器应保持不变。
   - 置位 TIMG*n*_T0_ALARM_EN 重新使能报警。
6. （最后一次报警时）关闭定时器。
   - 置位定时器在 TIMG*n*_T0_INT_CLR 的对应位清除中断。
   - 清零 TIMG*n*_T0_EN 关闭定时器。

12.4.4 通过 ETM 设置定时器用于周期性报警

1. 使能 ETM 模块的时钟
2. 将 ETM 事件映射到相应的 ETM 任务（使用 ETM 事件触发对应的 ETM 任务）
   - 如果 TIMG*n*_T0_AUTORELOAD 置 1，通过一个 ETM 通道映射 TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) 事件到 TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) 任务。
   - 如果 TIMG*n*_T0_AUTORELOAD 置 0，除了映射 TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) 事件到 TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) 任务外，还需要用另一个 ETM 通道映射 TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) 事件到 TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1) 任务。
3. 选择使能一个或两个 ETM 通道。
4. 将 TIMG*n*_ETM_EN 置 1 从而使能定时器组的 ETM 事件和任务。
5. 按照第 12.4.1 节的第 1 步配置时基计数器。
6. 按照第 12.4.2 节的第 2 步配置报警。
7. 通过 TIMG*n*_T0_LOAD_LO 和 TIMG*n*_T0_LOAD_HI 配置重新加载值。
8. 处理 TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1)
   - 当报警产生时，TIMER*n*_EVT_CNT_CMP_TIMER0 (*n*:0-1) 事件也会产生，此时报警器关闭报警产生。
   - 如果 TIMG*n*_T0_AUTORELOAD 为 1，重新加载值会覆盖当前计数器的值。TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) 会重新使能报警产生。
   - 如果 TIMG*n*_T0_AUTORELOAD 为 0，由于 TIMER*n*_TASK_CNT_RELOAD_TIMER0 (*n*:0-1) 任务，重新加载值会覆盖当前计数器的值。TIMER*n*_TASK_ALARM_START_TIMER0 (*n*:0-1) 任务会重新使能报警产生。
9. （最后一次报警时）关闭定时器。
   - 关闭用于映射定时器组事件和任务的 ETM 通道。
   - 设置 TIMG*n*_ETM_EN 为 0。
   - 置位定时器在 TIMG*n*_T0_INT_CLR 的对应位清除中断。
   - 清零 TIMG*n*_T0_EN 关闭定时器。

12.4.5 RTC_SLOW_CLK 频率计算

1. 单次计算
   - 设置 TIMG*0*_RTC_CALI_CLK_SEL 选择需要计算频率的时钟（RTC_SLOW_CLK 的时钟源），设置 TIMG*0*_RTC_CALI_MAX 配置频率计算时间。
   - 清空 TIMG*0*_RTC_CALI_START_CYCLING 选择单次校准模式，然后配置 TIMG*0*_RTC_CALI_START 开启两个计数器。
   - 等待 TIMG*0*_RTC_CALI_RDY 的值变为 1，读取 TIMG*0*_RTC_CALI_VALUE 获取 XTAL_CLK 计数器值，根据章节 12.3.6 的公式计算 RTC_SLOW_CLK 频率。
2. 周期性计算

- 设置 TIMG*0*_RTC_CALI_CLK_SEL 选择需要计算频率的时钟（RTC_SLOW_CLK 的时钟源），设置 TIMG*0*_RTC_CALI_MAX 配置频率计算时间。
- 使能 TIMG*0*_RTC_CALI_START_CYCLING，硬件将不间断进行频率计算过程。
- 只要 TIMG*0*_RTC_CALI_CYCLING_DATA_VLD 为 1，即表示 TIMG*0*_RTC_CALI_VALUE 有效。

3. 超时
   如果 RTC_SLOW_CLK 的计数器没有在 TIMG*0*_RTC_CALI_TIMEOUT_RST_CNT 的 XTAL_CLK 计数器内完成计数，将置位 TIMG*0*_RTC_CALI_TIMEOUT 标记计算超时。

12.5 寄存器列表

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

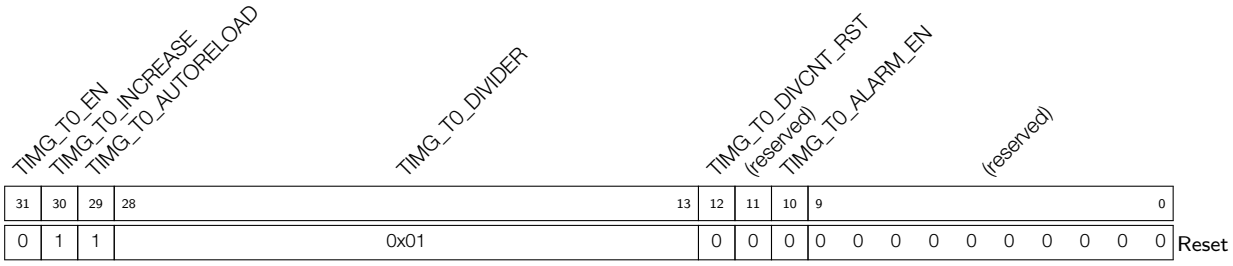
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

Name	Description	Address	Access
定时器 0 控制和配置寄存器			
TIMG_T0CONFIG_REG	定时器 0 配置寄存器	0x0000	varies
TIMG_T0LO_REG	定时器 0 的当前值，低 32 位	0x0004	RO
TIMG_T0HI_REG	定时器 0 的当前值，高 22 位	0x0008	RO
TIMG_T0UPDATE_REG	写值将当前定时器的值复制到 TIMGn_T0LO_REG 或 TIMGn_T0HI_REG	0x000C	R/W/SC
TIMG_T0ALARMLO_REG	定时器 0 的报警值，低 32 位	0x0010	R/W
TIMG_T0ALARMHI_REG	定时器 0 的报警值，高位	0x0014	R/W
TIMG_T0LOADLO_REG	定时器 0 的重新加载值，低 32 位	0x0018	R/W
TIMG_T0LOADHI_REG	定时器 0 的重新加载值，高 22 位	0x001C	R/W
TIMG_T0LOAD_REG	写值从 TIMG_T0LOADLO_REG 或 TIMG_T0LOADHI_REG 上加载定时器	0x0020	WT
看门狗定时器控制和配置寄存器			
TIMG_WDTCONFIG0_REG	看门狗定时器配置寄存器	0x0048	varies
TIMG_WDTCONFIG1_REG	看门狗定时器预分频器寄存器	0x004C	varies
TIMG_WDTCONFIG2_REG	看门狗定时器阶段 0 超时值	0x0050	R/W
TIMG_WDTCONFIG3_REG	看门狗定时器阶段 1 超时值	0x0054	R/W
TIMG_WDTCONFIG4_REG	看门狗定时器阶段 2 超时值	0x0058	R/W
TIMG_WDTCONFIG5_REG	看门狗定时器阶段 3 超时值	0x005C	R/W
TIMG_WDTFEED_REG	写值喂看门狗定时器	0x0060	WT
TIMG_WDTWPROTECT_REG	看门狗写保护寄存器	0x0064	R/W
RTC 频率计算控制和配置寄存器			
TIMG_RTCCALICFG_REG	RTC 频率计算配置寄存器 0	0x0068	varies
TIMG_RTCCALICFG1_REG	RTC 频率计算配置寄存器 1	0x006C	RO
TIMG_RTCCALICFG2_REG	RTC 频率计算配置寄存器 2	0x0080	varies
中断寄存器			
TIMG_INT_ENA_TIMERS_REG	中断使能位	0x0070	R/W
TIMG_INT_RAW_TIMERS_REG	原始中断状态	0x0074	R/SS/WTC
TIMG_INT_ST_TIMERS_REG	屏蔽中断状态	0x0078	RO
TIMG_INT_CLR_TIMERS_REG	中断清除位	0x007C	WT
版本寄存器			
TIMG_NTIMERS_DATE_REG	版本控制寄存器	0x00F8	R/W
时钟配置寄存器			
TIMG_REGCLK_REG	定时器组时钟门控寄存器	0x00FC	R/W

12.6 寄存器

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

Register 12.1. TIMG_T0CONFIG_REG (0x0000)



TIMG_T0_ALARM_EN 配置是否使能定时器 0 的报警功能。报警时，此位自动清零。

0：禁用

1：使能

(R/W/SC)

TIMG_T0_DIVCNT_RST 配置是否复位定时器 0 时钟分频器的计数器。

0：无效

1：复位

(WT)

TIMG_T0_DIVIDER 表示定时器 0 时钟 (T0_clk) 的预分频器值。(R/W)

TIMG_T0_AUTORELOAD 配置是否使能定时器 0 报警时自动重新加载使能。

0：无效

1：使能

(R/W)

TIMG_T0_INCREASE 配置定时器 0 的时基计数器的计数方向。

0：递减

1：递增

(R/W)

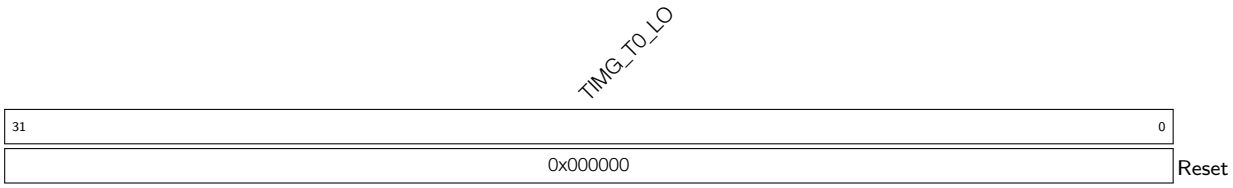
TIMG_T0_EN 配置是否使能定时器 0 的时基计数器。

0：禁用

1：使能

(R/W/SS/SC)

Register 12.2. TIMG_T0LO_REG (0x0004)

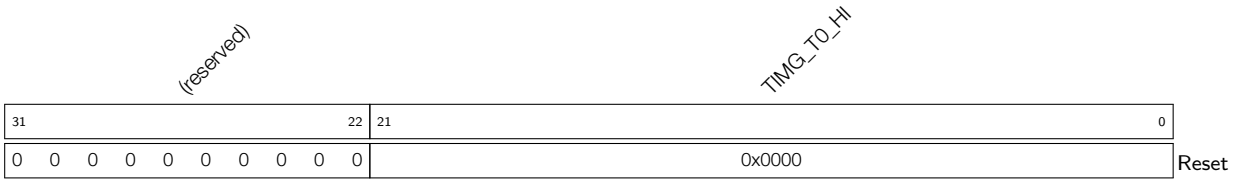


TIMG_T0_LO 表示定时器 0 时基计数器的低 32 位的值。仅在 TIMG_T0UPDATE_REG 写值后有效。

单位：T0_clk.

(RO)

Register 12.3. TIMG_T0HI_REG (0x0008)

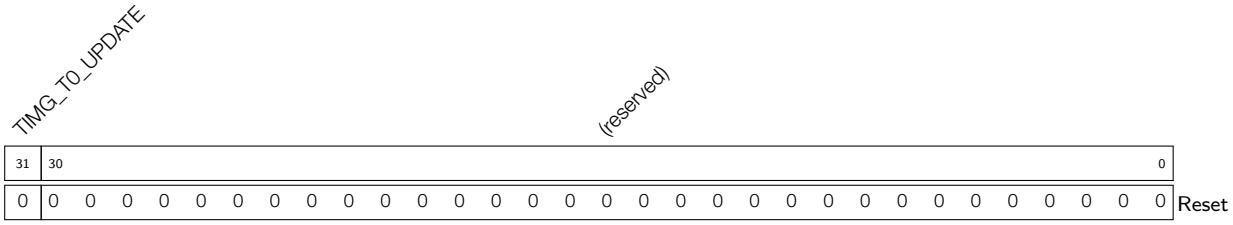


TIMG_T0_HI 表示定时器 0 时基计数器的高 22 位的值。仅在 TIMG_T0UPDATE_REG 写值后有效。

单位：T0_clk.

(RO)

Register 12.4. TIMG_T0UPDATE_REG (0x000C)



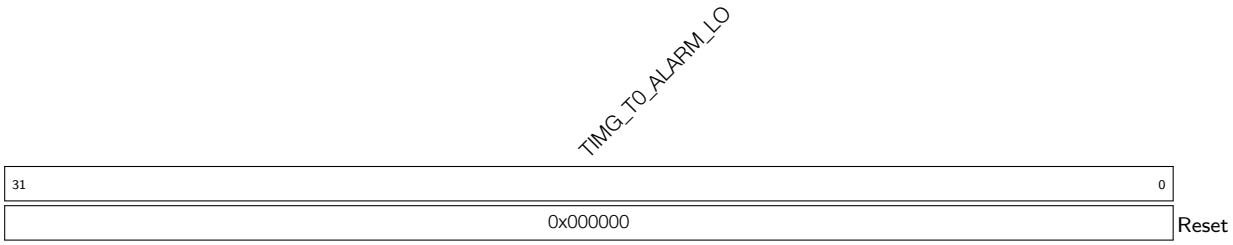
TIMG_T0_UPDATE 配置锁存计数器的值。

0：锁存

1：锁存

(R/W/SC)

Register 12.5. TIMG_T0ALARMLO_REG (0x0010)

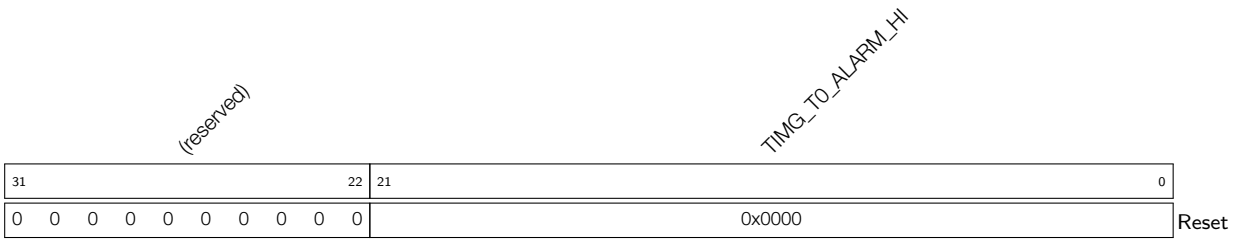


TIMG_T0_ALARM_LO 配置定时器 0 时基计数器触发报警值的低 32 位。仅在 TIMG_T0_ALARM_EN 为 1 时有效。

单位：T0_clk

(R/W)

Register 12.6. TIMG_T0ALARMHI_REG (0x0014)

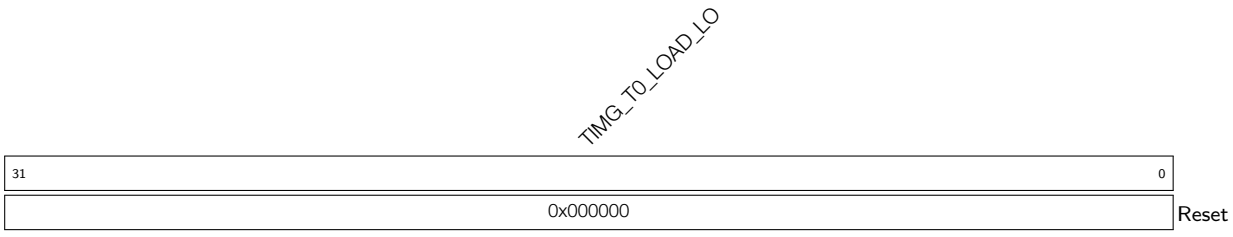


TIMG_T0_ALARM_HI 配置定时器 0 时基计数器触发报警值的高 22 位。仅在 TIMG_T0_ALARM_EN 为 1 时有效。

单位：T0_clk.

(R/W)

Register 12.7. TIMG_T0LOADLO_REG (0x0018)

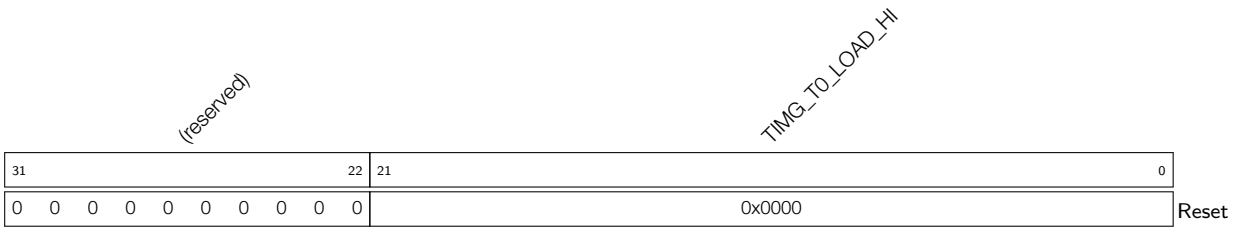


TIMG_T0_LOAD_LO 配置定时器 0 时基计数器重新加载的低 32 位值。

单位：T0_clk.

(R/W)

Register 12.8. TIMG_T0LOADHI_REG (0x001C)

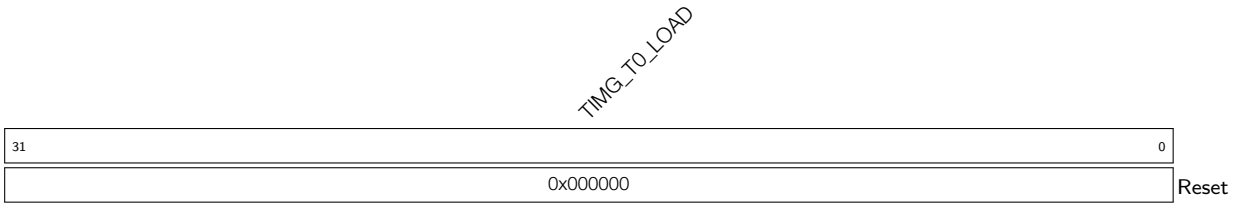


TIMG_T0_LOAD_HI 配置定时器 0 时基计数器重新加载的高 22 位值。

单位：T0_clk.

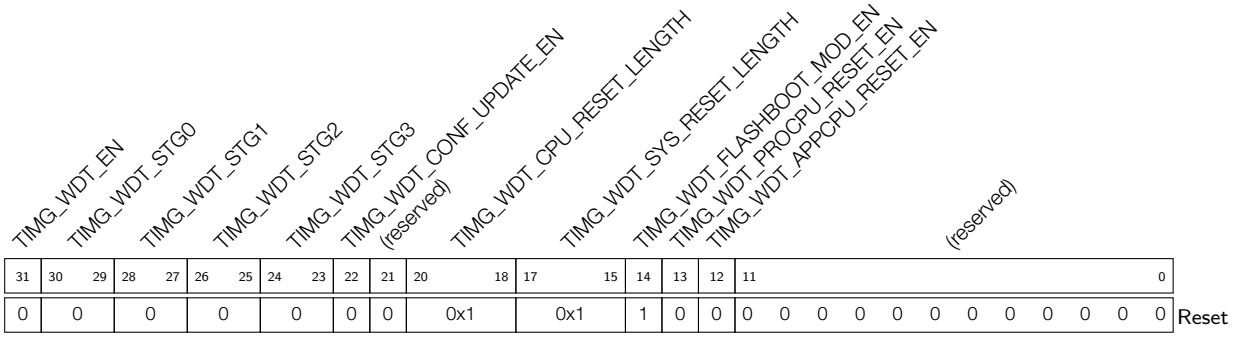
(R/W)

Register 12.9. TIMG_T0LOAD_REG (0x0020)



TIMG_T0_LOAD 写任意值触发定时器 0 时基计数器重新加载。(WT)

Register 12.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_APPCPU_RESET_EN 配置是否屏蔽 MWDT 产生的 CPU 复位。仅在禁用写保护时有效。

0：屏蔽

1：不屏蔽

(R/W)

TIMG_WDT_PROCPU_RESET_EN 配置是否屏蔽 MWDT 产生的 CPU 复位。仅在禁用写保护时有效。

0：屏蔽

1：不屏蔽

(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN 配置是否使能 flash 启动保护。

0：禁用

1：使能

(R/W)

TIMG_WDT_SYS_RESET_LENGTH 配置系统复位信号长度。仅在禁用写保护时有效。

单位：mwdt_clk.

0: 8
1: 16
2: 24
3: 32
4: 40
5: 64
6: 128
7: 256

(R/W)

TIMG_WDT_CPU_RESET_LENGTH 配置 CPU 复位信号长度。仅在禁用写保护时有效。

单位：mwdt_clk.

0: 8
1: 16
2: 24
3: 32
4: 40
5: 64
6: 128
7: 256

(R/W)

见下页...

Register 12.10. TIMG_WDTCONFIG0_REG (0x0048)

接上页...

TIMG_WDT_CONF_UPDATE_EN 配置以更新看门狗定时器的配置寄存器。

0：无效

1：更新

(WT)

TIMG_WDT_STG3 配置阶段 3 的超时动作。具体可参考 TIMG_WDT_STG0。仅在禁用写保护时有效。(R/W)

TIMG_WDT_STG2 配置阶段 2 的超时动作。具体可参考 TIMG_WDT_STG0。仅在禁用写保护时有效。(R/W)

TIMG_WDT_STG1 配置阶段 1 的超时动作。具体可参考 TIMG_WDT_STG0。仅在禁用写保护时有效。(R/W)

TIMG_WDT_STG0 配置阶段 0 的超时动作。仅在禁用写保护时有效。

0：无效

1：中断

2：复位 CPU

3：复位系统

(R/W)

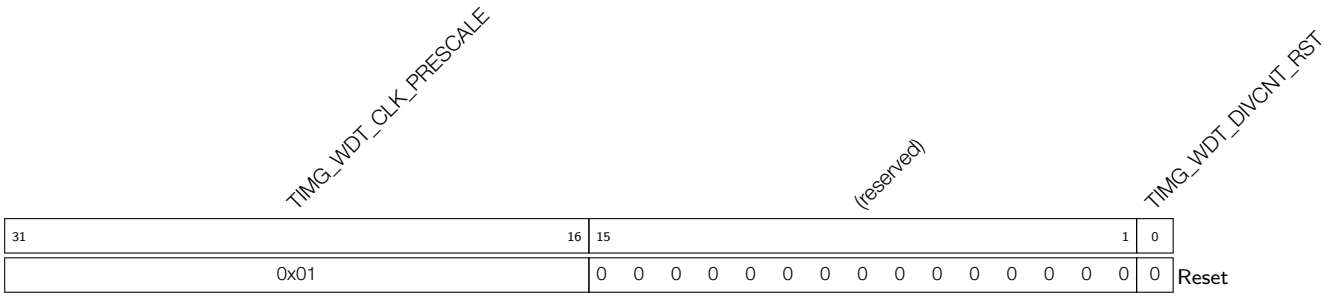
TIMG_WDT_EN 配置是否使能 MWDT。仅在禁用写保护时有效。

0：禁用

1：使能

(R/W)

Register 12.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST 配置是否复位看门狗定时器时钟分频器的计数器。

0：无效

1：复位

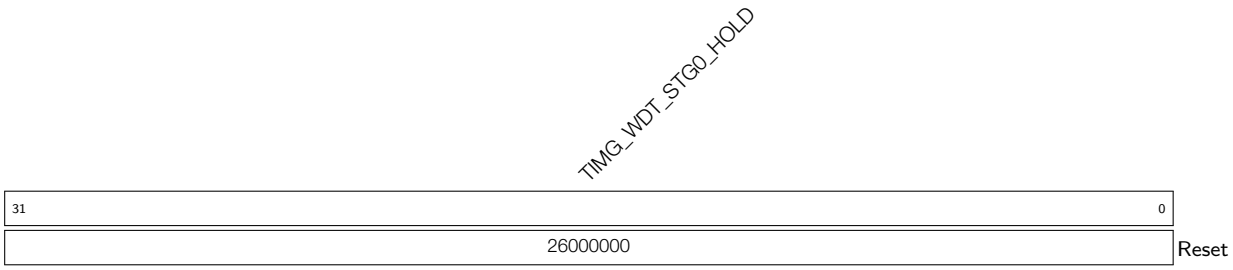
(WT)

TIMG_WDT_CLK_PRESCALE 配置 MWDT 时钟预分频器值。仅在禁用写保护时有效。

MWDT 时钟周期 = MWDT 时钟源周期 * TIMG_WDT_CLK_PRESCALE.

(R/W)

Register 12.12. TIMG_WDTCONFIG2_REG (0x0050)

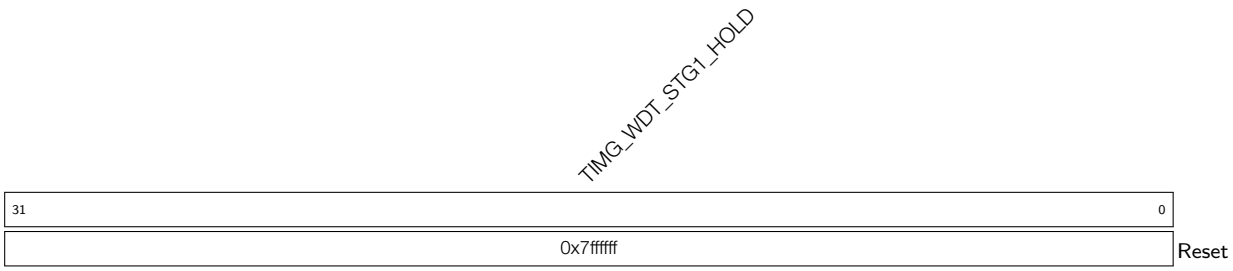


TIMG_WDT_STG0_HOLD 配置阶段 0 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk.

(R/W)

Register 12.13. TIMG_WDTCONFIG3_REG (0x0054)

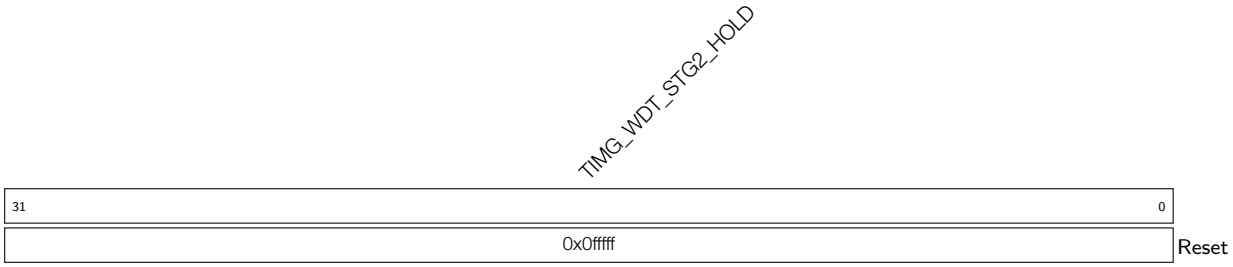


TIMG_WDT_STG1_HOLD 配置阶段 1 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk.

(R/W)

Register 12.14. TIMG_WDTCONFIG4_REG (0x0058)

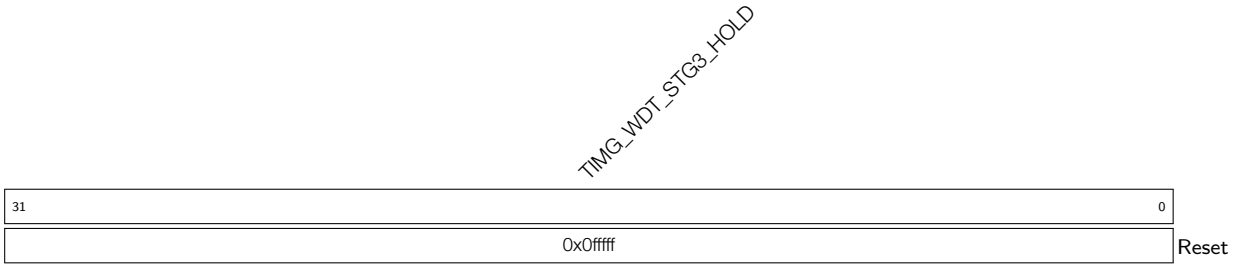


TIMG_WDT_STG2_HOLD 配置阶段 2 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk.

(R/W)

Register 12.15. TIMG_WDTCONFIG5_REG (0x005C)

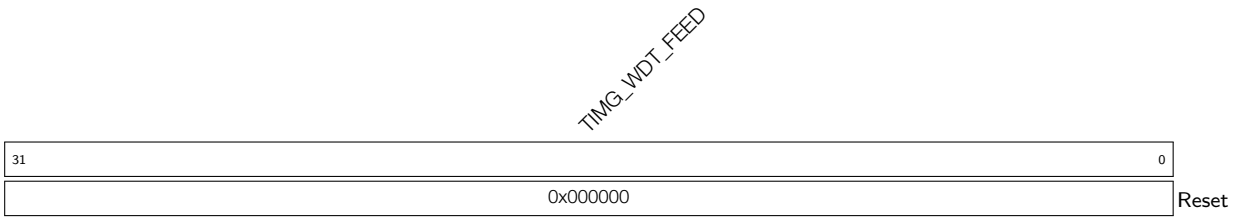


TIMG_WDT_STG3_HOLD 配置阶段 3 超时时间。仅在禁用写保护时有效。

单位：mwdt_clk.

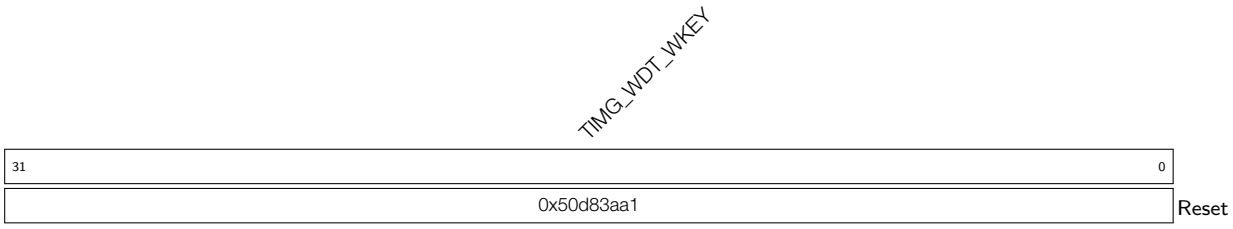
(R/W)

Register 12.16. TIMG_WDTFEED_REG (0x0060)



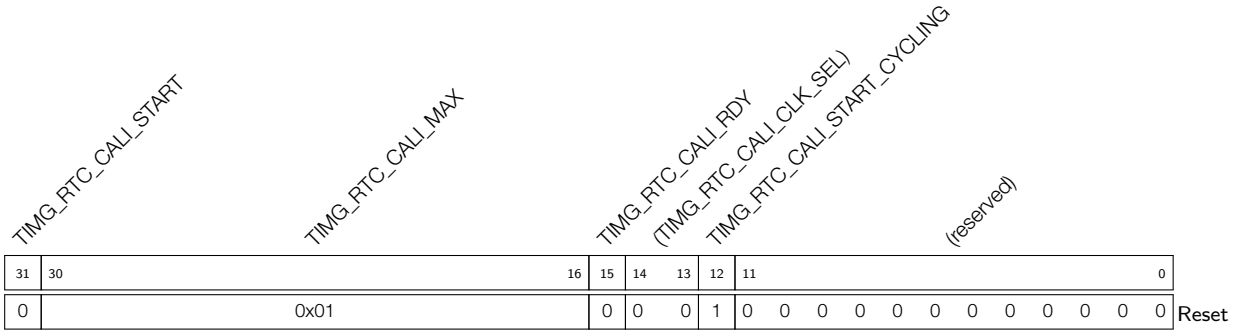
TIMG_WDT_FEED 写任意值喂 MWDT。仅在禁用写保护时有效。(WT)

Register 12.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY 配置一个与复位值不同的值以使能写保护。(R/W)

Register 12.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING 配置频率计算模式。

0：单次频率计算模式；

1：周期性频率计算模式。

(R/W)

TIMG_RTC_CALI_CLK_SEL 选择待校准时钟。

0：RTC_SLOW_CLK

1：RC_FAST_DIV_CLK

2：XTAL32K_CLK

(R/W)

TIMG_RTC_CALI_RDY 表示单次频率计算是否完成。

0：未完成

1：完成

(RO)

TIMG_RTC_CALI_MAX 配置计算 RTC 慢速时钟 SLOW_CLK 频率的时间。

单位：XTAL_CLK.

(R/W)

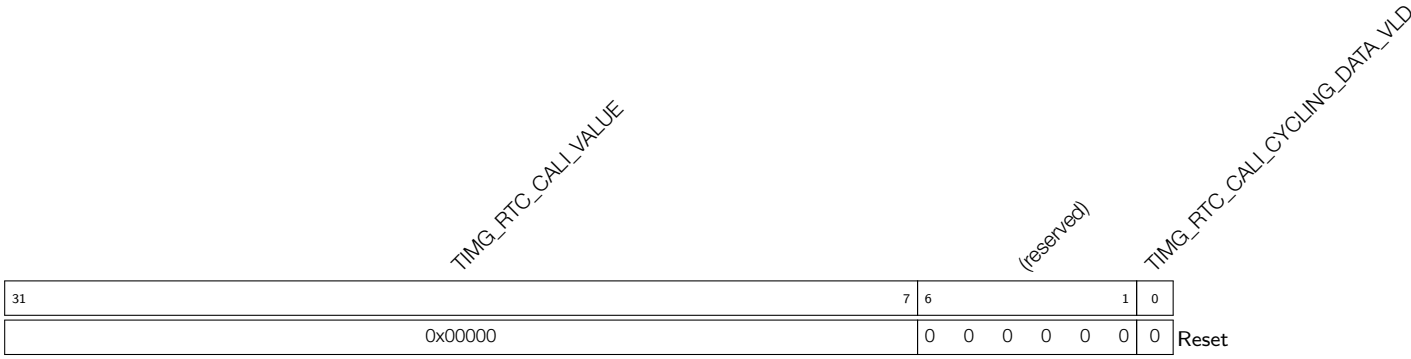
TIMG_RTC_CALI_START 配置是否使能单次频率计算。

0：禁用

1：使能

(R/W)

Register 12.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD 表示周期性频率计算是否完成。

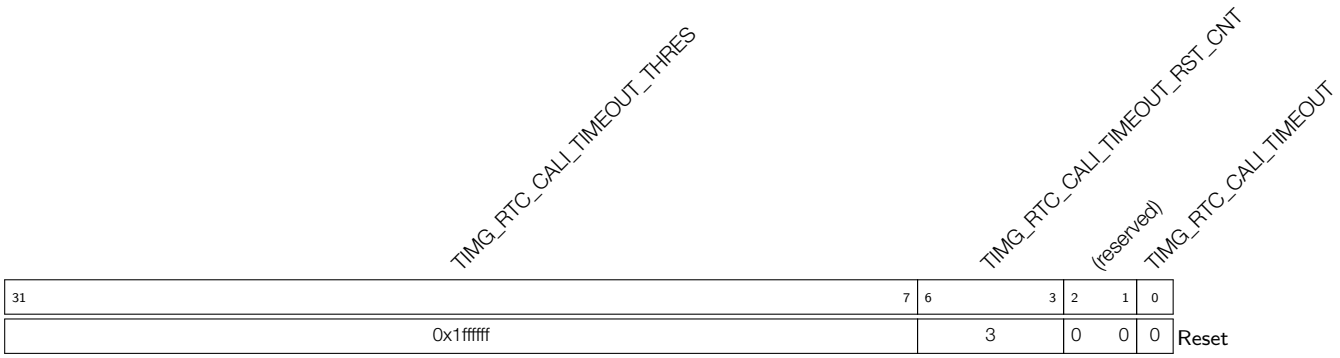
0：未完成

1：完成

(RO)

TIMG_RTC_CALI_VALUE 表示单次或周期性频率计算完成时 XTAL_CLK 计数的值。用于计算 RTC 慢时钟的频率。(RO)

Register 12.20. TIMG_RTCCALICFG2_REG (0x0080)



TIMG_RTC_CALI_TIMEOUT 表示频率计算是否超时。

0：未超时

1：超时

(RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT 配置频率计算超时复位的周期。

单位：XTAL_CLK.

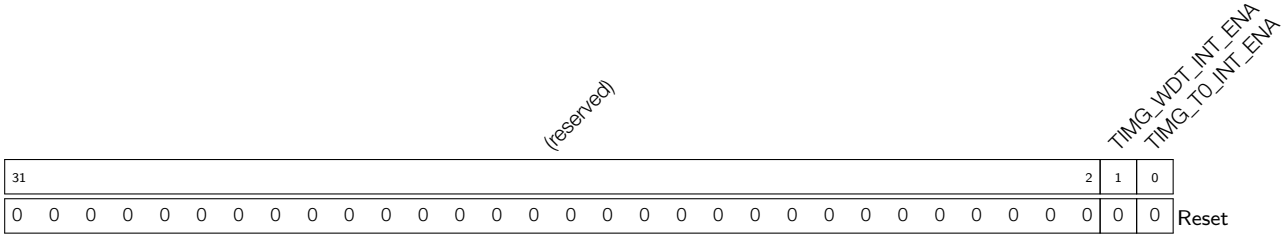
(R/W)

TIMG_RTC_CALI_TIMEOUT_THRES 配置 RTC 频率计算定时器的阈值。频率计算定时器的值超过此值时触发超时。

单位：XTAL_CLK.

(R/W)

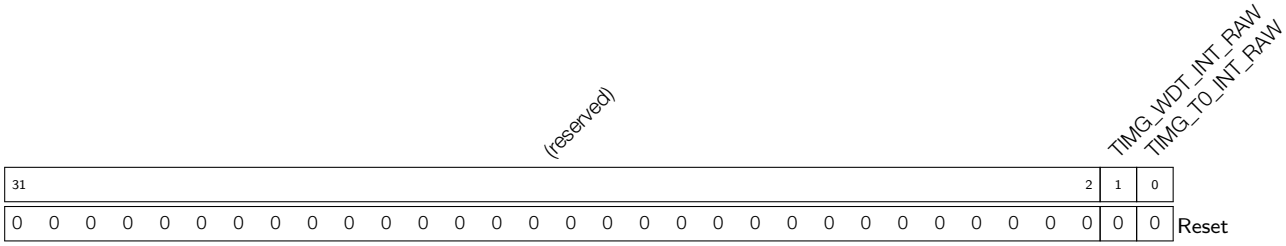
Register 12.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_T0_INT_ENA 写 1 使能 TIMG_T0_INT 中断。(R/W)

TIMG_WDT_INT_ENA 写 1 使能 TIMG_WDT_INT 中断。(R/W)

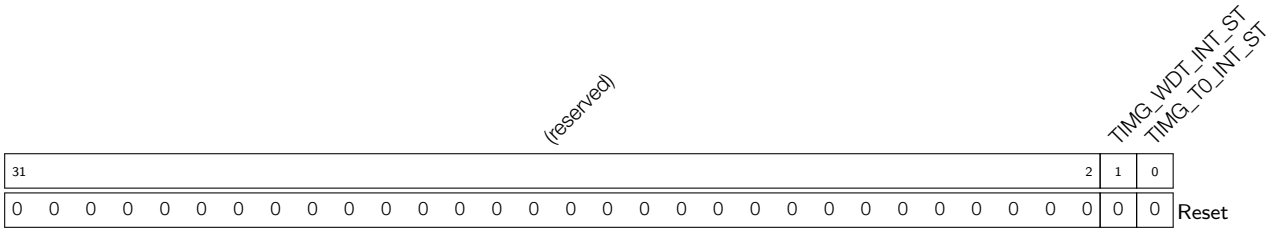
Register 12.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_T0_INT_RAW TIMG_T0_INT 的原始中断状态。(R/SS/WTC)

TIMG_WDT_INT_RAW TIMG_WDT_INT 的原始中断状态。(R/SS/WTC)

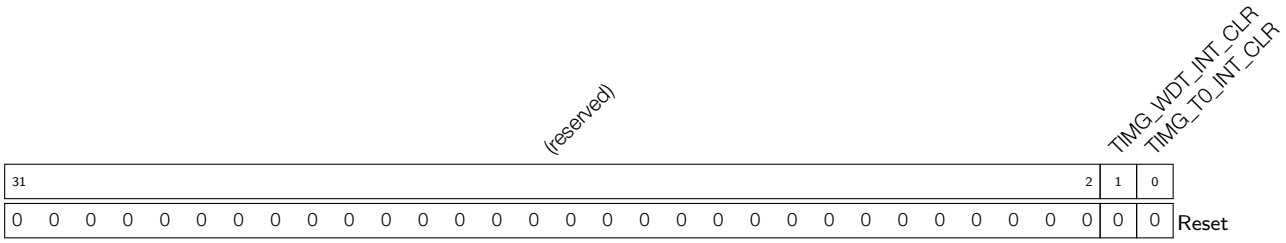
Register 12.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_T0_INT_ST TIMG_T0_INT 的屏蔽中断状态。(RO)

TIMG_WDT_INT_ST TIMG_WDT_INT 的屏蔽中断状态。(RO)

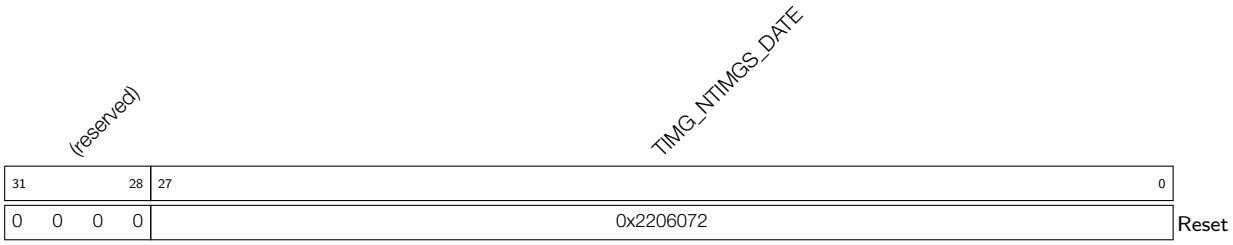
Register 12.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T0_INT_CLR 写 1 清除 TIMG_T0_INT 中断。(WT)

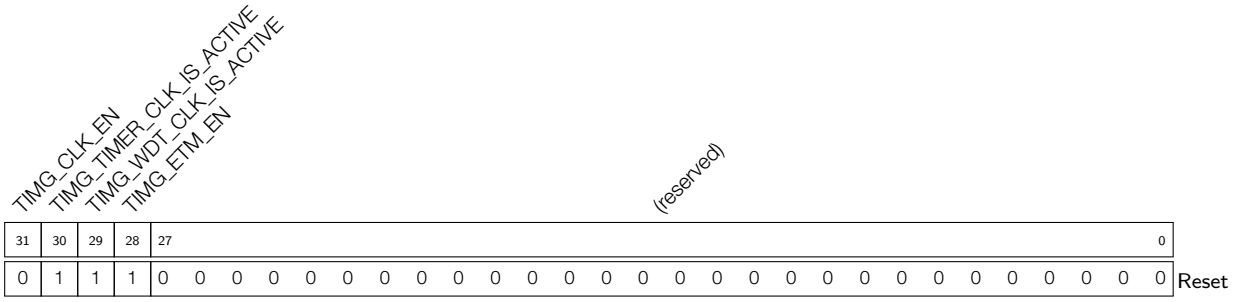
TIMG_WDT_INT_CLR 写 1 清除 TIMG_WDT_INT 中断。(WT)

Register 12.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE 版本控制寄存器。(R/W)

Register 12.26. TIMG_REGCLK_REG (0x00FC)



TIMG_ETM_EN 配置是否使能定时器的 ETM 任务和事件。

0：禁用

1：使能

(R/W)

TIMG_WDT_CLK_IS_ACTIVE 配置是否使能看门狗定时器的时钟。

0：禁用

1：使能

(R/W)

TIMG_TIMER_CLK_IS_ACTIVE 配置是否使能定时器 0 的时钟。

0：禁用

1：使能

(R/W)

TIMG_CLK_EN 配置是否使能寄存器的门时钟信号。

0：强制使能寄存器的时钟

1：仅在软件读取或写入寄存器时启动时钟。

(R/W)

13 看门狗定时器 (WDT)

13.1 概述

看门狗定时器是一种硬件定时器，用于检测和修复故障。软件必须定期喂狗（复位），以防超时。系统或软件若出现不可预知的问题（比如软件卡在某个循环或逾期事件中）将无法按时喂狗，造成看门狗超时。因此，看门狗定时器有助于检测、处理系统或软件的错误行为。

如图 13-1 所示，ESP32-H2 中有三个数字看门狗定时器：章节 12 定时器组 *(TIMG)* 描述的两个定时器组中各有一个（称作主系统看门狗定时器，缩写为 MWDT），RTC 模块中有一个（称作 RTC 看门狗定时器，缩写为 RWDT）。数字看门狗在运行期间会经历四个阶段（除非看门狗按时喂狗或者处于关闭状态），每个阶段均可配置单独的超时时间和超时动作，其中 MWDT 支持中断、CPU 复位和内核复位三种超时动作，RWDT 支持中断、CPU 复位、内核复位和系统复位四种超时动作（详见章节 13.2.2.2 阶段与超时动作）。每个阶段的超时时间都可单独设置。

在 flash 引导模式下，RWDT 和定时器组 0 的 MWDT 会默认使能，以检测引导过程中发生的错误，并恢复运行。

ESP32-H2 中还有一个模拟看门狗定时器——超级看门狗 (SWD)。超级看门狗是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。

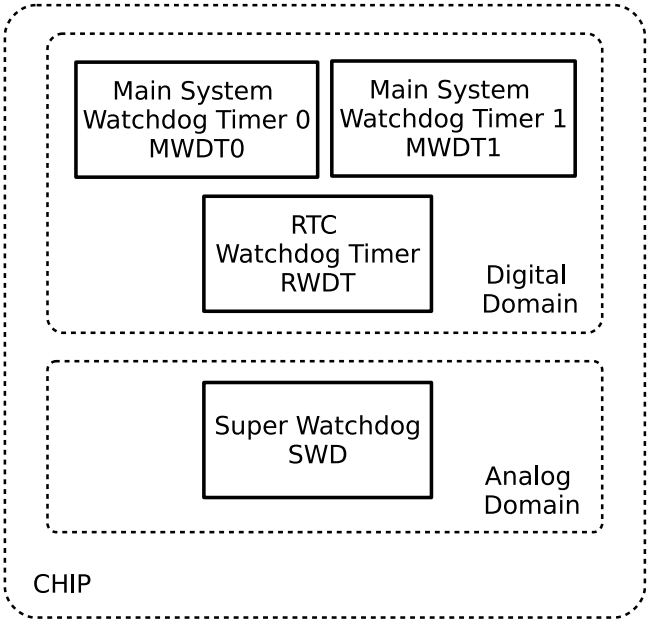


图 13-1. 看门狗定时器概览

请注意，MWDT 寄存器部分详见章节 12 定时器组 *(TIMG)*，RWDT 和 SWD 寄存器部分详见 13.5 寄存器列表。

> **说明：**
> 除非特别说明，本章中的 MWDT 均指 MWDT0 和 MWDT1.

13.2 数字看门狗定时器

13.2.1 主要特性

看门狗定时器具有如下特性：

- 四个阶段，每个阶段都可配置超时时间和超时动作
- 超时动作
  - MWDT：中断、CPU 复位、内核复位
  - RWDT：中断、CPU 复位、内核复位、系统复位
- 阶段 0 flash 启动保护：
  - MWDT0：超时触发内核复位
  - RWDT：超时触发系统复位
- 写保护，使能时寄存器仅可读取
- 32 位超时计数器
- 时钟源：
  - MWDT：PLL_F48M_CLK、RC_FAST_CLK 或 XTAL_CLK
  - RWDT：RTC_SLOW_CLK

13.2.2 功能描述

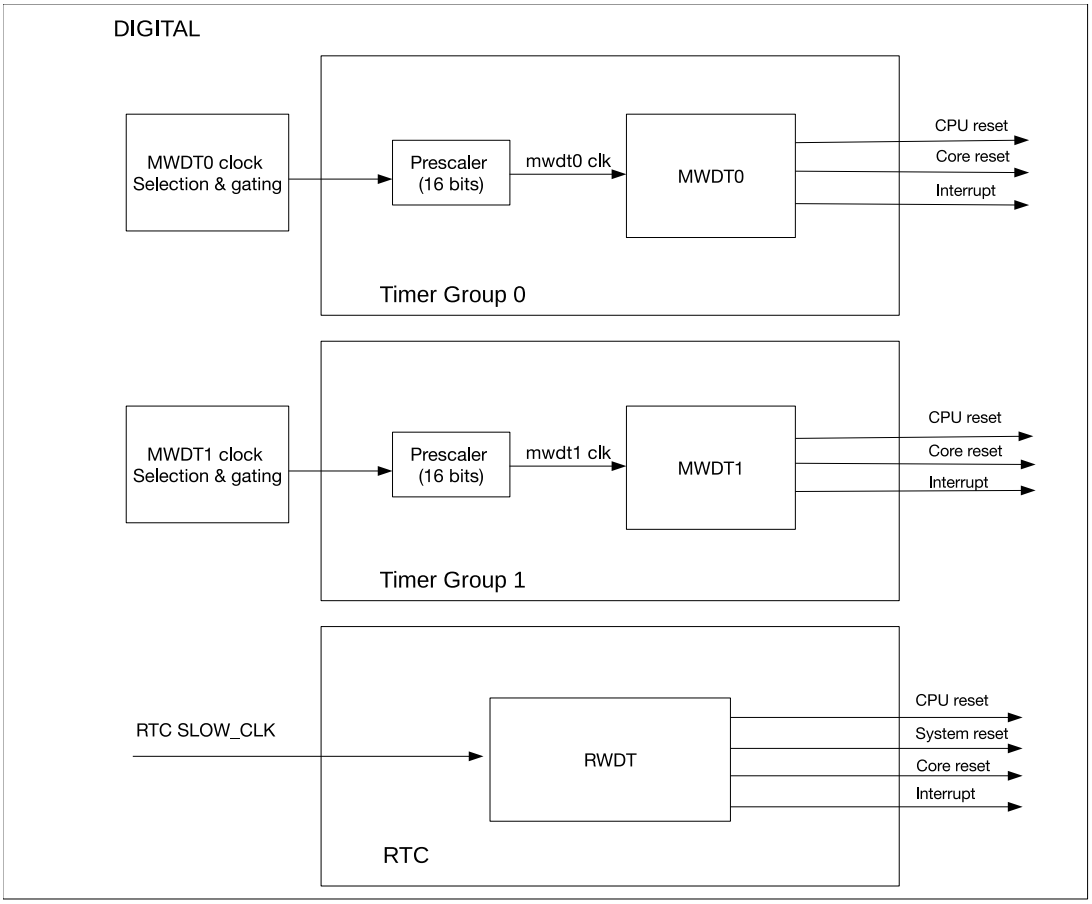


图 13-2. ESP32-H2 的数字看门狗定时器

图 13-2 为 ESP32-H2 数字系统中的三个看门狗定时器。

13.2.2.1 时钟源与 32 位计数器

看门狗定时器的核心是一个 32 位计数器。

以 MWDT0 为例：

- MWDT0 可通过设置 PCR_TIMERGROUP0_WDT_CLK_CONF_REG 寄存器的 PCR_TG0_WDT_CLK_SEL 字段选择 PLL_F48M_CLK 时钟、RC_FAST_CLK 或 XTAL_CLK （外部时钟）作为时钟源。
- 将 PCR_TIMERGROUP0_WDT_CLK_CONF_REG 寄存器的 PCR_TG0_WDT_CLK_EN 字段置 1 开启时钟，清零关闭时钟。时钟经由可配置的 16 位预分频器分频。详见章节 7 复位和时钟 的表 7-2 高性能系统时钟。
- MWDT0 的 16 位预分频器可通过 TIMG*n*_WDTCONFIG1_REG (*n*: 0 ~ 1, 此处对于定时器组 0，*n* 的取值应为 0) 寄存器的 TIMG*n*_WDT_CLK_PRESCALE 字段配置。TIMG*n*_WDT_DIVCNT_RST 字段置位时，预分频器复位，可立即重新配置。

RWDT 直接将 RTC 慢速时钟 RTC_SLOW_CLK（详见章节 7 复位和时钟）用作时钟源。

MWDT 和 RWDT 看门狗可分别通过设置 TIMG*n*_WDT_EN 和 LP_WDT_RWDT_EN 字段使能。看门狗使能后，其内部 32 位计数器的值会在每个时钟源周期内累加 1，直到达到该阶段的超时时间（即在该阶段发生超时）。如发生超时，计数器的值会重置为 0，同时看门狗进入下一阶段。如果软件成功喂狗，看门狗定时器会回到阶段 0，并将计数器的值重置为 0。软件向 TIMG*n*_WDTFEED_REG 寄存器内写入任意值，便可为 MWDT 喂狗；向 LP_WDT_RWDT_FEED 写 1，便可为 RWDT 喂狗。

13.2.2.2 阶段与超时动作

定时器在各阶段可以配置不同的超时时间和对应的超时动作。某一阶段超时会触发对应的超时动作，同时计数器的值被重置为 0，看门狗进入下一阶段。

MWDT0、MWDT1 和 RWDT 各提供四个阶段，称为阶段 0 到阶段 3。看门狗定时器会循环工作，即从阶段 0 开始，然后前进到阶段 3，再返回到阶段 0。

MWDT 每个阶段的超时时间可用 TIMG*n*_WDTCONFIG*i*_REG（i 的范围是 2 到 5）寄存器配置，RWDT 的超时时间可用 LP_WDT_RWDT_STG*j*_HOLD（j 的范围是 0 到 3）字段配置。

值得注意的是，RWDT 在阶段 0 的超时时间 (T_{hold0}) 受 eFuse 寄存器 EFUSE_RD_REPEAT_DATA0_REG 的 EFUSE_WDT_DELAY_SEL 字段和 LP_WDT_RWDT_STG0_HOLD 字段共同影响，关系如下：

$$T_{hold0} = LP\_WDT\_RWDT\_STG0\_HOLD << (EFUSE\_WDT\_DELAY\_SEL + 1)$$

其中，<< 为左移运算符。例如，LP_WDT_RWDT_STG0_HOLD 配置为 100，EFUSE_WDT_DELAY_SEL为 1，则 T_{hold0} 为 400 个时钟周期。

如某个阶段超时，下列超时动作之一将会执行：

表 13-1. 超时动作

超时动作	描述
中断	触发中断
CPU 复位	复位 CPU 核心
内核复位	复位除低功耗系统以外的其他数字系统，包括 CPU、外设、数字 GPIO、Bluetooth® LE、802.15.4
系统复位	复位包括低功耗系统在内的整个数字系统，此动作仅可在 RWDT 中实现
关闭	对系统不产生影响

MWDT 所有阶段的超时动作均在 TIMG*n*_WDTCONFIG0_REG 寄存器中配置。RWDT 的超时动作可在 LP_WDT_RWDT_CONFIG0_REG 寄存器配置。

13.2.2.3 写保护

看门狗定时器对于检测和处理系统或软件错误而言至关重要，不应轻易关闭（例如，因写寄存器位置错误而误将看门狗关闭）。因此，MWDT 和 RWDT 引入写保护机制，防止看门狗因无意的写操作而被关闭或篡改。

写保护机制通过每个看门狗定时器的写密钥字段运行（MWDT 看门狗使用 TIMG*n*_WDT_WKEY，RWDT 看门狗使用 LP_WDT_RWDT_WKEY）。必须向看门狗定时器的写密钥字段写入 0x50D83AA1，才能修改其它看门狗寄存器。如果写密钥字段的值不是 0x50D83AA1，任何试图向看门狗定时器寄存器（除了向写密钥字段本身）写值的操作都会被忽略。推荐按以下步骤访问看门狗定时器：

1. 将 0x50D83AA1 写入看门狗定时器的写密钥字段，关闭写保护。
2. 根据需要修改看门狗，如喂狗或改变配置。
3. 向看门狗定时器的写密钥字段上写入除 0x50D83AA1 以外的任意值，重新使能写保护。

13.2.2.4 Flash 引导保护

在 flash 引导模式下，MWDT0 和 RWDT 会默认使能。MWDT0 的阶段 0 的默认超时动作为内核复位（复位主系统）。RWDT 的阶段 0 超时动作为系统复位（复位主系统和 RTC）。引导后，应将 TIMG*n*_WDT_FLASHBOOT_MOD_EN 和 LP_WDT_RWDT_FLASHBOOT_MOD_EN 位清零，分别关闭 MWDT0 和 RWDT 的 flash 引导保护。然后，软件可以配置 MWDT0 和 RWDT。

13.3 超级看门狗定时器

超级看门狗 (SWD) 是模拟域的超低功耗电路，可以防止系统在数字电路异常状态下运行，并在必要时复位系统（即超时动作中的系统复位）。SWD 包含一个看门狗电路，需在每个超时阶段（约不足一秒）至少喂狗一次。该电路会在看门狗超时时间约 100 ms 之前发送 WD_INTR 信号提醒系统喂狗。

如果系统不回应 SWD 的喂狗请求，看门狗超时，SWD 会产生系统电平信号 SWD_RSTB，复位芯片上的整个数字电路（即超时动作中的系统复位）。

SWD 的时钟源固定，不可选择。

13.3.1 主要特性

SWD 具有如下特性：

- 超低功耗
- 用中断提醒 SWD 即将超时
- 软件有多种专用的方法喂 SWD，让 SWD 监控整个操作系统的工作状态

13.3.2 SWD 控制器

13.3.2.1 结构

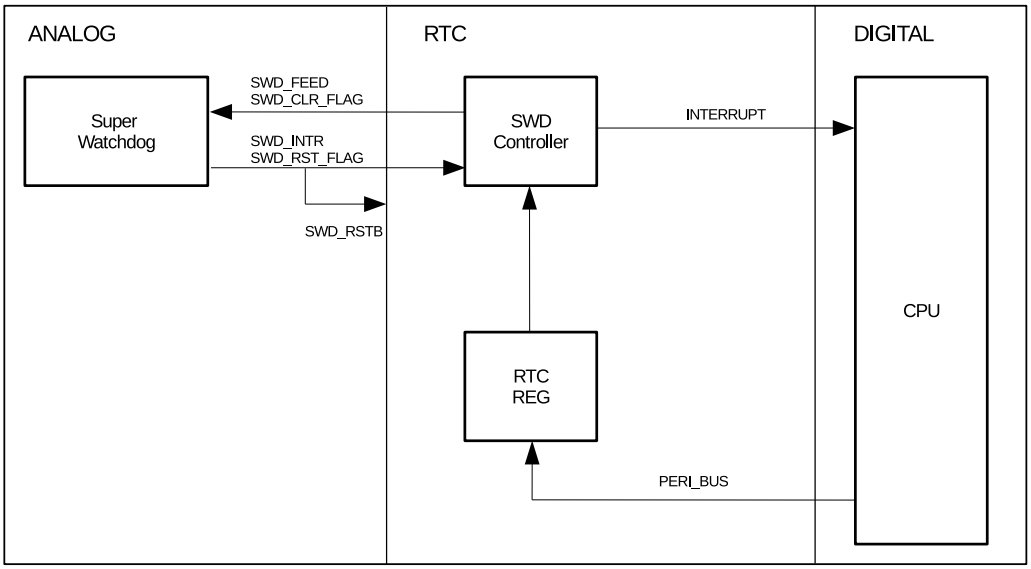


图 13-3. SWD 控制器结构

13.3.2.2 工作流程

正常状态下：

- SWD 控制器收到 SWD 的喂狗请求。
- SWD 控制器向主 CPU 发送中断。
- 主 CPU 通过置位 LP_WDT_SWD_FEED 直接喂狗。
- CPU 喂狗时，需要先向 LP_WDT_SWD_WKEY 写 0x50D83AA1 关闭 SWD 控制器的写保护。这样做可以防止系统在数字电路异常状态下运行时误喂 SWD。
- 如将 LP_WDT_SWD_AUTO_FEED_EN 置 1，SWD 控制器也可配置为在不需要 CPU 干预的情况下喂 SWD。

复位后：

- 可查看 LP_CLKRST_RESET_CAUSE[4:0] 获知 CPU 复位原因。
  如 LP_CLKRST_RESET_CAUSE[4:0] == 0x12，则表示上一次复位的原因是 SWD 复位。
- 置位 LP_WDT_SWD_RST_FLAG_CLR 清除 SWD 复位标志。

13.4 中断

看门狗定时器中断，请前往章节 12 定时器组 *(TIMG)* 的第 12.3.7 节 中断 查看。

13.5 寄存器列表

本小节的地址均为相对于低功耗看门狗定时器 (LP_WDT) 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

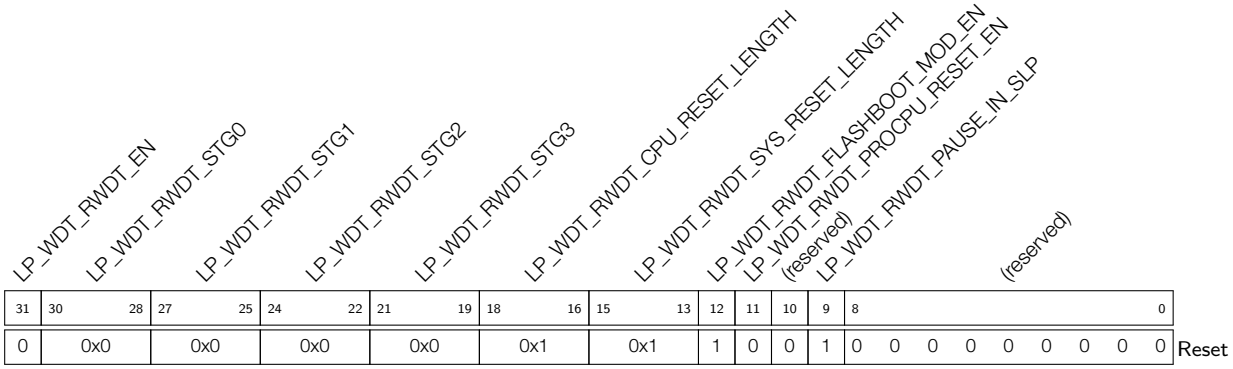
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
configuration register			
LP_WDT_RWDT_CONFIG0_REG	RWDT 配置寄存器	0x0000	R/W
LP_WDT_RWDT_CONFIG1_REG	配置 RWDT 阶段 0 的超时时间	0x0004	R/W
LP_WDT_RWDT_CONFIG2_REG	配置 RWDT 阶段 1 的超时时间	0x0008	R/W
LP_WDT_RWDT_CONFIG3_REG	配置 RWDT 阶段 2 的超时时间	0x000C	R/W
LP_WDT_RWDT_CONFIG4_REG	配置 RWDT 阶段 2 的超时时间	0x0010	R/W
LP_WDT_RWDT_FEED_REG	配置喂狗 RWDT	0x0014	WT
LP_WDT_RWDT_WPROTECT_REG	配置锁住 RWDT 的配置寄存器	0x0018	R/W
LP_WDT_SWD_CONFIG_REG	SWD 配置寄存器	0x001C	varies
LP_WDT_SWD_WPROTECT_REG	配置锁住 SWD 的配置寄存器	0x0020	R/W
LP_WDT_INT_RAW_REG	配置是否产生 WDT 超时中断	0x0024	R/WTC/SS
LP_WDT_INT_ST_REG	WDT 中断状态寄存器	0x0028	RO
LP_WDT_INT_ENA_REG	WDT 中断使能寄存器	0x002C	R/W
LP_WDT_INT_CLR_REG	WDT 中断清除寄存器	0x0030	WT
LP_WDT_DATE_REG	版本控制寄存器	0x03FC	R/W

13.6 寄存器

MWDT 寄存器是定时器组模块的一部分，在章节 12 定时器组 (TIMG) 的第 12.5 节 寄存器列表 中有详细描述。本小节包含 RWDT 和 SWD 相关寄存器介绍，其地址均为相对于低功耗看门狗定时器 (LP_WDT) 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 13.1. LP_WDT_RWDT_CONFIG0_REG (0x0000)



LP_WDT_RWDT_PAUSE_IN_SLP 配置芯片处于 sleep 模式时是否禁用 RWDT。

0：使能

1：禁用

(R/W)

LP_WDT_RWDT_PROCPU_RESET_EN 配置是否使能 RWDT 进行 CPU 复位。

0：禁用

1：使能

(R/W)

LP_WDT_RWDT_FLASHBOOT_MOD_EN 配置在 SPI boot 模式下是否自动使能 RWDT。

0：禁用

1：使能

(R/W)

LP_WDT_RWDT_SYS_RESET_LENGTH 配置内核复位时间。

单位：LP_DYN_FAST_CLK

(R/W)

LP_WDT_RWDT_CPU_RESET_LENGTH 配置 CPU 复位时间。

单位：LP_DYN_FAST_CLK

(R/W)

LP_WDT_RWDT_STG3 配置阶段 3 的超时复位动作。

0：无效

1：中断

2：CPU 复位

3：内核复位

4：系统复位

(R/W)

见下页...

Register 13.1. LP_WDT_RWDT_CONFIG0_REG (0x0000)

接上页...

LP_WDT_RWDT_STG2 配置阶段 2 的超时复位动作。

0：无效
1：中断
2：CPU 复位
3：内核复位
4：系统复位
(R/W)

LP_WDT_RWDT_STG1 配置阶段 1 的超时复位动作。

0：无效
1：中断
2：CPU 复位
3：内核复位
4：系统复位
(R/W)

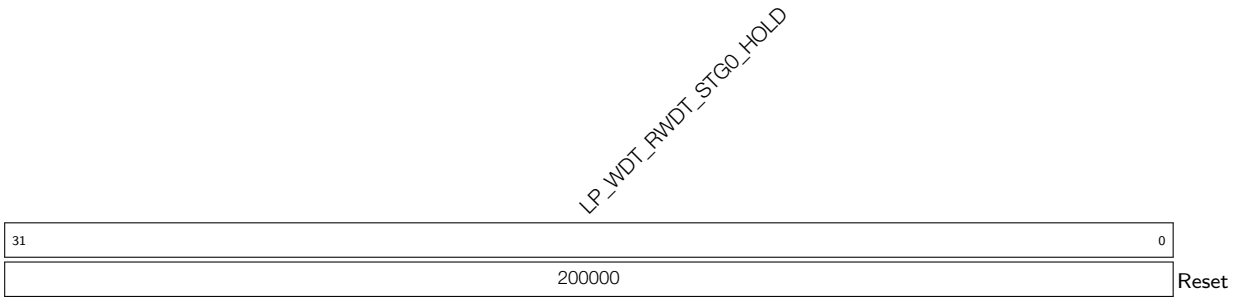
LP_WDT_RWDT_STG0 配置阶段 0 的超时复位动作。

0：无效
1：中断
2：CPU 复位
3：内核复位
4：系统复位
(R/W)

LP_WDT_RWDT_EN 配置是否使能 RWDT。

0：禁用 RWDT
1：使能 RWDT
(R/W)

Register 13.2. LP_WDT_RWDT_CONFIG1_REG (0x0004)

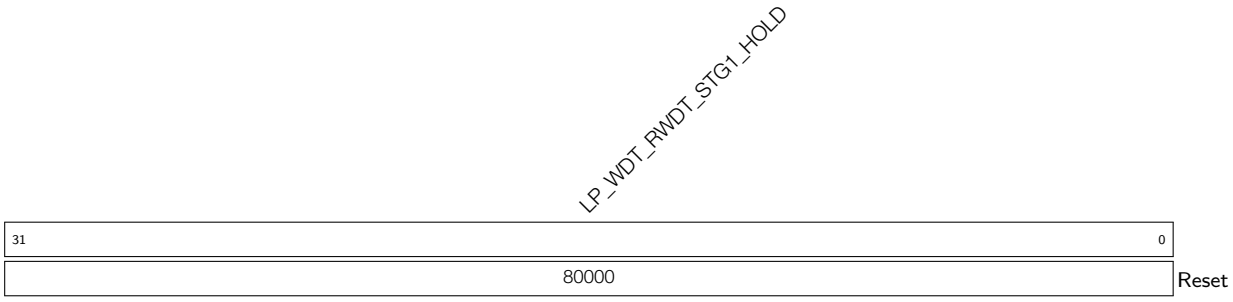


LP_WDT_RWDT_STG0_HOLD 配置阶段 0 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 13.3. LP_WDT_RWDT_CONFIG2_REG (0x0008)

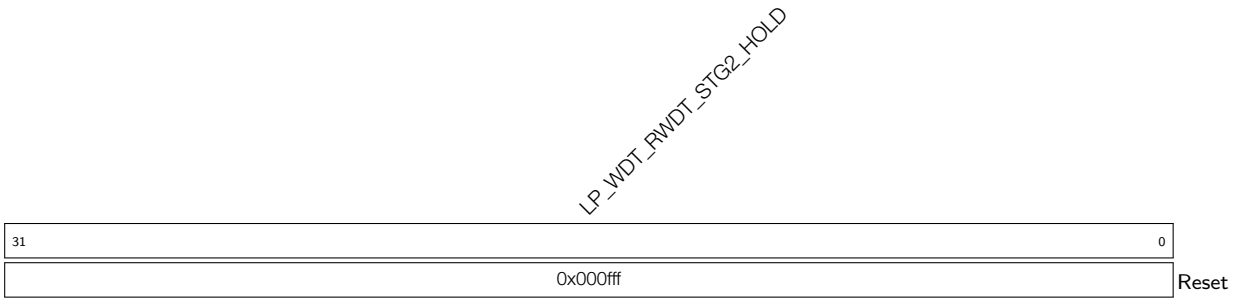


LP_WDT_RWDT_STG1_HOLD 配置阶段 1 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 13.4. LP_WDT_RWDT_CONFIG3_REG (0x000C)

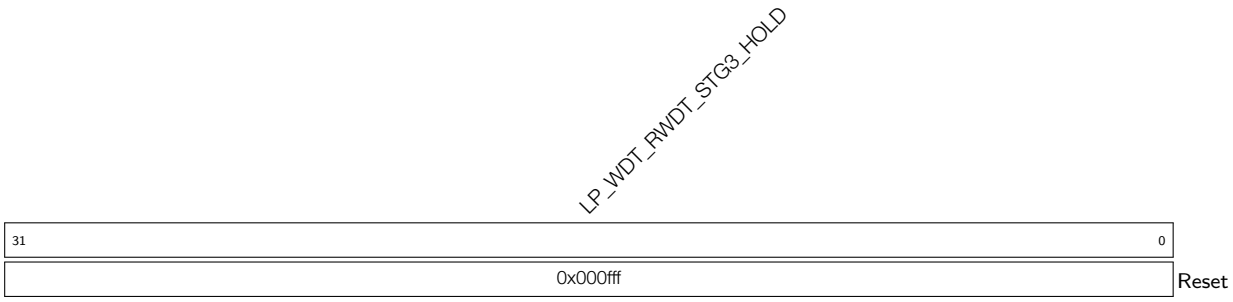


LP_WDT_RWDT_STG2_HOLD 配置阶段 2 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 13.5. LP_WDT_RWDT_CONFIG4_REG (0x0010)

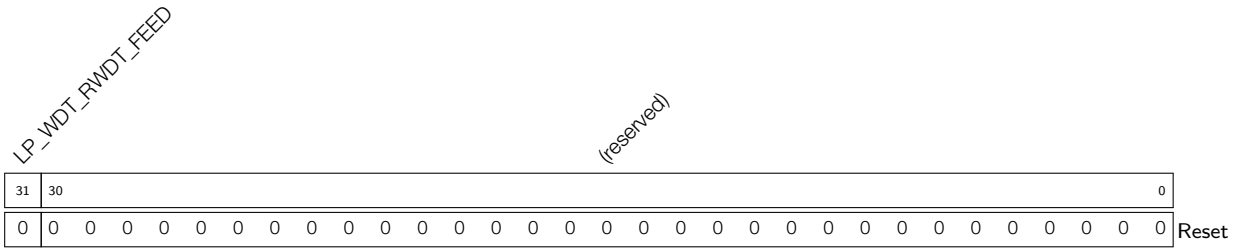


LP_WDT_RWDT_STG3_HOLD 配置阶段 3 的超时时间。

单位：LP_DYN_SLOW_CLK

(R/W)

Register 13.6. LP_WDT_RWDT_FEED_REG (0x0014)



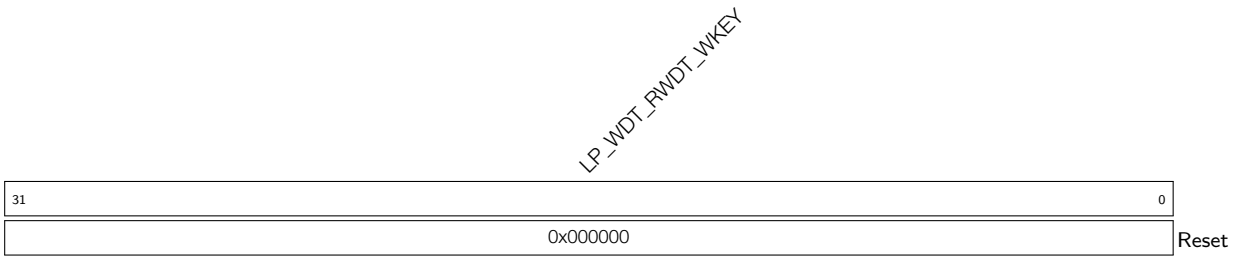
LP_WDT_RWDT_FEED 配置此位喂狗 RWDT。

0：无效

1：喂狗

(WT)

Register 13.7. LP_WDT_RWDT_WPROTECT_REG (0x0018)



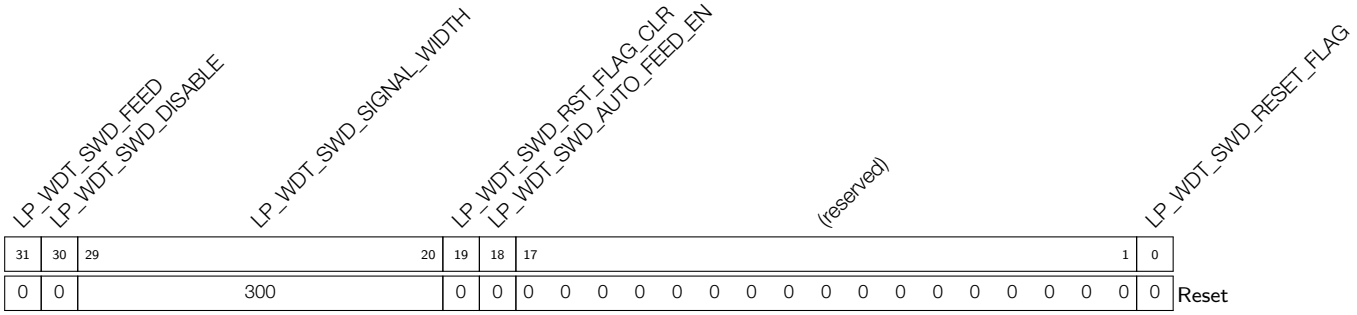
LP_WDT_RWDT_WKEY 配置该寄存器来锁住或解锁 RWDT 的配置寄存器。

0x50D83AA1: 解锁 RWDT 的配置寄存器

Others value: 锁住 RWDT 的配置寄存器, 软件不能修改 RWDT 的配置

(R/W)

Register 13.8. LP_WDT_SWD_CONFIG_REG (0x001C)



LP_WDT_SWD_RESET_FLAG 表示 SWD 是否产生复位信号。

0：否

1：是

(RO)

LP_WDT_SWD_AUTO_FEED_EN 配置此位使能硬件自动喂狗。

0：禁止硬件自动喂狗

1：使能硬件自动喂狗

(R/W)

LP_WDT_SWD_RST_FLAG_CLR 配置此位清除 SWD 复位标志。

0：无效

1：清除复位标志

(WT)

LP_WDT_SWD_SIGNAL_WIDTH 配置输出到模拟电路的信号长度。

单位：LP_DYN_FAST_CLK

(R/W)

LP_WDT_SWD_DISABLE 配置此位禁用 SWD。

0：使能

1：禁用

(R/W)

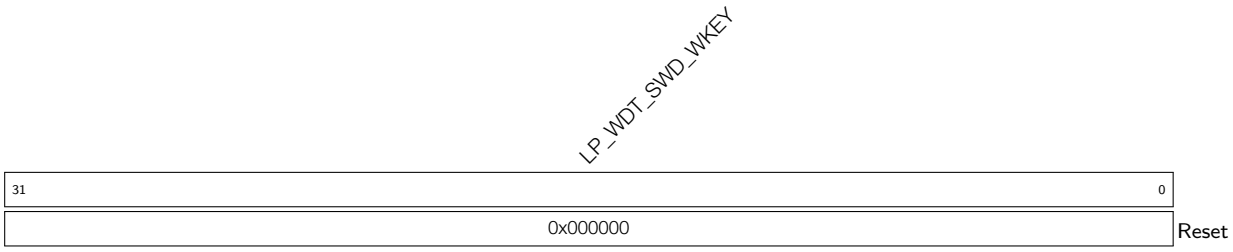
LP_WDT_SWD_FEED 配置此位喂狗 SWD。

0：无效

1: 喂狗 SWD

(WT)

Register 13.9. LP_WDT_SWD_WPROTECT_REG (0x0020)



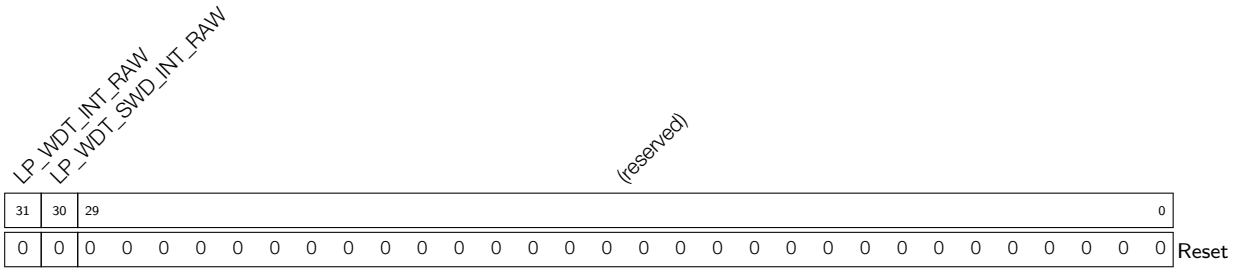
LP_WDT_SWD_WKEY 配置该寄存器来锁住或解锁 SWDT 的配置寄存器。

0x50D83AA1: 解锁 SWD 的配置寄存器

Others value: 锁住 SWD 的配置寄存器, 软件不能修改 SWD 的配置

(R/W)

Register 13.10. LP_WDT_INT_RAW_REG (0x0024)



LP_WDT_SWD_INT_RAW 表示 SWD 是否产生超时中断。

0：否

1：是

(R/WTC/SS)

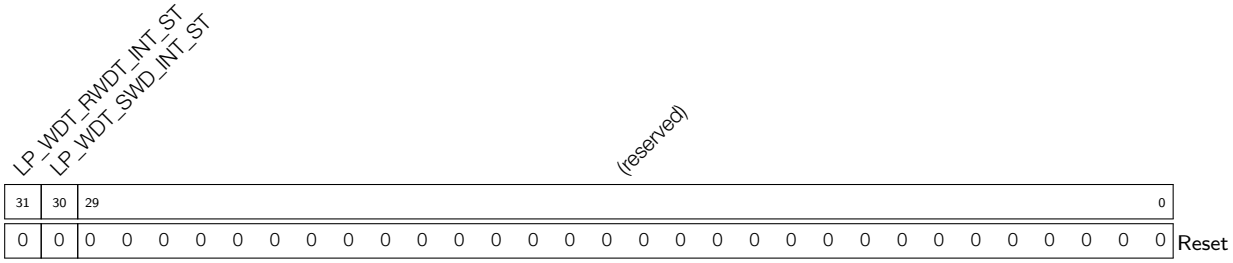
LP_WDT_RWDT_INT_RAW 表示 RWDT 是否产生超时中断。

0：否

1：是

(R/WTC/SS)

Register 13.11. LP_WDT_INT_ST_REG (0x0028)



LP_WDT_SWD_INT_ST 表示 SWD 是否产生并发送超时中断给 CPU。

0：否

1：是

(RO)

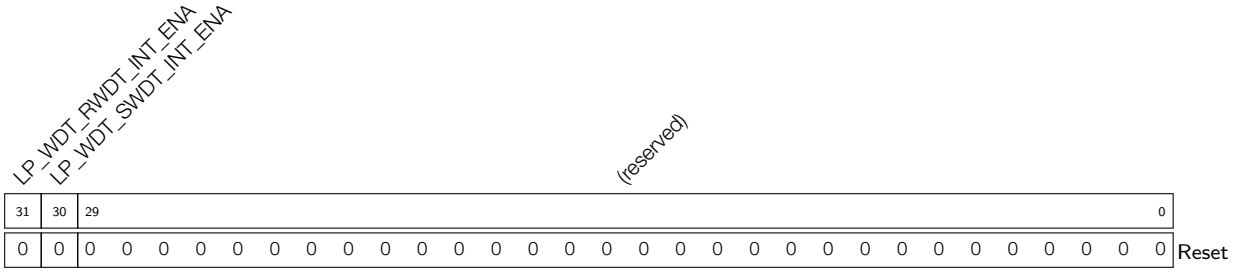
LP_WDT_RWDT_INT_ST 表示 RWDT 是否产生并发送超时中断给 CPU。

0：否

1：是

(RO)

Register 13.12. LP_WDT_INT_ENA_REG (0x002C)



LP_WDT_SWD_INT_ENA 配置是否使能 SWD 发送超时中断。

0：否

1：是

(R/W)

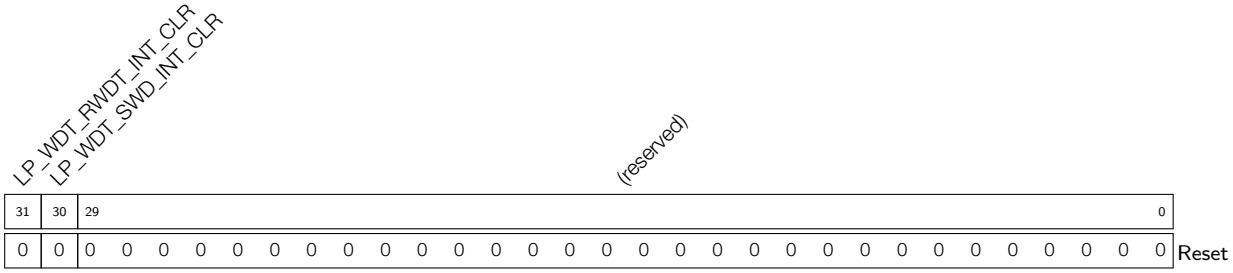
LP_WDT_RWDT_INT_ENA 配置是否使能 RWDT 发送超时中断。

0：否

1：是

(R/W)

Register 13.13. LP_WDT_INT_CLR_REG (0x0030)



LP_WDT_SWD_INT_CLR 配置是否清除 SWD 发送到 CPU 的超时中断信号。

0：否

1：是

(WT)

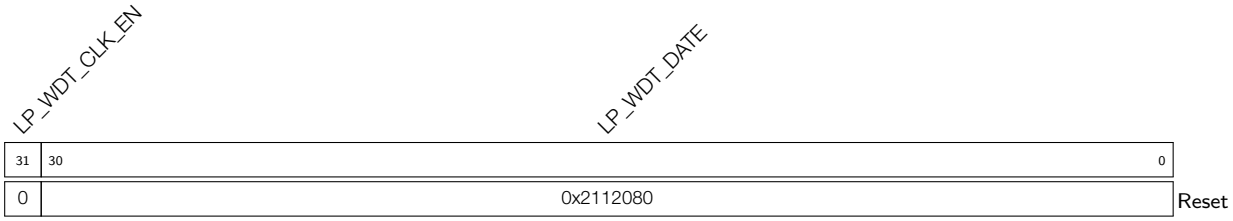
LP_WDT_RWDT_INT_CLR 配置是否清除 RWDT 发送到 CPU 的超时中断信号。

0：否

1：是

(WT)

Register 13.14. LP_WDT_DATE_REG (0x03FC)



LP_WDT_DATE 版本控制寄存器 (R/W)

LP_WDT_CLK_EN 保留 (R/W)

14 访问权限管理 (APM)

14.1 概述

ESP32-H2 整个系统的权限管理可以分为两部分：PMP（Physical Memory Protection，物理存储器保护）和 APM（Access Permission Management，访问权限管理）。

PMP 和 APM 管控的区域分布如表 14-1 所示。

表 14-1. PMP 和 APM 管控的区域分布

从机 / 主机	ROM	HP SRAM	LP SRAM	CPU_PERI[1]	HP_PERI[2]	LP_PERI[3]	EX_MEM[4]
CPU	PMP	PMP	PMP + APM	PMP + APM	PMP + APM	PMP + APM	PMP
其他主机[5]	N/A	APM	APM	N/A	N/A	N/A	N/A

[1] CPU 相关外设寄存器，地址范围：0x600C_0000 –0x600C_FFFF

[2] 高性能系统相关外设寄存器，地址范围：0x6000_0000 –0x600A_FFFF

[3] 低功耗系统相关外设寄存器，地址范围：0x600B_0000 –0x600B_FFFF

[4] 外部存储器，如 flash。

[5] 如 GDMA、MEM_MONITOR 等其他可以发起总线读写请求的主机，可参考表 14-4。

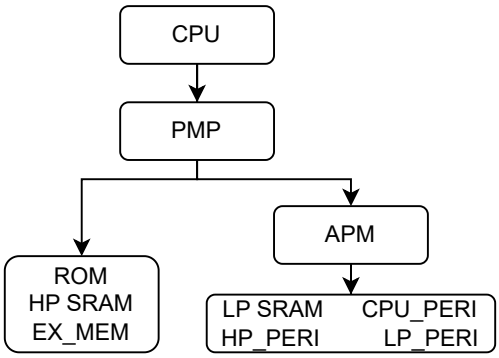


图 14-1. PMP-APM 管控关系图

对于主机 CPU，PMP 与 APM 的管控关系如图 14-1 所示。PMP 能够管控 CPU 访问所有的地址空间。APM 不能管控 CPU 访问 ROM、HP SRAM 和 EX_MEM。主机 CPU 如果需要访问 ROM、HP SRAM 和 EX_MEM 数据通路，只需要受 PMP 管控；如果要访问 LP SRAM 等其他数据通路，则需要先经过 PMP 的权限管控再经过 APM 的权限管控，如果 PMP 检查不通过，则不会触发 APM 权限管控。

PMP 相关的寄存器位于 HP CPU 内部，需要通过特殊指令进行获取或配置。关于如何配置 PMP，请参考章节 *ESP-RISC-V CPU* > *物理存储器保护*。以下章节将详细介绍 APM 模块的功能和配置。

APM 权限管控模块包含两部分：TEE（Trusted Execution Environment，可信执行环境）控制器和 APM 控制器。他们分别包含自己的寄存器模块：TEE 寄存器和 APM 寄存器。

- TEE 控制器负责配置 ESP32-H2 中主机（如 GDMA）访问内存或外设寄存器的安全模式，共支持四种安全模式：TEE、REE0 (Rich Execution Environment)、REE1 和 REE2。
- APM 控制器负责管理主机访问内存和外设寄存器的权限。通过将预先配置好的地址范围及每个地址范围对应的读、写以及执行权限，与总线上携带的信息，如 ID 号（具体可参考表 14-4)、安全模式、访问地址、

读写执行等信息做对比，从而判断是否允许访问。

TEE 寄存器用于配置每个主机的安全模式，APM 寄存器用于指定每一个安全模式的访问权限和访问地址范围。通过 TEE 控制器和 APM 控制器的配合，可以精确控制所有主机对存储器和外设寄存器的访问权限。

14.2 主要特性

ESP32-H2 的 TEE 控制器具有以下特性：

- 支持 4 种安全模式
- 支持 32 个主机安全模式配置

ESP32-H2 的 APM 控制器具有以下特性：

- 最多支持 16 组地址范围的配置
- 支持内部存储器以及外设寄存器的访问权限管理
- 支持中断功能
- 支持异常信息记录功能

14.3 TEE 与 REE 术语

TEE 代表可信执行环境 (Trusted Execution Environment)，它是与主操作系统隔离的安全区域，可以为执行敏感操作提供安全环境。

REE 代表常规执行环境 (Rich Execution Environment)，作为一般用途的执行环境，用于运行主大多数应用程序。

表 14-3. TEE 与 REE 之间的比较

方面	TEE	REE
安全性	安全增强	普通
访问权限	TEE 模式下的主机始终拥有地址范围内的读、写、执行权限。	由用户配置安全权限。REE0/1/2 的安全等级也由用户自定义。

14.4 功能描述

14.4.1 TEE 控制器功能描述

ESP32-H2 芯片支持四种安全模式：TEE、REE0、REE1、REE2。

当 CPU 作为主机访问内存或外设寄存器时，可先配置 CPU 的用户模式或机器模式，然后配置其安全模式。关于 CPU 机器模式及用户模式的配置方法请参考 RISC-V 指令集手册 v1.10 第二卷“特权架构”(RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Version 1.10)。

- 当 CPU 为机器模式时，其安全模式为 TEE 模式。
- 当 CPU 为用户模式时，其安全模式为 REE 模式，具体还需设置 TEE_M0_MODE_CTRL_REG 寄存器中的 TEE_M0_MODE 指定 REE0，REE1 或 REE2 模式：
  - 若 TEE_M0_MODE 为 0，即选择 TEE 模式，但是在 CPU 的用户模式下，其实际的安全模式为 REE0

模式。

- 若 TEE_M0_MODE 为 1，2 或 3，则安全模式分别为 REE0，REE1，REE2。

对于其他主机访问内存或外设寄存器，可通过配置 TEE 寄存器模块的 TEE_M*n*_MODE 来配置其安全模式，其中 *n* 与表 14-4 中的主机 ID 号相对应。

表 14-4. 主机访问来源

值	来源
0	CPU
1	保留
2	保留
3	保留
4	保留
5	MEM_MONITOR
6	TRACE
7 ~ 15	保留
16 ~ 31	详见章节 3 *通用 DMA 控制器 (GDMA)* > 表 3-1 中 0 ~ 15 对应的外设。例如，16 对应的来源为该表中 0 对应的外设，17 对应的来源为该表中 1 对应的外设，以此类推。

14.4.2 APM 控制器功能描述

14.4.2.1 结构概览

图 14-2 为 APM 控制器的结构图，展示了 APM 控制器管控的总线访问路径。

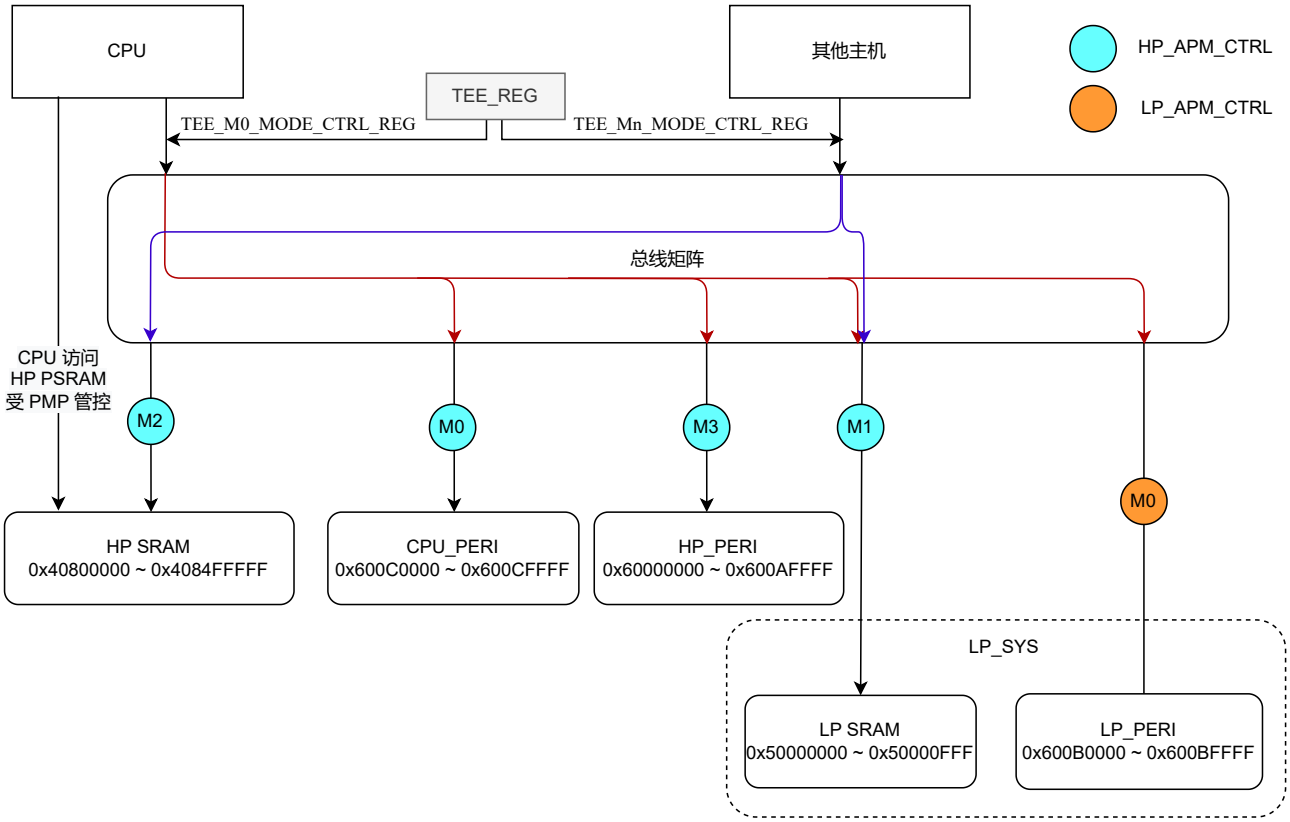


图 14-2. APM 控制器结构图

如图所示，APM 控制器共有两个功能模块：HP_APM_CTRL 和 LP_APM_CTRL。这两个功能模块分别由 高性能 APM 寄存器 (HP_APM_REG) 和 低功耗 APM 寄存器 (LP_APM_REG) 两个寄存器模块配置。

- HP_APM_CTRL 管控 4 条访问路径，即图 14-2 中的 M0 - M3，可通过配置 HP_APM_FUNC_CTRL_REG 使能各路径的权限检查（默认使能）。
- LP_APM_CTRL 管控 1 条访问路径，即图 14-2 中的 M0，可通过配置 LP_APM _FUNC_CTRL_REG 使能该路径的权限检查（默认使能）。

各功能模块的详细信息可参考下表 14-5：

表 14-5. 各功能模块配置信息

功能模块	寄存器模块	管控的访问路径数	访问路径权限检查使能	可配地址范围数	地址范围使能
HP_APM_CTRL	HP_APM_REG	4	HP_APM_ FUNC_CTRL_REG	16	HP_APM_REGION_ FILTER_EN_REG
LP_APM_CTRL	LP_APM_REG	1	LP_APM _FUNC_CTRL_REG	4	LP_APM_REGION_ FILTER_EN_REG

14.4.2.2 地址范围

HP_APM_REG 寄存器模块可配置 16 组地址范围。HP_APM_REGION*n*_ADDR_START 和 HP_APM_REGION*n*_ADDR_END 分别控制高性能寄存器的第 *n+1* 组地址范围的起始和结束地址。可通过配置 HP_APM_REGION_FILTER_EN_REG 寄存器的 bit *n* 使能第 *n+1* 组地址范围，默认使能第 1 组地址范围。

LP_APM_REG 寄存器模块可配置 4 组地址范围。LP_APM_REGION*n*_ADDR_START 和

LP_APM_REGION*n*_ADDR_END 分别控制低功耗寄存器的第 *n+1* 组地址范围的起始和结束地址。可通过配置 LP_APM_REGION_FILTER_EN_REG 寄存器的 bit *n* 使能第 *n+1* 组地址范围，默认使能第 1 组地址范围。

配置地址范围时，地址的最小对齐单位是 4 字节，即地址的低两位为 0。例如，地址范围可以设置为 0x4080000C ~ 0x40808774 或 0x600C0008 ~ 0X600CFF70。

14.4.2.3 地址范围的访问权限

每个地址范围内，均可以为不同的安全模式单独配置访问权限：

- TEE 模式下的主机始终拥有地址范围内的读、写、执行权限。
- REE0、REE1 和 REE2 模式下的主机，根据不同的访问路径，其读写执行权限由 HP_APM_REGION*n*_ATTR_REG 或 LP_APM_REGION*n*_ATTR_REG 配置。

由同一寄存器模块控制的所有访问路径共享该寄存器模块的配置，即配置的地址范围及其权限。例如图 14-2 中 HP_APM_CTRL M0-M3 在进行访问管控时，都依照 HP_APM_REG 寄存器模块中配置的 16 组地址范围及其对应的读、写、执行权限。LP_APM_CTRL M0 在进行访问管控时，依照 LP_APM_REG 寄存器模块中配置的 4 组地址范围及其对应的读、写、执行权限。

如图 14-2，所有主机访问 LP SRAM 时都会通过 HP_APM_CTRL M1。假设 HP_APM_M1_FUNC_EN 使能，REE1 安全模式的主机需要读取 LP SRAM 的数据，则具体流程如下：

1. HP_APM_CTRL M1 会先判断请求访问的地址是否在 HP_APM_REG 寄存器模块中配置的 16 组地址范围内。
2. 假设该访问地址在配置的第 2 组地址范围内，则 HP_APM_CTRL M1 判断该组地址范围是否使能，即 HP_APM_REGION_FILTER_EN 的 bit 1 是否为 1。
3. 若地址范围已使能，HP_APM_CTRL M1 判断主机在 REE1 模式下对第 2 组地址范围是否具有读的权限，即 HP_APM_REGION1_ATTR_REG 中的 HP_APM_REGION1_R1_R 是否有效（即为 1），若有效，则此次读请求会被正确响应，否则读到的数据恒为 0。

在 APM 寄存器配置的多个区域的地址空间重叠的情况下，例如一个区域设置为不可读，一个区域设置为可读，两个区域的地址重叠，这部分重叠的地址空间的属性是可读的。写以及执行权限适用同样的规则。

说明：

- 上电时，只有 CPU 默认处于 TEE 模式，其他主机均处于 REE2 模式。并且默认状态下，APM 控制器会阻止所有处于 REE0、REE1 和 REE2 模式的主机的访问请求。
- 章节 14.7 寄存器列表 中列出的所有寄存器均只有处于 TEE 安全模式的主机才可以配置。

14.5 配置流程

主机访问内存或外设寄存器路径中 APM 管控的配置流程为：

1. 配置 CPU 到机器模式（即 TEE 模式）。
2. 根据表 14-4 中的主机 ID 号，配置 TEE_M*n*_MODE 寄存器，配置 ESP32-H2 中主机的安全模式。
3. 配置 HP_APM_REGION*n*_ADDR_START、HP_APM_REGION*n*_ADDR_END 寄存器，或者 LP_APM_REGION*n*_ADDR_START、LP_APM_REGION*n*_ADDR_END 寄存器，配置 APM 寄存器中的各组地址范围。

4. 配置 HP_APM_REGION*n*_ATTR_REG 寄存器，或者 LP_APM_REGION*n*_ATTR_REG 寄存器，配置 APM 寄存器中的各组地址范围的访问权限。
5. 配置 HP_APM_REGION_FILTER_ EN_REG 寄存器，或者 LP_APM_REGION_FILTER_ EN_REG 寄存器，使能对应地址范围。
6. 配置 HP_APM_ FUNC_CTRL_REG 或 LP_APM _FUNC_CTRL_REG 寄存器使能对应路径的权限检查，默认使能。

以 I2S 通过 GDMA 访问 HP SRAM 为例，假设仅允许其在第 4 组地址范围 0x40805000 ~ 0x4080F000 地址范围内读写，则配置流程为：

1. 配置 CPU 为机器模式（即 TEE 模式）。
2. 根据表 14-4 中的主机 ID 号，配置 TEE_M19_MODE 寄存器为 1，将 I2S 通过 GDMA 访问的安全模式设置为 REE0 模式。
3. 配置 HP_APM_REGION3_ADDR_START 为 0x40805000，配置 HP_APM_REGION3_ADDR_END 为 0x4080F000。
4. 配置 HP_APM_REGION3_R0_W 和 HP_APM_REGION3_R0_R 为 1。
5. 配置 HP_APM_REGION_FILTER_EN 的 bit3 为 1。
6. 配置 HP_APM_M2_FUNC_EN 为 1。

14.6 非法访问与中断

若总线上携带的信息与配置不一致时，ESP32-H2 将视其为非法访问，并进行如下处理：

- 拒绝该访问请求，并返回默认值，具体表现为：
  - 读取和执行操作时返回 0
  - 写入操作时无效
- 触发中断

APM 控制器会自动记录非法访问的相关信息，包括主机的 ID 、安全模式、访问地址、非法访问原因（地址越限 / 权限不足）以及各访问路径的权限判定状态。所有这些信息都可以从章节 14.7 寄存器列表 列出的相关寄存器中获取。

以 HP_APM_CTRL M0 数据访问路径为例，当出现非法访问时：

- HP_APM_M0_EXCEPTION_ID 记录访问主机的 ID。
- HP_APM_M0_EXCEPTION_MODE 记录访问安全模式。
- HP_APM_M0_EXCEPTION_ADDR 记录访问地址。
- HP_APM_M0_EXCEPTION_STATUS 记录非法访问原因。
  - 当访问的地址不在 16 组地址范围中配置的区域时，bit1 会被置 1，表示地址越限。
  - 当访问的地址在某一组或多组地址范围内，但访问主机在这一组或多组地址范围内没有对应的读/写/执行操作权限时，bit0 会被置 1，表示权限不足。
- HP_APM_M0_EXCEPTION_REGION 记录各地址范围的权限判定状态。该寄存器总共 16 个 bit，对应 16 组地址范围，bit0 对应第 1 组地址范围。若访问的地址在配置使能的某一组地址范围内，但在这组地址范围内没有要求的读/写/执行操作权限时，本寄存器对应于这组地址范围的 bit 位会被置 1。

ESP32-H2 中 APM 控制器可产生 5 个中断信号，并将其发送给 中断矩阵 (INTMTX)：

- HP_APM_M0_INTR
- HP_APM_M1_INTR
- HP_APM_M2_INTR
- HP_APM_M3_INTR
- LP_APM_M0_INTR

这 5 个中断信号与图 14-2 中受管控的访问路径一一对应，当一个受管控的访问路径有非法访问，则对应的中断会被产生。

14.7 寄存器列表

14.7.1 高性能 APM 寄存器 (HP_APM_REG)

本小节的地址均为相对于访问权限管理 (HP_APM) 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
HP_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
HP_APM_REGION*n*_ADDR_START_REG (*n*: 0-15)	配置区域地址寄存器	0x0004+0xC**n*	R/W
HP_APM_REGION*n*_ADDR_END_REG (*n*: 0-15)	配置区域地址寄存器	0x0008+0xC**n*	R/W
HP_APM_REGION*n*_ATTR_REG (*n*: 0-15)	区域访问权限配置寄存器	0x000C+0xC**n*	R/W
HP_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
HP_APM_M0_STATUS_REG	HP_APM_CTRL M0 状态寄存器	0x00C8	RO
HP_APM_M0_STATUS_CLR_REG	HP_APM_CTRL M0 状态清除寄存器	0x00CC	WT
HP_APM_M0_EXCEPTION_INFO0_REG	HP_APM_CTRL M0 异常信息寄存器	0x00D0	RO
HP_APM_M0_EXCEPTION_INFO1_REG	HP_APM_CTRL M0 异常地址寄存器	0x00D4	RO
HP_APM_M1_STATUS_REG	HP_APM_CTRL M1 状态寄存器	0x00D8	RO
HP_APM_M1_STATUS_CLR_REG	HP_APM_CTRL M1 状态清除寄存器	0x00DC	WT
HP_APM_M1_EXCEPTION_INFO0_REG	HP_APM_CTRL M1 异常信息寄存器	0x00E0	RO
HP_APM_M1_EXCEPTION_INFO1_REG	HP_APM_CTRL M1 异常地址寄存器	0x00E4	RO
HP_APM_M2_STATUS_REG	HP_APM_CTRL M2 状态寄存器	0x00E8	RO
HP_APM_M2_STATUS_CLR_REG	HP_APM_CTRL M2 状态清除寄存器	0x00EC	WT
HP_APM_M2_EXCEPTION_INFO0_REG	HP_APM_CTRL M2 异常信息寄存器	0x00F0	RO
HP_APM_M2_EXCEPTION_INFO1_REG	HP_APM_CTRL M2 异常地址寄存器	0x00F4	RO
HP_APM_M3_STATUS_REG	HP_APM_CTRL M3 状态寄存器	0x00F8	RO
HP_APM_M3_STATUS_CLR_REG	HP_APM_CTRL M3 状态清除寄存器	0x00FC	WT
HP_APM_M3_EXCEPTION_INFO0_REG	HP_APM_CTRL M3 异常信息寄存器	0x0100	RO
HP_APM_M3_EXCEPTION_INFO1_REG	HP_APM_CTRL M3 异常地址寄存器	0x0104	RO

名称	描述	地址	访问
中断寄存器			
HP_APM_INT_EN_REG	HP_APM_CTRL M0/1/2/3 中断使能寄存器	0x0108	R/W
时钟门控寄存器			
HP_APM_CLOCK_GATE_REG	时钟门控寄存器	0x010C	R/W
版本控制寄存器			
HP_APM_DATE_REG	版本控制寄存器	0x07FC	R/W

14.7.2 低功耗 APM 寄存器 (LP_APM_REG)

本小节的地址均为相对于低功耗 APM (LP_APM) 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LP_APM_REGION_FILTER_EN_REG	使能区域地址范围寄存器	0x0000	R/W
LP_APM_REGION*n*_ADDR_START_REG (*n*: 0-3)	配置区域地址寄存器	0x0004+0xC**n*	R/W
LP_APM_REGION*n*_ADDR_END_REG (*n*: 0-3)	配置区域地址寄存器	0x0008+0xC**n*	R/W
LP_APM_REGION*n*_ATTR_REG (*n*: 0-3)	区域访问权限配置寄存器	0x000C+0xC**n*	R/W
LP_APM_FUNC_CTRL_REG	APM 权限检查寄存器	0x00C4	R/W
状态寄存器			
LP_APM_M0_STATUS_REG	LP_APM_CTRL M0 状态寄存器	0x00C8	RO
LP_APM_M0_STATUS_CLR_REG	LP_APM_CTRL M0 状态清除寄存器	0x00CC	WT
LP_APM_M0_EXCEPTION_INFO0_REG	LP_APM_CTRL M0 异常信息寄存器	0x00D0	RO
LP_APM_M0_EXCEPTION_INFO1_REG	LP_APM_CTRL M0 异常地址寄存器	0x00D4	RO
中断寄存器			
LP_APM_INT_EN_REG	LP_APM_CTRL M0 中断使能寄存器	0x00E8	R/W
时钟门控寄存器			
LP_APM_CLOCK_GATE_REG	时钟门控寄存器	0x00EC	R/W
版本控制寄存器			
LP_APM_DATE_REG	版本控制寄存器	0x00FC	R/W

14.7.3 高性能 TEE 寄存器

本小节的地址均为相对于可信执行环境寄存器 (TEE) 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

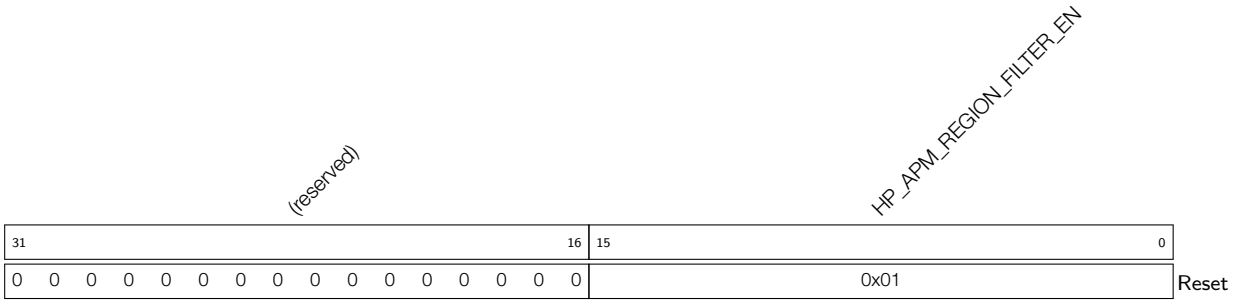
名称	描述	地址	权限
配置寄存器			
TEE_M*n*_MODE_CTRL_REG (*n*: 0-31)	安全模式配置寄存器	0x0000+0x4**n*	R/W

名称	描述	地址	权限
时钟门控寄存器			
TEE_CLOCK_GATE_REG	时钟门控寄存器	0x0080	R/W
版本控制寄存器			
TEE_DATE_REG	版本控制寄存器	0x0FFC	R/W

14.8 寄存器

14.8.1 高性能 APM 寄存器描述 (HP_APM_REG)

Register 14.1. HP_APM_REGION_FILTER_EN_REG (0x0000)



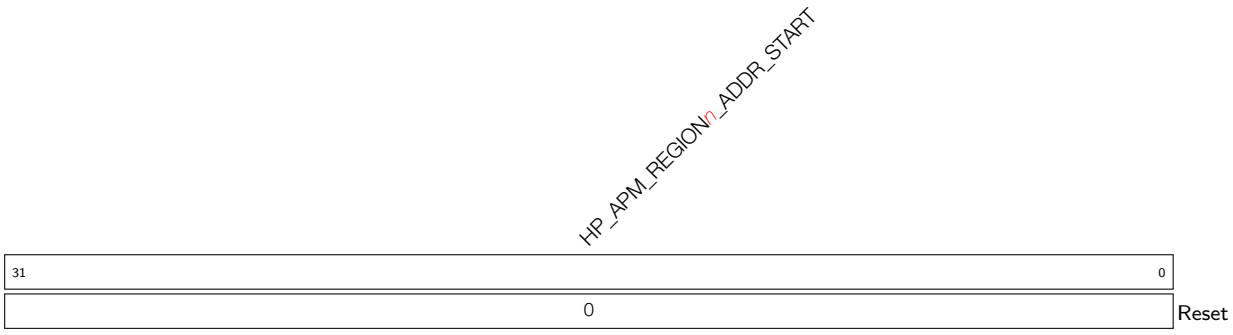
HP_APM_REGION_FILTER_EN 配置 bit *n* (0-15) 使能区域 *n* (0-15) 的地址范围。

0：禁用

1：使能

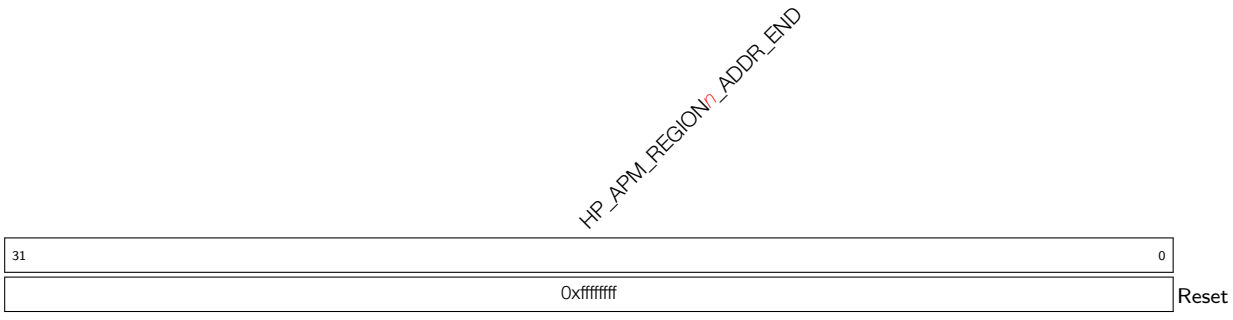
(R/W)

Register 14.2. HP_APM_REGION*n*_ADDR_START_REG (*n*: 0-15) (0x0004+0xC**n*)



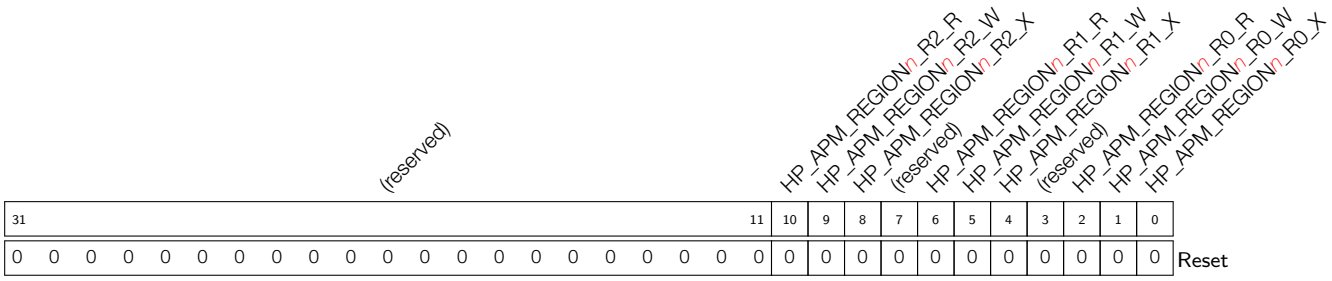
HP_APM_REGION*n*_ADDR_START 配置区域 *n* 的起始地址。(R/W)

Register 14.3. HP_APM_REGION*n*_ADDR_END_REG (*n*: 0-15) (0x0008+0xC**n*)



HP_APM_REGION*n*_ADDR_END 配置区域 *n* 的结束地址。(R/W)

Register 14.4. HP_APM_REGIONn_ATTR_REG (n: 0-15) (0x000C+0xC*n)



HP_APM_REGIONn_R0_X 配置 REE0 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGIONn_R0_W 配置 REE0 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGIONn_R0_R 配置 REE0 模式下对区域 n 的读取权限。(R/W)

HP_APM_REGIONn_R1_X 配置 REE1 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGIONn_R1_W 配置 REE1 模式下对区域 n 的写入权限。(R/W)

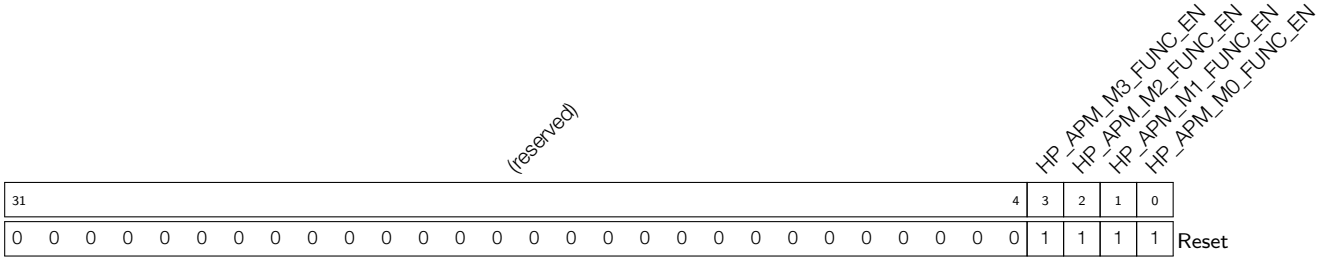
HP_APM_REGIONn_R1_R 配置 REE1 模式下对区域 n 的读取权限。(R/W)

HP_APM_REGIONn_R2_X 配置 REE2 模式下对区域 n 的执行权限。(R/W)

HP_APM_REGIONn_R2_W 配置 REE2 模式下对区域 n 的写入权限。(R/W)

HP_APM_REGIONn_R2_R 配置 REE2 模式下对区域 n 的读取权限。(R/W)

Register 14.5. HP_APM_FUNC_CTRL_REG (0x00C4)



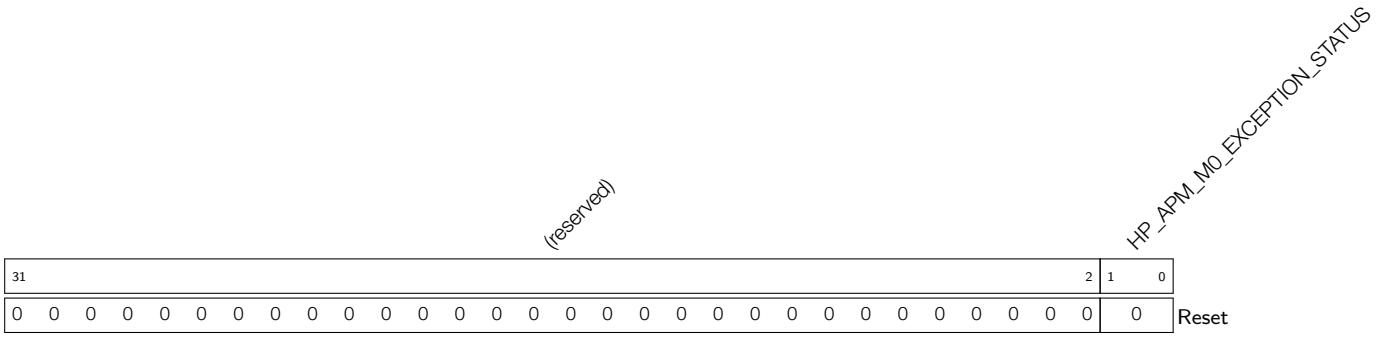
HP_APM_M0_FUNC_EN 配置使能 HP_APM_CTRL M0 权限检查。(R/W)

HP_APM_M1_FUNC_EN 配置使能 HP_APM_CTRL M1 权限检查。(R/W)

HP_APM_M2_FUNC_EN 配置使能 HP_APM_CTRL M2 权限检查。(R/W)

HP_APM_M3_FUNC_EN 配置使能 HP_APM_CTRL M3 权限检查。(R/W)

Register 14.6. HP_APM_M0_STATUS_REG (0x00C8)



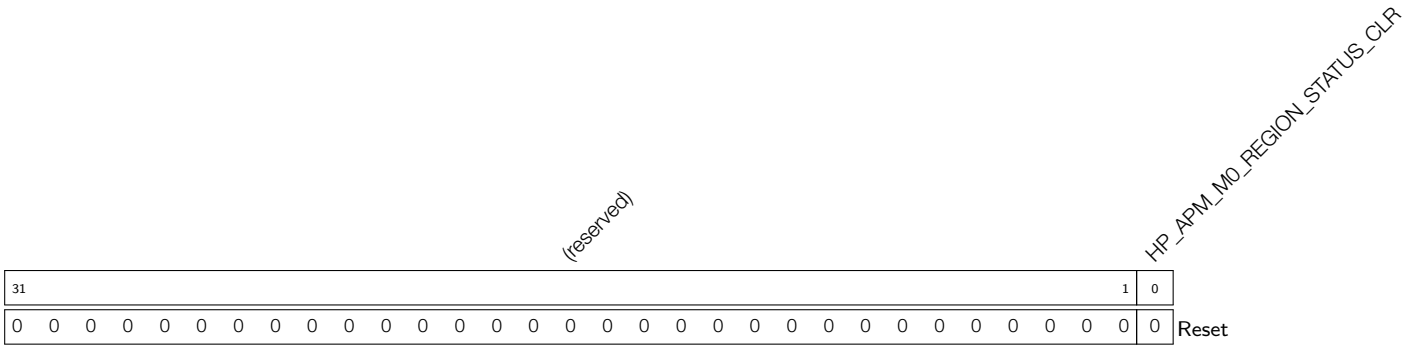
HP_APM_M0_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

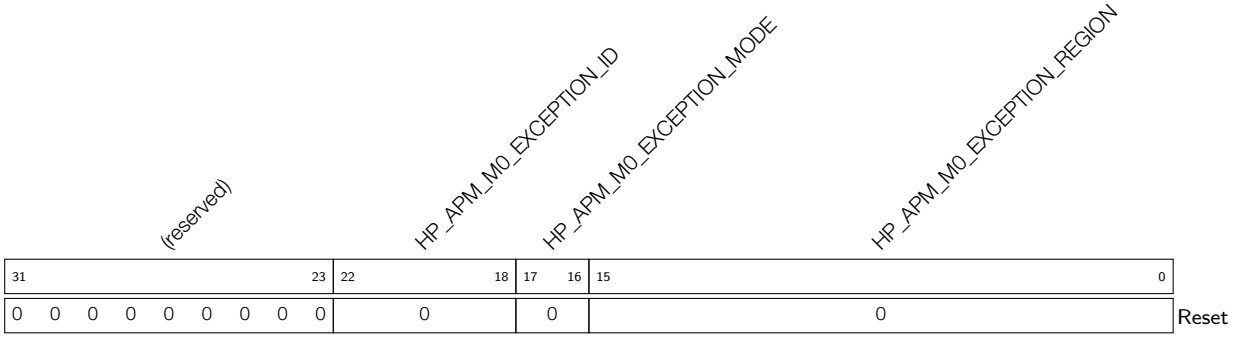
(RO)

Register 14.7. HP_APM_M0_STATUS_CLR_REG (0x00CC)



HP_APM_M0_REGION_STATUS_CLR 配置清除异常状态。(WT)

Register 14.8. HP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

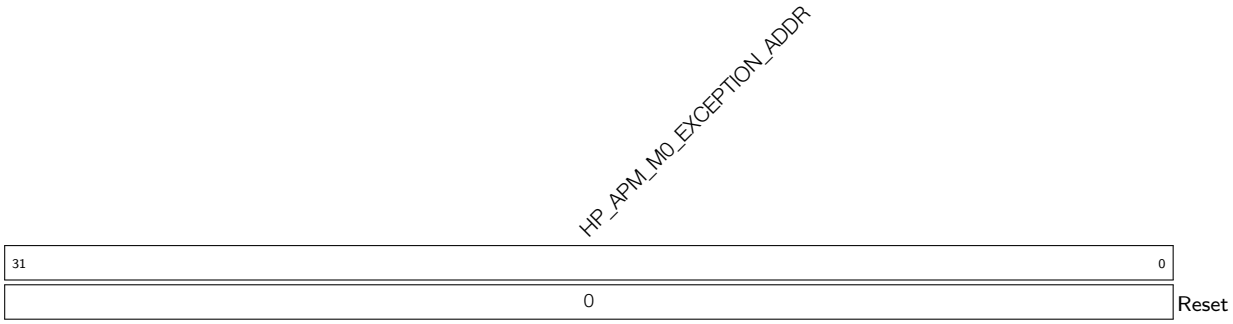


HP_APM_M0_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M0_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

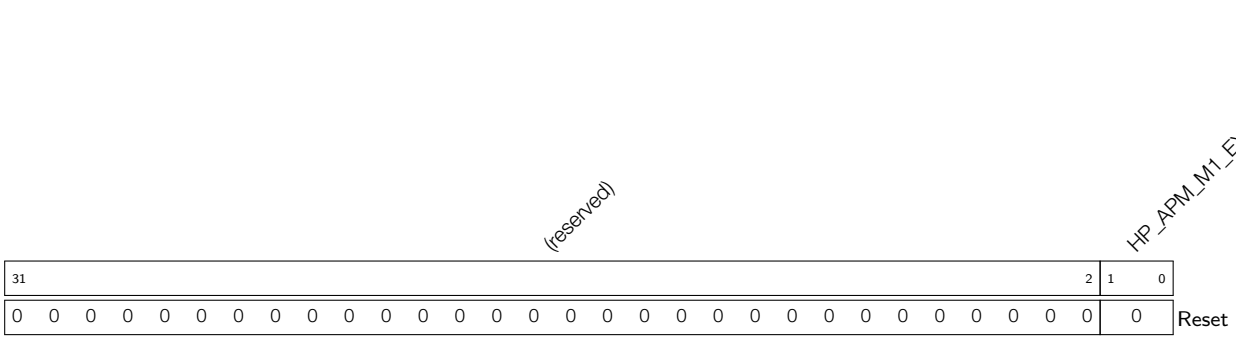
HP_APM_M0_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 14.9. HP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



HP_APM_M0_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 14.10. HP_APM_M1_STATUS_REG (0x00D8)



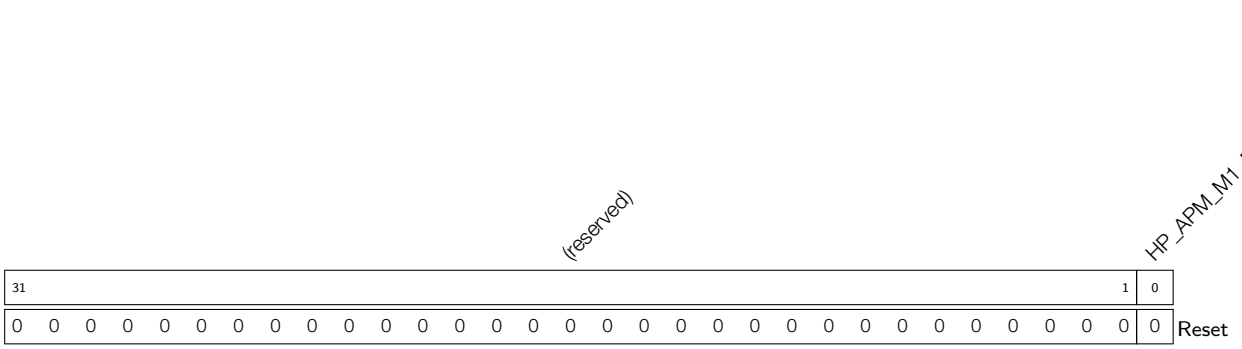
HP_APM_M1_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

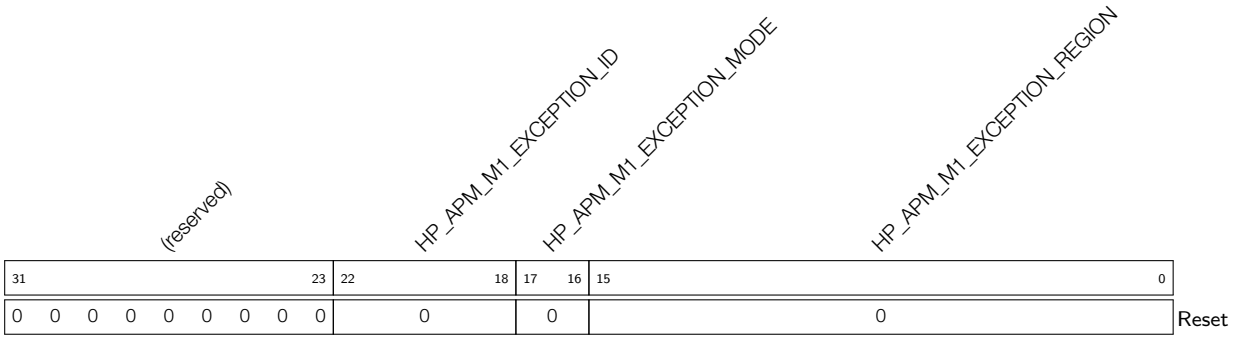
(RO)

Register 14.11. HP_APM_M1_STATUS_CLR_REG (0x00DC)



HP_APM_M1_REGION_STATUS_CLR 配置清除异常状态。(WT)

Register 14.12. HP_APM_M1_EXCEPTION_INFO0_REG (0x00E0)

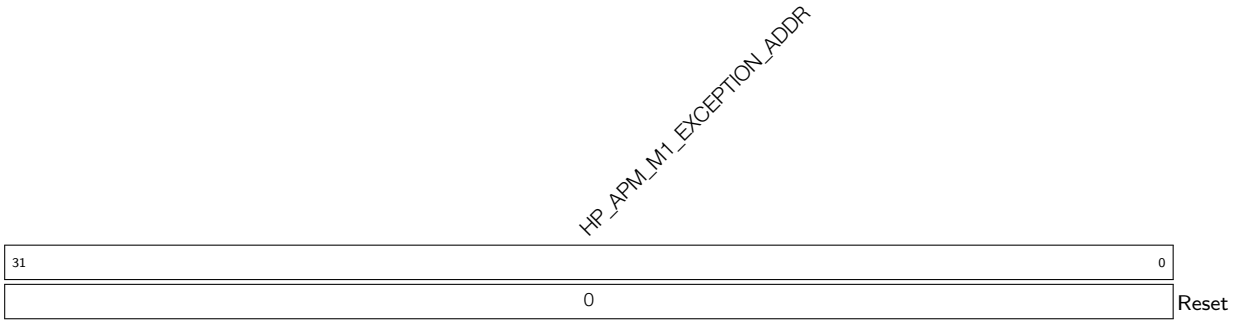


HP_APM_M1_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M1_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

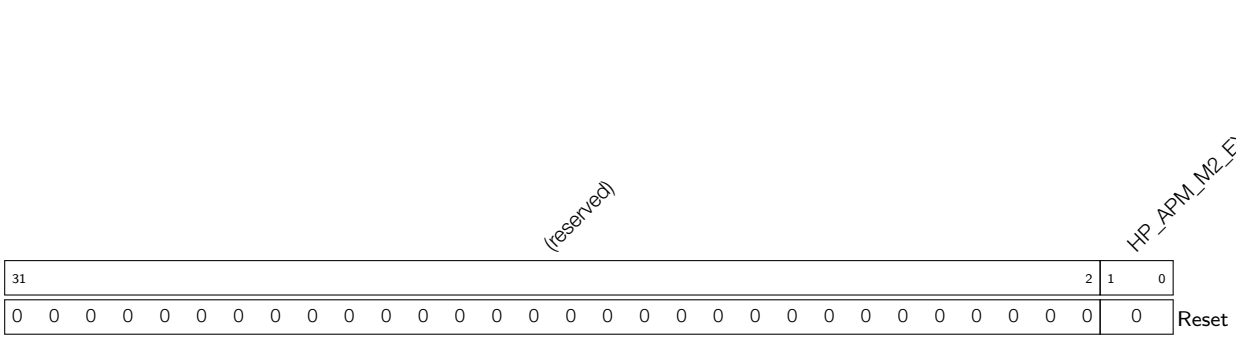
HP_APM_M1_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 14.13. HP_APM_M1_EXCEPTION_INFO1_REG (0x00E4)



HP_APM_M1_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 14.14. HP_APM_M2_STATUS_REG (0x00E8)



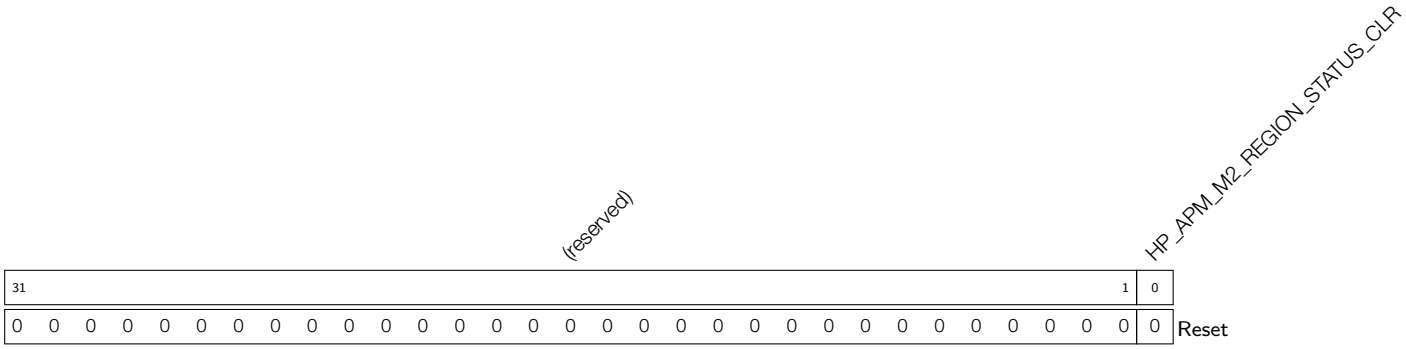
HP_APM_M2_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

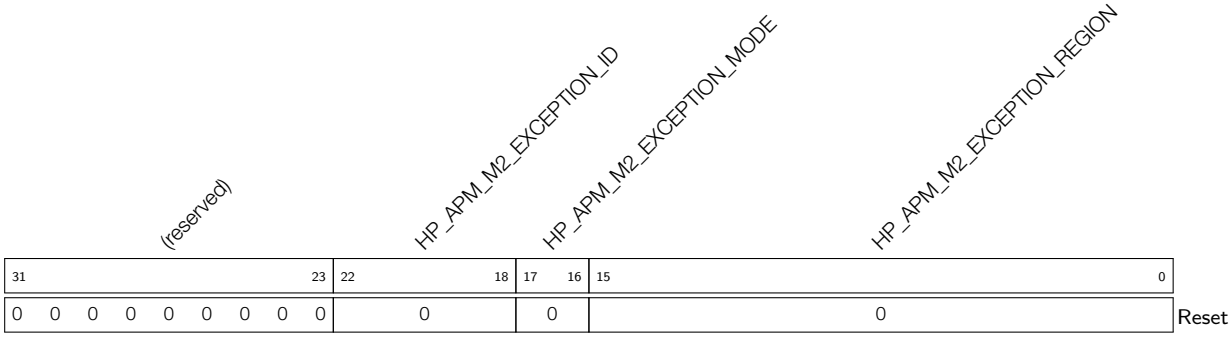
(RO)

Register 14.15. HP_APM_M2_STATUS_CLR_REG (0x00EC)



HP_APM_M2_REGION_STATUS_CLR 配置清除异常状态。(WT)

Register 14.16. HP_APM_M2_EXCEPTION_INFO0_REG (0x00F0)

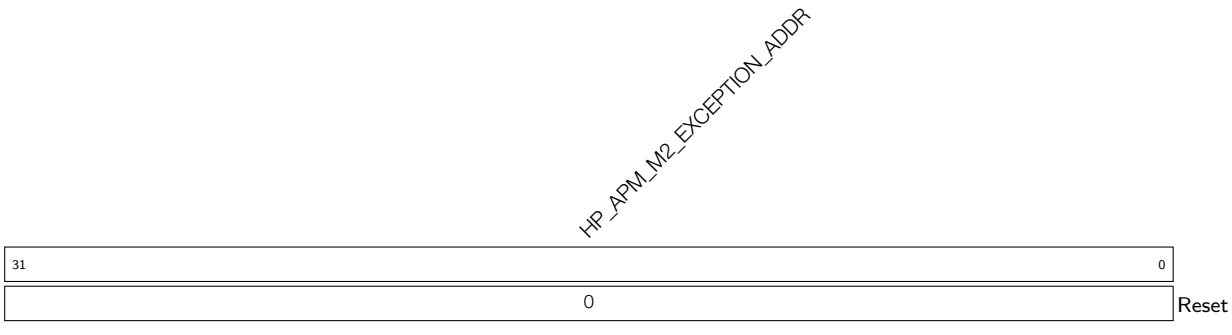


HP_APM_M2_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M2_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

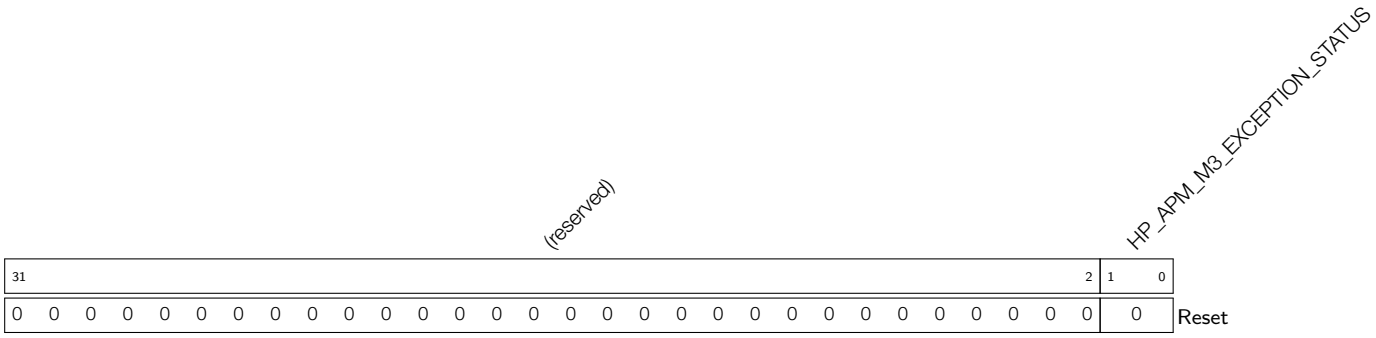
HP_APM_M2_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 14.17. HP_APM_M2_EXCEPTION_INFO1_REG (0x00F4)



HP_APM_M2_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 14.18. HP_APM_M3_STATUS_REG (0x00F8)



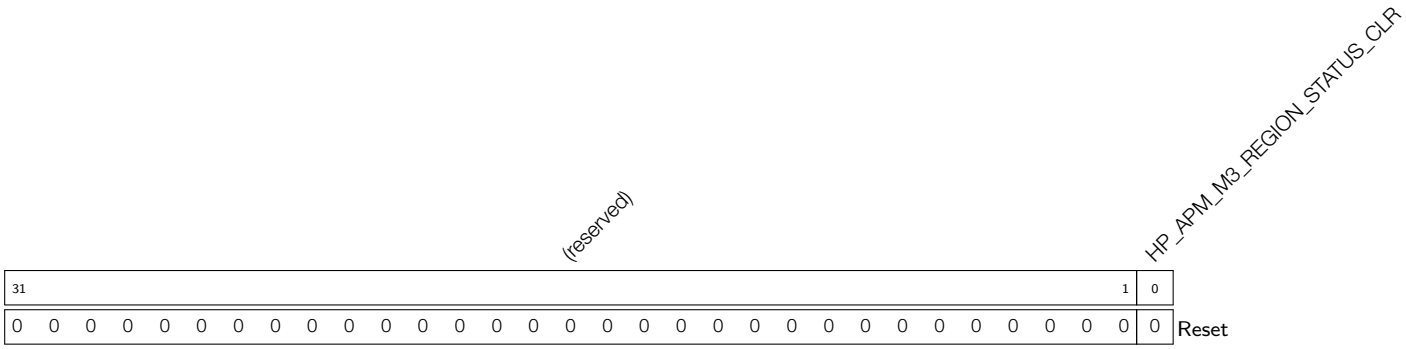
HP_APM_M3_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

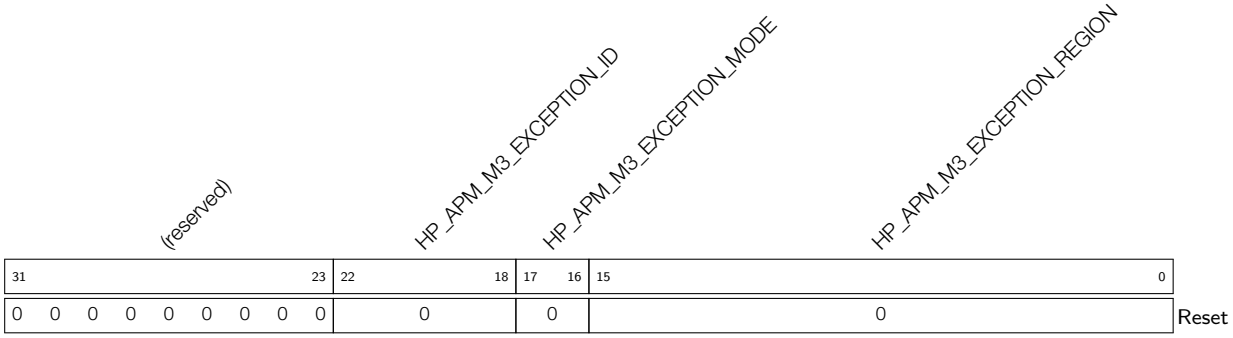
(RO)

Register 14.19. HP_APM_M3_STATUS_CLR_REG (0x00FC)



HP_APM_M3_REGION_STATUS_CLR 配置清除异常状态。(WT)

Register 14.20. HP_APM_M3_EXCEPTION_INFO0_REG (0x0100)

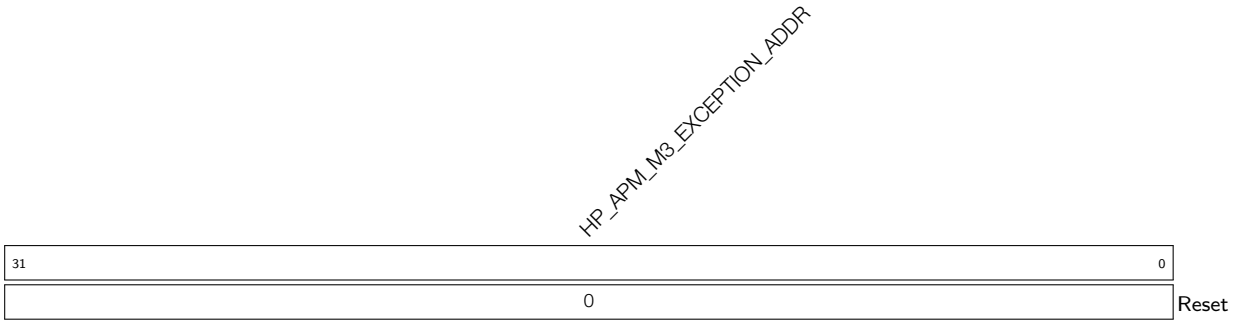


HP_APM_M3_EXCEPTION_REGION 表示异常区域。(RO)

HP_APM_M3_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

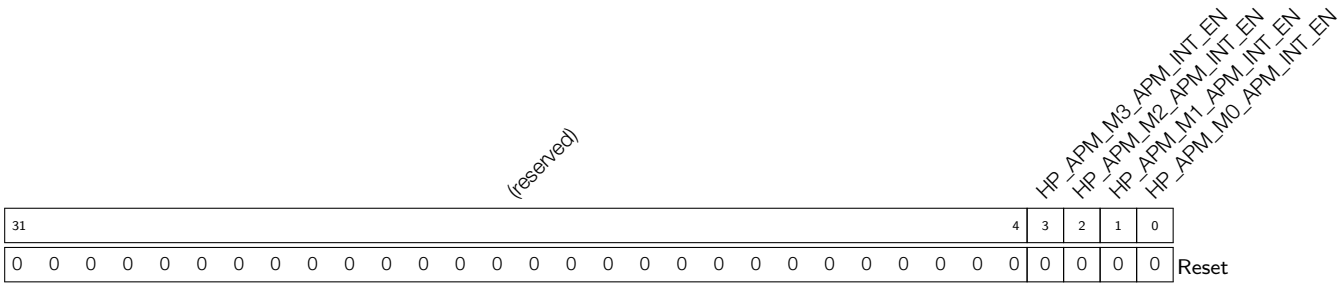
HP_APM_M3_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 14.21. HP_APM_M3_EXCEPTION_INFO1_REG (0x0104)



HP_APM_M3_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 14.22. HP_APM_INT_EN_REG (0x0108)



HP_APM_M0_APM_INT_EN 配置是否使能 HP_APM_CTRL M0 中断。

0：禁用

1：使能

(R/W)

HP_APM_M1_APM_INT_EN 配置是否使能 HP_APM_CTRL M1 中断。

0：禁用

1：使能

(R/W)

HP_APM_M2_APM_INT_EN 配置是否使能 HP_APM_CTRL M2 中断。

0：禁用

1：使能

(R/W)

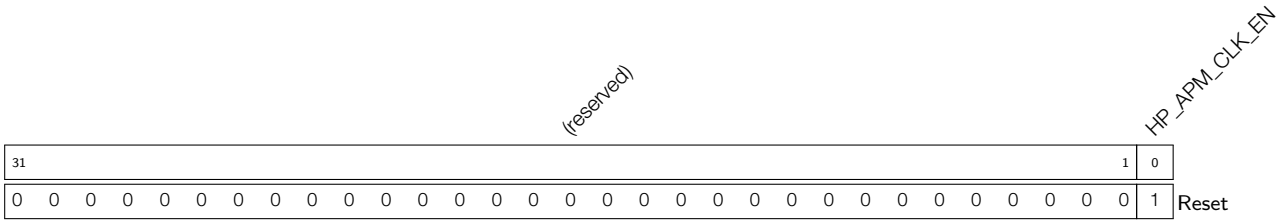
HP_APM_M3_APM_INT_EN 配置是否使能 HP_APM_CTRL M3 中断。

0：禁用

1：使能

(R/W)

Register 14.23. HP_APM_CLOCK_GATE_REG (0x010C)



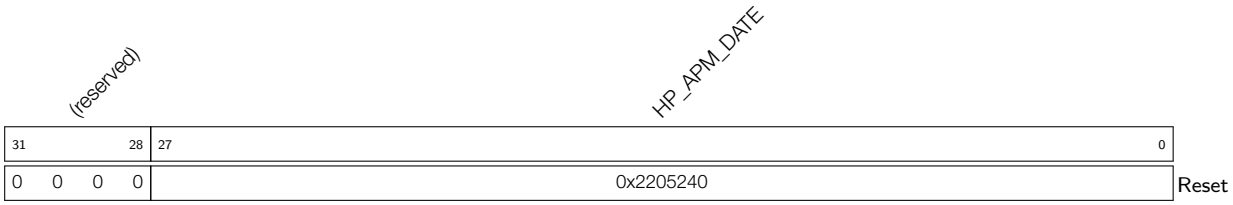
HP_APM_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

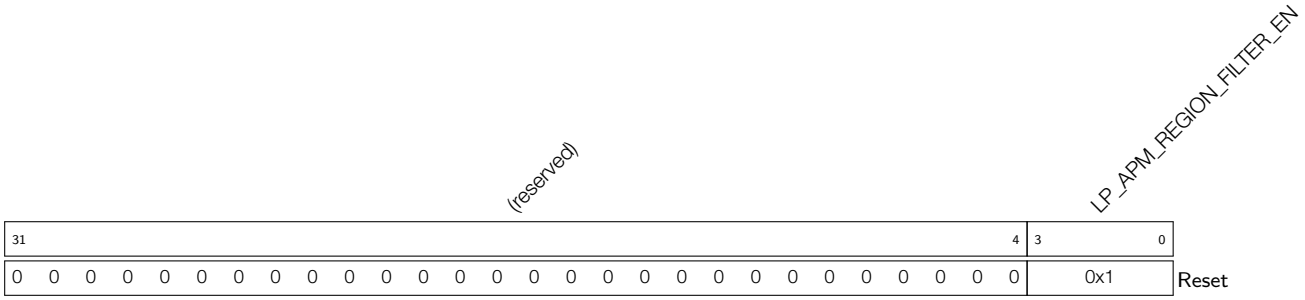
Register 14.24. HP_APM_DATE_REG (0x07FC)



HP_APM_DATE 版本控制寄存器。(R/W)

14.8.2 低功耗 APM 寄存器描述 (LP_APM_REG)

Register 14.25. LP_APM_REGION_FILTER_EN_REG (0x0000)



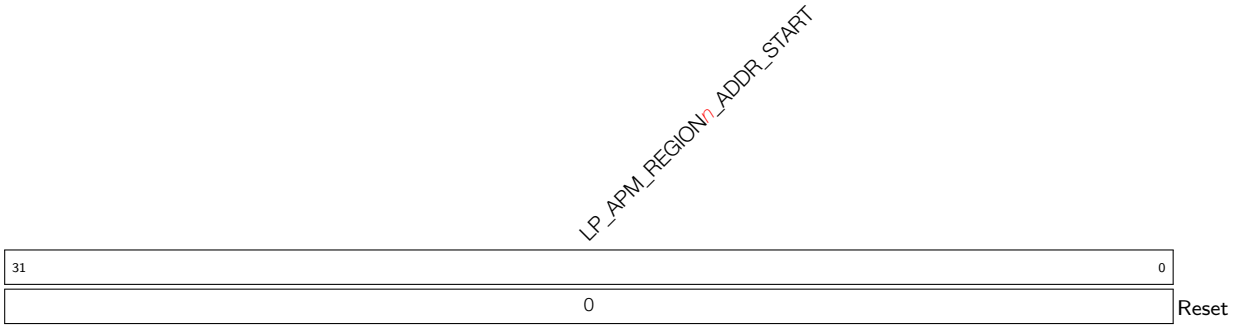
LP_APM_REGION_FILTER_EN 配置 bit *n* (0-3) 使能区域 *n* (0-3) 的地址范围。

0：禁用

1：使能

(R/W)

Register 14.26. LP_APM_REGIONn_ADDR_START_REG (n: 0-3) (0x0004+0xC*n)

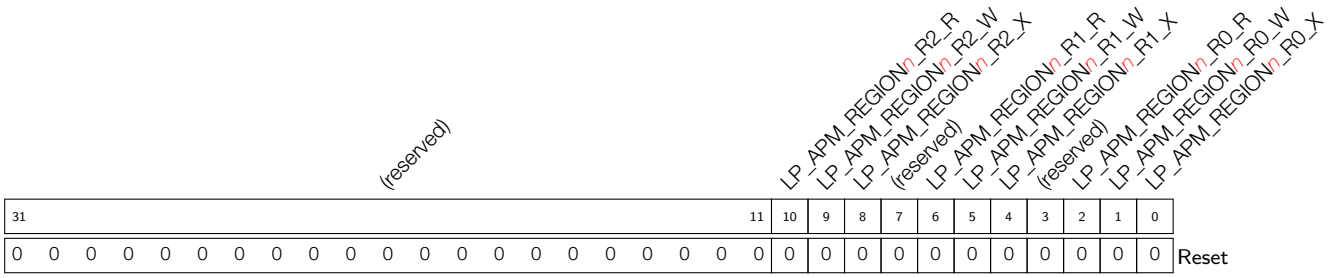


LP_APM_REGIONn_ADDR_START 配置区域 n 的起始地址。(R/W)

Register 14.27. LP_APM_REGIONn_ADDR_END_REG (n: 0-3) (0x0008+0xC*n)

LP_APM_REGIONn_ADDR_END 配置区域 n 的结束地址。(R/W)

Register 14.28. LP_APM_REGION*n*_ATTR_REG (*n*: 0-3) (0x000C+0xCn*)**



LP_APM_REGION*n*_R0_X 配置 REE0 模式下对区域 *n* 的执行权限。(R/W)

LP_APM_REGION*n*_R0_W 配置 REE0 模式下对区域 *n* 的写入权限。(R/W)

LP_APM_REGION*n*_R0_R 配置 REE0 模式下对区域 *n* 的读取权限。(R/W)

LP_APM_REGION*n*_R1_X 配置 REE1 模式下对区域 *n* 的执行权限。(R/W)

LP_APM_REGION*n*_R1_W 配置 REE1 模式下对区域 *n* 的写入权限。(R/W)

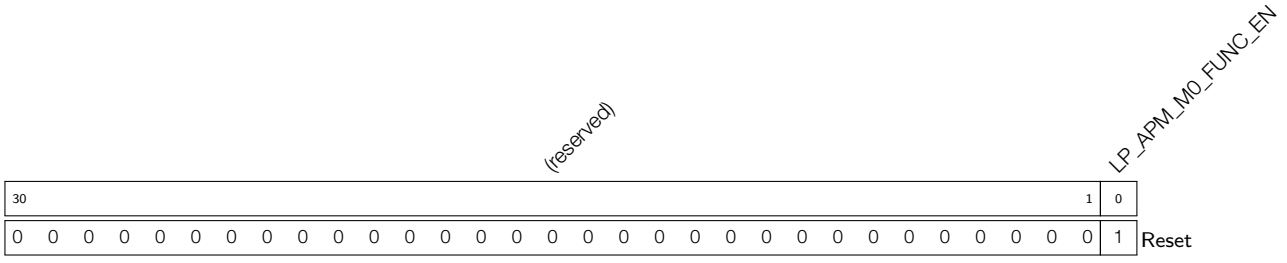
LP_APM_REGION*n*_R1_R 配置 REE1 模式下对区域 *n* 的读取权限。(R/W)

LP_APM_REGION*n*_R2_X 配置 REE2 模式下对区域 *n* 的执行权限。(R/W)

LP_APM_REGION*n*_R2_W 配置 REE2 模式下对区域 *n* 的写入权限。(R/W)

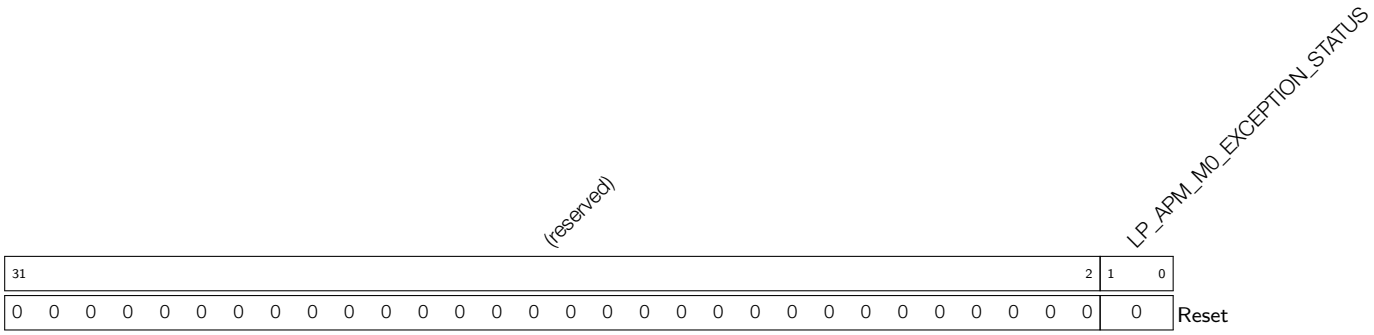
LP_APM_REGION*n*_R2_R 配置 REE2 模式下对区域 *n* 的读取权限。(R/W)

Register 14.29. LP_APM_FUNC_CTRL_REG (0x00C4)



LP_APM_M0_FUNC_EN 配置使能 LP_APM_CTRL M0 权限检查。(R/W)

Register 14.30. LP_APM_M0_STATUS_REG (0x00C8)



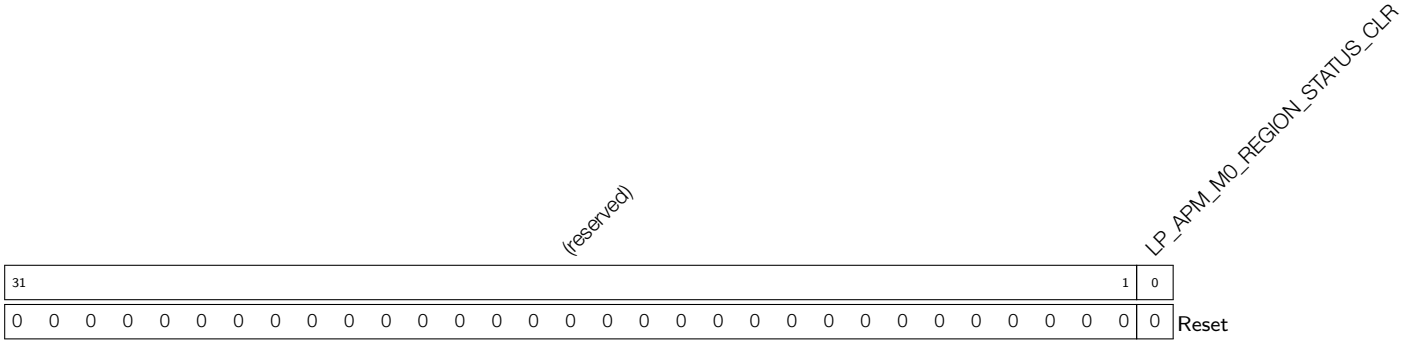
LP_APM_M0_EXCEPTION_STATUS 表示异常状态。

bit0：1 代表权限不足

bit1：1 代表地址越限

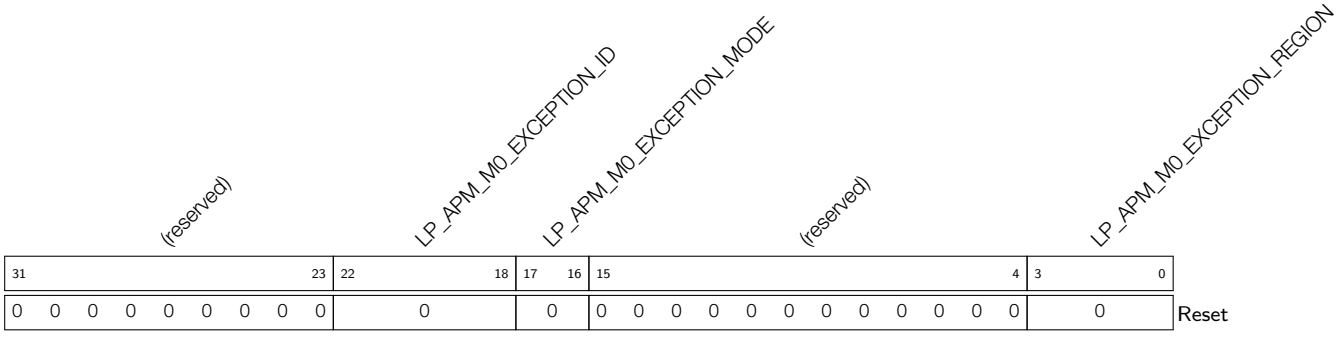
(RO)

Register 14.31. LP_APM_M0_STATUS_CLR_REG (0x00CC)



LP_APM_M0_REGION_STATUS_CLR 配置清除异常状态。(WT)

Register 14.32. LP_APM_M0_EXCEPTION_INFO0_REG (0x00D0)

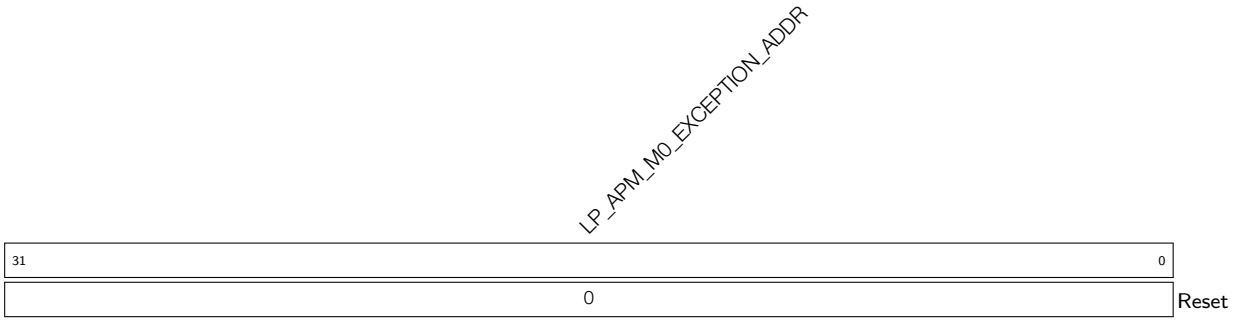


LP_APM_M0_EXCEPTION_REGION 表示异常区域。(RO)

LP_APM_M0_EXCEPTION_MODE 表示非法访问的主机的安全模式。(RO)

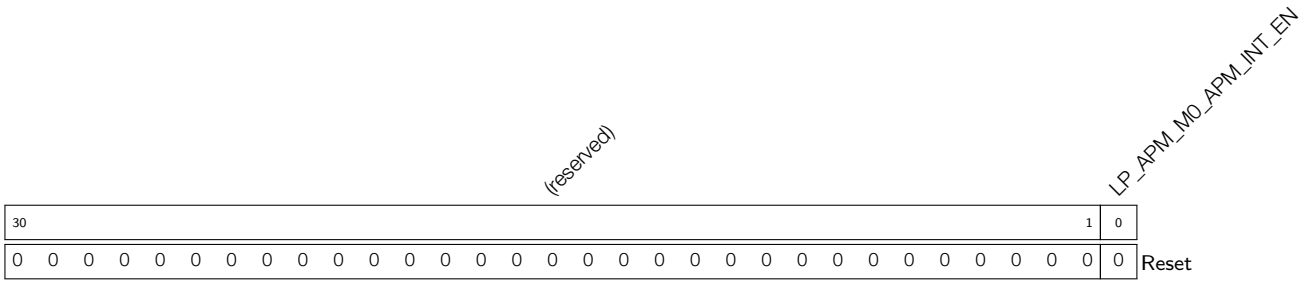
LP_APM_M0_EXCEPTION_ID 表示非法访问的主机的 ID。(RO)

Register 14.33. LP_APM_M0_EXCEPTION_INFO1_REG (0x00D4)



LP_APM_M0_EXCEPTION_ADDR 表示非法访问的地址。(RO)

Register 14.34. LP_APM_INT_EN_REG (0x00E8)



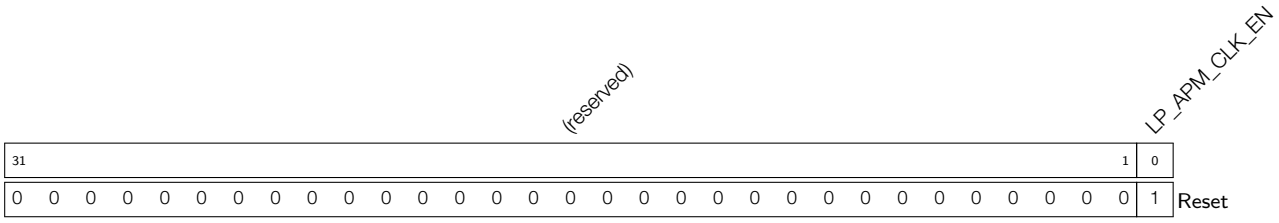
LP_APM_M0_APM_INT_EN 配置使能 LP_APM_CTRL M0 中断。

0：禁用

1：使能

(R/W)

Register 14.35. LP_APM_CLOCK_GATE_REG (0x00EC)



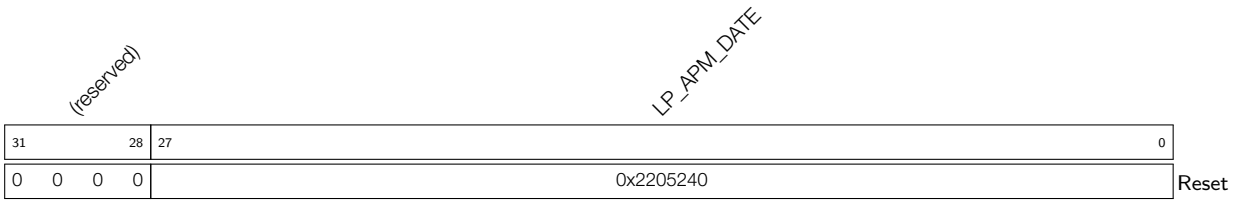
LP_APM_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

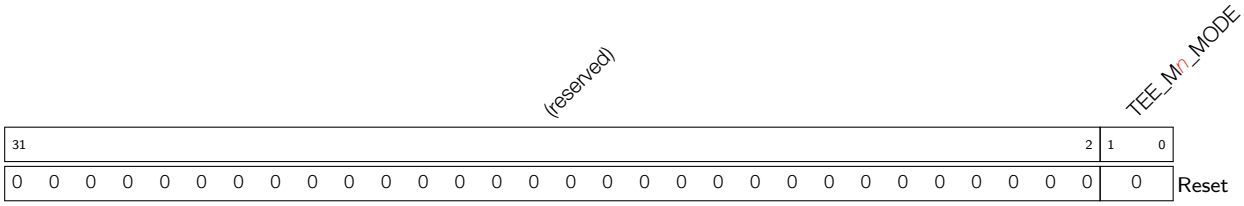
Register 14.36. LP_APM_DATE_REG (0x00FC)



LP_APM_DATE 版本控制寄存器。(R/W)

14.8.3 高性能 TEE 寄存器描述

Register 14.37. TEE_M*n*_MODE_CTRL_REG (*n*: 0-31) (0x0000+0x4**n*)



TEE_M*n*_MODE 配置主机 *n* 的安全模式。

0: TEE

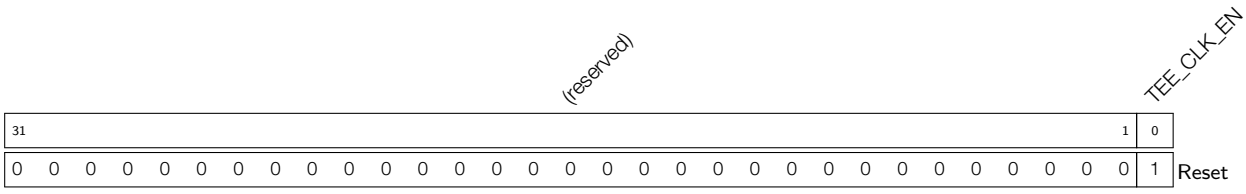
1: REE0

2: REE1

3: REE2

(R/W)

Register 14.38. TEE_CLOCK_GATE_REG (0x0080)



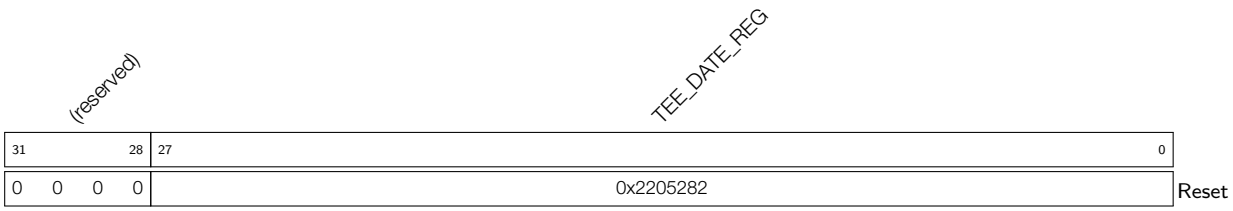
TEE_CLK_EN 配置是否保持时钟始终开启。

0：使能自动时钟门控

1：保持时钟始终开启

(R/W)

Register 14.39. TEE_DATE_REG (0x0FFC)



TEE_DATE_REG 版本控制寄存器。(R/W)

15 系统寄存器

15.1 概述

ESP32-H2 支持以下外设和核心模块：

- 外部存储器加密/解密
- 防 DPA 攻击安全
- 软件 ROM 表寄存器
- 总线超时保护

这些外设和核心模块都可以使用专门的系统寄存器进行控制。本章描述了如何配置这些寄存器。

15.2 功能描述

15.2.1 外部存储器加密/解密配置

HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 配置片外存储器的加密和解密选项。有关加密与解密模块的详细信息，请参考章节 *24 片外存储器加密与解密 (XTS_AES)*.

15.2.2 防 DPA 攻击安全控制

ESP32-H2 在硬件层面上具有防差分功耗分析 (DPA) 攻击的双重保护机制。

- 首先，ESP32-H2 在对称加密运算过程中引入了 mask 机制，通过掩盖运算过程中的真实数据来干扰功耗轨迹。这种安全机制不可关闭。
- 其次，运算时所选用的时钟会动态实时变化，模糊化运算过程中的功耗轨迹。对于此种安全机制，ESP32-H2 提供三种安全级别供用户选用，以适应不同的应用。

表 15-1. 安全等级

安全等级	对应值	96M_PLL_CLK (MHz)	64M_PLL_CLK (MHz)	XTAL_CLK (MHz)
SEC_DPA_OFF	0	96	64	32
SEC_DPA_LOW	1 或 2	(48,96][A]	(32,64][A]	(16,32][A]
SEC_DPA_HIGH	3	(32,96][A]	(21.3,64][A]	(10.6,32][A]

[A] (x,y] 表示工作频率大于 x MHz，等于或小于 y MHz。

默认情况下，寄存器 HP_SYSTEM_SEC_DPA_CONF_REG 中的字段 HP_SYSTEM_SEC_DPA_CFG_SEL 置 0。此时，安全等级由 eFuse 字段 EFUSE_SEC_DPA_LEVEL 决定。若字段 HP_SYSTEM_SEC_DPA_CFG_SEL 置 1，则安全等级由寄存器 HP_SYSTEM_SEC_DPA_CONF_REG 中的 HP_SYSTEM_SEC_DPA_CFG_LEVEL 决定。

15.2.3 软件 ROM 表寄存器

ESP32-H2 提供了两个特殊的寄存器，可以作为 ROM 的补充，用来存储一些特殊或是重要的标志值：HP_SYSTEM_ROM_TABLE_LOCK_REG 和 HP_SYSTEM_ROM_TABLE_REG。其中，HP_SYSTEM_ROM_TABLE_LOCK_REG 的值只能为 0 或 1。HP_SYSTEM_ROM_TABLE_REG 包含 32 个可用位，支持用户读取或修改。

当 HP_SYSTEM_ROM_TABLE_LOCK_REG 的值为 0 时，寄存器 HP_SYSTEM_ROM_TABLE_REG 的值可由软件重复读取或修改。当 HP_SYSTEM_ROM_TABLE_LOCK_REG 置 1 时，HP_SYSTEM_ROM_TABLE_LOCK_REG 仅支持读取，无法被修改。

需注意，内核复位将重置 HP_SYSTEM_ROM_TABLE_LOCK_REG 和 HP_SYSTEM_ROM_TABLE_REG，但 CPU 复位则无法重置这两个寄存器。有关复位类型的更多信息，请参阅章节 7 复位和时钟 中的 7.1.3 特性 小节。

15.2.4 总线超时保护

ESP32-H2 可以通过配置寄存器启用超时保护功能并配置超时阈值。当一次传输开始时，每个时钟周期都将使超时保护模块内部的计数器增加 1。当累积值小于超时阈值，并且总线收到从属设备的响应时，清除内部计数器。当累积值大于超时阈值时，如果从属设备尚未对传输作出响应，超时保护模块将强制拉高总线返回信号。同时，模块会报告中断情况，并记录异常访问地址和主站 ID。

15.2.4.1 CPU 外设超时保护寄存器

HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG 是对访问 CPU 外设寄存器的超时保护配置寄存器。CPU 外设是指地址段落在 0x600C_0000 ~ 0x600C_FFFF 区段中的外设或模块。请参考章节 4 系统和存储器 中的 4.3.5 模块/外设地址空间映射 小节，获取对应的外设信息。

发生超时异常时，会抛出 CPU_PERI_TIMEOUT_INTR 中断。

- HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG：开启超时保护，配置超时阈值。
- HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG：超时发生时，该寄存器将记录超时发生的地址。
- HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG：超时发生时，该寄存器将记录超时发生的主站 ID。

15.2.4.2 HP 外设超时保护寄存器

HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG 是对访问 HP 外设寄存器的超时保护配置寄存器。HP 外设是指地址段落在 0x6000_0000 ~ 0x6009_FFFF 区段中的外设或模块。请参考章节 4 系统和存储器 中的 4.3.5 模块/外设地址空间映射 小节，获取对应的外设信息。

发生超时异常时，会抛出 HP_PERI_TIMEOUT_INTR 中断。

- HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG：开启超时保护，配置超时阈值。
- HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG：超时发生时，该寄存器将记录超时发生的地址。
- HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG：超时发生时，该寄存器将记录超时发生的主站 ID。

15.2.4.3 LP 外设超时保护寄存器

LP_PERI_BUS_TIMEOUT_CONF_REG 是对访问 LP 外设寄存器的超时保护配置寄存器。LP 外设是指地址段落在 0x600B_0000 ~ 0x600B_FFFF 区段中的外设或模块。请参考章节 4 系统和存储器 中的 4.3.5 模块/外设地址空间映射 小节，获取对应的外设信息。

发生超时异常时，会抛出 LP_PERI_TIMEOUT_INTR 中断。

- LP_PERI_BUS_TIMEOUT_CONF_REG：开启超时保护，配置超时阈值。
- LP_PERI_BUS_TIMEOUT_ADDR_REG：超时发生时，该寄存器将记录超时发生的地址。

- LP_PERI_BUS_TIMEOUT_UID_REG：超时发生时，该寄存器将记录超时发生的主站 ID。

15.3 寄存器列表

本小节的绝大部分地址均为相对于系统寄存器基地址的地址偏移量（相对地址），与 LP Peripheral 超时相关的寄存器的地址则是相对于低功耗外设基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

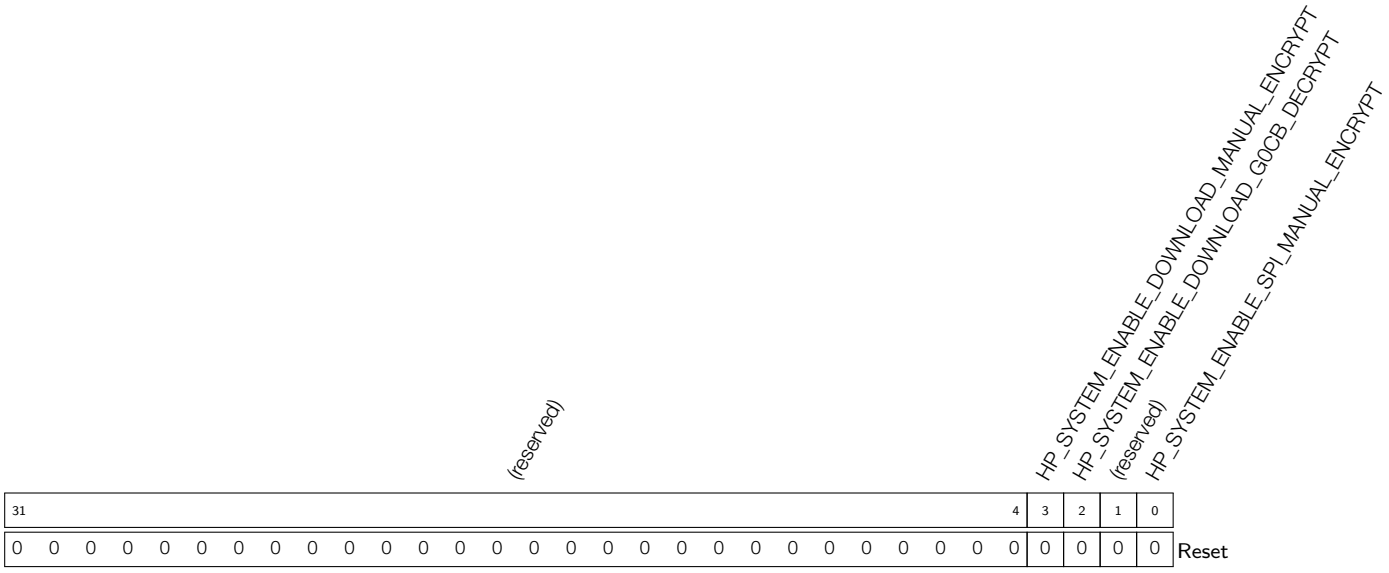
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG	外部设备加密/解密配置寄存器	0x0000	R/W
HP_SYSTEM_SEC_DPA_CONF_REG	HP 防 DPA 攻击安全配置寄存器	0x0008	R/W
HP_SYSTEM_ROM_TABLE_LOCK_REG	ROM 表锁定寄存器	0x0024	R/W
HP_SYSTEM_ROM_TABLE_REG	ROM 表寄存器	0x0028	R/W
CPU Peripheral 超时寄存器			
HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG	超时阈值、超时保护使能配置寄存器	0x000C	varies
HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG	超时访问地址信息寄存器	0x0010	RO
HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG	Master ID 和 Master 权限寄存器	0x0014	WTC
HP Peripheral 超时寄存器			
HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG	超时阈值、超时保护使能配置寄存器	0x0018	varies
HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG	超时访问地址信息寄存器	0x001C	RO
HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG	Master ID 和 Master 权限寄存器	0x0020	WTC
LP Peripheral 超时寄存器			
LP_PERI_BUS_TIMEOUT_CONF_REG	超时阈值、超时保护使能配置寄存器	0x0010	varies
LP_PERI_BUS_TIMEOUT_ADDR_REG	超时访问地址信息寄存器	0x0014	RO
LP_PERI_BUS_TIMEOUT_UID_REG	Master ID 和 Master 权限寄存器	0x0018	WTC
版本寄存器			
HP_SYSTEM_DATE_REG	日期和版本控制寄存器	0x03FC	R/W

15.4 寄存器

本小节的绝大部分地址均为相对于系统寄存器基地址的地址偏移量（相对地址），与 LP Peripheral 超时相关的寄存器的地址则是相对于低功耗外设基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 15.1. HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0000)



HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 配置是否在 SPI Boot 模式下开启 MSPI XTS 手动加密。

0：关闭

1：开启

(R/W)

HP_SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT 配置是否在下载 Boot 模式下开启 MSPI XTS 自动解密。

0：关闭

1：开启

(R/W)

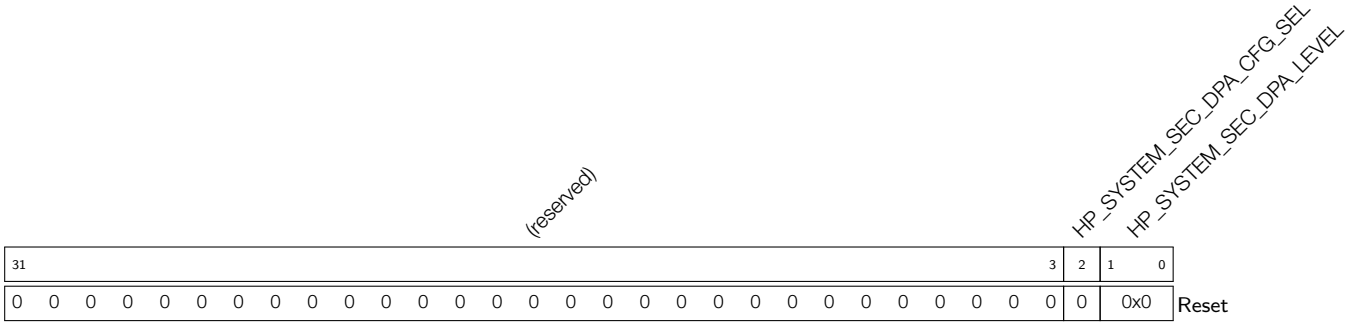
HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 配置是否在下载 Boot 模式下开启 MSPI XTS 手动加密。

0：关闭

1：开启

(R/W)

Register 15.2. HP_SYSTEM_SEC_DPA_CONF_REG (0x0008)



HP_SYSTEM_SEC_DPA_LEVEL 配置是否开启防 DPA 攻击。仅当 HP_SYSTEM_SEC_DPA_CFG_SEL 置 0 时可用。

0：关闭

1-3：开启。值越大，安全等级越高，防 DPA 攻击的能力就越强。不过，硬件加密加速器的计算开销也随之增加。

(R/W)

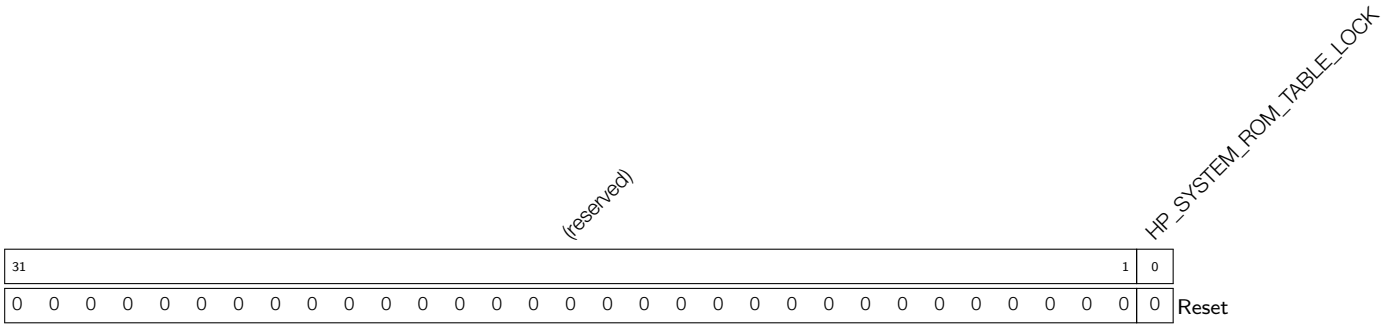
HP_SYSTEM_SEC_DPA_CFG_SEL 配置选择 HP_SYSTEM_SEC_DPA_LEVEL 或是 EFUSE_SEC_DPA_LEVEL (eFuse 寄存器) 来控制 DPA 等级。

0：选择 EFUSE_SEC_DPA_LEVEL

1：选择 HP_SYSTEM_SEC_DPA_LEVEL

(R/W)

Register 15.3. HP_SYSTEM_ROM_TABLE_LOCK_REG (0x0024)



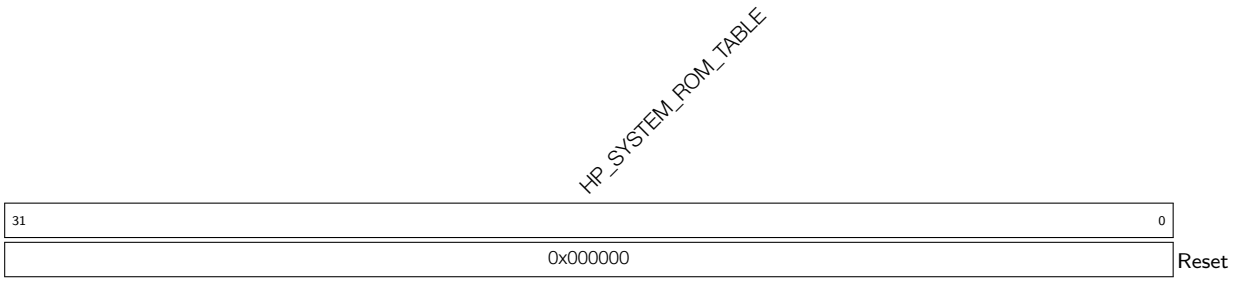
HP_SYSTEM_ROM_TABLE_LOCK 配置是否锁定 HP_SYSTEM_ROM_TABLE 中的值。

0：解锁

1：锁定

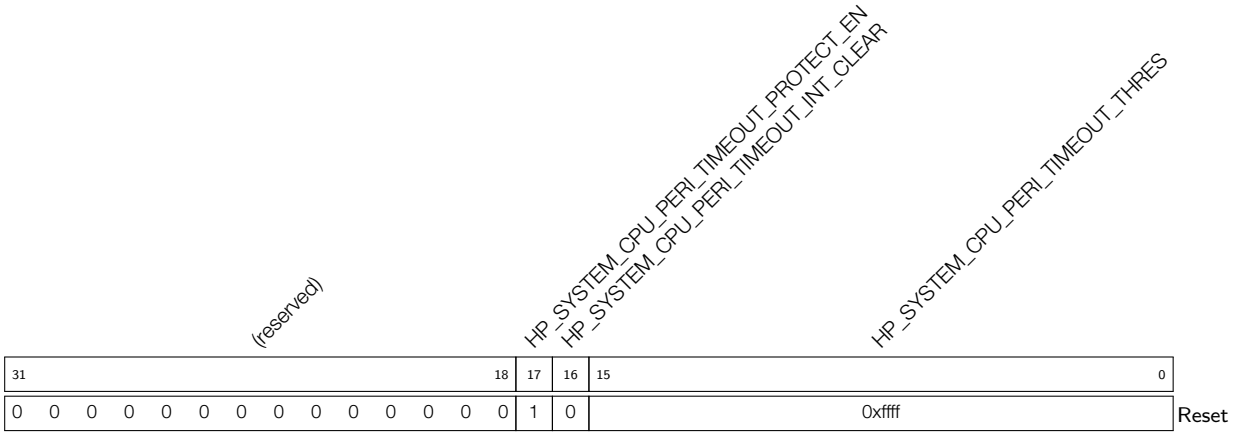
(R/W)

Register 15.4. HP_SYSTEM_ROM_TABLE_REG (0x0028)



HP_SYSTEM_ROM_TABLE 软件 ROM 表寄存器，其值仅支持在 HP_SYSTEM_ROM_TABLE_LOCK 置 0 时进行修改。(R/W)

Register 15.5. HP_SYSTEM_CPU_PERI_TIMEOUT_CONF_REG (0x000C)



HP_SYSTEM_CPU_PERI_TIMEOUT_THRES 配置总线访问 CPU 外设寄存器的超时阈值，与时钟域的时钟周期数相对应。(R/W)

HP_SYSTEM_CPU_PERI_TIMEOUT_INT_CLEAR 配置是否清除超时中断。

0：无效

1：清除超时中断

(WT)

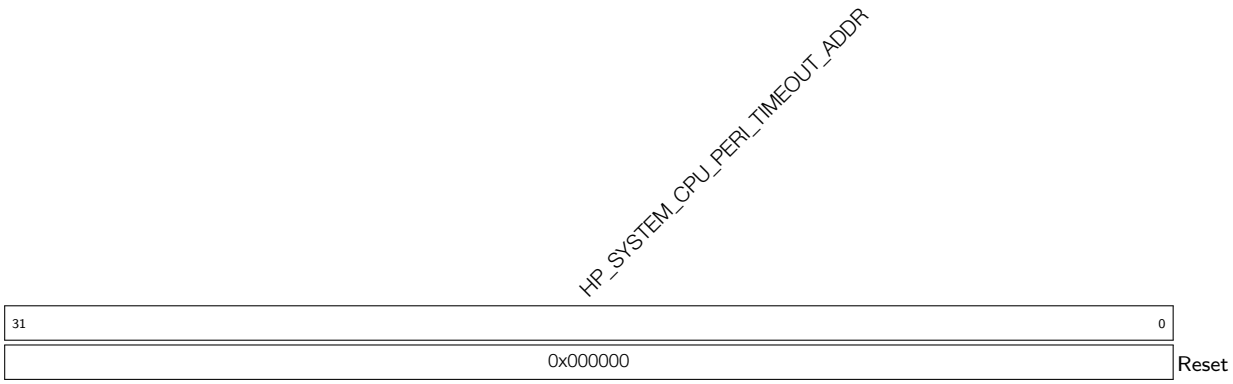
HP_SYSTEM_CPU_PERI_TIMEOUT_PROTECT_EN 配置是否启用访问 CPU 外设寄存器的超时保护。

0：关闭

1：开启

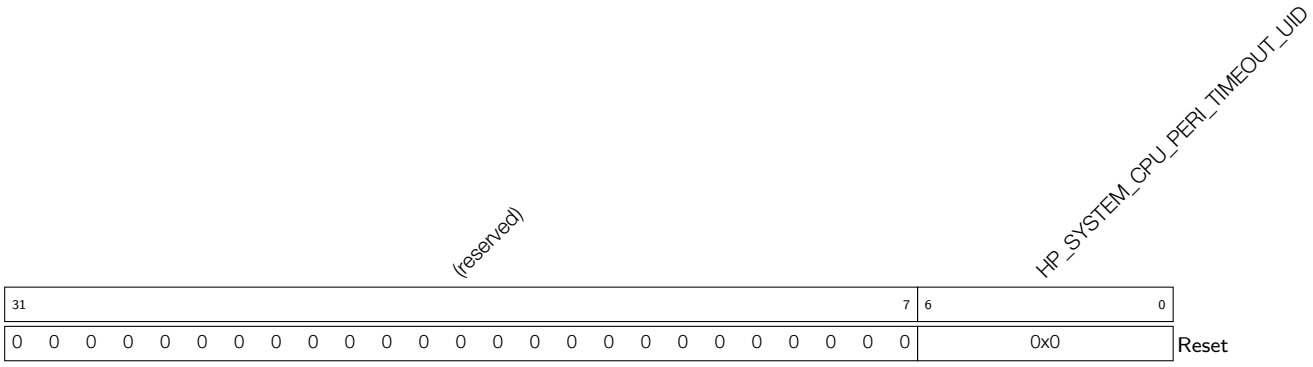
(R/W)

Register 15.6. HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR_REG (0x0010)



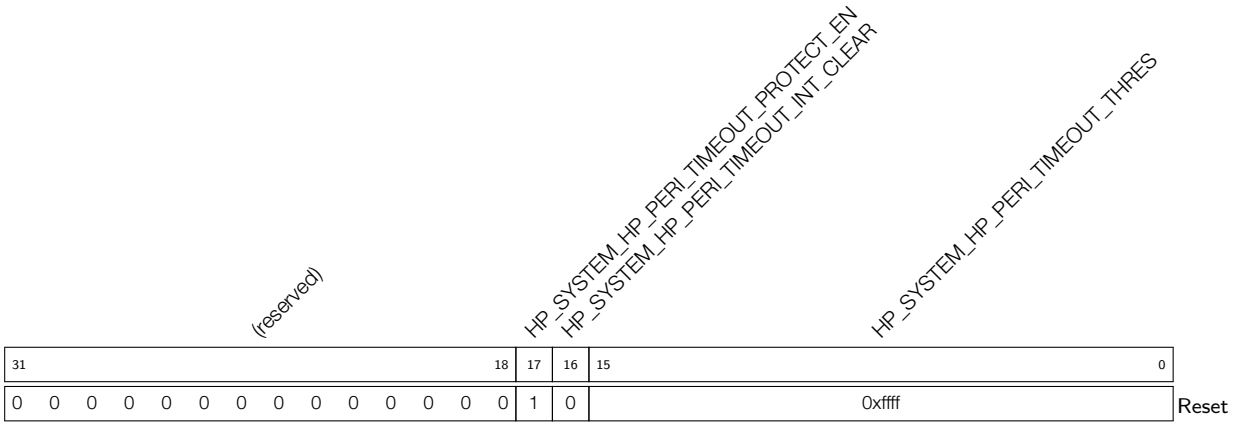
HP_SYSTEM_CPU_PERI_TIMEOUT_ADDR 表示异常访问的地址信息。(RO)

Register 15.7. HP_SYSTEM_CPU_PERI_TIMEOUT_UID_REG (0x0014)



HP_SYSTEM_CPU_PERI_TIMEOUT_UID 表示超时发生时的主站 ID [4:0] 和主站权限 [6:5]。该寄存器将在清除超时中断后清除。(WTC)

Register 15.8. HP_SYSTEM_HP_PERI_TIMEOUT_CONF_REG (0x0018)



HP_SYSTEM_HP_PERI_TIMEOUT_THRES 配置总线访问 HP 外设寄存器的超时阈值，单位为时钟域的时钟周期数。(R/W)

HP_SYSTEM_HP_PERI_TIMEOUT_INT_CLEAR 写 1 清除超时中断。(WT)

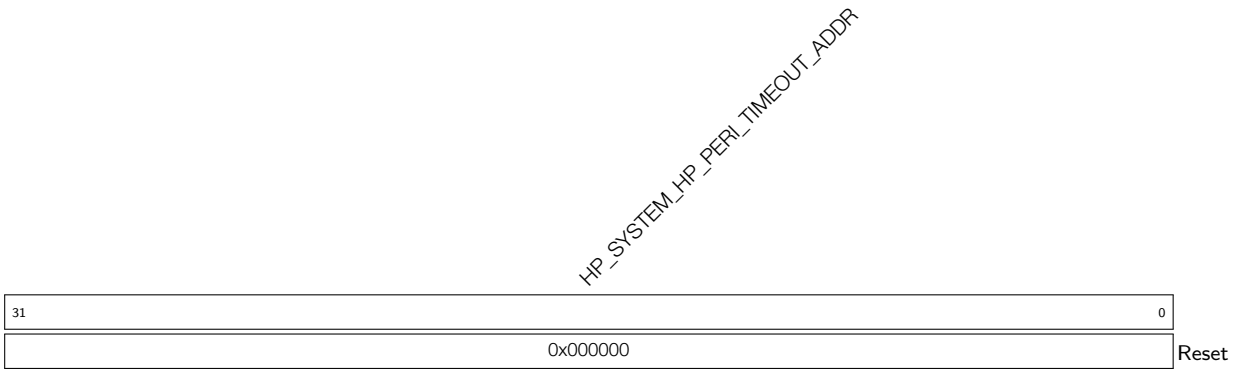
HP_SYSTEM_HP_PERI_TIMEOUT_PROTECT_EN 配置是否启用访问 HP 外设寄存器的超时保护。

0：关闭

1：开启

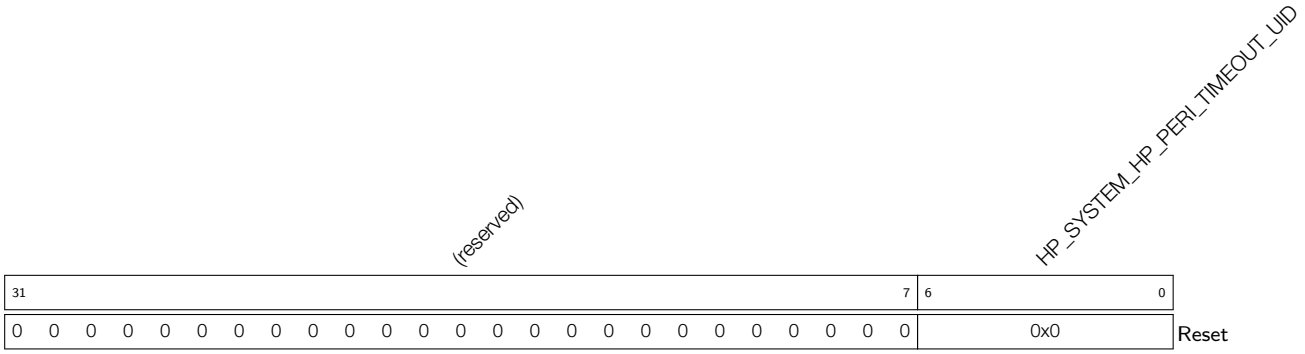
(R/W)

Register 15.9. HP_SYSTEM_HP_PERI_TIMEOUT_ADDR_REG (0x001C)



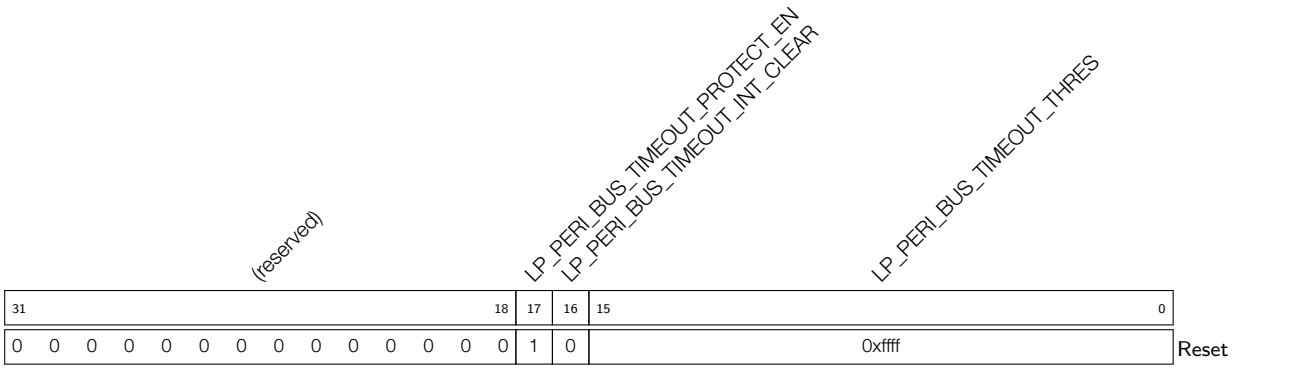
HP_SYSTEM_HP_PERI_TIMEOUT_ADDR 表示异常访问的地址信息。(RO)

Register 15.10. HP_SYSTEM_HP_PERI_TIMEOUT_UID_REG (0x0020)



HP_SYSTEM_HP_PERI_TIMEOUT_UID 表示超时发生时的主站 ID [4:0] 和主站权限 [6:5]。该寄存器将在清除超时中断后清除。(WTC)

Register 15.11. LP_PERI_BUS_TIMEOUT_CONF_REG (0x0010)



LP_PERI_BUS_TIMEOUT_THRES 配置总线访问 LP 外设寄存器的超时阈值，单位为时钟域的时钟周期数。(R/W)

LP_PERI_BUS_TIMEOUT_INT_CLEAR 写 1 清除超时中断。(WT)

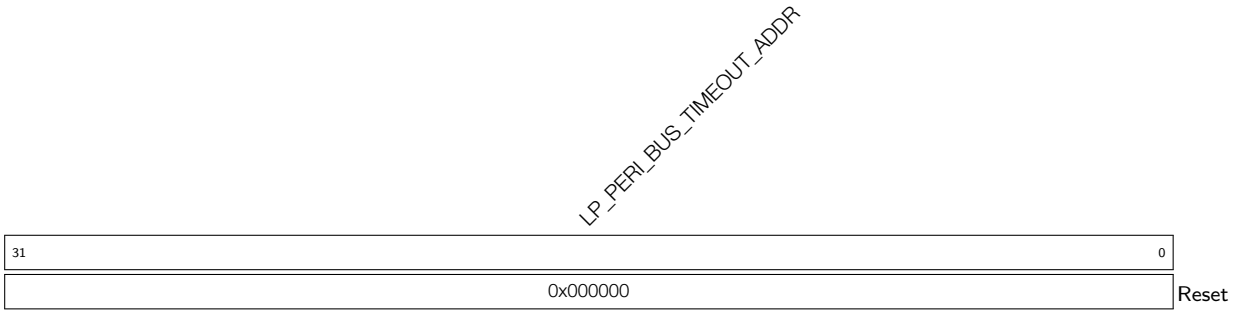
LP_PERI_BUS_TIMEOUT_PROTECT_EN 配置是否启用访问 LP 外设寄存器的超时保护。

0：关闭

1：开启

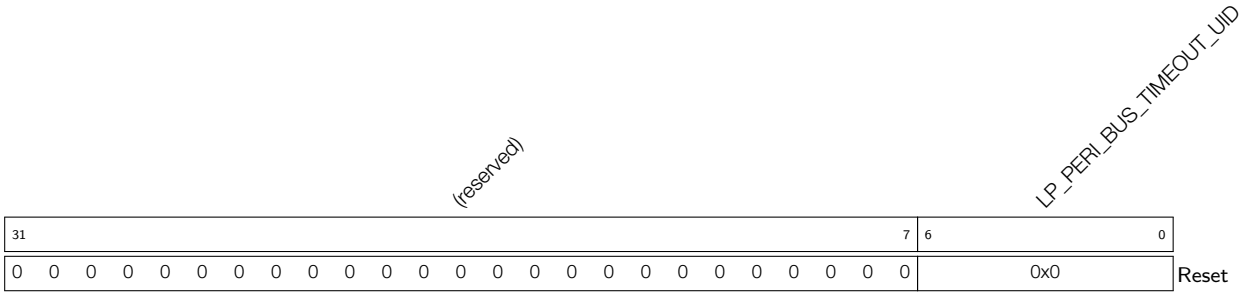
(R/W)

Register 15.12. LP_PERI_BUS_TIMEOUT_ADDR_REG (0x0014)



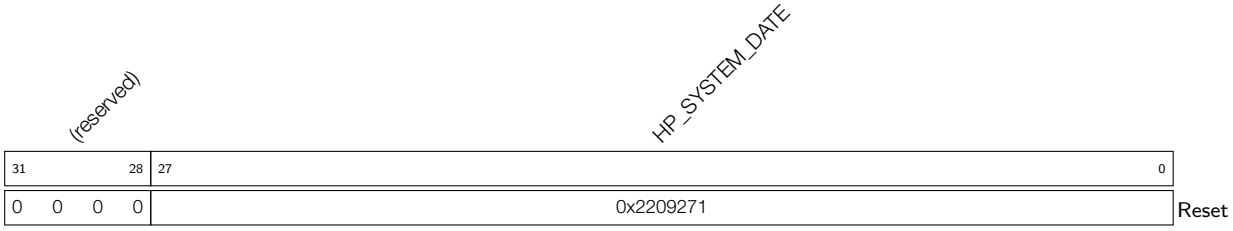
LP_PERI_BUS_TIMEOUT_ADDR 表示异常访问的地址信息。(RO)

Register 15.13. LP_PERI_BUS_TIMEOUT_UID_REG (0x0018)



LP_PERI_BUS_TIMEOUT_UID 表示超时发生时的主站 ID [4:0] 和主站权限 [6:5]。该寄存器将在清除超时中断后清除。(WTC)

Register 15.14. HP_SYSTEM_DATE_REG (0x03FC)



HP_SYSTEM_DATE 版本控制寄存器。(R/W)

16 辅助调试 (ASSIST_DEBUG，MEM_MONITOR)

16.1 概述

辅助调试模块提供一套调试功能，可用于软件开发时进行调试定位问题所在。

16.2 主要特性

- **读写监测**：监测 CPU 总线是否在限定的存储器地址范围内进行读写操作，若在该地址范围内发生读写操作则触发中断。
- **栈指针 (SP) 监测**：监测栈指针是否超出限定的范围，若超出范围则产生中断。
- **程序计数器 (PC) 记录**：记录 PC，可以获得上一次 CPU 复位时的 PC 值。
- **总线访问记录**：记录总线访问信息，当 CPU 或 DMA 写了某个特殊值时，会记录此次写操作的总线类型、地址和 PC 值 (仅记录 CPU 写操作的 PC)，并将这些信息记录到 HP SRAM 中。

16.3 功能描述

16.3.1 区域读写监测

为确认 CPU 是否在某段地址范围（即区域）进行过读写行为，辅助调试模块能够监测 CPU 的数据总线和外设总线，当总线在监测地址范围进行读写操作时会触发中断。辅助调试模块可同时监测数据总线在两段地址范围内的读写行为，假设命名为数据总线区域 0 和数据总线区域 1，外设总线也可同时监测两段地址范围，假设命名为外设总线区域 0 和外设总线区域 1。区域 0 和区域 1 根据需要进行配置。

16.3.2 栈指针监测

为防止栈溢出或者错误的压栈弹栈，辅助调试模块能够监测栈指针，当栈指针超过限定的上下边界时会记录 PC 指针并产生中断，边界由软件进行配置。

16.3.3 PC 记录

在某些时候软件开发者希望知道上次 CPU 复位时的 PC 指针。比如，在程序卡死只能复位时，开发者可能希望读取复位时的 PC 指针以便知道程序在哪里卡死，然后进行调试。辅助调试模块可以记录 CPU 复位时的 PC，方便开发者进行调试。

16.3.4 CPU/DMA 总线访问记录

辅助调试模块能够实时记录 CPU 数据总线和 DMA 总线的写行为，当总线在某个范围内发生写操作或者在某个范围内写某个特定的值时，此次操作的总线类型、地址和 PC (仅记录 CPU 写操作的 PC) 等信息会被记录下来，并按照一定的格式存储到 HP SRAM 中。

16.4 工作流程

16.4.1 区域读写监测和栈监测配置

区域监测可监测 CPU 的数据总线和外设总线的读写行为，每个总线可同时监测两个区域。详细监测模式如下。

- 数据总线读写监测
  - 监测数据总线在区域 0 是否进行读操作
  - 监测数据总线在区域 0 是否进行写操作
  - 监测数据总线在区域 1 是否进行读操作
  - 监测数据总线在区域 1 是否进行写操作
- 外设总线读写监测
  - 监测外设总线在区域 0 是否进行读操作
  - 监测外设总线在区域 0 是否进行写操作
  - 监测外设总线在区域 1 是否进行读操作
  - 监测外设总线在区域 1 是否进行写操作
- 栈指针监测
  - 监测栈指针是否大于栈指针监测区域的结束地址
  - 监测栈指针是否小于栈指针监测区域的起始地址

区域监测和栈监测的配置流程如下：

1. 配置监测区域和栈指针
   - 数据总线区域 0 由 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG 进行配置。
   - 数据总线区域 1 由 ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG 进行配置。
   - 外设总线区域 0 由 ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG 进行配置。
   - 外设总线区域 1 由 ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG 和 ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG 进行配置。
   - 栈指针监测区域由 ASSIST_DEBUG_CORE_0_SP_MIN_REG 和 ASSIST_DEBUG_CORE_0_SP_MAX_REG 进行配置。
2. 配置中断
   - 配置 ASSIST_DEBUG_CORE_0_INTR_ENA_REG 用于使能不同模式的中断。
   - 配置 ASSIST_DEBUG_CORE_0_INTR_RAW_REG 用于查询不同模式的中断状态。
   - 配置 ASSIST_DEBUG_CORE_0_INTR_CLR_REG 用于清除不同模式的中断。
3. 配置 ASSIST_DEBUG_CORE_0_MONTR_ENA_REG 使能不同的监测模式，可同时使能。

比如，若想监测数据总线地址 A 到地址 B 范围内是否进行过写操作，可通过数据总线区域 0 或者区域 1 进行监测，以区域 0 为例：

1. 配置 ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 为 1 ，使能 CPU 更新 PC 信号给辅助调试模块。
2. 配置 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG 为 A 地址。
3. 配置 ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG 为 B 地址。

4. 配置 ASSIST_DEBUG_CORE_0_INTR_ENA_REG bit[1] 使能数据总线区域 0 写监测中断。
5. 配置 ASSIST_DEBUG_CORE_0_MONTR_ENA_REG bit[1] 使能数据总线区域 0 写监测。
6. 配置中断矩阵将 ASSIST_DEBUG_INT 映射到 CPU 中断上（参考章节 9 中断矩阵 *(INTMTX)*）。
7. 发生中断后
   - 读取 ASSIST_DEBUG_CORE_0_INTR_RAW_REG 确认是什么操作触发的中断。
   - 如果是区域监测引发的中断，读取 ASSIST_DEBUG_CORE_0_AREA_PC 可获取触发中断时刻的 PC 值，读取 ASSIST_DEBUG_CORE_0_AREA_SP可获取触发中断时刻的 SP 值。
   - 如果是栈监测引发的中断，读取 ASSIST_DEBUG_CORE_0_SP_PC 可获取触发中断时刻的 PC 值。
   - 配置 ASSIST_DEBUG_CORE_0_INTR_CLR_REG 不同位可清除不同的中断。

16.4.2 PC 记录配置

配置 ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 为 1 ，使能 CPU 更新 PC 信号给辅助调试模块。同时当 ASSIST_DEBUG_CORE_0_RCD_RECORDEN 配置为 1 时，
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 会去记录 CPU PC 信号，同时
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG 会记录 SP 值，否则保持原值。

当 CPU 发生复位时，ASSIST_DEBUG_CORE_0_RCD_EN_REG 会被复位，但是
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 和 ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG 不会被复位，因此这两个寄存器会一直保持复位时刻的 PC 值和 SP 值。

16.4.3 CPU/DMA 总线访问记录配置

总线访问记录的配置流程如下：

1. 配置监测地址范围
   - 配置 MEM_MONITOR_LOG_MIN_REG 和 MEM_MONITOR_LOG_MAX_REG 确定地址范围。
2. 配置监测模式 (MEM_MONITOR_LOG_MODE)
   - 写监测（监测总线是否发生写操作）
   - 字 (word) 监测（监测总线是否写某个特定 word）
   - 半字 (halfword) 监测（监测总线是否写某个特定 halfword）
   - 字节 (byte) 监测（监测总线是否写某个特定 byte）
3. 配置需要监测的特殊值
   - 监测模式为 word 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG 即为监测的特定 word。
   - 监测模式为 halfword 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG 的 [15:0] 即为监测的特定 halfword。
   - 监测模式为 byte 监测时，MEM_MONITOR_LOG_CHECK_DATA_REG 的 [7:0] 即为监测的特定 byte。
   - MEM_MONITOR_LOG_DATA_MASK_REG 用于屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 的相应 byte。当某一 byte 被屏蔽，表示该 byte 可为任意的值，比如 word 监测，
     MEM_MONITOR_LOG_CHECK_DATA_REG 配置为 0x01020304，
     MEM_MONITOR_LOG_DATA_MASK_REG 配置为 0x1，因此只要总线写 0x010203XX 都会被记录。

4. 配置记录信息的存储地址范围

   - MEM_MONITOR_LOG_MEM_START_REG 和 MEM_MONITOR_LOG_MEM_END_REG 为存储地址范围配置寄存器。记录信息的存储地址范围为 0x4080_0000 ~ 0x4084_FFFF。
   - 置位 MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG，使 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 更新为 MEM_MONITOR_LOG_MEM_START_REG。
   - 配置辅助调试模块对 HP SRAM 的使用权限，只有开启了辅助调试模块对 HP SRAM 的使用权限才能访问 HP SRAM，辅助调试模块对 HP SRAM 的使用权限默认关闭。关于如何配置，详情见章节 14 访问权限管理 (APM)。

5. 配置存储记录信息的内存的存储模式：loop 模式和非 loop 模式

   - loop 模式下，将记录信息循环写进配置地址范围内，当写到结束地址时回到起始地址开始写，覆盖之前记录的信息。置位 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使用 loop 模式。
     例如，存储记录信息的地址范围为 0 ~ 4 ，总线访问过程中有 1 ~ 10 共十次写记录信息的操作，那么第 5 次写操作写到地址 4 后，第 6 次写操作会回到地址 0 开始写，第 6 ~ 10 写操作会覆盖之前的第 1 ~ 5 写操作的数据。
   - 非 loop 模式下，当将记录信息写到结束地址后，会一直停留在结束地址，并不再写记录信息，不会覆盖最开始的记录信息，并且丢弃后面的记录信息。清零 MEM_MONITOR_LOG_MEM_LOOP_ENABLE 使用非 loop 模式。
     例如，存储记录信息的地址范围为 0 ~ 4 ，总线访问过程中有 1 ~ 10 共十次写记录信息的操作，当第 5 次写操作写到地址 4 后，此后所有的记录信息都会被舍弃。

6. 配置总线使能

   - 配置 MEM_MONITOR_LOG_ENA 使能 CPU 或 DMA 总线访问记录，可同时使能。

辅助调试模块会先将记录信息缓存到一个内部 buffer 里，然后从 buffer 取出记录信息再存储到配置的存储区域内。当监测行为被连续触发时，连续产生的记录数据包可能会使得 buffer 满而不能缓存新的记录数据包，此时模块会舍弃掉不能及时缓存到 buffer 的记录数据包，并在 buffer 不满的时候缓存一个 LOST 数据包作为替代，此时不能知道舍弃的记录数据包的总线类型，以及不能知道舍弃了多少个记录数据包。

当总线访问记录结束后需要从内存中读取记录数据并进行解码。记录数据里只有三种包格式，CPU 数据包、DMA 数据包和 LOST 数据包，前两种数据包对应 CPU 数据总线记录信息和 DMA 总线访问记录信息，三种包格式如表 16-1、16-2 和 16-3 所示：

表 16-1. CPU 包格式

Bit[63:34]	Bit[33:32]	Bit[31:4]	Bit[3:2]	Bit[1:0]
pc_offset	anchored(2)	addr_offset	format	anchored(1)

表 16-2. DMA 包格式

Bit[31:9]	Bit[8:4]	Bit[3:2]	Bit[1:0]
addr_offset	dma_source	format	anchored(1)

表 16-3. LOST 包格式

Bit[31:4]	Bit[3:2]	Bit[1:0]

reserved	format	anchored(1)

从包格式可以看出，CPU 数据包共 64 位，DMA 数据包共 32 位，LOST 数据包共 32 位。下面介绍各个域的含义：

- **format** 表示此次数据包的类型，0 表示 CPU 数据包，1 表示 DMA 数据包，3 表示 LOST 数据包。
- **pc_offset** 记录了 CPU 的 PC 指针的偏移量，实际 PC = pc_offset + 0x4000_0000。
- **addr_offset** 记录了此次写操作的地址偏移，实际地址 = addr_offset + MEM_MONITOR_LOG_MIN_REG。
- **dma_source** 记录了哪一个外设发起的 DMA 访问，具体见章节 14 访问权限管理 *(APM)* > 表14-4。
- **anchored** 表示该 32 位数据在数据包中的位置，1 表示该 32 位数据是数据包的低 32 位，2 表示该 32 位数据是数据包的高 32 位。

模块内部 buffer 的数据宽度是 32 比特。当同时使能 CPU 和 DMA 总线访问记录并且在某一时刻同时生成了记录数据，先将 DMA 的数据包缓存到 buffer 中，再缓存 CPU 的数据包。辅助调试模块会自动地将缓存数据从 buffer 取出，并将取出的数据再以 32 比特数据宽度存入指定范围的存储器中。

当采用 loop 模式时，若在配置的存储地址范围内循环了若干次，可能存在残留数据干扰解析，即 CPU 数据包的低 32 比特数据被覆盖而使得高 32 比特数据成为残留数据，因此确定第一个有效数据包的位置时必须过滤可能存在的残留数据。使用 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG 确定了数据包的起始位置之后，检查数据包 anchored 比特位的数值，是 1 则保留，是 2 则舍弃。

数据包解析流程如下：

- MEM_MONITOR_LOG_MEM_FULL_FLAG 确定数据是否溢出配置范围：
  - 如果未溢出，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4 ；
  - 若溢出并且开启了 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG ~ MEM_MONITOR_LOG_MEM_END_REG 和MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG - 4；
  - 若溢出并且未开启 loop 模式，则数据读取的地址范围为 MEM_MONITOR_LOG_MEM_START_REG ~ MEM_MONITOR_LOG_MEM_END_REG。
- 从起始地址处开始读取数据并解析，每次读取 32 位。

数据解析完成之后需要通过配置 MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 清除 MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

16.5 寄存器列表

本小节中**总线记录配置寄存器**的地址为相对于 **MEM_MONITOR** 基地址的地址偏移量（相对地址），详见 16.5.1，其它寄存器的所有地址均为相对于 **ASSIST_DEBUG** 基地址的地址偏移量（相对地址），详见 16.5.2，具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

16.5.1 总线记录配置寄存器列表

名称	描述	地址	访问
总线记录配置寄存器			
MEM_MONITOR_LOG_SETTING_REG	总线访问记录配置寄存器	0x0000	R/W
MEM_MONITOR_LOG_CHECK_DATA_REG	总线访问监测数据配置寄存器	0x0004	R/W
MEM_MONITOR_LOG_DATA_MASK_REG	总线访问监测数据屏蔽配置寄存器	0x0008	R/W
MEM_MONITOR_LOG_MIN_REG	总线访问监测范围配置寄存器	0x000C	R/W
MEM_MONITOR_LOG_MAX_REG	总线访问监测范围配置寄存器	0x0010	R/W
MEM_MONITOR_LOG_MEM_START_REG	记录数据写入内存的起始地址	0x0014	R/W
MEM_MONITOR_LOG_MEM_END_REG	记录数据写入内存的结束地址	0x0018	R/W
MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG	表示下一次写内存的写地址	0x001C	RO
MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG	更新下一次写内存的写地址为记录数据写入内存的起始地址	0x0020	R/W
MEM_MONITOR_LOG_MEM_FULL_FLAG_REG	记录溢出状态寄存器	0x0024	varies
时钟控制寄存器			
MEM_MONITOR_CLOCK_GATE_REG	寄存器时钟控制	0x0028	R/W
版本寄存器			
MEM_MONITOR_DATE_REG	版本寄存器	0x03FC	R/W

16.5.2 其它寄存器列表

名称	描述	地址	访问
监测配置寄存器			
ASSIST_DEBUG_CORE_0_MONTR_ENA_REG	监测使能配置寄存器	0x0000	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG	数据总线区域 0 的起始地址配置寄存器	0x0010	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG	数据总线区域 0 的结束地址配置寄存器	0x0014	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG	数据总线区域 1 的起始地址配置寄存器	0x0018	R/W
ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG	数据总线区域 1 的结束地址配置寄存器	0x001C	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG	外设总线区域 0 的起始地址配置寄存器	0x0020	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG	外设总线区域 0 的结束地址配置寄存器	0x0024	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG	外设总线区域 1 的起始地址配置寄存器	0x0028	R/W
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG	外设总线区域 1 的结束地址配置寄存器	0x002C	R/W
ASSIST_DEBUG_CORE_0_AREA_PC_REG	区域监测 CPU PC 状态寄存器	0x0030	RO
ASSIST_DEBUG_CORE_0_AREA_SP_REG	区域监测 CPU SP 状态寄存器	0x0034	RO

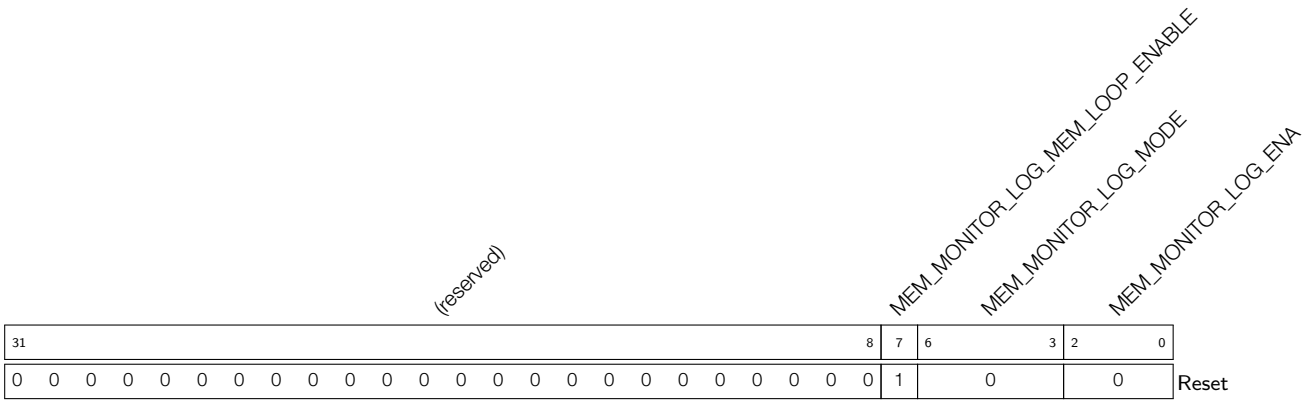
名称	描述	地址	访问
ASSIST_DEBUG_CORE_0_SP_MIN_REG	栈监测区域的起始地址配置寄存器	0x0038	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	栈监测区域的结束地址配置寄存器	0x003C	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	栈监测 CPU PC 状态寄存器	0x0040	RO
中断配置寄存器			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	中断状态寄存器	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_ENA_REG	中断使能配置寄存器	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	清除中断配置寄存器	0x000C	R/W
PC 记录配置寄存器			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	CPU PC 记录使能配置寄存器	0x0044	R/W
PC 记录状态寄存器			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	PC 记录寄存器	0x0048	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	SP 记录寄存器	0x004C	RO
CPU 状态寄存器			
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	CPU 异常前的最后一条指令的 PC	0x0070	RO
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	CPU 处于调试模式的标志寄存器	0x0074	RO
时钟控制寄存器			
ASSIST_DEBUG_CLOCK_GATE_REG	寄存器时钟控制	0x0078	R/W
版本寄存器			
ASSIST_DEBUG_DATE_REG	版本寄存器	0x03FC	R/W

16.6 寄存器

本小节中**总线记录配置寄存器**的地址为相对于 **MEM_MONITOR** 基地址的地址偏移量（相对地址），详见 16.6.1，其它寄存器的所有地址均为相对于 **ASSIST_DEBUG** 基地址的地址偏移量（相对地址），详见 16.6.2，具体基地址请见章节 4 *系统和存储器* 中的表 4-2。

16.6.1 总线记录配置寄存器

Register 16.1. MEM_MONITOR_LOG_SETTING_REG (0x0000)



MEM_MONITOR_LOG_ENA 配置是否使能 CPU 或 DMA 总线访问记录。

bit[0]：配置是否使能 CPU 总线访问记录

0：不使能

1：使能

bit[1]：保留

bit[2]：配置是否使能 DMA 总线访问记录

0：不使能

1：使能

(R/W)

MEM_MONITOR_LOG_MODE 配置监测模式。

bit[0]：写监测

0：不使能

1：使能

bit[1]：word 监测

0：不使能

1：使能

bit[2]：halfword 监测

0：不使能

1：使能

bit[3]：byte 监测

0：不使能

1：使能

(R/W)

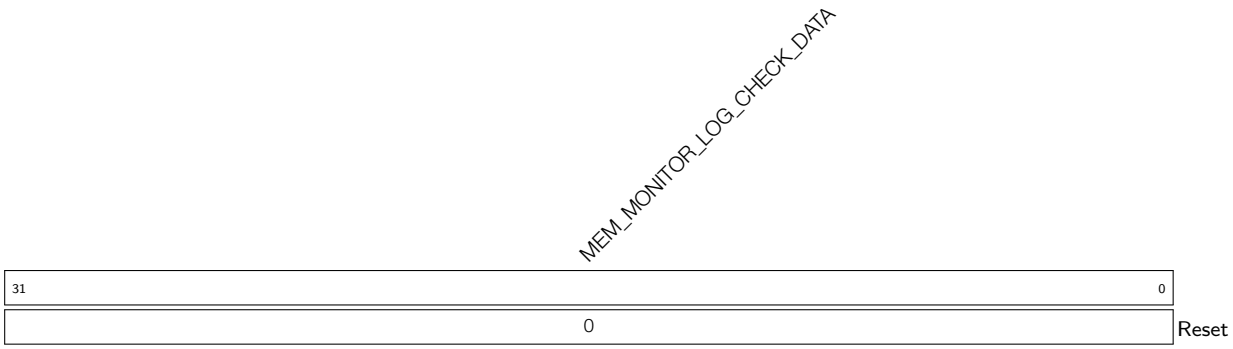
MEM_MONITOR_LOG_MEM_LOOP_ENABLE 配置写内存的存储模式。

1：loop 模式

0：非 loop 模式

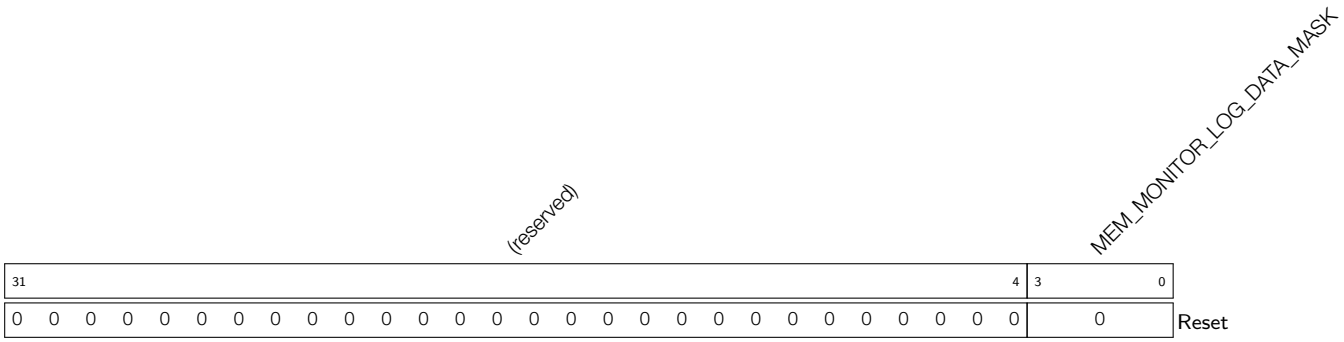
(R/W)

Register 16.2. MEM_MONITOR_LOG_CHECK_DATA_REG (0x0004)



MEM_MONITOR_LOG_CHECK_DATA 配置总线访问监测的特殊值。(R/W)

Register 16.3. MEM_MONITOR_LOG_DATA_MASK_REG (0x0008)



MEM_MONITOR_LOG_DATA_MASK 是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 的相应字节。

bit[0]：是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 最低位字节。

0：不屏蔽

1：屏蔽

bit[1]：是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 第二个低位字节。

0：不屏蔽

1：屏蔽

bit[2]：是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 第二个高位字节。

0：不屏蔽

1：屏蔽

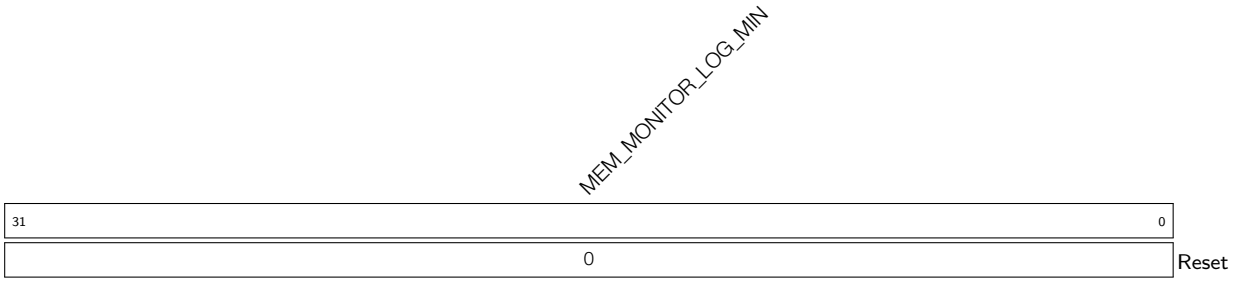
bit[3]：是否屏蔽 MEM_MONITOR_LOG_CHECK_DATA_REG 最高位字节。

0：不屏蔽

1：屏蔽

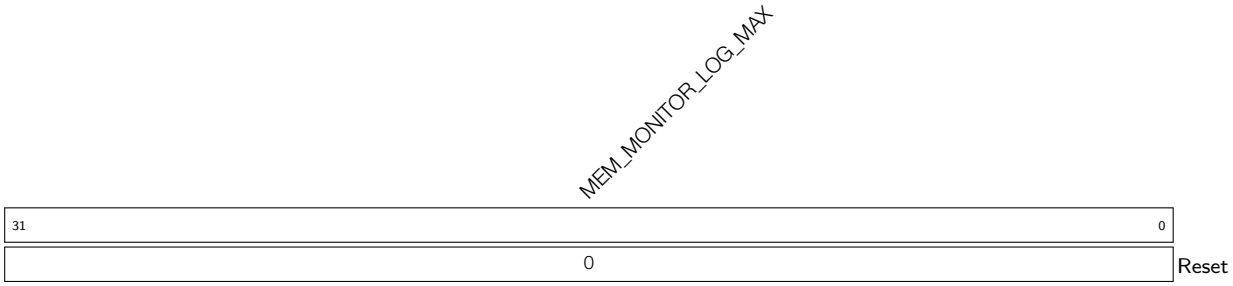
(R/W)

Register 16.4. MEM_MONITOR_LOG_MIN_REG (0x000C)



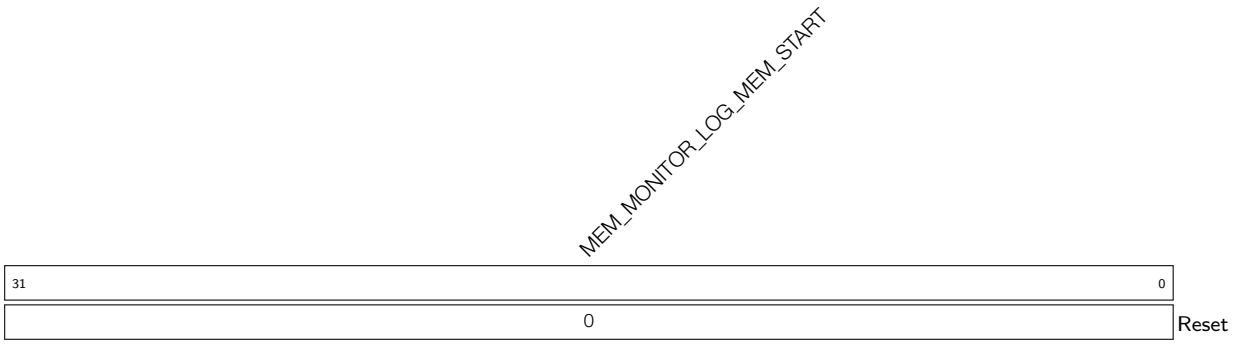
MEM_MONITOR_LOG_MIN 配置监测区域的起始地址。(R/W)

Register 16.5. MEM_MONITOR_LOG_MAX_REG (0x0010)



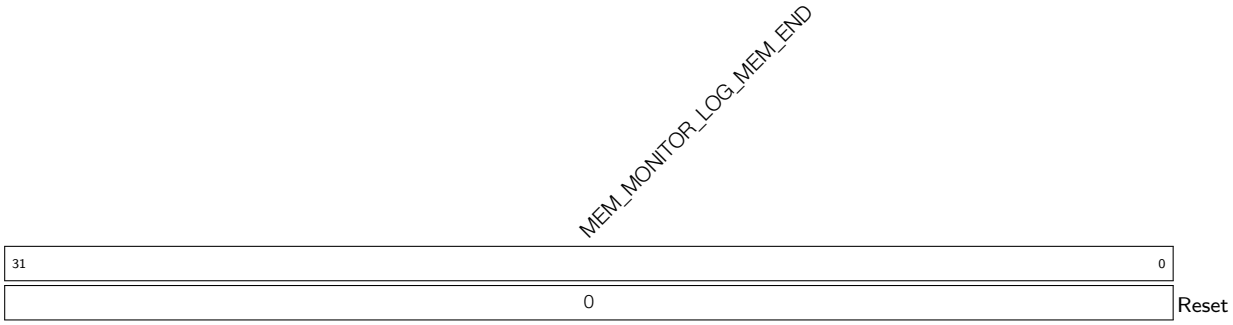
MEM_MONITOR_LOG_MAX 配置监测区域的结束地址。(R/W)

Register 16.6. MEM_MONITOR_LOG_MEM_START_REG (0x0014)



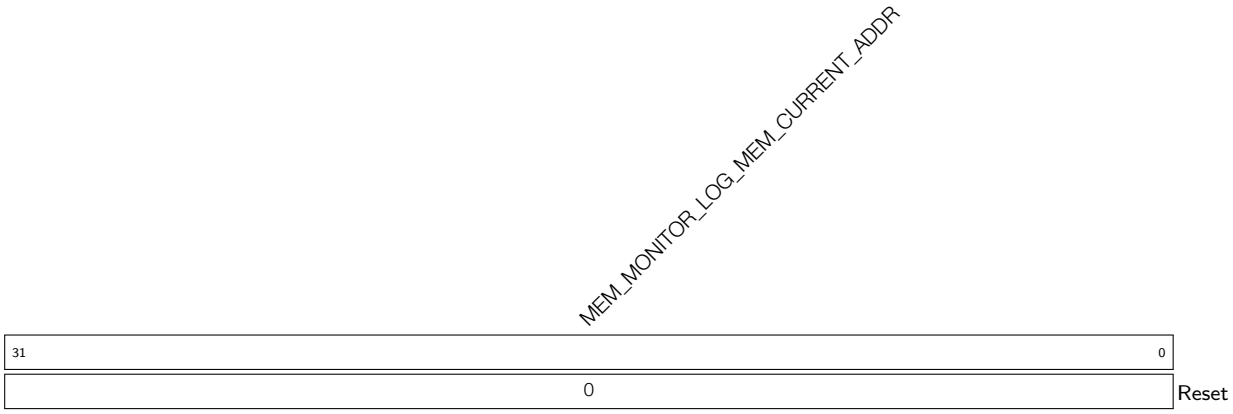
MEM_MONITOR_LOG_MEM_START 配置记录数据写入内存的起始地址。(R/W)

Register 16.7. MEM_MONITOR_LOG_MEM_END_REG (0x0018)



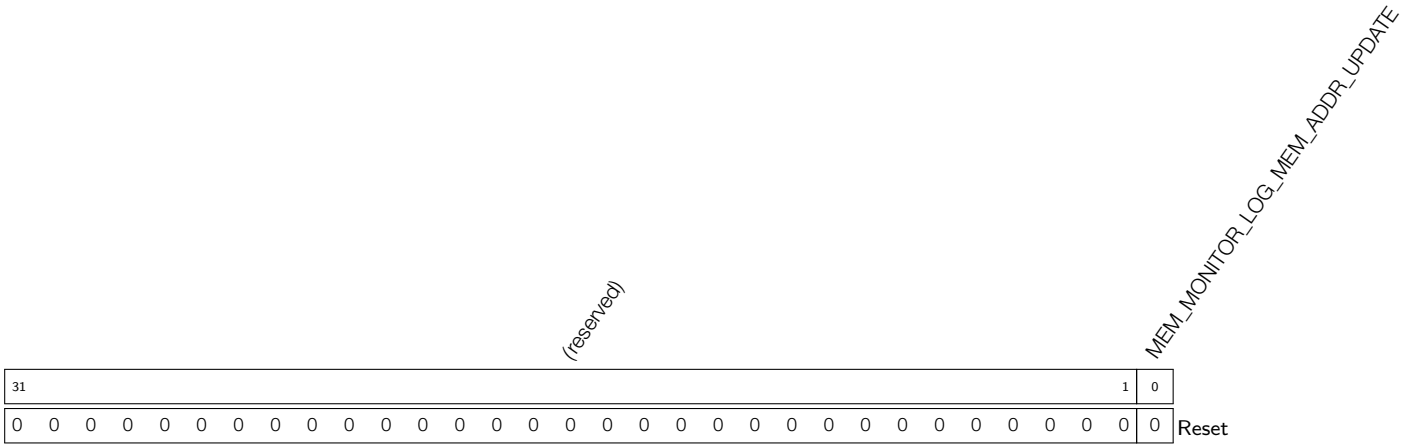
MEM_MONITOR_LOG_MEM_END 配置记录数据写入内存的结束地址。(R/W)

Register 16.8. MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG (0x001C)



MEM_MONITOR_LOG_MEM_CURRENT_ADDR 表示下一次写内存的写地址。(RO)

Register 16.9. MEM_MONITOR_LOG_MEM_ADDR_UPDATE_REG (0x0020)



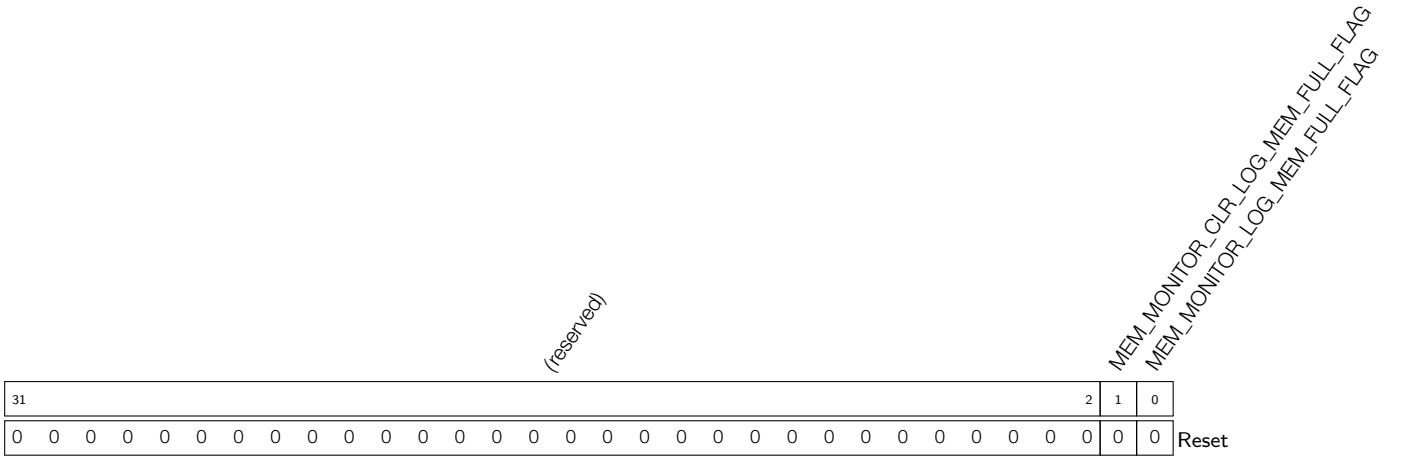
MEM_MONITOR_LOG_MEM_ADDR_UPDATE 配置是否将 MEM_MONITOR_LOG_MEM_START_REG 的值更新至 MEM_MONITOR_LOG_MEM_CURRENT_ADDR_REG。

1：更新

0：不更新（默认值）

(R/W)

Register 16.10. MEM_MONITOR_LOG_MEM_FULL_FLAG_REG (0x0024)



MEM_MONITOR_LOG_MEM_FULL_FLAG 表示数据是否溢出存储地址范围。

0：未溢出

1：溢出

(RO)

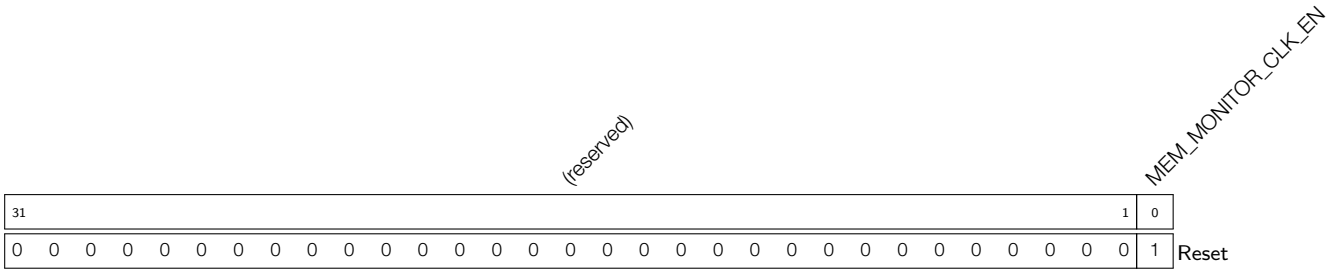
MEM_MONITOR_CLR_LOG_MEM_FULL_FLAG 配置是否清除 MEM_MONITOR_LOG_MEM_FULL_FLAG 标志位。

0：不清除（默认值）

1：清除

(R/W)

Register 16.11. MEM_MONITOR_CLOCK_GATE_REG (0x0028)



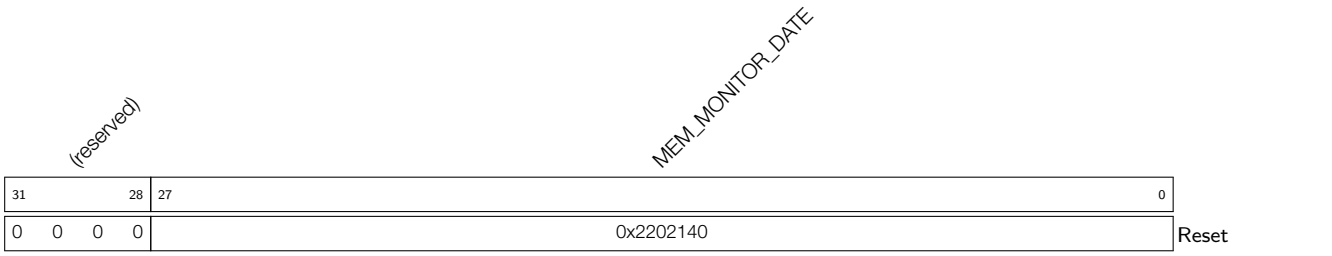
MEM_MONITOR_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

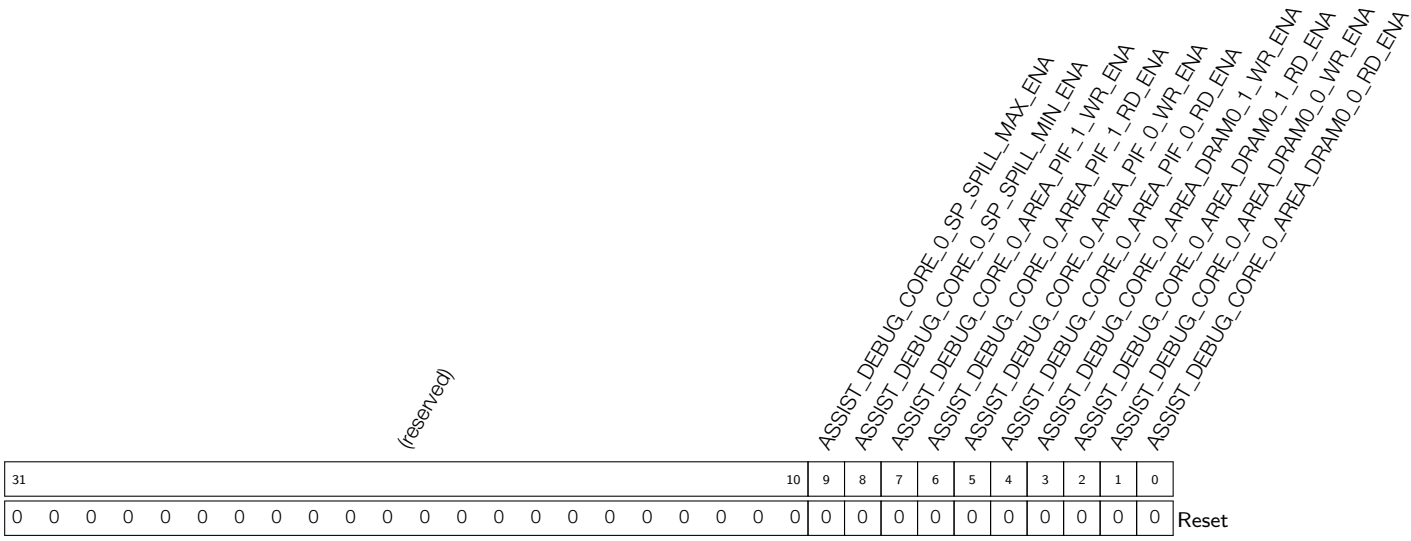
Register 16.12. MEM_MONITOR_DATE_REG (0x03FC)



MEM_MONITOR_DATE 版本控制寄存器。(R/W)

16.6.2 其它寄存器

Register 16.13. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_ENA 配置是否监测数据总线在区域 0 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_ENA 配置是否监测数据总线在区域 0 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_ENA 配置是否监测数据总线在区域 1 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_ENA 配置是否监测数据总线在区域 1 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_ENA 配置是否监测外设总线在区域 0 内读操作。

0：不监测

1：监测

(R/W)

见下页……

Register 16.13. ASSIST_DEBUG_CORE_0_MONTR_ENA_REG (0x0000)

接上页……

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_ENA 配置是否监测外设总线在区域 0 内写操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_ENA 配置是否监测外设总线在区域 1 内读操作。

0：不监测

1：监测

(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_ENA 配置是否监测外设总线在区域 1 内写操作。

0：不监测 1：监测 (R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_ENA 配置是否监测栈指针小于栈监测区域的起始地址。

0：不监测

1：监测

(R/W)

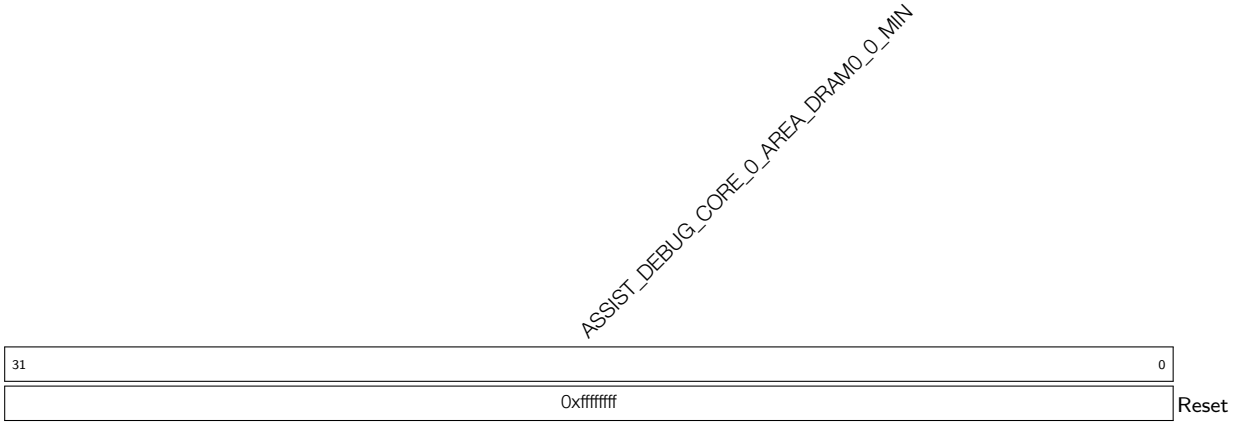
ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_ENA 配置是否监测栈指针大于栈监测区域的结束地址。

0：不监测

1：监测

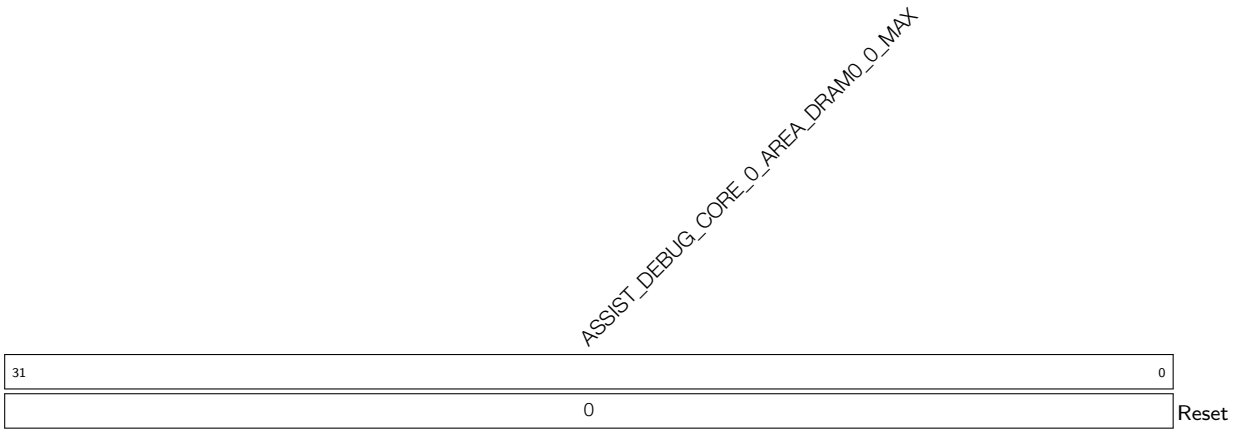
(R/W)

Register 16.14. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN_REG (0x0010)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MIN 配置数据总线区域 0 的起始地址。(R/W)

Register 16.15. ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX_REG (0x0014)

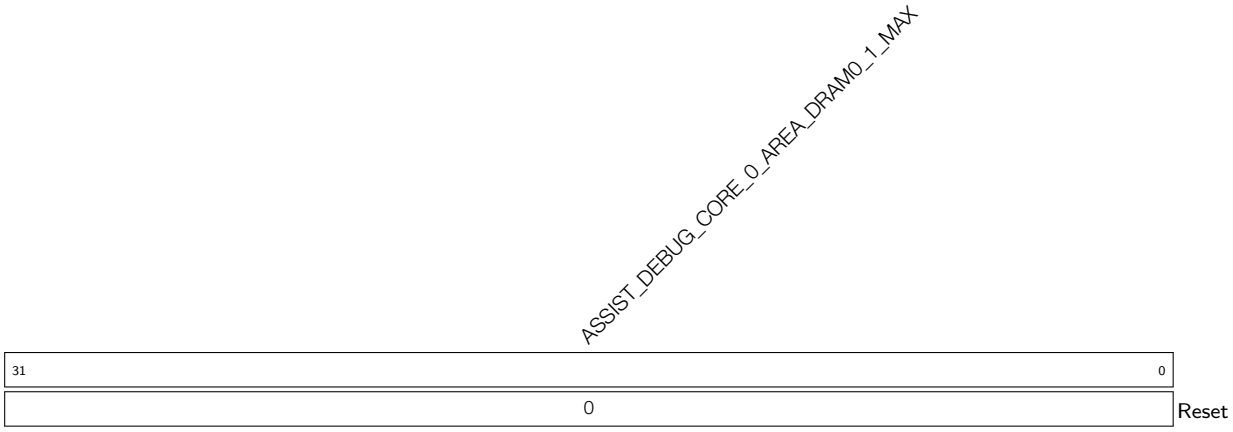


ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_MAX 配置数据总线区域 0 的结束地址。(R/W)

Register 16.16. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN_REG (0x0018)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MIN 配置数据总线区域 1 的起始地址。(R/W)

Register 16.17. ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX_REG (0x001C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_MAX 配置数据总线区域 1 的结束地址。(R/W)

Register 16.18. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN_REG (0x0020)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_MIN 配置外设总线区域 0 的起始地址。(R/W)

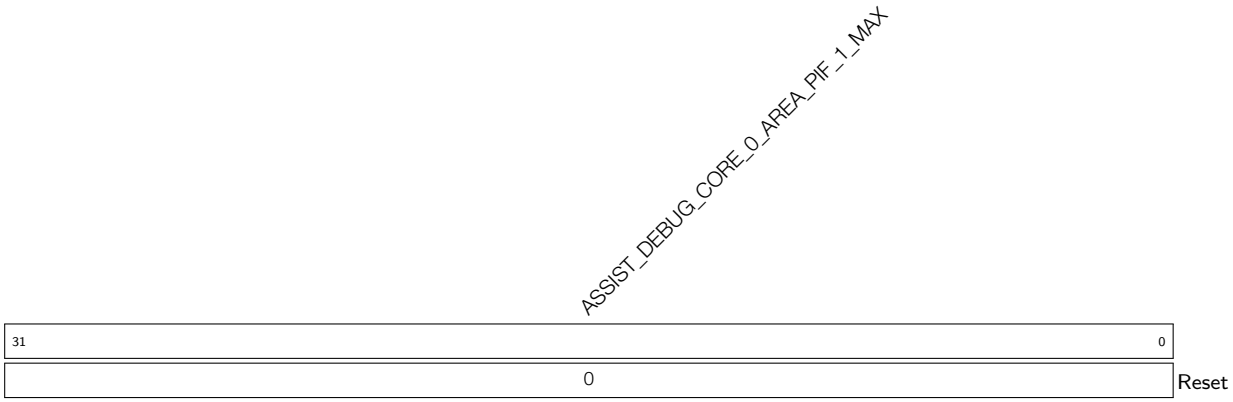
Register 16.19. ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX_REG (0x0024)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_MAX 配置外设总线区域 0 的结束地址。(R/W)

Register 16.20. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN_REG (0x0028)

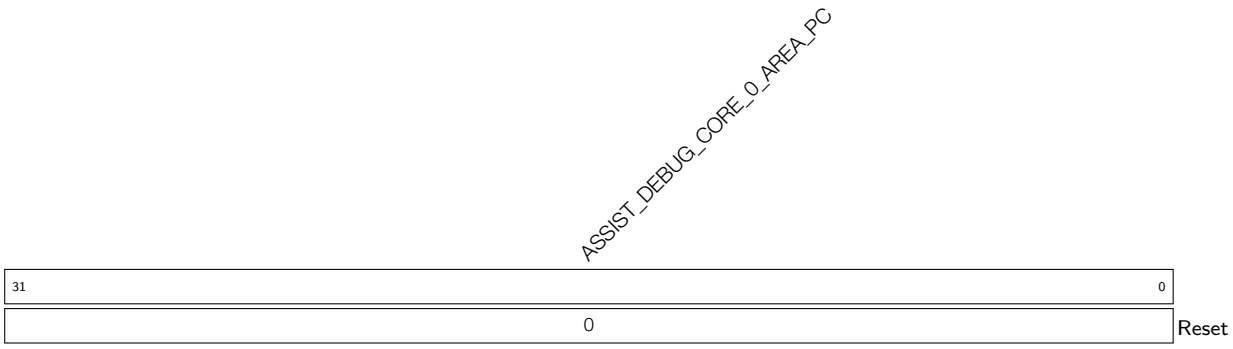
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MIN 配置外设总线区域 1 的起始地址。(R/W)

Register 16.21. ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX_REG (0x002C)



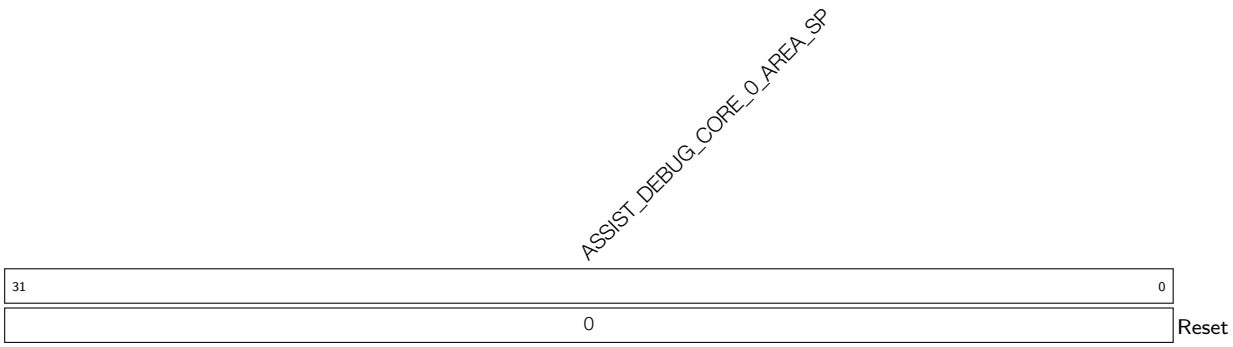
ASSIST_DEBUG_CORE_0_AREA_PIF_1_MAX 配置外设总线区域 1 的结束地址。(R/W)

Register 16.22. ASSIST_DEBUG_CORE_0_AREA_PC_REG (0x0030)



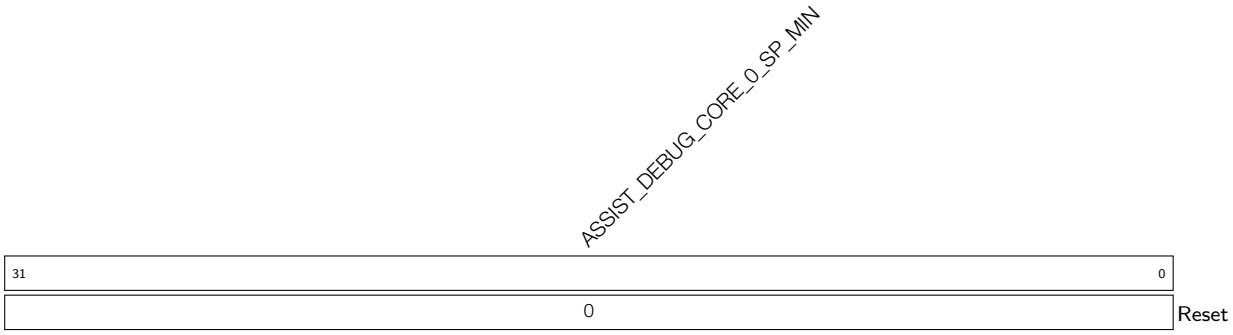
ASSIST_DEBUG_CORE_0_AREA_PC 表示区域监测下触发中断时刻的 CPU PC 值。(RO)

Register 16.23. ASSIST_DEBUG_CORE_0_AREA_SP_REG (0x0034)



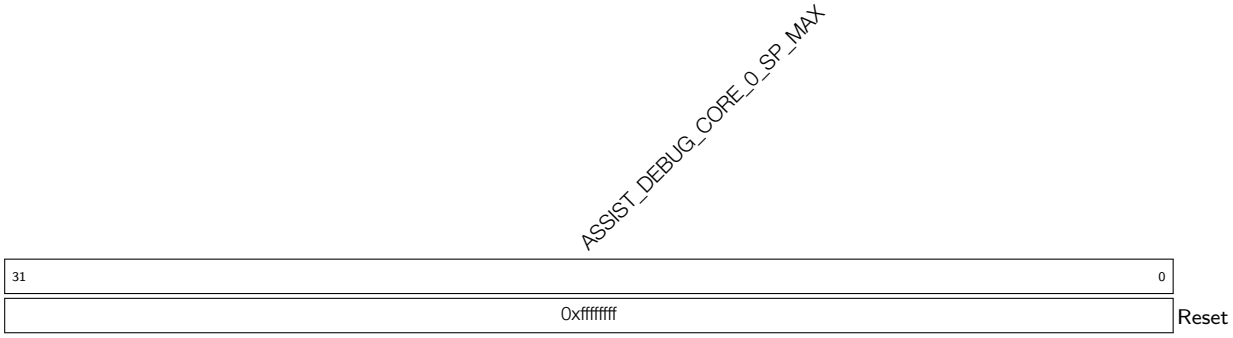
ASSIST_DEBUG_CORE_0_AREA_SP 表示区域监测下触发中断时刻的 CPU SP 值。(RO)

Register 16.24. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0038)



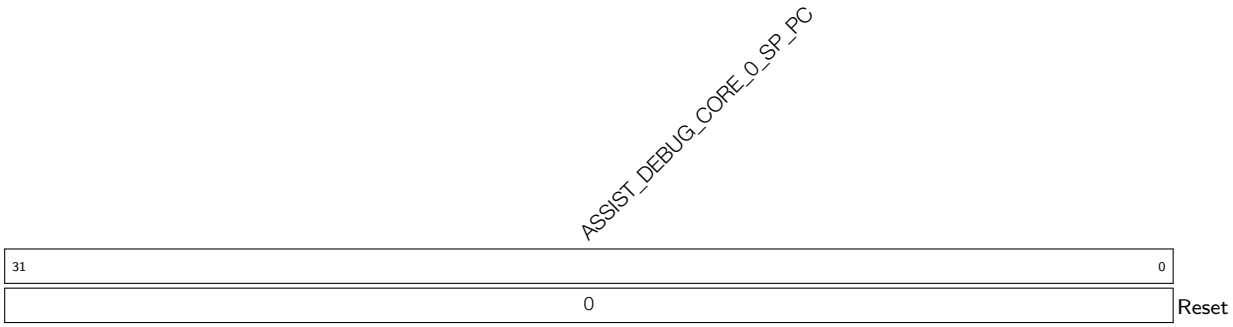
ASSIST_DEBUG_CORE_0_SP_MIN 配置栈监测区域的起始地址。(R/W)

Register 16.25. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x003C)



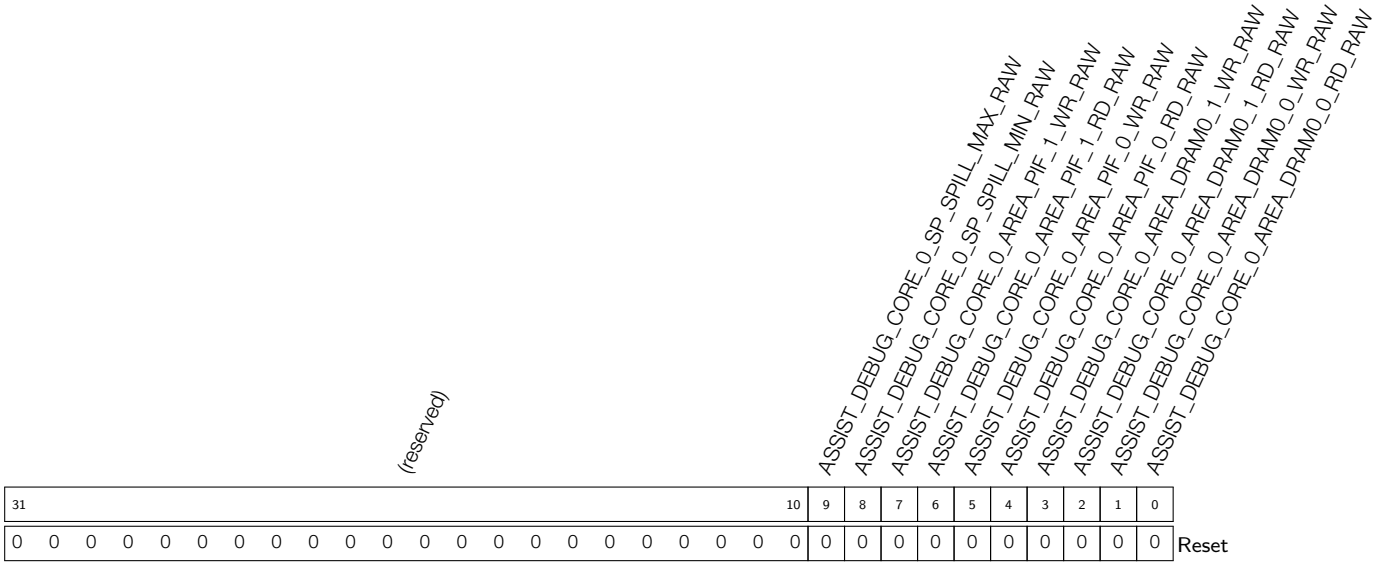
ASSIST_DEBUG_CORE_0_SP_MAX 配置栈监测区域的结束地址。(R/W)

Register 16.26. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0040)



ASSIST_DEBUG_CORE_0_SP_PC 表示栈指针监测的 CPU PC 值。(RO)

Register 16.27. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_RAW 数据总线在区域 0 内读操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_RAW 数据总线在区域 0 内写操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_RAW 数据总线在区域 1 内读操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_RAW 数据总线在区域 1 内写操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_RAW 外设总线在区域 0 内读操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_RAW 外设总线在区域 0 内写操作的原始中断状态。(RO)

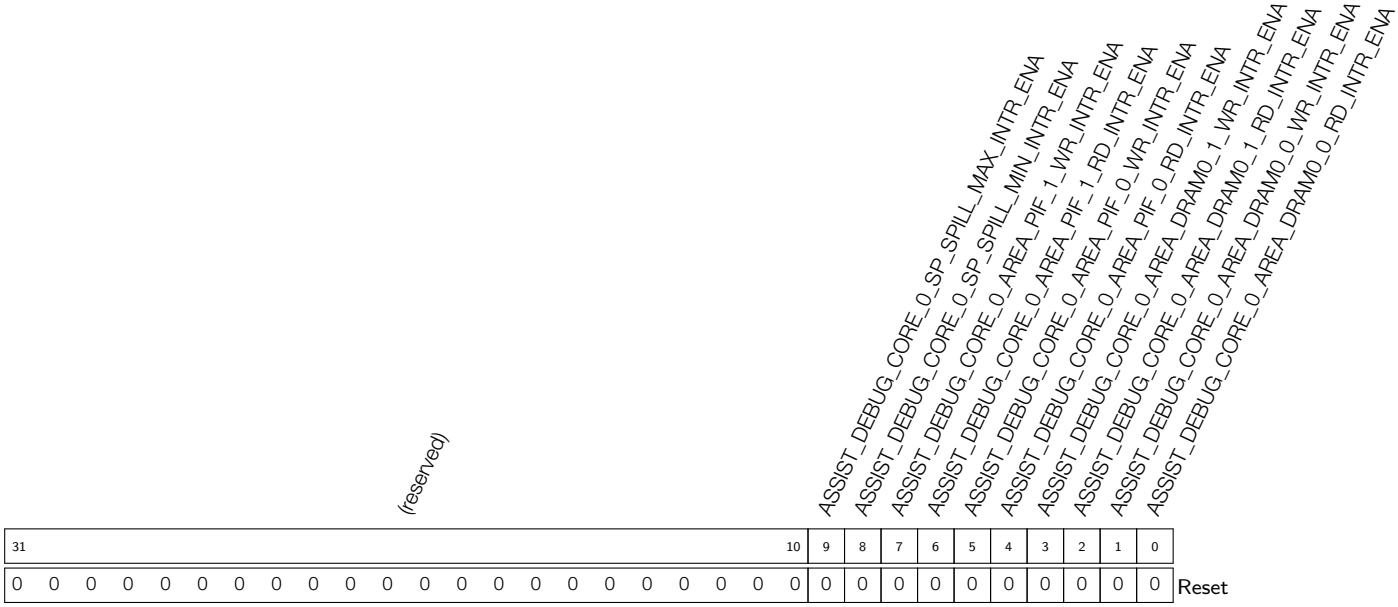
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_RAW 外设总线在区域 1 内读操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_RAW 外设总线在区域 1 内写操作的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW 栈指针小于栈监测区域的起始地址的原始中断状态。(RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW 栈指针大于栈监测区域的结束地址的原始中断状态。(RO)

Register 16.28. ASSIST_DEBUG_CORE_0_INTR_ENA_REG (0x0008)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_INTR_ENA 写 1 使能数据总线在区域 0 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_INTR_ENA 写 1 使能数据总线在区域 0 内写操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_INTR_ENA 写 1 使能数据总线在区域 1 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_INTR_ENA 写 1 使能数据总线在区域 1 内写操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_INTR_ENA 写 1 使能外设总线在区域 0 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_INTR_ENA 写 1 使能外设总线在区域 0 内写操作的中断。(R/W)

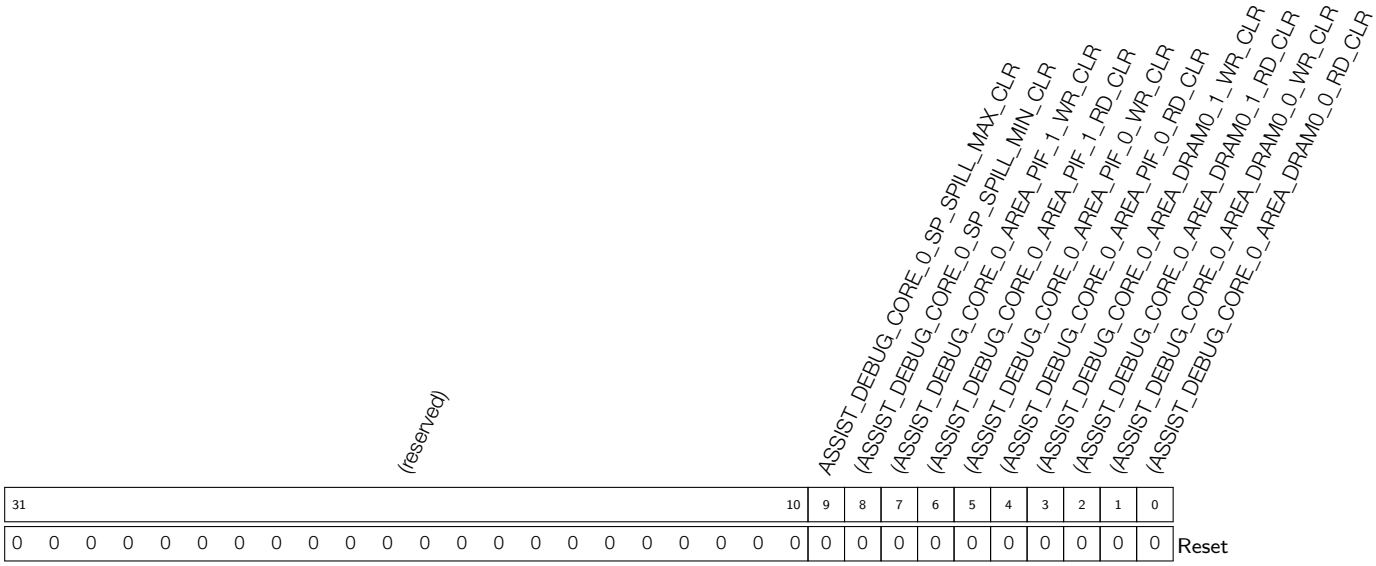
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_INTR_ENA 写 1 使能外设总线在区域 1 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_INTR_ENA 写 1 使能外设总线在区域 1 内写操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_INTR_ENA 写 1 使能栈指针小于栈监测区域的起始地址的中断。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_INTR_ENA 写 1 使能栈指针大于栈监测区域的结束地址的中断。(R/W)

Register 16.29. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_RD_CLR 写 1 清除数据总线在区域 0 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_0_WR_CLR 写 1 清除数据总线在区域 0 内写操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_RD_CLR 写 1 清除数据总线在区域 1 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_DRAM0_1_WR_CLR 写 1 清除数据总线在区域 1 内写操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_RD_CLR 写 1 清除外设总线在区域 0 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_0_WR_CLR 写 1 清除外设总线在区域 0 内写操作的中断。(R/W)

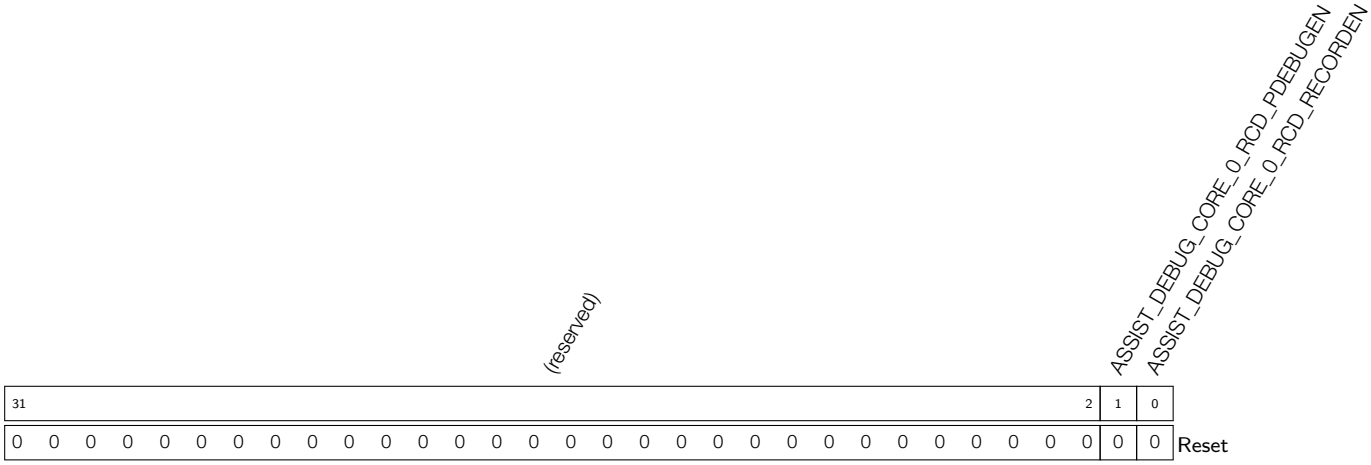
ASSIST_DEBUG_CORE_0_AREA_PIF_1_RD_CLR 写 1 清除外设总线在区域 1 内读操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_AREA_PIF_1_WR_CLR 写 1 清除外设总线在区域 1 内写操作的中断。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR 写 1 清除栈指针小于栈监测区域的起始地址的中断。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR 写 1 清除栈指针大于栈监测区域的结束地址的中断。(R/W)

Register 16.30. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x0044)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN 配置是否使能 CPU PC 记录。

0：不使能

1：ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 开始实时记录 CPU PC

(R/W)

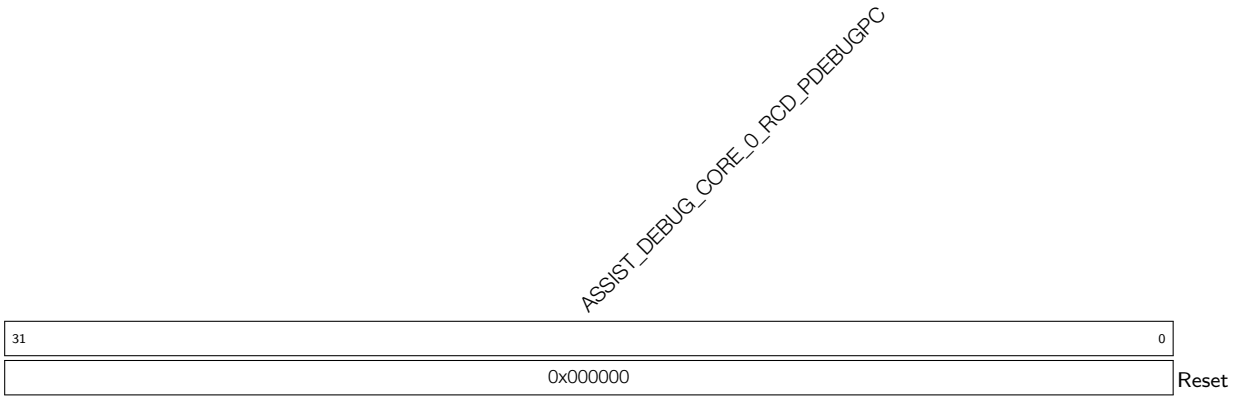
ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 配置是否使能 CPU 调试。

0：不使能

1：CPU 输出 CPU PC

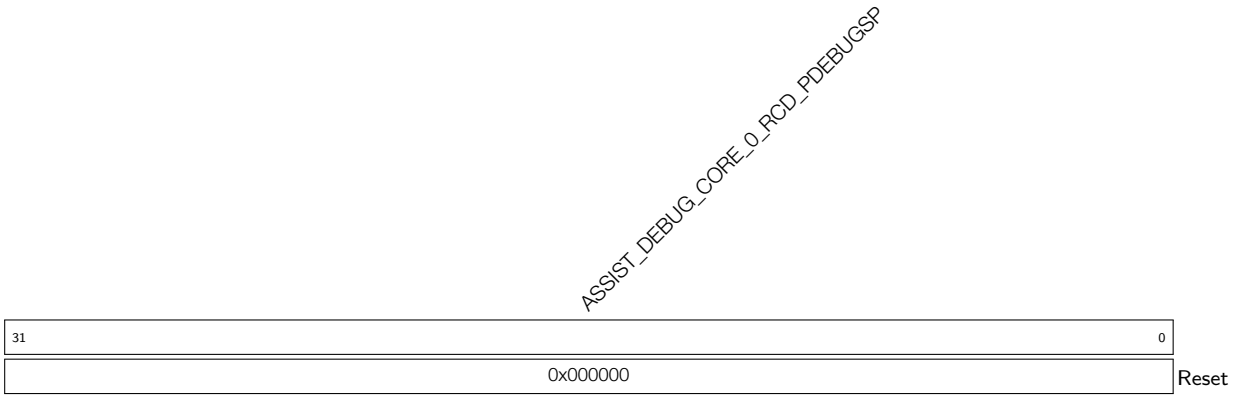
(R/W)

Register 16.31. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0048)



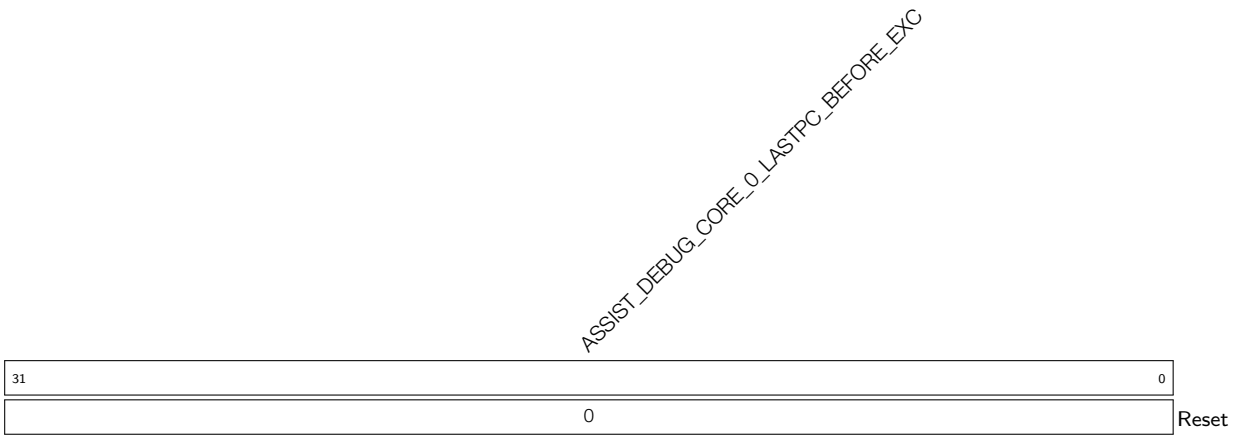
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC 表示 CPU 复位时刻的 PC 值。(RO)

Register 16.32. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x004C)



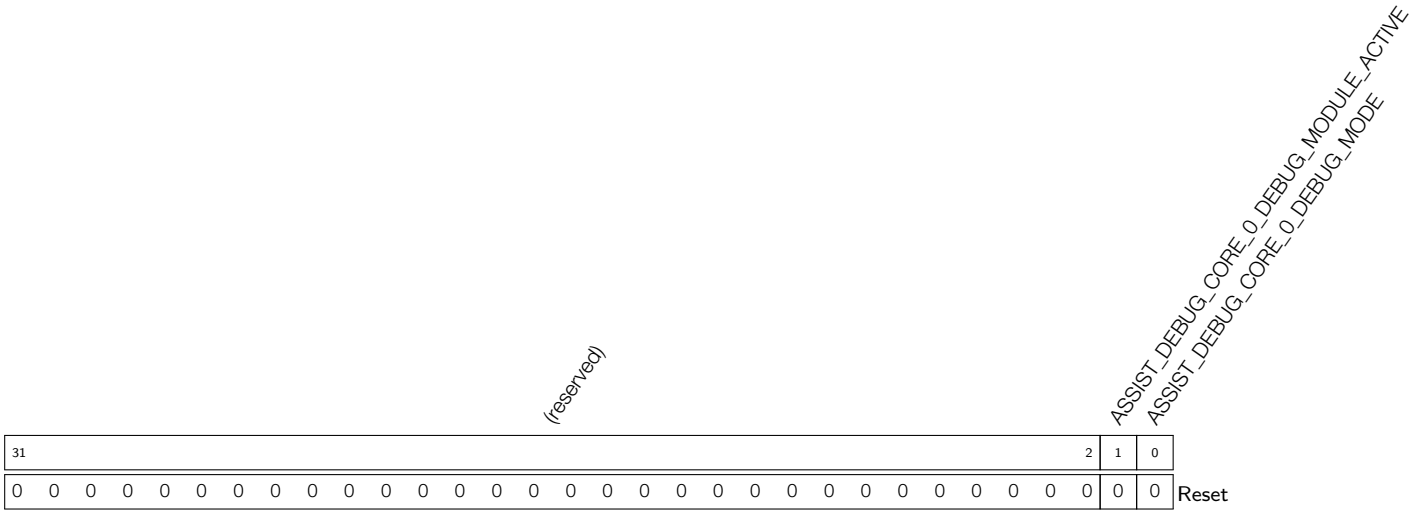
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP 表示 CPU SP。(RO)

Register 16.33. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0070)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC 表示 CPU 异常前的最后一条指令的 PC。(RO)

Register 16.34. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x0074)



ASSIST_DEBUG_CORE_0_DEBUG_MODE 表示 ESP-RISC-V CPU 是否处于调试模式。

1：处于调试模式

0：未处于调试模式

(RO)

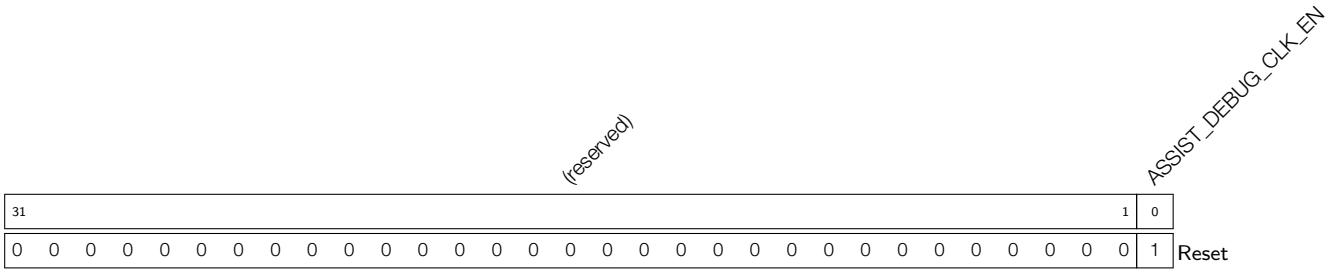
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE 表示 ESP-RISC-V CPU 调试模块的工作状态。

1：处于工作状态

其它：未处于工作状态

(RO)

Register 16.35. ASSIST_DEBUG_CLOCK_GATE_REG (0x0078)



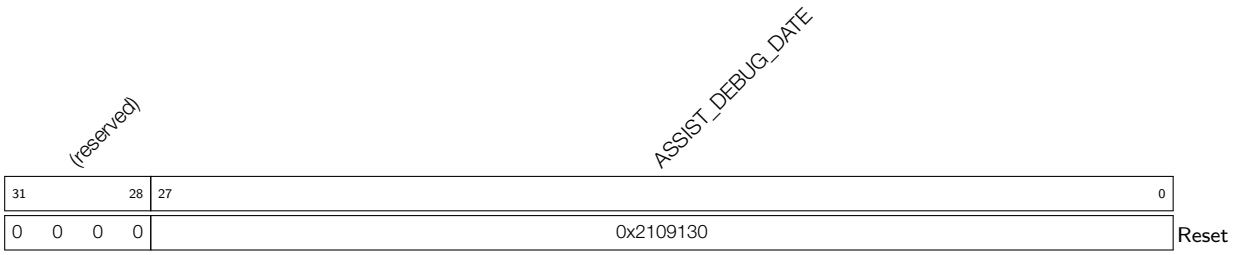
ASSIST_DEBUG_CLK_EN 配置是否使能寄存器时钟门控。

0：不使能

1：使能

(R/W)

Register 16.36. ASSIST_DEBUG_DATE_REG (0x03FC)



ASSIST_DEBUG_DATE 版本控制寄存器。(R/W)

17 AES 加速器 (AES)

17.1 概述

ESP32-H2 内置 AES（高级加密标准）硬件加速器可使用 AES 算法，完成数据的加解密运算，具有 Typical AES 和 DMA-AES 两种工作模式。整体而言，相比基于纯软件的 AES 运算，AES 硬件加速器能够极大地提高运算速度。

17.2 主要特性

ESP32-H2 支持以下特性：

- Typical AES 工作模式
  - AES-128/AES-256 加解密运算
- DMA-AES 工作模式
  - AES-128/AES-256 加解密运算
  - 块（加密）模式
    - ECB (Electronic Codebook)
    - CBC (Cipher Block Chaining)
    - OFB (Output Feedback)
    - CTR (Counter)
    - CFB8 (8-bit Cipher Feedback)
    - CFB128 (128-bit Cipher Feedback)
  - 中断发生

17.3 时钟和复位

AES 加速器的激活需使能 PCR_AES_CONF_REG 寄存器中的 PCR_AES_CLK_EN 位，并同时清零 PCR_AES_RST_EN 位。此外，由于密码学加速器模块之间的资源复用，用户还需要额外清零 PCR_DS_CONF_REG 寄存器中的 PCR_DS_RST_EN 位。

17.4 工作模式简介

ESP32-H2 内置的 AES 加速器支持 Typical AES 和 DMA-AES 两种工作模式。

- Typical AES 工作模式：
  - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 NIST FIPS 197 标准中的 AES-128 和 AES-256 加解密运算。

  这种情况下，明文和密文的读写操作统一通过 CPU 访问完成。
- DMA-AES 工作模式：
  - 支持使用 128 位或 256 位密钥进行加密与解密运算，即 NIST FIPS 197 标准中的 AES-128 和 AES-256 加解密运算；

– 还支持 NIST SP 800-38A 标准中的 ECB、CBC、OFB、CTR、CFB8、CFB128 块加密模式运算。

在这种情况下，明文和密文的传输通过硬件上的 DMA 完成，计算完成时会有中断发生。

用户可通过配置 AES_DMA_ENABLE_REG 选择 AES 加速器的工作模式，具体参考表 17-1。

表 17-1. 工作模式

AES_DMA_ENABLE_REG	工作模式
0	Typical AES
1	DMA-AES

用户可通过配置 AES_MODE_REG 寄存器选择密钥长度和加解密方向，具体可参考表 17-2。

表 17-2. 密钥长度和加解密方向

AES_MODE_REG[2:0]	密钥长度和加解密方向
0	AES-128 加密
1	保留
2	AES-256 加密
3	保留
4	AES-128 解密
5	保留
6	AES-256 解密
7	保留

有关 Typical AES 和 DMA-AES 两种工作模式的具体介绍，请见下方 17.5 章节和 17.6 章节。

注意：

ESP32-H2 的数字签名算法 (DSA)模块也会调用 AES 加速器。此时，用户无法正常访问 AES 加速器。

17.5 Typical AES 工作模式

在 Typical AES 工作模式下，AES 加速器的状态值可查看寄存器 AES_STATE_REG，具体见表 17-3 所示：

表 17-3. 状态返回值

返回值	描述	状态说明
0	IDLE	加速器空闲或计算完成
1	WORK	加速器忙于计算

17.5.1 密钥、明文、密文

寄存器 AES_KEY_*n*_REG 用于存放密钥，由 8 个 32 位寄存器组成。

- 如果为 AES-128 加解密运算，则 128 位密钥在寄存器 AES_KEY_0_REG ~ AES_KEY_3_REG 中。
- 如果为 AES-256 加解密运算，则 256 位密钥在寄存器 AES_KEY_0_REG ~ AES_KEY_7_REG 中。

寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG 用于存放明文和密文，各由 4 个 32 位寄存器组成。

- 如果为 AES-128 或 AES-256 加密运算，则运算开始之前用明文初始化寄存器 AES_TEXT_IN_*m*_REG。运算完成之后，AES 加速器将把密文更新入寄存器 AES_TEXT_OUT_*m*_REG。
- 如果为 AES-128 或 AES-256 解密运算，则运算开始之前用密文初始化寄存器 AES_TEXT_IN_*m*_REG。运算完成之后，AES 加速器将把明文更新入寄存器 AES_TEXT_OUT_*m*_REG。

17.5.2 字节序

文本字节序

在 Typical AES 工作模式下，AES 加速器可以使用密钥对 128 位的 block 进行加解密。在操作寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG 中的数据时，用户应遵循表 17-4 中定义的文本字节序。

表 17-4. Typical AES 文本字节序

明文/密文					
State[1]		c[2]			
		0	1	2	3
r	0	AES_TEXT_*x*_0_REG[7:0]	AES_TEXT_*x*_1_REG[7:0]	AES_TEXT_*x*_2_REG[7:0]	AES_TEXT_*x*_3_REG[7:0]
	1	AES_TEXT_*x*_0_REG[15:8]	AES_TEXT_*x*_1_REG[15:8]	AES_TEXT_*x*_2_REG[15:8]	AES_TEXT_*x*_3_REG[15:8]
	2	AES_TEXT_*x*_0_REG[23:16]	AES_TEXT_*x*_1_REG[23:16]	AES_TEXT_*x*_2_REG[23:16]	AES_TEXT_*x*_3_REG[23:16]
	3	AES_TEXT_*x*_0_REG[31:24]	AES_TEXT_*x*_1_REG[31:24]	AES_TEXT_*x*_2_REG[31:24]	AES_TEXT_*x*_3_REG[31:24]

[1] 有关 “State（以及 c 和 r）” 的详细定义，请参考 NIST FIPS 197 中 3.4 The State 章节。
[2] 其中，*x* = IN 或 OUT。

密钥字节序

在 Typical AES 工作模式下，在向寄存器 AES_KEY_*n*_REG 中填入数据时，用户应遵循表 17-5 和表 17-6 中定义的文本字节序。

表 17-5. AES-128 密钥字节序

Bit[1]	w[0]	w[1]	w[2]	w[3][2]
[31:24]	AES_KEY_0_REG[7:0]	AES_KEY_1_REG[7:0]	AES_KEY_2_REG[7:0]	AES_KEY_3_REG[7:0]
[23:16]	AES_KEY_0_REG[15:8]	AES_KEY_1_REG[15:8]	AES_KEY_2_REG[15:8]	AES_KEY_3_REG[15:8]
[15:8]	AES_KEY_0_REG[23:16]	AES_KEY_1_REG[23:16]	AES_KEY_2_REG[23:16]	AES_KEY_3_REG[23:16]
[7:0]	AES_KEY_0_REG[31:24]	AES_KEY_1_REG[31:24]	AES_KEY_2_REG[31:24]	AES_KEY_3_REG[31:24]

[1] Bit 列代表 w[0] ~ w[3] 每个 word 中的各个字节。
[2] w[0] ~ w[3] 符合标准 NIST FIPS 197 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

表 17-6. AES-256 密钥字节序

Bit[1]	w[0]	w[1]	w[2]	w[3]	w[4]	w[5]	w[6]	w[7][2]
[31:24]	AES_KEY_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]

[1] Bit 列代表 w[0] ~ w[7] 每个 word 中的各个字节。

[2] w[0] ~ w[7] 符合标准 NIST FIPS 197 中 5.2 Key Expansion 章节中对 “the first Nk words of the expanded key” 的描述。

GoBack

17.5.3 Typical AES 工作模式的流程

单次运算

1. 对寄存器 AES_DMA_ENABLE_REG 写入 0。
2. 初始化寄存器 AES_MODE_REG、AES_KEY_*n*_REG、AES_TEXT_IN_*m*_REG。
3. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 0。
5. 从寄存器 AES_TEXT_OUT_*m*_REG 读取结果。

连续运算

在连续运算过程中，每次运算完成之后，只有寄存器 AES_TEXT_IN_*m*_REG 和 AES_TEXT_OUT_*m*_REG (*m*: 0-3) 会被 AES 加速器更新，而 AES_DMA_ENABLE_REG、AES_MODE_REG、AES_KEY_*n*_REG 等寄存器中的内容不会变化。所以进行连续运算时可以简化初始化操作。

1. 第一次运算之前对寄存器 AES_DMA_ENABLE_REG 写入 0。
2. 第一次运算之前初始化寄存器 AES_MODE_REG 和 AES_KEY_*n*_REG。
3. 更新寄存器 AES_TEXT_IN_*m*_REG。
4. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
5. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 0。
6. 从寄存器 AES_TEXT_OUT_*m*_REG 读取结果 。返回步骤 3，进行下一轮运算。

17.6 DMA-AES 工作模式

在 DMA-AES 工作模式下，AES 加速器可支持 ECB、CBC、OFB、CTR、CFB8、CFB128 六种块模式运算。用户可通过配置 AES_BLOCK_MODE_REG 寄存器选择具体运算类型，具体可参考表 17-7。

表 17-7. 块模式选择

AES_BLOCK_MODE_REG[2:0]	块模式
0	ECB (Electronic Code Book)
1	CBC (Cipher Block Chaining)
2	OFB (Output FeedBack)
3	CTR (Counter)
4	CFB8 (8-bit Cipher FeedBack)
5	CFB128 (128-bit Cipher FeedBack)
6	保留
7	保留

AES 加速器的状态值可查看寄存器 AES_STATE_REG，具体见表 17-8 所示：

表 17-8. 状态返回值

返回值	描述	状态说明
0	IDLE	加速器空闲
1	WORK	加速器忙于计算
2	DONE	加速器计算完成

AES 加速器在 DMA-AES 工作模式下允许中断发生，软件清零。中断功能默认关闭，用户可通过将 AES_INT_ENA_REG 寄存器配置为 1 开启中断。如开启中断功能，AES 加速器在完成计算时，中断发生。

17.6.1 密钥、明文、密文

块运算模式

在块运算模式下，AES 加速器的源数据来自 DMA，结果数据也将被写入 DMA。

- 如果为加密运算，则 DMA 从存储器中读取明文数据流并将其传给 AES。AES 计算出密文后将密文写入 DMA。DMA 再将密文写入存储器。
- 如果为解密运算，则 DMA 从存储器中读取密文数据流并将其传给 AES。AES 计算出明文后将明文写入 DMA。DMA 再将明文写入存储器。

AES 加速器在进行块运算时，结果数据与源数据的大小保持一致。此时，DMA 的数据搬运过程和 AES 的计算过程有所交叠，因此总工作时间有所减少。

值得注意的是，AES 加速器在 DMA-AES 工作模式下要求源数据的大小必须是 128 位的整数倍，否则需要将原始明文封装为 128 位的整数倍，即在原比特串 (bit string) 尾部尽可能少的补 “0”，具体过程见表 17-9 所示。

表 17-9. TEXT-PADDING

Function : TEXT-PADDING( )
Input : X, bit string. **Output** : Y = **TEXT-PADDING**(X), whose length is the nearest integral multiples of 128 bits.
Steps Let us assume that X is a data-stream that can be split into n parts as following: $X = X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n$ Here, the lengths of $X_1, X_2, \cdots, X_{n-1}$ all equal to 128 bits, and the length of X_n is t (0<=t<=127). If $t = 0$, then **TEXT-PADDING**(X) = X; If $0 < t <= 127$, define a 128-bit block, X_n^*, and let $X_n^* = X_n \| \| 0^{128-t}$, then **TEXT-PADDING**($X$) = $X_1 \| \| X_2 \| \| \cdots \| \| X_{n-1} \| \| X_n^* = X \| \| 0^{128-t}$

17.6.2 字节序

在 DMA-AES 工作模式下，源数据和结果数据的传输完全由 DMA 完成，因此不支持字节序的控制调节，但要求它们在存储器中以一定的方式来存放，且要求数据量必须是 block 的整数倍。

举例说明，假设 DMA 需要搬运源数据：

- 十六进制：0102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F20
- 数据大小：相当于 2 个 block

假设起始地址为 0x0280，则源数据在存储器中的存放位置如表 17-10 所示。结果数据也遵从相同的存放规则，在此不多做介绍。

表 17-10. DMA-AES 存储字节序

地址	字节	地址	字节	地址	字节	地址	字节
0x0280	0x01	0x0281	0x02	0x0282	0x03	0x0283	0x04
0x0284	0x05	0x0285	0x06	0x0286	0x07	0x0287	0x08
0x0288	0x09	0x0289	0x0A	0x028A	0x0B	0x028B	0x0C
0x028C	0x0D	0x028D	0x0E	0x028E	0x0F	0x028F	0x10
0x0290	0x11	0x0291	0x12	0x0292	0x13	0x0293	0x14
0x0294	0x15	0x0295	0x16	0x0296	0x17	0x0297	0x18
0x0298	0x19	0x0299	0x1A	0x029A	0x1B	0x029B	0x1C
0x029C	0x1D	0x029D	0x1E	0x029E	0x1F	0x029F	0x20

17.6.3 标准增量函数

AES 加速器在进行 CTR 块运算时，还可提供两种标准增量函数供用户选择：INC_{32} 和 INC_{128} 。用户可通过将寄存器 AES_INC_SEL_REG 置为 0 或 1 选择 INC_{32} 或 INC_{128} 标准增量函数。更多有关标准增量函数的内容，请见 NIST SP 800-38A 标准中的 B.1 The Standard Incrementing Function 章节。

17.6.4 块个数

寄存器 AES_BLOCK_NUM_REG 存放明文或密文的块个数 (Block Number)，其值等于 length(**TEXT-PADDING**(P))/128，也等于 length(**TEXT-PADDING**(C))/128。这里的 P 指明文（plaintext），C 指密文（ciphertext）。该寄存器仅在 DMA-AES 工作模式下有意义。

17.6.5 初始向量

存储器 AES_IV_MEM 的空间大小为 16 字节，仅在块运算模式下有效。对于 CBC/OFB/CFB8/CFB128 操作，AES_IV_MEM 用于存放初始向量 (Initialization Vector, IV) 的值。对于 CTR 操作，AES_IV_MEM 存放初始计数器 (Initial Counter Block, ICB) 的值。

IV 和 ICB 都是 128 位的比特串，从左向右被分割成 16 个字节 (Byte0, Byte1, Byte2, ···, Byte15)，构成一个字节序列，在 AES_IV_MEM 中存放时需要遵循表 17-10 中的字节序规则，即 Byte0 存放在 AES_IV_MEM 中的最低地址中，Byte15 存放在 AES_IV_MEM 中的最高地址中。

更多有关 IV 和 ICB 的信息，请参考 NIST SP 800-38A 标准。

17.6.6 DMA-AES 工作模式的流程

1. 选择一条 DMA 通道与 AES 加速器连接，配置 DMA 链表，而后启动 DMA。详情请见章节 3 *通用 DMA 控制器 (GDMA)*。
2. 配置 AES：
    - 对寄存器 AES_DMA_ENABLE_REG 写入 1。

- 选择是否开启中断。根据需要设置寄存器 AES_INT_ENA_REG 的值。
- 初始化 AES_MODE_REG 和 AES_KEY_*n*_REG 寄存器。
- 配置 AES_BLOCK_MODE_REG 寄存器，选择具体块加密模式。详见表 17-7。
- 初始化寄存器 AES_BLOCK_NUM_REG，请参照章节 17.6.4。
- 初始化寄存器 AES_INC_SEL_REG（仅在 CTR 块模式下使用）。
- 初始化存储器 AES_IV_MEM（在 ECB 块模式下不使用）。

3. 启动运算。对寄存器 AES_TRIGGER_REG 写入 1。
4. 等待运算完成。轮询寄存器 AES_STATE_REG，直到读到 2。如果开启了中断功能，也可以等待 AES_INT 中断产生。
5. 确认 DMA 完成从 AES 到内存的数据传输。此时，结果数据已经被 DMA 写入 memory，可以直接从中读取。详情请参考章节 3 *通用 DMA 控制器 (GDMA)*。
6. 如果开启了中断，当处理中断程序完成后，请及时对寄存器 AES_INT_CLR_REG 写 1 以清除中断。
7. 对寄存器 AES_DMA_EXIT_REG 写入 1 释放 AES 加速器。之后如果再读取寄存器 AES_STATE_REG 将读到 0。该步操作可以提前完成，但必须在步骤 4 之后。

17.7 存储器列表

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

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	大小（比特）	起始地址	结束地址	访问权限
AES_IV_MEM	存储器 IV	16 字节	0x0050	0x005F	(R/W)

17.8 寄存器列表

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

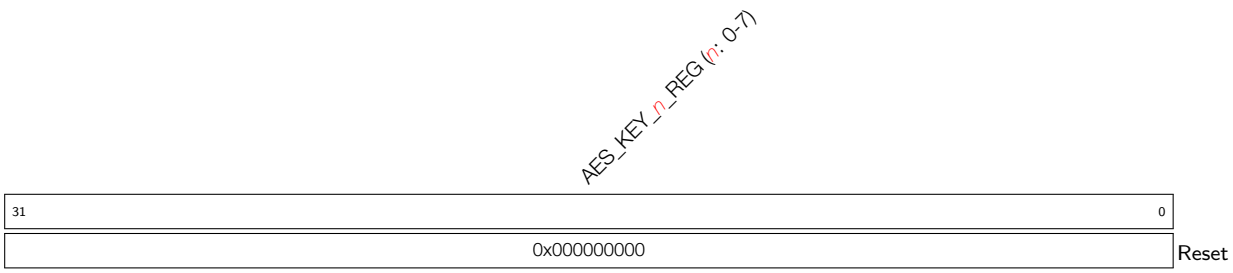
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
密钥寄存器			
AES_KEY_0_REG	AES 密钥资料寄存器 0	0x0000	R/W
AES_KEY_1_REG	AES 密钥资料寄存器 1	0x0004	R/W
AES_KEY_2_REG	AES 密钥资料寄存器 2	0x0008	R/W
AES_KEY_3_REG	AES 密钥资料寄存器 3	0x000C	R/W
AES_KEY_4_REG	AES 密钥资料寄存器 4	0x0010	R/W
AES_KEY_5_REG	AES 密钥资料寄存器 5	0x0014	R/W
AES_KEY_6_REG	AES 密钥资料寄存器 6	0x0018	R/W
AES_KEY_7_REG	AES 密钥资料寄存器 7	0x001C	R/W
TEXT_IN 寄存器			
AES_TEXT_IN_0_REG	源数据资料寄存器 0	0x0020	R/W
AES_TEXT_IN_1_REG	源数据资料寄存器 1	0x0024	R/W
AES_TEXT_IN_2_REG	源数据资料寄存器 2	0x0028	R/W
AES_TEXT_IN_3_REG	源数据资料寄存器 3	0x002C	R/W
TEXT_OUT 寄存器			
AES_TEXT_OUT_0_REG	结果数据资料寄存器 0	0x0030	RO
AES_TEXT_OUT_1_REG	结果数据资料寄存器 1	0x0034	RO
AES_TEXT_OUT_2_REG	结果数据资料寄存器 2	0x0038	RO
AES_TEXT_OUT_3_REG	结果数据资料寄存器 3	0x003C	RO
控制 / 配置寄存器			
AES_MODE_REG	配置密钥长度和加解密方向	0x0040	R/W
AES_DMA_ENABLE_REG	配置 AES 加速器工作模式	0x0090	R/W
AES_BLOCK_MODE_REG	配置 DMA-AES 下的块运算模式	0x0094	R/W
AES_BLOCK_NUM_REG	配置块数量	0x0098	R/W
AES_INC_SEL_REG	配置标准增量函数	0x009C	R/W
AES_TRIGGER_REG	开始运算	0x0048	WT
AES_DMA_EXIT_REG	退出运算	0x00B8	WO
状态寄存器			
AES_STATE_REG	运算状态	0x004C	RO
中断寄存器			
AES_INT_CLR_REG	DMA-AES 中断清除	0x00AC	WT
AES_INT_ENA_REG	DMA-AES 中断使能	0x00B0	R/W

17.9 寄存器

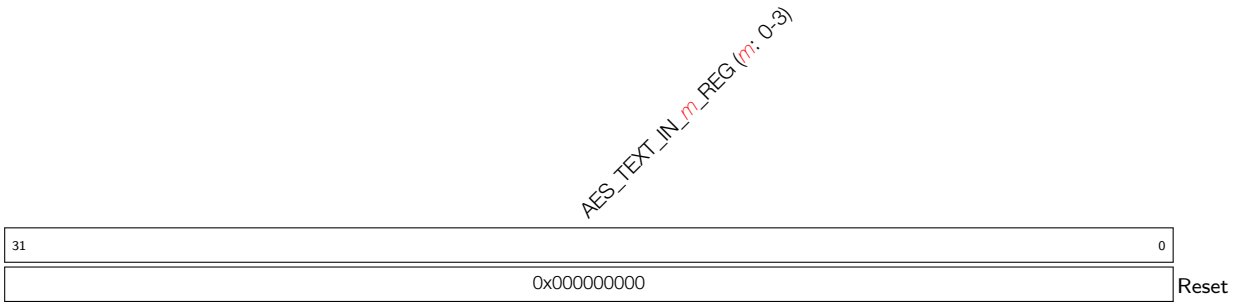
本小节的所有地址均为相对于 AES 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 17.1. AES_KEY_*n*_REG (*n*: 0-7) (0x0000+4n*)**



AES_KEY_*n*_REG (*n*: 0-7) 表示 AES 密钥资料。(R/W)

Register 17.2. AES_TEXT_IN_*m*_REG (*m*: 0-3) (0x0020+4m*)**

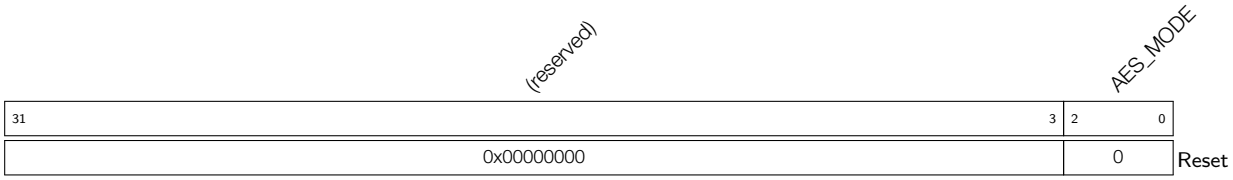


AES_TEXT_IN_*m*_REG (*m*: 0-3) 表示 Typical AES 工作模式下的输入文本。(R/W)

Register 17.3. AES_TEXT_OUT_*m*_REG (*m*: 0-3) (0x0030+4m*)**

AES_TEXT_OUT_*m*_REG (*m*: 0-3) 表示 Typical AES 工作模式下的输出文本。(RO)

Register 17.4. AES_MODE_REG (0x0040)



AES_MODE 配置 AES 加速器的密钥长度和加解密方向。

0：AES-128 加密

1：保留

2：AES-256 加密

3：保留

4：AES-128 解密

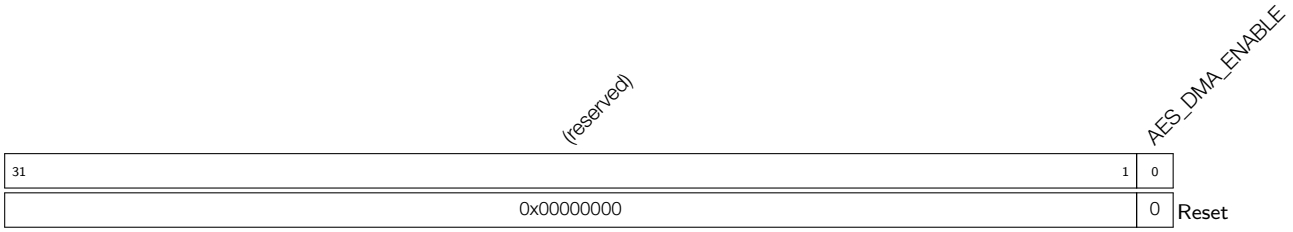
5：保留

6：AES-256 解密

7：保留

(R/W)

Register 17.5. AES_DMA_ENABLE_REG (0x0090)



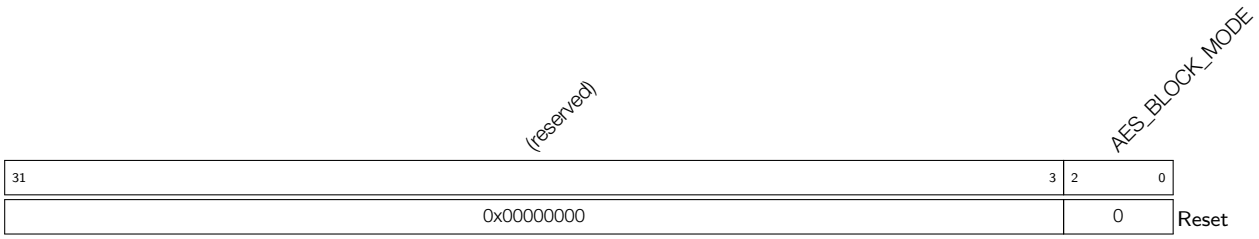
AES_DMA_ENABLE 配置 AES 加速器的工作模式。

0：Typical AES

1：DMA-AES

(R/W)

Register 17.6. AES_BLOCK_MODE_REG (0x0094)



AES_BLOCK_MODE 配置 AES 加速器在 DMA-AES 工作模式下的块模式。

0：ECB (Electronic Code Block)

1：CBC (Cipher Block Chaining)

2：OFB (Output FeedBack)

3：CTR (Counter)

4：CFB8 (8-bit Cipher FeedBack)

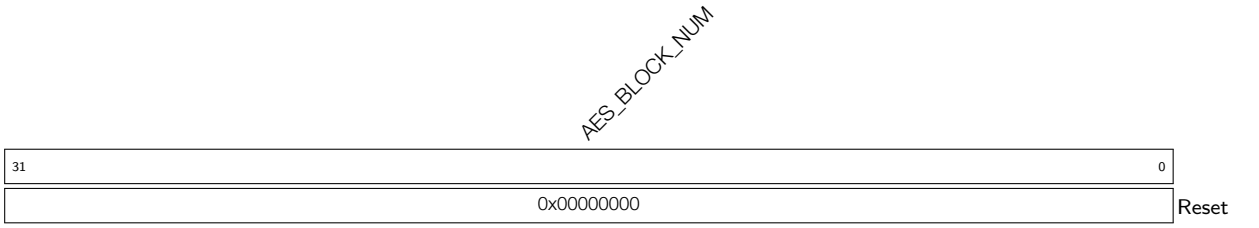
5：CFB128 (128-bit Cipher FeedBack)

6：保留

7：保留

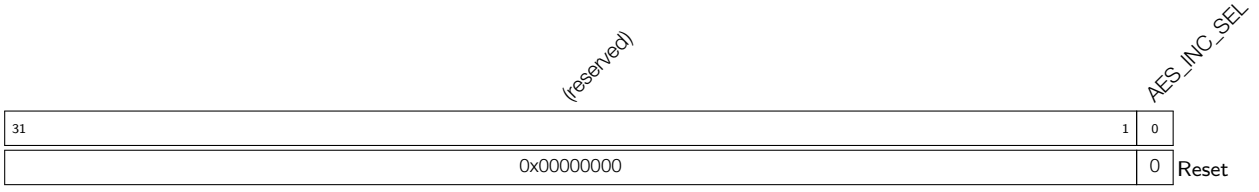
(R/W)

Register 17.7. AES_BLOCK_NUM_REG (0x0098)



AES_BLOCK_NUM 代表 DMA-AES 工作模式下待加解密的文本块数。详情请见章节 17.6.4。(R/W)

Register 17.8. AES_INC_SEL_REG (0x009C)



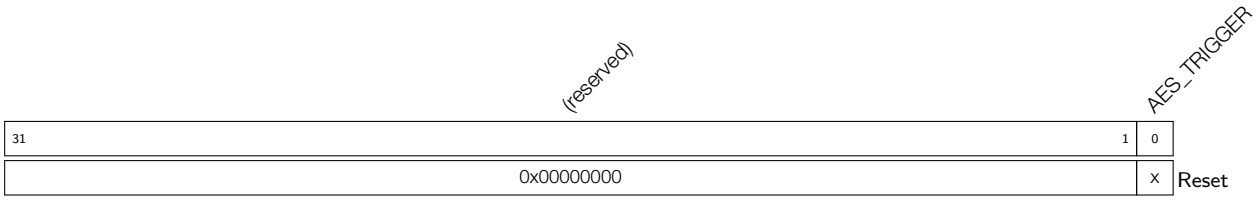
AES_INC_SEL 配置 CTR 块模式使用的标准增量函数。

0：INC_{32} 标准增量函数

1：INC_{128} 标准增量函数

(R/W)

Register 17.9. AES_TRIGGER_REG (0x0048)



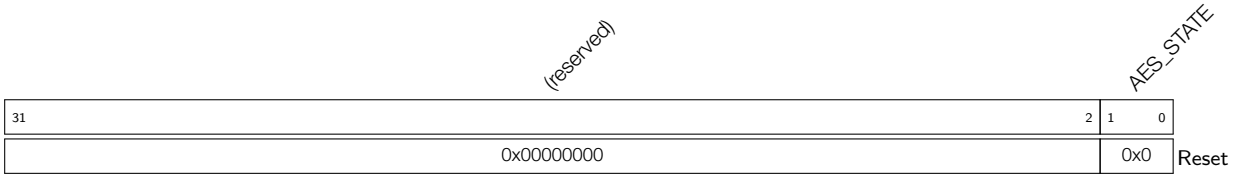
AES_TRIGGER 配置是否启动 AES 运算。

0：无效果

1：启动

(WT)

Register 17.10. AES_STATE_REG (0x004C)



AES_STATE 代表 AES 加速器的状态。

在 Typical AES 工作模式下，

0：IDLE

1：WORK

2：无效果

3：无效果

在 DMA-AES 工作模式下，

0：IDLE

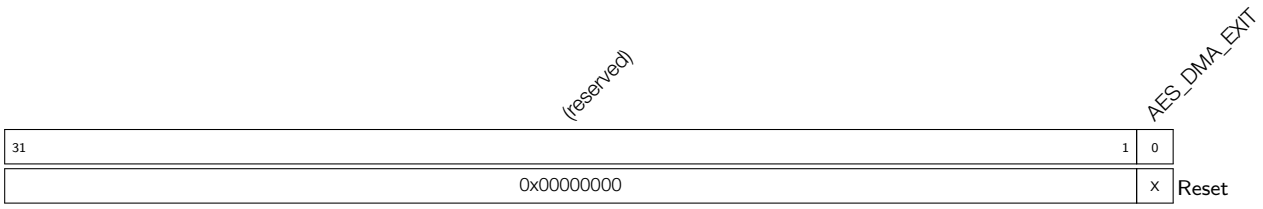
1：WORK

2：DONE

3：无效果

(RO)

Register 17.11. AES_DMA_EXIT_REG (0x00B8)



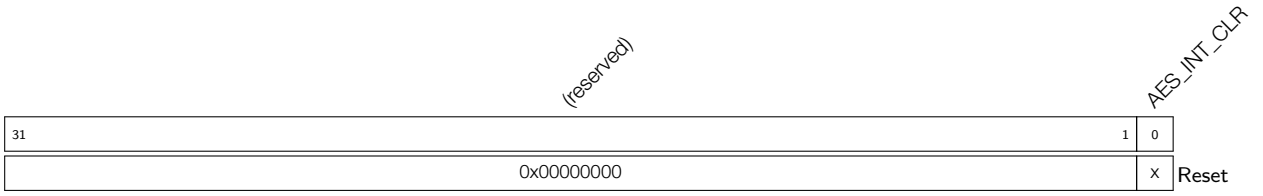
AES_DMA_EXIT 配置是否退出 AES 运算。

0：无效果

1：退出

仅在 DMA-AES 模式下有效。(WO)

Register 17.12. AES_INT_CLR_REG (0x00AC)



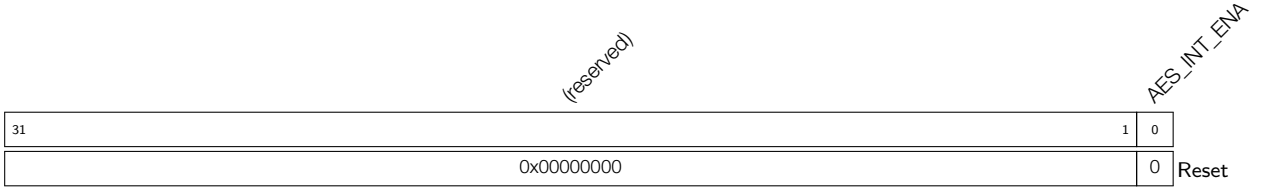
AES_INT_CLR 配置是否清除 AES 中断。

0：无效果

1：清除

(WT)

Register 17.13. AES_INT_ENA_REG (0x00B0)



AES_INT_ENA 配置是否使能 AES 中断功能。

0：禁用

1：使能

(R/W)

18 ECC 加速器 (ECC)

18.1 概述

椭圆曲线密码学 (Elliptic Curve Cryptography) 是一种基于椭圆曲线数学的公开密钥加密演算法，其优势在于相对于 RSA 算法，使用较小长度的密钥就能够提供相当等级的加密安全性。

ESP32-H2 ECC 硬件加速器支持基于可选曲线的多种基础运算，用以实现对 ECC 基本运算及其衍生算法（如 ECDSA 等算法）的加速。

18.2 主要特性

ESP32-H2 ECC 硬件加速器具备以下特性：

- 支持 2 种可选 ECC 曲线，即 FIPS 186-3 中定义的 P-192 和 P-256
- 提供 11 种可选工作模式
- 提供计算完成的中断和中断控制

18.3 ECC 背景知识

本节介绍了与 ECC 硬件加速器相关的背景知识以及章节中会应用到的专业名词。

18.3.1 椭圆曲线与曲线上的点

ECC 是一种基于大素数的有限域椭圆曲线的算法，这一椭圆曲线的数学表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

其中，

- p 是素数，
- a 和 b 为两个小于 p 的非负整数，
- (x, y) 为满足该椭圆曲线方程的点。

18.3.2 仿射坐标系与 Jacobian 坐标系

一条椭圆曲线：

- 在仿射坐标系下的表达式为：

$$y^2 = x^3 + ax + b \bmod p$$

- 在 Jacobian 坐标系下的表达式为：

$$Y^2 = X^3 + aXZ^4 + bZ^6 \bmod p$$

一个曲线上的点在仿射坐标系下的表示 (x, y) 与在 Jacobian 坐标系下的表示 (X, Y, Z) 有如下转换关系：

- 从 Jacobian 坐标系到仿射坐标系的转换：

$$x = X/Z^2 \bmod p$$

$$y = Y/Z^3 \bmod p$$

- 从仿射坐标系到 Jacobian 坐标系的转换：

$$X = x$$

$$Y = y$$

$$Z = 1$$

18.3.3 内存块

ECC 硬件加速器的内存块用于存储 ECC 运算中所用到的输入数据和输出数据。

表 18-1. ECC 硬件加速器内存块

内存块	大小（比特）	起始地址*	结束地址 *	访问权限
ECC_Mem_k	32	0x100	0x11F	R/W
ECC_Mem_Px	32	0x120	0x13F	R/W
ECC_Mem_Py	32	0x140	0x15F	R/W
ECC_Mem_Qx	32	0x160	0x17F	R/W
ECC_Mem_Qy	32	0x180	0x19F	R/W
ECC_Mem_Qz	32	0x1A0	0x1BF	R/W

* 采用相对于 ECC 加速器基地址的偏移量。详见章节 4 系统和存储器 中的表 4-2。

18.3.4 数据与数据块

在 ECC 硬件加速器模块中会用到的数据位宽均为 256 位，假设一个数据为 $D[255:0]$，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n = 0, 1, \cdots, 7)$，序号数低的数据块对应二进制低位，即：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

18.3.5 数据存储

数据存储即将数据存储进一个内存块的操作，该数据为 ECC 算法的输入数据。具体而言，将数据写入一个 ECC 内存块相当于将该数据 $D[n][31:0](n = 0, 1, \cdots, 7)$ 依次写入 “该内存块起始地址 + $4 \times n$”：

- 写入 $D[0]$ 至 “起始地址”
- 写入 $D[1]$ 至 “起始地址 + 4”
- ...
- 写入 $D[7]$ 至 “起始地址 + 28”

说明：
在 192 位模式下进行数据存储操作时，需要在数据的高位补 0，保证存储的数据为 256 位。

18.3.6 数据读取

数据读取即将一个数据从一个内存块读出的操作，该数据为 ECC 算法的输出数据。具体来说，从一个 ECC 内存块读数据相当于从 “该内存块起始地址 + $4 \times n$” 依次读出 $D[n][31:0](n = 0, 1, \cdots, 7)$：

- 从 “起始地址” 读出 $D[0]$

- 从“起始地址 + 4”读出 $D[1]$
- ...
- 从“起始地址 + 28”读出 $D[7]$

说明：
在 192 位模式下进行数据读取操作时，只需要读取低 192 位（即 6 个数据块）的数据。

18.3.7 标准运算与 Jacobian 运算

ESP32-H2 ECC 硬件加速器中，所有标准运算（包括标准点验证、标准点加和标准点乘）的输入数据以及输出数据中的点均在仿射坐标系中；相对应地，所有 Jacobian 运算（包括 Jacobian 点验证、Jacobian 点加和 Jacobian 点乘）的输入数据以及输出数据中的点均在 Jacobian 坐标系中。

18.4 功能描述

18.4.1 密钥长度模式

ESP32-H2 ECC 硬件加速器共支持 2 种密钥长度模式，每种密钥长度模式唯一对应一条椭圆曲线。用户通过配置 ECC_KEY_LENGTH 可选定密钥长度模式，其与椭圆曲线的对应关系如表 18-2 所示。

表 18-2. ECC 加速器密钥长度模式控制

ECC_KEY_LENGTH	对应椭圆曲线*
0	FIPS P-192
1	FIPS P-256

* FIPS P-192/P-256 的曲线定义，请参考 FIPS 186-3。

18.4.2 工作模式

ESP32-H2 ECC 硬件加速器共有 11 种工作模式，每种工作模式基于选定曲线进行不同操作。用户通过配置 ECC_WORK_MODE 可选定工作模式，其值与工作模式的对应关系如表 18-3 所示。

表 18-3. ECC 硬件加速器工作模式控制

ECC_WORK_MODE	对应模式
0	标准点乘
1	**保留项，不可用**
2	标准点验证
3	标准点验证 + 标准点乘
4	Jacobian 点乘
5	点加
6	Jacobian 点验证
7	标准点验证 + 标准点乘
8	模加
9	模减
10	模乘
11	模除

说明：
请注意，Jacobian 点乘模式的运算速度比标准点乘模式约快 10%。

每种工作模式的具体计算和输入/输出数据，请参照下述子章节。

18.4.2.1 标准点乘模式

标准点乘模式计算公式如下：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_Mem_Px、ECC_Mem_Py 和 ECC_Mem_k。
- 输出数据：Q_x 和 Q_y 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。

18.4.2.2 标准点验证模式

标准点验证模式用于计算点 (P_x, P_y) 是否在选定的椭圆曲线上。

- 输入数据：P_x 和 P_y 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
- 输出数据：点验证的结果存储在 ECC_VERIFICATION_RESULT 中。

18.4.2.3 标准点验证 + 标准点乘模式

标准点验证 + 标准点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y) = (J_x, J_y, J_z) = k \cdot (P_x, P_y)$$

其中，

- (Q_x, Q_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_Mem_Px、ECC_Mem_Py 和 ECC_Mem_k。
- 输出数据：
  - 点验证的结果存储在 ECC_VERIFICATION_RESULT 中。
  - Q_x 和 Q_y 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - J_x、J_y 和 J_z 对应的内存块为 ECC_Mem_Qx、ECC_Mem_Qy 和 ECC_Mem_Qz。

18.4.2.4 Jacobian 点乘模式

Jacobian 点乘模式计算公式如下：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_Mem_Px、ECC_Mem_Py 和 ECC_Mem_k。
- 输出数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_Mem_Qx、ECC_Mem_Qy 和 ECC_Mem_Qz。

18.4.2.5 点加模式

点加模式计算公式如下：

$$R = (R_x, R_y) = (J_x, J_y, J_z) = (P_x, P_y, 1) + (Q_x, Q_y, Q_z)$$

其中，

- (R_x, R_y) 为仿射表示的曲线上的点。
- (J_x, J_y, J_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：
  - P_x 和 P_y 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - Q_x、Q_y 和 Q_z 对应的内存块为 ECC_Mem_Qx、ECC_Mem_Qy 和 ECC_Mem_Qz。
- 输出数据：
  - R_x 和 R_y 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - J_x、J_y 和 J_z 对应的内存块为 ECC_Mem_Qx、ECC_Mem_Qy 和 ECC_Mem_Qz。

18.4.2.6 Jacobian 点验证模式

Jacobian 点验证模式用于计算点 (Q_x, Q_y, Q_z) 是否在选定的椭圆曲线上。

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。

- 输入数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_Mem_Qx、ECC_Mem_Qy 和 ECC_Mem_Qz。
- 输出数据：点验证的结果存储在 ECC_VERIFICATION_RESULT 中。

18.4.2.7 标准点验证 + Jacobian 点乘模式

标准点验证 + Jacobian 点乘模式先计算点 (P_x, P_y) 是否在选定的椭圆曲线上，如果其在选定的椭圆曲线上，则继续计算如下公式：

$$Q = (Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的曲线上的点。
- 输入点 P 的 Jacobian 表示中添加的 1 为硬件默认补全，不需要输入。
- 输入数据：P_x、P_y 和 k 对应的内存块为 ECC_Mem_Px, ECC_Mem_Py 和 ECC_Mem_k。
- 输出数据：
  - 点验证的结果存储在 ECC_VERIFICATION_RESULT 中。
  - Q_x、Q_y 和 Q_z 对应的内存块为 ECC_Mem_Qx、ECC_Mem_Qy 和 ECC_Mem_Qz。

18.4.2.8 模加模式

模加模式计算公式如下：

$$R = A + B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    - 配置 ECC_KEY_LENGTH 来选择曲线。
    - 配置 ECC_MOD_BASE 来选择使用曲线模数或曲线阶数。
- 输出数据：R 对应的内存块为 ECC_Mem_Px。

18.4.2.9 模减模式

模减模式计算公式如下：

$$R = A - B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    - 配置 ECC_KEY_LENGTH 来选择曲线。
    - 配置 ECC_MOD_BASE 来选择使用曲线模数或曲线阶数。
- 输出数据：R 对应的内存块为 ECC_Mem_Px。

18.4.2.10 模乘模式

模乘模式计算公式如下：

$$R = A \cdot B \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_KEY_LENGTH 来选择曲线。
    * 配置 ECC_MOD_BASE 来选择使用曲线模数或曲线阶数。
- 输出数据：R 对应的内存块为 ECC_Mem_Py。

18.4.2.11 模除模式

模除模式计算公式如下：

$$R = A \cdot B^{-1} \bmod N$$

其中，

- 输入数据：
  - A 和 B 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。
  - N 的值可通过以下寄存器字段配置：
    * 配置 ECC_KEY_LENGTH 来选择曲线。
    * 配置 ECC_MOD_BASE 来选择使用曲线模数或曲线阶数。
- 输出数据：R 对应的内存块为 ECC_Mem_Py。

18.5 时钟与复位

ESP32-H2 ECC 硬件加速器模块仅有一个模块时钟 CRYPTO_ECC_CLK 和一个模块复位 CRYPTO_ECC_RST。

ECC 时钟与复位由电源/时钟/复位寄存器 (PCR) 处理，详见章节 7 复位和时钟。在使用 ECC 硬件加速器之前，需要配置 PCR_ECC_CLK_EN 来启用 ECC 时钟，并清除 PCR_ECC_RST_EN 来释放 ECC 复位。此外，由于密码学加速器之间的资源复用，还需额外清除 PCR_ECDSA_RST_EN。

18.6 中断

ESP32-H2 ECC 硬件加速器可产生一个中断信号 ECC_INTR，并将其发送给 中断矩阵。

说明：
每个中断信号均由其包含的所有中断源的状态位共同产生，即任意一个其包含的中断源的状态位触发，该中断信号就会被触发。

ECC 硬件加速器的中断信号 ECC_INTR 仅包含中断源 ECC_CALC_DONE_INT，ECC 硬件加速器运算完成即触发该中断源。中断源 ECC_CALC_DONE_INT 由以下寄存器位控制：

- ECC_CALC_DONE_INT_RAW：ECC 硬件加速器运算完成时置 1。
- ECC_CALC_DONE_INT_ST：反映 ECC 硬件加速器运算完成中断的状态，通过用 ECC_CALC_DONE_INT_ENA 使能/屏蔽 ECC_CALC_DONE_INT_RAW 位来生成。
- ECC_CALC_DONE_INT_ENA：用于使能或屏蔽 ECC 硬件加速器运算完成中断状态位。
- ECC_CALC_DONE_INT_CLR：置 1 此位清除 ECC 硬件加速器运算完成中断，对应的 ECC_CALC_DONE_INT_RAW 和 ECC_CALC_DONE_INT_ST 位会清零。

18.7 软件配置流程

软件配置 ECC 硬件加速器的流程如下：

1. 配置 ECC 硬件加速器的时钟与复位。配置细节请参考 18.5 小节。
2. 根据 18.4 小节的描述，按照需求配置 ECC 加速器密钥长度模式和工作模式。
3. 根据 18.6 小节的描述，使能 ECC_CALC_DONE_INT 中断。
4. 置位 ECC_START 以启动 ECC 运算。
5. 等待 ECC_CALC_DONE_INT 中断的产生，即 ECC 运算结束。
6. 根据 18.4 小节的描述，查看运算结果。

18.8 寄存器列表

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

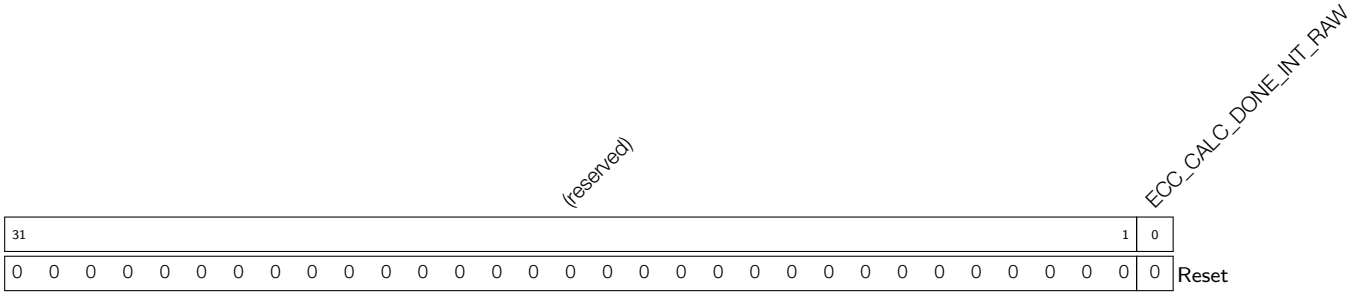
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
中断寄存器			
ECC_MULT_INT_RAW_REG	原始中断状态寄存器	0x000C	R/SS/WTC
ECC_MULT_INT_ST_REG	中断屏蔽状态寄存器	0x0010	RO
ECC_MULT_INT_ENA_REG	中断使能寄存器	0x0014	R/W
ECC_MULT_INT_CLR_REG	中断清除寄存器	0x0018	WT
配置寄存器			
ECC_MULT_CONF_REG	ECC 加速器配置寄存器	0x001C	varies
版本寄存器			
ECC_MULT_DATE_REG	版本控制寄存器	0x00FC	R/W

18.9 寄存器

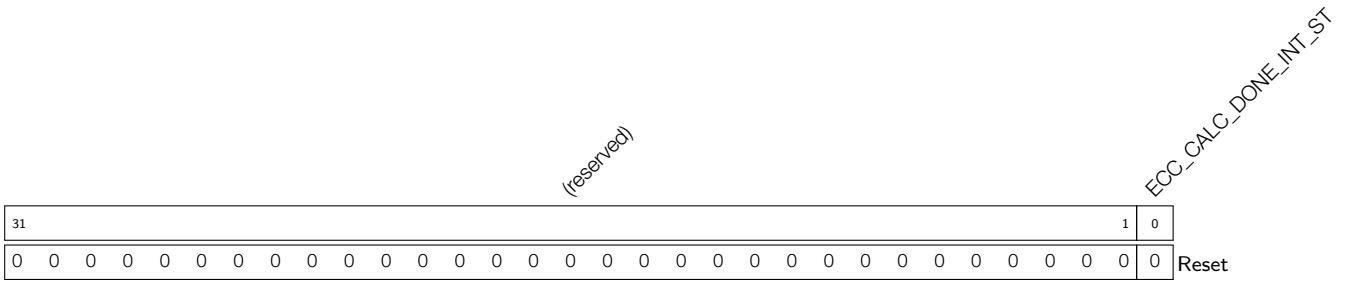
本小节的所有地址均为相对于 ECC 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 18.1. ECC_MULT_INT_RAW_REG (0x000C)



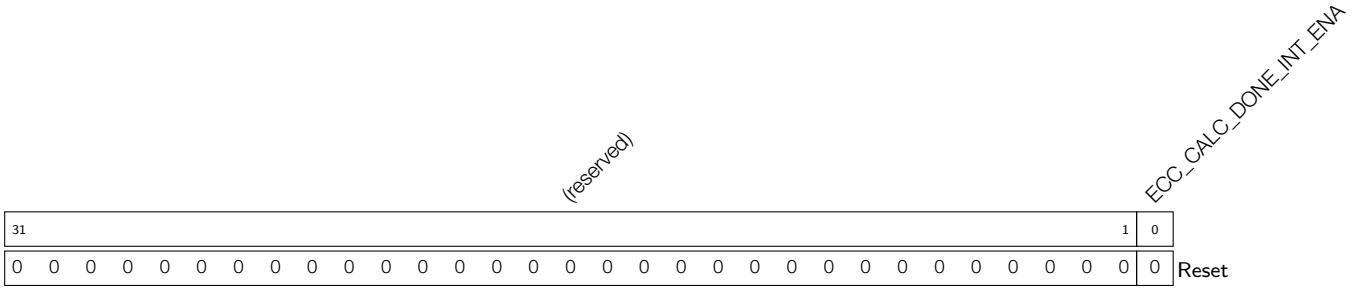
ECC_CALC_DONE_INT_RAW ECC_CALC_DONE_INT 的原始中断状态。(R/SS/WTC)

Register 18.2. ECC_MULT_INT_ST_REG (0x0010)



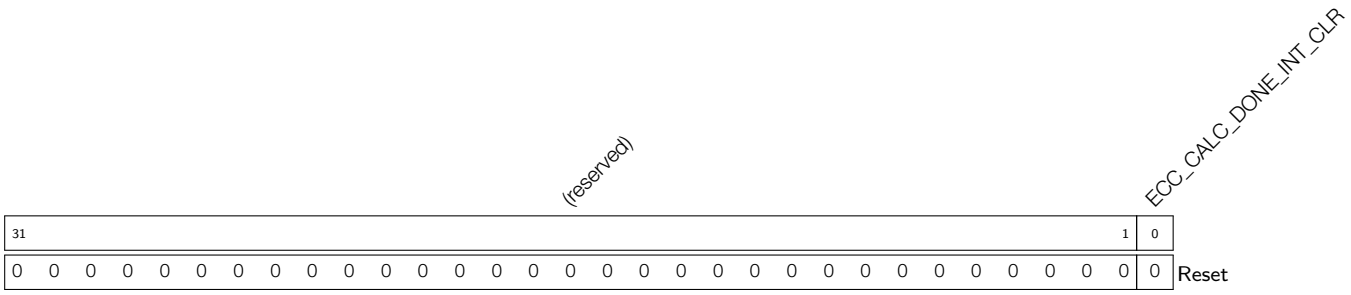
ECC_CALC_DONE_INT_ST ECC_CALC_DONE_INT 的屏蔽中断状态。(RO)

Register 18.3. ECC_MULT_INT_ENA_REG (0x0014)



ECC_CALC_DONE_INT_ENA 写 1 使能 ECC_CALC_DONE_INT 中断。(R/W)

Register 18.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_CALC_DONE_INT_CLR 写 1 清除 ECC_CALC_DONE_INT 中断。(WT)

Register 18.5. ECC_MULT_CONF_REG (0x001C)

ECC_MEM_CLOCK_GATE_FORCE_ON	ECC_CLK_EN	ECC_VERIFICATION_RESULT	(reserved)	ECC_WORK_MODE	ECC_MOD_BASE	ECC_KEY_LENGTH	ECC_RESET	ECC_START	
31	30	29	28 … 8	7 … 4	3	2	1	0	
1	0	0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	0	0	0	0	Reset

ECC_START 配置是否启动 ECC 加速器。此位运算结束后自动清 0。

0：无效

1：启动 ECC 加速器

(R/W/SC)

ECC_RESET 配置是否复位 ECC 加速器。

0：无效

1：复位

(WT)

ECC_KEY_LENGTH 配置 ECC 加速器的密钥长度。

0：P-192

1：P-256

(R/W)

ECC_MOD_BASE 配置在模运算操作中选择使用曲线模数或曲线阶数。仅在工作模式 8-11 中有效。

0: n（曲线阶数）

1: p（曲线模数）

(R/W)

ECC_WORK_MODE 配置 ECC 加速器的工作模式。

0：标准点乘

1：保留项，不可用

2：标准点验证

3：标准点验证 + 标准点乘

4：Jacobian 点乘

5：点加

6：Jacobian 点验证

7：标准点验证 + Jacobian 点乘

8：模加

9：模减

10：模乘

11：模除

(R/W)

见下页...

Register 18.5. ECC_MULT_CONF_REG (0x001C)

接上页...

ECC_VERIFICATION_RESULT 表示 ECC 加速器的验证结果，仅在运算完成时有效。

0：验证失败

1：验证成功

(R/SS)

ECC_CLK_EN 配置是否强制打开寄存器门控。

0：无效

1：强制打开

(R/W)

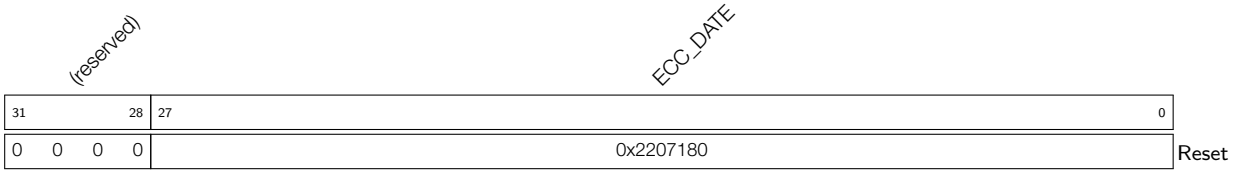
ECC_MEM_CLOCK_GATE_FORCE_ON 配置是否强制打开 ECC 内存时钟门控。

0：无效

1：强制打开

(R/W)

Register 18.6. ECC_MULT_DATE_REG (0x00FC)



ECC_DATE 版本控制寄存器。(R/W)

19 HMAC 加速器 (HMAC)

如 RFC 2104 中所述，HMAC 模块通过 hash 算法 SHA-256 和密钥计算得到数据信息的信息认证码 (MAC)。长度为 256 位的 HMAC 密钥存储在 eFuse 的密钥块中，可配置为读保护，即用户无法直接从 HMAC 加速器外部访问密钥。

19.1 主要特性

- 使用标准 HMAC-SHA-256 算法
- 仅支持可配的硬件外设访问 HMAC 计算的 hash 结果（下行模式）
- 兼容挑战-应答身份验证算法
- 支持生成数字签名算法 (DSA) 外设所需的密钥（下行模式）
- 支持重启软禁用的 JTAG（下行模式）

19.2 功能描述

HMAC 模块有两种工作模式，即上行模式和下行模式。上行模式中，由用户提供 HMAC 信息并回读计算结果；下行模式中，HMAC 模块作为其他内部硬件的密钥导出函数 (KDF)。例如，烧写奇数个 1 至 EFUSE_SOFT_DIS_JTAG，可暂时关闭 JTAG。此时，用户可使用 HMAC 模块在下行模式下暂时重启 JTAG。

芯片释放复位信号后，HMAC 模块将检查 eFuse 中是否烧写有 DSA 模块所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行数字签名算法模式，完成相应密钥计算。

19.2.1 上行模式

使用上行模式的较多为支持 HMAC-SHA-256 算法的挑战-应答协议。假设挑战-应答协议中的通讯双方分别称为 A 和 B，想要交换的数据信息为 M，则协议的一般验证流程为：

- A 计算出一个特殊的随机数 M。
- A 将 M 发送给 B。
- B（通过 M 和密钥）计算 HMAC 结果，并将结果发送给 A。
- A（通过 M 和密钥）内部计算 HMAC 结果。
- A 比较两次计算结果。如比较结果相同，则验证通过 B 的身份。

计算 HMAC 值，用户应执行以下步骤：

1. 初始化 HMAC 模块，进入上行模式。
2. 将正确填充的信息写入 HMAC 中，一次写入一个块。
3. 从 HMAC 回读计算结果。

有关此过程的详细步骤，可参见章节 19.2.5。

19.2.2 下行 JTAG 启动模式

JTAG 可被暂时关闭，关闭后，用户可使用 HMAC 模块重启 JTAG 功能。该模式下，用户应提供某 eFuse 密钥的 HMAC 计算结果，HMAC 模块将检查用户提供的数值与计算的数值是否匹配。如果二者相同，JTAG 将被启动，直到用户再次调用 HMAC 模块清除计算结果并关闭 JTAG。

eFuse 存储器中有两个参数可以关闭 JTAG 调试：EFUSE_DIS_PAD_JTAG 和 EFUSE_SOFT_DIS_JTAG。前者烧写为 1，JTAG 将被永久关闭；后者烧写奇数个 1，JTAG 将被暂时关闭。详细信息可参见章节 5 *eFuse 控制器 (EFUSE)*。置位 EFUSE_SOFT_DIS_JTAG 后，可在 HMAC 模块的下行模式下计算得到重启 JTAG 所需的密钥。当用户配置的密钥与计算结果一致时，完成 JTAG 重启。

重启 JTAG，用户应执行以下步骤：

1. 通过初始化时钟和 HMAC 复位信号启动 HMAC 模块，并通过配置 HMAC_SET_PARA_PURPOSE_REG 位进入下行 JTAG 启动模式，等待计算完成。详细步骤可参见章节 19.2.5。
2. 将 1 写入 HMAC_SOFT_JTAG_CTRL_REG 寄存器，进入 JTAG 重启比较模式。
3. 将 256 位的 HMAC 值写入寄存器 HMAC_WR_JTAG_REG。该值是预先在本地使用 SHA-256 和已知的随机密钥对 32 字节的 0x00 进行 HMAC 计算得到的。需以大端字序分 8 次写入，每次 32 位。
4. 如果 HMAC 计算的结果与用户本地计算的数值结果一致，则 JTAG 重启。否则，JTAG 保持关闭状态。
5. 在用户将 1 烧写入寄存器 HMAC_SET_INVALIDATE_JTAG_REG 或复位重启之前，JTAG 将保持关闭状态。如需再次重启 JTAG，请重复上述步骤。

19.2.3 下行数字签名算法模式

数字签名算法 (DSA) 模块使用 AES-CBC 算法加密其参数。HMAC 模块作为密钥导出函数 (KDF) 导出解密上述参数的 AES 密钥。这些用于 HMAC 的密钥作为 KDF 存储于 eFuse 密钥块中。

在启用 DSA 模块之前，用户必须先通过 HMAC 模块计算得到 DSA 模块工作时所需的密钥。详细信息可参见章节 22 *数字签名算法 (DSA)*。芯片上电后，HMAC 模块将检查 eFuse 中是否烧写有 DSA 模块所需要的功能密钥。如果存在该密钥，HMAC 模块将自动进入下行数字签名算法模式，完成相应密钥计算。

19.2.4 HMAC eFuse 配置

每个烧写入 eFuse 密钥块的 HMAC 密钥都有其功能，指定该密钥用于实现哪一功能。HMAC 中，未配置匹配的功能数值的密钥无法执行计算。当前 HMAC 模块共支持三种功能：下行模式下的 JTAG 重启功能和 DSA 密钥导出功能，以及上行模式下的 HMAC 计算功能。表 19-1 列出了上述的各项功能，以及一种可指定密钥同时用于 JTAG 重启和 DSA 密钥导出的功能。

启动 HMAC 模块进行计算之前，用户应先读取章节 5 *eFuse 控制器 (EFUSE)* 中 EFUSE_KEY_PURPOSE_x 的值（基于 eFuse 中共存在 6 个密钥，故 x 的值为 0 ~ 5，其中 EFUSE_KEY_PURPOSE_0 ~ EFUSE_KEY_PURPOSE_1 属于寄存器 EFUSE_RD_REPEAT_DATA1_REG，EFUSE_KEY_PURPOSE_2 ~ EFUSE_KEY_PURPOSE_5 属于寄存器 EFUSE_RD_REPEAT_DATA2_REG），检查 eFuse 中是否已烧写密钥。例如，在上行模式下，如果 EFUSE_KEY_PURPOSE_0~5 中没有 EFUSE_KEY_PURPOSE_HMAC_UP，则说明 eFuse 中没有上行模式可用的密钥。此时，可按照下述步骤将密钥烧写入 eFuse 中：

1. 准备一个 256 位 HMAC 密钥，将其烧写到空的 eFuse 密钥块 y 中。eFuse 中共有 6 个密钥块，分别为 4 ~ 9，因此 y 的值为 4 ~ 9。也就是说，若密钥为 key0，则对应的密钥块为 block4。接着，将功能对应数值烧写至 EFUSE_KEY_PURPOSE_$(y-4)$。例如在上行模式下，烧写密钥后，应将 EFUSE_KEY_PURPOSE_HMAC_UP（对应值为 6）写入 EFUSE_KEY_PURPOSE_$(y-4)$。更多烧写 eFuse 密钥的详细信息，可参见章节 5 *eFuse 控制器 (EFUSE)*。

2. 配置 eFuse 密钥块读保护功能，使用户无法读取密钥值。用户应保留步骤中生成的随机密钥副本以便验证。

请注意，功能为 EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL 的密钥既可用于 JTAG 重启，也可用于 DSA 密钥导出。

表 19-1. HMAC 功能及配置数值

功能	模式	数值	描述
JTAG 重启	下行模式	6	EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG
DS 密钥导出	下行模式	7	EFUSE_KEY_PURPOSE_HMAC_DOWN_DIGITAL_SIGNATURE
HMAC 计算	上行模式	8	EFUSE_KEY_PURPOSE_HMAC_UP
JTAG 重启和 DS 密钥导出	下行模式	5	EFUSE_KEY_PURPOSE_HMAC_DOWN_ALL

配置 HMAC 的功能

用户应将所使用功能对应的数值写入寄存器 HMAC_SET_PARA_PURPOSE_REG 完成配置（可参见章节 19.2.5）。否则，HMAC 将终止计算。

选取 eFuse 的密钥块

eFuse 控制器共有 6 个密钥块，即 KEY0 ~ 5 。用户将编号 n 写入寄存器 HMAC_SET_PARA_KEY_REG，表示选择 KEYn 作为本次 HMAC 模块运行时使用的密钥。

需要注意的是，eFuse 存储器中的密钥在烧写时都定义了功能用途，只有当 HMAC 的配置功能与 KEYn 定义的功能用途相匹配时，HMAC 模块才会执行配置好的计算任务。否则，HMAC 模块将返回匹配错误结果并结束当前计算任务。

比如，如果用户选择了 KEY3 作为本次计算的密钥，且烧写入 EFUSE_KEY_PURPOSE_3 中的数值为 6，即功能 EFUSE_KEY_PURPOSE_HMAC_DOWN_JTAG。参照表 19-1 可知，KEY3 是用于 JTAG 重启的密钥。如果此时寄存器 HMAC_SET_PARA_PURPOSE_REG 中配置的数值也是 6，HMAC 模块才会启动 JTAG 重启功能的计算。

19.2.5 调用 HMAC 流程（详细说明）

用户调用 ESP32-H2 中 HMAC 流程如下：

1. 启动 HMAC 模块：
   (a) 置位寄存器 PCR_HMAC_CLK_EN 中 HMAC 和 SHA 的外设时钟位，清除寄存器 PCR_HMAC_RST_EN 中相应的外设复位位。关于如何配置时钟和重启信号，请参见章节 7 复位和时钟。
   (b) 将 1 写入寄存器 HMAC_SET_START_REG。
2. 配置 HMAC 密钥和密钥功能：
   (a) 将密钥功能 m 写入寄存器 HMAC_SET_PARA_PURPOSE_REG。表 19-1 列出了 m 对应的数值，可参见章节 19.2.4。
   (b) 将数值 n 写入寄存器 HMAC_SET_PARA_KEY_REG，选择 eFuse 存储器中的 KEYn 作为本次计算的密钥（n 的取值范围为 0 ~ 5），可参见章节 19.2.4。
   (c) 将 1 写入寄存器 HMAC_SET_PARA_FINISH_REG，完成配置。

(d) 读取寄存器 HMAC_QUERY_ERROR_REG。如果返回值为 1，表明选取的密钥块与配置的密钥功能不匹配，本次计算任务中止。如果返回值为 0，表明选取的密钥块与配置的密钥功能匹配，可以执行计算流程。

(e) 如果 HMAC_SET_PARA_PURPOSE_REG 的数值不为 8，表明 HMAC 模块当前处于下行模式，请跳转到步骤 3。如果数值为 8，表明 HMAC 模块当前处于上行模式，请跳转到步骤 4。

3. 下行模式：

(a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

(b) 如需清除计算结果，并确保后续不再使用相关硬件（JTAG 或 DSA），用户可将 1 写入寄存器 HMAC_SET_INVALIDATE_JTAG_REG，清除 JTAG 密钥生成的结果；也可将 1 写入寄存器 HMAC_SET_INVALIDATE_DS_REG，清除数字签名算法密钥生成的结果。此后，如需重启 JTAG 或 DSA 模块，需重新进行 HMAC 调用流程。

4. 上行模式下传输数据块 Block_n（$n >= 1$）：

(a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

(b) 将 512 位的数据块 Block_n 写入寄存器 HMAC_WDATA0~15_REG 中。随后，将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG，HMAC 模块将计算该数据块。

(c) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

(d) 根据待处理数据总位数是否为 512 的整数倍，将产生不同数据块。

- 如果待处理数据总位数是 512 的整数倍，有以下 3 种选项：
  i. 如果 Block_n+1 存在，则将 1 写入寄存器 HMAC_SET_MESSAGE_ING_REG，令 $n = n+1$，随后跳转到步骤 4.(b)。
  ii. 如果 Block_n 是最后一个待处理数据块，用户希望由硬件进行 SHA 附加填充，则将 1 写入寄存器 HMAC_SET_MESSAGE_END_REG，随后跳转到步骤 6。
  iii. 如果 Block_n 是最后一个填充的数据块，且用户已在软件中进行 SHA 附加填充，则将 1 写入寄存器 HMAC_SET_MESSAGE_PAD_REG，随后跳转到步骤 5。
- 如果待处理数据总位数不是 512 的倍数，有以下 3 种选项。注意，这种情况下用户应对数据进行 SHA 附加填充，且填充后待输入数据总位数应为 512 的整数倍。
  i. 如果 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位，则将 1 写入寄存器 HMAC_ONE_BLOCK_REG，随后跳转到步骤 6。
  ii. 如果 Block_n 是倒数第二个填充数据块，则将 1 写入寄存器 HMAC_SET_MESSAGE_PAD_REG，随后跳转到步骤 5。
  iii. 如果 Block_n 既不是最后一个也不是倒数第二个数据块，则将 1 写入寄存器 HMAC_SET_MESSAGE_ING_REG，令 $n = n+1$，随后跳转到步骤 4.(b)。

5. 进行 SHA 附加填充：

(a) 用户根据章节 19.3.1 的描述对最后一个数据块进行 SHA 附加填充，并将该数据块写入寄存器 HMAC_WDATA0~15_REG，随后将 1 写入寄存器 HMAC_SET_MESSAGE_ONE_REG，HMAC 模块开始计算该数据块。

(b) 跳转到步骤 6。

6. 读取上行模式下的结果 hash 值：

(a) 轮询状态寄存器 HMAC_QUERY_BUSY_REG，当读取到该寄存器的值为 0 时，继续下一步骤。

(b) 从寄存器 HMAC_RD_RESULT_*n*_REG (*n*: 0-7) 中读取 hash 结果数值。

(c) 将 1 写入寄存器 HMAC_SET_RESULT_FINISH_REG，结束当次计算，并同时清除计算结果。

(d) 上行模式下的操作完成。

说明：

DSA 模块和 HMAC 模块可直接调用或在内部使用 SHA 加速器，但两者不能同时共享硬件资源。因此在 HMAC 模块运行过程中，SHA 模块无法被 CPU 和 DSA 模块调用。

19.3 HMAC 算法细节

19.3.1 附加填充位

HMAC 模块中采用 SHA-256 作为加密 HASH 算法。该算法中，若待输入数据的总位数不是 512 的倍数，用户须在软件中应用 SHA-256 附加填充算法。SHA-256 附加填充算法与 FIPS PUB 180-4 中章节 *Padding the Message* 所描述相同。下行模式下，用户无需输入数据或进行数据填充，HMAC 模块默认使用 32 字节 0x00 方式重启 JTAG，以及 32-byte 0xff 方式生成 DSA 模块的 AES 密钥。

如图 19-1 所示，假设待处理数据长度为 m 个位，填充步骤如下：

1. 在待处理数据末尾附加一个 “1”。
2. 附加 k 个 “0”。其中，k 为满足 $m+1+k \equiv 448 (mod 512)$ 的最小非负数。
3. 附加一个 64 位的整数值作为二进制块。该二进制块的内容为待填充数据作为一个大端二进制整数值 m 的长度。

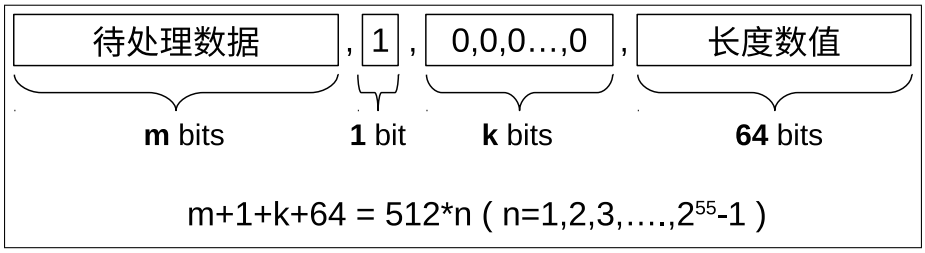


图 19-1. HMAC 附加填充位示意图

上行模式下，若待填充数据总位数是 512 的整数倍，则用户可通过将 1 写入 HMAC_SET_MESSGAE_END_REG 配置由硬件完成 SHA 附加填充操作，也可通过将 1 写入 HMAC_SET_MESSAGE_PAD_REG 自行完成填充操作。若待填充数据总位数不是 512 的整数倍，则用户只能自行完成 SHA 附加填充操作。数据填充操作完成后，用户应参照章节 19.2.5 完成后续配置。

19.3.2 HMAC 算法结构

HMAC 模块中应用的算法结构示意图如 19-2 所示。这是 RFC 2104 中描述的标准 HMAC 算法。

图 19-2 中，

1. ipad 是由 64 个 0x36 字节组成的 512 位数据块。
2. opad 是由 64 个 0x5c 字节组成的 512 位数据块。

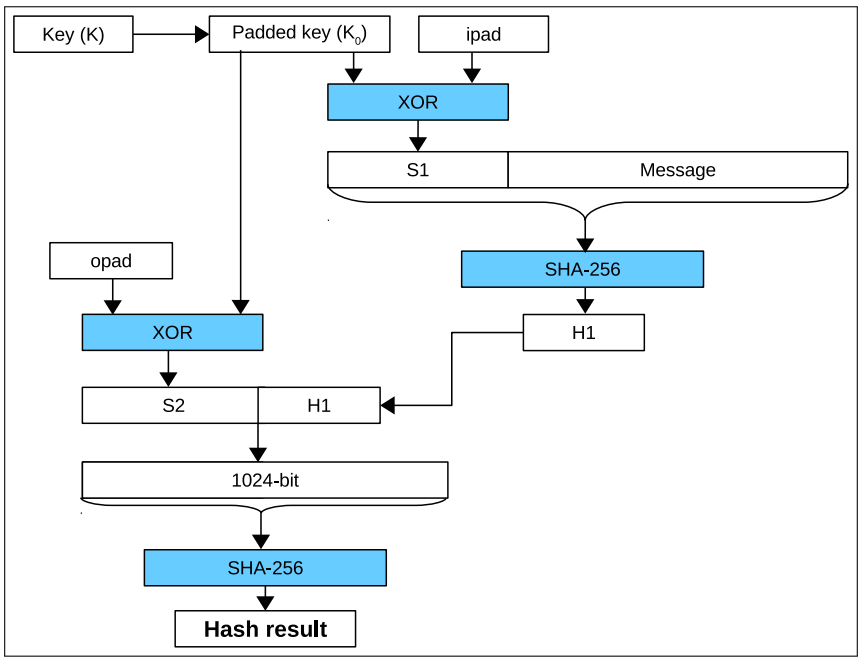


图 19-2. HMAC 结构示意图

首先，HMAC 模块在 256 位的密钥 K 的位序列后附加上 256 位的 0 序列，得到 512 位的 K_0。再对 K_0 和 ipad 进行异或运算，得到 512 位的 S1。将总位数为 512 倍数的待输入数据附加到 512 位的 S1 数值后，使用 SHA-256 加密算法计算得到 256 位的 H1。

HMAC 模块通过对 K_0 和 opad 进行异或运算得到 S2，将 256 位的 hash 计算结果附加到 512 位的 S2 数值后，得到 768 位长度的序列，使用章节 19.3.1 中描述的 SHA 附加填充算法将该序列填充成 1024 位的序列，最后使用 SHA-256 加密算法计算得到的最终 hash 结果（256 位）。

19.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

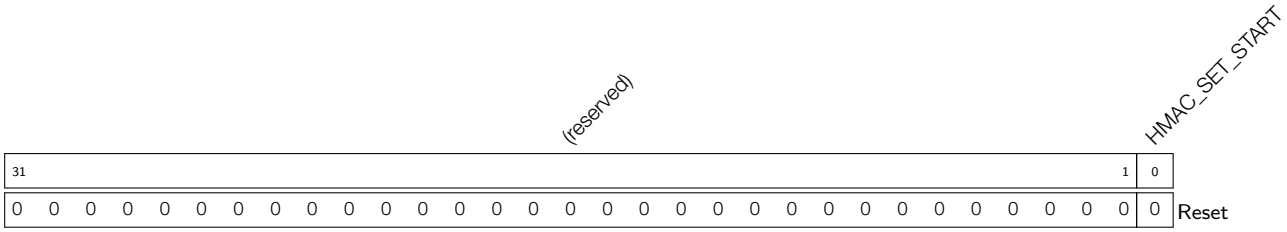
名称	描述	地址	访问
控制/状态寄存器			
HMAC_SET_START_REG	HMAC 开始控制寄存器	0x0040	WO
HMAC_SET_PARA_FINISH_REG	HMAC 配置完成寄存器	0x004C	WO
HMAC_SET_MESSAGE_ONE_REG	HMAC 信息控制寄存器	0x0050	WO
HMAC_SET_MESSAGE_ING_REG	HMAC 信息继续寄存器	0x0054	WO
HMAC_SET_MESSAGE_END_REG	HMAC 信息终止寄存器	0x0058	WO
HMAC_SET_RESULT_FINISH_REG	HMAC 读取结果完成寄存器	0x005C	WO
HMAC_SET_INVALIDATE_JTAG_REG	注销 JTAG 结果寄存器	0x0060	WO
HMAC_SET_INVALIDATE_DS_REG	注销数字签名结果寄存器	0x0064	WO
HMAC_QUERY_ERROR_REG	存储用户配置的密钥和功能的配对结果	0x0068	RO
HMAC_QUERY_BUSY_REG	存储 HMAC 模块的忙碌状态	0x006C	RO
HMAC_SET_MESSAGE_PAD_REG	软件填充寄存器	0x00F0	WO
HMAC_ONE_BLOCK_REG	One block 信息寄存器	0x00F4	WO
配置寄存器			
HMAC_SET_PARA_PURPOSE_REG	HMAC 参数配置寄存器	0x0044	WO
HMAC_SET_PARA_KEY_REG	HMAC 密钥配置寄存器	0x0048	WO
HMAC_SOFT_JTAG_CTRL_REG	重启 JTAG 寄存器 0	0x00F8	WO
HMAC_WR_JTAG_REG	重启 JTAG 寄存器 1	0x00FC	WO
HMAC 信息块			
HMAC_WR_MESSAGE_0_REG	信息寄存器 0	0x0080	WO
HMAC_WR_MESSAGE_1_REG	信息寄存器 1	0x0084	WO
HMAC_WR_MESSAGE_2_REG	信息寄存器 2	0x0088	WO
HMAC_WR_MESSAGE_3_REG	信息寄存器 3	0x008C	WO
HMAC_WR_MESSAGE_4_REG	信息寄存器 4	0x0090	WO
HMAC_WR_MESSAGE_5_REG	信息寄存器 5	0x0094	WO
HMAC_WR_MESSAGE_6_REG	信息寄存器 6	0x0098	WO
HMAC_WR_MESSAGE_7_REG	信息寄存器 7	0x009C	WO
HMAC_WR_MESSAGE_8_REG	信息寄存器 8	0x00A0	WO
HMAC_WR_MESSAGE_9_REG	信息寄存器 9	0x00A4	WO
HMAC_WR_MESSAGE_10_REG	信息寄存器 10	0x00A8	WO
HMAC_WR_MESSAGE_11_REG	信息寄存器 11	0x00AC	WO
HMAC_WR_MESSAGE_12_REG	信息寄存器 12	0x00B0	WO
HMAC_WR_MESSAGE_13_REG	信息寄存器 13	0x00B4	WO
HMAC_WR_MESSAGE_14_REG	信息寄存器 14	0x00B8	WO
HMAC_WR_MESSAGE_15_REG	信息寄存器 15	0x00BC	WO
HMAC 上行结果			
HMAC_RD_RESULT_0_REG	Hash 结果寄存器 0	0x00C0	RO
HMAC_RD_RESULT_1_REG	Hash 结果寄存器 1	0x00C4	RO

名称	描述	地址	访问
HMAC_RD_RESULT_2_REG	Hash 结果寄存器 2	0x00C8	RO
HMAC_RD_RESULT_3_REG	Hash 结果寄存器 3	0x00CC	RO
HMAC_RD_RESULT_4_REG	Hash 结果寄存器 4	0x00D0	RO
HMAC_RD_RESULT_5_REG	Hash 结果寄存器 5	0x00D4	RO
HMAC_RD_RESULT_6_REG	Hash 结果寄存器 6	0x00D8	RO
HMAC_RD_RESULT_7_REG	Hash 结果寄存器 7	0x00DC	RO
版本寄存器			
HMAC_DATE_REG	版本控制寄存器	0x01FC	R/W

19.5 寄存器

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

Register 19.1. HMAC_SET_START_REG (0x0040)



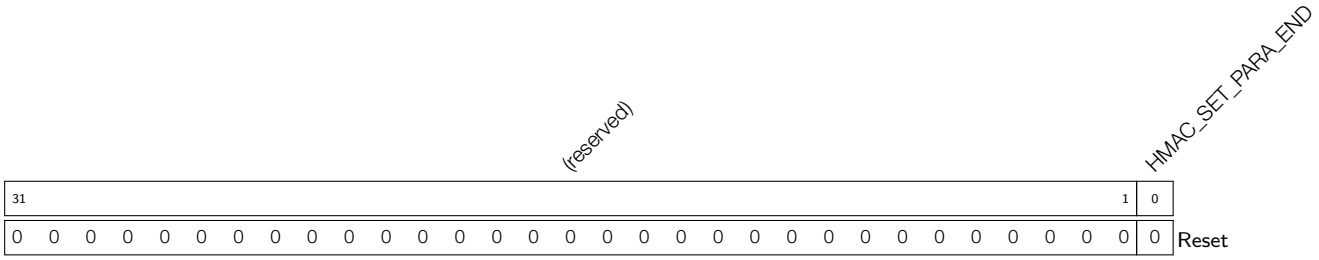
HMAC_SET_START 配置是否启动 HMAC。

0：关闭 HMAC

1：启动 HMAC

(WO)

Register 19.2. HMAC_SET_PARA_FINISH_REG (0x004C)



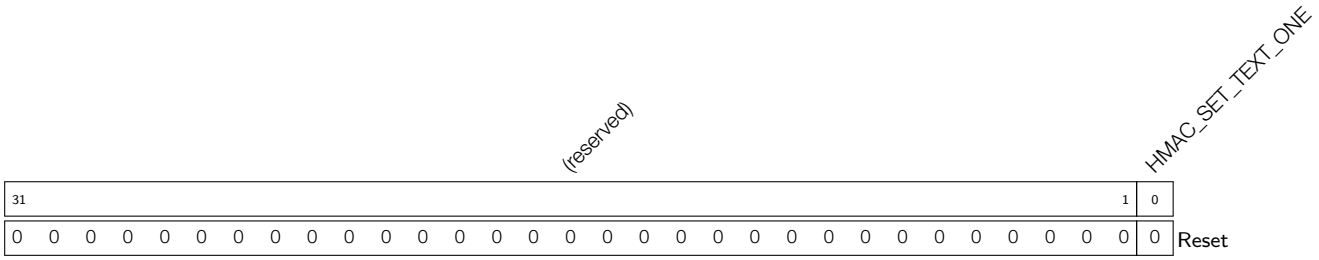
HMAC_SET_PARA_END 配置是否完成 HMAC 配置。

0：无效

1：完成配置

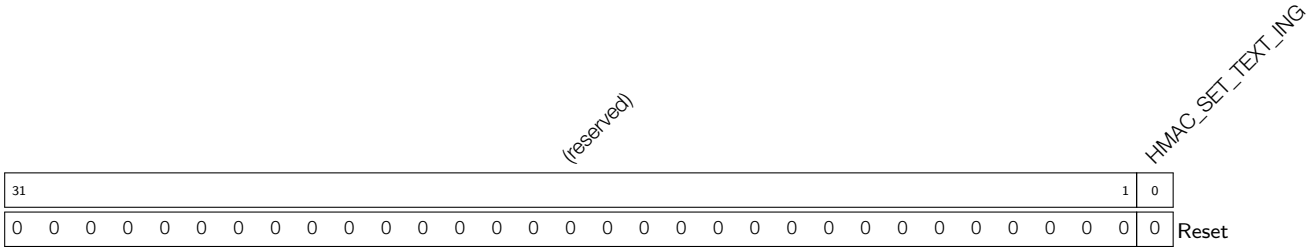
(WO)

Register 19.3. HMAC_SET_MESSAGE_ONE_REG (0x0050)



HMAC_SET_TEXT_ONE 调用 SHA 计算一个数据块。(WO)

Register 19.4. HMAC_SET_MESSAGE_ING_REG (0x0054)



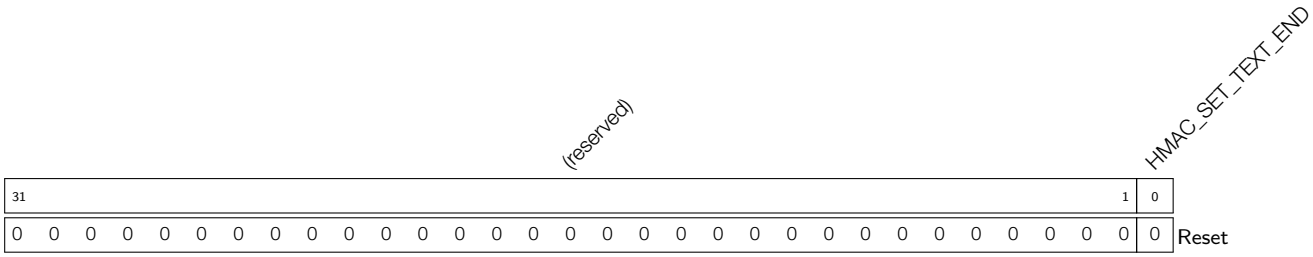
HMAC_SET_TEXT_ING 配置是否存在未处理的数据块。

0：不存在

1：存在

(WO)

Register 19.5. HMAC_SET_MESSAGE_END_REG (0x0058)



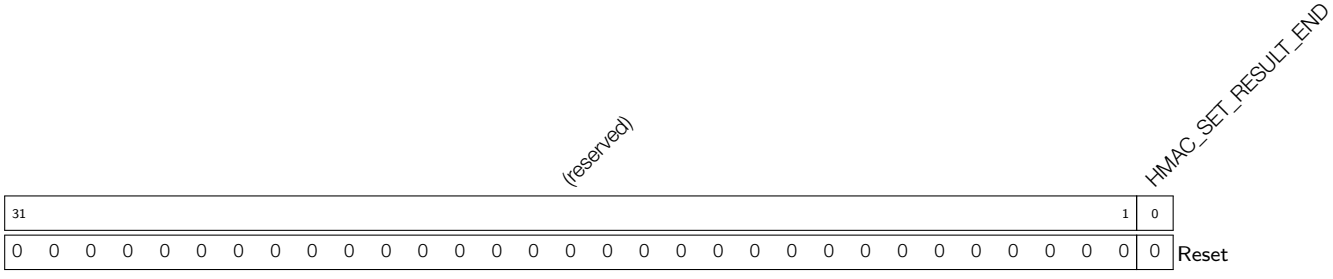
HMAC_SET_TEXT_END 配置是否开始硬件填充。

0：无效

1：开始硬件填充

(WO)

Register 19.6. HMAC_SET_RESULT_FINISH_REG (0x005C)



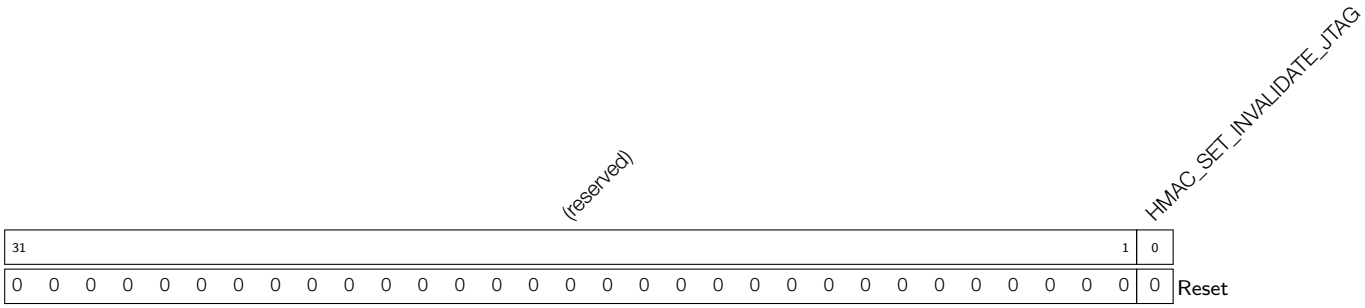
HMAC_SET_RESULT_END 配置是否退出上行模式并清空计算结果。

0：无效

1：退出上行模式并清空计算结果

(WO)

Register 19.7. HMAC_SET_INVALIDATE_JTAG_REG (0x0060)



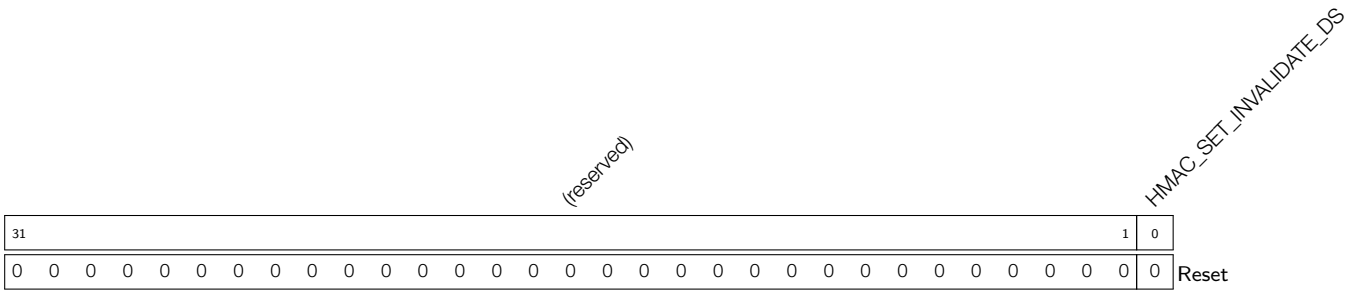
HMAC_SET_INVALIDATE_JTAG 配置是否清空下行模式下 JTAG 重启功能的计算结果。

0：无效

1：清空计算结果

(WO)

Register 19.8. HMAC_SET_INVALIDATE_DS_REG (0x0064)



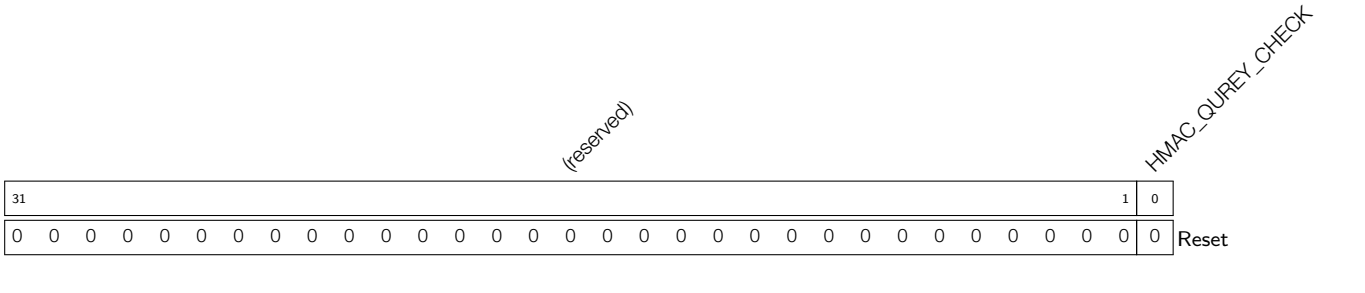
HMAC_SET_INVALIDATE_DS 配置是否清空下行模式下 DSA 功能的计算结果。

0：无效

1：清空计算结果

(WO)

Register 19.9. HMAC_QUERY_ERROR_REG (0x0068)



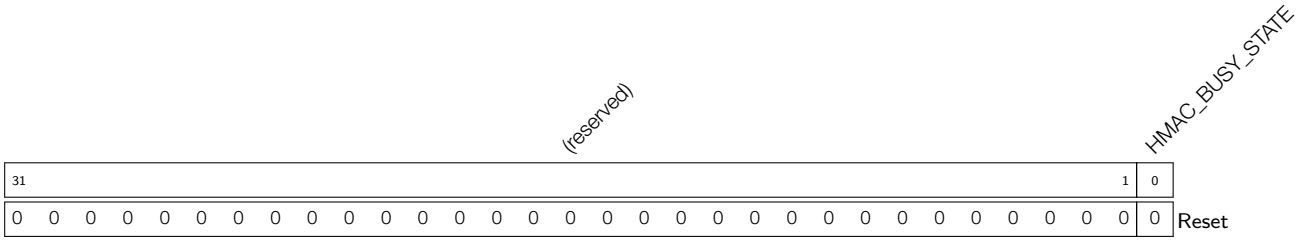
HMAC_QUREY_CHECK 表示 HMAC 密钥与功能是否匹配。

0: HMAC 密钥和功能匹配。

1: 不匹配

(RO)

Register 19.10. HMAC_QUERY_BUSY_REG (0x006C)



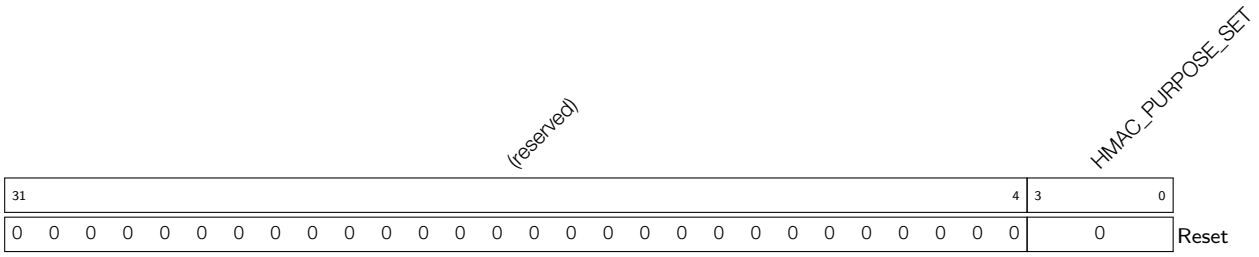
HMAC_BUSY_STATE 表示 HMAC 是否处于忙碌状态。执行计算任务之前，请确保 HMAC 已空闲。

0：空闲

1：HMAC 仍处于工作状态

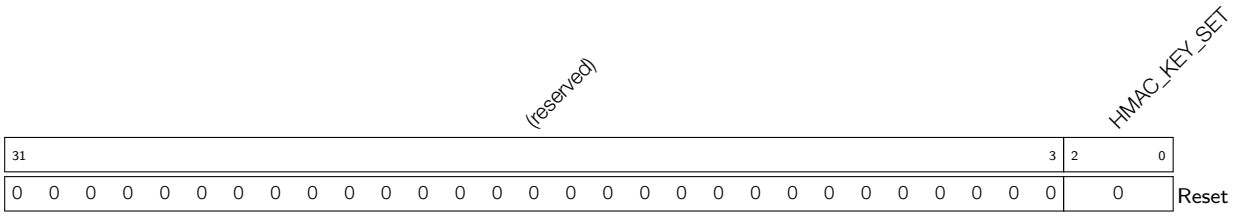
(RO)

Register 19.11. HMAC_SET_PARA_PURPOSE_REG (0x0044)



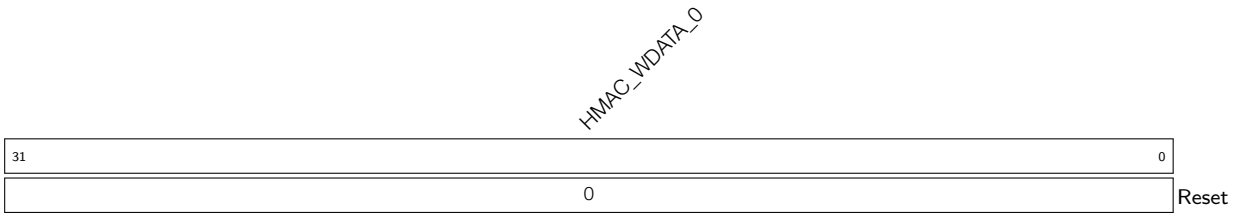
HMAC_PURPOSE_SET 设置 HMAC 功能，请参阅表 19-1。(WO)

Register 19.12. HMAC_SET_PARA_KEY_REG (0x0048)



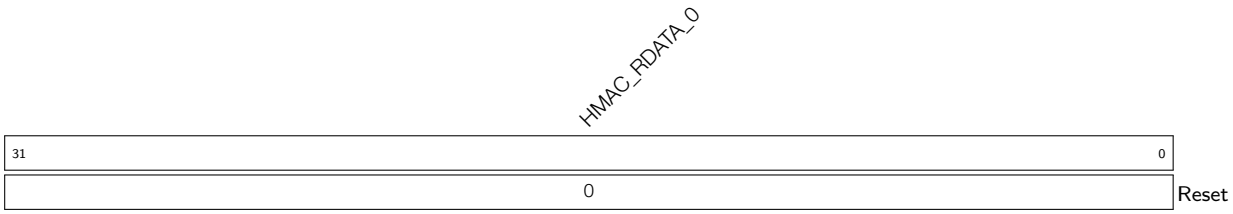
HMAC_KEY_SET 选择 HMAC 密钥。共有 6 个密钥，编号 0 至 5，将选择的密钥编号写入该字段即可。(WO)

Register 19.13. HMAC_WR_MESSAGE_*n*_REG (*n*: 0-15) (0x0080+4n*)**



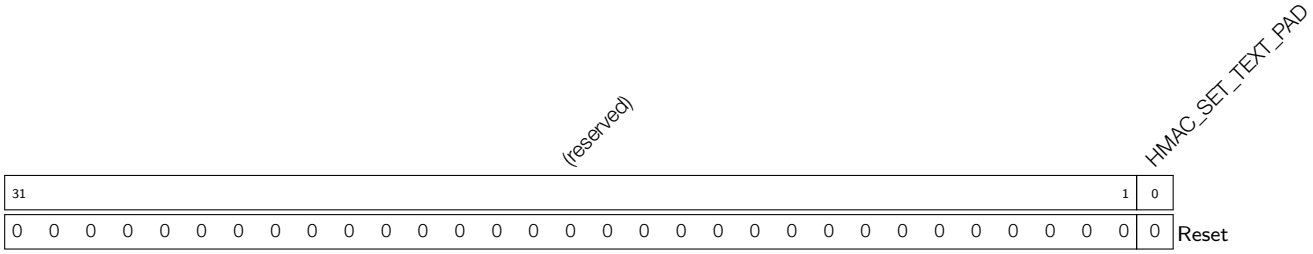
HMAC_WDATA_*n* 表示信息的第 *n* 个 32 位数据信息。(WO)

Register 19.14. HMAC_RD_RESULT_*n*_REG (*n*: 0-7) (0x00C0+4n*)**



HMAC_RDATA_*n* 读取 hash 结果的第 *n* 个 32 位。(RO)

Register 19.15. HMAC_SET_MESSAGE_PAD_REG (0x00F0)



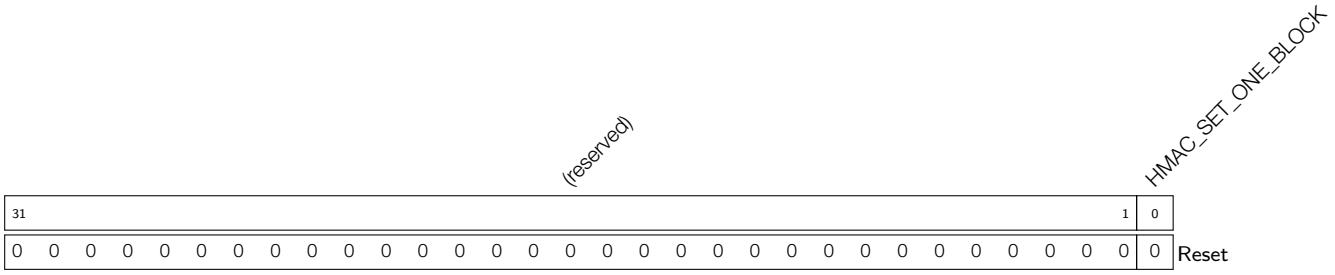
HMAC_SET_TEXT_PAD 表示是否由软件执行填充操作。

0：不由软件执行

1：由软件执行

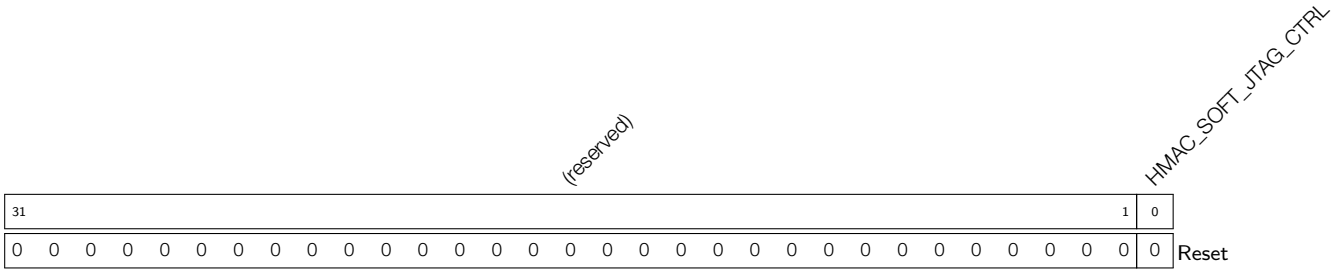
(WO)

Register 19.16. HMAC_ONE_BLOCK_REG (0x00F4)



HMAC_SET_ONE_BLOCK 置 1 表示 Block_n 是唯一一个数据块，同时 Block_1 已经包含了所有的填充位，无需填充。(WO)

Register 19.17. HMAC_SOFT_JTAG_CTRL_REG (0x00F8)



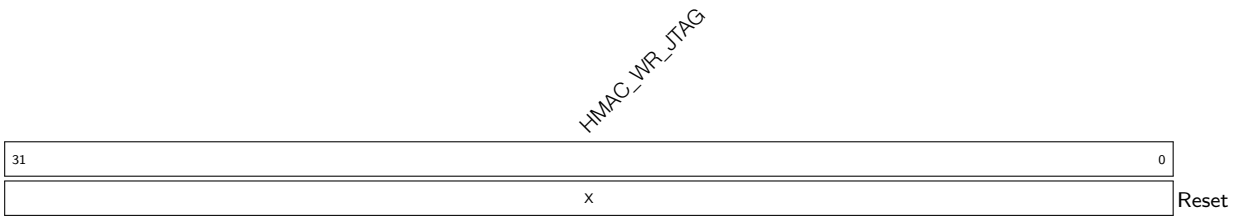
HMAC_SOFT_JTAG_CTRL 配置是否开启 JTAG 验证模式。

0：关闭

1：开启

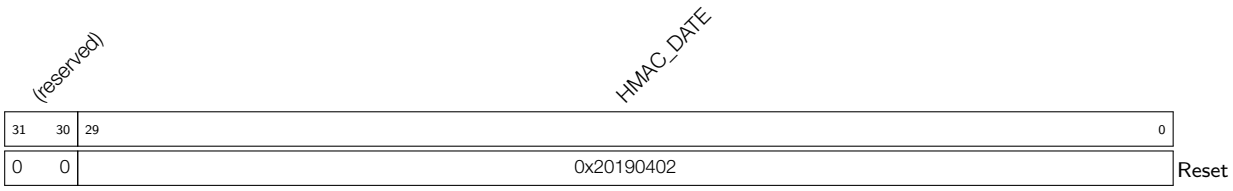
(WO)

Register 19.18. HMAC_WR_JTAG_REG (0x00FC)



HMAC_WR_JTAG 写入用于重启 JTAG 的输入比较数值。(WO)

Register 19.19. HMAC_DATE_REG (0x00F8)



HMAC_DATE 版本控制寄存器。(R/W)

20 RSA 加速器 (RSA)

20.1 概述

RSA 加速器可为多种运用于“RSA 非对称式加密演算法”的高精度计算提供硬件支持，能够极大地降低此类运算的运行时间和软件复杂度。与纯软件 RSA 算法相比，硬件 RSA 加速器的运算速度更快。RSA 加速器还支持多种“运算子长度”，具有很高的灵活性。

20.2 主要特性

RSA 加速器支持以下功能：

- 大数模幂运算（支持两个加速选项）
- 大数模乘运算
- 大数乘法运算
- 多种运算子长度
- 支持在运算完成后触发中断

20.3 功能描述

RSA 加速器的激活需使能 PCR_RSA_CONF_REG 外围时钟的 PCR_RSA_CLK_EN 位，并同时清零 PCR_RSA_RST_EN 位。此外，还需要清零 PCR_DS_RST_EN 和 PCR_ECDSA_RST_EN 复位 数字签名算法 (DSA)和椭圆曲线数字签名算法 (ECDSA)。

不过，RSA 加速器激活后还须等待 RSA 相关存储器初始化完成后才能开始工作。具体来说，RSA_QUERY_CLEAN_REG 读 0 时初始化开始，读 1 时初始化完成。因此，在复位后首次使用 RSA 加速器时，软件需要先查询 RSA_QUERY_CLEAN_REG 的值是否为 1，以确保 RSA 加速器可正常工作。

此外，RSA 加速器支持中断功能，可对寄存器 RSA_INT_ENA_REG 写 1 / 0 以开启 / 关闭中断。RSA 加速器的中断功能默认开启。

注意：

ESP32-H2 的数字签名算法 (DSA)模块在工作期间会调用 RSA 加速器。此时，用户无法正常访问 RSA 加速器。

20.3.1 大数模幂运算

大数模幂运算的算法是 $Z = X^Y \bmod M$，它是基于 Montgomery Multiplication（蒙哥马利乘法）实现的。因此，对于大数模幂运算，除了需要运算子 X、Y、M 外，还需要额外两个运算子，即参数 $\overline{r}$ 和 M'。这两个参数需要通过软件提前运算得到。

RSA 加速器支持运算子长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \ldots, 96\}$) 的大数模幂运算。$Z$、$X$、$Y$、$M$ 和 $\overline{r}$ 的位宽为这 96 种中的任意一种，要求它们的位宽必须相同，而 M' 的位宽始终是 32。

设进制数

$$b = 2^{32}$$

则运算子可以由若干个 b 进制数来表示：

$$n = \frac{N}{32}$$
$$Z = (Z_{n-1}Z_{n-2}\cdots Z_0)_b$$
$$X = (X_{n-1}X_{n-2}\cdots X_0)_b$$
$$Y = (Y_{n-1}Y_{n-2}\cdots Y_0)_b$$
$$M = (M_{n-1}M_{n-2}\cdots M_0)_b$$
$$\overline{r} = (\overline{r}_{n-1}\overline{r}_{n-2}\cdots \overline{r}_0)_b$$

其中 $Z_{n-1}\cdots Z_0$、$X_{n-1}\cdots X_0$、$Y_{n-1}\cdots Y_0$、$M_{n-1}\cdots M_0$、$\overline{r}_{n-1}\cdots \overline{r}_0$ 分别表示一个 b 进制数，位宽皆为 32。且 Z_{n-1}、X_{n-1}、Y_{n-1}、M_{n-1}、$\overline{r}_{n-1}$ 分别为 Z、X、Y、M、$\overline{r}$ 最高位的 b 进制数，而 Z_0、X_0、Y_0、M_0、$\overline{r}_0$ 分别为 Z、X、Y、M、$\overline{r}$ 最低位的 b 进制数。

另设 $R = b^n$，则计算得参数 $\overline{r} = R^2 \bmod M$。

另外 M' 可使用下方公式计算：

$$M' = -M^{-1} \bmod b$$

其中，M^{-1} 为 M 对 R 的模逆元，可使用扩展二进制 GCD 算法计算。大数模幂运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 / 0 以开启 / 关闭中断。
2. 配置相关寄存器。
   (a) 对寄存器 RSA_MODE_REG 写入 $(\frac{N}{32} - 1)$。
   (b) 对寄存器 RSA_M_PRIME_REG 写入 M'。
   (c) 根据需要配置加速选项相关寄存器。请参照章节 20.3.4 获取详细信息。
3. 将 X_i、Y_i、M_i、$\overline{r}_i (i \in \{0, 1, \ldots, n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Y_MEM、RSA_M_MEM、RSA_Z_MEM。每块存储器的容量都是 96 字 (word)。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

   只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MODEXP 写入 1 启动计算。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 RSA_Z_MEM 读出运算结果 $Z_i (i \in \{0, 1, \ldots, n-1\})$。
7. 若中断功能已开启，对 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_Y_MEM 中的 Y_i、存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 都不会变化。但是，存储器 RSA_X_MEM 中的 X_i 与存储器 RSA_Z_MEM 中的 $\overline{r}_i$ 都已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

20.3.2 大数模乘运算

大数模乘运算 $Z = X \times Y \bmod M$ 也是基于 Montgomery Multiplication 实现的。因此，与大数模幂运算类似，也需要预先通过软件计算额外的两个运算子 $\overline{r}$ 和 M'。

RSA 加速器也支持 96 种运算子长度的大数模乘运算。

大数模乘运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 / 0 以开启 / 关闭中断。
2. 配置相关寄存器。
   (a) 对寄存器 RSA_MODE_REG 写入 $(\frac{N}{32}-1)$。
   (b) 对寄存器 RSA_M_PRIME_REG 写入 M'。
3. 将 X_i、Y_i、M_i、$\overline{r}_i$ $(i \in \{0,1,\ldots,n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Y_MEM、RSA_M_MEM、RSA_Z_MEM。每块存储器的容量都是 96 字 (word)。

   每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。

   只需要根据运算子长度，将各个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MODMULT 写入 1。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in \{0,1,\ldots,n-1\})$。
7. 若中断功能已开启，对寄存器 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_X_MEM 中的 X_i、存储器 RSA_Y_MEM 中的 Y_i、存储器 RSA_M_MEM 中的 M_i、寄存器 RSA_M_PRIME_REG 中的 M' 都不会变化。但是，存储器 RSA_Z_MEM 中的 $\overline{r}_i$ 已经被覆盖。所以当需要连续运算时，只需要更新被覆盖的存储器即可。

20.3.3 大数乘法运算

大数乘法运算实现了 $Z = X \times Y$。其中 Z 的长度是运算子 X、Y 长度的两倍，所以 RSA 加速器只支持运算子 X、Y 长度为 $N = 32 \times x$ $(x \in \{1,2,3,\ldots,48\})$ 的大数乘法运算。运算子 Z 的长度 $\hat{N}$ 为 $2 \times N$。

大数乘法运算的软件流程为：

1. 对寄存器 RSA_INT_ENA_REG 写 1 / 0 以开启 / 关闭中断。
2. 配置相关寄存器。对寄存器 RSA_MODE_REG 写入 $(\frac{\hat{N}}{32}-1)$，即 $(\frac{N}{16}-1)$。
3. 将 X_i、Y_i $(i \in \{0,1,\ldots,n-1\})$ 分别写入存储器 RSA_X_MEM、RSA_Z_MEM。每块存储器的每一个字刚好存放一个 b 进制数。这些存储器使用小端序存储数据，低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。n 为 $\frac{N}{32}$ 。

   X_i $(i \in \{0,1,\ldots,n-1\})$ 要填充到存储器 RSA_X_MEM 中的第 i 个字对应的地址中，但需要注意的是，Y_i $(i \in \{0,1,\ldots,n-1\})$ 并不是要填充到存储器 RSA_Z_MEM 中的第 i 个字对应的地址中，而是需要填充到存储器 RSA_Z_MEM 中的第 $n+i$ 个字对应的地址中，即存储器 RSA_Z_MEM 的基地址加上偏移量 $4 \times (n+i)$。

   只需要根据运算子长度，将这两个运算子中有效的数据写入存储器，没有使用到的存储器可以是任意值。
4. 对 RSA_SET_START_MULT 写入 1。
5. 等待运算结束。轮询 RSA_QUERY_IDLE 直到读到 1，或者等待 RSA 中断产生。
6. 从存储器 RSA_Z_MEM 读出运算结果 Z_i $(i \in \{0,1,\ldots,\hat{n}-1\})$。$\hat{n}$ 为 $2 \times n$ 。
7. 若中断功能已开启，对 RSA_CLEAR_INTERRUPT 写入 1 以清除中断。

运算结束后，寄存器 RSA_MODE_REG 中存储的运算子长度信息以及存储器 RSA_X_MEM 中的 X_i 都不会变化。但是，存储器 RSA_Z_MEM 中的 Y_i 已经被覆盖。所以当需要连续运算时，只需要更新必需的寄存器与存储器即可。

20.3.4 控制加速

对于大数模幂运算，ESP32-H2 的 RSA 加速器还特别提供 SEARCH 和 CONSTANT_TIME 两个选项，可进一步提高运算速度。默认情况下，这两个选项均处于不额外加速状态，可以单独使用，也可以同时使用。值得注意的是，即使这两个选项均处于不额外加速状态，硬件 RSA 加速器也比纯软件实施的 RSA 算法快很多。

具体来说，当这两个选项均处于不额外加速状态时，求解 $Z = X^Y \bmod M$ 的时间开销完全由运算子长度决定。否则，只要有某个选项携带有额外加速效果，那么运算的时间开销还与 Y 的 0/1 分布有关。

为了更清楚地说明问题，首先假设 Y 的二进制表示为：

$$Y = (\widetilde{Y}_{N-1}\widetilde{Y}_{N-2}\cdots\widetilde{Y}_{t+1}\widetilde{Y}_t\widetilde{Y}_{t-1}\cdots\widetilde{Y}_0)_2$$

其中，

- N 代表 Y 的长度，
- $\widetilde{Y}_t$ 的值为 1，
- $\widetilde{Y}_{N-1}, \widetilde{Y}_{N-2}, \ldots, \widetilde{Y}_{t+1}$ 的值均为 0，
- 且 $\widetilde{Y}_{t-1}, \widetilde{Y}_{t-2}, \ldots, \widetilde{Y}_0$ 中包括 m 个 0，其余 t-m 全部为 1，即 $\widetilde{Y}_{t-1}\widetilde{Y}_{t-2}, \cdots, \widetilde{Y}_0$ 的汉明重量 (Hamming weight) 为 $t - m$。

此时，当启动任一额外加速选项时：

- SEARCH 选项（RSA_SEARCH_ENABLE 置 1 开启额外加速）
  - RSA 加速器将忽略所有 $\widetilde{Y}_i$（$i > \alpha$）位。其中，加速位置 α 可通过 RSA_SEARCH_POS_REG 寄存器配置。α 在配置时需小于 N-1，否则相当于没有额外加速；且不建议小于 t，否则无法正确求解 $Z = X^Y \bmod M$。当设置 α 为 t 时，加速效果最佳。此时，$\widetilde{Y}_{N-1}, \widetilde{Y}_{N-2}, \ldots, \widetilde{Y}_{t+1}$ 中的 0 位将在运算中全部被忽略。
  - 注意，这个选项会忽略一些位，进而导致密钥长度变短，使安全性降低。因此，不应使用在安全要求高的应用中。
- CONSTANT_TIME 选项（RSA_CONSTANT_TIME_REG 置 0 开启额外加速）
  - RSA 加速器在运算过程中将简化对 Y 中 0 位的处理。因此不难想象，Y 中的 0 越多，加速效果越明显。
  - 注意，这个选项的加速效果与密钥中的 0/1 分布密切相关，这可能会被侧信道攻击 (SCA) 等攻击加以利用，使安全性降低。因此，不应使用在安全要求高的应用中。

为了直观地展示这两个选项带来的额外加速效果，下面通过一个典型实例加以说明。在 $Z = X^Y \bmod M$ 中，N 等于 3072，Y 等于 65537。表 20-1 展示了 4 种选项组合对应的时间开销。注意，这里 SEARCH 选项开启时设定 α 为 16。

表 20-1. 加速效果

SEARCH 选项	CONSTANT_TIME 选项	时间开销 (ms)
不加速	不加速	451.69
加速	不加速	2.71
不加速	加速	1.45
加速	加速	1.40

可以看到：

- 当两个选项均处于不额外加速状态时，时间开销最大。
- 当两个选项均处于额外加速状态时，时间开销最小。
- 相比于不额外加速状态，任一选项处于额外加速状态时的时间开销明显大幅度降低。

20.4 存储器列表

请注意，这里的地址都是相对于 RSA 加速器基地址的地址偏移量（相对地址），详见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	大小（字节）	起始地址	结束地址	访问
RSA_M_MEM	存储器 M	384	0x0000	0x017F	R/W
RSA_Z_MEM	存储器 Z	384	0x0200	0x037F	R/W
RSA_Y_MEM	存储器 Y	384	0x0400	0x057F	R/W
RSA_X_MEM	存储器 X	384	0x0600	0x077F	R/W

20.5 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

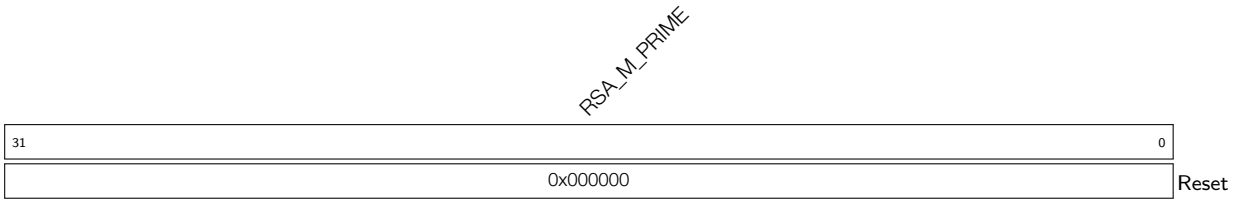
名称	描述	地址	访问
控制 / 配置寄存器			
RSA_M_PRIME_REG	代表 M’	0x0800	R/W
RSA_MODE_REG	配置 RSA 长度模式	0x0804	R/W
RSA_SET_START_MODEXP_REG	启动模幂运算	0x080C	WT
RSA_SET_START_MODMULT_REG	启动模乘运算	0x0810	WT
RSA_SET_START_MULT_REG	启动乘法运算	0x0814	WT
RSA_QUERY_IDLE_REG	代表 RSA 状态	0x0818	RO
RSA_CONSTANT_TIME_REG	配置 constant_time 选项	0x0820	R/W
RSA_SEARCH_ENABLE_REG	配置 search 选项	0x0824	R/W
RSA_SEARCH_POS_REG	配置 search 开始位置	0x0828	R/W
状态寄存器			
RSA_QUERY_CLEAN_REG	表示 RSA 内存初始化状态	0x0808	RO
中断寄存器			
RSA_INT_CLR_REG	清除 RSA 中断	0x081C	WT
RSA_INT_ENA_REG	使能 RSA 中断	0x082C	R/W
版本控制寄存器			
RSA_DATE_REG	版本控制寄存器	0x0830	R/W

20.6 寄存器

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

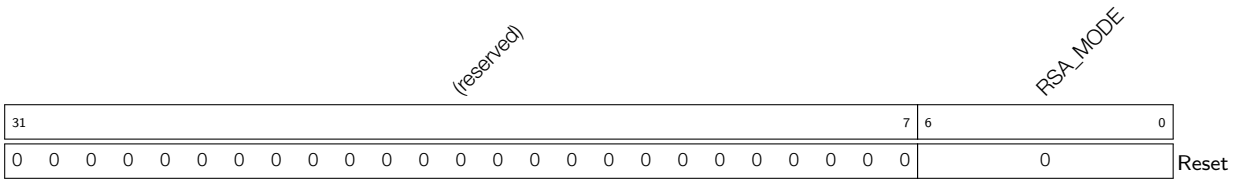
关于保留 (reserved) 域的处理，请查看章节 如何配置寄存器的保留域。

Register 20.1. RSA_M_PRIME_REG (0x0800)



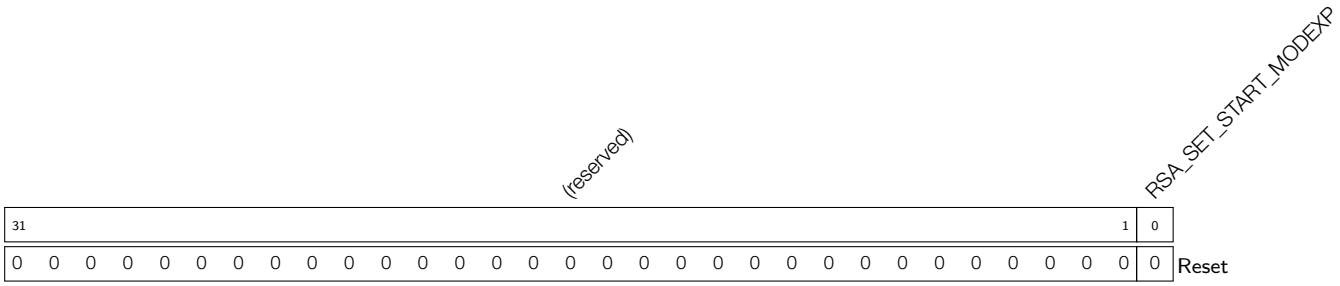
RSA_M_PRIME 代表 M'。(R/W)

Register 20.2. RSA_MODE_REG (0x0804)



RSA_MODE 配置 RSA 长度。(R/W)

Register 20.3. RSA_SET_START_MODEXP_REG (0x080C)



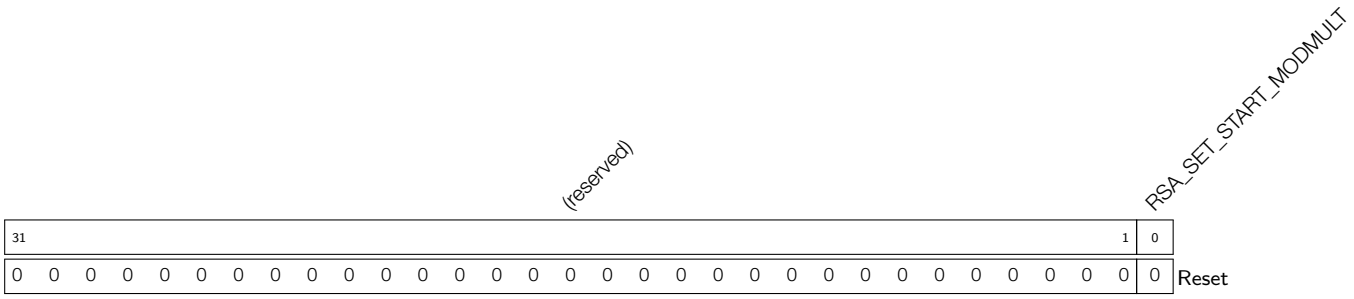
RSA_SET_START_MODEXP 配置是否启动模幂运算。

0：无效果

1：启动

(WT)

Register 20.4. RSA_SET_START_MODMULT_REG (0x0810)



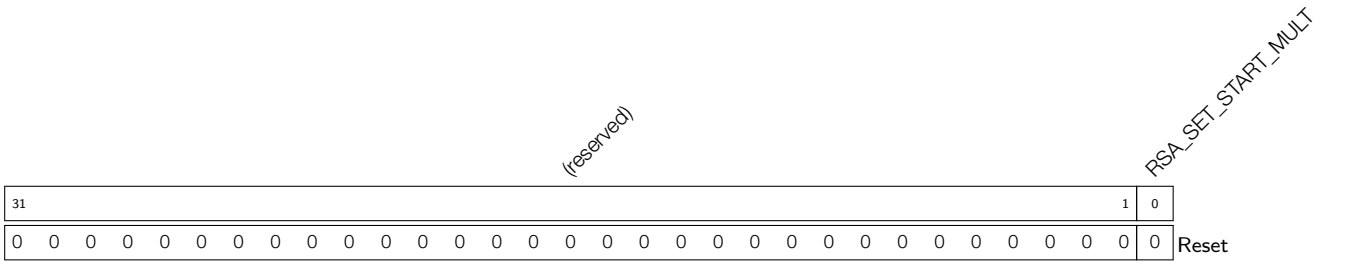
RSA_SET_START_MODMULT 配置是否启动模乘运算。

0：无效果

1：启动

(WT)

Register 20.5. RSA_SET_START_MULT_REG (0x0814)



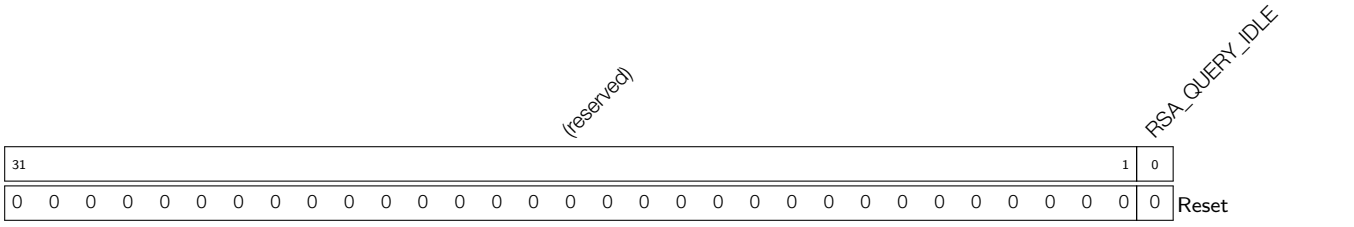
RSA_SET_START_MULT 配置是否启动乘法运算。

0：无效果

1：启动

(WT)

Register 20.6. RSA_QUERY_IDLE_REG (0x0818)



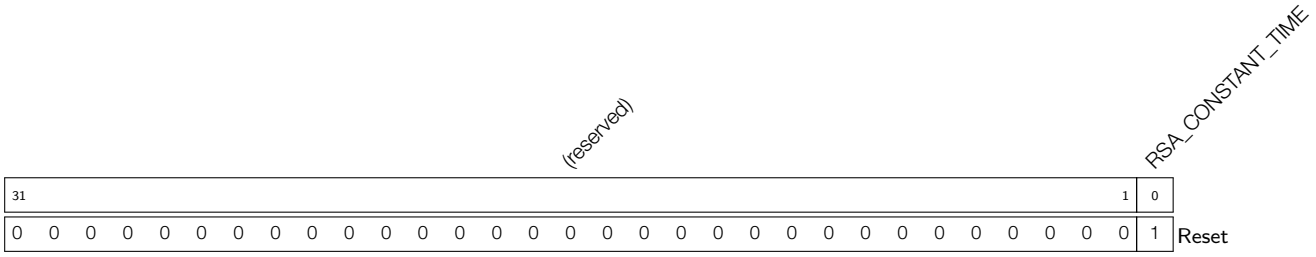
RSA_QUERY_IDLE 代表 RSA 状态。

0：忙碌

1：空闲

(RO)

Register 20.7. RSA_CONSTANT_TIME_REG (0x0820)



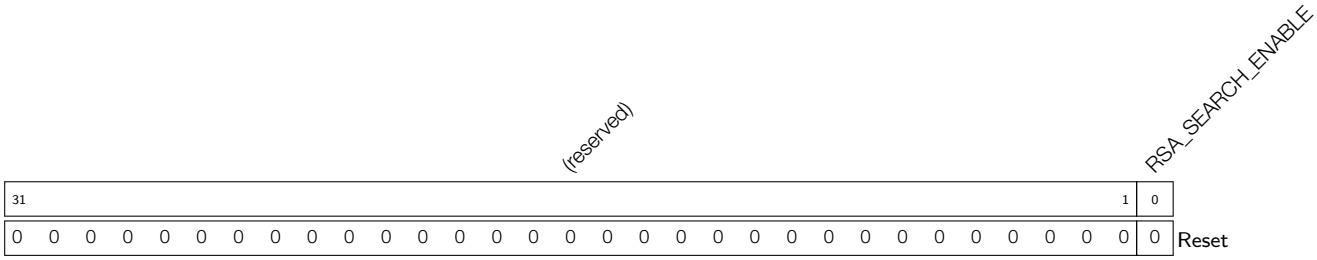
RSA_CONSTANT_TIME 配置 constant_time 选项。

0：加速

1：不加速（默认）

(R/W)

Register 20.8. RSA_SEARCH_ENABLE_REG (0x0824)



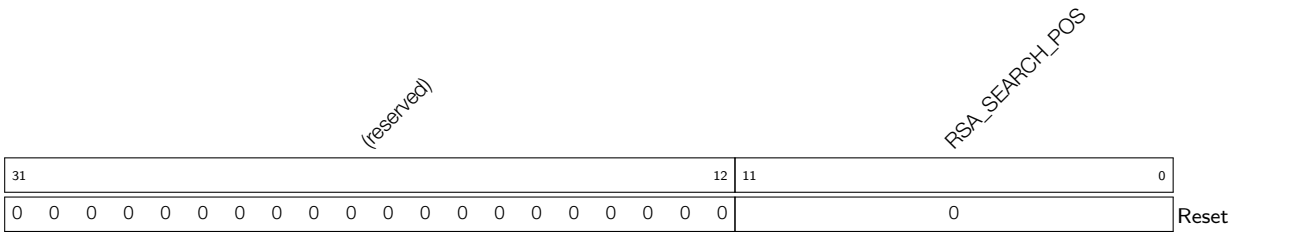
RSA_SEARCH_ENABLE 配置 search 选项。

0：不加速（默认）

1：加速

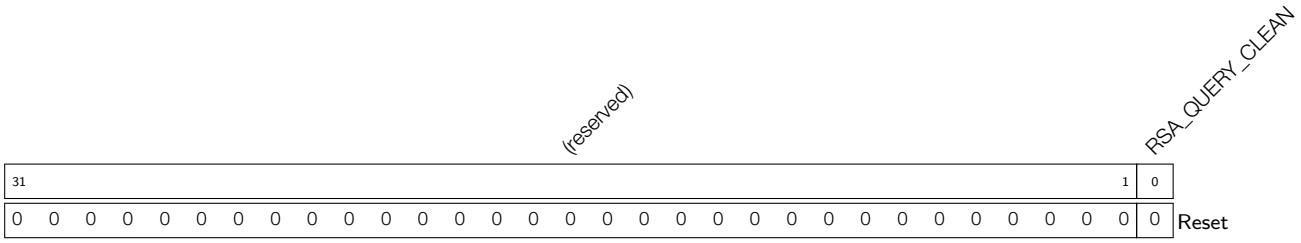
此选项需配合 RSA_SEARCH_POS_REG 使用。(R/W)

Register 20.9. RSA_SEARCH_POS_REG (0x0828)



RSA_SEARCH_POS 配置 search 的开始地址。此选项需配合 RSA_SEARCH_ENABLE_REG 使用，仅在 RSA_SEARCH_ENABLE 为 1 时有效。(R/W)

Register 20.10. RSA_QUERY_CLEAN_REG (0x0808)



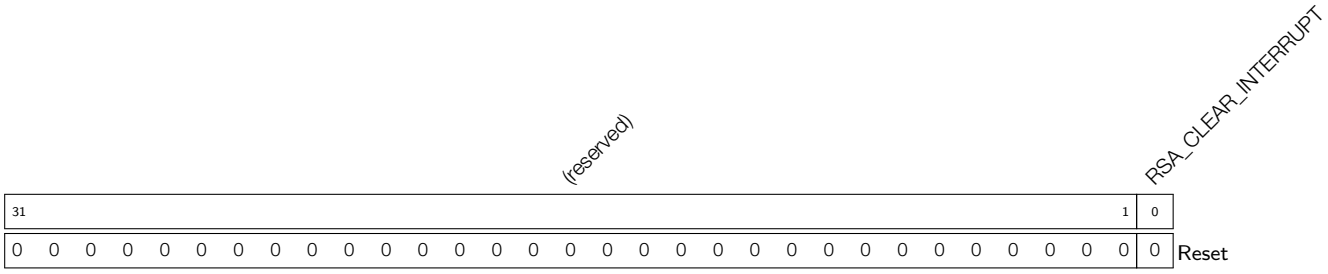
RSA_QUERY_CLEAN 代表 RSA 内存初始化的状态。

0：没有完成

1：已经完成

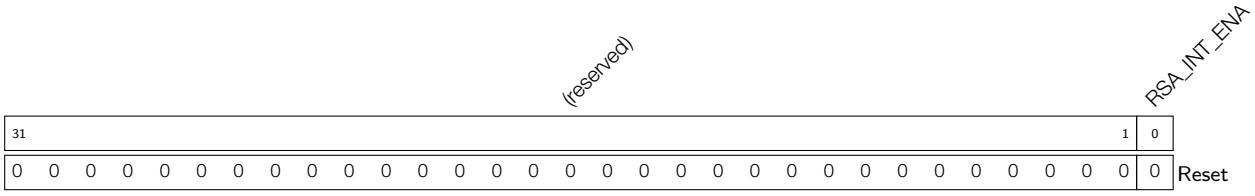
(RO)

Register 20.11. RSA_INT_CLR_REG (0x081C)



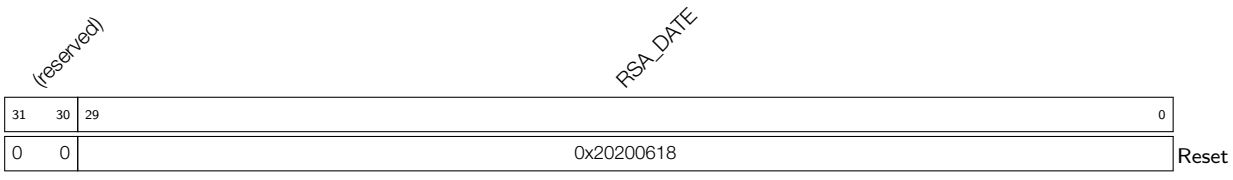
RSA_CLEAR_INTERRUPT 写 1 清除 RSA 中断。(WT)

Register 20.12. RSA_INT_ENA_REG (0x082C)



RSA_INT_ENA 写 1 使能 RSA 中断。(R/W)

Register 20.13. RSA_DATE_REG (0x0830)



RSA_DATE 版本控制寄存器。(R/W)

21 SHA 加速器 (SHA)

21.1 概述

ESP32-H2 内置 SHA（安全哈希算法）硬件加速器可完成 SHA 运算，具有 Typical SHA 和 DMA-SHA 两种工作模式。整体而言，相比基于纯软件的 SHA 运算，SHA 硬件加速器能够极大地提高运算速度。

21.2 主要特性

ESP32-H2 的 SHA 硬件加速器：

- 支持 FIPS PUB 180-4 规范中的以下运算标准
  - SHA-1 运算
  - SHA-224 运算
  - SHA-256 运算
- 提供两种工作模式
  - Typical SHA 工作模式
  - DMA-SHA 工作模式
- 允许插入 (interleaved) 功能（仅限 Typical SHA 工作模式）
- 允许中断功能（仅限 DMA-SHA 工作模式）

21.3 工作模式简介

ESP32-H2 内置的 SHA 加速器支持两种工作模式。

- Typical SHA 工作模式：所有数据读写统一通过 CPU 访问完成。
- DMA-SHA 工作模式：所有读数据通过硬件上的 DMA 完成。具体来说，用户可配置 DMA 控制器，由 DMA 控制器提供 SHA 运算过程中所需的数据信息。因此，可以释放 CPU 执行其他任务。

SHA 加速器的激活仅需使能 PCR_SHA_CONF_REG 外围时钟的 PCR_SHA_CLK_EN 位，并同时清零 PCR_SHA_RST_EN 位。此外，还需要清零 PCR_DS_RST_EN, PCR_HMAC_RST_EN 和 PCR_ECDSA_RST_EN 复位 数字签名算法 (DSA)，HMAC 加速器 (HMAC) 和椭圆曲线数字签名算法 (ECDSA)。

用户可通过配置 SHA_START_REG 或 SHA_DMA_START_REG 选择 SHA 加速器的工作模式，先配置的工作模式生效，具体请见表 21-1。

表 21-1. 工作模式选择

工作模式	选择方式
Typical SHA	SHA_START_REG 置 1
DMA-SHA	SHA_DMA_START_REG 置 1

用户可通过配置 SHA_MODE_REG 寄存器选择 SHA 加速器的运算标准，具体请见表 21-2。

表 21-2. 运算标准选择

哈希运算标准	SHA_MODE_REG 的配置
SHA-1	0
SHA-224	1
SHA-256	2

注意：

ESP32-H2 的 数字签名算法 (DSA) 和 HMAC 模块在工作时也会调用 SHA 加速器。此时，用户无法正常访问 SHA 加速器。

21.4 功能描述

SHA 加速器可以提取信息摘要 (message digest)，其主要工作流程分为两步：信息预处理和哈希运算。

21.4.1 信息预处理

信息预处理分为三个主要步骤：附加填充比特、信息解析和设置初始哈希值。

21.4.1.1 附加填充比特

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件操作将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 m 位，则填充步骤如下：

1. 首先，在待处理信息后填充 1 个 “1”；
2. 随后，再填充 k 个 “0”。其中，k 为满足 $m + 1 + k \equiv 448 \ mod \ 512$ 的最小非负数解；
3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 m 的值。

更多详情，请参考 《FIPS PUB 180-4 规范》 > 章节 “Padding the Message”。

21.4.1.2 信息解析

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$、$M^{(2)}$、…、$M^{(N)}$。一个 512 位信息块包括 16 个 32 位的字 (word)，则第 i 个信息块的第一个 32 位字表示为 $M_0^{(i)}$，第二个 32 位字表示为 $M_1^{(i)}$，…，第 16 个 32 位字表示为 $M_{15}^{(i)}$。

SHA 加速器在工作时，每次处理的信息块数据均将按照如下规则写入相应的寄存器中：将 $M_0^{(i)}$ 存放在 SHA_M_0_REG 中，$M_1^{(i)}$ 存放在 SHA_M_1_REG，…，$M_{15}^{(i)}$ 存放在 SHA_M_15_REG 中。

说明：

有关 “信息块” 及相关概念的描述，请参考 《FIPS PUB 180-4 规范》 > 章节 “Glossary of Terms and Acronyms”。

21.4.1.3 哈希初始值 (Initial Hash Value)

在进行哈希运算前，首先必须设置哈希初始值 $H^{(0)}$，其中 SHA-1、SHA-224 和 SHA-256 运算的哈希初始值为常量 C，且已经固定在硬件中，无需额外配置。

21.4.2 哈希运算流程

在完成信息预处理后，ESP32-H2 SHA 加速器将正式开始哈希运算，最终根据不同运算标准得到不同长度的信息摘要。正如上文所述，ESP32-H2 SHA 加速器支持 Typical SHA 和 DMA-SHA 两种工作模式，下面将对这两种工作模式的具体流程进行介绍。

21.4.2.1 Typical SHA 模式下的运算流程

通常情况下，ESP32-H2 的 SHA 会先处理完当前信息的所有信息块并生成该信息的信息摘要，之后再开始计算新的信息摘要。

不过，ESP32-H2 SHA 加速器还支持 “interleaved” 运算（仅 Typical SHA 支持），即在完成当前信息的所有运算前，用户在每个信息块计算完成后均可插入新的运算。

具体来说，用户可以将存储在 SHA_H_*n*_REG 寄存器中的信息摘要暂时保存到其他地方，然后让 SHA 加速器来完成其他优先级更高的运算任务。当插入的运算结束后，用户再将之前暂存的信息摘要重新写入 SHA_H_*n*_REG 中，并继续完成之前中断的计算。

Typical SHA 的具体运算流程

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器，设置运算标准。具体配置，请参考表 21-2。
2. 处理当前信息块。
   - 将当前信息块写入 SHA_M_*n*_REG 寄存器。
3. 启动 SHA 加速器 [1]。
   - 如果为首次运算，则对 SHA_START_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器按照步骤 1 中选定的运算标准，使用硬件中固定的哈希初始值进行运算；
   - 如果非首次运算[2]，则对 SHA_CONTINUE_REG 寄存器置 1，启动 SHA 加速器的运算。此时，SHA 加速器使用 SHA_H_*n*_REG 寄存器中的值作为哈希初始值进行运算。
4. 查询当前信息块的处理进度。
   - 轮询寄存器 SHA_BUSY_REG 一直到读回的值为 0，代表 SHA 硬件加速器已完成对当前信息块的计算，进入 “空闲” 状态[3]。
5. 选择是否有后续的待处理信息块。
   - 如果存在后续待处理信息块，则跳回执行步骤 2。
   - 否则，继续执行。
6. 获取信息摘要：
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

说明：

1. 这里，在 SHA 加速器进行硬件运算时，如果存在后续待处理信息块，软件还可以同时将后续信息块写入 SHA_M_*n*_REG 寄存器，以节省时间。
2. 比如重新启动 SHA 加速器完成之前暂停任务的情况。
3. 这里，你可以选择是否需要插入其他任务。如需插入，请前往 插入任务工作流程 具体查看。

如上文所述，ESP32-H2 SHA 加速器**支持在 Typical SHA 模式下“插入”任务**。

具体工作流程如下。

1. 保存插入前任务的以下数据，准备将 SHA 加速器的使用权移交给插入的任务。
   - 读取并保存寄存器 SHA_MODE_REG 中的运算标准类型。
   - 读取并保存寄存器堆 SHA_H_*n*_REG 中的信息摘要。
2. 执行插入的任务。具体按照插入运算类型的不同，请见 Typical SHA 或 DMA-SHA 工作流程。
3. 恢复插入前任务的以下数据，准备将 SHA 加速器的使用权交还给插入前的任务。
   - 将获得使用权前保存的运算标准类型重新写入寄存器 SHA_MODE_REG;
   - 将获得使用权前保存的信息摘要写入寄存器堆 SHA_H_*n*_REG。
4. 将之前任务的下一个待处理信息块写入 SHA_M_*n*_REG 寄存器，并对 SHA_CONTINUE_REG 寄存器置 1，重新启动 SHA 加速器，完成之前暂停的任务。

21.4.2.2 DMA-SHA 模式下的运算流程

ESP32-H2 SHA 加速器在 DMA-SHA 工作模式下不支持 “interleaved” 运算，即用户必须在每次 DMA 运算（可能包括 1 个或多个信息块）全部结束后才能插入新的运算。这种情况下，用户如有插入运算需求，可将较大信息块进行拆分，并进行多次 DMA 运算。每次 DMA 运算之间允许插入其他运算标准的计算任务。

单次 DMA 运算最多可以处理 63 个数据块。

与 Typical SHA 不同，SHA 在 DMA-SHA 工作模式下，运算过程中的数据搬运过程均由硬件完成。具体配置可见章节 3 *通用 DMA 控制器 (GDMA)*。

DMA-SHA 的具体工作流程

1. 选择运算标准。
   - 配置 SHA_MODE_REG 寄存器，设置运算标准。具体配置，请参考表 21-2。
2. 选择是否启用中断。请将 SHA_INT_ENA_REG 寄存器配置为 1 以启动中断。
3. 配置块个数。
   - 将待加密数据的总块数 *M* 写入 SHA_DMA_BLOCK_NUM_REG 寄存器。
4. 开始 DMA-SHA 运算。
   - 如果当前 DMA-SHA 运算为接着另一次 DMA-SHA 的运算，需要提前将另一次计算得到的信息摘要写入寄存器堆 SHA_H_*n*_REG 中，随后将 1 写入寄存器 SHA_DMA_CONTINUE_REG;
   - 否则，只需要将 1 写入寄存器 SHA_DMA_START_REG。
5. 等待 DMA-SHA 运算结束。判断 DMA-SHA 运算结束有以下两种方法：

- 轮询寄存器 SHA_BUSY_REG 结果为 0。
- 等待中断信号产生。此时，应及时通过软件将 SHA_INT_CLEAR_REG 寄存器置为 1 以清除中断。

6. 获取信息摘要
   - 从寄存器堆 SHA_H_*n*_REG 取出信息摘要。

21.4.3 信息摘要存储

哈希运算完成之后，计算得到的信息摘要被 SHA 加速器更新至对应的 SHA_H_*n*_REG (*n*: 0 ~ 7) 寄存器中。不同运算标准得到的信息摘要长度也不同，详情见表 21-3：

表 21-3. 不同运算标准信息摘要的寄存器占用情况

哈希运算标准	信息摘要长度（位）	寄存器占用情况[1]
SHA-1	160	SHA_H_0_REG ~ SHA_H_4_REG
SHA-224	224	SHA_H_0_REG ~ SHA_H_6_REG
SHA-256	256	SHA_H_0_REG ~ SHA_H_7_REG

[1] 信息摘要从最高有效位至最低有效位存放，第一个 word 存放在寄存器 SHA_H_0_REG 中，第二个 word 存放在寄存器 SHA_H_1_REG 中，以此类推。

21.4.4 中断

在 DMA-SHA 工作模式下，SHA 加速器允许中断发生。

- 用户可通过将 SHA_INT_ENA_REG 寄存器配置为 1 开启中断。如开启中断功能，SHA 加速器在完成运算时，中断发生。
- 注意，该中断必须由软件将 SHA_INT_CLEAR_REG 寄存器置为 1 进行清除。

在 Typical SHA 工作模式下，SHA 加速器将会快速完成运算，无需中断，故不支持中断功能。

21.5 寄存器列表

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

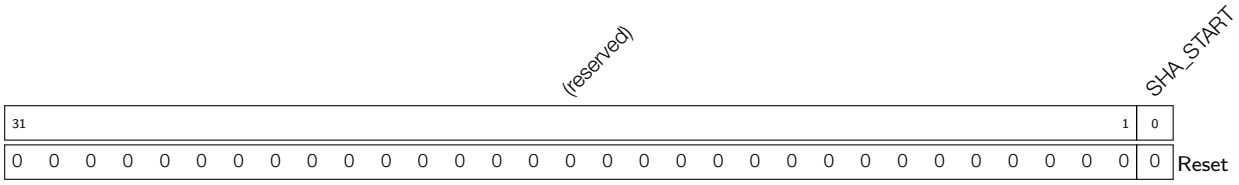
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	权限
控制与配置寄存器			
SHA_MODE_REG	配置 SHA 加速器的运算标准	0x0000	R/W
SHA_CONTINUE_REG	继续 SHA 运算（仅用于 Typical SHA 模式）	0x0014	WO
SHA_DMA_START_REG	启动 SHA 加速器的 DMA-SHA 模式	0x001C	WO
SHA_START_REG	启动 SHA 加速器的 Typical SHA 模式	0x0010	WO
SHA_DMA_CONTINUE_REG	继续 SHA 运算（仅用于 DMA-SHA 模式）	0x0020	WO
SHA_DMA_BLOCK_NUM_REG	信息块个数寄存器（仅用于 DMA-SHA 模式）	0x000C	R/W
状态寄存器			
SHA_BUSY_REG	表示 SHA 加速器是否处于“忙碌”状态	0x0018	RO
中断寄存器			
SHA_INT_CLEAR_REG	DMA-SHA 中断清除寄存器	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA 中断使能寄存器	0x0028	R/W
数据寄存器			
SHA_H_0_REG	哈希值	0x0040	R/W
SHA_H_1_REG	哈希值	0x0044	R/W
SHA_H_2_REG	哈希值	0x0048	R/W
SHA_H_3_REG	哈希值	0x004C	R/W
SHA_H_4_REG	哈希值	0x0050	R/W
SHA_H_5_REG	哈希值	0x0054	R/W
SHA_H_6_REG	哈希值	0x0058	R/W
SHA_H_7_REG	哈希值	0x005C	R/W
SHA_M_1_REG	输入信息	0x0084	R/W
SHA_M_2_REG	输入信息	0x0088	R/W
SHA_M_3_REG	输入信息	0x008C	R/W
SHA_M_4_REG	输入信息	0x0090	R/W
SHA_M_5_REG	输入信息	0x0094	R/W
SHA_M_6_REG	输入信息	0x0098	R/W
SHA_M_7_REG	输入信息	0x009C	R/W
SHA_M_8_REG	输入信息	0x00A0	R/W
SHA_M_9_REG	输入信息	0x00A4	R/W
SHA_M_10_REG	输入信息	0x00A8	R/W
SHA_M_11_REG	输入信息	0x00AC	R/W
SHA_M_12_REG	输入信息	0x00B0	R/W
SHA_M_13_REG	输入信息	0x00B4	R/W
SHA_M_14_REG	输入信息	0x00B8	R/W
SHA_M_15_REG	输入信息	0x00BC	R/W
版本寄存器			
SHA_DATE_REG	版本控制寄存器	0x00FC	R/W

21.6 寄存器

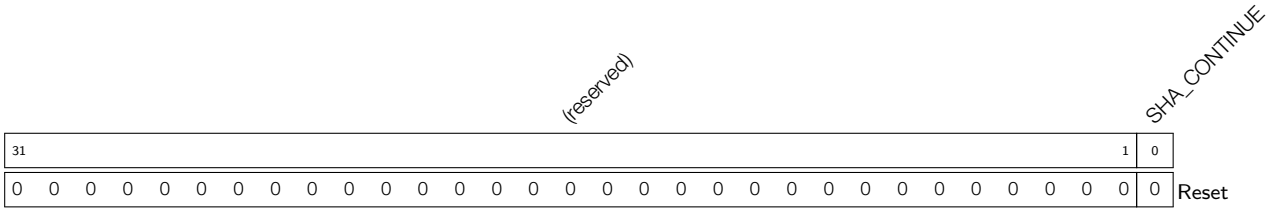
本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 21.1. SHA_START_REG (0x0010)



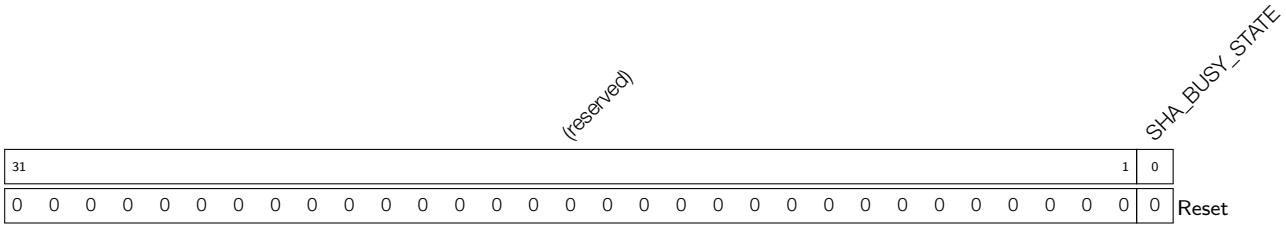
SHA_START 写 1 启动 SHA 加速器的 Typical SHA 模式。(WO)

Register 21.2. SHA_CONTINUE_REG (0x0014)



SHA_CONTINUE 写 1 继续 SHA 加速器的 Typical SHA 运算。(WO)

Register 21.3. SHA_BUSY_REG (0x0018)



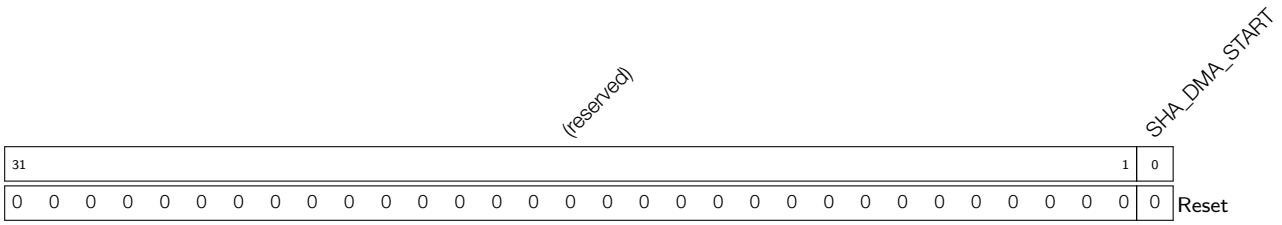
SHA_BUSY_STATE 表示 SHA 是否处于“忙碌”状态。

0：空闲

1：忙碌

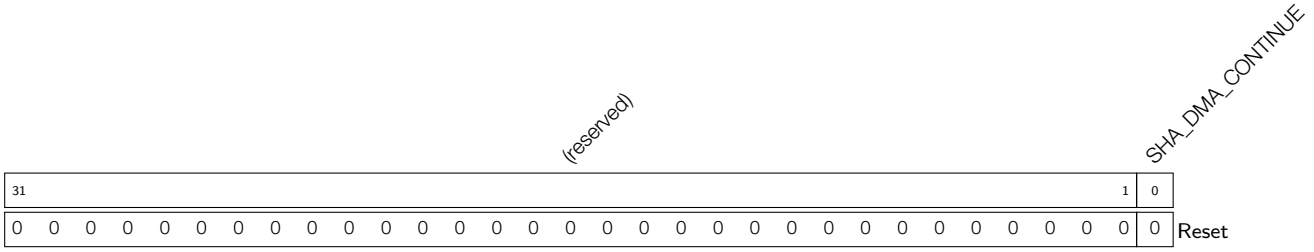
(RO)

Register 21.4. SHA_DMA_START_REG (0x001C)



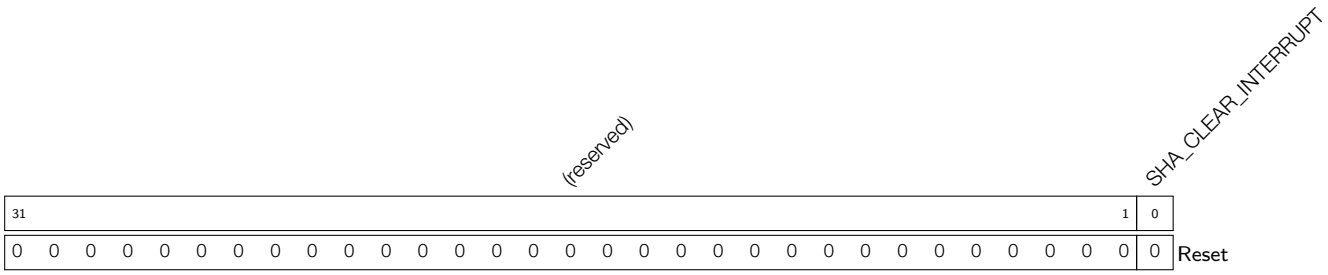
SHA_DMA_START 写 1 启动 SHA 加速器的 DMA-SHA 模式。(WO)

Register 21.5. SHA_DMA_CONTINUE_REG (0x0020)



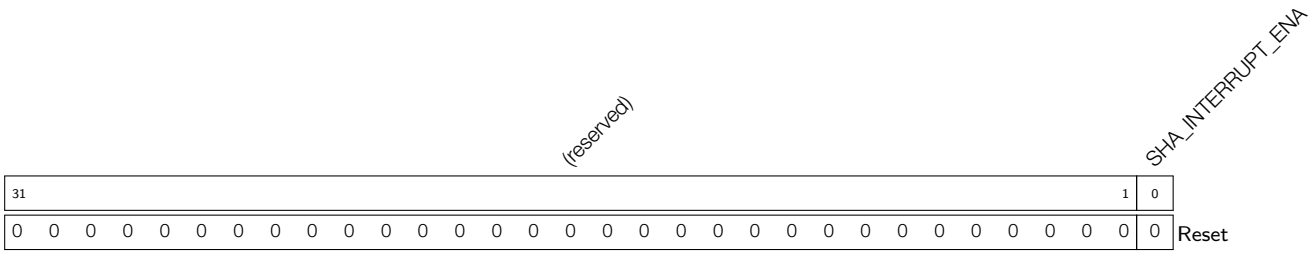
SHA_DMA_CONTINUE 写 1 继续 SHA 加速器的 DMA-SHA 运算。(WO)

Register 21.6. SHA_INT_CLEAR_REG (0x0024)



SHA_CLEAR_INTERRUPT 写 1 清除 DMA-SHA 中断。(WO)

Register 21.7. SHA_INT_ENA_REG (0x0028)



SHA_INTERRUPT_ENA 写 1 使能 DMA-SHA 中断。(R/W)

Register 21.8. SHA_DATE_REG (0x002C)

(reserved)	SHA_DATE	
31 30	29 … 0	
0 0	0x20190402	Reset

SHA_DATE 版本控制寄存器。(R/W)

Register 21.9. SHA_MODE_REG (0x0000)

(reserved)	SHA_MODE	
31 … 3	2 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

SHA_MODE 配置 SHA 加速器的运算标准。

0：SHA-1

1：SHA-224

2：SHA-256

(R/W)

Register 21.10. SHA_DMA_BLOCK_NUM_REG (0x000C)

(reserved)	SHA_DMA_BLOCK_NUM	
31 … 6	5 … 0	
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0x0	Reset

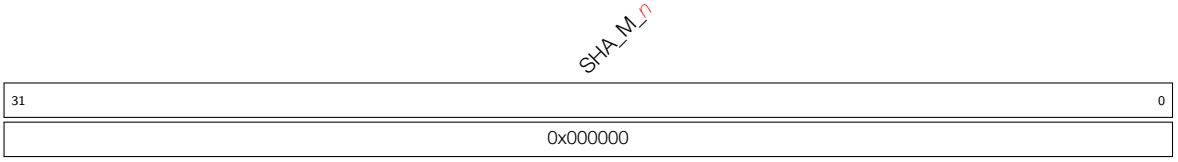
SHA_DMA_BLOCK_NUM 配置 DMA-SHA 工作模式下的信息块个数。(R/W)

Register 21.11. SHA_H_*n*_REG (*n*: 0-7) (0x0040+4**n*)

SHA_H_*n*	
31 … 0	
0x000000	Reset

SHA_H_*n* 表示第 *n* 个 32 位哈希值。(R/W)

Register 21.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4n*)**



SHA_M_*n* 表示第 *n* 个 32 位输入信息。(R/W)

22 数字签名算法 (DSA)

22.1 概述

数字签名算法 (DSA) 使用密码学算法，用于验证消息的真实性和完整性。该算法也可用于向服务器验证设备身份，或验证消息是否经过篡改。

ESP32-H2 包含一个数字签名算法 (Digital Signature Algorithm, DSA) 模块，可提供硬件加速，高效生成基于 RSA 的数字签名。HMAC 作为密钥导出函数 (Key Derivation Function, KDF)，使用存储在 eFuse 中的密钥作为输入密钥，输出 DSA_KEY 密钥。随后，数字签名算法模块使用 DSA_KEY 解密预先加密的参数，计算出签名。上述过程都发生在硬件层面，因此，在计算过程中，不论是解密 RSA 参数的密钥，还是用于 HMAC 密钥导出函数的输入密钥，都对用户不可见。

22.2 主要特性

- 支持长度最大为 3072 位的 RSA 数字签名密钥
- 支持仅限 DSA 读取的加密私钥数据
- 支持 SHA-256 摘要，保护私钥数据免遭攻击者篡改

22.3 功能描述

22.3.1 概述

DSA 模块计算 RSA 签名的方式为 $Z = X^Y \bmod M$，其中 Z 是签名，X 是输入消息，Y 和 M 是 RSA 私钥参数。

私钥参数以密文形式存储在 flash 中。用于解密私钥参数的密钥 (DSA_KEY) 只能由 DSA 模块通过 HMAC 模块计算获取，而生成密钥所需的 HMAC 输入 $HMAC\_KEY$ 则存放在 eFuse 中，只允许被 HMAC 模块访问。也就是说，只有 DSA 硬件才能解密私钥密文，软件无法获取私钥明文。关于 eFuse 和 HMAC 的相关细节请参照章节 5 *eFuse 控制器 (EFUSE)* 和章节 19 *HMAC 加速器 (HMAC)*。

需要签名时，软件直接将输入消息 X 发送到 DSA 外设。RSA 签名计算完成后，软件将读取签名结果 Z。

为方便描述，此处约定几个符号，它们的作用域仅限于本章。

- $\mathbf{1}^s$: 表示一个完全由“1”组成的长度为 s 位的位串。
- $[x]_s$: 一个长度为 s 位的位串，要求 s 为 8 的整数倍。如果 x 是一个数 ($x < 2^s$)，那么其在位串中遵循小端字节序。x 可以是一个变量，例如 $[Y]_{4096}$，或一个十六进制的常数，比如 $[0x0C]_8$。根据需要，$[x]_t$ 右边可以加上$(s-t)$ 个 0，使字串长度扩展成 s 位，得到 $[x]_s$。例如：$[0x05]_8 = 00000101$，$[0x05]_{16} = 0000010100000000$，$[0x0005]_{16} = 0000000000000101$，$[0x13]_8 = 00010011$，$[0x13]_{16} = 0001001100000000$，$[0x0013]_{16} = 0000000000010011$。
- $\|$: 表示位串粘接操作符，用于将两个位串前后粘成一个较长的位串。

22.3.2 私钥运算子

私钥运算子 Y（私钥指数）和 M（密钥模数）由用户生成。它们具有特定的 RSA 密钥长度（最大为 3072 位）。RSA 签名操作还额外需要两个运算子，即参数 $\overline{r}$ 和 M'，由 Y 和 M 运算得到。

用户加密运算子 Y、M、$\overline{r}$、M' 与验证摘要，并以密文 C 的形式存储。密文 C 输入到 DSA 模块之后，先由硬件解密，然后参与 RSA 签名运算。有关生成 C 的方法，请参考章节 22.3.3。

DSA 模块支持运算子长度为 $N = 32 \times x$ ($x \in \{1, 2, 3, \ldots, 96\}$) 的 RSA 签名运算 $Z = X^Y \bmod M$。其中，参数的位宽需要满足 RSA 计算的运算子长度要求，即 Z、X、Y、M 和 $\overline{r}$ 的位宽必须相同，且为 N 中的任意值，而 M' 的位宽则始终是 32。更多 RSA 计算相关信息，请参考章节 20 *RSA 加速器 (RSA)* 中的 20.3.1 大数模幂运算 部分。

22.3.3 软件准备工作

如果用户想使用 DSA 模块进行数字签名，则需要提前做好软件准备工作，如图 22-1 所示。图中左半部分列出了在硬件开始 RSA 签名计算之前软件需要做的准备工作，右半部分则列出了硬件在整个签名计算过程中的计算流程。

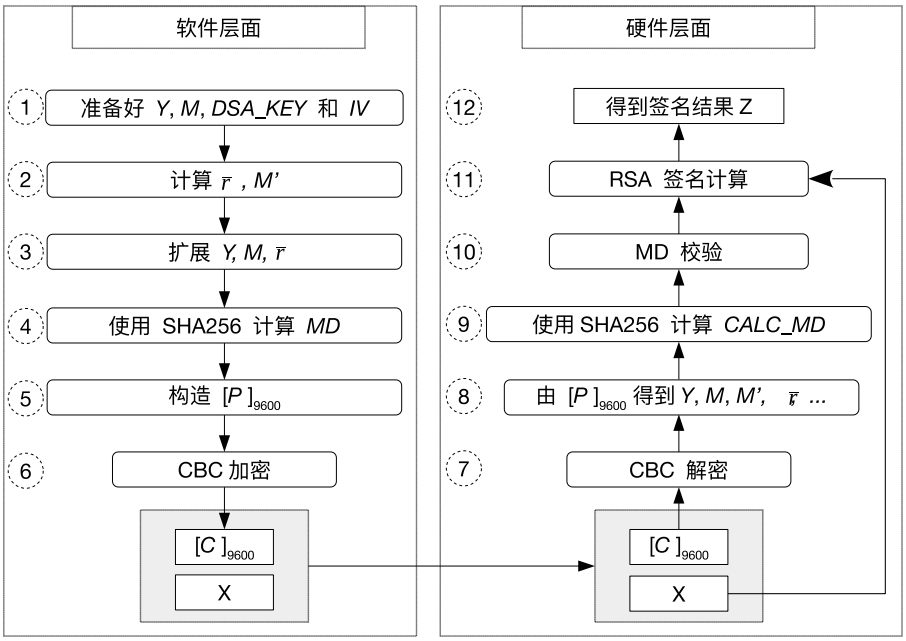


图 22-1. 软件准备工作与硬件工作流程

说明：

1. 即便有多次签名计算，也只需进行一次软件准备工作（图 22-1 左侧），但每一次签名计算都需要重新进行硬件运算（图 22-1 右侧）。
2. 软件准备需要配置时钟复位，相关信息请参考章节 7 复位和时钟。

用户需要依照图 22-1 左侧中指定的步骤来计算 C。详细过程如下：

- **步骤 1**：按照章节 22.3.2 所述，准备好符合运算子长度要求的 RSA 私钥运算子 Y 和 M。定义 $[L]_{32} = \frac{N}{32}$（比如，对于 RSA 3072，$[L]_{32}$ == $[0x60]_{32}$）。准备好 $[HMAC\_KEY]_{256}$，并计算出 $[DSA\_KEY]_{256}$，即 DSA_KEY = HMAC-SHA256($[HMAC\_KEY]_{256}$, 1^{256})。接着，生成一个符合 AES-CBC 块加密算法要求的随机 $[IV]_{128}$。有关 AES 的更多信息，请参考章节 17 *AES 加速器 (AES)*。
- **步骤 2**：根据 M 求解 $\overline{r}$ 和 M'。
- **步骤 3**：扩展 Y、M 和 $\overline{r}$，得到 $[Y]_{3072}$、$[M]_{3072}$ 和 $[\overline{r}]_{3072}$。此步骤是由于 Y、M 和 $\overline{r}$ 的最大位宽为 3072，故位宽小于 3072 的运算子需要扩展至 3072，而位宽等于 3072 的则不需要扩展。

- **步骤 4**：使用 SHA-256 计算 MD 校验码：$[MD]_{256}$ = SHA256 ($[Y]_{3072}||[M]_{3072}||[\overline{r}]_{3072}||[M']_{32}||[L]_{32}||[IV]_{128}$)
- **步骤 5**：构造 $[P]_{9600}$ = ($[Y]_{3072}||[M]_{3072}||[\overline{r}]_{3072}||[Box]_{384}$)。其中，$[Box]_{384}$ = ($[MD]_{256}||[M']_{32}||[L]_{32}||[\beta]_{64}$)，$[\beta]_{64}$ 是符合 PKCS#7 封装方式的追加码，由 8 个字节码 0x80 组成，即 $[0x0808080808080808]_{64}$，目的在于使 P 的长度为 128 位的整数倍。
- **步骤 6**：计算密文形式的私钥参数 C = $[C]_{9600}$ = AES-CBC-ENC ($[P]_{9600}$, $[DSA\_KEY]_{256}$, $[IV]_{128}$)，长度为 1200 字节。C 也可以表示为 C = $[C]_{9600}$ = ($[\widehat{Y}]_{3072}||[\widehat{M}]_{3072}||[\widehat{\overline{r}}]_{3072}||[\widehat{Box}]_{384}$)。其中，$[\widehat{Y}]_{3072}$、$[\widehat{M}]_{3072}$、$[\widehat{\overline{r}}]_{3072}$、$[\widehat{Box}]_{384}$ 是 C 的四个子参数，分别对应 $[Y]_{3072}$、$[M]_{3072}$、$[\overline{r}]_{3072}$、$[Box]_{384}$ 的密文形式。

22.3.4 硬件工作流程

每次需要计算数字签名时，都会触发硬件操作。硬件需要三个输入信息：预先生成的私钥密文 C、唯一消息 X 以及 IV。

DSA 模块的工作流程可以分为三个阶段：

1. **解析阶段，即图 22-1 中的步骤 7 和步骤 8**

   解析过程是图 22-1 中步骤 6 的逆过程。DSA 模块将调用 AES 硬件加速器以 CBC 块模式对输入的密文信息 C 进行解密，获取明文信息。该过程可以表示为 P = AES-CBC-DEC (C, DSA_KEY, IV)。其中，IV 就是 $[IV]_{128}$，由用户直接指定；$[DSA\_KEY]_{256}$ 由硬件 HMAC 提供，通过存储在 eFuse 中的 $HMAC\_KEY$ 得到，软件无法获取。

   通过 P，DSA 模块能够解析出 $[Y]_{3072}$、$[M]_{3072}$、$[\overline{r}]_{3072}$、$[M']_{32}$、$[L]_{32}$、MD 校验码和追加码 $[\beta]_{64}$，这相当于步骤 5 的逆过程。

2. **校验阶段，即图 22-1 中的步骤 9 和步骤 10**

   DSA 模块会执行两种校验操作：MD 校验和填充 (padding) 校验。由于填充校验和 MD 校验同步进行，因此填充校验未在图 22-1 中体现。

   - MD 校验：DSA 模块调用 SHA-256 进行哈希计算获取哈希值 $[CALC\_MD]_{256}$（即步骤 4）。然后，将 $[CALC\_MD]_{256}$ 与 MD 校验码 $[MD]_{256}$ 作比较，当且仅当二者相同时，MD 校验通过。
   - 填充校验：DSA 模块检查解析阶段解析出的追加码 $[\beta]_{64}$ 是否符合 PKCS#7 标准，当且仅当符合标准时，填充校验通过。

   仅当 MD 校验通过后，DSA 模块才会执行后续计算，否则 DSA 模块将拒绝执行。如果填充校验失败，将生成警告信息，但不会影响 DSA 模块的后续操作。

3. **计算阶段，即图 22-1 中的步骤 11 和步骤 12**

   DSA 模块将把用户输入的 X，以及解析得到的 Y、M 和 $\overline{r}$ 都视为大数，结合解析得到的 M'，构成了大数模幂运算 X^Y mod M 的所有必要输入参数。大数模幂运算的运算长度由 L 的值唯一指定。DSA 模块调用 RSA 完成大数模幂运算 $Z = X^Y$ mod M，Z 为签名结果。

22.3.5 软件工作流程

每次需要计算数字签名时，都应执行以下软件操作。输入内容为预先生成的私钥密文 C、唯一的消息 X、IV。本节中的软件步骤会触发章节 22.3.4 中描述的硬件工作流程。
下述流程基于一个假设：软件已经调用了 HMAC 模块，HMAC 已经根据 $HMAC\_KEY$ 计算出了 DSA_KEY。

1. **准备工作**：准备好 C、X、IV。具体方法请参考章节 22.3.3 。

2. **启动 DSA**：对寄存器 DS_SET_START_REG 写 1。
3. **检查 DSA_KEY 是否已经准备好**：轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。

   如果 DS_QUERY_BUSY_REG 超过 1 ms 还没读到 0，则说明 HMAC 初始化出现了问题。此时，软件应当读寄存器 DS_QUERY_KEY_WRONG_REG，根据返回值判断具体是下述哪一种情况：

   - 如果读到零值，说明 HMAC 未被调用。
   - 如果读到非零值 (1 ~ 15)，则说明 HMAC 被调用过，但是 DSA 模块没有成功从 HMAC 模块获取到 DSA_KEY。此种情况可能是因为受到了其他程序的干扰。
4. **配置寄存器**：将 IV block 中的内容写入寄存器 DS_IV_*m*_REG (*m*: 0 ~ 3)。有关 IV block 的更多信息，请参考章节 17 *AES 加速器 (AES)*。
5. **将 X 写入存储器 DS_X_MEM**：将 X_i ($i \in \{0, 1, \ldots, n-1\}$) 写入存储器 DS_X_MEM，容量为 96 个字 (word)，其中 $n = \frac{N}{32}$。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。当 X 的长度小于 96 个字时，存储器 DS_X_MEM 中有一部分空间未使用，该部分空间中的数据可以是任意值。
6. **将 C 写入存储器**：即将 C 中的四个子参数分别写入对应的存储器。
   - 将 $\widehat{Y_i}$ ($i \in \{0, 1, \ldots, 95\}$) 写入存储器 DS_Y_MEM。
   - 将 $\widehat{M_i}$ ($i \in \{0, 1, \ldots, 95\}$) 写入存储器 DS_M_MEM。
   - 将 $\widehat{r_i}$ ($i \in \{0, 1, \ldots, 95\}$) 写入存储器 DS_RB_MEM。
   - 将 $\widehat{Box_i}$ ($i \in \{0, 1, \ldots, 11\}$) 写入存储器 DS_BOX_MEM。

     其中存储器 DS_Y_MEM、DS_M_MEM、DS_RB_MEM 的容量均为 96 个字，而存储器 DS_BOX_MEM 容量只有 12 个字。每一个字刚好存放一个 b 进制数。存储块以小端字节序进行存储，即存储器的低地址存放运算子的低位进制数，高地址存放运算子的高位进制数。
7. **启动计算**：对寄存器 DS_SET_ME_REG 写入 1。
8. **等待运算结束**：轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。
9. **检查校验结果**：读寄存器 DS_QUERY_CHECK_REG，根据返回值决定后续操作。
   - 如果返回值为 0，则说明填充校验通过，MD 校验通过，可以继续读取 Z 结果值。
   - 如果返回值为 1，则说明填充校验通过，但 MD 校验失败。Z 结果值无效，跳至步骤 11。
   - 如果返回值为 2，则说明填充校验失败，但 MD 校验通过，用户可以继续读取 Z 结果值。但仍需注意的是，此时数据填充不符合 PKCS#7 封装方式。
   - 如果返回值为 3，则说明填充校验失败，且 MD 校验失败。这种情况说明有致命错误发生。Z 结果值无效。跳至步骤 11。
10. **读出运算结果**：从存储器 DS_Z_MEM 读出运算结果 Z_i ($i \in \{0, 1, \ldots, n-1\}$)，其中 $n = \frac{N}{32}$。Z 以小端字节序存储在存储器中。
11. **退出计算环境**：对寄存器 DS_SET_FINISH_REG 写入 1，然后轮询寄存器 DS_QUERY_BUSY_REG 直到读到 0。

DSA 退出计算环境后，所有输入/输出寄存器和存储器中的数据都已经被抹除（清零）。

22.4 存储器列表

请注意，这里的起始地址和结束地址都是相对于数字签名算法基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

名称	描述	大小（字节）	起始地址	结束地址	访问
DS_Y_MEM	存储器 Y	384	0x0000	0x017F	WO
DS_M_MEM	存储器 M	384	0x0200	0x037F	WO
DS_RB_MEM	存储器 $\bar{r}$	384	0x0400	0x057F	WO
DS_BOX_MEM	存储器 Box	48	0x0600	0x062F	WO
DS_X_MEM	存储器 X	384	0x0800	0x097F	WO
DS_Z_MEM	存储器 Z	384	0x0A00	0x0B7F	RO

22.5 寄存器列表

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

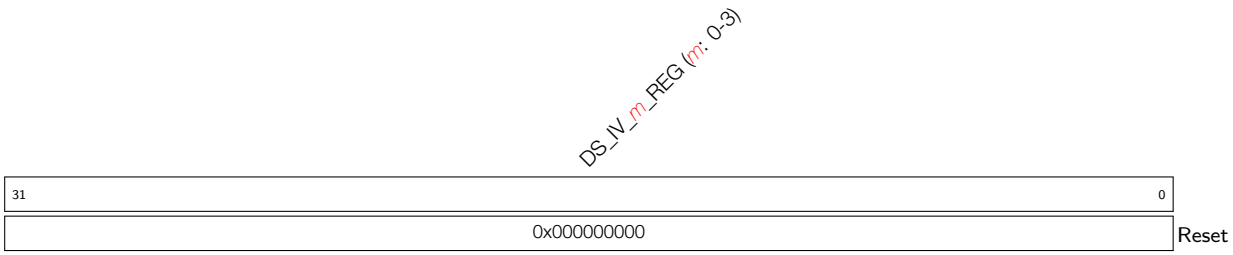
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
DS_IV_0_REG	IV block 数据	0x0630	WO
DS_IV_1_REG	IV block 数据	0x0634	WO
DS_IV_2_REG	IV block 数据	0x0638	WO
DS_IV_3_REG	IV block 数据	0x063C	WO
状态/控制寄存器			
DS_SET_START_REG	启动 DS 模块	0x0E00	WO
DS_SET_ME_REG	开始计算	0x0E04	WO
DS_SET_FINISH_REG	结束计算	0x0E08	WO
DS_QUERY_BUSY_REG	DS 模块状态	0x0E0C	RO
DS_QUERY_KEY_WRONG_REG	查询 DS_KEY 未准备好的原因	0x0E10	RO
DS_QUERY_CHECK_REG	查询校验结果	0x0814	RO
版本寄存器			
DS_DATE_REG	版本控制寄存器	0x0820	R/W

22.6 寄存器

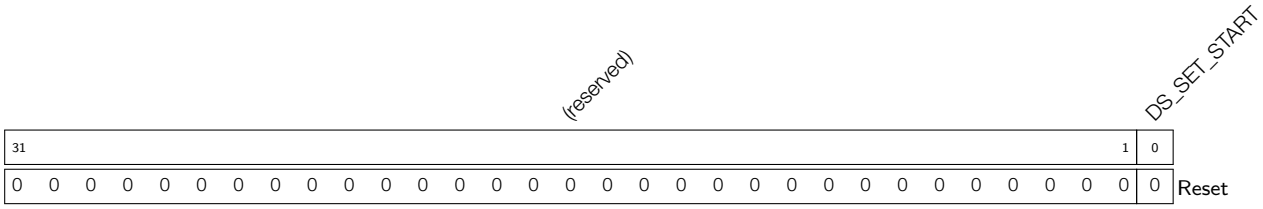
本小节的所有地址均为相对于数字签名算法基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 22.1. DS_IV_*m*_REG (*m*: 0-3) (0x0630+4m*)**



DS_IV_*m*_REG (*m*: 0-3) 写入 IV block 数据。(WO)

Register 22.2. DS_SET_START_REG (0x0E00)



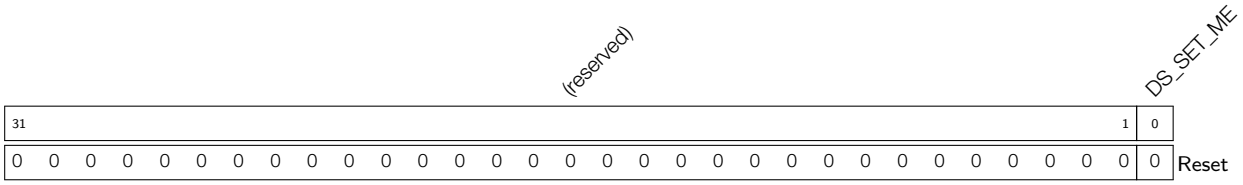
DS_SET_START 配置是否启动 DS 模块。

0：无效

1：启动 DS 模块

(WO)

Register 22.3. DS_SET_ME_REG (0x0E04)



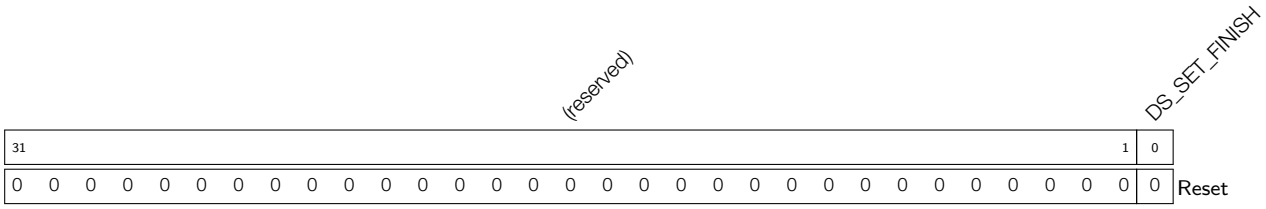
DS_SET_ME 配置是否启动计算。

0：无效

1：启动计算

(WO)

Register 22.4. DS_SET_FINISH_REG (0x0E08)



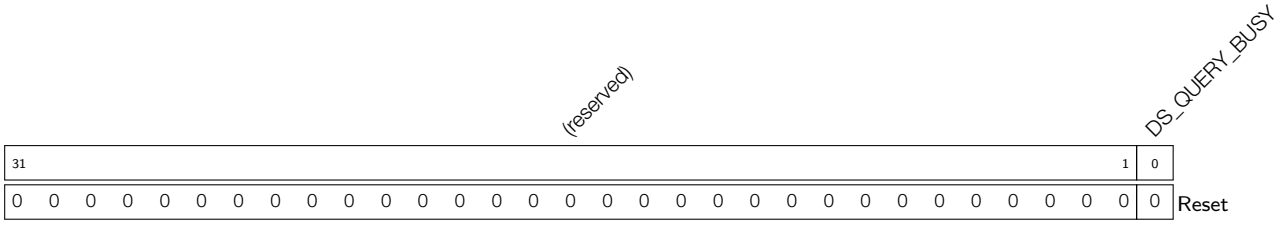
DS_SET_FINISH 配置是否结束运算。

0：无效

1：结束运算

(WO)

Register 22.5. DS_QUERY_BUSY_REG (0x0E0C)



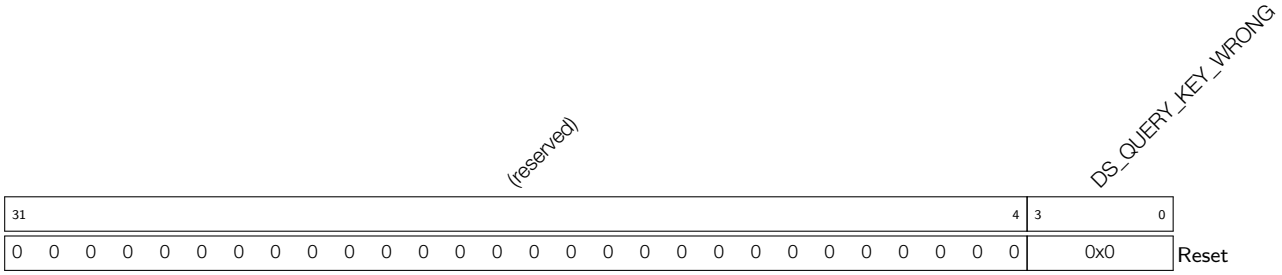
DS_QUERY_BUSY 表示 DS 模块是否空闲。

0：DS 模块空闲

1：DS 模块正忙

(RO)

Register 22.6. DS_QUERY_KEY_WRONG_REG (0x0E10)



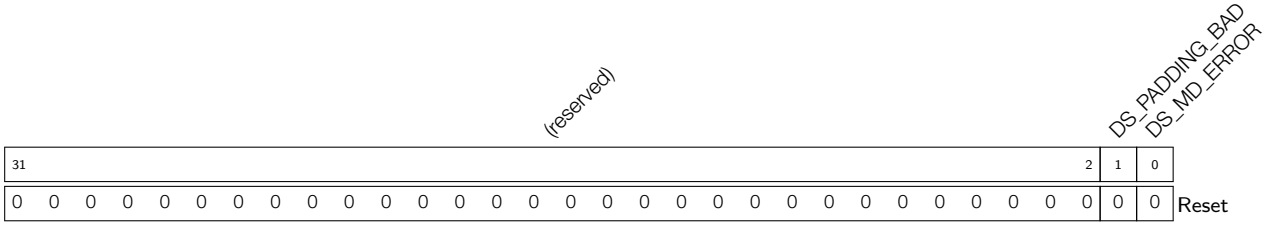
DS_QUERY_KEY_WRONG 表示 HMAC 模块的错误情况。

0：HMAC 未被调用

1-15：HMAC 曾被调用，DS 模块没有成功从 HMAC 模块获取到 DS_KEY

(RO)

Register 22.7. DS_QUERY_CHECK_REG (0x0E14)



DS_PADDING_BAD 表示填充校验是否通过。

0：填充校验通过

1：填充校验失败

(RO)

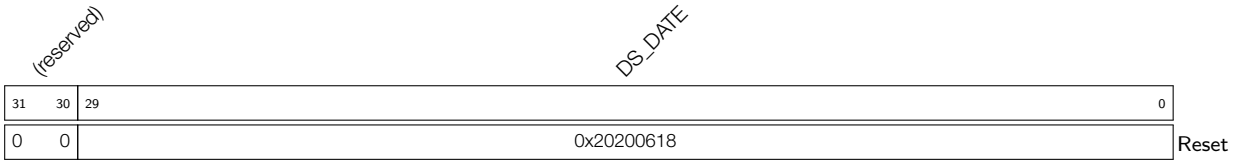
DS_MD_ERROR 表示 MD 校验是否通过。

0：MD 校验通过

1：MD 校验失败

(RO)

Register 22.8. DS_DATE_REG (0x0E20)



DS_DATE 版本控制寄存器。(R/W)

23 椭圆曲线数字签名算法 (ECDSA)

23.1 概述

在密码学中，椭圆曲线数字签名算法 (ECDSA) 是数字签名算法 (DSA) 的一种变体，使用了椭圆曲线密码学。

ESP32-H2 的 ECDSA 加速器可在一个安全高效的环境中计算 ECDSA 签名，不仅可保证签名过程的机密性，还实现了快速计算，非常适合一些对加密安全性和加密速度有高要求的应用。ECDSA 加速器允许用户在不牺牲性能的情况下，保证数据得到保护。

23.2 主要特性

ESP32-H2 的 ECDSA 加速器支持：

- 数字签名的生成和验证
- 两种椭圆曲线，分别是 P-192 和 P-256（具体定义见 FIPS 186-3 规范）
- 两种哈希算法，分别是 SHA-224 和 SHA-256（具体定义见 FIPS PUB 180-4 规范）
- 不同工作状态下的动态访问权限，以确保信息安全

23.3 ECDSA 背景知识

23.3.1 域参数

ECDSA 使用的参数通常被称为域参数，可用于生成密钥、生成签名和验证签名等运算，主要包括有限域中定义的椭圆曲线参数、曲线上的基点和曲线阶数等。

ECDSA 加速器中使用的域参数主要包括两大类：

- 椭圆曲线域参数：
  - 质数模数 p，指定定义椭圆曲线的有限域的大小。
  - 曲线上的基点 G，用于生成公钥。
  - 曲线的阶数 n，代表重复将 G 进行点加而生成曲线上的点的次数。
- 哈希函数参数：
  - 哈希算法，用于将要签名的消息生成固定长度的哈希值。
  - 哈希值的长度，决定签名的大小。

这些参数共同定义了 ECDSA 域，并且对于安全使用 ECDSA 至关重要。

23.3.2 密钥生成

ECDSA 中的密钥生成过程如下所述：

1. 选择一个椭圆曲线域，定义如下参数：
   - 质数模数 p
   - 系数 a 和 b

- 基点 G 和其阶数 n

2. 生成私钥：
   - 私钥 d 为在 1 到 n-1 之间随机选择的整数。注意，这里必须确保私钥是真正随机的（不可预测或复制）。因此，使用一个安全的随机数发生器非常重要。
   - 私钥用于生成签名，需妥善保密保管。
   - 私钥选择完成后，应将其烧录到 ESP32-H2 中的 eFuse OTP 中。（请参考第 5 章 *eFuse 控制器 (EFUSE)*）。
3. 计算公钥：
   - 公钥 Q 的计算方式为 $Q = dG$。
4. 导出公钥：
   - 公钥 Q 通常以 (Qx，Qy) 表示，可公开给他人。

23.3.3 签名生成

ECDSA 签名生成过程如下所述：

1. 选择一条消息 m，用于签名。
2. 计算消息 m 的哈希值 e：e = HASH(m)，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息哈希值 e 的摘要 z：令 z 为 e 的左端 L_n 位，其中 L_n 是曲线阶数 n 的位长度。
4. 选择一个随机数 k：范围在 1 到 n-1 之间，其中 n 是椭圆曲线上基点的阶数。
5. 计算签名：签名计算如下：
   (a) 计算点 (x, y) = kG
   (b) 计算 r = x mod n。如果 r 等于零，返回上一步并选择新的 k 值。
   (c) 计算 s = k^{-1} * ($z + d * r$) mod n。如果 s 等于零，返回步骤 4 并选择新的 k 值。
   (d) 签名为 (r, s)。
6. 发送消息 m 和签名 (r, s) 给接收者。

23.3.4 签名验证

接收方可以使用与用于签名生成的私钥相关联的公钥来验证签名。验证过程涉及检查签名是否是使用正确的私钥生成的，以及签名是否对给定的消息有效。

ECDSA 签名验证过程如下所述：

1. 接收消息 m 和签名 (r, s)。
2. 计算消息的哈希值：e 等于 HASH(m)，其中 HASH 是一个密码学哈希函数，例如 SHA-256。
3. 计算消息的摘要：令 z 为 e 的左端 L_n 位，其中 L_n 是曲线阶数 n 的位长度。
4. 验证签名：签名验证如下所示：
   (a) 验证 r 和 s 是否是介于 1 到 n-1 之间的整数，其中 n 是椭圆曲线上基点的阶数。如果 r 或 s 超出此范围，则签名无效。

(b) 计算 $u_1 = z * w \bmod n$ 和 $u_2 = r * w \bmod n$。

(c) 计算点 $(x_1, y_1) = u_1 * G + u_2 * Q$，其中 G 是椭圆曲线上的基点，Q 是与用于签名生成的私钥相关联的公钥。

(d) 验证 $r = x_1 \bmod n$。如果 r 不等于 $x_1 \bmod n$，则签名无效。

5. 接受或拒绝签名：如果签名有效，接收方可以接受消息作为真实的。否则，接收方将拒绝消息作为无效的。

如果签名有效，接收方可以确信消息没有被篡改，且消息来自预期的发送方。

23.4 功能描述

本节描述了 ESP32-H2 的 ECDSA 加速器的详细信息。

23.4.1 ECDSA 工作模式

ESP32-H2 内置的 ECDSA 加速器具有两种工作模式，即签名生成模式和签名验证模式。

用户可以通过配置 ECDSA_WORK_MODE，根据下表 23-1 选择 ECDSA 加速器的工作模式。

表 23-1. ECDSA 工作模式

ECDSA_WORK_MODE	工作模式
0	签名验证
1	签名生成

用户可以通过配置 ECDSA_ECDSA_CURVE，根据下表 23-2 选择使用的椭圆曲线。

表 23-2. ECDSA 椭圆曲线选择

ECDSA_ECC_CURVE	椭圆曲线
0	P-192
1	P-256

用户可以通过配置 ECDSA_SHA_MODE，根据下表 23-3 选择用于消息哈希的 SHA 算法。

表 23-3. ECDSA SHA 算法

ECDSA_SHA_MODE	SHA 算法
1	SHA-224
2	SHA-256
其他	无效

此外，用户可以通过查询 ECDSA_STATE_REG 寄存器并将返回值与下表 23-4 进行比较，来检查 ECDSA 加速器的工作状态。

表 23-4. ECDSA 工作状态

ECDSA_STATE_REG	状态	描述
0	IDLE	空闲或已完成的运算，对应于 IDLE 阶段。
1	LOAD	等待用户将信息加载到 ECDSA 中，对应于 LOAD 阶段。
2	GAIN	等待用户从 ECDSA 获取信息，对应于 GAIN 阶段。
3	BUSY	正在硬件运行中，对应于 PREP、PROC 和 POST 阶段。

23.4.2 数据和数据块

ESP32-H2 的 ECDSA 加速器会用到的数据位宽均为 256 或 512 位。假设一个数据为 $D[255:0]$，则其可以被划分为八个 32 位的数据块 $D[n][31:0](n=0,1,\cdots,7)$，序号数低的数据块对应二进制低位。以 256 位的数据为例：

$$D[255:0]=D[7][31:0],D[6][31:0],D[5][31:0],D[4][31:0],D[3][31:0],D[2][31:0],D[1][31:0],D[0][31:0]$$

23.4.2.1 数据存储

数据存储即将数据存储进一个内存块的运算，该数据为 ECDSA 算法的输入数据。具体而言，将数据写入一个 ECDSA 内存块相当于将该数据 $D[n][31:0](n=0,1,\cdots,7)$ 依次写入 “该内存块起始地址 + $4\times n$”：以 256 位的数据为例：

- 将 $D[0]$ 写入 “起始地址”
- 将 $D[1]$ 写入 “起始地址 + 4”
- ...
- 将 $D[7]$ 写入 “起始地址 + 28”

说明：
在 192 位模式下进行数据存储运算时，需要在数据的高位补 0，保证存储的数据为 256 位。

23.4.2.2 数据读取

数据读取即将一个数据从一个内存块读出的运算，该数据为 ECDSA 算法的输出数据。具体来说，从一个 ECDSA 内存块读数据相当于从“该内存块起始地址 + $4 \times n$”依次读出 $D[n][31:0](n = 0, 1, \cdots, 7)$：以 256 位的数据为例：

- 从“起始地址”读出 $D[0]$
- 从“起始地址 + 4”读出 $D[1]$
- ...
- 从“起始地址 + 28”读出 $D[7]$

说明：
在 192 位模式下进行数据读取运算时，只需要读取低 192 位（即 6 个数据块）的数据。

23.4.2.3 消息填充

SHA 加速器仅能处理长度为 512 位及其整倍数的信息。因此，在将信息送至 SHA 加速器进行运算前，应先通过软件运算将信息填充为符合要求的长度。

假设待处理信息 M 的长度为 L_M 位，则填充步骤如下：

1. 首先，在待处理信息后填充 1 个“1”；
2. 随后，再填充 L_A 个“0”。其中，L_A 为满足 $L_M + 1 + L_A \equiv 448 \ mod \ 512$ 的最小非负数解；
3. 最后，在末尾填充一个 64 位的信息块。该信息块的内容为用二进制表示的待处理信息的长度，即 L_M 的值。

更多详情，请参考 《FIPS PUB 180-4 规范》 > 章节 “Padding the Message”。

23.4.2.4 解析消息

在完成信息填充后，我们还需将待处理信息（及其填充）解析为 N 个 512 位的信息块，即 $M^{(1)}$、$M^{(2)}$、...、$M^{(N)}$。

说明：
1. 有关“解析消息”的详细信息，请参考 FIPS PUB 180-4 规范 > 章节 “Parsing the Message”。
2. 有关“信息块”及相关概念的描述，请参考 FIPS PUB 180-4 规范 > 章节 “Glossary of Terms and Acronyms”。

23.4.3 安全功能

为确保 ECDSA 运算过程的安全性，ECDSA 加速器实现了多种安全功能。

23.4.3.1 动态访问权限

ESP32-H2 的 ECDSA 加速器实施了动态访问权限机制，可有效防止运行过程中可能发生的非法配置篡改或数据访问，从而保证密钥的安全性。

具体来说，ECDSA 的寄存器在不同工作状态下的访问权限是不同的。例如，ECDSA_CONF_REG 寄存器仅在加速器处于 IDLE 状态时允许读写运算。这样一来，当加速器处于其他状态（如 LOAD、GAIN 和 BUSY）时，该寄存器中的配置信息则不允许读取或写入，从而得到很好的保护。有关 ECDSA 所有工作状态的详细信息，请见表 23-4。

有关每个 ECDSA 寄存器动态访问权限的详细信息，请参阅寄存器摘要。

23.4.3.2 硬件占用

在 ECDSA 运算过程中，ESP32-H2 的 ECDSA 加速器将占用以下硬件模块：

- SHA 加速器
- RSA 加速器
- ECC 加速器

其中，SHA 加速器在触发 ECDSA_SHA_RELEASE_INT 时将释放，而 RSA 加速器和 ECC 加速器在整个 ECDSA 运算中将被占用。

说明：

硬件占用是一种保护复用模块和存储空间的机制。当一个模块被硬件占用时，用户将无法执行以下运算：

- 读取或写入模块的寄存器或存储器数据。
- 关闭模块时钟。
- 复位模块。

在硬件占用结束时，占用的模块将自动复位。此外，当用户对主模块执行软件复位时，所有占用的模块将同时复位。

23.5 编程指南

23.5.1 ECDSA 流程

ECDSA 的完整工作流程包括以下六个阶段。

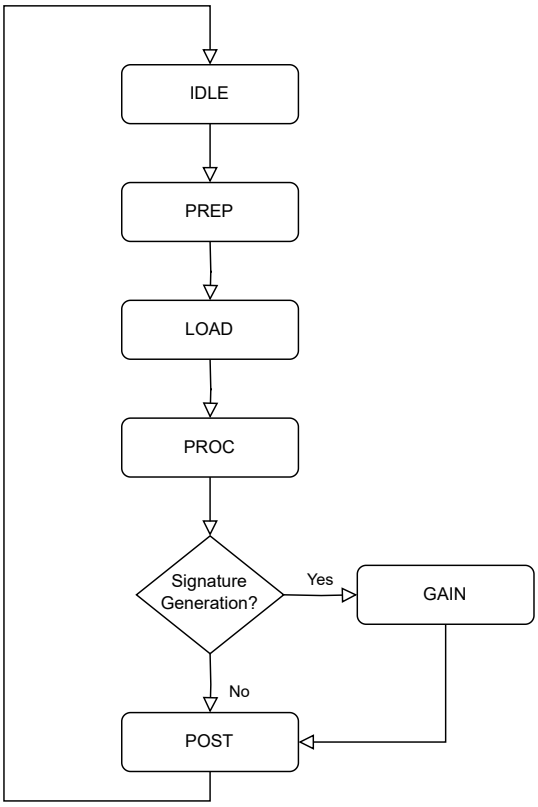


图 23-1. ECDSA 流程

每个阶段的详细编程流程在下面的章节中描述。

23.5.1.1 IDLE 阶段

在 IDLE 阶段：

1. 配置静态参数，包括 eFuse 位：
   (a) ECDSA_KEY：ECDSA 中的私钥 d 的值。为了正确配置 eFuse 中的密钥值，用户需要将密钥值写入 KEY*n*（n = 0 ~ 5），并将相应的 EFUSE_KEY_PURPOSE_*n* 设置为 ECDSA_KEY。有关更详细的配置步骤，请参阅第 5 章 *eFuse 控制器 (EFUSE)*。
   (b) EFUSE_ECDSA_FORCE_USE_HARDWARE_K：配置是否允许用户软件输入 k。有关配置步骤，请参阅第 5 章 *eFuse 控制器 (EFUSE)*。
2. 配置配置寄存器，包括以下字段：
   (a) ECDSA_WORK_MODE：选择 ECDSA 加速器的工作模式。
   (b) ECDSA_ECC_CURVE：选择 ECDSA 加速器的椭圆曲线。
   (c) ECDSA_SOFTWARE_SET_Z：选择使用直接输入的 z。
3. 配置寄存器字段 ECDSA_START 以进入 PREP 阶段。

23.5.1.2 PREP 阶段

在 PREP 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA 加速器执行准备工作。

1. 通过轮询 ECDSA_BUSY 的状态直到其不为 BUSY，表示 PREP 阶段已结束。然后 ECDSA 加速器将自动进入 LOAD 阶段。

23.5.1.3 LOAD 阶段

在 LOAD 阶段：

1. 使用以下选项之一将输入 z 提供给 ECDSA 加速器：
   - 直接输入 z：将 z 写入 ECDSA_Z_MEM。
   - ECDSA-SHA 接口：从消息生成 z。有关详细信息，请参阅第 23.5.1.7 节。
2. 根据所选的 ECDSA_WORK_MODE，进行以下配置：
   - 签名验证：
     (a) 将签名（r，s）写入 ECDSA_R_MEM 和 ECDSA_S_MEM。
     (b) 将公钥（Qx，Qy）写入 ECDSA_QX_MEM 和 ECDSA_QY_MEM。
   - 签名生成：
     (a) 不需要其他配置。
3. 将 1 写入 ECDSA_LOAD_DONE，表示配置完成。然后加速器将自动进入 PROC 阶段。

23.5.1.4 PROC 阶段

在 PROC 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA 加速器根据所选的工作模式执行 ECDSA 运算。

1. 通过轮询 ECDSA_BUSY 直到其不为 BUSY，表示 PROC 阶段已结束。然后 ECDSA 加速器将根据所选的工作模式自动进入 GAIN 阶段或 POST 阶段：
   - 签名生成：GAIN 阶段
   - 签名验证：POST 阶段

23.5.1.5 GAIN 阶段

当选择了签名生成模式时，ECDSA 加速器在 PROC 阶段之后进入 GAIN 阶段：

1. 从 ECDSA 内存中读取数据：
   - 从 ECDSA_R_MEM 和 ECDSA_S_MEM 读取签名（r, s）。
   - 从 ECDSA_QX_MEM 和 ECDSA_QY_MEM 读取公钥（Qx, Qy）。
2. 向 ECDSA_GAIN_DONE 写入 1，表示 GAIN 阶段已完成。然后加速器将自动进入 POST 阶段。

23.5.1.6 POST 阶段

在 POST 阶段，ECDSA_STATE_REG 为 BUSY，ECDSA 加速器执行 ECDSA 运算的一些收尾工作。

1. 通过轮询 ECDSA_BUSY 直到不再为 BUSY，表示 POST 阶段已结束。然后 ECDSA 加速器将自动返回 IDLE 阶段。

23.5.1.7 ECDSA SHA 接口

ESP32-H2 的 ECDSA 加速器可以自动执行哈希运算并基于直接输入的消息生成 z。

对于消息哈希，ECDSA 加速器支持 SHA 算法 SHA-224（仅当选择 P192 作为椭圆曲线时有效）和 SHA-256。

要使用 ECDSA SHA 接口，请完成以下步骤：

1. 根据第 23.4.2.3 节中的描述，对消息进行填充。
2. 将消息及其填充解析为消息块。详细信息请参见第 23.4.2.4 节。
3. 处理当前消息块。
   - 将当前消息块写入 ECDSA_MEM_M。
4. 启动 ECDSA SHA 接口 [1]。
   - 如果这是第一次执行此步骤，请将 1 写入 ECDSA_SHA_START 以启动 ECDSA SHA 接口；
   - 如果不是第一次执行此步骤 [2]，请将 1 写入 ECDSA_SHA_CONTINUE 以继续运算。
5. 查询当前信息块的处理进度。
   - 轮询寄存器 ECDSA_SHA_BUSY，直到其为 0，表示接口已完成当前消息块的运算，现在处于 “IDLE” 状态。
6. 处理下一个消息块：
   - 如有，则返回到步骤 3。
   - 如没有更多的消息块，退出。

说明：

1. 在此步骤中，软件还可以在接口启动 SHA 运算时将下一个要处理的消息块（如有）写入寄存器 ECDSA_MEM_M，以节省时间。
2. 您正在继续之前暂停的 ECDSA SHA 接口运算。

23.5.2 时钟和复位

ESP32-H2 的 ECDSA 加速器有一个时钟模块和一个复位模块，其激活仅需使能 PCR_ECDSA_CONF_REG 外围时钟的 PCR_ECDSA_CLK_EN 位，并同时清零 PCR_ECDSA_RST_EN 位。有关如何配置 ECDSA 时钟和复位的详细信息，请参考第 7 章复位和时钟。

23.5.3 中断

ESP32-H2 的 ECDSA 加速器可以生成一个中断信号 ECDSA_INTR 并发送给 中断矩阵。

ECDSA 加速器有两个中断源可产生中断信号 ECDSA_INTR：

- ECDSA_CALC_DONE_INT：在 ECC 运算完成时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_CALC_DONE_INT_RAW：存储 ECDSA_CALC_DONE_INT 的原始中断状态。
  - ECDSA_CALC_DONE_INT_ST：代表 ECDSA_CALC_DONE_INT 中断的状态。通过 ECDSA_CALC_DONE_INT_RAW 字段启用/禁用此字段生成。

  - ECDSA_CALC_DONE_INT_ENA：启用/禁用 ECDSA_CALC_DONE_INT 中断。
  - ECDSA_CALC_DONE_INT_CLR：将此位设置为 1 以清除 ECDSA_CALC_DONE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_CALC_DONE_INT_RAW 和 ECDSA_CALC_DONE_INT_ST 将被清除。

- ECDSA_SHA_RELEASE_INT：在 SHA 释放时触发。可以通过以下寄存器配置此中断源：
  - ECDSA_SHA_RELEASE_INT_RAW：存储 ECDSA_SHA_RELEASE_INT 的原始中断状态。
  - ECDSA_SHA_RELEASE_INT_ST：代表 ECDSA_SHA_RELEASE_INT 中断的状态。通过 ECDSA_SHA_RELEASE_INT_RAW 字段启用/禁用此字段生成。
  - ECDSA_SHA_RELEASE_INT_ENA：启用/禁用 ECDSA_SHA_RELEASE_INT 中断。
  - ECDSA_SHA_RELEASE_INT_CLR：将此位设置为 1 以清除 ECDSA_SHA_RELEASE_INT 中断状态。通过将此位设置为 1，字段 ECDSA_SHA_RELEASE_INT_RAW 和 ECDSA_SHA_RELEASE_INT_ST 将被清除。

说明：

中断、中断源、中断信号等术语的解释以及它们之间的联系请参考章节 9 中断矩阵 *(INTMTX)* > 9.2 *ESP32-H2 中断术语*。

23.6 存储器块

ECDSA 的存储器块用于存储 ECDSA 运算的输入数据和输出数据。

表 23-5. ECDSA 存储器块

存储器块	大小（字节）	起始地址*	结束地址 *	访问权限
ECDSA_MEM_M	64	0x280	0x2BF	R/W
ECDSA_MEM_R	32	0xA00	0xA1F	R/W
ECDSA_MEM_S	32	0xA20	0xA3F	R/W
ECDSA_MEM_Z	32	0xA40	0xA5F	R/W
ECDSA_MEM_Qx	32	0xA60	0xA7F	R/W
ECDSA_MEM_Qy	32	0xA80	0xA9F	R/W

* 相对于 ECDSA 加速器基地址的地址偏移在第 4 章*系统和存储器*中的表 4-2 中提供。

23.7 寄存器列表

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

为了提高安全性，在不同工作状态下，ECDSA 寄存器的读写权限不同。以下是每个工作状态的缩写及其与阶段的对应关系：

- PI：IDLE 状态，对应 IDLE 阶段。
- PL：LOAD 状态，对应 LOAD 阶段。
- PG：GAIN 状态，对应 GAIN 阶段。
- PB：BUSY 状态，对应 PREP、PROC 和 POST 阶段。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

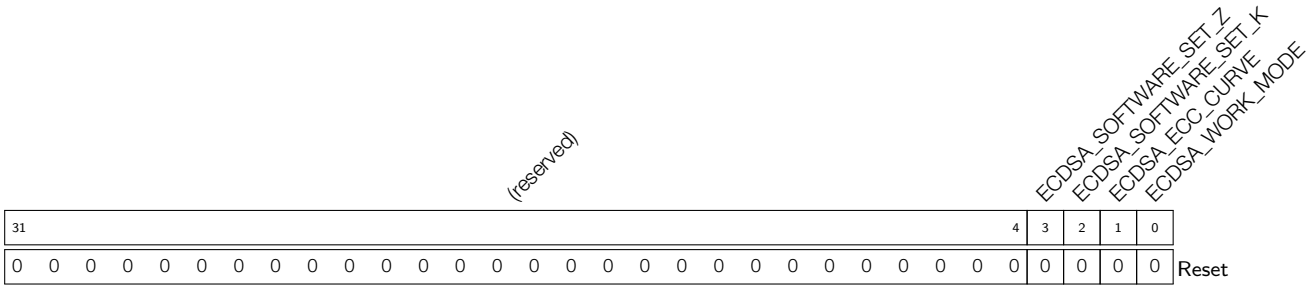
名称	描述	地址	访问权限			
			PI	PL	PG	PB
数据存储器	见表 18-1。					
配置寄存器						
ECDSA_CONF_REG	ECDSA 配置寄存器	0x0004	R/W	N/A		
ECDSA_START_REG	ECDSA 启动寄存器	0x001C	WT			N/A
时钟和复位寄存器						
ECDSA_CLK_REG	ECDSA 时钟门控寄存器	0x0008	R/W	N/A		
中断寄存器						
ECDSA_INT_RAW_REG	ECDSA 中断原始状态寄存器	0x000C	RO/WTC/SS			
ECDSA_INT_ST_REG	ECDSA 中断状态寄存器	0x0010	RO			
ECDSA_INT_ENA_REG	ECDSA 中断使能寄存器	0x0014	R/W			
ECDSA_INT_CLR_REG	ECDSA 中断清除寄存器	0x0018	WT			
状态寄存器						
ECDSA_STATE_REG	ECDSA 状态寄存器	0x0020	RO			
结果寄存器						
ECDSA_RESULT_REG	ECDSA 结果寄存器	0x0024	RO/SS			N/A
SHA 寄存器						
ECDSA_SHA_MODE_REG	ECDSA 控制 SHA 寄存器	0x0200	N/A	R/W	N/A	
ECDSA_SHA_START_REG	ECDSA 控制 SHA 寄存器	0x0210	N/A	WT	N/A	
ECDSA_SHA_CONTINUE_REG	ECDSA 控制 SHA 寄存器	0x0214	N/A	WT	N/A	
ECDSA_SHA_BUSY_REG	ECDSA 控制 SHA 状态寄存器	0x0218	N/A	RO	N/A	
版本寄存器						
ECDSA_DATE_REG	版本控制寄存器	0x00FC	R/W	N/A		

23.8 寄存器

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

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

Register 23.1. ECDSA_CONF_REG (0x0004)



ECDSA_WORK_MODE 配置 ECDSA 加速器的工作模式。

0：签名生成模式

1：签名验证模式

(R/W)

ECDSA_ECC_CURVE 配置使用的椭圆曲线。

0：P-192

1：P-256

(R/W)

ECDSA_SOFTWARE_SET_K 配置参数 k 的配置方式。

0：TRNG 自动生成

1：软件写入

(R/W)

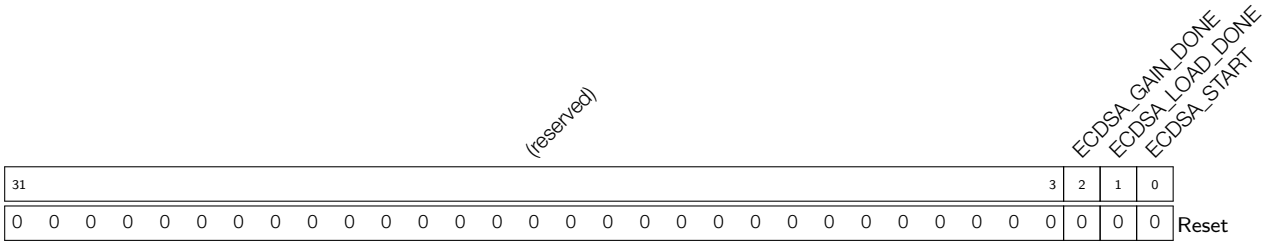
ECDSA_SOFTWARE_SET_Z 配置参数 z 的配置方式。

0：SHA 结果生成

1：软件写入

(R/W)

Register 23.2. ECDSA_START_REG (0x001C)



ECDSA_START 配置是否启动 ECDSA 运算。此位配置后自动清 0。

0：无效

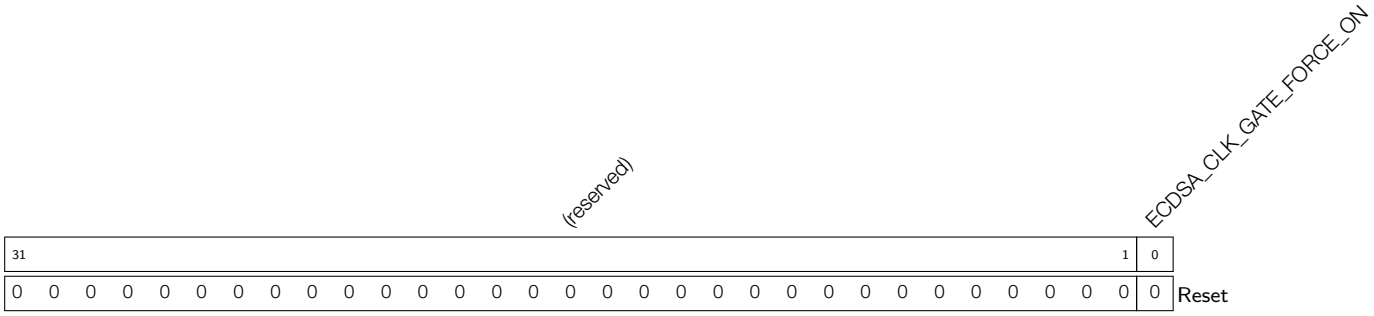
1：启动 ECDSA 运算

(WT)

ECDSA_LOAD_DONE 写 1 生成中断信号，表示 ECDSA 加速器的 LOAD 阶段已完成。此位配置后自动清 0。(WT)

ECDSA_GAIN_DONE 写 1 生成中断信号，表示 ECDSA 加速器的 GET 阶段已完成。此位配置后自动清 0。(WT)

Register 23.3. ECDSA_CLK_REG (0x0008)



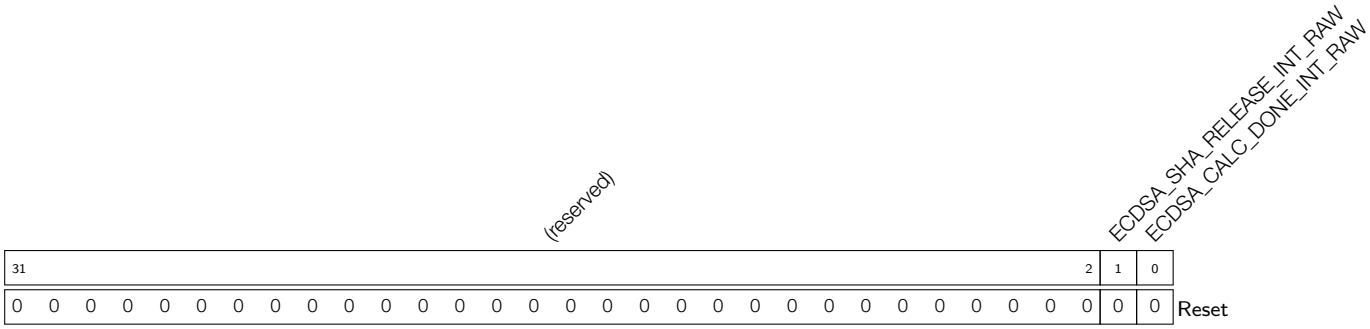
ECDSA_CLK_GATE_FORCE_ON 配置是否强制启动 ECC 内存的时钟门控。

0：无效

1：强制启动

(R/W)

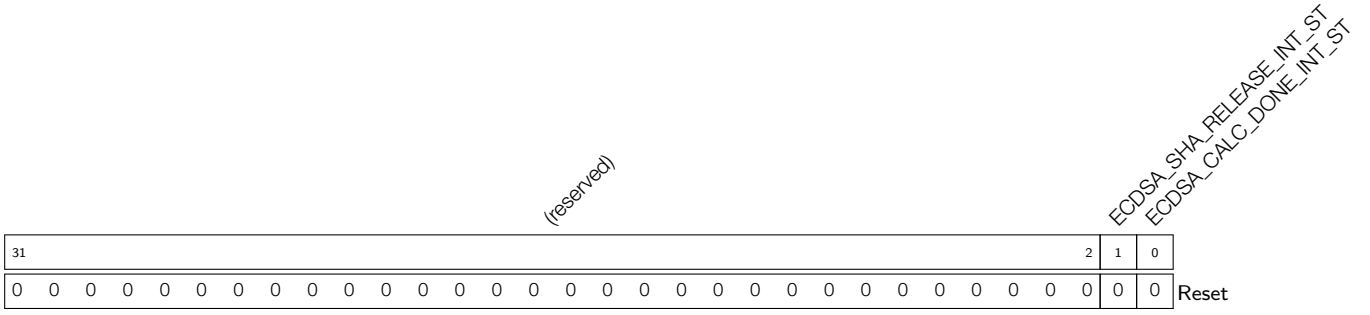
Register 23.4. ECDSA_INT_RAW_REG (0x000C)



ECDSA_CALC_DONE_INT_RAW ECDSA_CALC_DONE_INT 的原始中断状态。(RO/WTC/SS)

ECDSA_SHA_RELEASE_INT_RAW ECDSA_SHA_REKEASE_INT 的原始中断状态。(RO/WTC/SS)

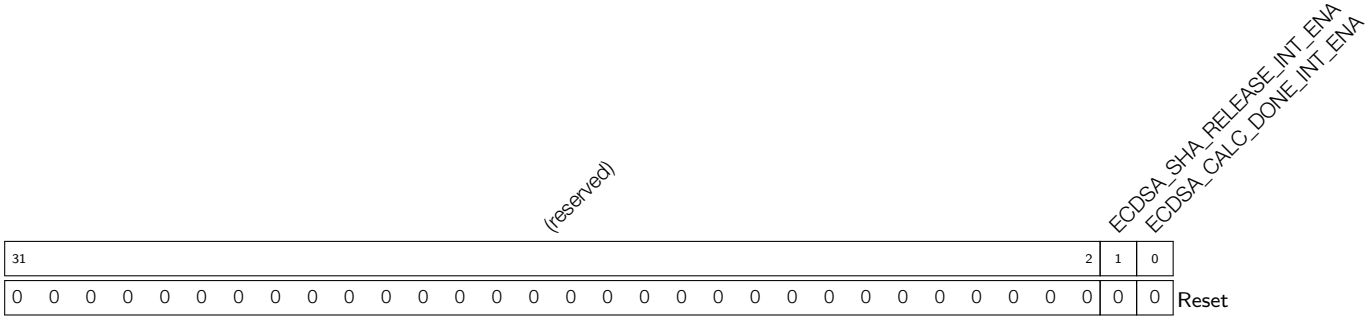
Register 23.5. ECDSA_INT_ST_REG (0x0010)



ECDSA_CALC_DONE_INT_ST ECDSA_CALC_DONE_INT 的屏蔽中断状态。(RO)

ECDSA_SHA_RELEASE_INT_ST ECDSA_SHA_REKEASE_INT 的屏蔽中断状态。(RO)

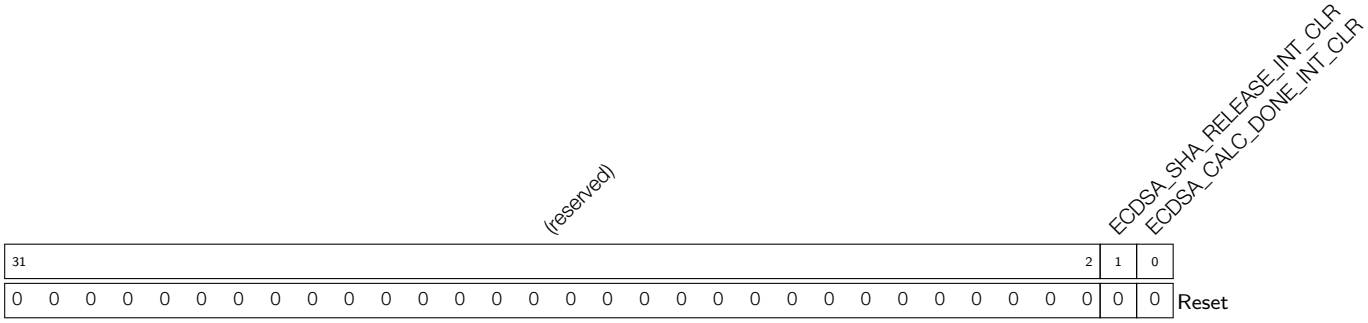
Register 23.6. ECDSA_INT_ENA_REG (0x0014)



ECDSA_CALC_DONE_INT_ENA 写 1 使能 ECDSA_CALC_DONE_INT 中断。(R/W)

ECDSA_SHA_RELEASE_INT_ENA 写 1 使能 ECDSA_SHA_REKEASE_INT 中断。(R/W)

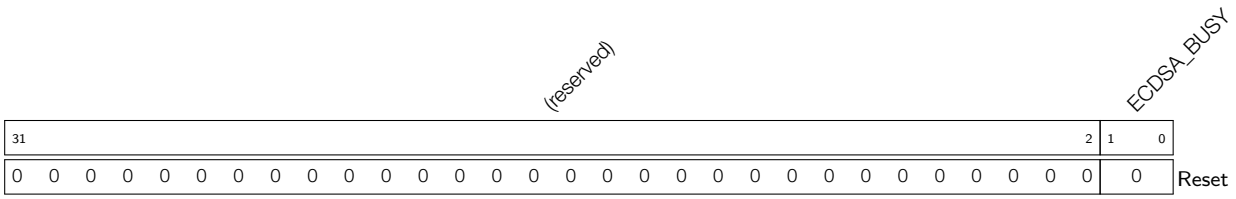
Register 23.7. ECDSA_INT_CLR_REG (0x0018)



ECDSA_CALC_DONE_INT_CLR 写 1 清除 ECDSA_CALC_DONE_INT 中断。(WT)

ECDSA_SHA_RELEASE_INT_CLR 写 1 清除 ECDSA_SHA_REKEASE_INT 中断。(WT)

Register 23.8. ECDSA_STATE_REG (0x0020)



ECDSA_BUSY 表示 ECDSA 加速器的工作状态。

0：IDLE

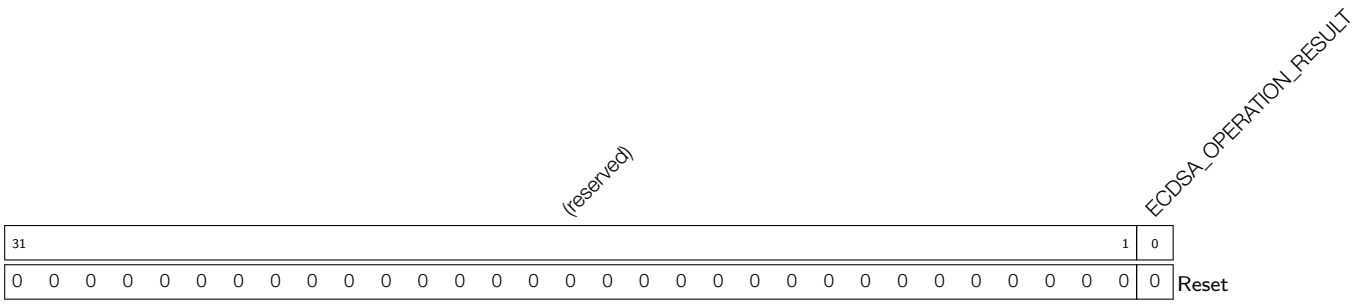
1：LOAD

2：GET

3：BUSY

(RO)

Register 23.9. ECDSA_RESULT_REG (0x0024)



ECDSA_OPERATION_RESULT 表示 ECDSA 运算是否成功。

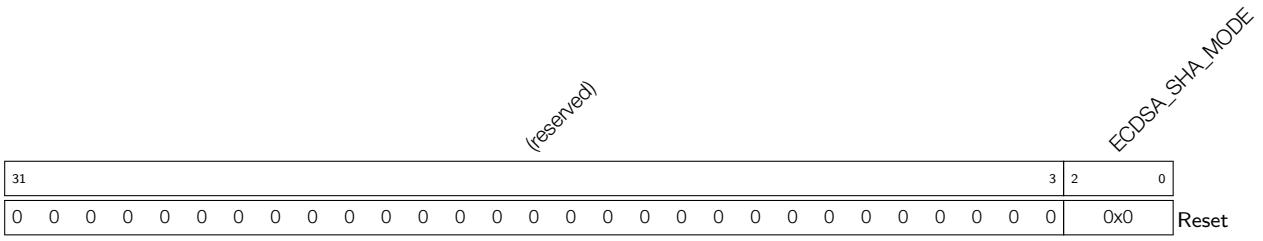
0：失败

1：成功

仅在 ECDSA 运算完成后有效。

(RO/SS)

Register 23.10. ECDSA_SHA_MODE_REG (0x0200)



ECDSA_SHA_MODE 配置哈希运算的 SHA 算法。

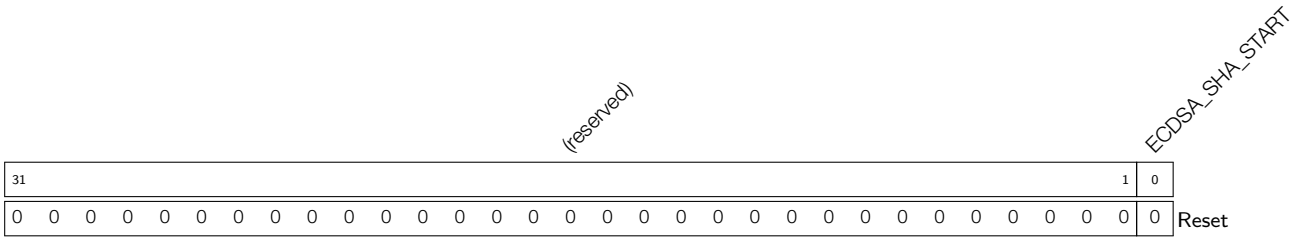
1：SHA-224

2：SHA-256

其他：无效

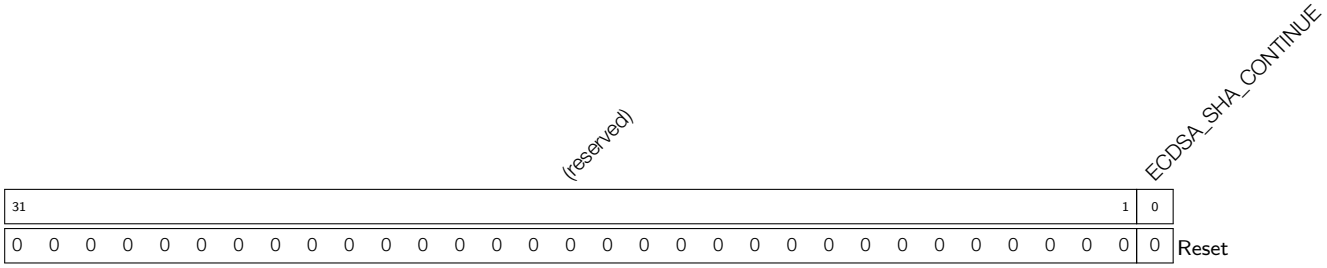
(R/W)

Register 23.11. ECDSA_SHA_START_REG (0x0210)



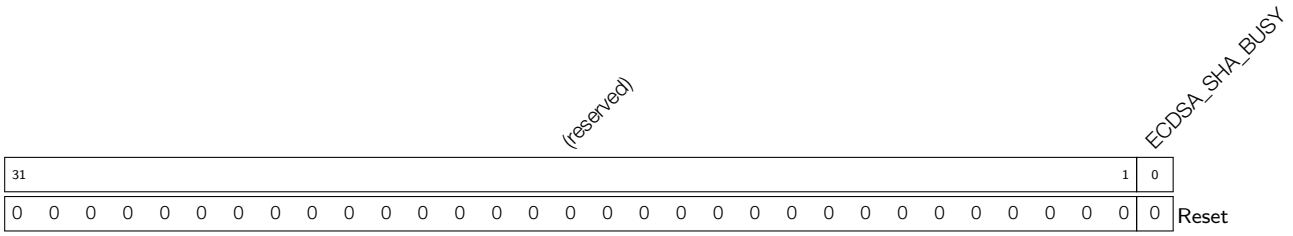
ECDSA_SHA_START 写 1 启动 ECDSA 运算中的首个 SHA 运算。此位配置后自动清 0。(WT)

Register 23.12. ECDSA_SHA_CONTINUE_REG (0x0214)



ECDSA_SHA_CONTINUE 写 1 启动 ECDSA 运算中的后续 SHA 运算。此位配置后自动清 0。(WT)

Register 23.13. ECDSA_SHA_BUSY_REG (0x0218)



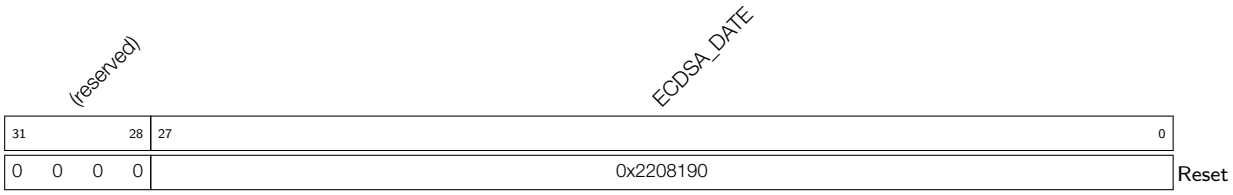
ECDSA_SHA_BUSY 表示 ECDSA 运算过程中 SHA 加速器的工作状态。

0：IDLE

1：BUSY

(RO)

Register 23.14. ECDSA_DATE_REG (0x00FC)



ECDSA_DATE ECDSA 版本控制寄存器。(R/W)

24 片外存储器加密与解密 (XTS_AES)

24.1 概述

ESP32-H2 芯片集成了片外存储器加密与解密模块，使用 IEEE Std 1619-2007 指定的 XTS-AES 标准算法，为用户存放在片外存储器 (flash) 的应用代码和数据提供了安全保障。用户可以将专有固件、敏感的用户数据（如用来访问私有网络的证书）存放在片外 flash 中。

24.2 主要特性

- 使用通用 XTS-AES 算法，符合 IEEE Std 1619-2007
- 支持手动加密，需要软件参与
- 支持高速自动解密，无需软件参与
- 由寄存器配置、eFuse 参数、启动 (boot) 模式共同决定开启/关闭加解密功能
- 支持可配置的抗 DPA 攻击功能

24.3 模块结构

片外存储器加解密模组包含两个模块：手动加密 (Manual Encryption) 模块和自动解密 (Auto Decryption) 模块。模块结构如图 24-1 所示。

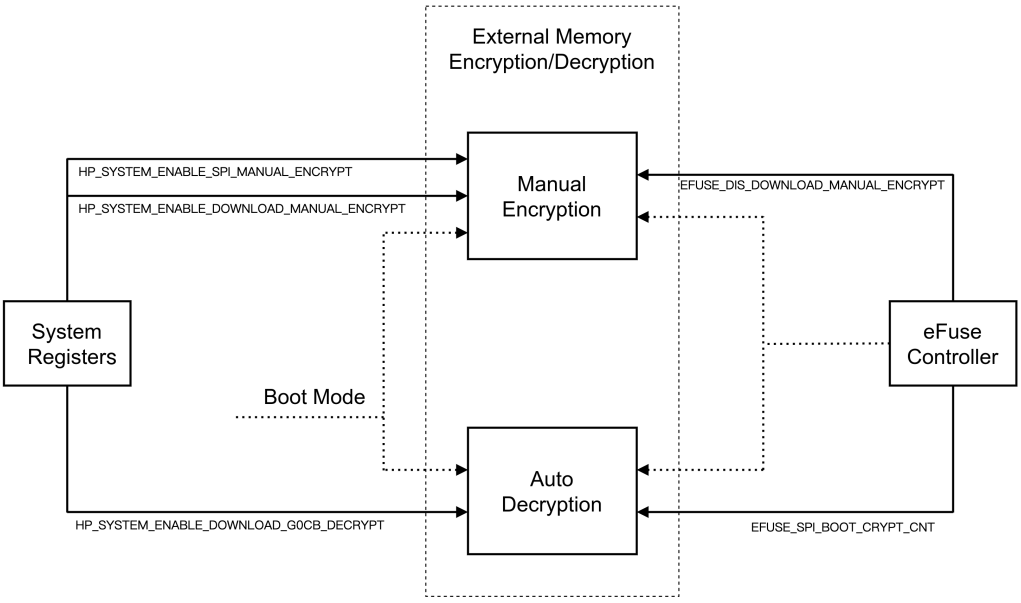


图 24-1. 片外存储器加解密结构

手动加密模块能够对指令和数据进行加密，加密后的指令/数据随即以密文状态由 SPI1 写入片外 flash。

系统寄存器外设中（请参见章节 15 系统寄存器），寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 内与片外存储器加解密相关的有以下 3 位：

- HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT

- HP_SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT
- HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

此外，片外存储器加解密模组还会从外设 eFuse 控制器中获取两个参数：EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 和 EFUSE_SPI_BOOT_CRYPT_CNT。更多详细信息，请参考章节 5 *eFuse 控制器 (EFUSE)*。

24.4 功能描述

24.4.1 XTS 算法

手动加密模块和自动解密模块均使用 XTS 算法。具体实现中，XTS 算法使用 1024 位为一个数据单元 (data unit)，此处的“数据单元”由 *XTS-AES Tweakable Block Cipher* 标准中的章节 *XTS-AES encryption procedure* 定义。更多关于 XTS-AES 算法的信息，请参考 IEEE Std 1619-2007。

24.4.2 密钥

执行 XTS 算法时，手动加密模块和自动解密模块共用同一密钥 Key。密钥 Key 来自硬件 eFuse，软件无法访问获取。

Key 的长度为 256 位。Key 的值由 BLOCK4 ~ BLOCK9 中一个 eFuse 块的参数信息决定。为方便阐述如何通过 eFuse 参数信息推导出 Key 的值，现定义：

- Block$_A$：BLOCK4 ~ BLOCK9 中密钥用途为 EFUSE_KEY_PURPOSE_XTS_AES_128_KEY 的 BLOCK（请参考表 5-2 *结构*）。如果 Block$_A$ 存在，那么 Block$_A$ 中存放着 256 位的 Key_A。

根据 Block$_A$ 是否存在，有两种对应的生成 Key 的方式。任意情况下，Key 值都唯一确定，如表 24-1 所示。

表 24-1. 根据 Key_A 生成的 Key 值

Block$_A$ 是否存在	Key	Key 长度（位）
是	Key_A	256
否	0^{256}	256

说明：

- “0^{256}” 表示由 256 个位 0 组成的位串。
- 使用 0^{256} 作为密钥 Key 存在风险，建议配置有效密钥。

更多有关密钥用途的信息，请参考章节 5 *eFuse 控制器 (EFUSE)* 中的表 5-2 *结构*。

24.4.3 目标空间

目标空间指的是片外存储器 (flash) 中存放密文的一段连续地址空间。目标空间可由目标大小和目标基地址唯一确定。这两个参数的定义如下：

- 目标大小：目标空间的大小 ($size$)，即单次加密的数据量。以字节为单位，支持 16 字节、32 字节或 64 字节。
- 目标基地址：目标空间的基地址 ($base\_addr$)。目标基地址为 24 位的物理地址，取值范围为 0x0000_0000 ~ 0x00FF_FFFF，但要求以 $size$ 为单位对齐，即 $base\_addr\%size == 0$。

例如，在一次加密操作中，需要将 16 字节的指令数据加密后写入片外 flash 中的地址段 0x130 ~ 0x13F，则目标空间为 0x130 ~ 0x13F，目标大小为 16 (字节)，目标基地址为 0x130。

对于任意长度（必须是 16 字节的整数倍）的明文指令或数据的加密，整个加密过程可拆分成多次进行，每次加密均具备相应的目标空间和相应参数。

对于自动解密模块，目标空间等参数由硬件自动调节。对于手动加密模块，目标空间等参数需由用户主动配置。

说明：

IEEE Std 1619-2007 中的章节 *Data units and tweaks* 中定义的 “tweak” 是一个 128-bit 的非负整数（$tweak$），其值可以通过公式求出：$tweak$ = ($base\_addr$ & 0x00FFFF80)。$tweak$ 中低 7 位和高 97 位恒为零。

24.4.4 数据写入

对于自动解密模块，数据写入由硬件自动完成。对于手动加密模块，数据写入需由用户主动配置。手动加密模块中包含一个由 16 个寄存器组成的寄存器块 XTS_AES_PLAIN_n_REG (n: 0 ~ 15)，专用于数据写入，一次可以存放最多 256 位明文指令/数据。

实际上，在手动加密模块中，相对于明文的来源，密文将要存放的位置更为重要。基于明文和密文之间严格的对应关系，为更好地描述明文在寄存器块中的存放方式，现假设明文从一开始就存放在目标空间中，在加密完成后替换为密文。因此，本节中将不再出现 “明文” 的概念，而用 “目标空间” 来代替。

目标空间映射到寄存器块的方法：

假设目标空间中某个字的存放地址为 $address$，记 $offset = address\%64$，$n = offset/4$，那么该字将被存放在寄存器 XTS_AES_PLAIN_n_REG 中。

例如，当目标大小为 64 时，寄存器块中的所有寄存器都将被使用，目标空间中的地址与寄存器块之间的映射关系如表 24-2 所示。

表 24-2. 目标空间与寄存器堆的映射关系

$offset$	寄存器	$offset$	寄存器
0x00	XTS_AES_PLAIN_0_REG	0x20	XTS_AES_PLAIN_8_REG
0x04	XTS_AES_PLAIN_1_REG	0x24	XTS_AES_PLAIN_9_REG
0x08	XTS_AES_PLAIN_2_REG	0x28	XTS_AES_PLAIN_10_REG
0x0C	XTS_AES_PLAIN_3_REG	0x2C	XTS_AES_PLAIN_11_REG
0x10	XTS_AES_PLAIN_4_REG	0x30	XTS_AES_PLAIN_12_REG
0x14	XTS_AES_PLAIN_5_REG	0x34	XTS_AES_PLAIN_13_REG
0x18	XTS_AES_PLAIN_6_REG	0x38	XTS_AES_PLAIN_14_REG
0x1C	XTS_AES_PLAIN_7_REG	0x3C	XTS_AES_PLAIN_15_REG

24.4.5 手动加密模块

手动加密模块是一个外设模块，自身带有寄存器，可以被 CPU 直接访问。模块内的寄存器、系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，手动加密模块才允许手动加密：

- SPI Boot 模式下：

当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 位为 1 时，手动加密模块拥有工作权限，否则无法工作。

- Download Boot 模式下：

  当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 位为 1，且 eFuse 参数 EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 为 0 时，手动加密模块拥有工作权限，否则无法工作。

说明：

- 即使 CPU 可以越过 cache，直接读取片外存储器从而得到加密指令/数据，但用户依旧无法获取密钥 Key。

24.4.6 自动解密模块

自动解密并非传统外设模块，自身不带寄存器，不能被 CPU 直接访问。系统寄存器外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当满足下列对应条件时，自动解密模块才允许自动解密：

- SPI Boot 模式下

  当参数 SPI_BOOT_CRYPT_CNT（3 位）中奇数个位为 1 时，自动解密模块拥有工作权限，否则无法工作。

- Download Boot 模式下

  当寄存器 HP_SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 HP_SYSTEM_ENABLE_DOWNLOAD_G0CB_DECRYPT 位为 1 时，自动解密模块拥有工作权限，否则无法工作。

说明：

1. 当自动解密模块拥有工作权限时，如果 CPU 通过 cache 读取片外存储器中的指令/数据，自动解密将自动对读取到的密文进行解密以恢复指令/数据。解密的整个过程无需软件参与，且对 cache 透明。解密算法过程中，密钥 Key 无法被软件获取。
2. 当自动解密模块没有工作权限时，无论数据加密与否，自动解密模块不对片外存储器中的数据产生作用。此时，CPU 通过 cache 读取到的是片外存储器中的原始内容。

24.5 软件流程

手动加密模块工作时需要软件参与，软件流程为：

1. 配置 XTS_AES：
   - 将寄存器 XTS_AES_PHYSICAL_ADDRESS_REG 的值设置为 $base\_addr$。
   - 将寄存器 XTS_AES_LINESIZE_REG 的值设置为 $\frac{size}{32}$。

   关于 $base\_addr$ 和 $size$ 的定义，请参考章节 24.4.3。
2. 将明文指令/数据写入至寄存器块 XTS_AES_PLAIN_n_REG (n: 0 ~ 15)。更多详细信息，请参考章节 24.4.4。
   请根据实际需求写入寄存器，未使用的寄存器可为任意值。

3. 等待手动加密模块进入空闲状态。轮询寄存器 XTS_AES_STATE_REG 直到软件读取到 0，表明手动加密模块已进入空闲状态。
4. 向寄存器 XTS_AES_TRIGGER_REG 写入 1，启动手动加密。
5. 等待加密完成。轮询寄存器 XTS_AES_STATE_REG，直到软件读取到 2。
   上述步骤为使用 *Key* 操作手动加密模块对明文指令/数据进行加密的过程。
6. 向寄存器 XTS_AES_RELEASE_REG 写入 1，使 SPI1 获得密文的访问权限。该操作完成后，寄存器 XTS_AES_STATE_REG 的值将为 3。
7. 调用 SPI1，将密文写入片外 flash（请参阅 ESP-IDF 编程指南 中的 Flash 加密 API 参考）。
8. 向寄存器 XTS_AES_DESTROY_REG 写入 1，销毁密文。该操作完成后，寄存器 XTS_AES_STATE_REG 的值将为 0。

请根据所需加密的明文指令/数据的数量，重复上述步骤。

24.6 抗 DPA 攻击

DPA（Differential Power Analysis, 差分能量分析）是密码学中的一种旁路攻击手段，攻击者可以统计分析从多个加密操作收集的数据，从而计算加密计算中的中间值。ESP32-H2 XTS_AES 模块支持抗 DPA 攻击功能，以抵御来自外界的 DPA 攻击。根据 IEEE Std 1619-2007，XTS-AES 算法可分为两步：

- 第一步：计算 T 的值。下文中，将此步骤称为“计算 T”。
- 第二步：计算密文/明文。下文中，将此步骤称为“计算 D”。

用户可以通过寄存器配置实现不同安全等级：

- 首先，为更好地描述各步骤，定义以下参数：
  - $select\_reg$ = XTS_AES_CRYPT_DPA_SELECT_REGISTER
  - $reg\_d\_dpa\_en$ = XTS_AES_CRYPT_CALC_D_DPA_EN
  - $efuse\_dpa\_en$ = EFUSE_CRYPT_DPA_ENABLE
  - $reg\_anti\_dpa\_level$ = XTS_AES_CRYPT_SECURITY_LEVEL
  - $efuse\_anti\_dpa\_level = 3$
- 配置 XTS_AES 模块抗 DPA 攻击的安全等级：

$$Anti\_DPA\_level = select\_reg\ ?\ (reg\_anti\_dpa\_level) : (efuse\_dpa\_en * efuse\_anti\_dpa\_level)$$

  当 $Anti\_DPA\_level$ 为 0 时，关闭抗 DPA 攻击功能。$Anti\_DPA\_level$ 的值越大，抗 DPA 攻击的能力就越强。

- 配置是否在 XTS-AES 算法计算 D 时开启抗 DPA 功能：

$$Anti\_DPA\_enabled\_in\_calc\_D = select\_reg\ ?\ reg\_d\_dpa\_en : efuse\_dpa\_en$$

  在 $Anti\_DPA\_level$ 不为 0 的情况下，当 $Anti\_DPA\_enabled\_in\_calc\_D$ 等于 1 时，XTS-AES 算法在计算 D 时开启抗 DPA 功能。
  在 $Anti\_DPA\_level$ 不为 0 的情况下，XTS-AES 算法计算 T 时总是开启抗 DPA 功能。

说明：

- 在 efuse_dpa_en 置 1 后，用户仍可通过配置 $select\_reg = 1$ 及 $reg\_anti\_dpa\_level = 0$ 关闭抗 DPA 功能。
- 配置是否开启抗 DPA 功能会对外存访问带宽造成影响：
  - 在计算 D 过程中配置开启抗 DPA 功能时，当抗攻击等级 >=4，读写带宽将降低超过 50%。
  - 在计算 D 过程中配置关闭抗 DPA 功能时，当抗攻击等级 >=6，读写带宽将降低超过 50%。

24.7 寄存器列表

本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

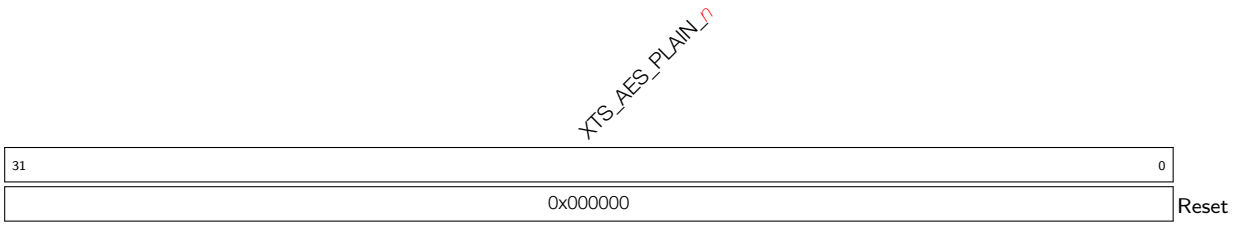
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
明文寄存器堆			
XTS_AES_PLAIN_0_REG	明文寄存器 0	0x0300	R/W
XTS_AES_PLAIN_1_REG	明文寄存器 1	0x0304	R/W
XTS_AES_PLAIN_2_REG	明文寄存器 2	0x0308	R/W
XTS_AES_PLAIN_3_REG	明文寄存器 3	0x030C	R/W
XTS_AES_PLAIN_4_REG	明文寄存器 4	0x0310	R/W
XTS_AES_PLAIN_5_REG	明文寄存器 5	0x0314	R/W
XTS_AES_PLAIN_6_REG	明文寄存器 6	0x0318	R/W
XTS_AES_PLAIN_7_REG	明文寄存器 7	0x031C	R/W
XTS_AES_PLAIN_8_REG	明文寄存器 8	0x0320	R/W
XTS_AES_PLAIN_9_REG	明文寄存器 9	0x0324	R/W
XTS_AES_PLAIN_10_REG	明文寄存器 10	0x0328	R/W
XTS_AES_PLAIN_11_REG	明文寄存器 11	0x032C	R/W
XTS_AES_PLAIN_12_REG	明文寄存器 12	0x0330	R/W
XTS_AES_PLAIN_13_REG	明文寄存器 13	0x0334	R/W
XTS_AES_PLAIN_14_REG	明文寄存器 14	0x0338	R/W
XTS_AES_PLAIN_15_REG	明文寄存器 15	0x033C	R/W
配置寄存器			
XTS_AES_LINESIZE_REG	配置目标空间的大小	0x0340	R/W
XTS_AES_DESTINATION_REG	配置片外存储器的类型	0x0344	R/W
XTS_AES_PHYSICAL_ADDRESS_REG	配置物理地址	0x0348	R/W
XTS_AES_DPA_CTRL_REG	配置抗 DPA 功能	0x0388	R/W
控制/状态寄存器			
XTS_AES_TRIGGER_REG	启动 AES 算法	0x034C	WO
XTS_AES_RELEASE_REG	释放控制寄存器	0x0350	WO
XTS_AES_DESTROY_REG	销毁控制寄存器	0x0354	WO
XTS_AES_STATE_REG	状态寄存器	0x0358	RO
版本寄存器			
XTS_AES_DATE_REG	版本控制寄存器	0x035C	R/W

24.8 寄存器

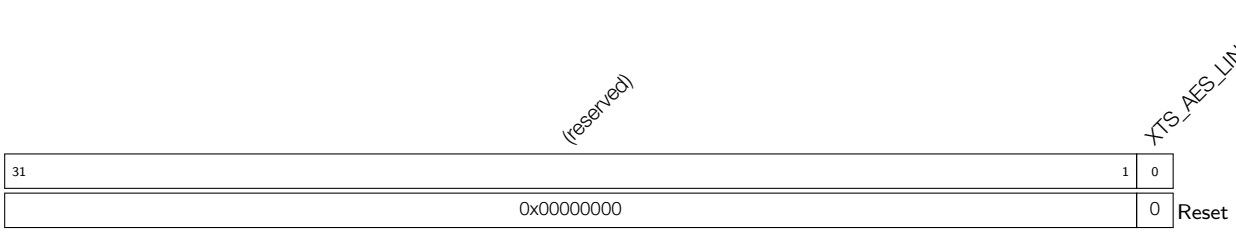
本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 24.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-15) (0x0300+4n*)**



XTS_AES_PLAIN_*n* 配置明文的第 *n* 个 32 位部分。(R/W)

Register 24.2. XTS_AES_LINESIZE_REG (0x0340)



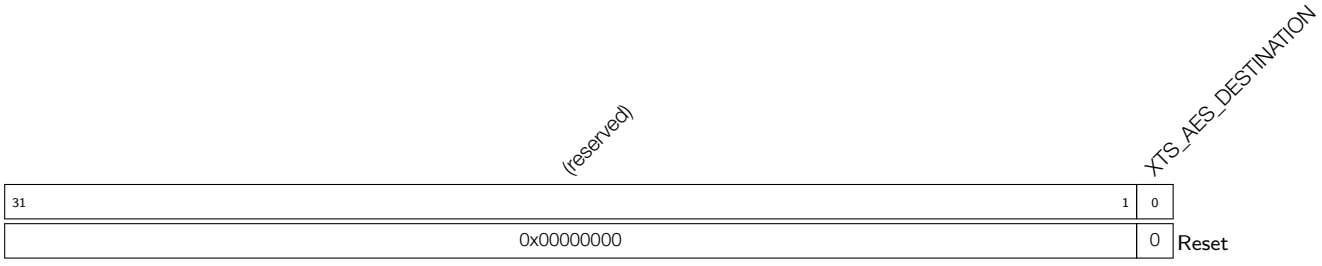
XTS_AES_LINESIZE 配置单次加密的数据大小。

0：加密 16 字节

1：加密 32 字节

(R/W)

Register 24.3. XTS_AES_DESTINATION_REG (0x0344)



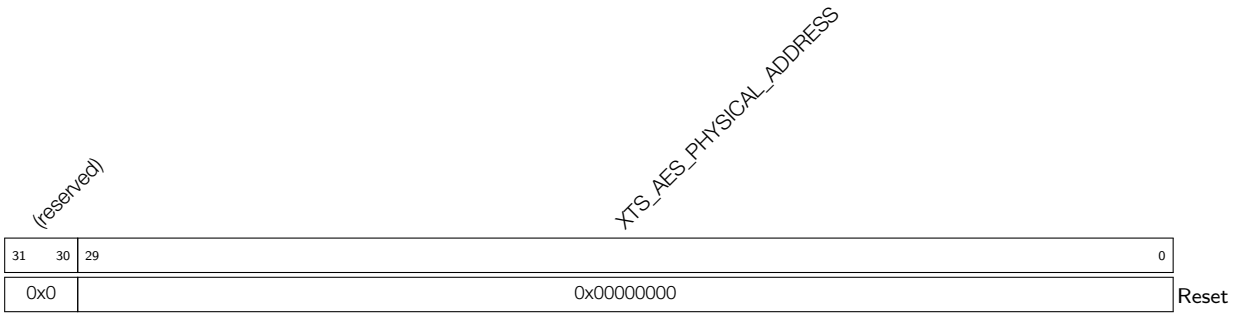
XTS_AES_DESTINATION 决定手动加密类型，目前只能手动加密 flash，所以只能为 0。用户不能写入 1，否则将发生错误。

0：加密 flash

1：加密片外 RAM

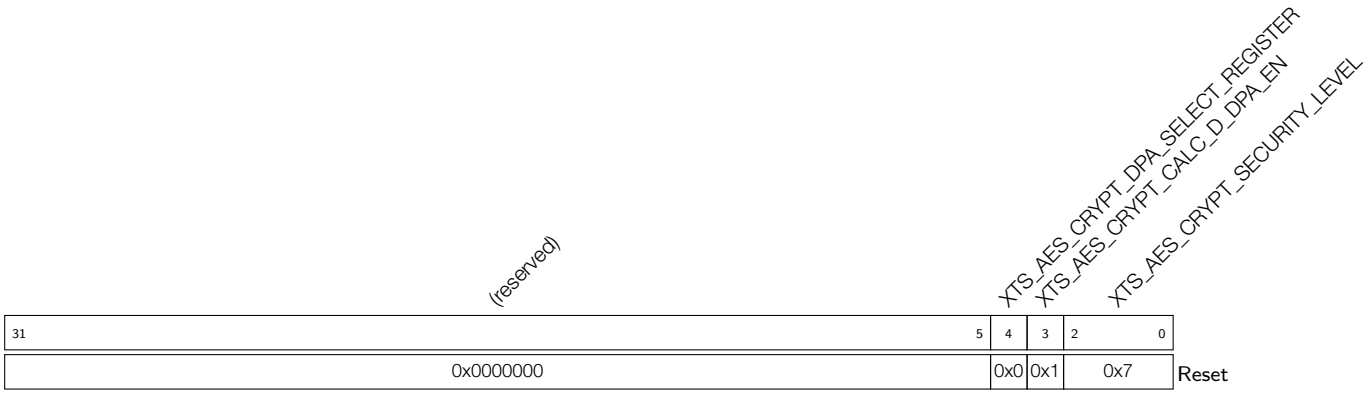
(R/W)

Register 24.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0348)



XTS_AES_PHYSICAL_ADDRESS 配置物理地址。请注意，该值范围必须为 0x0000_0000 ~ 0x00FF_FFFF。(R/W)

Register 24.5. XTS_AES_DPA_CTRL_REG (0x0388)



XTS_AES_CRYPT_DPA_SELECT_REGISTER 配置由 eFuse 或寄存器控制抗 DPA 功能。

0：由寄存器控制抗 DPA 功能

1：由 eFuse 控制抗 DPA 功能

(R/W)

XTS_AES_CRYPT_CALC_D_DPA_EN 配置是否在 XTS_AES 算法中启用抗 DPA 功能。

0：仅在计算 T 时启用抗 DPA 功能

1：计算 T 或是计算 D 时都启用抗 DPA 功能

请注意，此字段仅在 XTS_AES_CRYPT_SECURITY_LEVEL 不为 0 时有效。

(R/W)

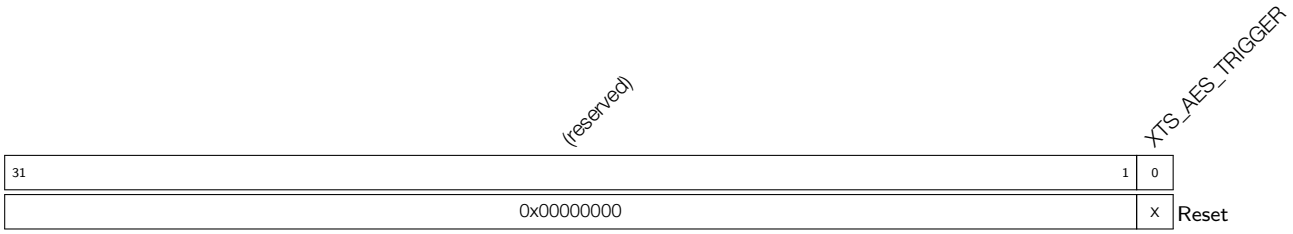
XTS_AES_CRYPT_SECURITY_LEVEL 配置外部加解密的安全等级。

0：抗 DPA 功能已关闭

1-7：数字越大，则加解密安全性更高

(R/W)

Register 24.6. XTS_AES_TRIGGER_REG (0x034C)



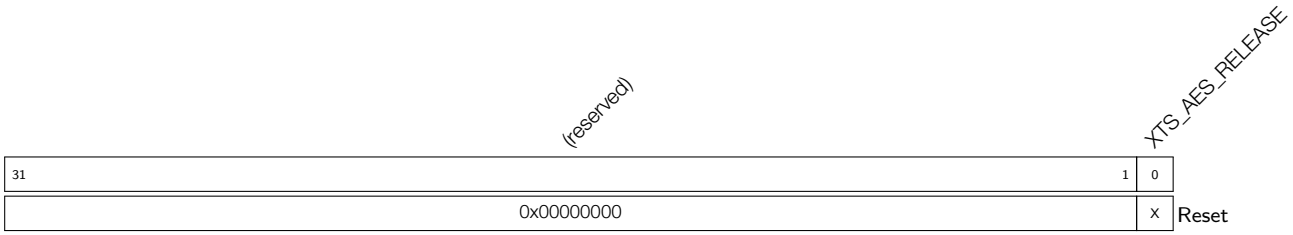
XTS_AES_TRIGGER 配置是否使能手动加密运算。

0：关闭手动加密运算

1：使能手动加密运算

(WO)

Register 24.7. XTS_AES_RELEASE_REG (0x0350)



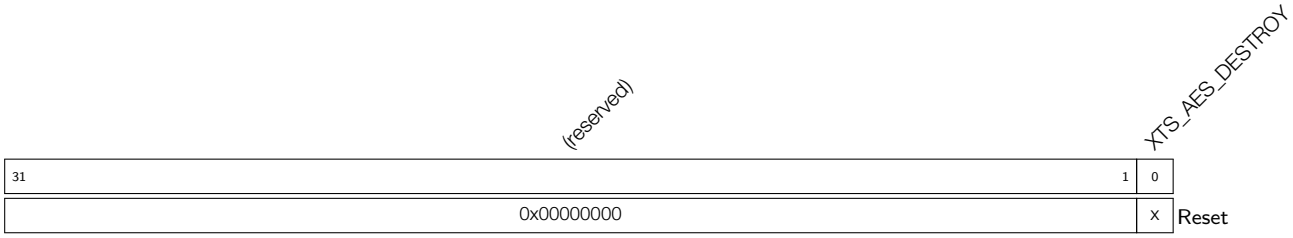
XTS_AES_RELEASE 配置是否使 SPI1 获取密文访问权限。

0：无效

1：使 SPI1 获取密文访问权限

(WO)

Register 24.8. XTS_AES_DESTROY_REG (0x0354)



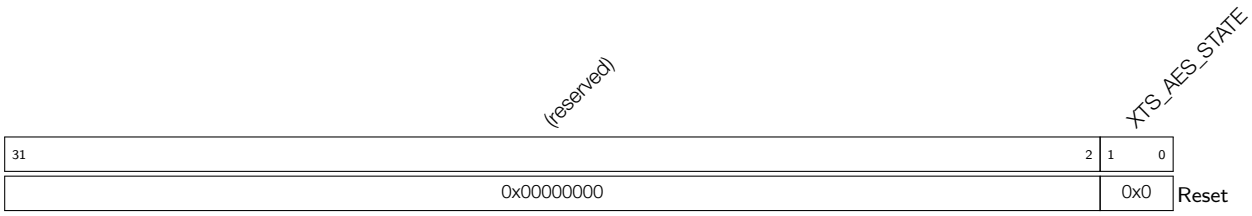
XTS_AES_DESTROY 配置是否销毁加密结果。

0：无效

1：销毁加密结果

(WO)

Register 24.9. XTS_AES_STATE_REG (0x0358)



XTS_AES_STATE 表示手动加密模块状态。

0 (XTS_AES_IDLE)：空闲

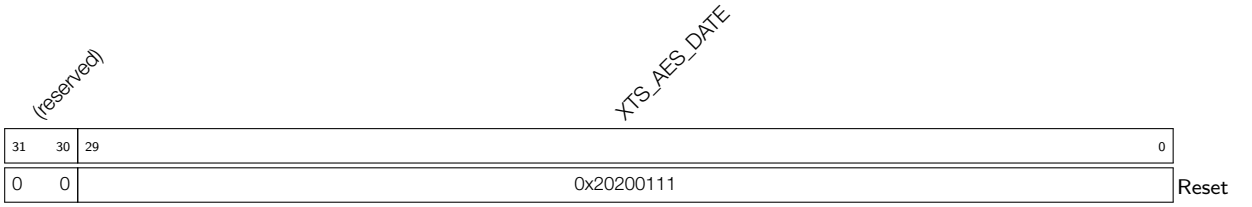
1 (XTS_AES_BUSY)：计算中

2 (XTS_AES_DONE)：计算完成，但手动加密结果数据对 SPI 不可见

3 (XTS_AES_RELEASE)：手动加密结果对 SPI 可见

(RO)

Register 24.10. XTS_AES_DATE_REG (0x035C)



XTS_AES_DATE 版本控制寄存器。(R/W)

25 UART 控制器 (UART)

25.1 概述

嵌入式应用通常要求一个简单的并且占用系统资源少的方法来传输数据。通用异步收发传输器 (UART) 即可以满足这些要求，它能够灵活地与外部设备进行全双工数据交换。ESP32-H2 芯片中共有两个 UART 控制器。UART 控制器可以兼容不同的 UART 设备。另外，UART 还可以用于红外数据交换 (IrDA) 或 RS485 调制解调器。

两个 UART 控制器分别有一组功能相同的寄存器。本文以 UART*n* 指代两个 UART 控制器，*n* 为 0、1。

UART 是一种以字符为导向的通用数据链，可以实现设备间的通信。异步传输的意思是不需要在发送数据上添加时钟信息。这也要求发送端和接收端的速率、停止位、奇偶校验位等都要相同，通信才能成功。

一个典型的 UART 帧开始于一个起始位，紧接着是有效数据，然后是奇偶校验位（可有可无），最后是停止位。ESP32-H2 上的 UART 控制器支持多种字符长度和停止位。另外，控制器还支持软硬件流控和 GDMA，可以实现高速的数据传输。开发者可以使用多个 UART 端口，同时又能保证很少的软件开销。

25.2 主要特性

UART 控制器具有如下特性：

- 可编程收发波特率，最大为 5 MBaud
- 260 x 8 位 RAM，由 UART 的发送 FIFO 和接收 FIFO 共用
- 全双工异步通信
- 数据位（5 到 8 位）
- 停止位（1、1.5 或 2 位）
- 奇偶校验位
- AT_CMD 特殊字符检测
- RS485 协议
- IrDA 协议
- GDMA 高速数据通信
- 接收超时
- UART 唤醒模式
- 软件流控和硬件流控
- 三个可预分频的时钟源：PLL_F48M_CLK，XTAL_CLK，RC_FAST_CLK

25.3 UART 架构

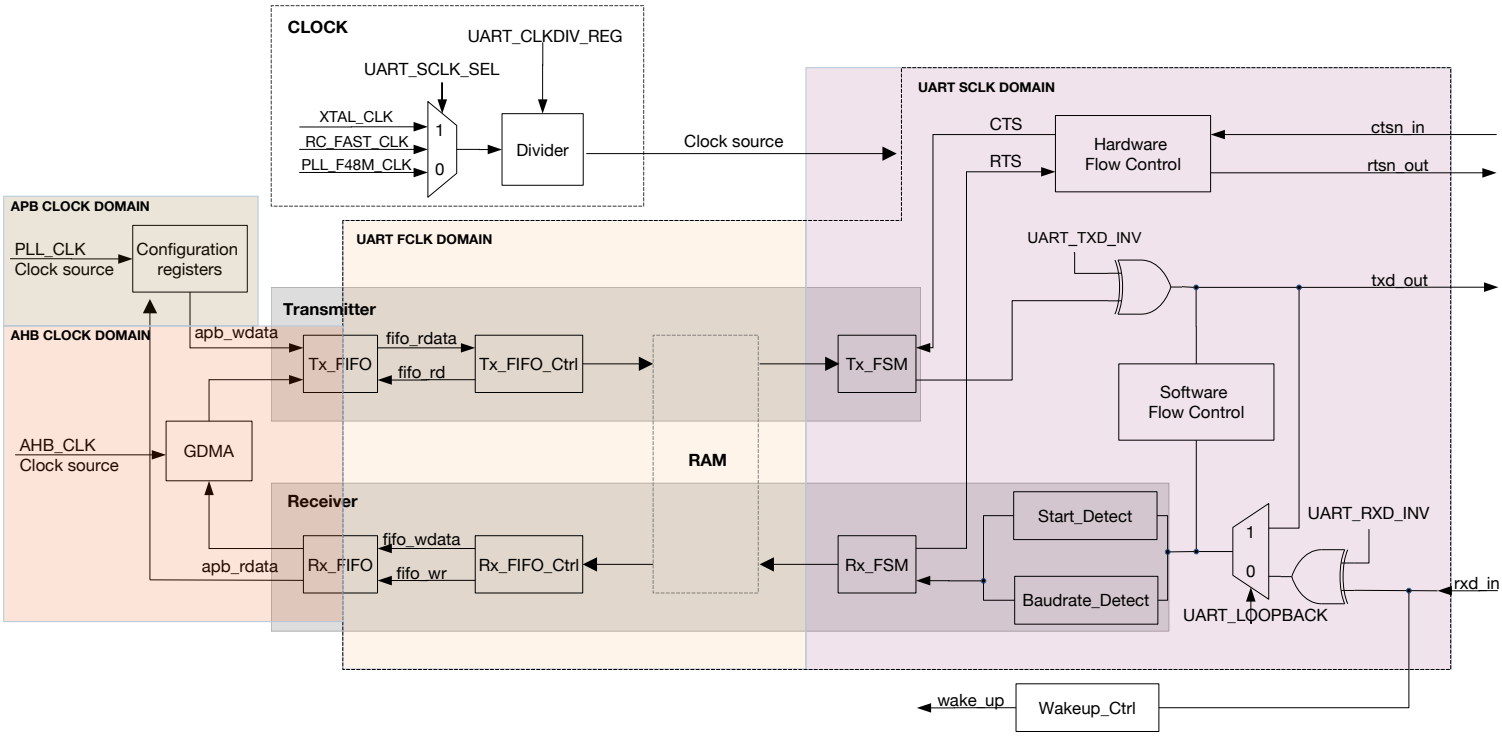


图 25-1. UART 基本架构图

图 25-1 为 UART 基本架构图。UART 模块工作在 4 个时钟域，分别为 APB_CLK，AHB_CLK，UART_SCLK，UART_FCLK。其中 APB_CLK 与 AHB_CLK 时钟关系为同步不同频（APB_CLK 由 AHB_CLK 分频得到），UART_SCLK 与 UART_FCLK 时钟关系为同步不同频（UART_SCLK 由 UART_FCLK 分频得到）。UART_FCLK 有三个时钟源：48 MHz PLL_F48M_CLK、RC_FAST_CLK 以及晶振时钟 XTAL_CLK （详情请参考章节 7 复位和时钟）。可以通过配置 PCR_UART*n*_SCLK_SEL 来选择时钟源，之后使用分频器对时钟源进行分频，然后产生时钟信号 UART_SCLK。分频系数的整数部分由 PCR_UART*n*_SCLK_DIV_NUM 配置，小数部分的分母由 PCR_UART*n*_SCLK_DIV_A 配置，小数部分的分子由 PCR_UART*n*_SCLK_DIV_B 配置。支持的分频范围为：1 ~ 256。

UART 控制器可以分为两个功能块：发送块和接收块。

发送块包含一个发送 FIFO 用于缓存待发送的数据。软件可以通过 APB 总线向 Tx_FIFO 写数据，也可以通过 GDMA 将数据搬入 Tx_FIFO。Tx_FIFO_Ctrl 用于控制 Tx_FIFO 的读写过程，当 Tx_FIFO 非空时，Tx_FSM 通过 Tx_FIFO_Ctrl 读取数据，并将数据按照配置的帧格式转化成比特流。比特流输出信号 txd_out 可以通过配置 UART_TXD_INV 寄存器实现取反功能。

接收块包含一个接收 FIFO 用于缓存待处理的数据。输入比特流 rxd_in 可以输入到 UART 控制器。可以通过 UART_RXD_INV 寄存器实现取反。Baudrate_Detect 通过检测最小比特流输入信号的脉宽来测量输入信号的波特率。Start_Detect 用于检测数据的起始位，当检测到起始位之后，Rx_FSM 通过 Rx_FIFO_Ctrl 将帧解析后的数

据存入 Rx_FIFO 中。软件可以通过 APB 总线读取 Rx_FIFO 中的数据，也可以使用 GDMA 进行数据接收。

HW_Flow_Ctrl 通过标准 UART RTS 和 CTS（rtsn_out 和 ctsn_in）流控信号来控制 rxd_in 和 txd_out 的数据流。SW_Flow_Ctrl 通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来进行数据流的控制。当 UART 处于 Light-sleep 模式（详情请参考章节 2 *低功耗管理 (RTC_CNTL) [to be added later]*）时，可以通过四种不同的方式产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒 ESP32-H2 芯片。具体唤醒方式请参考章节 25.4.8。

25.4 功能描述

25.4.1 时钟与复位

UART 为异步外设。其寄存器配置模块工作在 APB_CLK 时钟域，TX/RX FIFO 作为跨时钟域模块工作在 AHB_CLK 和 UART_FCLK 时钟域。UART RAM 控制单元工作在 UART_FCLK 时钟域，控制 UART 发送与接收的模块工作在 UART_SCLK 时钟域，即 UART 模块的 Core 时钟域。

如果 UART_SCLK 时钟频率满足生成通信波特率的需求，可通过预分频使 UART Core 模块工作在较小的时钟频率，从而减小 UART 外设的功耗。UART Core 模块时钟小于 PLL_CLK 时钟，并且在满足 UART 波特率的情况下，UART Core 时钟分频系数可以配置到最大值。另外，UART TX/RX 的 Core 时钟可以被单独控制。置位 UART_TX_SCLK_EN 使能 UART TX 的 Core 时钟；置位 UART_RX_SCLK_EN 使能 UART RX 的 Core 时钟。

为确保配置寄存器的值成功从 APB_CLK 时钟域同步到 UART Core 时钟域，寄存器配置需要遵循一定的流程，详情请参考章节 25.5。

对整个 UART 的复位，需要遵循如下配置流程：

- 将 PCR_UART*n*_CLK_EN 置 1 打开 UART*n* Core 时钟。
- 向 PCR_UART*n*_RST_EN 位写 1。
- 将 PCR_UART*n*_RST_EN 位清 0。

25.4.2 UART FIFO

ESP32-H2 芯片中两个 UART 控制器分别使用了一块 256×8-bit RAM 存储空间。UART 控制器通过一个 4×8-bit 的异步 FIFO 接口访问 RAM，且访问地址固定不变。即 UART 控制器的总存储空间相当于一块 260×8-bit 的 FIFO。

UART0 和 UART1 的 Tx_FIFO 可以通过置位 UART_TXFIFO_RST 来复位，UART0 和 UART1 的 Rx_FIFO 可以通过置位 UART_RXFIFO_RST 来复位。

对于 TX FIFO，可以通过 APB 总线或 GDMA 向其写入数据，硬件 Tx_FSM 自动从其中读取数据，数据将按照配置的帧格式转换成比特流；对于 RX FIFO，可以通过 APB 总线或 GDMA 读取其中的数据，并存储到内存，硬件 Rx_FSM 将接收到的比特流转换成字节并写入 RX FIFO。两个 UART 共享同一个 GDMA 通道。

配置 UART_TXFIFO_EMPTY_THRHD 可以设置 Tx_FIFO 空信号阈值，当存储在 Tx_FIFO 中的数据量小于 UART_TXFIFO_EMPTY_THRHD 时会产生中断 UART_TXFIFO_EMPTY_INT；配置 UART_RXFIFO_FULL_THRHD 可以设置 Rx_FIFO 满信号阈值，当储存在 Rx_FIFO 中的数据量大于 UART_RXFIFO_FULL_THRHD 会产生中断 UART_RXFIFO_FULL_INT。另外，当 Rx_FIFO 中储存的数据量超过其能存储的最大值时，会产生 UART_RXFIFO_OVF_INT 中断。

UART*n* 可以通过寄存器 UART_FIFO_REG 访问 FIFO。您可以写 UART_RXFIFO_RD_BYTE 将数据存入 TX FIFO，也可以读 UART_RXFIFO_RD_BYTE 获取 RX FIFO 中的数据。

25.4.3 波特率产生与检测

25.4.3.1 波特率产生

在 UART 发送或接收数据之前，需要配置寄存器来设置波特率。波特率发生器主要通过对输入时钟源的分频来实现，支持小数分频。UART_CLKDIV_SYNC_REG 将分频系数分成两个部分：UART_CLKDIV 用于配置整数部分，UART_CLKDIV_FRAG 用于配置小数部分。在输入时钟为 80 MHz 的情况下，UART 能支持的最大波特率为 5 MBaud。

波特率分频器系数为

$$UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}$$

也就是说，最终波特率为

$$\frac{INPUT\_FREQ}{UART\_CLKDIV + \frac{UART\_CLKDIV\_FRAG}{16}}$$

其中，INPUT_FREQ 为 UART Core 时钟。例如，若 UART_CLKDIV = 694，UART_CLKDIV_FRAG = 7，则分频系数为

$$694 + \frac{7}{16} = 694.4375$$

UART_CLKDIV_FRAG 为 0 时，分频器为整数分频，每 UART_CLKDIV 个输入脉冲都会产生一个输出脉冲。

UART_CLKDIV_FRAG 不为 0 时，分频器为小数分频，输出波特率脉冲不完全统一。如图 25-2 所示，每 16 个输出脉冲，波特率发生器分频 (UART_CLKDIV + 1) 个输入脉冲或 UART_CLKDIV 个输入脉冲。分频 (UART_CLKDIV + 1) 个输入脉冲产生 UART_CLKDIV_FRAG 个输出脉冲，分频 UART_CLKDIV 个输入脉冲产生剩余的 (16 - UART_CLKDIV_FRAG) 个输出脉冲。

如图 25-2 所示，输出脉冲相互交错，使得输出时序更加统一。

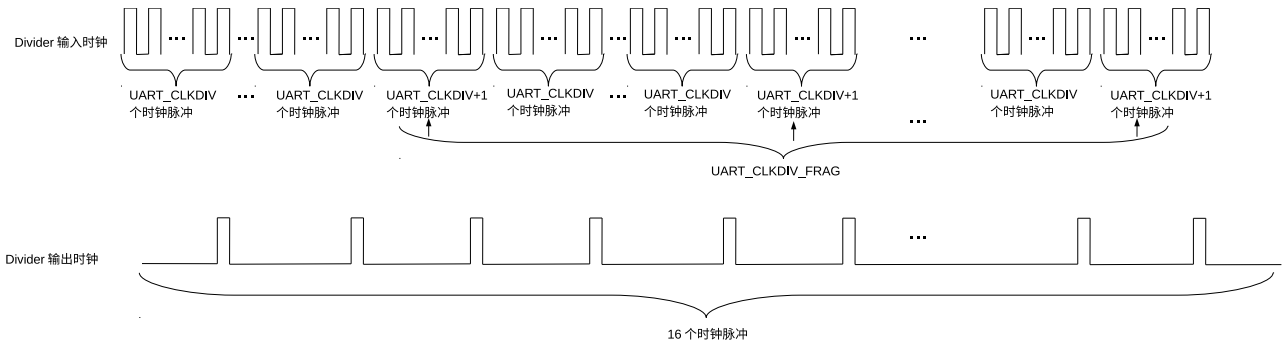


图 25-2. UART 控制器分频

为了支持 IrDA（详情见章节 25.4.7），IrDA 小数分频器会产生 16 × UART_CLKDIV_SYNC_REG 分频的时钟用于 IrDA 数据传输。产生 IrDA 数据传输时钟的小数分频器原理与上述小数分频器一样，取 UART_CLKDIV/16 作为分频值的整数部分，取 UART_CLKDIV 的低 4 比特作为小数部分。

25.4.3.2 波特率检测

置位 UART_AUTOBAUD_EN 可以开启 UART 波特率自检测功能。图 25-3 中的 Baudrate_Detect 可以滤除信号脉宽小于 UART_GLITCH_FILT 的噪声。

在 UART 双方进行通信之前，可以通过发送随机数据，让具有波特率检测功能的数据接收方进行波特率分析。UART_LOWPULSE_MIN_CNT 存储了最小低电平脉冲宽度，UART_HIGHPULSE_MIN_CNT 存储了最小高电平

脉冲宽度，UART_POSEDGE_MIN_CNT 存储了两个上升沿之间的最小脉冲宽度，UART_NEGEDGE_MIN_CNT 存储了两个下降沿之间最小的脉冲宽度。软件可以通过读取这四个寄存器获取发送方的波特率。

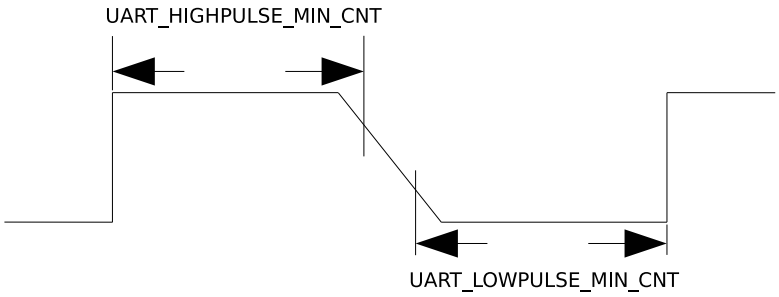


图 25-3. UART 信号下降沿较差时序图

波特率的计算分为三种情况：

1. 正常情况下，为防止因亚稳态在上升沿或下降沿附近采样数据错误而导致 UART_LOWPULSE_MIN_CNT 或者 UART_HIGHPULSE_MIN_CNT 不准确，单比特脉冲宽度可以通过将这两个值相加取平均消除误差。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

2. 对于 UART 信号的下降沿信号比较差的情况，如图25-3所示，这时通过取 UART_LOWPULSE_MIN_CNT 与 UART_HIGHPULSE_MIN_CNT 的和平均得到的值不准确，可以通过 UART_POSEDGE_MIN_CNT 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_POSEDGE\_MIN\_CNT} + 1)/2}$$

3. 对于 UART 信号的上升沿信号比较差的情况，可以通过 UART_NEGEDGE_MIN_CNT 获取发送方波特率。计算公式如下：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

25.4.4 UART 数据帧

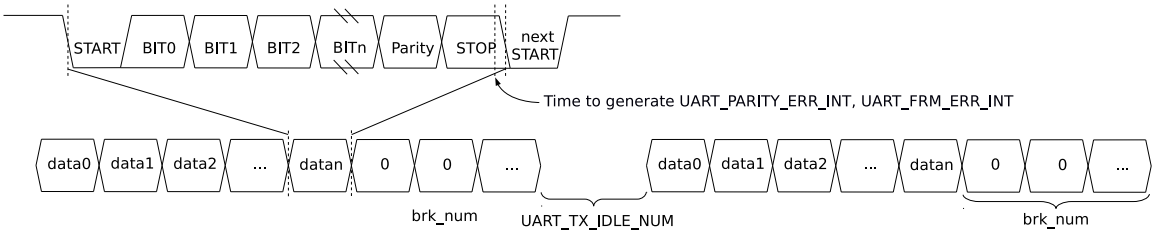


图 25-4. UART 数据帧结构

图 25-4 所示为基本数据帧格式，数据帧从起始位开始以停止位结束。起始位占用 1 位，停止位可以通过配置 UART_STOP_BIT_NUM 实现 1、1.5、2 位宽（RS485 模式下可增加转换延时，详见章节 25.4.6.2）。起始位为低电平，停止位为高电平。

数据位宽 (BIT0 ~ BITn) 为 5 ~ 8 位，可以通过 UART_BIT_NUM 进行配置。当置位 UART_PARITY_EN 时，数据帧会在数据之后添加一位奇偶校验位。UART_PARITY 用于选择奇校验或是偶校验。当接收器检测到输入数据

的校验位错误时会产生 UART_PARITY_ERR_INT 中断，输入数据仍会存入 Rx_FIFO。当接收器检测到数据数据帧格式错误时会产生 UART_FRM_ERR_INT 中断，默认情况下，输入数据会被存入 Rx_FIFO。

Tx_FIFO 中数据都发送完成后会产生 UART_TX_DONE_INT 中断。置位 UART_TXD_BRK 时，Tx_FIFO 中数据发送完成后，发送端会进入终止状态 (break condition)，继续发送几个连续的特殊数据帧 NULL。在 NULL 数据帧，TX 数据线输出为低电平。NULL 数据帧的数量可由 UART_TX_BRK_NUM 进行配置。发送器发送完所有的 NULL 数据帧之后会产生 UART_TX_BRK_DONE_INT 中断。数据帧之间可以通过配置 UART_TX_IDLE_NUM 保持最小间隔时间。当一帧数据之后的空闲时间大于等于 UART_TX_IDLE_NUM 寄存器的配置值时则产生 UART_TX_BRK_IDLE_DONE_INT 中断。

在传输一个 NULL 数据帧所需的时间内，接收端 RX 数据线若一直检测到低电平，接收端会检测为终止状态，并触发 UART_BRK_DET_INT 中断表示终止状态已结束。

接收端通过 UART_RXFIFO_TOUT_INT 中断检测总线状态。接收端接收到至少一个字节数据后，总线处于空闲状态超过 UART_RX_TOUT_THRHD 位时间时，触发 UART_RXFIFO_TOUT_INT 中断。您可用此中断检测发送端是否已经发送所有数据。

25.4.5 AT_CMD 字符格式

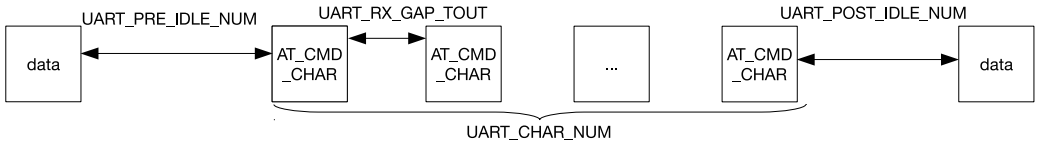


图 25-5. AT_CMD 字符格式

图 25-5 为一种特殊的 AT_CMD 字符格式。当接收器连续收到 AT_CMD_CHAR 字符且字符之间满足如下条件时将会产生 UART_AT_CMD_CHAR_DET_INT 中断。

- 接收到的第一个 AT_CMD_CHAR 与上一个非 AT_CMD_CHAR 之间间隔至少 UART_PRE_IDLE_NUM 个波特率周期。
- AT_CMD_CHAR 字符之间间隔小于 UART_RX_GAP_TOUT 个波特率周期。
- 接收的 AT_CMD_CHAR 字符个数必须大于等于 UART_CHAR_NUM。
- 接收到的最后一个 AT_CMD_CHAR 字符与下一个非 AT_CMD_CHAR 之间间隔至少 UART_POST_IDLE_NUM 个波特率周期。

注意：由于 AT_CMD_CHAR 字符之间间隔小于 UART_RX_GAP_TOUT 个波特率周期的限制，建议用户使用该功能时，配置的 PLL 时钟频率不要低于 8 MHz。

25.4.6 RS485

两个 UART 控制器支持 RS485 通讯模式，RS485 因使用差分信号传输数据，相比于 RS232 具有更远的传输距离及更高的传输速率。RS485 有两线半双工及四线全双工两种选择，UART 模块采用两线半双工模式，并支持侦听总线的功能。

25.4.6.1 驱动控制

如图 25-6 所示，RS485 两线 multidrop 系统中，需要一个外部 RS485 传输器实现单端信号与差分信号的转换。RS485 传输器包括一个驱动器与一个接收器。当 UART 不作为发送器时，通过关闭驱动器来断开与差分传输线

的连接。DE 为 1 时，使能驱动器；DE 为 0 关闭驱动器。

UART 接收端通过外部接收器将差分信号转为单端信号。RE 作为接收器的使能控制信号，RE 为 0，使能接收器；RE 为 1，关闭接收器。如果 RE 被配置为 0，从而允许 UART 保持侦听总线上的数据，包括 UART 发送的数据。

DE 信号的控制分为软件控制和硬件控制两种方法。为减少软件的开销，DE 信号采用硬件来控制。图 25-6 所示，DE 与 UART 的 dtrn_out 相连（详见 25.4.9.1 小节）。

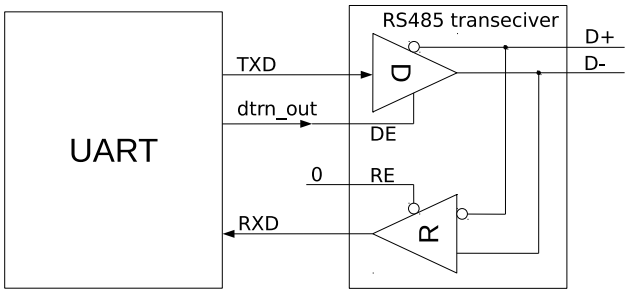


图 25-6. RS485 模式驱动控制结构图

25.4.6.2 转换延时

默认情况下，UART 处于接收状态。当从发送转为接收状态时，为保证发送数据被稳定接收，RS485 协议要求在发送停止位之后增加一个波特率的转换延时。UART 发送模块支持在起始位之前或在停止位之后增加一个波特率的延时。置位 UART_DL0_EN，在起始位之前增加一个波特率周期延时；置位 UART_DL1_EN，在停止位之后增加一个波特率周期延时。

25.4.6.3 总线侦听

RS485 两线 multidrop 系统中，当外部 RS485 传输器的 RE 被配置为 0 时，UART 支持侦听总线。默认情况下，不允许 UART 在发送数据时接收数据。置位 UART_RS485TX_RX_EN，允许在发送数据时接收数据，配合图 25-6 中外部 RS485 传输器的配置，UART 保持侦听传输总线。另外，默认情况下，不允许 UART 在接收数据时发送数据。置位 UART_RS485RXBY_TX_EN，允许在接收数据时发送数据。

UART 支持侦听 UART 发送的数据。UART 处于发送状态下，当侦听到 UART 发送的数据与 UART 接收的数据冲突时，触发 UART_RS485_CLASH_INT 中断；侦听到发送的数据帧错误时，触发 UART_RS485_FRM_ERR_INT 中断；侦听到发送数据极性错误时，触发 UART_RS485_PARITY_ERR_INT 中断。

25.4.7 IrDA

IrDA 数据协议由物理层，链路接入层和链路管理层三个基本层协议组成。UART 实现了其物理层协议。在 IrDA 编码模式下，支持最大信号速率到 115.2 Kbit/s，即 SIR 模式。如图 25-7 所示，IrDA 编码器将来自 UART 的非归零编码（NRZ）信号采用反向归零编码（RZI）并输出给外部驱动和红外 LED，用 3/16 Bit Time 的脉宽调制信号表示逻辑 “0”，用低电平表示逻辑 “1”。IrDA 解码器接收来自红外接收器的信号并输出为 UART 的 NRZ 编码。一般情况下，接收端信号空闲时为高电平，编码器输出极性与解码器输入极性相反。当检测到低脉冲表示接收到开始信号。

IrDA 使能时，一个位被划分为 16 个时钟周期，在其第 9、10、11 个时钟周期中，当需要发送的比特为 0 时，IrDA 输出为高。

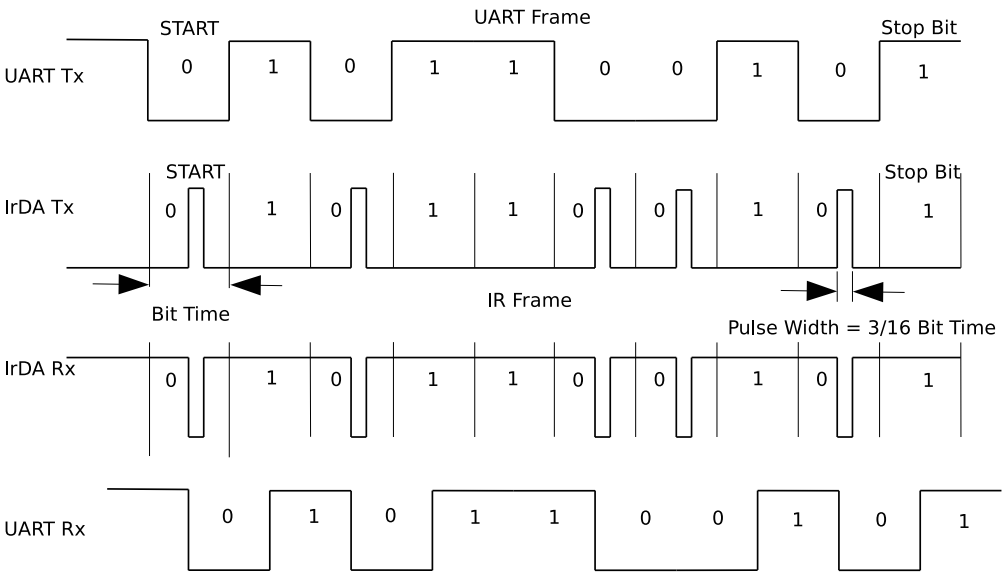


图 25-7. SIR 模式编解码时序图

IrDA 为半双工传输协议，不允许同时进行收发。如图 25-8 所示，置位 UART_IRDA_EN 使能 IrDA 功能。置位 UART_IRDA_TX_EN（置 1）使能 IrDA 发送数据，这时不允许 IrDA 接收数据；复位 UART_IRDA_TX_EN（清 0）使能 IrDA 接收数据，这时不允许 IrDA 发送数据。

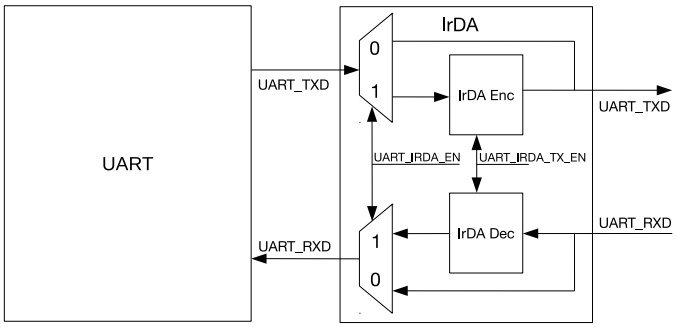


图 25-8. IrDA 编解码结构图

25.4.8 唤醒

UART 支持唤醒功能。当 UART 处于 Light-sleep 模式时，可以通过四种不同的方式产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒芯片。

- UART_WK_MODE_SEL=0，当所有的时钟都关闭时，此时可以通过使 RXD 翻转若干周期，当上升沿个数大于 UART_ACTIVE_THRESHOLD 时唤醒芯片。
- UART_WK_MODE_SEL=1，由于 UART Core 时钟保持工作，因此 UART RX 仍然可以接收数据并将数据暂存在 Rx FIFO 中。当 Rx FIFO 中的数据量超过 UART_RX_WAKE_UP_THRHD 时，可以将芯片从 light sleep 中唤醒。
- UART_WK_MODE_SEL=2，当 UART RX 监测到起始位后，唤醒芯片。
- UART_WK_MODE_SEL=3，当 UART RX 接收到特定字符序列后，唤醒芯片。用户通过配置 UART_WK_CHAR0、UART_WK_CHAR1、UART_WK_CHAR2、UART_WK_CHAR3、UART_WK_CHAR4 自定义唤醒字符。UART 监测到由 CHAR0/CHAR1/CHAR2/CHAR3/CHAR4 组成的字符序列流之后唤醒

芯片。可以通过配置 UART_CHAR_NUM 和 UART_WK_CHAR_MASK 来设置不同的字符序列。如下表所示，对于最后一个配置，UART 将依次检测 CHAR0 ~ CHAR4。

表 25-1. UART_CHAR_WAKEUP 模式配置

UART_CHAR_NAME	UART_WP_CHAR_MASK	检测字符序列
1	0xF	CHAR4
2	0x7	CHAR3/CHAR4
3	0x3	CHAR2/CHAR3/CHAR4
4	0x1	CHAR1/CHAR2/CHAR3/CHAR4
5	0x0	CHAR0/CHAR1/CHAR2/CHAR3/CHAR4

25.4.9 流控

UART 控制器有两种数据流控方式：硬件流控和软件流控。硬件流控主要通过输出信号 rtsn_out 以及输入信号 ctsn_in 进行数据流控制。软件流控主要通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来实现数据流控功能。

25.4.9.1 硬件流控

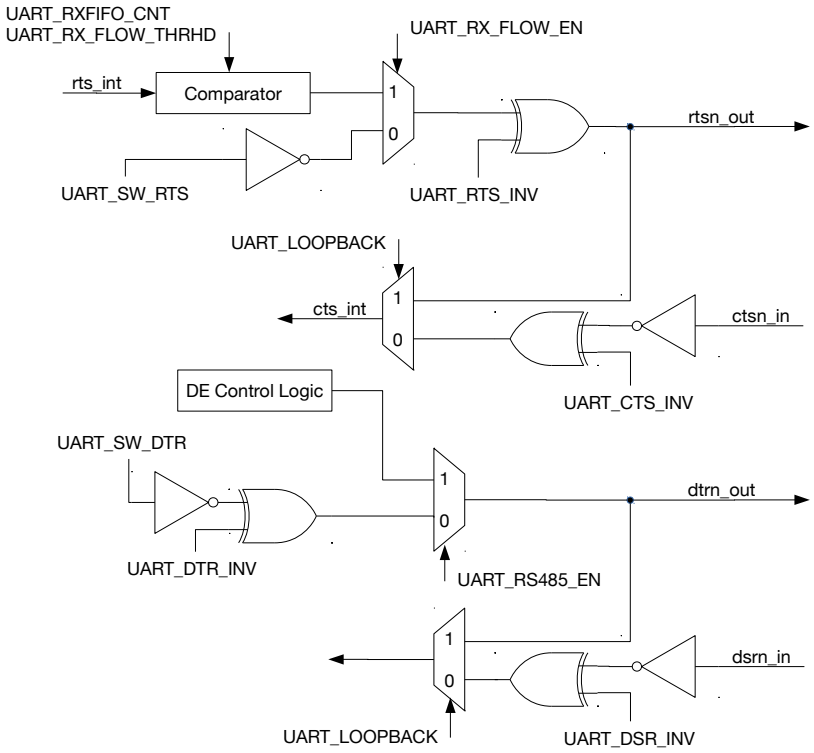


图 25-9. 硬件流控图

图 25-9 为 UART 硬件流控图。硬件流控的控制信号为输出信号 rtsn_out 及输入信号 ctsn_in。图 25-10 为两个 UART 之间硬件流控信号连接图。记 ESP32-H2 UART 为 IU0，External UART 为 EU0，下文将使用这两个标记来区分两个 UART。输出信号 rtsn_out （IU0）为低电平表示允许对方（EU0）发送数据，rtsn_out（IU0）为高电平表示通知对方（EU0）中止数据发送直到 rtsn_out （IU0）恢复低电平。rtsn_out 输出信号的控制有两种方

式。

- 软件控制：将 UART_RX_FLOW_EN 置 0 进入该模式。该模式下通过软件配置 UART_SW_RTS 改变 rtsn_out 的电平。
- 硬件控制：将 UART_RX_FLOW_EN 置 1 进入该模式。该模式下硬件会当 Rx_FIFO 中的数据大于 UART_RX_FLOW_THRHD 时拉高 rtsn_out 的电平。

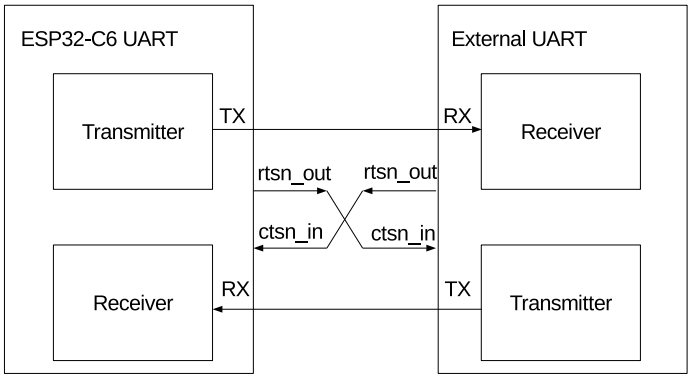


图 25-10. 硬件流控信号连接图

输入信号 ctsn_in（IU0）为低电平表示允许发送端（IU0）发送数据；ctsn_in（IU0）为高电平表示禁止发送端（IU0）发送数据。当 UART 检测到输入信号 ctsn_in （IU0）的沿变化时会产生 UART_CTS_CHG_INT 中断。

UART 发送设备（IU0）输出信号 dtrn_out 为高电平表示发送数据已经准备完毕，处于可用状态。dtrn_out 通过配置寄存器 UART_SW_DTR 产生。UART 接收设备（IU0）在检测到输入信号 dsrn_in 的沿变化时会产生 UART_DSR_CHG_INT 中断。软件在检测到中断后，通过读取 UART_DSRN 可以获取 dsrn_in 的输入信号电平，UART_DSRN 为高电平时，表示对方设备（EU0）处于可用状态。

对于 RS485 两线 multidrop 系统，使用 dtrn_out 来收发转换。置位 UART_RS485_EN 使能 RS485 功能，dtrn_out 由硬件产生。数据开始发送时，dtrn_out 拉高，使能外部驱动器；数据最后一位发送完成后，dtrn_out 拉低，关闭外部驱动器。注意，当使能停止位之后增加一个波特率延时时，dtrn_out 会在延时结束后才拉低。

置位 UART_LOOPBACK 即开启 UART 的回环测试功能。此时 UART 的输出信号 txd_out 和其输入信号 rxd_in 相连，rtsn_out 和 ctsn_in 相连，dtrn_out 和 dsrn_out 相连。当接收的数据与发送的数据相同时表明 UART 能够正常发送和接收数据。

25.4.9.2 软件流控

软件流控不使用硬件的 CTS/RTS 控制线，而是在发送数据流中嵌入 XON/XOFF 字符来通知对方是否可以使用数据发送来实现流控。将 UART_SW_FLOW_CON_EN 置 1 使能软件流控。

在使用软件流控后，硬件会自动检测接收数据流中是否有 XON/XOFF 字符，在检测到相应的字符后会产生
UART_SW_XOFF_INT 或 UART_SW_XON_INT 中断。在检测到接收数据流中有 XOFF 字符后，发送器将会在发送完当前数据后停止发送；在检测到接收数据流中有 XON 字符后，将会使能发送器发送数据。另外，软件可以

通过置位 UART_FORCE_XOFF 来强制发送器停止发送数据，发送器会在发送完当前字节后停止发送；也可以通过置位 UART_FORCE_XON 来使能发送器发送数据。

软件可以根据 Rx_FIFO 中剩余空间大小决定流控字符的发送。置位 UART_SEND_XOFF，发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 UART_XOFF_CHAR 配置；置位 UART_SEND_XON，发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 UART_XON_CHAR 配置。另外，当 UART 接收 FIFO 中的数据量超过 UART_XOFF_THRESHOLD 时，硬件会置位 UART_SEND_XOFF，UART 发送器会在发送完当前数据之后插入一个 XOFF 字符，该字符通过寄存器 UART_XOFF_CHAR 配置。当 UART 接收 FIFO 中的数据量小于 UART_XON_THRESHOLD 时，硬件会置位 UART_SEND_XON，UART 发送器会在发送完当前数据之后插入一个 XON 字符，该字符通过寄存器 UART_XON_CHAR 配置。

当 UART 工作在全双工模式时，如果 UART RX 接收到 XOFF，此时尽管 UART RX 接受的数据超过阈值，UART TX 将被禁止发送包括 XOFF 在内的任何数据。为了避免该种情况导致的软流控死锁或者 overflow，可以使能UART__XON_XOFF_STILL_SEND，此时允许 UART TX 在被禁止发送时，仍然可以发送一个 XOFF 字符。

25.4.10 GDMA 模式

ESP32-H2 中的两个 UART 接口通过通用主机控制器接口 (UHCI) 共用 1 组 GDMA TX/RX 通道。在 GDMA 模式下，支持对 HCI 协议数据包的解析 (decoder) 及数据包封装 (encoder) 。UHCI_UART*n*_CE 字段用于选择哪个串口占用 GDMA 通道。

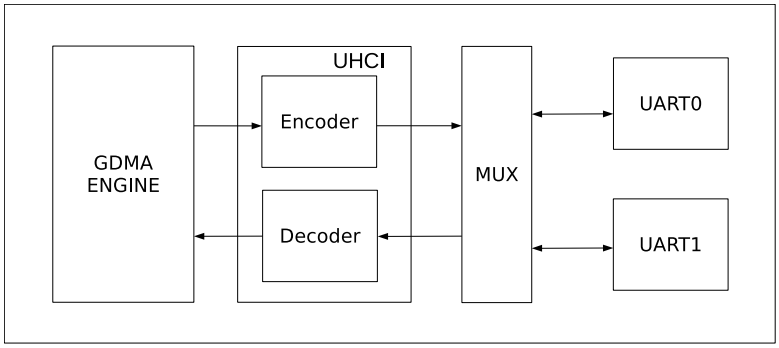


图 25-11. GDMA 模式数据传输

图 25-11 为 GDMA 方式数据传输图。在 GDMA Rx 通道接收数据前，软件将接收链表准备好。GDMA_INLINK_ADDR_CH*n* 用于指向第一个接收链表描述符。置位 GDMA_INLINK_START_CH*n* 之后，通用主机控制器接口 (UHCI) 会将 UART 接收到的数据传送给 Decoder。经过 Decoder 解析之后的数据在 GDMA 通道的控制下存入接收链表指定的 RAM 空间。

在 GDMA Tx 通道发送数据前，软件需要将发送链表和发送数据准备好，GDMA_OUTLINK_ADDR_CH*n* 用于指向第一个发送链表描述符。置位 GDMA_OUTLINK_START_CH*n* 之后，GDMA 引擎即从链表中指定的 RAM 地址读取数据，并通过 Encoder 进行数据包封装，然后经 UART 的发送模块串行发送出去。

HCI 的数据包格式为（分隔符 + 数据 + 分隔符）。Encoder 用于在数据前后加上分隔符，并将数据中和分隔符一样的数据替换为特殊字符。Decoder 用于去除数据包前后分隔符，并将数据中的特殊字符替换为分隔符。数据前后的分隔符可以有连续多个。分隔符可由 UHCI_SEPER_CHAR 进行配置，默认值为 0xC0。数据中与分隔符一样的数据可以用 UHCI_ESC_SEQ0_CHAR0（默认为 0xDB）和 UHCI_ESC_SEQ0_CHAR1（默认为 0xDD）进行替换。当数据全部发送完成后，会产生 GDMA_OUT_TOTAL_EOF_CH*n*_INT 中断。当数据接收完成后，会产生 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

25.4.11 UART 中断

- UART_AT_CMD_CHAR_DET_INT：当接收器检测到 AT_CMD 字符时触发此中断。
- UART_RS485_CLASH_INT：在 RS485 模式下检测到发送器和接收器之间的冲突时触发此中断。
- UART_RS485_FRM_ERR_INT：在 RS485 模式下检测到发送块发送的数据帧错误时触发此中断。
- UART_RS485_PARITY_ERR_INT：在 RS485 模式下检测到发送块发送的数据校验位错误时触发此中断。
- UART_TX_DONE_INT：当发送器发送完 FIFO 中的所有数据时触发此中断。
- UART_TX_BRK_IDLE_DONE_INT：当发送器在最后一个数据发送后保持了最短的间隔时间时触发此中断。
- UART_TX_BRK_DONE_INT：当发送 FIFO 中的数据发送完之后发送器完成了发送 NULL 则触发此中断。
- UART_GLITCH_DET_INT：当接收器在起始位的中点处检测到毛刺时触发此中断。
- UART_SW_XOFF_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XOFF 字符时触发此中断。
- UART_SW_XON_INT：UART_SW_FLOW_CON_EN 置位时，当接收器接收到 XON 字符时触发此中断。
- UART_RXFIFO_TOUT_INT：当接收器至少已经接收一个字节的数据，且之后在UART_RX_TOUT_THRHD位的时间内，总线一直处于 idle 状态将触发此中断。
- UART_BRK_DET_INT：当接收器在停止位之后检测到一个 NULL （即传输一个 NULL 的时间内保持逻辑低电平）时触发此中断。
- UART_CTS_CHG_INT：当接收器检测到 CTSn 信号的沿变化时触发此中断。
- UART_DSR_CHG_INT：当接收器检测到 DSRn 信号的沿变化时触发此中断。
- UART_RXFIFO_OVF_INT：当接收器接收到的数据量多于 FIFO 的存储量时触发此中断。
- UART_FRM_ERR_INT：当接收器检测到数据帧错误时触发此中断。
- UART_PARITY_ERR_INT：当接收器检测到校验位错误时触发此中断。
- UART_TXFIFO_EMPTY_INT：当发送 FIFO 中的数据量少于 UART_TXFIFO_EMPTY_THRHD 所指定的值时触发此中断。
- UART_RXFIFO_FULL_INT：当接收器接收到的数据多于 UART_RXFIFO_FULL_THRHD 所指定的值时触发此中断。
- UART_WAKEUP_INT：UART 被唤醒时产生此中断。

25.4.12 UHCI 中断

- UHCI_APP_CTRL1_INT：软件置位 UHCI_APP_CTRL1_INT_RAW 时触发此中断。
- UHCI_APP_CTRL0_INT：软件置位 UHCI_APP_CTRL0_INT_RAW 时触发此中断。
- UHCI_OUTLINK_EOF_ERR_INT：当检测到发送链表描述符中的 EOF 有错误时触发此中断。
- UHCI_SEND_A_REG_Q_INT：当使用 always_send 发送一串短包，UHCI 发送了短包后触发此中断。
- UHCI_SEND_S_REG_Q_INT：当使用 single_send 发送一串短包，UHCI 发送了短包后触发此中断。
- UHCI_TX_HUNG_INT：当 UHCI 利用 GDMA Tx 通道从 RAM 中读取数据的时间过长时触发此中断。
- UHCI_RX_HUNG_INT：当 UHCI 利用 GDMA Rx 通道接收数据的时间过长时触发此中断。
- UHCI_TX_START_INT：当检测到分隔符时触发此中断。

- UHCI_RX_START_INT：当分隔符已发送时触发此中断。

25.5 编程流程

25.5.1 寄存器类型

UART 的所有寄存器都处于 APB_CLK 时钟域。

UART 上的配置寄存器分为两类：第一类寄存器作用在 APB/AHB 时钟域，该类寄存器配置后不需要额外的操作；第二类寄存器作用在 UART Core 时钟域，因此需要进行跨时钟域处理。在配置完该类寄存器后，需要通过写寄存器 UART_REG_UPDATE 使配置的值同步到 Core 时钟域。当同步完成后，硬件会将 UART_REG_UPDATE 自动清零。推荐用户在配置需要同步的寄存器之后，检查 UART_REG_UPDATE 是否为 0，确保上一次的同步操作完成之后再执行接下来的寄存器配置。

为了方便用户区分这两类寄存器，章节 25.6 中将需要做跨时钟域操作的寄存器放在一起，该类寄存器名称附带 _SYNC 后缀。不需要做跨时钟域操作的寄存器放在一起，该类寄存器名称不带 _SYNC 后缀。

25.5.2 具体步骤

图25-12 显示了 UART 模块的编程流程。主要包括：初始化、寄存器配置、启动 UART TX/RX 和数据传输结束。

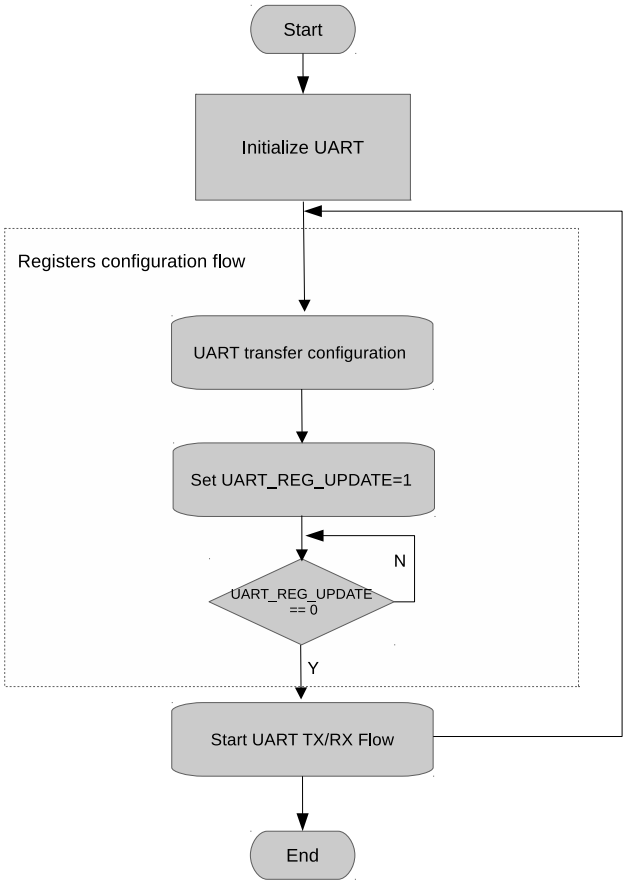


图 25-12. UART 编程流程

25.5.2.1 UART*n* 模块初始化

UART*n* 模块初始化流程如下：

- PCR_UART*n*_RST_EN 位写 1；
- PCR_UART*n*_RST_EN 位清 0。

25.5.2.2 UART*n* 通信配置

UART*n* 通信配置流程如下：

- 等待 UART_REG_UPDATE 为 0，确保上一次同步已经完成；
- 配置 PCR_UART*n*_SCLK_SEL 选择时钟源；
- 配置 PCR_UART*n*_SCLK_DIV_NUM、PCR_UART*n*_SCLK_DIV_A、PCR_UART*n*_SCLK_DIV_B 设置预分频器系数；
- 配置 UART_CLKDIV、UART_CLKDIV_FRAG 设置发送波特率；
- 配置 UART_BIT_NUM 设置数据长度；
- 配置 UART_PARITY_EN、UART_PARITY 设置奇偶校验；
- 可选步骤，根据应用不同存在差异...
- 向 UART_REG_UPDATE 写 1，将配置的值同步到 Core 时钟域。

25.5.2.3 启动 UART*n*

启动 UART*n* TX 发送数据：

- 配置 UART_TXFIFO_EMPTY_THRHD，设置 TX FIFO 空阈值；
- 对 UART_TXFIFO_EMPTY_INT_ENA 置 0，关闭 UART_TXFIFO_EMPTY_INT 中断；
- 向 UART_RXFIFO_RD_BYTE 写入需要发送的数据；
- 置位 UART_TXFIFO_EMPTY_INT_CLR，清除 UART_TXFIFO_EMPTY_INT 中断；
- 置位 UART_TXFIFO_EMPTY_INT_ENA，使能 UART_TXFIFO_EMPTY_INT 中断；
- 检测 UART_TXFIFO_EMPTY_INT_ST，等待发送数据结束。

启动 UART*n* RX 数据接收：

- 配置 UART_RXFIFO_FULL_THRHD，设置 RX FIFO 满阈值；
- 置位 UART_RXFIFO_FULL_INT_ENA，使能 UART_RXFIFO_FULL_INT 中断；
- 检测 UART_RXFIFO_FULL_INT_ST，等待 RX FIFO 接收数据满；
- 通过读 UART_RXFIFO_RD_BYTE，从 RX FIFO 中读出数据，并可通过 UART_RXFIFO_CNT 获得当前 RX FIFO 中的接收数据量。

25.6 寄存器列表

25.6.1 UART 寄存器列表

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

名称	描述	地址	访问
FIFO 配置			
UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
UART_TOUT_CONF_SYNC_REG	UART 阈值和分配配置	0x0064	R/W
UART 中断寄存器			
UART_INT_RAW_REG	原始中断状态	0x0004	R/WTC/SS
UART_INT_ST_REG	屏蔽中断状态	0x0008	RO
UART_INT_ENA_REG	中断使能位	0x000C	R/W
UART_INT_CLR_REG	中断清除位	0x0010	WT
配置寄存器			
UART_CLKDIV_SYNC_REG	时钟分频配置	0x0014	R/W
UART_RX_FILT_REG	RX 滤波器配置	0x0018	R/W
UART_CONF0_SYNC_REG	配置寄存器 0	0x0020	R/W
UART_CONF1_REG	配置寄存器 1	0x0024	R/W
UART_HWFC_CONF_SYNC_REG	硬件流控配置	0x002C	R/W
UART_SLEEP_CONF0_REG	UART 睡眠配置寄存器 0	0x0030	R/W
UART_SLEEP_CONF1_REG	UART 睡眠配置寄存器 1	0x0034	R/W
UART_SLEEP_CONF2_REG	UART 睡眠配置寄存器 2	0x0038	R/W
UART_SWFC_CONF0_SYNC_REG	软件流控字符配置	0x003C	varies
UART_SWFC_CONF1_REG	软件流控字符配置	0x0040	R/W
UART_TXBRK_CONF_SYNC_REG	TX 断开字符配置	0x0044	R/W
UART_IDLE_CONF_SYNC_REG	帧结束空闲配置	0x0048	R/W
UART_RS485_CONF_SYNC_REG	RS485 模式配置	0x004C	R/W
UART_CLK_CONF_REG	UART core 时钟配置	0x0088	R/W
UART_REG_UPDATE_REG	UART 寄存器配置更新	0x0098	R/W/SC
UART_ID_REG	UART ID 寄存器	0x009C	R/W
状态寄存器			
UART_STATUS_REG	UART 状态寄存器	0x001C	RO
UART_MEM_TX_STATUS_REG	TX FIFO 写入、读取偏移地址	0x0068	RO
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x006C	RO
UART_FSM_STATUS_REG	UART 发送和接收状态	0x0070	RO
UART_AFIFO_STATUS_REG	UART 异步 FIFO 状态	0x0090	RO
AT 转义序列检测配置			
UART_AT_CMD_PRECNT_SYNC_REG	序列发送前的时序配置	0x0050	R/W
UART_AT_CMD_POSTCNT_SYNC_REG	序列发送后的时序配置	0x0054	R/W
UART_AT_CMD_GAPTOUT_SYNC_REG	超时配置	0x0058	R/W
UART_AT_CMD_CHAR_SYNC_REG	AT 转义序列检测配置	0x005C	R/W

名称	描述	地址	访问
自动波特率检测寄存器			
UART_POSPULSE_REG	自动波特率检测高电平脉冲寄存器	0x0074	RO
UART_NEGPULSE_REG	自动波特率检测低电平脉冲寄存器	0x0078	RO
UART_LOWPULSE_REG	自动波特率检测最短低电平脉冲持续时间寄存器	0x007C	RO
UART_HIGHPULSE_REG	自动波特率检测最短高电平脉冲持续时间寄存器	0x0080	RO
UART_RXD_CNT_REG	自动波特率检测沿变化计数寄存器	0x0084	RO
版本寄存器			
UART_DATE_REG	UART 版本控制寄存器	0x008C	R/W

25.6.2 UHCI 寄存器列表

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

名称	描述	地址	访问
配置寄存器			
UHCI_CONF0_REG	UHCI 配置寄存器	0x0000	R/W
UHCI_CONF1_REG	UHCI 配置寄存器	0x0014	varies
UHCI_ESCAPE_CONF_REG	转义符配置	0x0020	R/W
UHCI_HUNG_CONF_REG	超时配置	0x0024	R/W
UHCI_ACK_NUM_REG	配置 UHCI ACK 值	0x0028	varies
UHCI_QUICK_SENT_REG	UHCI 快速发送配置寄存器	0x0030	varies
UHCI_REG_Q0_WORD0_REG	Q0 WORD0 快速发送寄存器	0x0034	R/W
UHCI_REG_Q0_WORD1_REG	Q0 WORD1 快速发送寄存器	0x0038	R/W
UHCI_REG_Q1_WORD0_REG	Q1 WORD0 快速发送寄存器	0x003C	R/W
UHCI_REG_Q1_WORD1_REG	Q1 WORD1 快速发送寄存器	0x0040	R/W
UHCI_REG_Q2_WORD0_REG	Q2 WORD0 快速发送寄存器	0x0044	R/W
UHCI_REG_Q2_WORD1_REG	Q2 WORD1 快速发送寄存器	0x0048	R/W
UHCI_REG_Q3_WORD0_REG	Q3 WORD0 快速发送寄存器	0x004C	R/W
UHCI_REG_Q3_WORD1_REG	Q3 WORD1 快速发送寄存器	0x0050	R/W
UHCI_REG_Q4_WORD0_REG	Q4 WORD0 快速发送寄存器	0x0054	R/W
UHCI_REG_Q4_WORD1_REG	Q4 WORD1 快速发送寄存器	0x0058	R/W
UHCI_REG_Q5_WORD0_REG	Q5 WORD0 快速发送寄存器	0x005C	R/W
UHCI_REG_Q5_WORD1_REG	Q5 WORD1 快速发送寄存器	0x0060	R/W
UHCI_REG_Q6_WORD0_REG	Q6 WORD0 快速发送寄存器	0x0064	R/W
UHCI_REG_Q6_WORD1_REG	Q6 WORD1 快速发送寄存器	0x0068	R/W
UHCI_ESC_CONF0_REG	转义序列配置寄存器 0	0x006C	R/W
UHCI_ESC_CONF1_REG	转义序列配置寄存器 1	0x0070	R/W
UHCI_ESC_CONF2_REG	转义序列配置寄存器 2	0x0074	R/W
UHCI_ESC_CONF3_REG	转义序列配置寄存器 3	0x0078	R/W
UHCI_PKT_THRES_REG	包长度配置寄存器	0x007C	R/W
UHCI 中断寄存器			

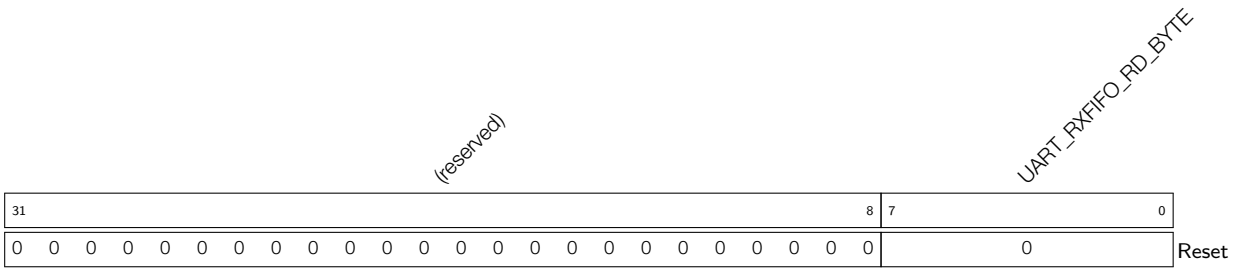
名称	描述	地址	访问
UHCI_INT_RAW_REG	原始中断状态	0x0004	varies
UHCI_INT_ST_REG	屏蔽中断状态	0x0008	RO
UHCI_INT_ENA_REG	中断使能位	0x000C	R/W
UHCI_INT_CLR_REG	中断清除位	0x0010	WT
UHCI 状态寄存器			
UHCI_STATE0_REG	UHCI 接收状态	0x0018	RO
UHCI_STATE1_REG	UHCI 发送状态	0x001C	RO
UHCI_RX_HEAD_REG	UHCI 包报头寄存器	0x002C	RO
版本寄存器			
UHCI_DATE_REG	UHCI 版本控制寄存器	0x0080	R/W

25.7 寄存器

25.7.1 UART 寄存器

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

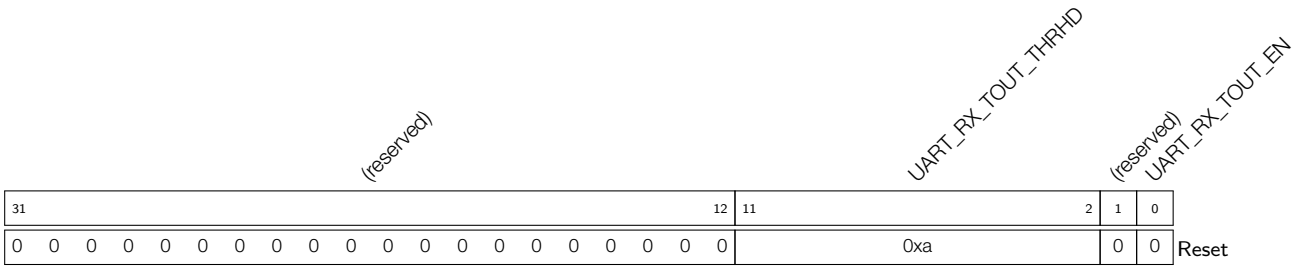
Register 25.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE 表示 UART *n* 从 FIFO 读取的数据。

单位：字节。(RO)

Register 25.2. UART_TOUT_CONF_SYNC_REG (0x0064)



UART_RX_TOUT_EN 配置是否开启 UART 接收器的超时功能。

0：关闭

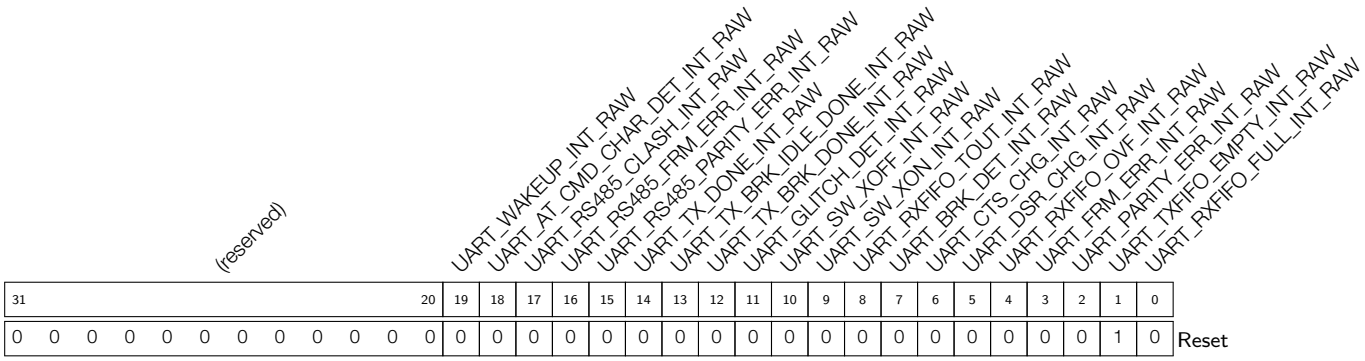
1：开启

(R/W)

UART_RX_TOUT_THRHD 配置触发超时前总线可保持空闲状态的时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 25.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW UART_RXFIFO_FULL_INT 的原始中断状态。(R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW UART_TXFIFO_EMPTY_INT 的原始中断状态。(R/WTC/SS)

UART_PARITY_ERR_INT_RAW UART_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_FRM_ERR_INT_RAW UART_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW UART_RXFIFO_OVF_INT 的原始中断状态。(R/WTC/SS)

UART_DSR_CHG_INT_RAW UART_DSR_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_CTS_CHG_INT_RAW UART_CTS_CHG_INT 的原始中断状态。(R/WTC/SS)

UART_BRK_DET_INT_RAW UART_BRK_DET_INT 的原始中断状态。(R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW UART_RXFIFO_TOUT_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XON_INT_RAW UART_SW_XON_INT 的原始中断状态。(R/WTC/SS)

UART_SW_XOFF_INT_RAW UART_SW_XOFF_INT 的原始中断状态。(R/WTC/SS)

UART_GLITCH_DET_INT_RAW UART_GLITCH_DET_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_DONE_INT_RAW UART_TX_BRK_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW UART_TX_BRK_IDLE_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_TX_DONE_INT_RAW UART_TX_DONE_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW UART_RS485_PARITY_ERR_INT 的原始中断状态。(R/WTC/SS)

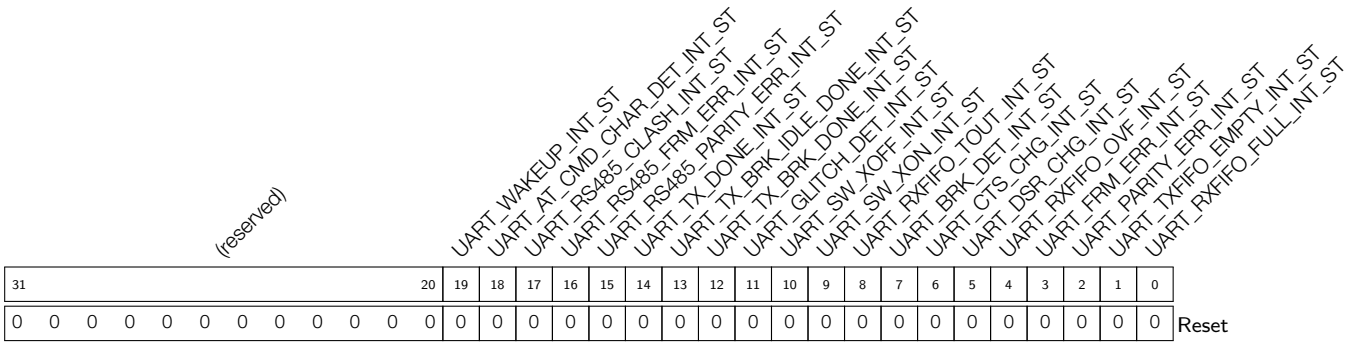
UART_RS485_FRM_ERR_INT_RAW UART_RS485_FRM_ERR_INT 的原始中断状态。(R/WTC/SS)

UART_RS485_CLASH_INT_RAW UART_RS485_CLASH_INT 的原始中断状态。(R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW UART_AT_CMD_CHAR_DET_INT 的原始中断状态。(R/WTC/SS)

UART_WAKEUP_INT_RAW UART_WAKEUP_INT 的原始中断状态。(R/WTC/SS)

Register 25.4. UART_INT_ST_REG (0x0008)



UART_RXFIFO_FULL_INT_ST UART_RXFIFO_FULL_INT 的屏蔽中断状态。(RO)

UART_TXFIFO_EMPTY_INT_ST UART_TXFIFO_EMPTY_INT 的屏蔽中断状态。(RO)

UART_PARITY_ERR_INT_ST UART_PARITY_ERR_INT 的屏蔽中断状态。(RO)

UART_FRM_ERR_INT_ST UART_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_OVF_INT_ST UART_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

UART_DSR_CHG_INT_ST UART_DSR_CHG_INT 的屏蔽中断状态。(RO)

UART_CTS_CHG_INT_ST UART_CTS_CHG_INT 的屏蔽中断状态。(RO)

UART_BRK_DET_INT_ST UART_BRK_DET_INT 的屏蔽中断状态。(RO)

UART_RXFIFO_TOUT_INT_ST UART_RXFIFO_TOUT_INT 的屏蔽中断状态。(RO)

UART_SW_XON_INT_ST UART_SW_XON_INT 的屏蔽中断状态。(RO)

UART_SW_XOFF_INT_ST UART_SW_XOFF_INT 的屏蔽中断状态。(RO)

UART_GLITCH_DET_INT_ST UART_GLITCH_DET_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_DONE_INT_ST UART_TX_BRK_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_BRK_IDLE_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT 的屏蔽中断状态。(RO)

UART_TX_DONE_INT_ST UART_TX_DONE_INT 的屏蔽中断状态。(RO)

UART_RS485_PARITY_ERR_INT_ST UART_RS485_PARITY_ERR_INT 的屏蔽中断状态。(RO)

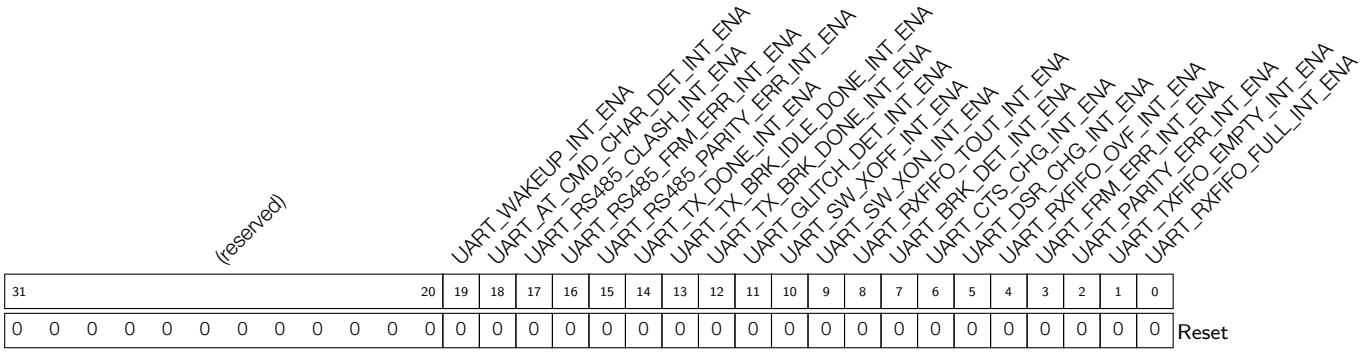
UART_RS485_FRM_ERR_INT_ST UART_RS485_FRM_ERR_INT 的屏蔽中断状态。(RO)

UART_RS485_CLASH_INT_ST UART_RS485_CLASH_INT 的屏蔽中断状态。(RO)

UART_AT_CMD_CHAR_DET_INT_ST UART_AT_CMD_CHAR_DET_INT 的屏蔽中断状态。(RO)

UART_WAKEUP_INT_ST UART_WAKEUP_INT 的屏蔽中断状态。(RO)

Register 25.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA 写 1 使能 UART_RXFIFO_FULL_INT。(R/W)

UART_TXFIFO_EMPTY_INT_ENA 写 1 使能 UART_TXFIFO_EMPTY_INT。(R/W)

UART_PARITY_ERR_INT_ENA 写 1 使能 UART_PARITY_ERR_INT。(R/W)

UART_FRM_ERR_INT_ENA 写 1 使能 UART_FRM_ERR_INT。(R/W)

UART_RXFIFO_OVF_INT_ENA 写 1 使能 UART_RXFIFO_OVF_INT。(R/W)

UART_DSR_CHG_INT_ENA 写 1 使能 UART_DSR_CHG_INT。(R/W)

UART_CTS_CHG_INT_ENA 写 1 使能 UART_CTS_CHG_INT。(R/W)

UART_BRK_DET_INT_ENA 写 1 使能 UART_BRK_DET_INT。(R/W)

UART_RXFIFO_TOUT_INT_ENA 写 1 使能 UART_RXFIFO_TOUT_INT。(R/W)

UART_SW_XON_INT_ENA 写 1 使能 UART_SW_XON_INT。(R/W)

UART_SW_XOFF_INT_ENA 写 1 使能 UART_SW_XOFF_INT。(R/W)

UART_GLITCH_DET_INT_ENA 写 1 使能 UART_GLITCH_DET_INT。(R/W)

UART_TX_BRK_DONE_INT_ENA 写 1 使能 UART_TX_BRK_DONE_INT。(R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA 写 1 使能 UART_TX_BRK_IDLE_DONE_INT。(R/W)

UART_TX_DONE_INT_ENA 写 1 使能 UART_TX_DONE_INT。(R/W)

UART_RS485_PARITY_ERR_INT_ENA 写 1 使能 UART_RS485_PARITY_ERR_INT。(R/W)

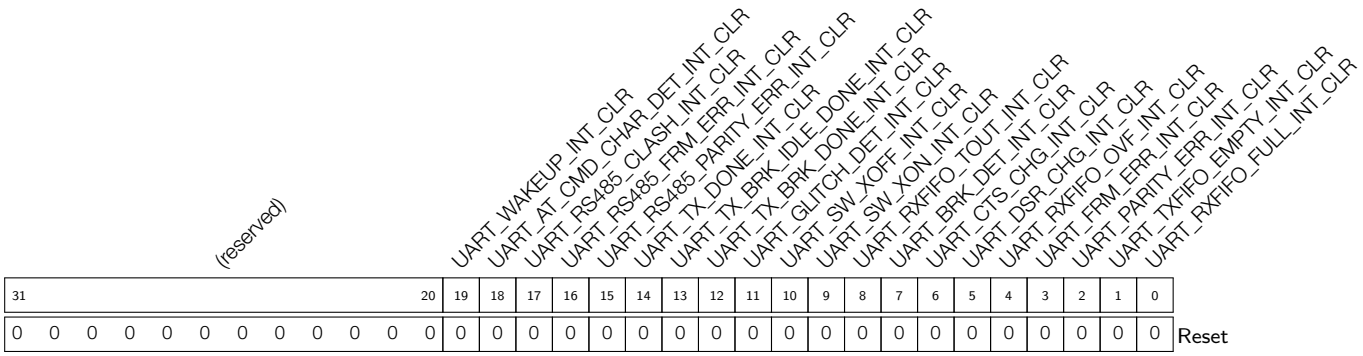
UART_RS485_FRM_ERR_INT_ENA 写 1 使能 UART_RS485_FRM_ERR_INT。(R/W)

UART_RS485_CLASH_INT_ENA 写 1 使能 UART_RS485_CLASH_INT。(R/W)

UART_AT_CMD_CHAR_DET_INT_ENA 写 1 使能 UART_AT_CMD_CHAR_DET_INT。(R/W)

UART_WAKEUP_INT_ENA 写 1 使能 UART_WAKEUP_INT。(R/W)

Register 25.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR 写 1 清除 UART_RXFIFO_FULL_INT。(WT)

UART_TXFIFO_EMPTY_INT_CLR 写 1 清除 UART_TXFIFO_EMPTY_INT。(WT)

UART_PARITY_ERR_INT_CLR 写 1 清除 UART_PARITY_ERR_INT。(WT)

UART_FRM_ERR_INT_CLR 写 1 清除 UART_FRM_ERR_INT。(WT)

UART_RXFIFO_OVF_INT_CLR 写 1 清除 UART_RXFIFO_OVF_INT。(WT)

UART_DSR_CHG_INT_CLR 写 1 清除 UART_DSR_CHG_INT。(WT)

UART_CTS_CHG_INT_CLR 写 1 清除 UART_CTS_CHG_INT。(WT)

UART_BRK_DET_INT_CLR 写 1 清除 UART_BRK_DET_INT。(WT)

UART_RXFIFO_TOUT_INT_CLR 写 1 清除 UART_RXFIFO_TOUT_INT。(WT)

UART_SW_XON_INT_CLR 写 1 清除 UART_SW_XON_INT。(WT)

UART_SW_XOFF_INT_CLR 写 1 清除 UART_SW_XOFF_INT。(WT)

UART_GLITCH_DET_INT_CLR 写 1 清除 UART_GLITCH_DET_INT。(WT)

UART_TX_BRK_DONE_INT_CLR 写 1 清除 UART_TX_BRK_DONE_INT。(WT)

UART_TX_BRK_IDLE_DONE_INT_CLR 写 1 清除 UART_TX_BRK_IDLE_DONE_INT。(WT)

UART_TX_DONE_INT_CLR 写 1 清除 UART_TX_DONE_INT。(WT)

UART_RS485_PARITY_ERR_INT_CLR 写 1 清除 UART_RS485_PARITY_ERR_INT。(WT)

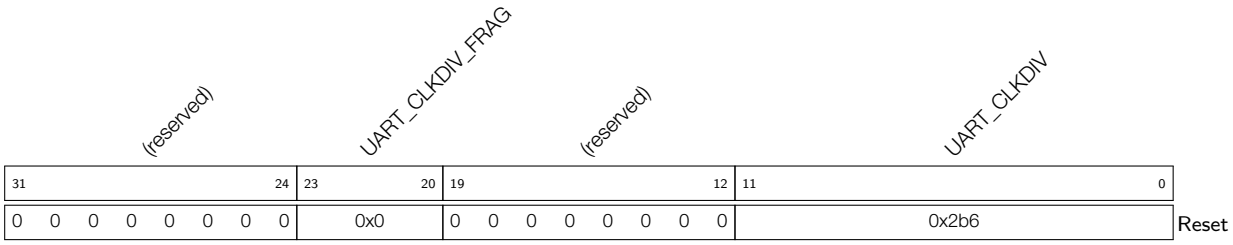
UART_RS485_FRM_ERR_INT_CLR 写 1 清除 UART_RS485_FRM_ERR_INT。(WT)

UART_RS485_CLASH_INT_CLR 写 1 清除 UART_RS485_CLASH_INT。(WT)

UART_AT_CMD_CHAR_DET_INT_CLR 写 1 清除 UART_AT_CMD_CHAR_DET_INT。(WT)

UART_WAKEUP_INT_CLR 写 1 清除 UART_WAKEUP_INT。(WT)

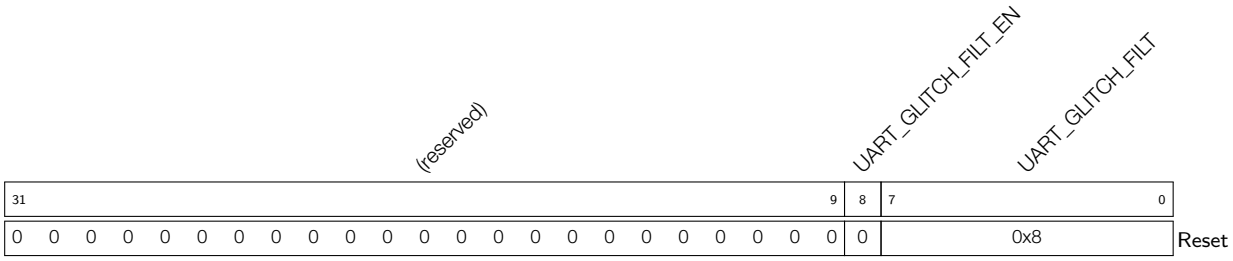
Register 25.7. UART_CLKDIV_SYNC_REG (0x0014)



UART_CLKDIV 配置波特率分频系数的整数部分。(R/W)

UART_CLKDIV_FRAG 配置波特率分频系数的小数部分。(R/W)

Register 25.8. UART_RX_FILT_REG (0x0018)



UART_GLITCH_FILT 配置滤波长度。

单位：UART Core 时钟周期。

宽度小于该数值的输入脉冲会被忽略。(R/W)

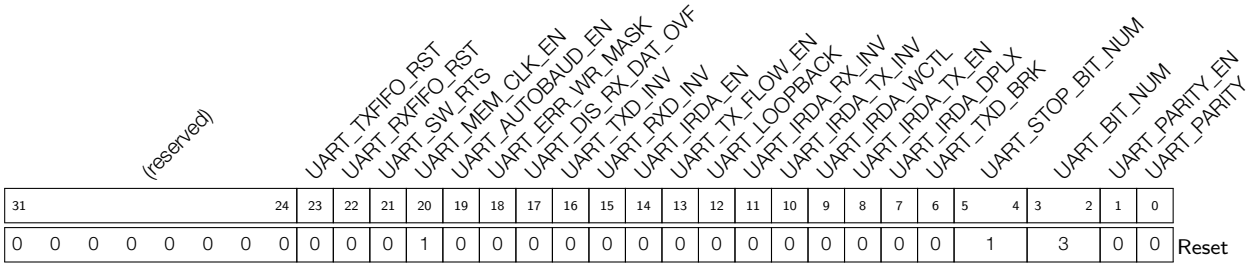
UART_GLITCH_FILT_EN 配置是否开启接收信号滤波器。

0：关闭

1：开启

(R/W)

Register 25.9. UART_CONF0_SYNC_REG (0x0020)



UART_PARITY 配置奇偶检验方式。

0：偶校验

1：奇校验

(R/W)

UART_PARITY_EN 配置是否开启 UART 奇偶校验。

0：关闭

1：开启

(R/W)

UART_BIT_NUM 配置数据位长度。

0：5 位

1：6 位

2：7 位

3：8 位

(R/W)

UART_STOP_BIT_NUM 配置停止位长度。

0: 无效值，没有作用

1: 1 位

2: 1.5 位

3: 2 位

(R/W)

UART_TXD_BRK 配置完成数据发送后，发送器是否发送 NULL。

0：不发送

1：发送

(R/W)

UART_IRDA_DPLX 配置是否开启 IrDA 回环测试模式。

0：关闭

1：开启

(R/W)

UART_IRDA_TX_EN 配置是否开启 IrDA 发送器。

0：关闭

1：开启

(R/W)

见下页...

Register 25.9. UART_CONF0_SYNC_REG (0x0020)

接上页...

UART_IRDA_WCTL 配置 IrDA 发送器的第 11 位。

0：该位为 0

1：该位与第 10 位相同

(R/W)

UART_IRDA_TX_INV 配置是否翻转 IrDA 发送器的电平。

0：不翻转

1：翻转

(R/W)

UART_IRDA_RX_INV 配置是否翻转 IrDA 接收器的电平。

0：不翻转

1：翻转

(R/W)

UART_LOOPBACK 配置是否开启 UART 回环测试模式。

0：关闭

1：开启

(R/W)

UART_TX_FLOW_EN 配置是否开启发送器的流控功能。

0：关闭

1：开启

(R/W)

UART_IRDA_EN 配置是否开启 IrDA。

0：关闭

1：开启

(R/W)

UART_RXD_INV 配置是否翻转 UART RXD 信号电平。

0：不翻转

1：翻转

(R/W)

UART_TXD_INV 配置是否翻转 UART TXD 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DIS_RX_DAT_OVF 配置是否关闭 UART 接收器数据溢出检测。

0：开启

1：关闭

(R/W)

见下页...

Register 25.9. UART_CONF0_SYNC_REG (0x0020)

接上页...

UART_ERR_WR_MASK 配置接收数据有错误时，是否仍将其存储至 FIFO。

0：存储

1：不存储

(R/W)

UART_AUTOBAUD_EN 配置是否开启波特率检测。

0：关闭

1：开启

(R/W)

UART_MEM_CLK_EN 配置是否开启 UART 存储器门控。

0：关闭

1：开启

(R/W)

UART_SW_RTS 配置软件流控使用的 RTS 信号。

0: 允许发送端发送数据

1: 禁止发送端发送数据

(R/W)

UART_RXFIFO_RST 配置是否复位 UART RX FIFO。

0：不复位

1：复位

(R/W)

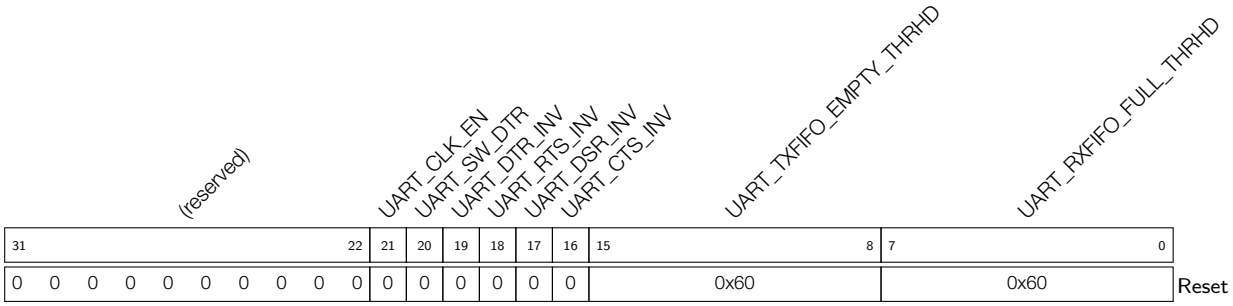
UART_TXFIFO_RST 配置是否复位 UART TX FIFO。

0：不复位

1：复位

(R/W)

Register 25.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD 配置 RX FIFO 为满的阈值。

单位：字节。(R/W)

UART_TXFIFO_EMPTY_THRHD 配置 TX FIFO 为空的阈值。

单位：字节。(R/W)

UART_CTS_INV 配置是否翻转 UART CTS 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DSR_INV 配置是否翻转 UART DSR 信号电平。

0：不翻转

1：翻转

(R/W)

UART_RTS_INV 配置是否翻转 UART RTS 信号电平。

0：不翻转

1：翻转

(R/W)

UART_DTR_INV 配置是否翻转 UART DTR 信号电平。

0：不翻转

1：翻转

(R/W)

UART_SW_DTR 配置软件流控使用的 DTR 信号。

0: 发送数据未准备完毕，处于不可用状态

1: 发送数据准备完毕，处于可用状态

(R/W)

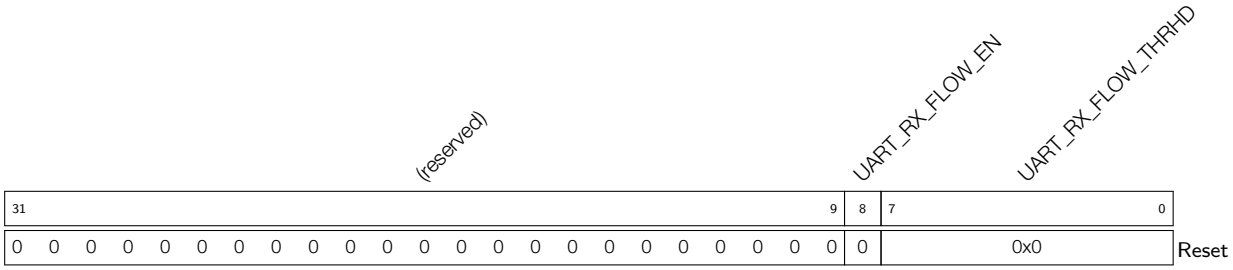
UART_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

Register 25.11. UART_HWFC_CONF_SYNC_REG (0x002C)



UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大数。

单位：字节。(R/W)

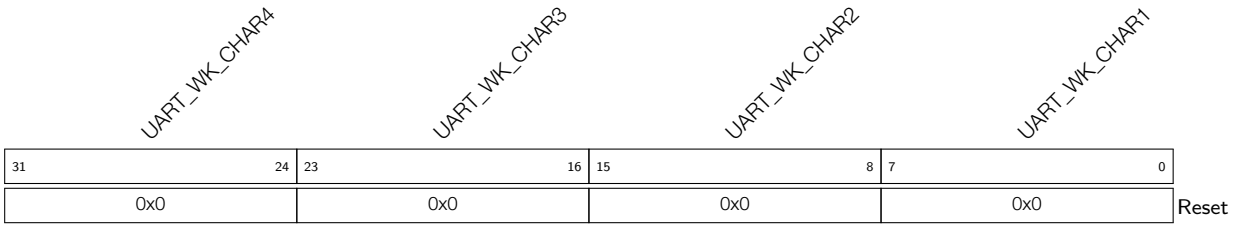
UART_RX_FLOW_EN 配置是否开启 UART 接收器。

0：关闭

1：开启

(R/W)

Register 25.12. UART_SLEEP_CONF0_REG (0x0030)



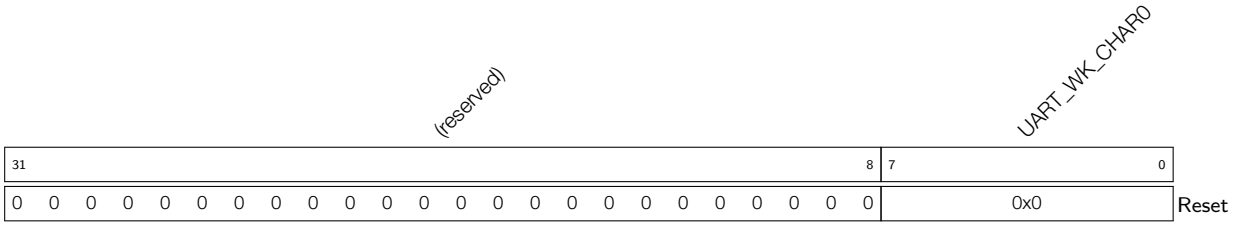
UART_WK_CHAR1 配置唤醒芯片的特定字符 1。(R/W)

UART_WK_CHAR2 配置唤醒芯片的特定字符 2。(R/W)

UART_WK_CHAR3 配置唤醒芯片的特定字符 3。(R/W)

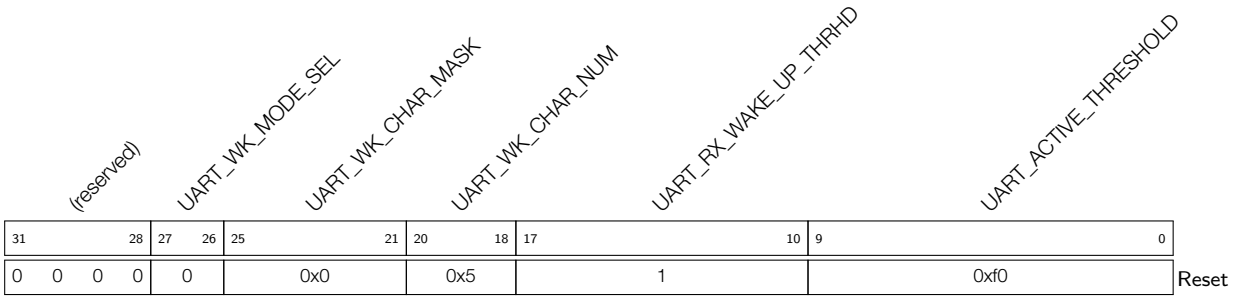
UART_WK_CHAR4 配置唤醒芯片的特定字符 4。(R/W)

Register 25.13. UART_SLEEP_CONF1_REG (0x0034)



UART_WK_CHAR0 配置唤醒芯片的特定字符 0。(R/W)

Register 25.14. UART_SLEEP_CONF2_REG (0x0038)



UART_ACTIVE_THRESHOLD 配置唤醒模式 0 唤醒芯片所需的 RXD 沿变化次数。(R/W)

UART_RX_WAKE_UP_THRHD 配置唤醒模式 1 唤醒芯片所需接收的数据量。

单位：字节。(R/W)

UART_WK_CHAR_NUM 配置选取多少个唤醒字符。(R/W)

UART_WK_CHAR_MASK 配置是否屏蔽唤醒字符。

0：不屏蔽

1：屏蔽

(R/W)

UART_WK_MODE_SEL 配置选取哪个唤醒模式。

0：唤醒模式 0

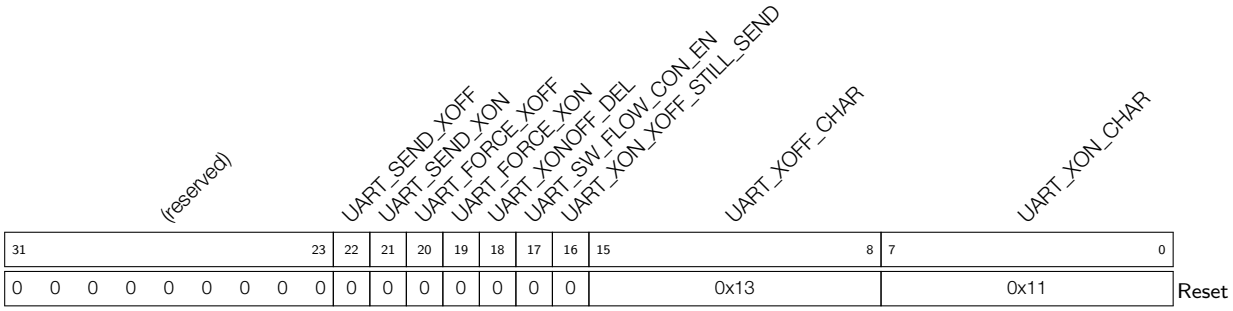
1：唤醒模式 1

2：唤醒模式 2

3：唤醒模式 3

(R/W)

Register 25.15. UART_SWFC_CONF0_SYNC_REG (0x003C)



UART_XON_CHAR 配置 XON 流控字符。(R/W)

UART_XOFF_CHAR 配置 XOFF 流控字符。(R/W)

UART_XON_XOFF_STILL_SEND 配置 UART 发送器关闭时，是否仍能发送 XON 或 XOFF 字符。

0：不能发送

1：可以发送

(R/W)

UART_SW_FLOW_CON_EN 配置是否开启软件流控。

0：关闭

1：开启

(R/W)

UART_XONOFF_DEL 配置是否移除接收数据中的流控字符。

0：不移除

1：移除

(R/W)

UART_FORCE_XON 配置是否让发送器继续发送数据。

0：不发送

1：继续发送

(R/W)

UART_FORCE_XOFF 配置是否让发送器停止发送数据。

0：不停止

1：停止

(R/W)

UART_SEND_XON 配置是否发送 XON 字符。

0：不发送

1：发送

(R/W/SS/SC)

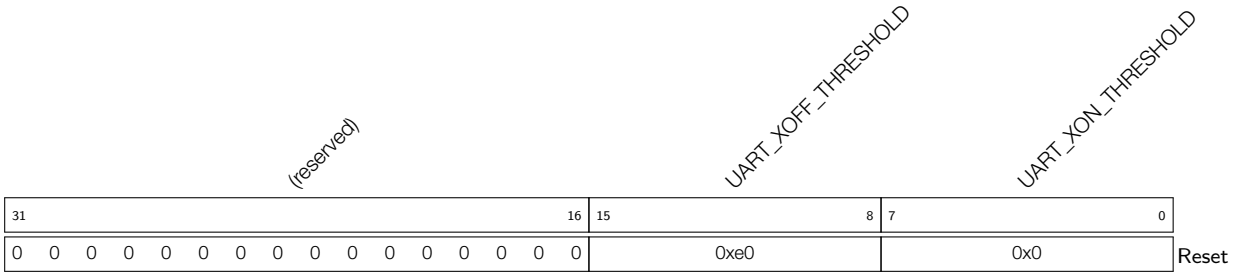
UART_SEND_XOFF 配置是否发送 XOFF 字符。

0：不发送

1：发送

(R/W/SS/SC)

Register 25.16. UART_SWFC_CONF1_REG (0x0040)



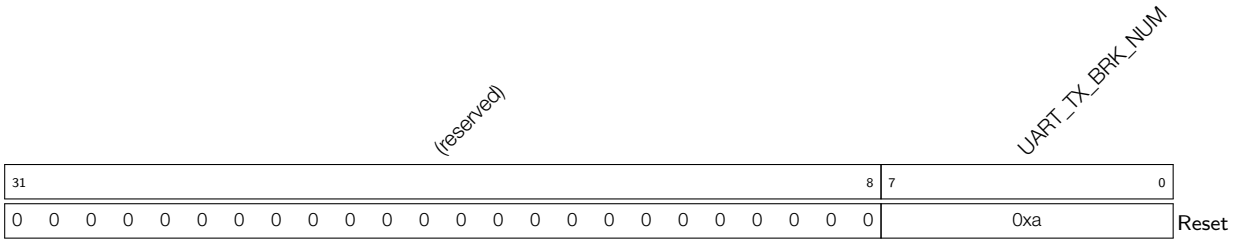
UART_XON_THRESHOLD 配置软件流控时，发送 XON 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

UART_XOFF_THRESHOLD 配置软件流控时，发送 XOFF 字符所需的 RX FIFO 数据阈值。

单位：字节。(R/W)

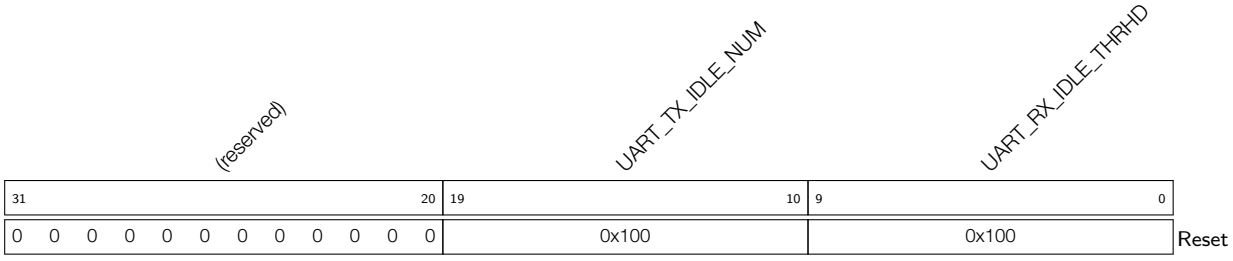
Register 25.17. UART_TXBRK_CONF_SYNC_REG (0x0044)



UART_TX_BRK_NUM 配置完成数据发送后待发 NULL 字符的数量。

仅在 UART_TXD_BRK 为 1 时有效。(R/W)

Register 25.18. UART_IDLE_CONF_SYNC_REG (0x0048)



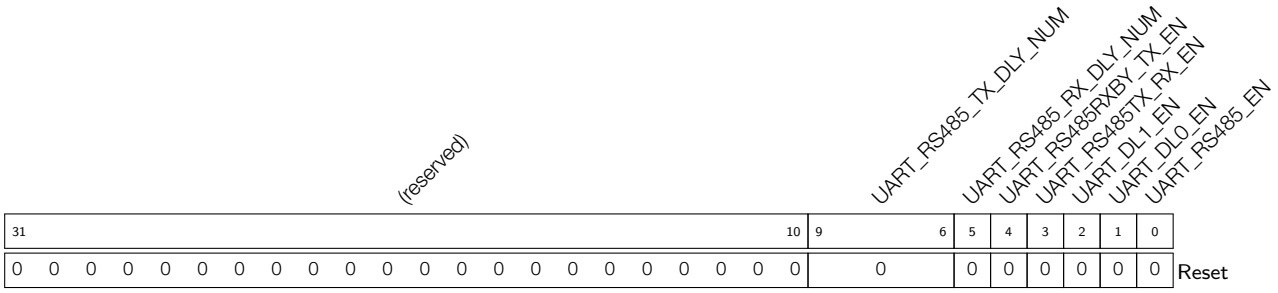
UART_RX_IDLE_THRHD 配置接收一字节数据时间过长、产生帧结束信号的阈值。

单位：位时间（传输一位所需的时间）。(R/W)

UART_TX_IDLE_NUM 配置两次数据传输的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

Register 25.19. UART_RS485_CONF_SYNC_REG (0x004C)



UART_RS485_EN 配置是否开启 RS485 模式。

0：关闭

1：开启

(R/W)

UART_DL0_EN 配置是否在起始位之前增加一位延时。

0：不增加

1：增加

(R/W)

UART_DL1_EN 配置是否在停止位之后增加一位延时。

0：不增加

1：增加

(R/W)

UART_RS485TX_RX_EN 配置发送器在 RS485 模式下发送数据时，是否开启接收器接收数据。

0：关闭

1：开启

(R/W)

UART_RS485RXBY_TX_EN 配置 RS484 接收器线路繁忙时，是否开启 RS485 发送器发送数据。

0：关闭

1：开启

(R/W)

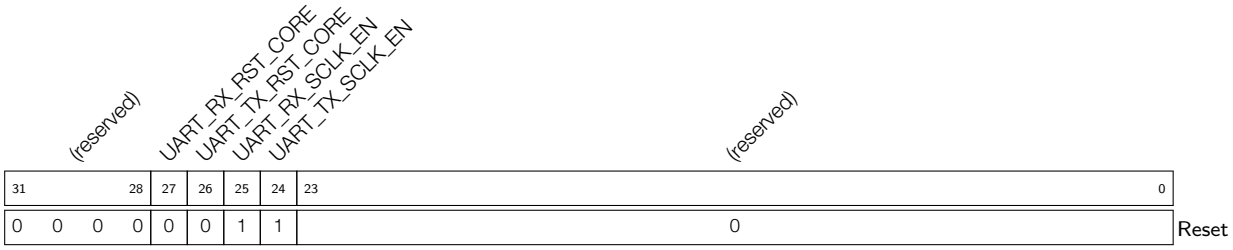
UART_RS485_RX_DLY_NUM 配置接收器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

UART_RS485_TX_DLY_NUM 配置发送器内部数据信号的延迟长度。

单位：位时间（传输一位所需的时间）。(R/W)

Register 25.20. UART_CLK_CONF_REG (0x0088)



UART_TX_SCLK_EN 配置是否开启 UART TX 时钟。

0：关闭

1：开启

(R/W)

UART_RX_SCLK_EN 配置是否开启 UART RX 时钟。

0：关闭

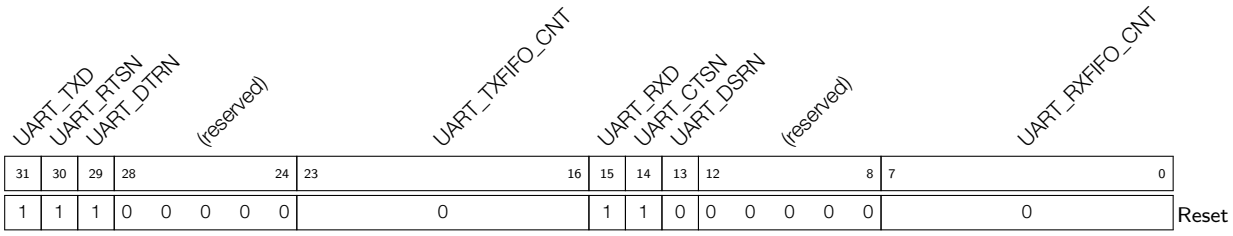
1：开启

(R/W)

UART_TX_RST_CORE 向此位先写 1 后写 0，复位 UART TX。(R/W)

UART_RX_RST_CORE 向此位先写 1 后写 0，复位 UART RX。(R/W)

Register 25.21. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT 表示 RX FIFO 中有效数据量。

单位：字节。(RO)

UART_DSRN 表示内部 UART DSR 信号的电平值。(RO)

UART_CTSN 表示内部 UART CTS 信号的电平值。(RO)

UART_RXD 表示内部 UART RXD 信号的电平值。(RO)

UART_TXFIFO_CNT 表示 TX FIFO 中的数据量。

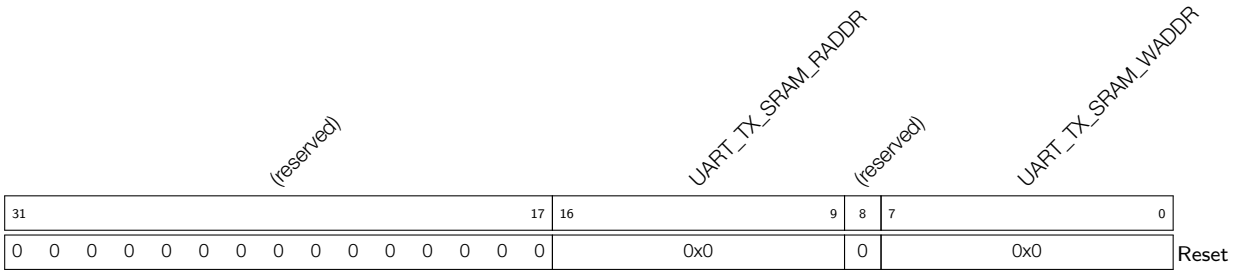
单位：字节。(RO)

UART_DTRN 表示内部 UART DTR 信号的电平。(RO)

UART_RTSN 表示内部 UART RTS 信号的电平。(RO)

UART_TXD 表示内部 UART TXD 信号的电平。(RO)

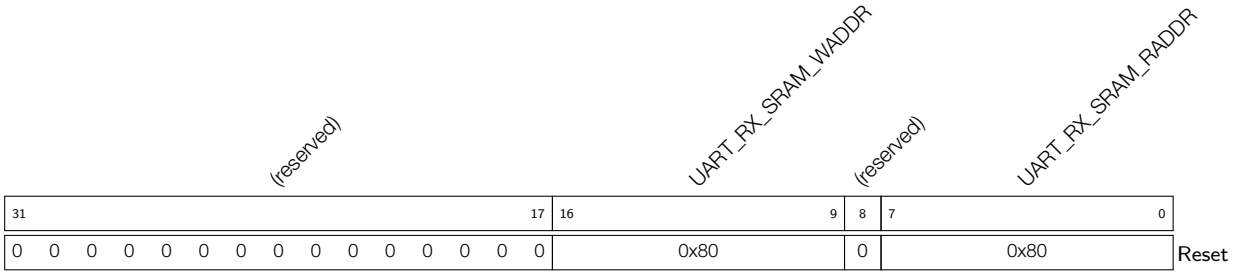
Register 25.22. UART_MEM_TX_STATUS_REG (0x0068)



UART_TX_SRAM_WADDR 表示写 TX FIFO 的偏移地址。(RO)

UART_TX_SRAM_RADDR 表示读 TX FIFO 的偏移地址。(RO)

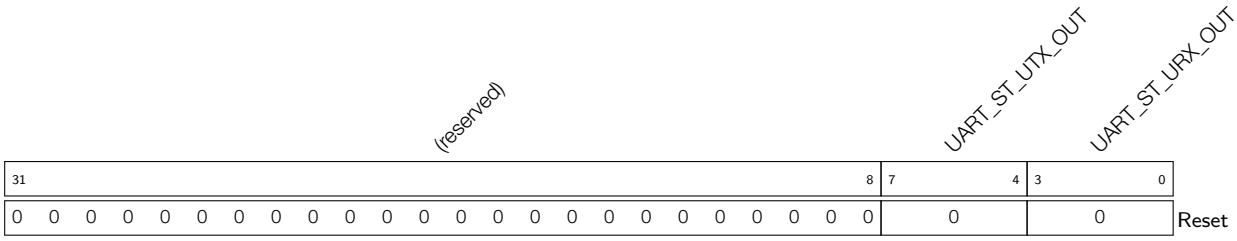
Register 25.23. UART_MEM_RX_STATUS_REG (0x006C)



UART_RX_SRAM_RADDR 表示读 RX FIFO 的偏移地址。(RO)

UART_RX_SRAM_WADDR 表示写 RX FIFO 的偏移地址。(RO)

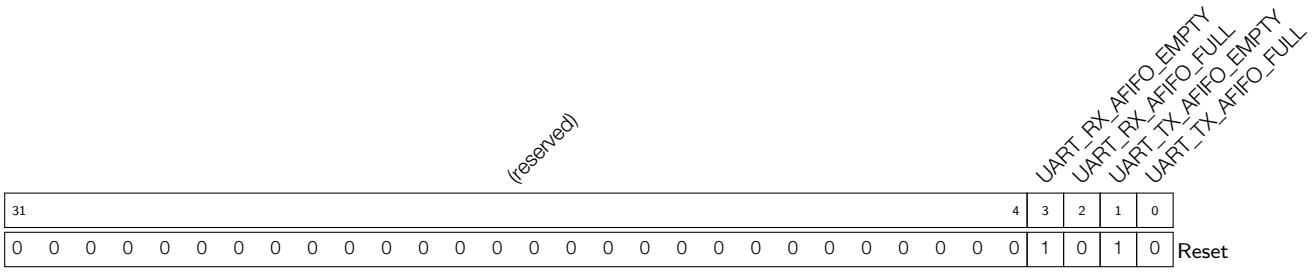
Register 25.24. UART_FSM_STATUS_REG (0x0070)



UART_ST_URX_OUT 表示 UART 接收器的状态。(RO)

UART_ST_UTX_OUT 表示 UART 发送器的状态。(RO)

Register 25.25. UART_AFIFO_STATUS_REG (0x0090)



UART_TX_AFIFO_FULL 表示 APB TX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

UART_TX_AFIFO_EMPTY 表示 APB TX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

UART_RX_AFIFO_FULL 表示 APB RX 异步 FIFO 是否已满。

0：未满

1：已满

(RO)

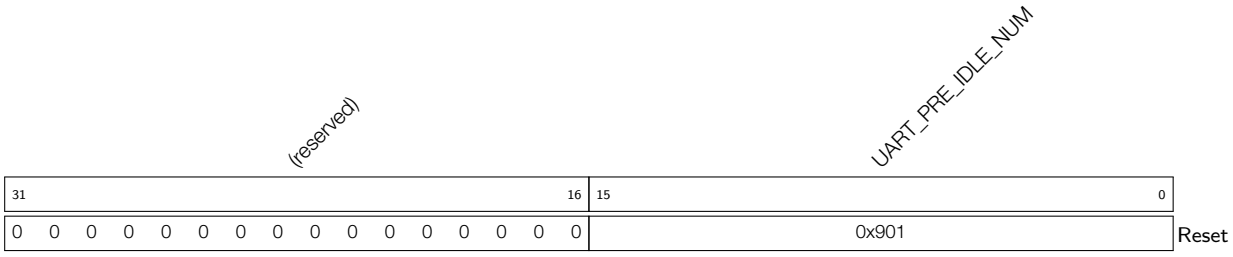
UART_RX_AFIFO_EMPTY 表示 APB RX 异步 FIFO 是否为空。

0：不为空

1：为空

(RO)

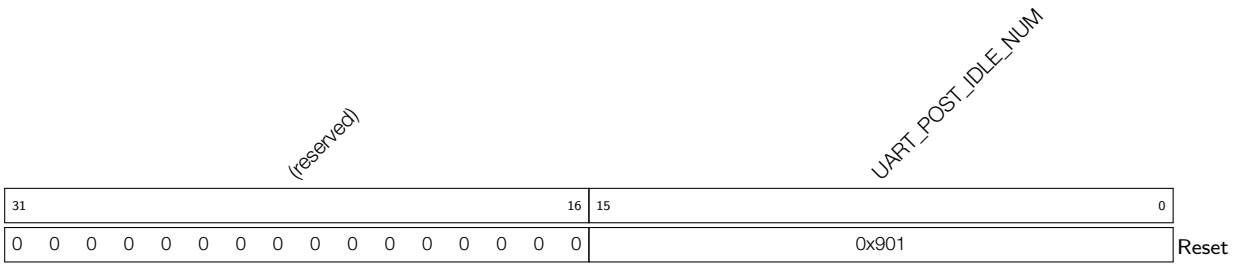
Register 25.26. UART_AT_CMD_PRECNT_SYNC_REG (0x0050)



UART_PRE_IDLE_NUM 配置接收器接收到第一个 AT_CMD 之前的空闲时间。

单位：位时间（传输一位所需的时间）。(R/W)

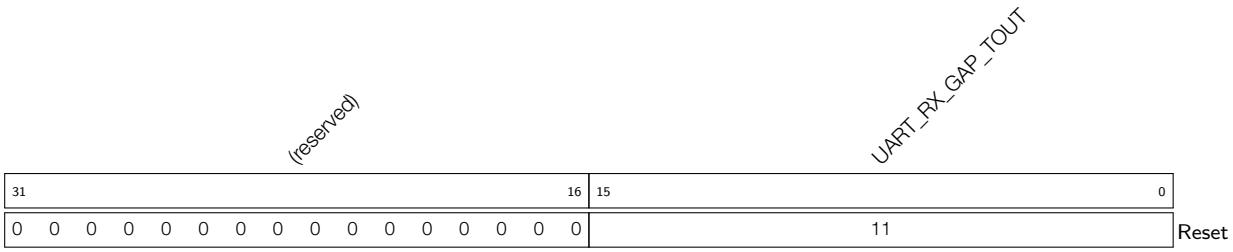
Register 25.27. UART_AT_CMD_POSTCNT_SYNC_REG (0x0054)



UART_POST_IDLE_NUM 配置最后一个 AT_CMD 和后续数据之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

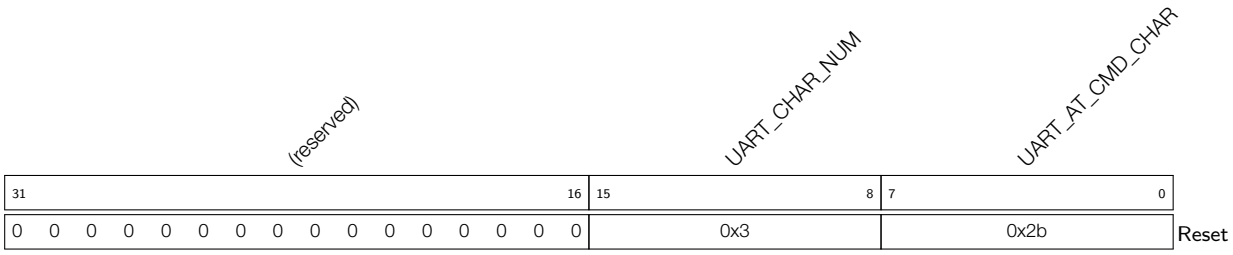
Register 25.28. UART_AT_CMD_GAPTOUT_SYNC_REG (0x0058)



UART_RX_GAP_TOUT 配置 AT_CMD 之间的间隔时间。

单位：位时间（传输一位所需的时间）。(R/W)

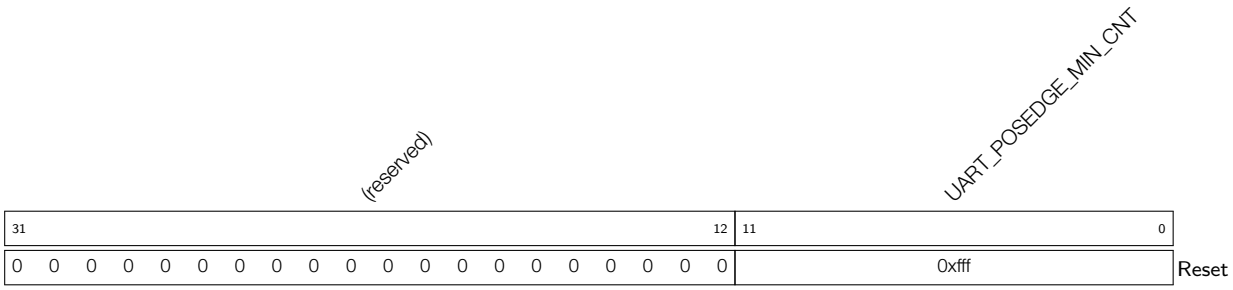
Register 25.29. UART_AT_CMD_CHAR_SYNC_REG (0x005C)



UART_AT_CMD_CHAR 配置 AT_CMD 字符。(R/W)

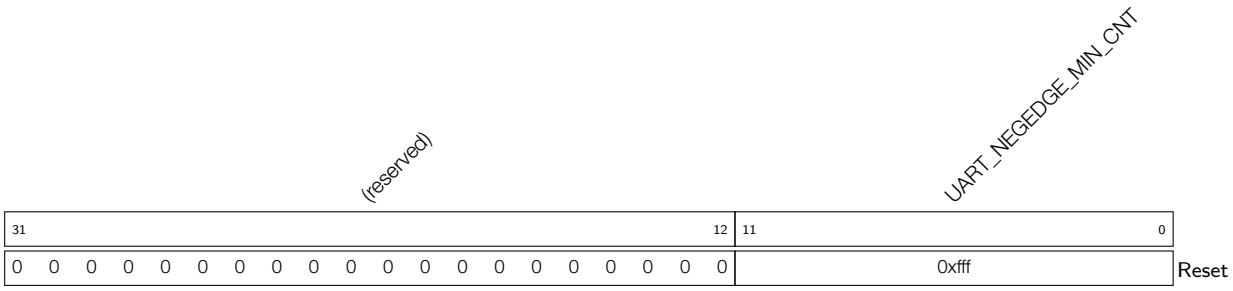
UART_CHAR_NUM 配置接收器可连续接收的 AT_CMD 字符数量。(R/W)

Register 25.30. UART_POSPULSE_REG (0x0074)



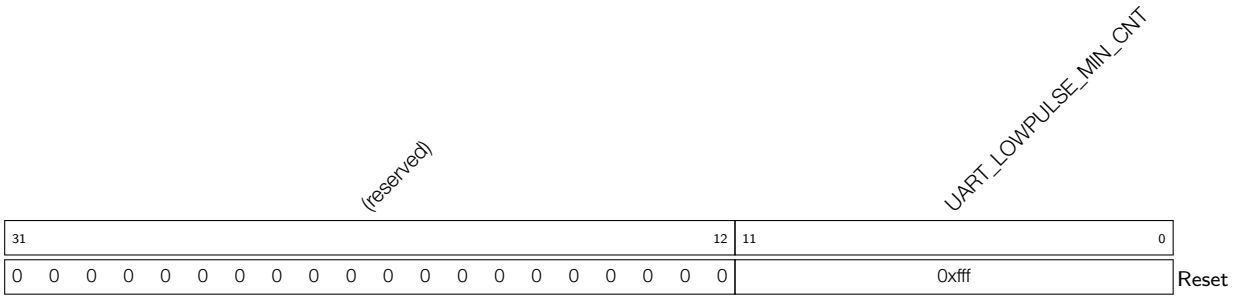
UART_POSEDGE_MIN_CNT 表示两个上升沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 25.31. UART_NEGPULSE_REG (0x0078)



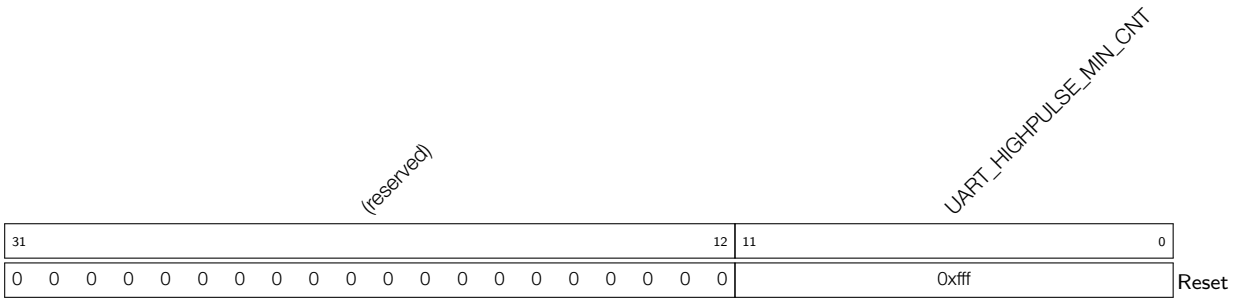
UART_NEGEDGE_MIN_CNT 两个下降沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 25.32. UART_LOWPULSE_REG (0x007C)



UART_LOWPULSE_MIN_CNT 表示低电平脉冲的最短持续时间，用于波特率检测。
单位：PLL_CLK 时钟周期。(RO)

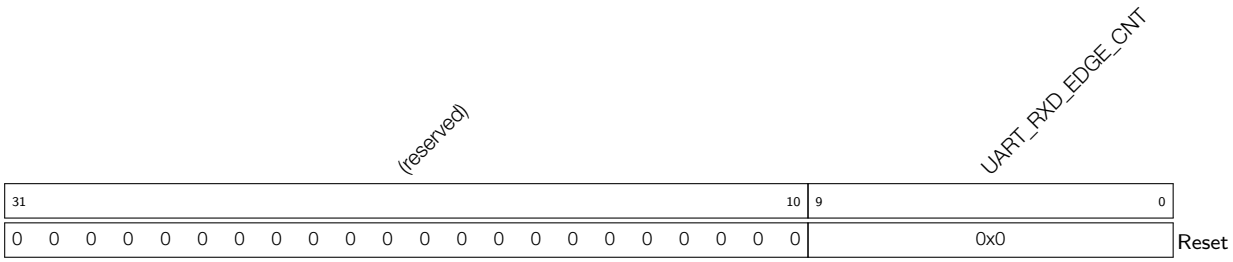
Register 25.33. UART_HIGHPULSE_REG (0x0080)



UART_HIGHPULSE_MIN_CNT 表示高电平脉冲最长持续时间，用于波特率检测。

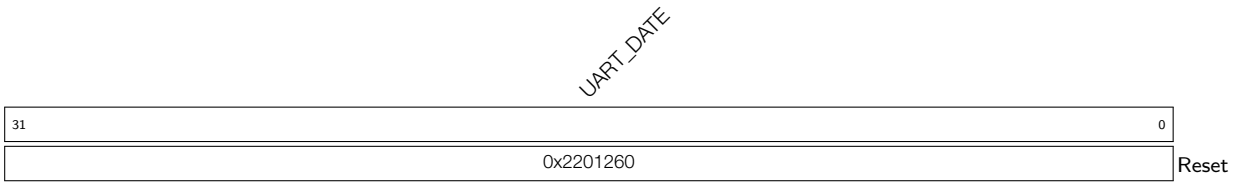
单位：PLL_CLK 时钟周期。(RO)

Register 25.34. UART_RXD_CNT_REG (0x0084)



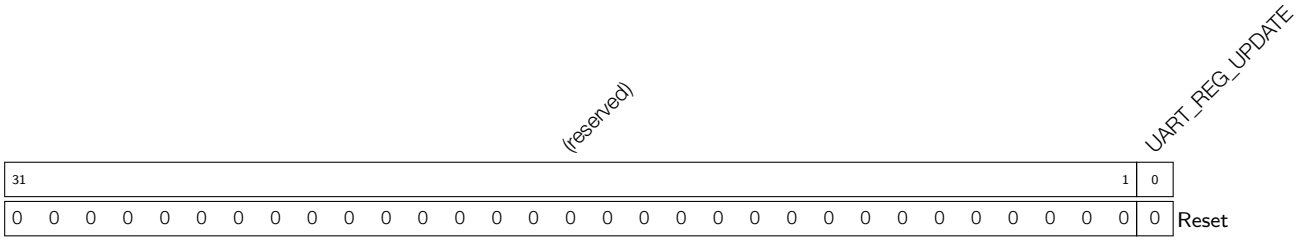
UART_RXD_EDGE_CNT 表示 RXD 沿变化的次数。用于波特率检测。(RO)

Register 25.35. UART_DATE_REG (0x008C)



UART_DATE 版本控制寄存器。(R/W)

Register 25.36. UART_REG_UPDATE_REG (0x0098)



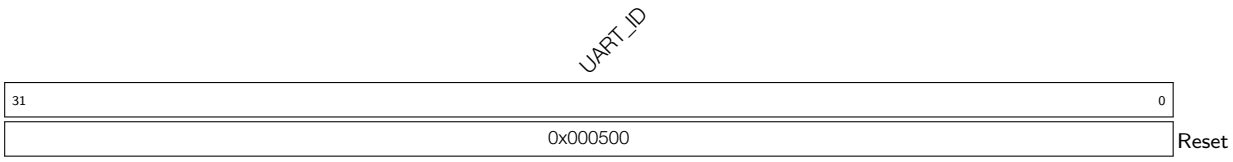
UART_REG_UPDATE 配置是否同步寄存器

0：不同步

1：同步

(R/W/SC)

Register 25.37. UART_ID_REG (0x009C)

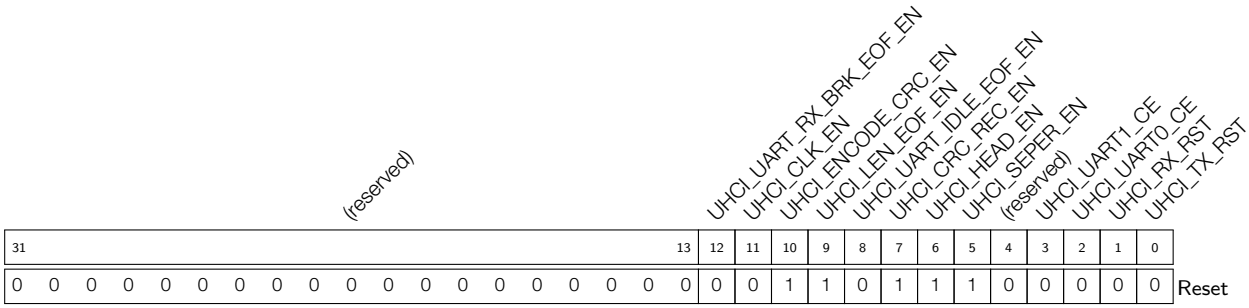


UART_ID 配置 UART ID。(R/W)

25.7.2 UHCI 寄存器

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

Register 25.38. UHCI_CONF0_REG (0x0000)



UHCI_TX_RST 先写 1 再写 0 复位解码器状态机。(R/W)

UHCI_RX_RST 先写 1 再写 0 复位编码器状态机。(R/W)

UHCI_UART0_CE 配置是否连接 UHCI 和 UART0。

0：不连接

1：连接

(R/W)

UHCI_UART1_CE 配置是否连接 UHCI 和 UART1。

0：不连接

1：连接

(R/W)

UHCI_SEPER_EN 配置是否使用特殊字符分隔数据帧。

0：不分隔

1：分隔

(R/W)

UHCI_HEAD_EN 配置是否用格式报头编码数据包。

0：不使用报头

1：使用报头

(R/W)

UHCI_CRC_REC_EN 配置 UHCI 是否开启接收 16 位 CRC。

0：关闭

1：开启

(R/W)

UHCI_UART_IDLE_EOF_EN 配置 UART 空闲时 UHCI 是否停止接收数据。

0：不停止

1：停止

(R/W)

见下页...

Register 25.38. UHCI_CONF0_REG (0x0000)

接上页...

UHCI_LEN_EOF_EN 配置 UHCI 解码器停止接收数据的条件。

0：接收到 0xC0 后停止

1：接收字节数达到指定值时停止。UHCI_HEAD_EN 为 1 时，指定值是 UCHI 数据包报头指定的数据长度。UHCI_HEAD_EN 为 0 时，指定值为配置值。

(R/W)

UHCI_ENCODE_CRC_EN 配置是否开启数据完整性检测，在数据末尾加 16 位 CCITT-CRC。

0：关闭

1：开启

(R/W)

UHCI_CLK_EN 配置时钟门控。

0：仅在应用写寄存器时开启时钟

1：一直强制为寄存器开启时钟

(R/W)

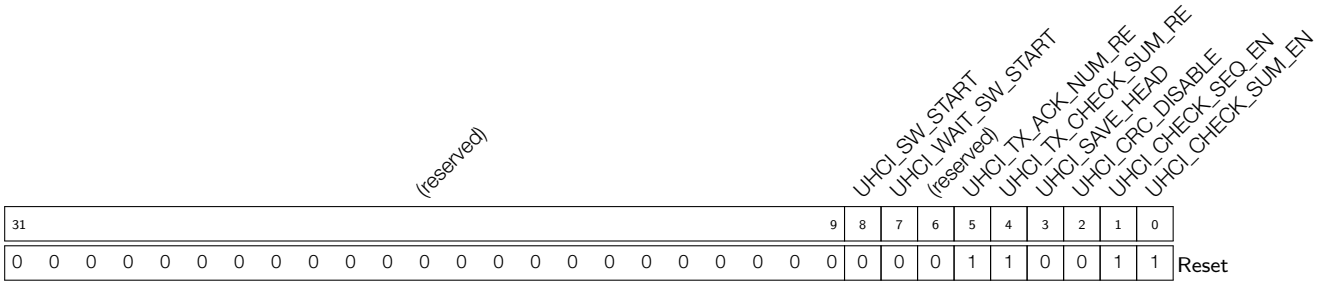
UHCI_UART_RX_BRK_EOF_EN 配置 UART 收到 NULL 帧后，UHCI 是否停止接收数据。

0：不停止

1：停止

(R/W)

Register 25.39. UHCI_CONF1_REG (0x0014)



UHCI_CHECK_SUM_EN 配置 UHCI 接收数据包时，是否开启报头校验和检查。

0：关闭

1：开启

(R/W)

UHCI_CHECK_SEQ_EN 配置 UHCI 接收数据包时，是否开启序列号检查。

0：关闭

1：开启

(R/W)

UHCI_CRC_DISABLE 配置是否开启 CRC 计算。

0：关闭

1：开启

仅在 UHCI 包中的数据完整性检测位为 1 时有效。

(R/W)

UHCI_SAVE_HEAD 配置 UHCI 接收数据包时，是否保存数据包报头。

0：不保存

1：保存

(R/W)

UHCI_TX_CHECK_SUM_RE 配置用校验和编码数据包。

0：不使用

1：使用

(R/W)

UHCI_TX_ACK_NUM_RE 配置准备发送可靠数据包时，是否用 ACK 编码该数据包。

0：不使用

1：使用

(R/W)

UHCI_WAIT_SW_START 配置是否将 UHCI 编码器的状态机调至 ST_SW_WAIT 状态。

0：否

1：是

(R/W)

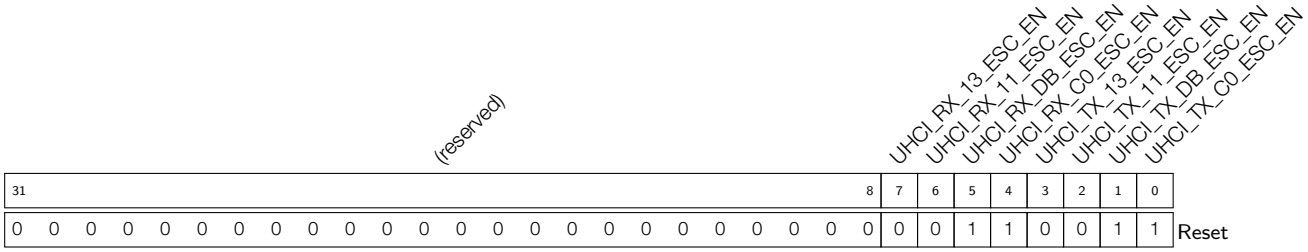
UHCI_SW_START 编码器的状态机为 ST_SW_WAIT 状态时，配置 UHCI 是否开始发送数据包。

0：不发送

1：开始发送

(R/W/SC)

Register 25.40. UHCI_ESCAPE_CONF_REG (0x0020)



UHCI_TX_C0_ESC_EN 配置 DMA 接收数据时，是否开启 0xC0 字符解码。

0：关闭

1：开启

(R/W)

UHCI_TX_DB_ESC_EN 配置 DMA 接收数据时，是否开启 0xDB 字符解码。

0：关闭

1：开启

(R/W)

UHCI_TX_11_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x11 解码。

0：关闭

1：开启

(R/W)

UHCI_TX_13_ESC_EN 配置 DMA 接收数据时，是否开启流控字符 0x13 解码。

0：关闭

1：开启

(R/W)

UHCI_RX_C0_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xC0。

0：不替换

1：替换

(R/W)

UHCI_RX_DB_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换 0xDB。

0：不替换

1：替换

(R/W)

UHCI_RX_11_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x11。

0：不替换

1：替换

(R/W)

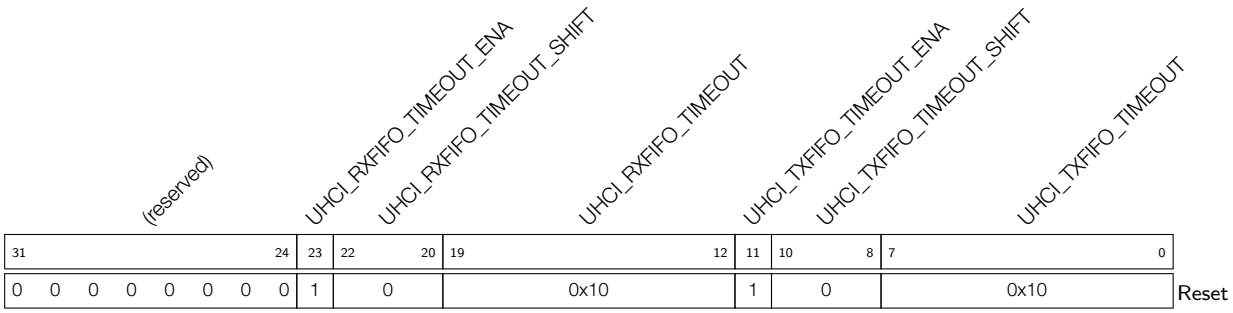
UHCI_RX_13_ESC_EN 配置 DMA 接收数据时，是否用特殊字符替换流控字符 0x13。

0：不替换

1：替换

(R/W)

Register 25.41. UHCI_HUNG_CONF_REG (0x0024)



UHCI_TXFIFO_TIMEOUT 配置 DMA 接收数据的超时值。

单位：毫秒。(R/W)

UHCI_TXFIFO_TIMEOUT_SHIFT 配置 TX FIFO 超时计数器的阈值上限。(R/W)

UHCI_TXFIFO_TIMEOUT_ENA 配置是否开启 TX FIFO 接收数据超时。

0：关闭

1：开启

(R/W)

UHCI_RXFIFO_TIMEOUT 配置 DMA 从 RAM 读取数据的超时值。单位：毫秒。(R/W)

UHCI_RXFIFO_TIMEOUT_SHIFT 配置 RX FIFO 超时计数器的阈值上限。(R/W)

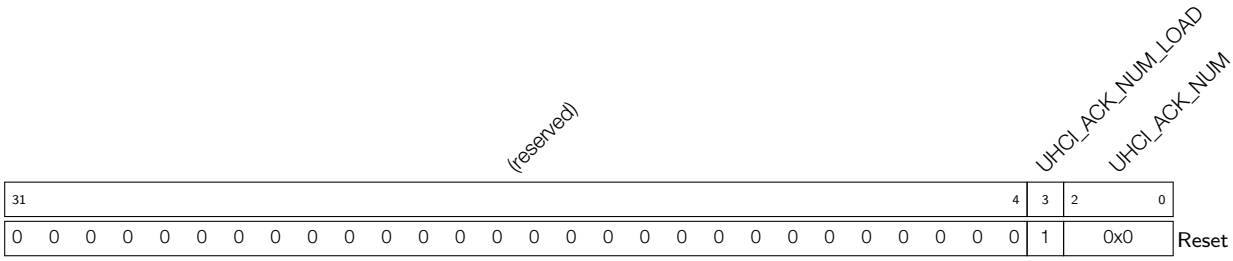
UHCI_RXFIFO_TIMEOUT_ENA 配置是否开启 DMA 发送数据超时。

0：关闭

1：开启

(R/W)

Register 25.42. UHCI_ACK_NUM_REG (0x0028)



UHCI_ACK_NUM 配置软件流控中使用的 ACK 数量。(R/W)

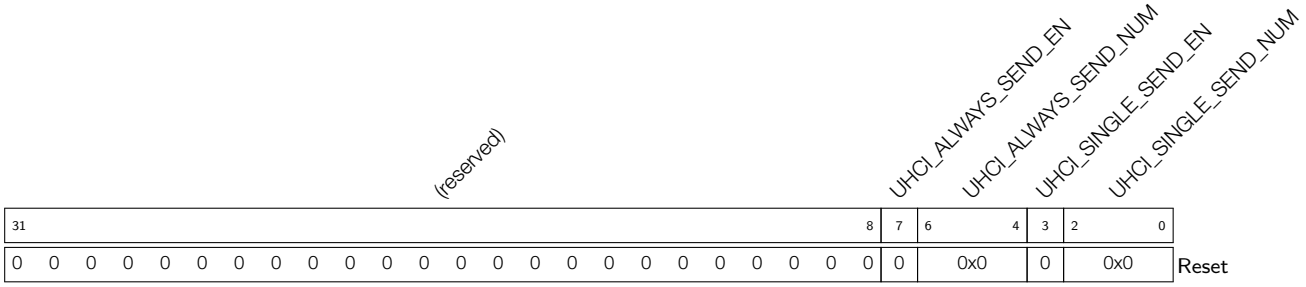
UHCI_ACK_NUM_LOAD 配置是否加载 ACK。

0：不加载

1：加载

(WT)

Register 25.43. UHCI_QUICK_SENT_REG (0x0030)



UHCI_SINGLE_SEND_NUM 配置 single_send 模式下发送哪个寄存器中的数据。

0：Q0 寄存器
1：Q1 寄存器
2：Q2 寄存器
3：Q3 寄存器
4：Q4 寄存器
5：Q5 寄存器
6：Q6 寄存器
7：无效值，没有作用
(R/W)

UHCI_SINGLE_SEND_EN 配置是否开启 single_send 模式。

0：关闭
1：开启
(R/W/SC)

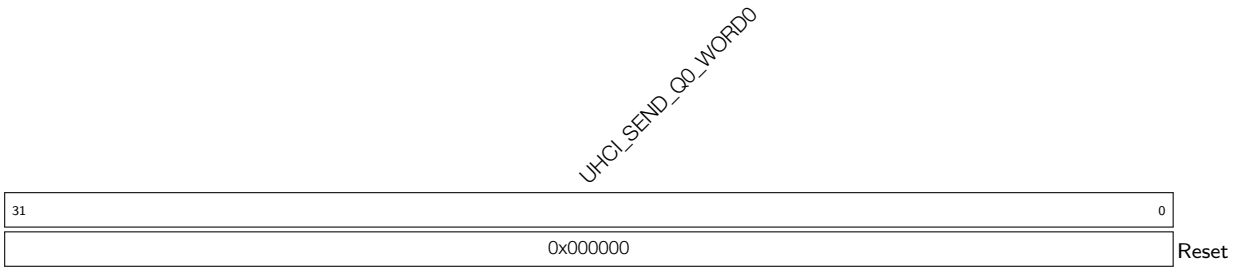
UHCI_ALWAYS_SEND_NUM 配置 always_send 模式下发送哪个寄存器中的数据。

0：Q0 寄存器
1：Q1 寄存器
2：Q2 寄存器
3：Q3 寄存器
4：Q4 寄存器
5：Q5 寄存器
6：Q6 寄存器
7：无效值，没有作用
(R/W)

UHCI_ALWAYS_SEND_EN 配置是否开启 always_send 模式。

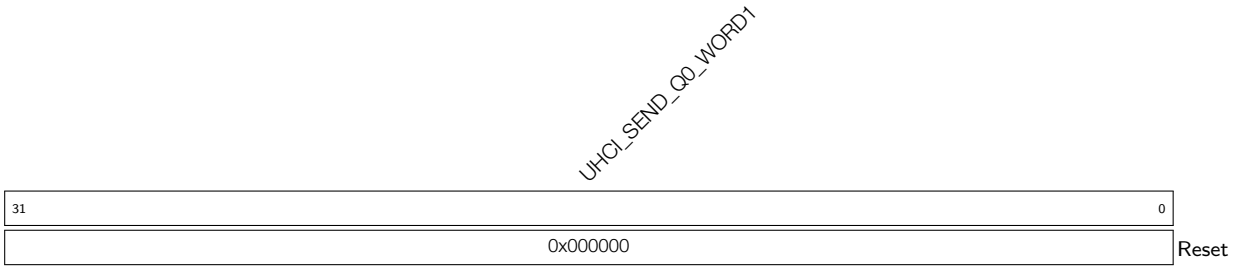
0：关闭
1：开启
(R/W)

Register 25.44. UHCI_REG_Q0_WORD0_REG (0x0034)



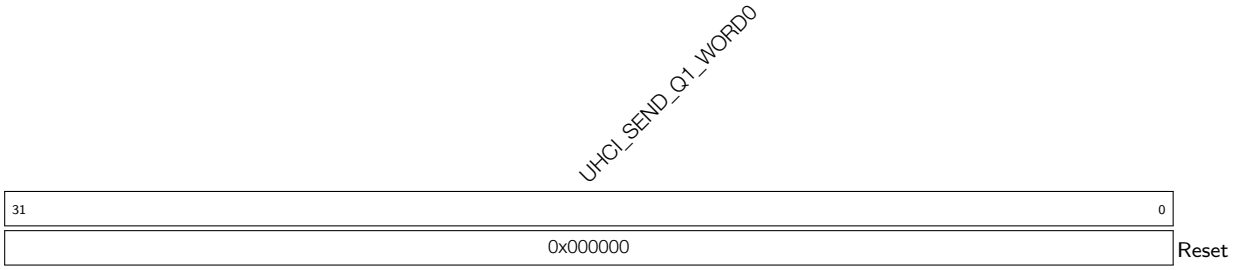
UHCI_SEND_Q0_WORD0 Q0 寄存器待发送数据。(R/W)

Register 25.45. UHCI_REG_Q0_WORD1_REG (0x0038)



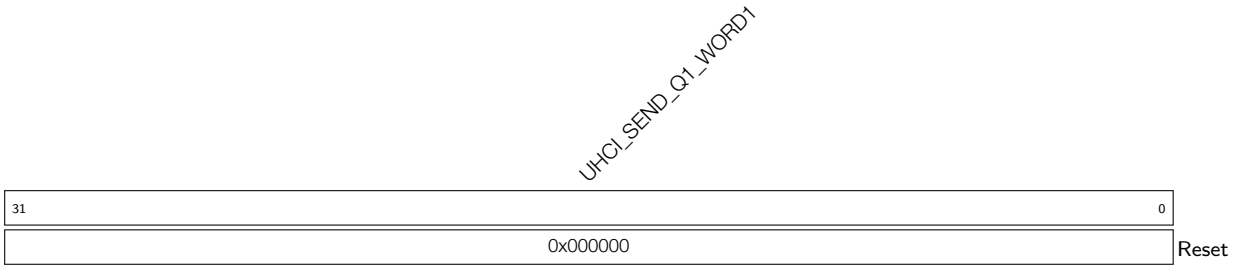
UHCI_SEND_Q0_WORD1 Q0 寄存器待发送数据。(R/W)

Register 25.46. UHCI_REG_Q1_WORD0_REG (0x003C)



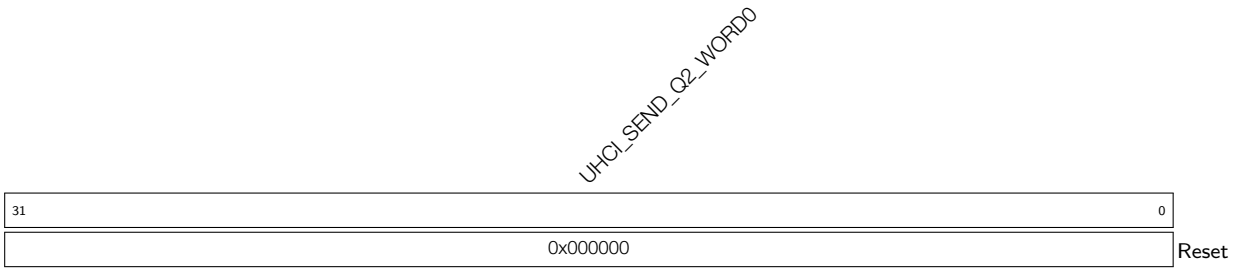
UHCI_SEND_Q1_WORD0 Q1 寄存器待发送数据。(R/W)

Register 25.47. UHCI_REG_Q1_WORD1_REG (0x0040)



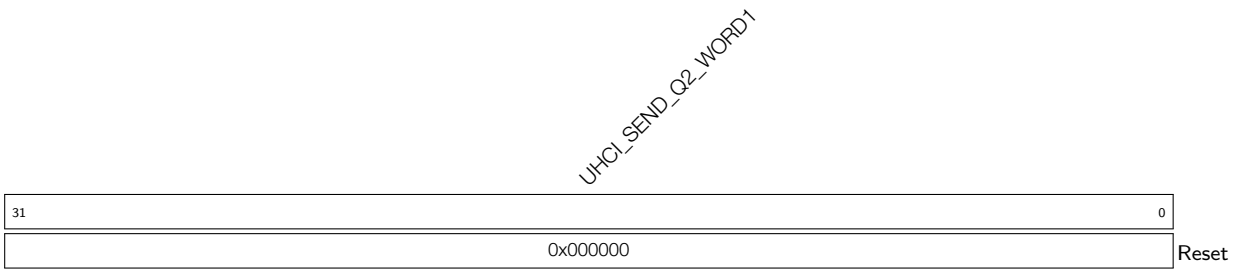
UHCI_SEND_Q1_WORD1 Q1 寄存器待发送数据。(R/W)

Register 25.48. UHCI_REG_Q2_WORD0_REG (0x0044)



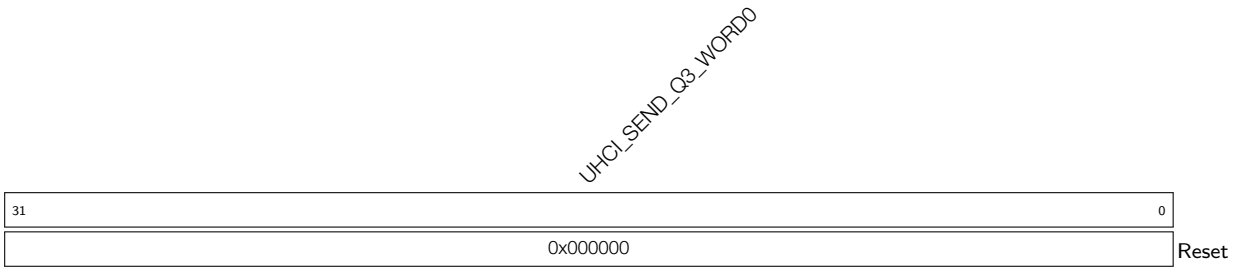
UHCI_SEND_Q2_WORD0 Q2 寄存器待发送数据。(R/W)

Register 25.49. UHCI_REG_Q2_WORD1_REG (0x0048)



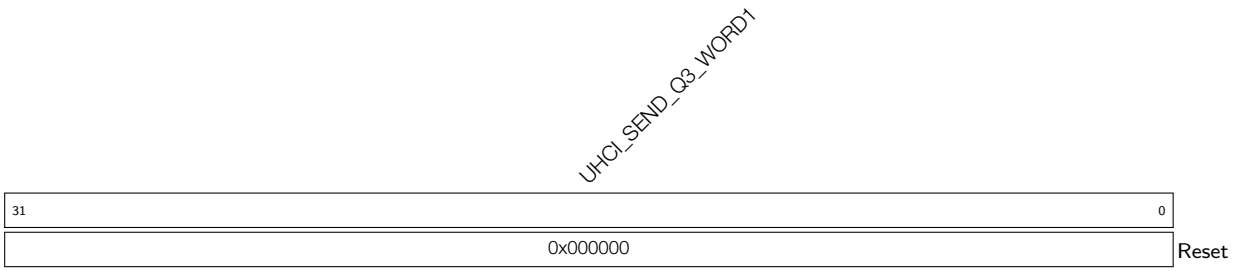
UHCI_SEND_Q2_WORD1 Q2 寄存器待发送数据。(R/W)

Register 25.50. UHCI_REG_Q3_WORD0_REG (0x004C)



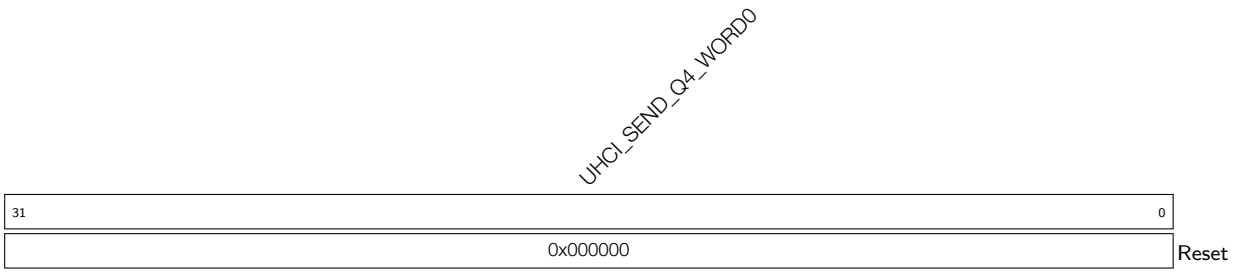
UHCI_SEND_Q3_WORD0 Q3 寄存器待发送数据。(R/W)

Register 25.51. UHCI_REG_Q3_WORD1_REG (0x0050)



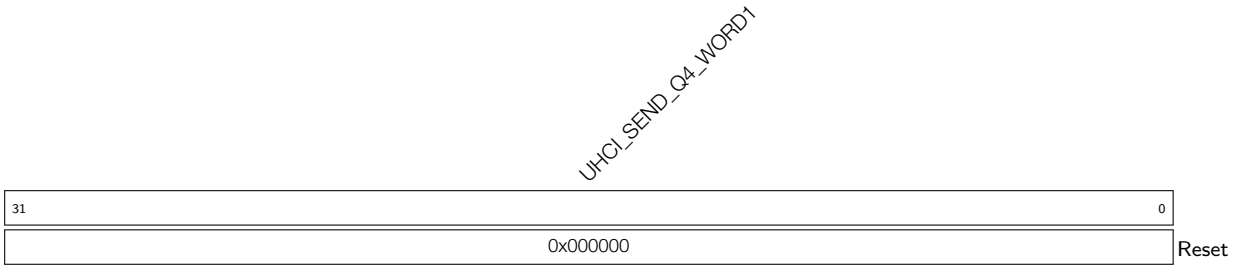
UHCI_SEND_Q3_WORD1 Q3 寄存器待发送数据。(R/W)

Register 25.52. UHCI_REG_Q4_WORD0_REG (0x0054)



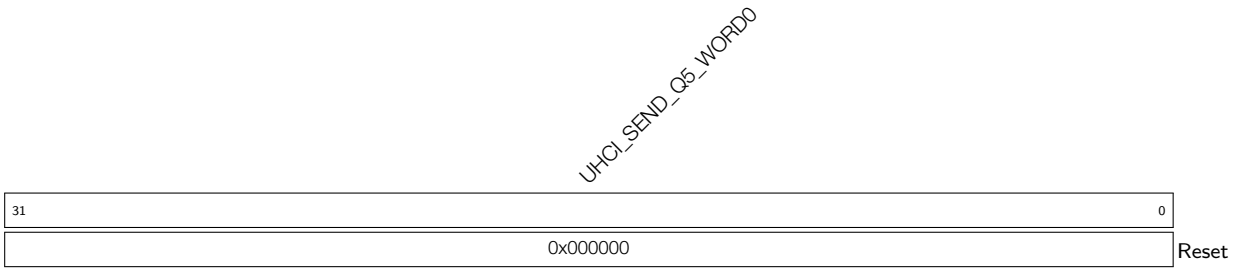
UHCI_SEND_Q4_WORD0 Q4 寄存器待发送数据。(R/W)

Register 25.53. UHCI_REG_Q4_WORD1_REG (0x0058)



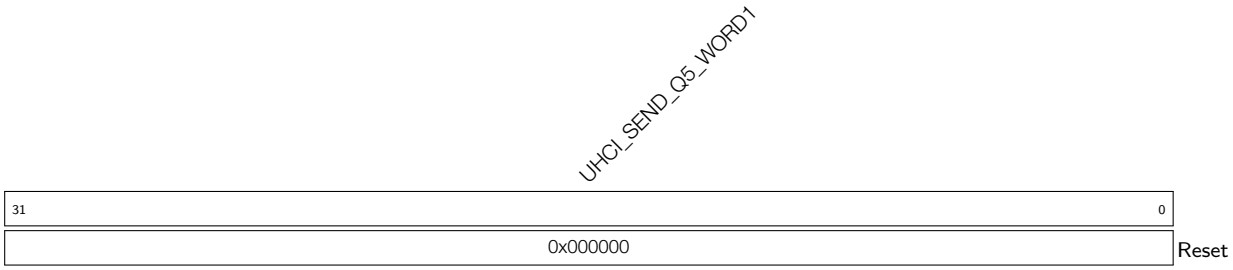
UHCI_SEND_Q4_WORD1 Q4 寄存器待发送数据。(R/W)

Register 25.54. UHCI_REG_Q5_WORD0_REG (0x005C)



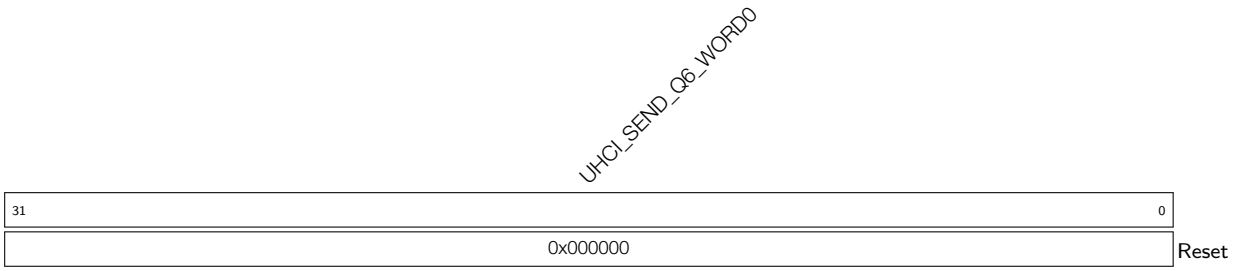
UHCI_SEND_Q5_WORD0 Q5 寄存器待发送数据。(R/W)

Register 25.55. UHCI_REG_Q5_WORD1_REG (0x0060)



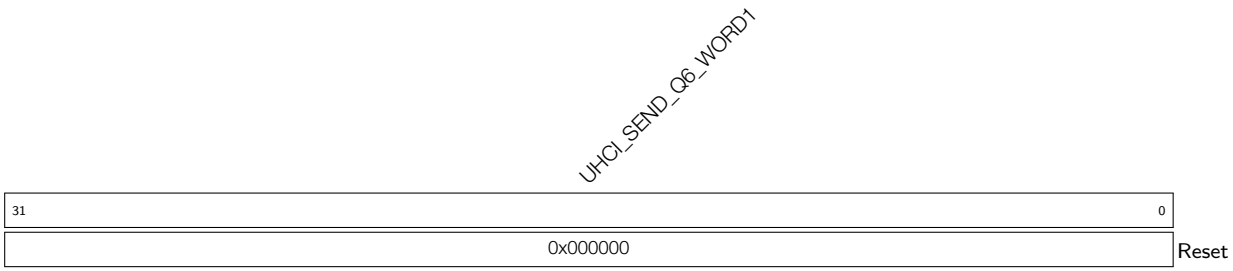
UHCI_SEND_Q5_WORD1 Q5 寄存器待发送数据。(R/W)

Register 25.56. UHCI_REG_Q6_WORD0_REG (0x0064)



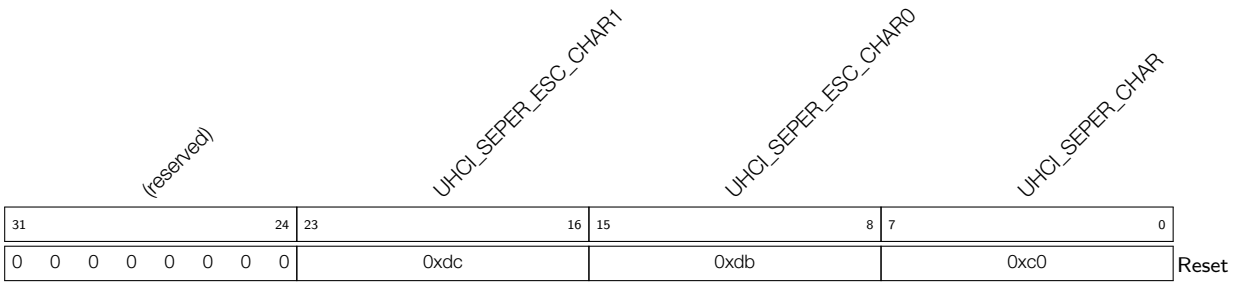
UHCI_SEND_Q6_WORD0 Q6 寄存器待发送数据。(R/W)

Register 25.57. UHCI_REG_Q6_WORD1_REG (0x0068)



UHCI_SEND_Q6_WORD1 Q6 寄存器待发送数据。(R/W)

Register 25.58. UHCI_ESC_CONF0_REG (0x006C)

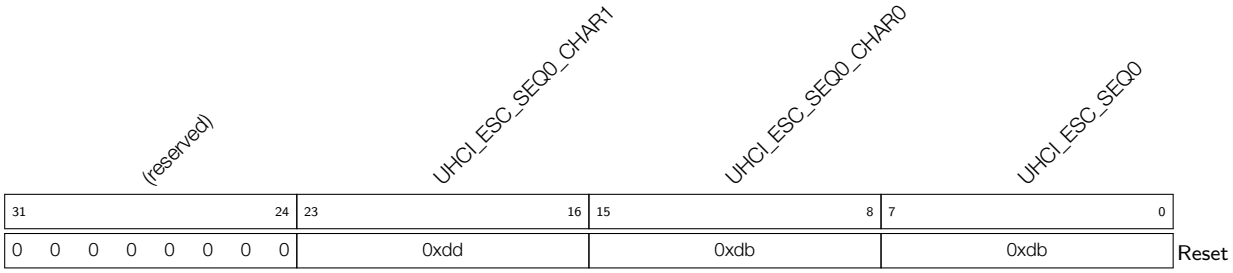


UHCI_SEPER_CHAR 配置编码数据包的分隔符，默认为 0xC0。(R/W)

UHCI_SEPER_ESC_CHAR0 配置 SLIP 转义序列的第一个字符，默认为 0xDB。(R/W)

UHCI_SEPER_ESC_CHAR1 配置 SLIP 转义序列的第二个字符，默认为 0xDC。(R/W)

Register 25.59. UHCI_ESC_CONF1_REG (0x0070)

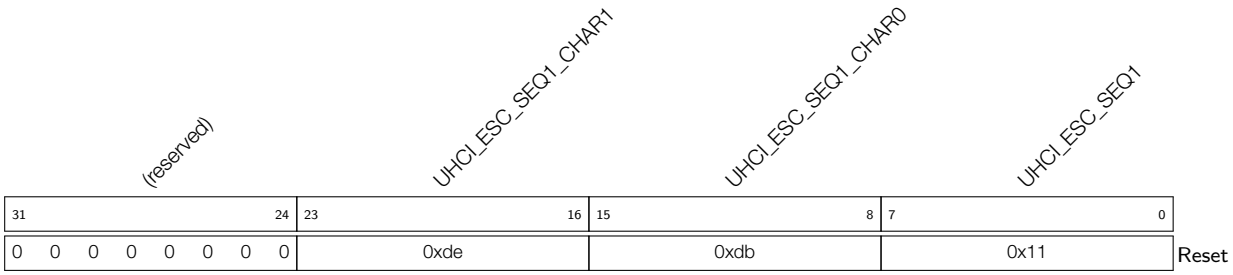


UHCI_ESC_SEQ0 配置需编码的字符，默认为用作 SLIP 转义序列第一个字符的 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ0 默认为 0xDB。(R/W)

UHCI_ESC_SEQ0_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ0 默认为 0xDD。(R/W)

Register 25.60. UHCI_ESC_CONF2_REG (0x0074)

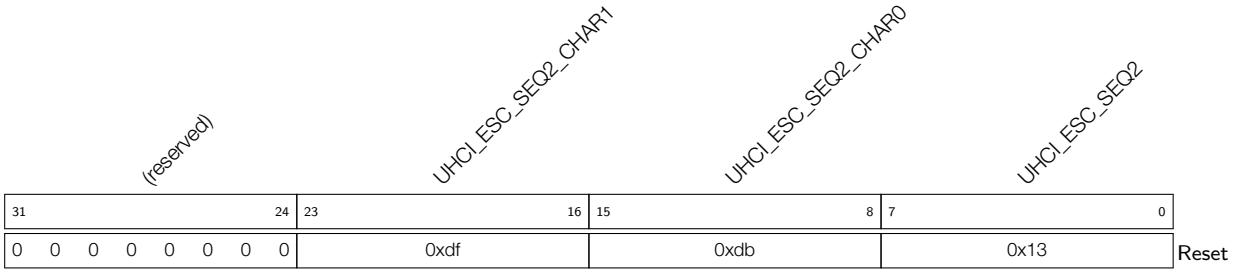


UHCI_ESC_SEQ1 配置需编码的字符，默认为流控字符的 0x11。(R/W)

UHCI_ESC_SEQ1_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ1 默认为 0xDB。(R/W)

UHCI_ESC_SEQ1_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ1 默认为 0xDE。(R/W)

Register 25.61. UHCI_ESC_CONF3_REG (0x0078)

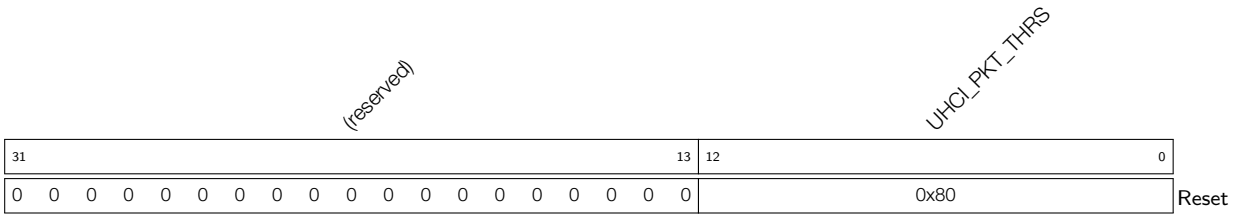


UHCI_ESC_SEQ2 配置需编码的字符，默认为流控字符的 0x13。(R/W)

UHCI_ESC_SEQ2_CHAR0 配置 SLIP 转义序列的第一个字符，UHCI_ESC_SEQ2 默认为 0xDB。(R/W)

UHCI_ESC_SEQ2_CHAR1 配置 SLIP 转义序列的第二个字符，UHCI_ESC_SEQ2 默认为 0xDF。(R/W)

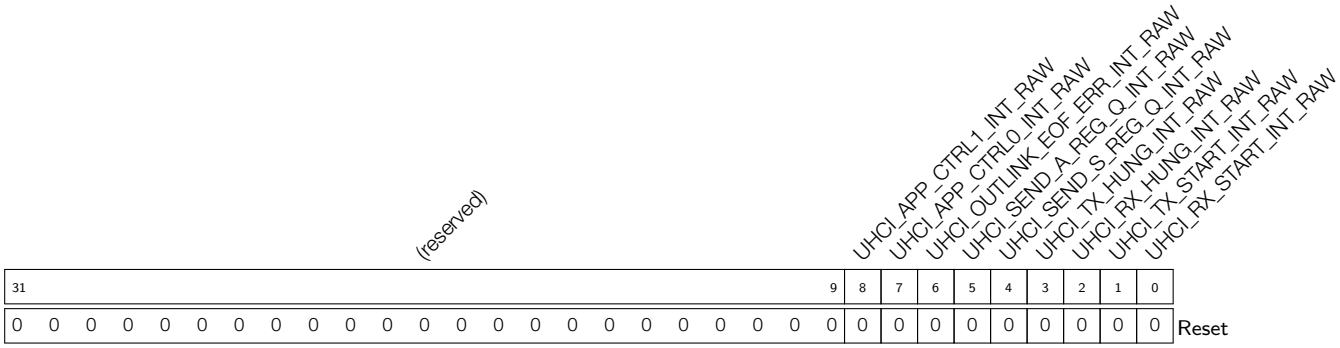
Register 25.62. UHCI_PKT_THRES_REG (0x007C)



UHCI_PKT_THRS 配置数据包的最大长度。

单位：字节。仅在 UHCI_HEAD_EN 为 0 时有效。(R/W)

Register 25.63. UHCI_INT_RAW_REG (0x0004)



UHCI_RX_START_INT_RAW UHCI_RX_START_INT 的原始中断状态。(R/WTC/SS)

UHCI_TX_START_INT_RAW UHCI_TX_START_INT 的原始中断状态。(R/WTC/SS)

UHCI_RX_HUNG_INT_RAW UHCI_RX_HUNG_INT 的原始中断状态。(R/WTC/SS)

UHCI_TX_HUNG_INT_RAW UHCI_TX_HUNG_INT 的原始中断状态。(R/WTC/SS)

UHCI_SEND_S_REG_Q_INT_RAW UHCI_SEND_S_REG_Q_INT 的原始中断状态。(R/WTC/SS)

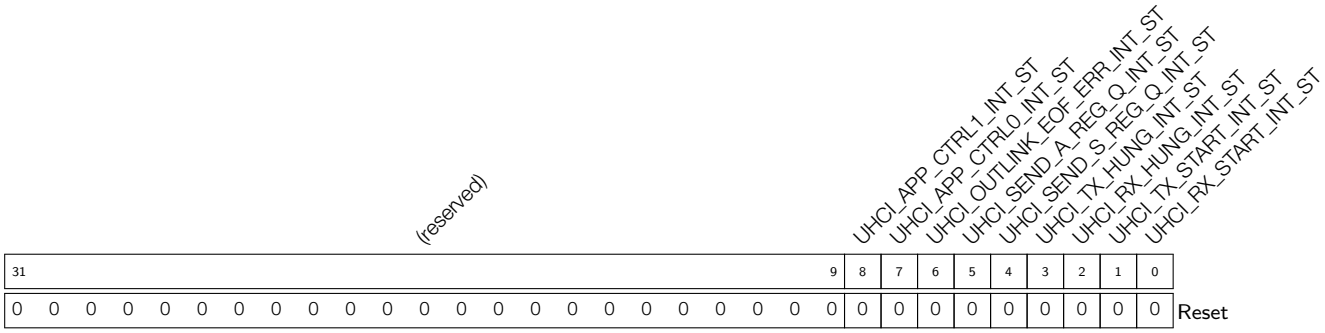
UHCI_SEND_A_REG_Q_INT_RAW UHCI_SEND_A_REG_Q_INT 的原始中断状态。(R/WTC/SS)

UHCI_OUTLINK_EOF_ERR_INT_RAW UHCI_OUTLINK_EOF_ERR_INT 的原始中断状态。(R/WTC/SS)

UHCI_APP_CTRL0_INT_RAW UHCI_APP_CTRL0_INT 的原始中断状态。(R/W)

UHCI_APP_CTRL1_INT_RAW UHCI_APP_CTRL1_INT 的原始中断状态。(R/W)

Register 25.64. UHCI_INT_ST_REG (0x0008)



UHCI_RX_START_INT_ST UHCI_RX_START_INT 的屏蔽中断状态。(RO)

UHCI_TX_START_INT_ST UHCI_TX_START_INT 的屏蔽中断状态。(RO)

UHCI_RX_HUNG_INT_ST UHCI_RX_HUNG_INT 的屏蔽中断状态。(RO)

UHCI_TX_HUNG_INT_ST UHCI_TX_HUNG_INT 的屏蔽中断状态。(RO)

UHCI_SEND_S_REG_Q_INT_ST UHCI_SEND_S_REG_Q_INT 的屏蔽中断状态。(RO)

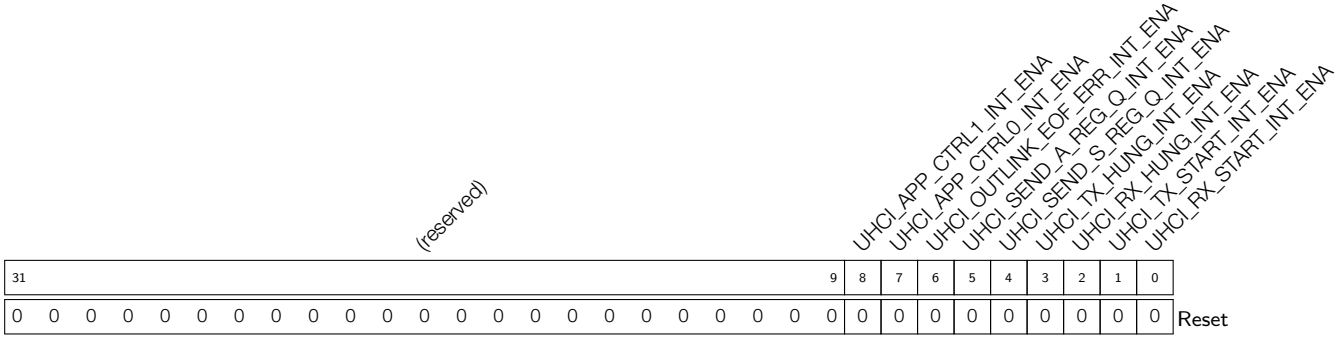
UHCI_SEND_A_REG_Q_INT_ST UHCI_SEND_A_REG_Q_INT 的屏蔽中断状态。(RO)

UHCI_OUTLINK_EOF_ERR_INT_ST UHCI_OUTLINK_EOF_ERR_INT 的屏蔽中断状态。(RO)

UHCI_APP_CTRL0_INT_ST UHCI_APP_CTRL0_INT 的屏蔽中断状态。(RO)

UHCI_APP_CTRL1_INT_ST UHCI_APP_CTRL1_INT 的屏蔽中断状态。(RO)

Register 25.65. UHCI_INT_ENA_REG (0x000C)



UHCI_RX_START_INT_ENA 写 1 使能 UHCI_RX_START_INT。(R/W)

UHCI_TX_START_INT_ENA 写 1 使能 UHCI_TX_START_INT。(R/W)

UHCI_RX_HUNG_INT_ENA 写 1 使能 UHCI_RX_HUNG_INT。(R/W)

UHCI_TX_HUNG_INT_ENA 写 1 使能 UHCI_TX_HUNG_INT。(R/W)

UHCI_SEND_S_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_S_REG_Q_INT。(R/W)

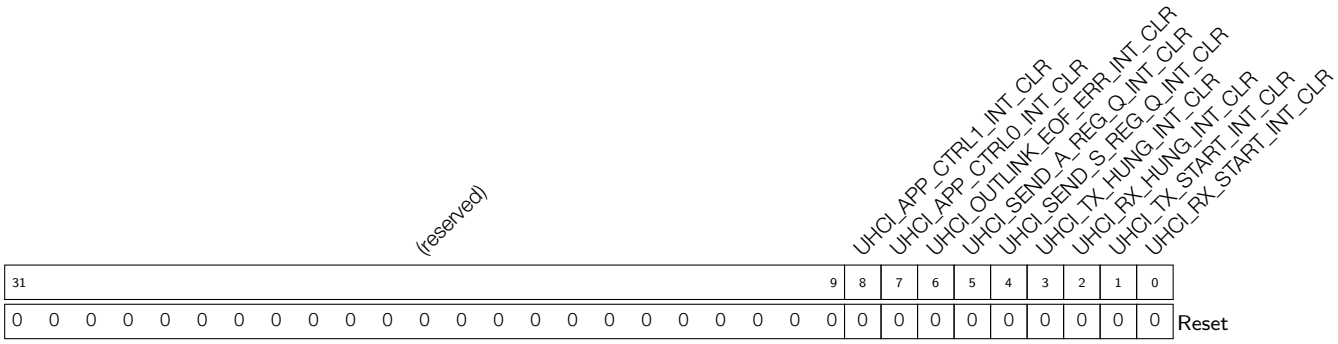
UHCI_SEND_A_REG_Q_INT_ENA 写 1 使能 UHCI_SEND_A_REG_Q_INT。(R/W)

UHCI_OUTLINK_EOF_ERR_INT_ENA 写 1 使能 UHCI_OUTLINK_EOF_ERR_INT。(R/W)

UHCI_APP_CTRL0_INT_ENA 写 1 使能 UHCI_APP_CTRL0_INT。(R/W)

UHCI_APP_CTRL1_INT_ENA 写 1 使能 UHCI_APP_CTRL1_INT。(R/W)

Register 25.66. UHCI_INT_CLR_REG (0x0010)



UHCI_RX_START_INT_CLR 写 1 清除 UHCI_RX_START_INT。(WT)

UHCI_TX_START_INT_CLR 写 1 清除 UHCI_TX_START_INT。(WT)

UHCI_RX_HUNG_INT_CLR 写 1 清除 UHCI_RX_HUNG_INT。(WT)

UHCI_TX_HUNG_INT_CLR 写 1 清除 UHCI_TX_HUNG_INT。(WT)

UHCI_SEND_S_REG_Q_INT_CLR 写 1 清除 UHCI_SEND_S_REG_Q_INT。(WT)

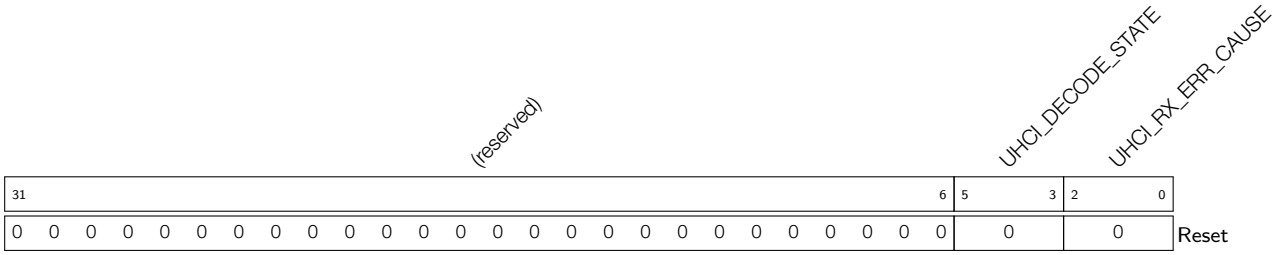
UHCI_SEND_A_REG_Q_INT_CLR 写 1 清除 UHCI_SEND_A_REG_Q_INT。(WT)

UHCI_OUTLINK_EOF_ERR_INT_CLR 写 1 清除 UHCI_OUTLINK_EOF_ERR_INT。(WT)

UHCI_APP_CTRL0_INT_CLR 写 1 清除 UHCI_APP_CTRL0_INT。(WT)

UHCI_APP_CTRL1_INT_CLR 写 1 清除 UHCI_APP_CTRL1_INT。(WT)

Register 25.67. UHCI_STATE0_REG (0x0018)

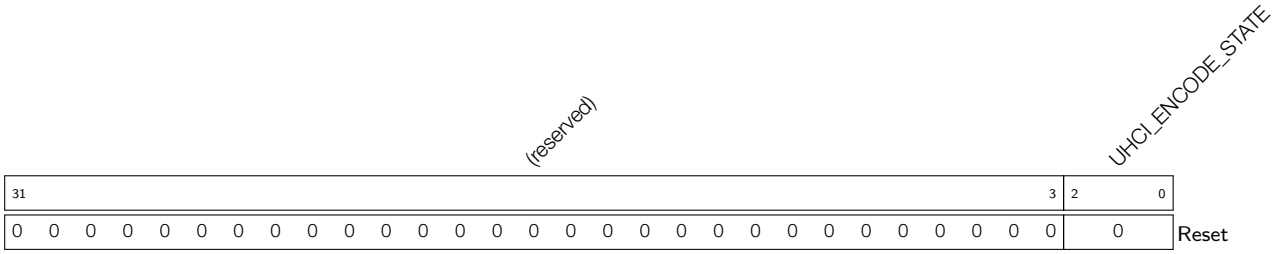


UHCI_RX_ERR_CAUSE 表示 DMA 接收数据错误的原因。

0：无效，没有作用
1：HCI 包校验和错误
2：HCI 包序列号错误
3：HCI 包 CRC 位错误
4：找到 0xC0 但接收的 HCI 包不完整
5：未找到 0xC0 但接收的 HCI 包完整
6：CRC 检测错误
7：无效，没有作用
(RO)

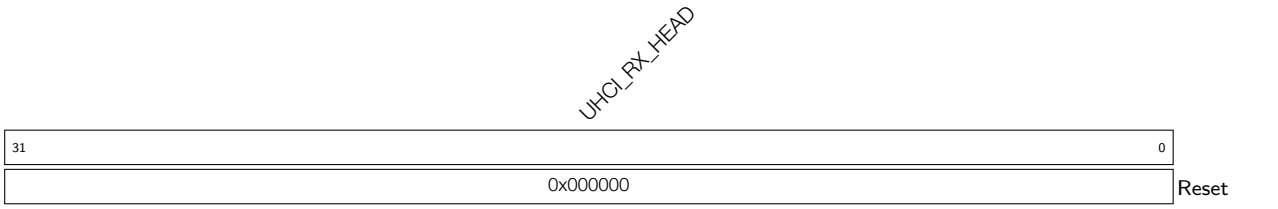
UHCI_DECODE_STATE 表示 UHCI 解码器状态。(RO)

Register 25.68. UHCI_STATE1_REG (0x001C)



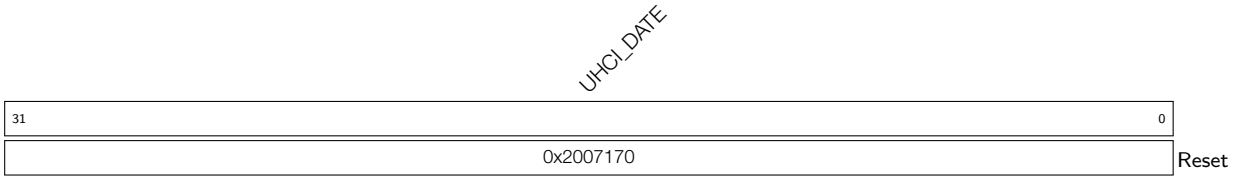
UHCI_ENCODE_STATE 表示 UHCI 编码器状态。(RO)

Register 25.69. UHCI_RX_HEAD_REG (0x002C)



UHCI_RX_HEAD 表示当前接收数据包的报头。(RO)

Register 25.70. UHCI_DATE_REG (0x0080)



UHCI_DATE 版本控制寄存器。(R/W)

26 SPI 控制器 (SPI)

26.1 概述

串行外设接口 (SPI) 是一种同步串行接口，可用于与外围设备进行通信。ESP32-H2 芯片集成了三个 SPI 控制器：

- SPI0
- SPI1
- 通用 SPI2，即 GP-SPI2

SPI0 和 SPI1 控制器 (MSPI) 主要供内部使用以访问外部 flash 及 PSRAM。因此，本章节将主要介绍 GP-SPI2 控制器。

26.2 术语

为了更好地说明 GP-SPI2 的功能，本章使用了以下术语。

主机模式	GP-SPI2 用作 SPI 主机，发起 SPI 传输事务。
从机模式	GP-SPI2 用作 SPI 从机，当其 CS 被拉低时，与 SPI 主机进行数据传输。
MISO	主机输入，从机输出。数据从从机发送至主机。
MOSI	主机输出，从机输入。数据从主机发送至从机。
传输事务	一次完整的传输事务：主机拉低从机的 CS 线，开始传输数据，然后再拉高从机的 CS 线。传输事务为原子操作，即不可打断。
SPI 传输	SPI 主机与从机完成数据交换的一次完整过程。一次 SPI 传输可以包含一个或多个 SPI 传输事务。
单次传输	在这种传输模式下，仅包含一次传输事务。
CPU 控制的传输	由 CPU 控制，SPI_W0_REG ~ SPI_W15_REG 与 SPI 设备之间的数据传输。
DMA 控制的传输	由 DMA 引擎控制，DMA 与 SPI 设备之间的数据传输。
分段配置传输	主机模式下，DMA 控制的数据传输。此类传输包含多个传输事务（分段），每个传输事务均可独立配置。
从机连续传输	从机模式下，DMA 控制的数据传输。此类传输包含多个传输事务（分段）。
全双工	主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。
半双工	主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。
四线全双工	四线包括：时钟线、片选线和两条数据线。其中，可使用两条数据线同时发送和接收数据。
四线半双工	四线包括：时钟线、片选线和两条数据线。其中，分时使用两条数据线，不可同时使用。
三线半双工	三线包括：时钟线、片选线和一条数据线。使用数据线分时发送和接收数据。
1-bit SPI	一个时钟周期传输一位数据。
(2-bit) Dual SPI	一个时钟周期传输两个数据位。

Dual Output Read	Dual SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或两位数据。
Dual I/O Read	Dual SPI 的另外一种数据模式，一个时钟周期可传输一位命令、或两位地址、或两位数据。
(4-bit) Quad SPI	一个时钟周期传输四个数据位。
Quad Output Read	Quad SPI 的一种数据模式，一个时钟周期可传输一位命令、或一位地址、或四位数据。
Quad I/O Read	Quad SPI 的另一种数据模式，一个时钟周期可传输一位命令、或四位地址、或四位数据。
QPI	一个时钟周期可传输四位命令、或四位地址、或四位数据。
FSPI	Fast SPI，GP-SPI2 输入输出信号的前缀。FSPI 总线信号可通过 GPIO 交换矩阵或 IO MUX 与 GPIO 管脚相连。

26.3 特性

GP-SPI2 具有以下特性：

- 用作主机或用作从机
- 支持半双工通信和全双工通信
- 支持 CPU 控制的传输类型以及 DMA 控制的传输类型
- 支持多种数据模式：
  - 1-bit SPI 模式
  - 2-bit Dual SPI 模式
  - 4-bit Quad SPI 模式
  - QPI 模式
- 时钟频率可配置：
  - 用作主机时：时钟频率可达 48 MHz
  - 用作从机时：时钟频率可达 32 MHz
- 数据长度可配置：
  - 在主机和从机 CPU 控制的传输下：数据长度为 1 ~ 64 B
  - 在主机 DMA 控制的单次传输下：数据长度为 1 ~ 32 KB
  - 在主机 DMA 控制的分段配置传输下：数据长度字节数无限制
  - 在从机 DMA 控制的单次或连续传输下：数据长度字节数无限制
- 读写数据的比特位顺序可配置
- 为 CPU 控制的传输和 DMA 控制的传输分别提供独立中断
- 时钟极性和相位可配置
- 四种 SPI 时钟模式：模式 0 ~ 模式 3
- 用作主机时，提供六条 CS 线：CS0 ~ CS5
- 支持访问 SPI 接口的传感器、显示屏控制器、flash 或 RAM 芯片

26.4 架构概览

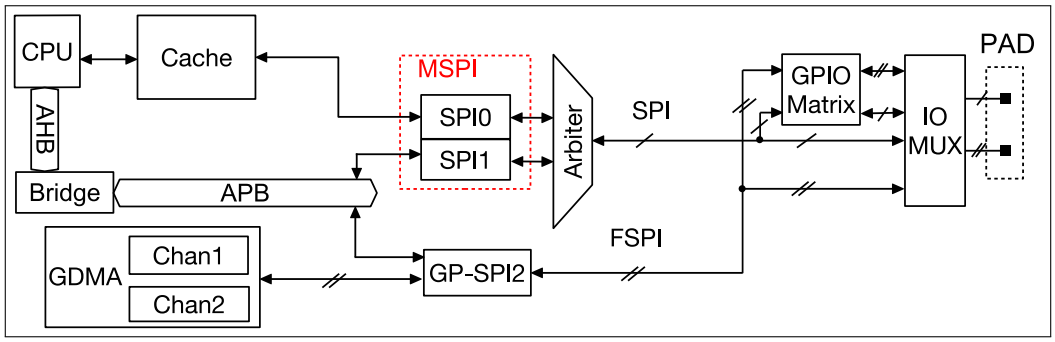


图 26-1. SPI 模块概览

图 26-1 所示为 SPI 模块的概览。GP-SPI2 通过以下方式与 SPI 设备进行数据交换：

- 在 CPU 控制的传输类型下：CPU <-> GP-SPI2 <-> SPI 设备
- 在 DMA 控制的传输类型下：GDMA <-> GP-SPI2 <-> SPI 设备

GP-SPI2 输入输出信号的前缀为 “FSPI”。FSPI 总线信号可通过 GPIO 交换矩阵或 IO MUX 与 GPIO 管脚相连。更多信息，见章节 6 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)*。

26.5 功能描述

26.5.1 数据模式

GP-SPI2 可配置成主机或从机模式，采用表 26-2 所示的数据模式与其他 SPI 设备进行通信。

表 26-2. GP-SPI2 支持的数据模式

数据模式		命令阶段	地址阶段	数据阶段
1-bit SPI		1-bit	1-bit	1-bit
Dual SPI	Dual Output Read	1-bit	1-bit	2-bit
	Dual I/O Read	1-bit	2-bit	2-bit
Quad SPI	Quad Output Read	1-bit	1-bit	4-bit
	Quad I/O Read	1-bit	4-bit	4-bit
QPI		4-bit	4-bit	4-bit

GP-SPI2 用作主机时，有效的阶段请参考第 26.5.8 小节；GP-SPI2 用作从机模式时，有效的阶段请参考第 26.5.9 小节。

26.5.2 FSPI 总线信号描述

FSPI 总线信号的功能描述如表 26-3。各种 SPI 模式下使用到的信号见表 26-4。

表 26-3. FSPI 总线信号功能描述

FSPI 总线信号	功能
FSPID	MOSI/SIO0[a]：串行输入输出数据，比特 0
FSPIQ	MISO/SIO1：串行输入输出数据，比特 1
FSPIWP	SIO2：串行输入输出数据，比特 2
FSPIHD	SIO3：串行输入输出数据，比特 3
FSPICLK	用作主机或从机时，输入输出时钟
FSPICS0	用作主机或从机时，输入输出片选信号
FSPICS1 ~ 5	用作主机时，输出片选信号

[a] SIO0：全称为 serial data input and output, bit0

表 26-4. 各种 SPI 模式下使用到的信号

FSPI 总线信号	用作主机						用作从机					
	1-bit SPI			2-bit Dual SPI	4-bit Quad SPI	QPI	1-bit SPI			2-bit Dual SPI	4-bit Quad SPI	QPI
	FD[1]	3-line HD[2]	4-line HD				FD	3-line HD	4-line HD			
FSPICLK	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS0	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
FSPICS1	Y	Y	Y	Y	Y	Y						
FSPICS2	Y	Y	Y	Y	Y	Y						
FSPICS3	Y	Y	Y	Y	Y	Y						
FSPICS4	Y	Y	Y	Y	Y	Y						
FSPICS5	Y	Y	Y	Y	Y	Y						
FSPID	Y	Y	(Y)[3]	Y[4]	Y[5]	Y	Y	Y	(Y)[6]	Y[7]	Y[8]	Y
FSPIQ	Y		(Y)[3]	Y[4]	Y[5]	Y	Y		(Y)[6]	Y[7]	Y[8]	Y
FSPIWP					Y[5]	Y					Y[8]	Y
FSPIHD					Y[5]	Y					Y[8]	Y

[1] FD：全双工

[2] HD：半双工

[3] 一次只使用两个信号中的一个

[4] 两个信号并行使用

[5] 四个信号并行使用

[6] 一次只使用两个信号中的一个

[7] 两个信号并行使用

[8] 四个信号并行使用

GoBack

26.5.3 数据位读/写顺序控制

用作主机时：

- GP-SPI2 主机发送的命令、地址和数据的位顺序由 SPI_WR_BIT_ORDER 控制；
- 接收数据的位顺序由 SPI_RD_BIT_ORDER 控制。

用作从机时：

- GP-SPI2 从机发送数据的位顺序由 SPI_WR_BIT_ORDER 控制；
- 从机接收的命令、地址和数据的位顺序由 SPI_RD_BIT_ORDER 控制。

表 26-5 所示为 SPI_RD/WR_BIT_ORDER 的功能。

表 26-5. GP-SPI2 用作主机和从机时的数据位控制

位模式	FSPI 总线信号	SPI_RD/WR_BIT_ORDER = 0 (MSB)	SPI_RD/WR_BIT_ORDER = 2 (MSB)	SPI_RD/WR_BIT_ORDER = 1 (LSB)	SPI_RD/WR_BIT_ORDER = 3 (LSB)
1-bit 模式	FSPID or FSPIQ	B7->B6->B5->B4->B3->B2->B1->B0	B7->B6->B5->B4->B3->B2->B1->B0	B0->B1->B2->B3->B4->B5->B6->B7	B0->B1->B2->B3->B4->B5->B6->B7
2-bit 模式	FSPIQ	B7->B5->B3->B1	B6->B4->B2->B0	B1->B3->B5->B7	B0->B2->B4->B6
	FSPID	B6->B4->B2->B0	B7->B5->B3->B1	B0->B2->B4->B6	B1->B3->B5->B7
4-bit 模式	FSPIHD	B7->B3	B4->B0	B3->B7	B0->B4
	FSPIWP	B6->B2	B5->B1	B2->B6	B1->B5
	FSPIQ	B5->B1	B6->B2	B1->B5	B2->B6
	FSPID	B4->B0	B7->B3	B0->B4	B3->B7

26.5.4 传输类型

GP-SPI2 用作主机或从机时所支持的传输类型见下表。

表 26-6. GP-SPI2 用作主机或从机时所支持的传输类型

模式		CPU 控制的单次传输	DMA 控制的单次传输	DMA 控制的分段配置传输	DMA 控制的从机连续传输
主机	全双工	Y	Y	Y	–
	半双工	Y	Y	Y	–
从机	全双工	Y	Y	–	Y
	半双工	Y	Y	–	Y

以下章节将详细介绍上表中所列的各种传输类型。

26.5.5 CPU 控制的数据传输

GP-SPI2 提供了 16 个 32-bit 的数据 buffer，即 SPI_W0_REG ~ SPI_W15_REG，见图 26-2。CPU 控制的传输表示在该次传输中发送的数据来自数据 buffer 或接收的数据存入数据 buffer。在这种传输类型下，每次传输事务均需要先配置相关寄存器，然后由 CPU 来触发。因此，CPU 控制的传输只能是单次传输，即仅包含一次传输事务。CPU 控制的传输支持全双工通信和半双工通信。

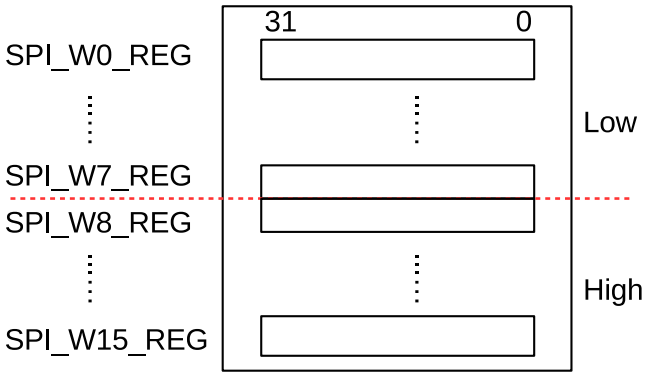


图 26-2. CPU 控制的传输中使用的数据 Buffer

26.5.5.1 CPU 控制的主机传输

GP-SPI2 用作主机时，无论全双工通信还是半双工通信，CPU 控制的数据传输均通过 SPI_W0_REG ~ SPI_W15_REG 完成。此外，用户可选择“高位模式”仅使用其中部分寄存器，具体见下方列表描述。

- **TX 数据：**
  - 未使能“高位模式”（SPI_USR_MOSI_HIGHPART 置 0）：此时，TX 数据取自 SPI_W0_REG ~ SPI_W15_REG，且每传输一个字节，取 TX 数据的地址即递增 1。**如果数据长度大于 64 字节，SPI_W0_REG ~ SPI_W15_REG 中的数据会被多次发送。**
    以 256 个字节为一个周期：
    * 字节 0 ~ 字节 63 依次发送 SPI_W0_REG ~ SPI_W15_REG 的数据。
    * 字节 64 ~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。

  * 字节 256 ~ 字节 319（另一组数据的前 64 个字节）重新依次发送 SPI_W0_REG ~ SPI_W15_REG 的数据。以此类推。

例如： 发送 258 个字节（字节 0 ~ 字节 257），则：

  * 字节 0 ~ 字节 63 依次发送 SPI_W0_REG ~ SPI_W15_REG 的数据。
  * 字节 64 ~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
  * 字节 256 ~ 字节 257 重新依次发送地址 0 (SPI_W0_REG[7:0]) 和地址 1 (SPI_W0_REG[15:8]) 的数据：
    - 字节 256 取数据的地址为 256 对 64 取模的结果（256 % 64 = 0），即地址 0 (SPI_W0_REG[7:0])。
    - 字节 257 取数据的地址为 257 对 64 取模的结果（257 % 64 = 1），即地址 1 (SPI_W0_REG[15:8])。

- 使能“高位模式”（SPI_USR_MOSI_HIGHPART 置 1）：此时，TX 数据取自 SPI_W8_REG ~ SPI_W15_REG，且每传输一个字节，取 TX 数据的地址即递增 1。**如果数据长度大于 32 字节，则 SPI_W8_REG ~ SPI_W15_REG 中的数据会被多次发送**。
  以 256 个字节为一个周期：
  * 字节 0 ~ 字节 31 依次发送 SPI_W8_REG ~ SPI_W15_REG 中的数据。
  * 字节 32 ~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
  * 字节 256 ~ 字节 287（另一组数据的前 32 个字节）重新依次发送 SPI_W8_REG ~ SPI_W15_REG 中的数据。以此类推。

  例如： 发送 258 个字节（字节 0 ~ 字节 257），则：
  * 字节 0 ~ 字节 31 依次发送 SPI_W8_REG ~ SPI_W15_REG 的数据。
  * 字节 32 ~ 字节 255 重复发送 SPI_W15_REG[31:24] 中的数据。
  * 字节 256 ~ 字节 257 重新依次发送地址 0 (SPI_W8_REG[7:0]) 和地址 1 (SPI_W8_REG[15:8]) 的数据：
    - 字节 256 取数据的地址为 256 对 32 取模的结果（256 % 32 = 0），即地址 0 (SPI_W8_REG[7:0])。
    - 字节 257 取数据的地址为 257 对 32 取模的结果（257 % 32 = 1），即地址 1 (SPI_W8_REG[15:8])。

- **RX 数据：**
  - 未使能“高位模式”（SPI_USR_MISO_HIGHPART 置 0）：此时，RX 数据存入 SPI_W0_REG ~ SPI_W15_REG，且每传输一个字节，存 RX 数据的地址即递增 1。**如果数据长度大于 64 字节，SPI_W0_REG ~ SPI_W15_REG 中的数据会被覆盖**。
    以 256 个字节为一个周期：
    * 字节 0 ~ 字节 63 依次存入 SPI_W0_REG ~ SPI_W15_REG。
    * 字节 64 ~ 字节 255 重复存入 SPI_W15_REG[31:24]。
    * 字节 256 ~ 字节 319（另一组数据的前 64 个字节）重新依次存入 SPI_W0_REG ~ SPI_W15_REG。以此类推。

例如： 接收 258 个字节（字节 0 ~ 字节 257），则：

* 字节 0 ~ 字节 63 依次存入 SPI_W0_REG ~ SPI_W15_REG。
* 字节 64 ~ 字节 255 重复存入 SPI_W15_REG[31:24]。
* 字节 256 ~ 字节 257 依次存入地址 0 (SPI_W0_REG[7:0]) 和地址 1 (SPI_W0_REG[15:8])：
  - 字节 256 存数据的地址为 256 对 64 取模的结果 (256 % 64 = 0)，即地址 0 (SPI_W0_REG[7:0])。
  - 字节 257 存数据的地址为 257 对 64 取模的结果 (257 % 64 = 1)，即地址 1 (SPI_W0_REG[15:8])。

– 使能“高位模式”（SPI_USR_MISO_HIGHPART 置 1）：此时，RX 数据存入 SPI_W8_REG ~ SPI_W15_REG，且每传输一个字节，存 RX 数据的地址即递增 1。**如果数据长度大于 32 字节，则 SPI_W8_REG ~ SPI_W15_REG 中的数据会被覆盖**。
以 256 个字节为一个周期：

* 字节 0 ~ 字节 31 依次存入 SPI_W8_REG ~ SPI_W15_REG。
* 字节 32 ~ 字节 255 重复存入 SPI_W15_REG[31:24]。
* 字节 256 ~ 字节 287（另一组数据的前 32 个字节）依次存入 SPI_W8_REG ~ SPI_W15_REG。以此类推。

例如： 接收 258 个字节（字节 256 ~ 字节 257），则：

* 字节 0 ~ 字节 31 依次存入 SPI_W8_REG ~ SPI_W15_REG。
* 字节 32 ~ 字节 255 重复存入 SPI_W15_REG[31:24]。
* 字节 256 ~ 字节 257 依次存入地址 0 (SPI_W8_REG[7:0]) 和地址 1 (SPI_W8_REG[15:8])：
  - 字节 256 存数据的地址为 256 对 32 取模的结果 (256 % 32 = 0)，即地址 0 (SPI_W8_REG[7:0])。
  - 字节 257 存数据的地址为 257 对 32 取模的结果 (257 % 32 = 1)，即地址 1 (SPI_W8_REG[15:8])。

说明：

- 上述的 TX/RX 数据均按字节寻址。
  – 如果未使能“高位模式”，地址 0 表示 SPI_W0_REG[7:0]，地址 1 表示 SPI_W0_REG[15:8]，以此类推。
  – 如果使能了“高位模式”，地址 0 表示 SPI_W8_REG[7:0]，地址 1 表示 SPI_W8_REG[15:8]，以此类推。
  最大地址为 SPI_W15_REG[31:24]。
- 为避免 TX/RX 数据传输错误，如 TX 数据重复发送或 RX 数据被覆盖等问题，请确保寄存器配置正确。

26.5.5.2 CPU 控制的从机传输

GP-SPI2 用作从机时，无论全双工通信或半双工通信，CPU 控制的数据传输均通过 SPI_W0_REG ~ SPI_W15_REG 完成，均采用按字节寻址。

- 全双工通信方式下：SPI_W0_REG ~ SPI_W15_REG 地址从 0 开始，且每传输一个字节，地址即递增 1。如果数据地址大于 63，则 SPI_W0_REG ~ SPI_W15_REG 中的数据会被覆盖。覆盖规律同主机模式下的

非“高位模式”。

- 半双工通信方式下：通信格式中的 ADDR 值即为 RX 数据或 TX 数据的起始地址，对应 SPI_W0_REG ~ SPI_W15_REG。每传输一个字节，则 RX 或 TX 地址即递增 1。如果地址大于 63（即大于最高地址：SPI_W15_REG[31:24]），SPI_W8_REG ~ SPI_W15_REG 中的数据会被覆盖。覆盖规律同主机模式下的“高位模式”。

用户可根据具体应用，将 SPI_W0_REG ~ SPI_W15_REG：

- 全部用作数据 buffer
- 部分用作数据 buffer，部分用作状态 buffer
- 全部用作状态 buffer

26.5.6 DMA 控制的数据传输

在 DMA 控制的传输中，GDMA RX 模块接收数据，GDMA TX 模块发送数据。GP-SPI2 用作主机和从机时均支持这种传输类型。

DMA 控制的传输可以是：

- 一次单次传输，仅包含一次传输事务。GP-SPI2 用作主机和从机时均支持这种单次传输。
- 分段配置传输，包含多个传输事务（即多个分段）。仅有 GP-SPI2 用作主机时支持这种分段配置传输。更多信息，见章节 26.5.8.5。
- 从机连续传输，包含多次传输事务。仅在 GP-SPI2 用作从机时支持这种从机连续传输。更多信息，见章节 26.5.9.3。

DMA 控制的传输只需由 CPU 触发一次即可完成多次传输事务。此类传输一旦被触发，GDMA 引擎从 DMA 链接的内存中发送数据，或将收到的数据存入 DMA 链接的内存中，无需 CPU 的干预。

DMA 控制的传输支持全双工通信、半双工通信以及章节 26.5.8 和章节 26.5.9 所描述的功能。同时，GDMA RX 模块与 GDMA TX 模块互不影响，即支持四种全双工通信：

- 在 DMA 控制模式下接收数据，并在 DMA 控制模式下发送数据；
- 在 DMA 控制模式下接收数据，但在 CPU 控制模式下发送数据；
- 在 CPU 控制模式下接收数据，但在 DMA 控制模式下发送数据；
- 在 CPU 控制模式下接收数据，并在 CPU 控制模式下发送数据。

26.5.6.1 GDMA 配置

- 选择 GDMA 通道 *n*，并配置 GDMA TX/RX 描述符，见章节 3 *通用 DMA 控制器 (GDMA)*；
- 置位 GDMA_INLINK_START_CH*n*/GDMA_OUTLINK_START_CH*n* 启动 GDMA RX/TX 引擎；
- 如果置位 GDMA_OUTLINK_RESTART_CH*n*，则在所有 GDMA TX buffer 用完之前，或在 GDMA TX 引擎重置之前，新的 TX buffer 将会被添加到最后使用中的 TX buffer 结尾；
- GDMA RX buffer 的链接方式与 GDMA TX buffer 的链接方式相同，可通过置位 GDMA_INLINK_START_CH*n*/GDMA_INLINK_RESTART_CH*n* 来实现；
- TX 数据长度和 RX 数据长度分别由 GDMA TX buffer 和 RX buffer 决定，长度范围为：0 ~ 32 KB；

- 启动 GDMA 前，先初始化 GDMA 接收链表 (inlink) 和发送链表 (outlink)。请置位寄存器 SPI_DMA_CONF_REG 中的 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA 位，否则读/写数据将存至或取自寄存器 SPI_W0_REG ~ SPI_W15_REG。

GP-SPI2 用作主机时，如果置位了 GDMA_IN_SUC_EOF_CH*n*_INT_ENA，则一次单次传输结束或一次分段配置传输结束，就会触发 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

GP-SPI2 用作从机时，如果置位了 GDMA_IN_SUC_EOF_CH*n*_INT_ENA，则下表中任一情况均可触发 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

表 26-7. GP-SPI2 用作从机时数据传输的中断触发条件

传输类型	控制位[1]	控制位[2]	触发条件
从机单次传输	0	0	一次单次传输结束即触发该中断。
	1	0	一次单次传输结束，或接收的数据长度等于 SPI_MS_DATA_BITLEN + 1，即触发该中断。
从机连续传输	0	1	正确接收 CMD7 或 End_SEG_TRANS 命令即触发该中断。
	1	1	正确接收 CMD7 或 End_SEG_TRANS 命令、或接收的数据长度等于 SPI_MS_DATA_BITLEN + 1，即触发该中断。

[1] SPI_RX_EOF_EN

[2] SPI_DMA_SLV_SEG_TRANS_EN

26.5.6.2 GDMA TX/RX Buffer 长度控制

配置的 GDMA TX/RX buffer 长度最好应等于实际传输数据的长度。如果配置的 GDMA TX/RX buffer 长度不等于实际传输数据的长度：

- 如果配置的 GDMA TX buffer 长度小于实际传输的数据长度，则多出来的数据将与最后传输的 TX buffer 数据相同。同时触发 SPI_OUTFIFO_EMPTY_ERR_INT 和 GDMA_OUT_EOF_CH*n*_INT 中断。
- 如果配置的 GDMA TX buffer 长度大于实际传输的数据长度，则 TX buffer 中的数据未被完全使用，即使稍后链接了新的 TX buffer，上个 TX buffer 中剩余的数据也将参与后续传输。请特别注意这一点，或保存未使用的数据并重置 DMA。
- 如果配置的 GDMA RX buffer 长度小于实际传输的数据长度，则多出来的数据将会丢失。同时触发 SPI_INFIFO_FULL_ERR_INT 和 SPI_TRANS_DONE_INT 中断。但不会触发 GDMA_IN_SUC_EOF_CH*n*_INT 中断。
- 如果配置的 GDMA RX buffer 长度大于实际传输的数据长度，则 RX buffer 未被使用的部分被丢弃，下次传输直接使用后面链接的 RX buffer。

26.5.7 GP-SPI2 用作主机和从机时的数据流控制

GP-SPI2 用作主机和从机时均支持 CPU 控制的数据传输和 DMA 控制的数据传输。CPU 控制的数据传输发生在 SPI_W0_REG ~ SPI_W15_REG 和外围 SPI 设备之间。DMA 控制的数据传输发生在配置好的 GDMA TX/RX buffer 和外围 SPI 设备之间。用户可在传输开始之前，配置 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA 来选择需要的传输类型。

26.5.7.1 GP-SPI2 功能块图

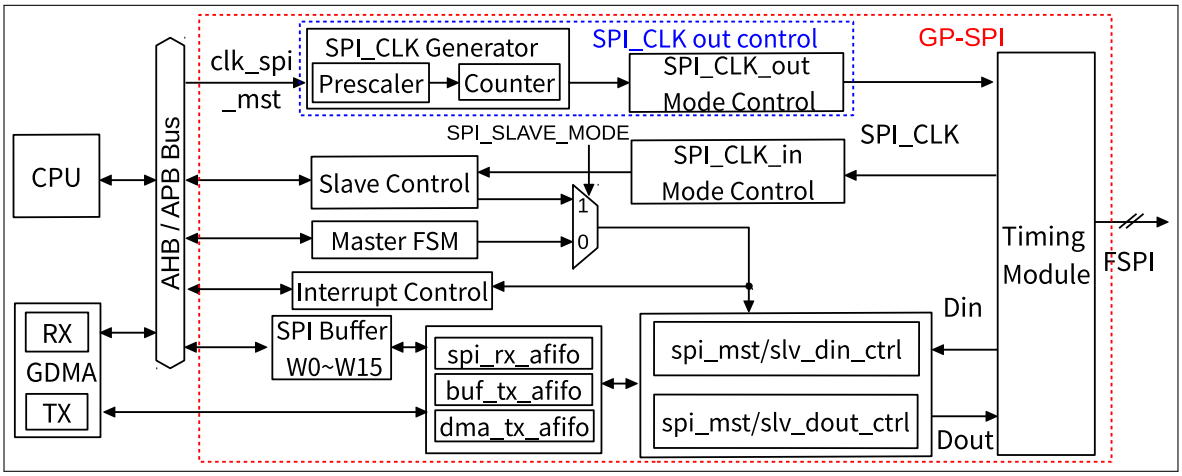


图 26-3. GP-SPI2 功能块图

图 26-3 所示为 GP-SPI2 主要的功能模块，包括：

- Master FSM：GP-SPI2 的主机状态机。GP-SPI2 用作主机时支持的所有功能，均由该状态机与寄存器共同控制。
- SPI Buffer: SPI_W0_REG ~ SPI_W15_REG，见图 26-2。CPU 控制的传输，数据在 SPI buffer 中准备。
- 时序模块 (Timing Module)：捕获 FSPI 总线上的数据。
- spi_mst/slv_din_ctrl 和 spi_mst/slv_dout_ctrl：用于将 TX/RX 数据转换成字节。
- spi_rx_afifo：暂存接收到的数据。
- buf_tx_afifo：暂存待发送的数据。
- dma_tx_afifo：暂存来自 GDMA 的数据。
- clk_spi_mst：GP-SPI2 模块时钟，由 PLL_CLK 分频所得。在 GP-SPI2 用作主机时用于生成数据传输以及从机所需的 SPI_CLK 信号。
- SPI_CLK 生成器 (SPI_CLK Generator)：对 clk_spi_mst 进行分频生成 SPI_CLK 信号。分频系数由 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 共同决定，见章节 26.7。
- SPI_CLK_out Mode Control：发送数据传输以及从机所需的 SPI_CLK 信号。
- SPI_CLK_in Mode Control：当 GP-SPI2 用作从机时，用于捕获 SPI 主机发出的 SPI_CLK 信号。

26.5.7.2 GP-SPI2 用作主机时的数据流控制

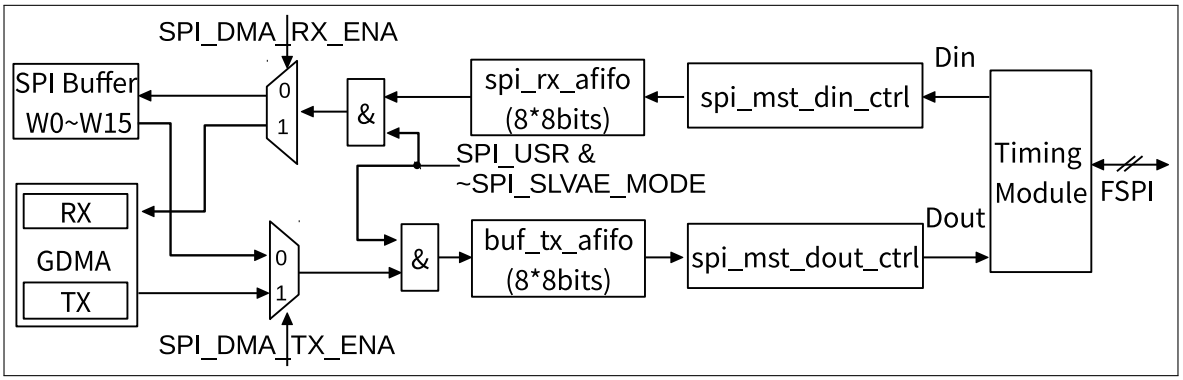


图 26-4. GP-SPI2 用作主机时的数据流控制

图 26-4 所示为 GP-SPI2 用作主机时的数据流。其控制逻辑如下：

- RX 数据：时序模块捕获 FSPI 总线上的数据，然后 *spi_mst_din_ctrl* 模块将比特数据转化为字节数据，暂存于 *spi_rx_afifo* 中，此后根据控制方式转存至不同的接收位置：
  - CPU 控制：转存至 SPI_W0_REG ~ SPI_W15_REG
  - DMA 控制：转存至 GDMA RX buffer
- TX 数据：*buf_tx_afifo* 模块暂存待发送数据。根据控制方式不同，待发送数据来自不同的位置：
  - CPU 控制：TX 数据来自 SPI_W0_REG ~ SPI_W15_REG
  - DMA 控制：TX 数据来自 GDMA TX buffer

buf_tx_afifo 中的数据会由时序模块以 1/2/4-bit 的模式发送出去。具体数据模式由 GP-SPI2 状态机控制。时序模块可用于时序补偿。

26.5.7.3 GP-SPI2 用作从机时的数据流控制

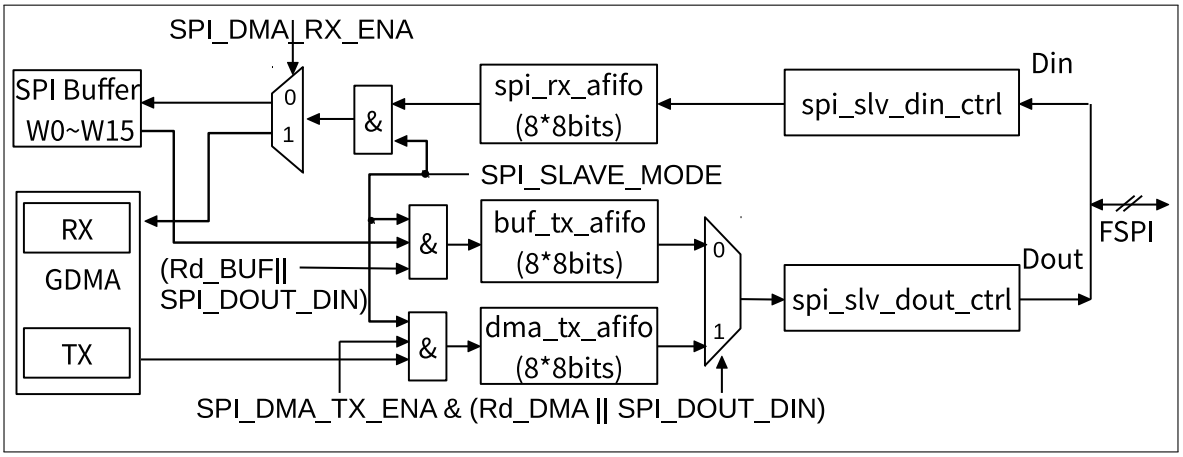


图 26-5. GP-SPI2 用作从机时的数据流控制

图 26-5 所示为 GP-SPI2 用作从机时的数据流控制。其控制逻辑如下：

- 在 CPU/DMA 控制的全双工/半双工传输下，当外部 SPI 主机发起 SPI 传输后，FSPI 总线上的数据将被捕获，然后由 *spi_slv_din_ctrl* 模块转换为字节，暂存于 *spi_rx_afifo* 中。
  - 在 CPU 控制的全双工传输中，暂存于 *spi_rx_afifo* 中的 RX 数据之后会被相继转存到 SPI_W0_REG ~ SPI_W15_REG。
  - 在半双工 Wr_BUF 传输中，收到地址值 (SLV_ADDR[7:0]) 后，*spi_rx_afifo* 中暂存的 RX 数据将转存至寄存器 SPI_W0_REG ~ SPI_W15_REG 的相应地址中。
  - 在 DMA 控制的全双工传输中，或在半双工 Wr_DMA 传输中，*spi_rx_afifo* 中暂存的 RX 数据将转存至配置好的 GDMA RX buffer 中。
- 在 CPU 控制的全双工/半双工传输中，待发送的数据暂存在 *buf_tx_afifo* 中；而在 DMA 控制的全双工/半双工传输中，待发送的数据暂存在 *dma_tx_afifo* 中。因此，在一次从机连续传输中，CPU 控制的 Rd_BUF 传输事务和 DMA 控制的 Rd_DMA 传输事务可同时发生。
  - 在 CPU 控制的全双工传输中，如果置位了 SPI_SLAVE_MODE 和 SPI_DOUTDIN，同时清零了 SPI_DMA_TX_ENA，则 SPI_W0_REG ~ SPI_W15_REG 中的数据将被转存至 *buf_tx_afifo* 中。
  - 在 CPU 控制的半双工传输中，如果置位了 SPI_SLAVE_MODE，清零了 SPI_DOUTDIN，且收到指令 Rd_BUF 和地址 SLV_ADDR[7:0]，则 SPI_W0_REG ~ SPI_W15_REG 相应地址中的数据将被转存至 *buf_tx_afifo* 中。
  - 在 DMA 控制的全双工传输中，如果置位了 SPI_SLAVE_MODE、SPI_DOUTDIN 和 SPI_DMA_TX_ENA，则 GDMA TX buffer 中的数据将被转存至 *dma_tx_afifo* 中。
  - 在 DMA 控制的半双工传输中，如果置位了 SPI_SLAVE_MODE，清零了 SPI_DOUTDIN，且收到指令 Rd_DMA，则 GDMA TX buffer 中的数据将被转存至 *dma_tx_afifo* 中。

  buf_tx_afifo 或 *dma_tx_afifo* 的数据将由 *spi_slv_dout_ctrl* 模块以 1/2/4-bit 的模式发送出去。

26.5.8 GP-SPI2 用作主机

清零 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位可将 GP-SPI2 配置成主机。此时，GP-SPI2 提供时钟信号（GP-SPI2 模块时钟的分频时钟）和六条 CS 线 (CS0 ~ CS5)。

说明：

- 数据以字节为单位进行传输，否则多余的位将丢失。此处多余的位表示总位长对 8 取模的结果。
- 如果需要传输非字节比特，推荐使用 CMD 阶段或 ADDR 阶段来实现。

26.5.8.1 主机状态机

GP-SPI2 用作主机时，状态机在数据传输中控制其各个阶段，包括配置阶段 (CONF)、准备阶段 (PREP)、命令阶段 (CMD)、地址阶段 (ADDR)、等待阶段 (DUMMY)、发送数据阶段 (DOUT) 和接收数据阶段 (DIN)。GP-SPI2 主要用于访问 1/2/4-bit SPI 设备，如 flash、外部 RAM 等。因此，GP-SPI2 各个阶段的命名规则应与 flash 以及外部 RAM 的时序名称保持一致。每个阶段的描述如下，GP-SPI2 状态机的工作流程见图 26-6。

1. 空闲阶段 (IDLE)：GP-SPI2 未处于工作状态或正在用作从机。
2. 配置阶段 (CONF)：仅用于DMA 控制的分段配置传输。置位 SPI_USR 和 SPI_USR_CONF 使能该阶段。如果未使能该阶段，则说明当前传输为单次传输。

3. 准备阶段 (PREP)：准备 SPI 传输事务，控制 SPI CS 建立时间。置位 SPI_USR 和 SPI_CS_SETUP 使能该阶段。
4. 命令阶段 (CMD)：发送命令序列。置位 SPI_USR 和 SPI_USR_COMMAND 使能该阶段。
5. 地址阶段 (ADDR)：发送地址序列。置位 SPI_USR 和 SPI_USR_ADDR 使能该阶段。
6. 等待阶段 (DUMMY)：发送 DUMMY 序列。置位 SPI_USR 和 SPI_USR_DUMMY 使能该阶段。
7. 传输数据阶段 (DATA)：传输数据。
    - DOUT：发送数据。置位 SPI_USR 和 SPI_USR_MOSI 使能该阶段。
    - DIN：接收数据。置位 SPI_USR 和 SPI_USR_MISO 使能该阶段。
8. 结束阶段 (DONE)：控制 SPI CS 保持时间。置位 SPI_USR 使能该阶段。

注意：

主机状态机启动必须配置 SPI_USR。SPI_MST_FD_WAIT_DMA_TX_DATA 可以控制 SPI_USR 生效的时刻：

- 当 SPI_MST_FD_WAIT_DMA_TX_DATA 为 0 时，状态机相关状态在配置 SPI_USR 和其他控制寄存器后立即生效；
- 当 SPI_MST_FD_WAIT_DMA_TX_DATA 为 1 时，如果状态机也配置了 DOUT 状态，则会等待 *buf_tx_fifo* 中至少有数据后，SPI_USR 和其他控制寄存器才会生效，状态机开始启动。

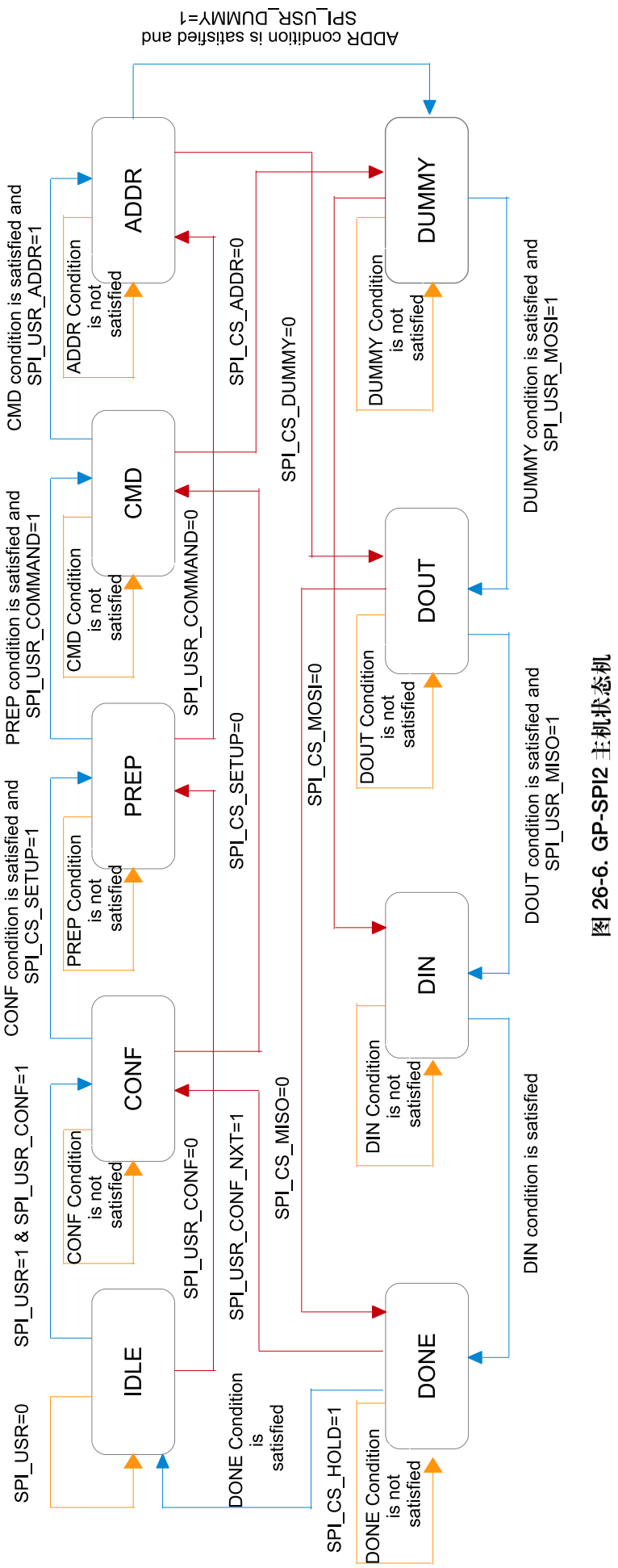


图 26-6. GP-SPI2 主机状态机

图标说明：

- ——：表示相应的状态条件不满足，重复当前状态。
- ——：表示相应的寄存器已配置，状态条件已满足，将进行下一个状态。
- ——：表示相应的寄存器未配置，跳过下一个状态，或跳过后续多个状态。

上图中的各个状态条件描述如下：

- CONF condition: gpc[17:0] >= SPI_CONF_BITLEN[17:0]
- PREP condition: gpc[4:0] >= SPI_CS_SETUP_TIME[4:0]
- CMD condition: gpc[3:0] >= SPI_USR_COMMAND_BITLEN[3:0]
- ADDR condition: gpc[4:0] >= SPI_USR_ADDR_BITLEN[4:0]
- DUMMY condition: gpc[7:0] >= SPI_USR_DUMMY_CYCLELEN[7:0]
- DOUT condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DIN condition: gpc[17:0] >= SPI_MS_DATA_BITLEN[17:0]
- DONE condition: (gpc[4:0] >= SPI_CS_HOLD_TIME[4:0] || SPI_CS_HOLD == 1'b0)

状态机中用到了一个计数器 (gpc[17:0]) 来控制每个状态的周期长度。CONF、PREP、CMD、ADDR、DUMMY、DOUT 和 DIN 各状态可单独使能或禁用，也可以单独配置其周期长度。

26.5.8.2 状态控制和位模式控制寄存器

概述

表 26-8 列出了与 GP-SPI2 状态控制相关的寄存器配置。如需使能 GP-SPI2 的 QPI 模式，请置位寄存器 SPI_USER_REG 中 SPI_QPI_MODE 位。

表 26-8. 1/2/4-bit 模式下状态控制寄存器

状态	1-bit FSPI 控制寄存器	2-bit FSPI 控制寄存器	4-bit FSPI 控制寄存器
CMD	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_DUAL SPI_USR_COMMAND	SPI_USR_COMMAND_VALUE SPI_USR_COMMAND_BITLEN SPI_FCMD_QUAD SPI_USR_COMMAND
ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_DUAL	SPI_USR_ADDR_VALUE SPI_USR_ADDR_BITLEN SPI_USR_ADDR SPI_FADDR_QUAD
DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY	SPI_USR_DUMMY_CYCLELEN SPI_USR_DUMMY
DIN	SPI_USR_MISO SPI_MS_DATA_BITLEN	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_DUAL	SPI_USR_MISO SPI_MS_DATA_BITLEN SPI_FREAD_QUAD
DOUT	SPI_USR_MOSI SPI_MS_DATA_BITLEN	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_DUAL	SPI_USR_MOSI SPI_MS_DATA_BITLEN SPI_FWRITE_QUAD

如表 26-8 所示，如果希望在表格第一栏所示的状态中将 FSPI 总线设置为相应的位模式（见表头），则需要配置该行中每一单元格的寄存器。

配置

例如，当 GP-SPI2 读取数据时，且希望实现：

- CMD 为 1-bit 模式
- ADDR 为 2-bit 模式
- DUMMY 为 8 个时钟周期
- DIN 为 4-bit 模式

则具体的寄存器配置如下：

1. 配置 CMD 状态相关寄存器。
    - 配置 SPI_USR_COMMAND_VALUE 为需要的命令值；
    - 配置 SPI_USR_COMMAND_BITLEN。SPI_USR_COMMAND_BITLEN 为所需要的命令位长 - 1；
    - 置位 SPI_USR_COMMAND；
    - 清除 SPI_FCMD_DUAL 和 SPI_FCMD_QUAD。
2. 配置 ADDR 状态相关寄存器。
    - 配置 SPI_USR_ADDR_VALUE 为需要的地址值；
    - 配置 SPI_USR_ADDR_BITLEN。SPI_USR_COMMAND_BITLEN 为所需要的地址位长 - 1；
    - 置位 SPI_USR_ADDR 和 SPI_FADDR_DUAL；
    - 清除 SPI_FADDR_QUAD。
3. 配置 DUMMY 状态相关寄存器。
    - 在 SPI_USR_DUMMY_CYCLELEN 中配置 DUMMY 周期，其中 SPI_USR_DUMMY_CYCLELEN 的值等于 DUMMY 阶段所需要的时钟周期数 - 1；
    - 置位 SPI_USR_DUMMY。
4. 配置 DIN 状态相关寄存器。
    - 在 SPI_MS_DATA_BITLEN 中配置读数据的位长；SPI_MS_DATA_BITLEN 的值等于所需要的位长 - 1；
    - 置位 SPI_FREAD_QUAD 和 SPI_USR_MISO；
    - 清除 SPI_FREAD_DUAL；
    - 如果选择了 DMA 控制的传输类型，则需要配置 GDMA。如果选择了 CPU 控制的传输类型，则无需任何操作；
5. 清除 SPI_USR_MOSI；
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer。
7. 置位 SPI_USR 开始 GP-SPI2 传输。

写数据时 (DOUT)，需要配置 SPI_USR_MOSI，同时清除 SPI_USR_MISO。输出数据的位长等于 SPI_MS_DATA_BITLEN 加 1。在 CPU 控制的传输类型下，需要在数据 buffer (SPI_W0_REG ~ SPI_W15_REG)

中准备数据；在 DMA 控制的数据传输下，需要在 GDMA TX buffer 中准备输出数据。字节顺序从 LSB (byte 0) 到 MSB 递增。

需特别注意 SPI_USR_COMMAND_VALUE 中的命令值以及 SPI_USR_ADDR_VALUE 中的地址值。

命令值的配置如下：

表 26-9. 命令值的发送顺序

COMMAND_BITLEN[1]	COMMAND_VALUE[2]	BIT_ORDER[3]	命令值的发送顺序
0 - 7	[7:0]	1	先发送 COMMAND_VALUE[COMMAND_BITLEN:0]。
		0	先发送 COMMAND_VALUE[7:7 - COMMAND_BITLEN]。
8 - 15	[15:0]	1	先发送 COMMAND_VALUE[7:0]，再发送 COMMAND_VALUE[COMMAND_BITLEN:8]。
		0	先发送 COMMAND_VALUE[7:0]，再发送 COMMAND_VALUE[15:15 - COMMAND_BITLEN]。

[1] SPI_USR_COMMAND_BITLEN：用于配置命令的位长。

[2] SPI_USR_COMMAND_VALUE：命令值写入的字段，见上表。

[3] SPI_WR_BIT_ORDER：0：先发送 LSB；1：先发送 MSB。

地址值配置如下：

表 26-10. 地址值的发送顺序

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	地址发送顺序
0 - 7	[31:24]	1	先发送 ADDR_VALUE[ADDR_BITLEN + 24:24]。
		0	先发送 ADDR_VALUE[31:31 - ADDR_BITLEN]。
8 - 15	[31:16]	1	先发送 ADDR_VALUE[31:24]，再发送 ADDR_VALUE[ADDR_BITLEN + 8:16]。
		0	先发送 ADDR_VALUE[31:24]，再发送 ADDR_VALUE[23:31 - ADDR_BITLEN]。
16 - 23	[31:8]	1	先发送 ADDR_VALUE[31:16]，再发送 ADDR_VALUE[ADDR_BITLEN - 8:8]。
		0	先发送 ADDR_VALUE[31:16]，再发送 ADDR_VALUE[15:31 - ADDR_BITLEN]。
24 - 31	[31:0]	1	先发送 ADDR_VALUE[31:8]，再发送 ADDR_VALUE[ADDR_BITLEN - 24:0]。
		0	先发送 ADDR_VALUE[31:8]，再发送 ADDR_VALUE[7:31 - ADDR_BITLEN]。

[1] SPI_USR_ADDR_BITLEN：用于配置地址值的位长。

[2] SPI_USR_ADDR_VALUE：地址值写入的字段，见上表。

[3] SPI_WR_BIT_ORDER：0：先发送 LSB；1：先发送 MSB。

26.5.8.3 主机全双工通信（仅支持 1-bit 模式）

概述

GP-SPI2 支持 SPI 全双工通信。在这种通信方式下，SPI 主机提供 CLK 和 CS 信号，然后与从机使用 1-bit 模式同时交换数据：MOSI（FSPID，发送），MISO（FSPIQ，接收）。用户可通过置位寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位使能全双工通信。GP-SPI2 与从机使用全双工通信时的连接方式见图 26-7。

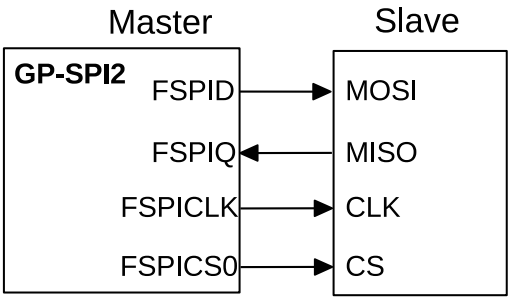


图 26-7. GP-SPI2 主机使用全双工模式与 SPI 从机通信框图

在全双工通信中，CMD、ADDR、DUMMY、DOUT 和 DIN 各个状态的具体行为可配置。通常，全双工通信跳过 CMD、ADDR 和 DUMMY 状态。传输数据的位长可在 SPI_MS_DATA_BITLEN 中配置。通信中使用的实际位长等于 (SPI_MS_DATA_BITLEN + 1)。

配置

按照以下操作步骤，开始数据传输：

- 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
- 配置 AHB、APB 时钟（即 AHB_CLK、APB_CLK，见章节 7 复位和时钟），并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
- 置位 SPI_DOUTDIN 同时清除 SPI_SLAVE_MODE，使能主机全双工通信方式；
- 配置表 26-8 中所列的 GP-SPI2 寄存器；
- 配置 SPI CS 建立时间和保持时间，见章节 26.6；
- 设置 FSPICLK 的极性，见章节 26.7；
- 根据选定的传输类型准备数据：
  - 如果选择的传输类型为 CPU 控制的 MOSI 传输，则需要在 SPI_W0_REG ~ SPI_W15_REG 中准备数据。
  - 如果选择了 DMA 控制的传输类型，则需要：
    * 配置 SPI_DMA_RX_ENA/SPI_DMA_TX_ENA；
    * 配置 GDMA TX/RX 链表；
    * 启动 GDMA TX/RX 引擎，更多描述见章节 26.5.6 和章节 26.5.7。
- 配置中断，然后等待 SPI 从机做好传输准备；
- 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
- 置位寄存器 SPI_CMD_REG 中 SPI_USR 位，开始数据传输，然后等待之前配置的中断。

26.5.8.4 主机半双工通信（支持 1/2/4-bit 模式）

概述

在半双工通信下，GP-SPI2 发送 CLK 和 CS 信号。在同一时刻，SPI 主机或从机只能有一个可以发送数据，另一个接收数据。用户可通过清除寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位使能半双工通信。SPI 半双工通信的通用格式为 CMD + [ADDR +] [DUMMY +] [DOUT or DIN]。其中，ADDR、DUMMY、DOUT 和 DIN 状态非必选，可单独禁用或启用。

如章节 26.5.8.2 所述，CMD、ADDR、DUMMY、DOUT 和 DIN 各个状态的周期、具体值和并行总线位模式等可独立配置。更多寄存器配置信息，见表 26-8。

半双工 GP-SPI2 的详细属性如下：

1. CMD：0 ~ 16 位，主机发送，从机接收 (MOSI)。
2. ADDR：0 ~ 32 位，主机发送，从机接收 (MOSI)。
3. DUMMY：0 ~ 256 个 FSPICLK 周期，主机发送，从机接收。
4. DOUT：在 CPU 控制的传输中，可传输 0 ~ 512 位（64 字节）数据；在 DMA 控制的传输中，可传输 0 ~ 256 Kbit (32 KB)。主机发送，从机接收。
5. DIN：在 CPU 控制的传输中，可传输 0 ~ 512 位（64 字节）数据；在 DMA 控制的传输中，可传输 0 ~ 256 Kbit (32 KB)。主机接收，从机发送。

配置

具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB、APB 时钟（即 AHB_CLK、APB_CLK），并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
3. 清除 SPI_DOUTDIN 和 SPI_SLAVE_MODE 位，使能主机半双工通信方式；
4. 配置表 26-8 中所列的 GP-SPI2 寄存器；
5. 配置 SPI CS 建立时间和保持时间，见章节 26.6；
6. 设置 FSPICLK 的极性，见章节 26.7；
7. 根据选定的传输类型准备数据：
   - 如果选择的传输类型为 CPU 控制的 MOSI 传输，则需要在 SPI_W0_REG ~ SPI_W15_REG 中准备数据。
   - 如果选择了 DMA 控制的传输类型，则需要：
     - 配置 SPI_DMA_RX_ENA/SPI_DMA_TX_ENA；
     - 配置 GDMA TX/RX 链表；
     - 启动 GDMA TX/RX 引擎，更多描述见章节 26.5.6 和章节 26.5.7。
8. 配置中断，然后等待 SPI 从机做好传输准备；
9. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
10. 置位寄存器 SPI_CMD_REG 中 SPI_USR 位，开始数据传输，然后等待之前配置的中断。

应用示例

以下示例展示了 GP-SPI2 如何在主机半双工模式下访问 flash 和外部 RAM。

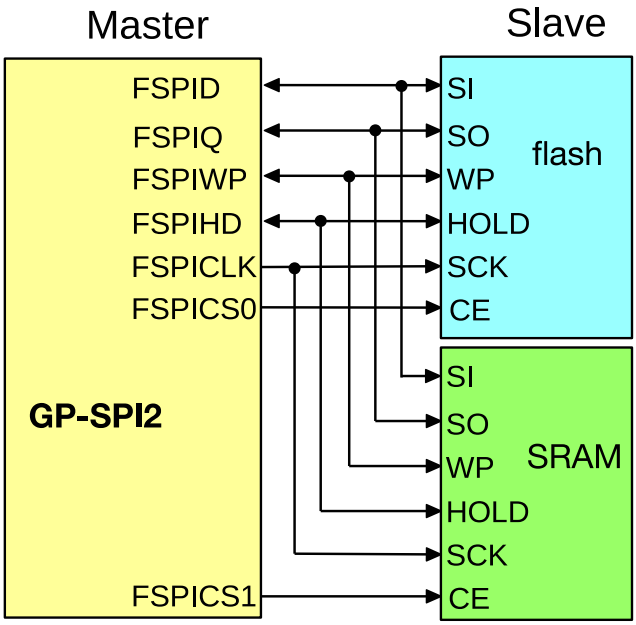


图 26-8. 4-bit 模式下 GP-SPI2 与 Flash 以及外部 RAM 的连接方式

图 26-9 所示为 GP-SPI2 按照标准 flash 规范进行 Quad I/O Read 操作。其他 GP-SPI2 命令序列可以根据 SPI 从机的要求实现。

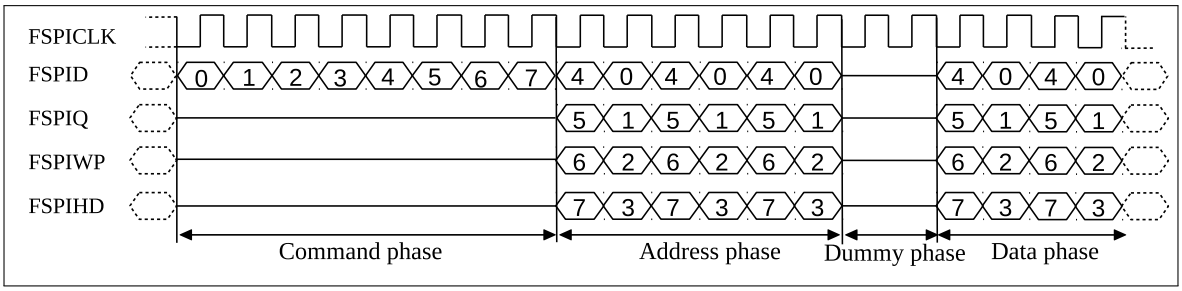


图 26-9. GP-SPI2 发送到 Flash 的 SPI Quad I/O 命令序列

26.5.8.5 DMA 控制的分段配置传输

说明:
注意，由于跳过 CONF 阶段即可实现单次传输，因此不再另起章节单独介绍如何在 GP-SPI2 用作主机时配置单次传输。

概述

GP-SPI2 用作主机时，可采用 DMA 控制的分段配置传输。

DMA 控制的主机传输可以是：

- 一次单次传输，仅包含一次传输事务。
- 分段配置传输，包括多个传输事务（即多个分段）。

如果选择了分段配置传输，则在每个分段中，寄存器均可单独配置。在分段配置传输下，仅需 CPU 触发一次，即可完成多次传输事务。具体工作流程见图 26-10。

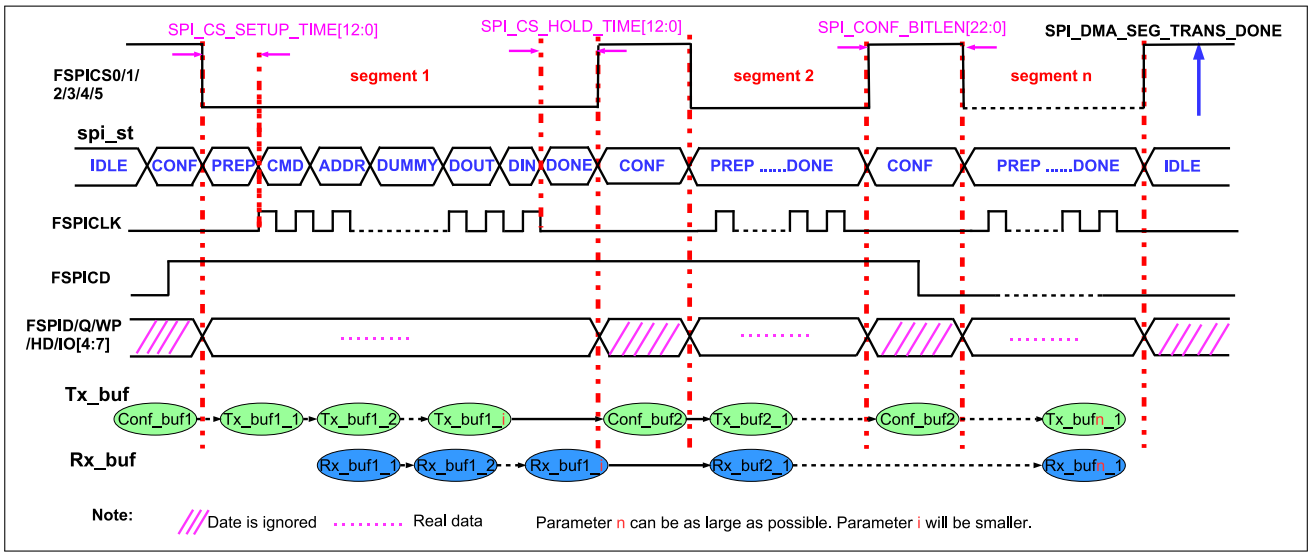


图 26-10. DMA 控制的分段配置传输

如图 26-10 所示，在分段配置传输中的某个单次传输事务 (segment *n*) 开始前，GP-SPI2 可在 CONF 阶段将寄存器重新按照 Conf_buf*n* 定义的内容进行配置。

建议为每个传输事务的 CONF 阶段提供单独的 GDMA CONF 链表和 CONF buffer（即图 26-10 中的 Conf_buf*i*）。GDMA TX 链表将所有的 CONF buffer 和 TX data buffer（即图 26-10 中的 Tx_buf*i*）链接起来，因此可以独立控制每个传输事务中的 FSPI 总线行为。

例如，在一次完整的分段配置传输中，传输事务 *i*、传输事务 *j* 和传输事务 *k* 可分别配置为全双工、半双工 MISO 和半双工 MOSI 模式。*i*、*j* 和 *k* 均为整数变量，代表传输事务的编号。

同时，每个传输事务中，GP-SPI2 所使用到的各个阶段、各个阶段的相关值和 FSPI 总线周期长、以及 GDMA 行为等，均可独立配置。当整个 DMA 控制的分段配置传输（包括多个传输事务）完成后，即触发 GP-SPI2 中断 SPI_DMA_SEG_TRANS_DONE_INT。

配置

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB、APB 时钟（即 AHB_CLK、APB_CLK），并为 GP-SPI2 配置模块时钟 (clk_spi_mst)；
3. 清除 SPI_DOUTDIN 和 SPI_SLAVE_MODE 位，使能主机半双工通信方式；
4. 配置表 26-8 中所列的 GP-SPI2 寄存器；
5. 配置 SPI CS 建立时间和保持时间，见章节 26.6；
6. 设置 FSPICLK 的相位和极性，见章节 26.7；
7. 为每个传输事务准备 GDMA CONF buffer 描述符和 TX data 描述符（可选）。把 CONF buffer 描述符和几次传输事务需要的 TX buffer 链接成一个链表；
8. 同样，为每个传输事务准备 RX buffer 描述符，并链接成一个链表；
9. 在该 DMA 控制的分段配置传输开始之前，为每个传输事务配置所需的 CONF buffer、TX buffer 和 RX buffer；

10. 配置 GDMA_OUTLINK_ADDR_CH*n* 指向 CONF 和 TX buffer 描述符链表的首地址，之后置位 GDMA_OUTLINK_START_CH*n*，启动 TX GDMA；
11. 清除 SPI_DMA_CONF_REG 中 SPI_RX_EOF_EN 位。配置 GDMA_INLINK_ADDR_CH*n* 指向 RX buffer 描述符链表的首地址，之后置位 GDMA_INLINK_START_CH*n* 启动 RX GDMA；
12. 置位 SPI_USR_CONF 使能 CONF 阶段；
13. 置位 SPI_DMA_SEG_TRANS_DONE_INT_ENA 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。如需配置其他中断，请参考章节 26.8；
14. 等待所有从机做好传输准备；
15. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
16. 置位 SPI_USR 开始本次 DMA 控制的分段配置传输；
17. 等待 SPI_DMA_SEG_TRANS_DONE_INT 中断，即 DMA 分段配置传输结束，数据已存储至相应内存。

配置 CONF Buffer 和 Magic 值

在 GP-SPI2 分段配置传输中，仅有较上次传输事务有变动的寄存器会在 CONF 阶段被重新配置。为节省时间和芯片资源，其他寄存器配置则保持不变。

GDMA CONF buffer*i* 中第一个字，即 SPI_BIT_MAP_WORD，记录传输事务 *i* 中，寄存器是否有改动。SPI_BIT_MAP_WORD 和待更新的 GP-SPI2 寄存器的对应关系见表 26-11，即位图 (BM) 表。如果位图表中某一位为 1，则在此次传输事务中，该位对应寄存器的值将被更新。如果其他寄存器不需要修改，则位图表中相应位应置为 0。

表 26-11. CONF 阶段 BM 位图

BM 位	寄存器	BM 位	寄存器
0	SPI_ADDR_REG	7	SPI_MISC_REG
1	SPI_CTRL_REG	8	SPI_DIN_MODE_REG
2	SPI_CLOCK_REG	9	SPI_DIN_NUM_REG
3	SPI_USER_REG	10	SPI_DOUT_MODE_REG
4	SPI_USER1_REG	11	SPI_DMA_CONF_REG
5	SPI_USER2_REG	12	SPI_DMA_INT_ENA_REG
6	SPI_MS_DLEN_REG	13	SPI_DMA_INT_CLR_REG

所有待修改的寄存器新值应紧跟在 SPI_BIT_MAP_WORD 之后，在 CONF buffer 中用连续的字表示。

为确保每个 CONF buffer 中内容正确，SPI_BIT_MAP_WORD[31:28] 位将用作 Magic 值，与寄存器 SPI_SLAVE_REG 中 SPI_DMA_SEG_MAGIC_VALUE 的值进行比较。SPI_DMA_SEG_MAGIC_VALUE 的值应在此 DMA 控制的分段配置传输开始之前配置，且在任何传输事务过程中均不可更改。

- 经比较，如果 SPI_BIT_MAP_WORD[31:28] == SPI_DMA_SEG_MAGIC_VALUE，则分段配置传输继续正常进行，整个传输过程结束则触发 SPI_DMA_SEG_TRANS_DONE_INT 中断。
- 如果 SPI_BIT_MAP_WORD[31:28] != SPI_DMA_SEG_MAGIC_VALUE，则 GP-SPI2 状态，即 spi_st 将返回至 IDLE 状态，分段配置传输立即结束。同时触发 SPI_DMA_SEG_TRANS_DONE_INT 中断，SPI_SEG_MAGIC_ERR_INT_RAW 位也将置 1。

CONF Buffer 配置示例

在一次分段配置传输中，传输事务 *i* 有 SPI_ADDR_REG、SPI_CTRL_REG、SPI_CLOCK_REG、SPI_USER_REG 和 SPI_USER1_REG 五个寄存器需要更新，则其 CONF buffer*i* 具体的配置示例见表 26-12 和表 26-13 。

表 26-12. 传输事务 *i* 中 CONF buffer*i* 配置示例

CONF buffer*i*	说明
SPI_BIT_MAP_WORD	Buffer 中的第一个字。如果 SPI_DMA_SEG_MAGIC_VALUE 设置为 0xA，则本示例中该字的值为 0xA000001F。由表 26-13 可知，被置 1 的位有第 0、1、2、3 和 4 位，表示下列寄存器将被更新
SPI_ADDR_REG	CONF buffer*i* 的第二个字，存储 SPI_ADDR_REG 寄存器的更新值
SPI_CTRL_REG	CONF buffer*i* 的第三个字，存储 SPI_CTRL_REG 寄存器的更新值
SPI_CLOCK_REG	CONF buffer*i* 的第四个字，存储 SPI_CLOCK_REG 寄存器的更新值
SPI_USER_REG	CONF buffer*i* 的第五个字，存储 SPI_USER_REG 寄存器的更新值
SPI_USER1_REG	CONF buffer*i* 的第六个字，存储 SPI_USER1_REG 寄存器的更新值

表 26-13. BM 位图与待更新的寄存器

位	值	寄存器	位	值	寄存器
0	1	SPI_ADDR_REG	7	0	SPI_MISC_REG
1	1	SPI_CTRL_REG	8	0	SPI_DIN_MODE_REG
2	1	SPI_CLOCK_REG	9	0	SPI_DIN_NUM_REG
3	1	SPI_USER_REG	10	0	SPI_DOUT_MODE_REG
4	1	SPI_USER1_REG	11	0	SPI_DMA_CONF_REG
5	0	SPI_USER2_REG	12	0	SPI_DMA_INT_ENA_REG
6	0	SPI_MS_DLEN_REG	13	0	SPI_DMA_INT_CLR_REG

说明

使用 DMA 分段配置传输功能时，应注意以下寄存器相关位：

- SPI_USR_CONF：在置位 SPI_USR 之前，需先置位 SPI_USR_CONF，以使能本次传输。
- SPI_USR_CONF_NXT：如果传输事务 *i* 不是本次 DMA 控制的分段配置传输中的最后一次传输事务，则需要置位 SPI_USR_CONF_NXT。
- SPI_CONF_BITLEN：此外，在每个单独的传输事务中，GP-SPI2 的 CS 建立时间和保持时间可独立编程，更多配置信息见章节 26.6。在每次传输事务中，CS 保持高电平的时长约为：

  $$(SPI\_CONF\_BITLEN + 5) \times T_{AHB\_CLK}$$

  f_{APB_CLK} 为 32 MHz 时，CONF 阶段的 CS 高电平时长可配置为 156.25 ns ~ 8.1918 ms。如果 SPI_CONF_BITLEN 大于 0x3FFFA，(SPI_CONF_BITLEN + 5) 将溢出 (0x40000 - SPI_CONF_BITLEN - 5)。

26.5.9 GP-SPI2 用作从机

GP-SPI2 可用作从机与另一 SPI 主机进行通信。用作从机时，GP-SPI2 支持特定格式的 1-bit SPI、2-bit Dual SPI、4-bit Quad SPI 和 QPI 模式。用户可置位寄存器 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位使能 GP-SPI2 从机。

在传输过程中，CS 信号应保持低电平，CS 信号的下降沿和上升沿代表一次传输的开始和结束。数据以字节为单位进行传输，否则多余的位将丢失。此处多余的位表示总位长对 8 取模的结果。

26.5.9.1 可配置的通信格式

GP-SPI2 用作从机时支持全双工通信和半双工通信。用户可配置寄存器 SPI_USER_REG 中 SPI_DOUTDIN 位选择需要的通信方式。

全双工通信下，传输一开始，则数据同时输入和输出。此时，所有数据位均被视为输入/输出数据，即不需要命令、地址或 DUMMY 阶段。传输结束即触发 SPI_TRANS_DONE_INT 中断。

在半双工通信下，通信格式为 CMD+ADDR+DUMMY+DATA (DIN or DOUT)。

- “DIN” 表示 SPI 主机从 GP-SPI2 中读取数据；
- “DOUT” 表示 SPI 主机向 GP-SPI2 中写入数据。

每个阶段的详细特性如下：

1. CMD：
   - 表明 SPI 从机用于何种功能；
   - 一个字节，主机输出，从机输入；
   - 仅支持表 26-14 和表 26-15 所列的命令值；
   - 以 1-bit SPI 模式或 4-bit QPI 模式发送。
2. ADDR：
   - 在 CPU 控制的传输中，可以为 Wr_BUF 和 Rd_BUF 命令提供地址，或在其他命令中用作占位符，具体由应用定义；
   - 一个字节，主机输出，从机输入；
   - 可根据命令，以 1-bit、2-bit 或 4-bit 模式发送；
3. DUMMY：
   - DUMMY 的值无实际意义；SPI 从机在这个阶段准备数据；
   - FSPI 总线的位模式在这里也没有实际意义；
   - 持续八个 SPI_CLK 时钟周期。
4. DIN 或 DOUT：
   - 在 CPU 控制的传输下，可传输 0 ~ 64 字节数据；在 DMA 控制的传输下，传输数据长度无限制。
   - 可根据具体的 CMD 值，以 1-bit、2-bit 或 4-bit 模式发送。

> **说明：**
> 半双工通信下，ADDR 和 DUMMY 阶段不可跳过。

半双工传输结束后，传输的 CMD 和 ADDR 的值分别锁存至 SPI_SLV_LAST_COMMAND 和 SPI_SLV_LAST_ADDR。如果 GP-SPI2 不支持传输的 CMD 值，SPI_SLV_CMD_ERR_INT_RAW 将被置位。SPI_SLV_CMD_ERR_INT_RAW 仅可由软件清零。

26.5.9.2 半双工通信支持的 CMD 值

在半双工传输中，CMD 定义的值将决定传输类型。不支持的 CMD 值及其相关数据传输均被忽略，且 SPI_SLV_CMD_ERR_INT_RAW 将被置 1。传输格式为：CMD（8 位）+ ADDR（8 位）+ DUMMY（8 个

SPI_CLK 周期）+ DATA，其中，DATA 的单位为字节。CMD[3:0] 的详细说明如下：

- 0x1 (Wr_BUF)：CPU 控制的写操作。主机发送数据，GP-SPI2 接收数据。数据将存储至相应地址的寄存器 SPI_W0_REG ~ SPI_W15_REG。
- 0x2 (Rd_BUF)：CPU 控制的读操作。主机接收 GP-SPI2 发送的数据。数据来自相应地址的寄存器 SPI_W0_REG ~ SPI_W15_REG。
- 0x3 (Wr_DMA)：DMA 控制的写操作。主机发送数据，GP-SPI2 接收数据。数据将存储至 GP-SPI2 的 GDMA RX buffer 中。
- 0x4 (Rd_DMA)：DMA 控制的读操作。主机接收 GP-SPI2 发送的数据。数据来自 GP-SPI2 的 GDMA TX buffer。
- 0x7 (CMD7)：用于生成 SPI_SLV_CMD7_INT 中断。在从机连续传输下，使用 DMA RX 链表时，也可用于生成 GDMA_IN_SUC_EOF_CH*n*_INT 中断。但不会结束 GP-SPI2 的从机连续传输。
- 0x8 (CMD8)：仅用于生成 SPI_SLV_CMD8_INT 中断，但不会结束 GP-SPI2 的从机连续传输。
- 0x9 (CMD9)：仅用于生成 SPI_SLV_CMD9_INT 中断，但不会结束 GP-SPI2 的从机连续传输。
- 0xA (CMDA)：仅用于生成 SPI_SLV_CMDA_INT 中断，但不会结束 GP-SPI2 的从机连续传输。

CMD7、CMD8、CMD9 和 CMDA 的具体用途可由用户自定义。这些命令可用作握手信号、某些特定功能的密码、或某些用户自定义操作的触发信号等。

CMD、ADDR 和 DATA 阶段均支持 1/2/4-bit 模式，具体由 CMD[7:4] 决定。DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。CMD[7:4] 的具体定义如下：

- 0x0：CMD、ADDR 和 DATA 阶段均为 1-bit 模式。
- 0x1：CMD 和 ADDR 均为 1-bit 模式。DATA 为 2-bit 模式。
- 0x2：CMD 和 ADDR 均为 1-bit 模式。DATA 为 4-bit 模式。
- 0x5：CMD 为 1-bit 模式。ADDR 和 DATA 均为 2-bit 模式。
- 0xA：CMD 为 1-bit 模式。ADDR 和 DATA 均为 4-bit 模式，或 QPI 模式。

此外，CMD[7:0] 的值为 0x05、0xA5、0x06 和 0xDD 时，将跳过 DUMMY 和 DATA 阶段。CMD[7:0] 的具体定义如下：

- 0x05 (End_SEG_TRANS)：主机发送 0x05 命令，结束 SPI 模式下从机连续传输。
- 0xA5 (End_SEG_TRANS)：主机发送 0xA5 命令，结束 QPI 模式下从机连续传输。
- 0x06 (En_QPI)：GP-SPI2 接收到 0x06 命令后，进入 QPI 模式。此时，寄存器 SPI_USER_REG 中 SPI_QPI_MODE 置位。
- 0xDD (Ex_QPI)：GP-SPI2 接收到 0xDD 命令后，退出 QPI 模式。此时，SPI_QPI_MODE 位清零。

GP-SPI2 支持的所有 CMD 值见表 26-14 和表 26-15。注意，DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。

表 26-14. GP-SPI2 从机 SPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
Wr_BUF	0x01	1-bit 模式	1-bit 模式	1-bit 模式
	0x11	1-bit 模式	1-bit 模式	2-bit 模式
	0x21	1-bit 模式	1-bit 模式	4-bit 模式

表 26-14. GP-SPI2 从机 SPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段
	0x51	1-bit 模式	2-bit 模式	2-bit 模式
	0xA1	1-bit 模式	4-bit 模式	4-bit 模式
Rd_BUF	0x02	1-bit 模式	1-bit 模式	1-bit 模式
	0x12	1-bit 模式	1-bit 模式	2-bit 模式
	0x22	1-bit 模式	1-bit 模式	4-bit 模式
	0x52	1-bit 模式	2-bit 模式	2-bit 模式
	0xA2	1-bit 模式	4-bit 模式	4-bit 模式
Wr_DMA	0x03	1-bit 模式	1-bit 模式	1-bit 模式
	0x13	1-bit 模式	1-bit 模式	2-bit 模式
	0x23	1-bit 模式	1-bit 模式	4-bit 模式
	0x53	1-bit 模式	2-bit 模式	2-bit 模式
	0xA3	1-bit 模式	4-bit 模式	4-bit 模式
Rd_DMA	0x04	1-bit 模式	1-bit 模式	1-bit 模式
	0x14	1-bit 模式	1-bit 模式	2-bit 模式
	0x24	1-bit 模式	1-bit 模式	4-bit 模式
	0x54	1-bit 模式	2-bit 模式	2-bit 模式
	0xA4	1-bit 模式	4-bit 模式	4-bit 模式
CMD7	0x07	1-bit 模式	1-bit 模式	-
	0x17	1-bit 模式	1-bit 模式	-
	0x27	1-bit 模式	1-bit 模式	-
	0x57	1-bit 模式	2-bit 模式	-
	0xA7	1-bit 模式	4-bit 模式	-
CMD8	0x08	1-bit 模式	1-bit 模式	-
	0x18	1-bit 模式	1-bit 模式	-
	0x28	1-bit 模式	1-bit 模式	-
	0x58	1-bit 模式	2-bit 模式	-
	0xA8	1-bit 模式	4-bit 模式	-
CMD9	0x09	1-bit 模式	1-bit 模式	-
	0x19	1-bit 模式	1-bit 模式	-
	0x29	1-bit 模式	1-bit 模式	-
	0x59	1-bit 模式	2-bit 模式	-
	0xA9	1-bit 模式	4-bit 模式	-
CMDA	0x0A	1-bit 模式	1-bit 模式	-
	0x1A	1-bit 模式	1-bit 模式	-
	0x2A	1-bit 模式	1-bit 模式	-
	0x5A	1-bit 模式	2-bit 模式	-
	0xAA	1-bit 模式	4-bit 模式	-
End_SEG_TRANS	0x05	1-bit 模式	-	-
En_QPI	0x06	1-bit 模式	-	-

表 26-15. QPI 模式支持的 CMD 值

传输类型	CMD[7:0]	CMD 阶段	ADDR 阶段	DATA 阶段

Wr_BUF	0xA1	4-bit 模式	4-bit 模式	4-bit 模式
Rd_BUF	0xA2	4-bit 模式	4-bit 模式	4-bit 模式
Wr_DMA	0xA3	4-bit 模式	4-bit 模式	4-bit 模式
Rd_DMA	0xA4	4-bit 模式	4-bit 模式	4-bit 模式
CMD7	0xA7	4-bit 模式	4-bit 模式	-
CMD8	0xA8	4-bit 模式	4-bit 模式	-
CMD9	0xA9	4-bit 模式	4-bit 模式	-
CMDA	0xAA	4-bit 模式	4-bit 模式	-
End_SEG_TRANS	0xA5	4-bit 模式	4-bit 模式	-
Ex_QPI	0xDD	4-bit 模式	4-bit 模式	-

GP-SPI2 收到主机发送的 0x06 CMD (En_QPI) 命令后，将进入 QPI 模式。GP-SPI2 在 QPI 模式下支持表 26-15中列出的传输类型，其后续所有阶段均为 4-bit 模式。如果收到 0xDD CMD (Ex_QPI)，则 GP-SPI2 从机将返回到 SPI 模式。

未在表 26-14 和表 26-15 中列出的传输类型将被忽略掉。如果传输的数据不以字节为单位，GP-SPI2 会发送或接收字节数据，但多余的比特数据（即总位长对 8 取模的结果）将会丢失。但如果 CS 低电平持续时长大于 2 个 APB_CLK 时钟周期，则将触发 SPI_TRANS_DONE_INT 中断。有关传输结束时触发的中断信息，请参考章节 26.8。

26.5.9.3 从机单次传输和从机连续传输

GP-SPI2 用作从机时，支持由 DMA 和 CPU 控制的全双工和半双工通信。DMA 控制的从机传输，可以是一次单次传输，也可以是从机连续传输（包含多次传输事务）。CPU 控制的传输只能是单次传输，因为每次传输均需由 CPU 触发。

一次从机连续传输包含多个传输事务，每个传输事务可以是表 26-14 和表 26-15 列出的任一传输类型。即在一次完整的连续传输过程中，可以包含 CPU 控制的数据传输，也可以包含 DMA 控制的数据传输。

在一次完整的连续传输过程中，推荐操作如下：

- CPU 控制的数据传输可用于握手通信以及少量数据传输；
- DMA 控制的数据传输可用于大量数据传输。

26.5.9.4 配置从机单次传输

用作从机时，GP-SPI2 支持 CPU 控制的和 DMA 控制的全/半双工单次传输。具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB、APB 时钟（即 AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 使能从机传输；
4. 配置 SPI_DOUTDIN：
    - 1：使能全双工通信；
    - 0：使能半双工通信。
5. 准备数据：

- 如果选择的传输类型为 CPU 控制的传输，且 GP-SPI2 发送数据，则在寄存器 SPI_W0_REG ~ SPI_W15_REG 中准备数据。
- 如果选择的传输类型为 DMA 控制的传输，则需要：
  - 配置 SPI_DMA_RX_ENA/SPI_DMA_TX_ENA 和 SPI_RX_EOF_EN；
  - 配置 GDMA TX/RX 链表；
  - 启动 GDMA TX/RX 引擎，更多描述见章节 26.5.6 和章节 26.5.7。

6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
7. 清零寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN 使能从机单次传输；
8. 置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_TRANS_DONE_INT_ENA，使能中断，并等待 SPI_TRANS_DONE_INT 中断。在 DMA 控制的传输下，使用 DMA RX buffer 时，推荐等待 GDMA_IN_SUC_EOF_CH*n*_INT 中断，即数据已存储至相应内存。其他中断见章节 26.8。

26.5.9.5 配置半双工通信下从机连续传输

此时必须使用 GDMA。具体的寄存器配置如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB、APB 时钟（即 AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 使能从机传输；
4. 清除 SPI_DOUTDIN 使能半双工通信方式；
5. 根据需求，确定是否需要在寄存器 SPI_W0_REG ~ SPI_W15_REG 中准备数据；
6. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；
7. 置位 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA。清零 SPI_RX_EOF_EN。配置 GDMA TX/RX 链表，并启动 GDMA TX/RX 引擎，更多描述见章节 26.5.6 和章节 26.5.7；
8. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN，使能从机连续传输；
9. 置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_DMA_SEG_TRANS_DONE_INT_ENA，使能中断，并等待 SPI_DMA_SEG_TRANS_DONE_INT 中断。中断发生，即表明从机连续传输已结束，且数据已放入相应的内存中。其他中断见章节 26.8。

GP-SPI2 收到 End_SEG_TRANS 命令（SPI 模式下为 0x05，QPI 模式下为 0xA5），从机连续传输结束，并触发 SPI_DMA_SEG_TRANS_DONE_INT 中断。

26.5.9.6 配置全双工通信下从机连续传输

在这一传输中，必须使用 GDMA。数据从 GDMA buffer 中输入输出。传输结束，触发 GDMA_IN_SUC_EOF_CH*n*_INT 中断。具体的配置程序如下：

1. 经 IO MUX 或 GPIO 交换矩阵配置 GP-SPI2 与外部 SPI 设备之间的 IO 通道；
2. 配置 AHB、APB 时钟（即 AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 和 SPI_DOUTDIN，使能从机全双工通信；
4. 置位 SPI_DMA_AFIFO_RST、SPI_BUF_AFIFO_RST 和 SPI_RX_AFIFO_RST 复位 buffer；

5. 置位 SPI_DMA_RX_ENA 和 SPI_DMA_TX_ENA。配置 GDMA TX/RX 链表，并启动 GDMA TX/RX 引擎，更多描述见章节 26.5.6 和章节 26.5.7；
6. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_RX_EOF_EN。在寄存器 SPI_MS_DLEN_REG 的 SPI_MS_DATA_BITLEN[17:0] 中配置 DMA 接收数据长度（单位：字节）；
7. 置位寄存器 SPI_DMA_CONF_REG 中 SPI_DMA_SLV_SEG_TRANS_EN，使能从机连续传输；
8. 置位 GDMA_IN_SUC_EOF_CH*n*_INT_ENA 使能中断，然后等待 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

26.6 CS 建立时间和保持时间控制

SPI CS 建立时间和保持时间对于满足各种 SPI 设备（如 flash 或 PSRAM）的时序要求非常重要。

CS 建立时间为 CS 下降沿至 SPI_CLK 第一个锁存边沿的时间。模式 0 和模式 3 的第一锁存边沿为上升沿，模式 2 和模式 4 的第一锁存边沿为下降沿。

CS 保持时间为 SPI_CLK 最后一个锁存边沿到 CS 上升沿之间的时间。

用作从机时，CS 建立时间和保持时间应大于 0.5 x T_SPI_CLK，否则 SPI 传输可能出错。这里的 T_SPI_CLK 指 SPI_CLK 时钟周期。

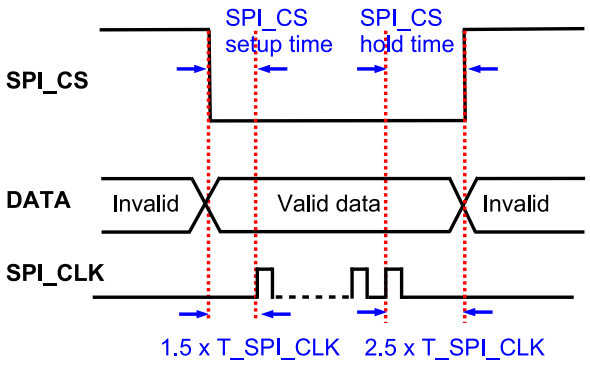
用作主机时，CS 建立时间由寄存器 SPI_USER_REG 中的 SPI_CS_SETUP 位和寄存器 SPI_USER1_REG 中的 SPI_CS_SETUP_TIME 位控制：

- 清零 SPI_CS_SETUP，则 SPI CS 建立时间为 0.5 x T_SPI_CLK；
- 置位 SPI_CS_SETUP，则 SPI CS 建立时间为 (SPI_CS_SETUP_TIME + 1.5) x T_SPI_CLK。

CS 保持时间由寄存器 SPI_USER_REG 中的 SPI_CS_HOLD 位和寄存器 SPI_USER1_REG 中的 SPI_CS_HOLD_TIME 位控制：

- 清零 SPI_CS_HOLD，则 SPI CS 保持时间为 0.5 x T_SPI_CLK；
- 置位 SPI_CS_HOLD，则 SPI CS 保持时间为 (SPI_CS_HOLD_TIME + 1.5) x T_SPI_CLK。

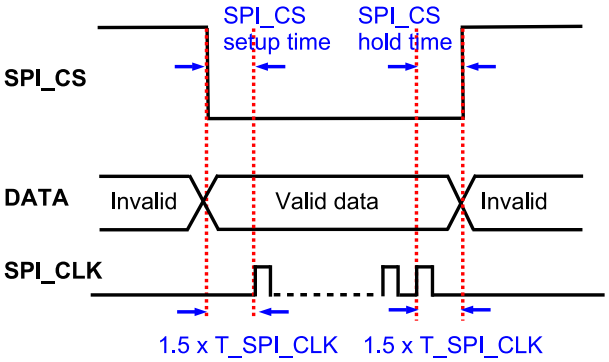
图 26-11 和图 26-12 所示为访问外部 RAM 和 flash 时推荐的 CS 时序配置和寄存器配置。



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 1.

图 26-11. GP-SPI2 访问外部 RAM 时推荐的 CS 时序配置



Register Configurations:

SPI_CS_SETUP = 1; SPI_CS_SETUP_TIME = 0;
SPI_CS_HOLD = 1; SPI_CS_HOLD_TIME = 0.

图 26-12. GP-SPI2 访问 Flash 时推荐的 CS 时序配置

26.7 GP-SPI2 时钟控制

GP-SPI2 中有以下三个时钟：

- clk_spi_mst：GP-SPI2 模块时钟，在 GP-SPI2 用作主机时用于生成数据传输以及从机所需的 SPI_CLK 信号；
- SPI_CLK：主机输出时钟；
- AHB_CLK：用于寄存器配置的时钟（最高 32M）。

其中，clk_spi_mst 时钟的开关寄存器为 PCR_SPI2_MST_CLK_ACTIVE_I，时钟源选择寄存器为 PCR_SPI2_MST_CLK_SEL_I[1:0]：

- 0：时钟源为 XTAL_CLK；
- 1：时钟源为 PLL_F48M_CLK；
- 2：时钟源为 RC_FAST_CLK。

用作主机时，GP-SPI2 最高输出时钟频率为 $f_{clk\_spi\_mst}$。如果需要较低的时钟频率，可以采用如下分频方式：

$$f_{SPI\_CLK} = \frac{f_{clk\_spi\_mst}}{(SPI\_CLKCNT\_N + 1)(SPI\_CLKDIV\_PRE + 1)}$$

用户可配置寄存器 SPI_CLOCK_REG 中 SPI_CLKCNT_N 和 SPI_CLKDIV_PRE 设置分频系数。寄存器 SPI_CLOCK_REG 中 SPI_CLK_EQU_SYSCLK 位置 1 时，GP-SPI 的输出时钟频率为 $f_{clk\_spi\_mst}$。如果采用其他整数分频，则 SPI_CLK_EQU_SYSCLK 应置 0。用作从机时，GP-SPI2 支持的输入时钟频率为 $f_{clk\_spi\_slv}$，且有如下频率关系：

- 如果 f_{AHB_CLK} >= 40 MHz，则输入时钟频率为：$f_{clk\_spi\_slv}$ <= 40 MHz；
- 如果 f_{AHB_CLK} < 40 MHz，则输入时钟频率为：$f_{clk\_spi\_slv}$ <= f_{AHB_CLK}。

26.7.1 时钟相位和极性

SPI 协议支持四种时钟模式，即模式 0 ~ 3，见图 26-13 和图 26-14。注，图片来源于 SPI 协议。

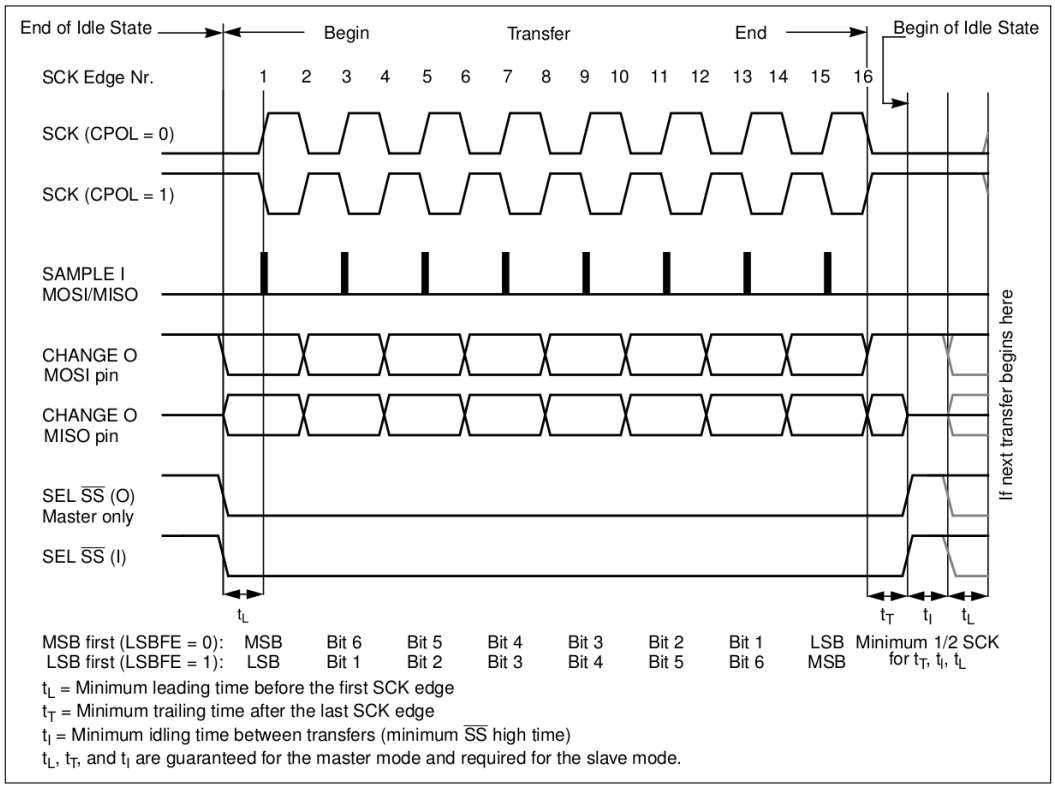


图 26-13. SPI 时钟模式 0 和时钟模式 2

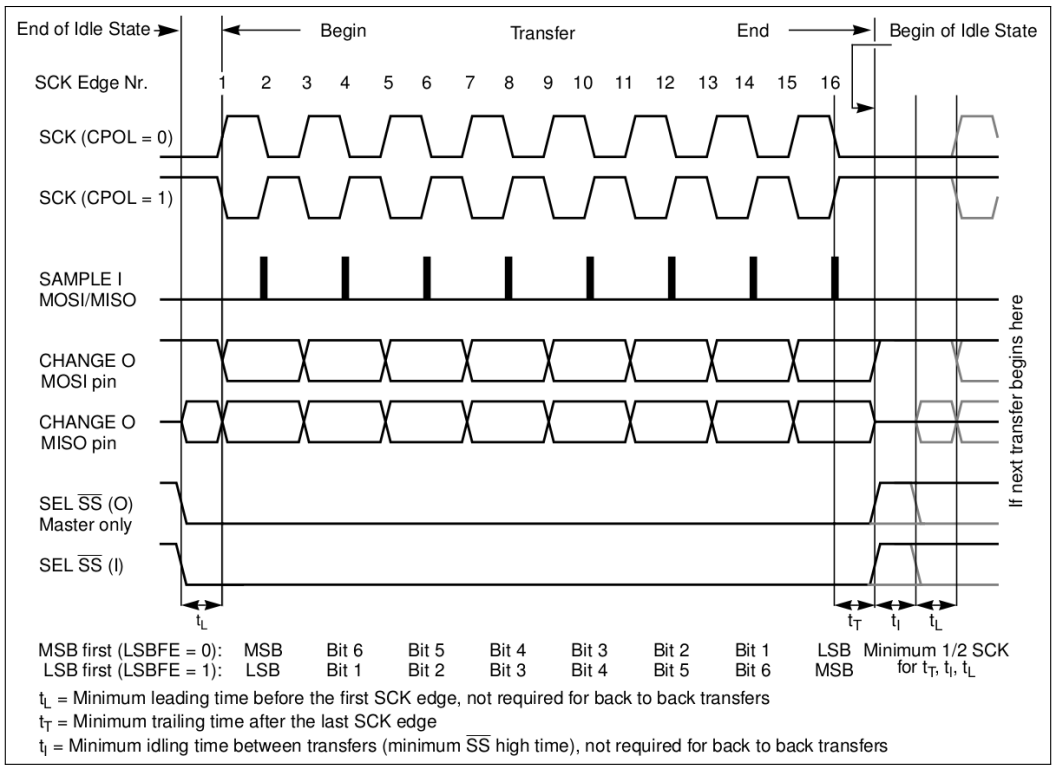


图 26-14. SPI 时钟模式 1 和时钟模式 3

1. 模式 0：CPOL = 0，CPHA = 0；SPI 处于空闲模式时，SCK 为 0；数据在 SCK 下降沿变化，在上升沿采样。第一个数据在 SCK 的第一个下降沿之前被移出。
2. 模式 1：CPOL = 0，CPHA = 1；SPI 处于空闲模式时，SCK 为 0；数据在 SCK 上升沿变化，在下降沿采样。
3. 模式 2：CPOL = 1，CPHA = 0；SPI 处于空闲模式时，SCK 为 1；数据在 SCK 上升沿变化，在下降沿采样。第一个数据在 SCK 的第一个上升沿之前被移出。
4. 模式 3：CPOL = 1，CPHA = 1；SPI 处于空闲模式时，SCK 为 1；数据在 SCK 下降沿变化，在上升沿采样。

26.7.2 主机时钟控制

GP-SPI2 主机支持四种 SPI 时钟模式：模式 0 ~ 3。GP-SPI2 极性和相位由寄存器 SPI_MISC_REG 中 SPI_CK_IDLE_EDGE 位和寄存器 SPI_USER_REG 中 SPI_CK_OUT_EDGE 位控制。SPI 时钟模式 0 ~ 3 的寄存器配置见表 26-16，可根据应用的路径延迟进行更改。

表 26-16. 主机时钟相位和极性配置

寄存器控制位	模式 0	模式 1	模式 2	模式 3
SPI_CK_IDLE_EDGE	0	0	1	1
SPI_CK_OUT_EDGE	0	1	1	0

此外，SPI_CLK_MODE 可用于选择 CS 拉高时 SPI_CLK 的上升沿个数：0、1、2 或 SPI_CLK 一直有效。

说明：
SPI_CLK_MODE 配置成 1 或 2 时，必须置位 SPI_CS_HOLD 且 SPI_CS_HOLD_TIME 的值需大于 1。

26.7.3 从机时钟控制

GP-SPI2 从机也支持四种 SPI 时钟模式：即模式 0 ~ 3。寄存器 SPI_USER_REG 中 SPI_TSCK_I_EDGE 和 SPI_RSCK_I_EDGE 位可用于配置时钟极性和相位。数据的输出沿则由寄存器 SPI_SLAVE_REG 中的 SPI_CLK_MODE_13 位控制。寄存器具体配置见表 26-17。

表 26-17. 从机时钟相位和极性配置

寄存器控制位	模式 0	模式 1	模式 2	模式 3
SPI_TSCK_I_EDGE	0	1	1	0
SPI_RSCK_I_EDGE	0	1	1	0
SPI_CLK_MODE_13	0	1	0	1

26.8 中断

中断描述

GP-SPI2 提供一个 SPI 接口中断：SPI_INT。一次 SPI 传输结束时，GP-SPI2 即生成一次中断。

- SPI_DMA_INFIFO_FULL_ERR_INT：GDMA RX FIFO 小于实际传输的数据长度时即触发此中断。
- SPI_DMA_OUTFIFO_EMPTY_ERR_INT：GDMA TX FIFO 小于实际传输的数据长度时即触发此中断。

- SPI_SLV_EX_QPI_INT：GP-SPI2 用作从机时，正确接收 Ex_QPI 命令，且 SPI 传输结束即触发此中断。
- SPI_SLV_EN_QPI_INT：GP-SPI2 用作从机时，正确接收 En_QPI 命令，且 SPI 传输结束即触发此中断。
- SPI_SLV_CMD7_INT：GP-SPI2 用作从机时，正确接收 CMD7 命令，且 SPI 传输结束即触发此中断。
- SPI_SLV_CMD8_INT：GP-SPI2 用作从机时，正确接收 CMD8 命令，且 SPI 传输结束即触发此中断。
- SPI_SLV_CMD9_INT：GP-SPI2 用作从机时，正确接收 CMD9 命令，且 SPI 传输结束即触发此中断。
- SPI_SLV_CMDA_INT：GP-SPI2 用作从机时，正确接收 CMDA 命令，且 SPI 传输结束即触发此中断。
- SPI_SLV_RD_DMA_DONE_INT：GP-SPI2 用作从机时，Rd_DMA 传输结束即触发此中断。
- SPI_SLV_WR_DMA_DONE_INT：GP-SPI2 用作从机时，Wr_DMA 传输结束即触发此中断。
- SPI_SLV_RD_BUF_DONE_INT：GP-SPI2 用作从机时，Rd_BUF 传输结束即触发此中断。
- SPI_SLV_WR_BUF_DONE_INT：GP-SPI2 用作从机时，Wr_BUF 传输结束即触发此中断。
- SPI_TRANS_DONE_INT：GP-SPI2 用作主机或从机时，SPI 总线传输结束均会触发此中断。
- SPI_DMA_SEG_TRANS_DONE_INT：GP-SPI2 从机连续传输下，End_SEG_TRANS 传输结束即触发此中断。GP-SPI2 用作主机时，分段配置传输结束也将触发此中断。
- SPI_SEG_MAGIC_ERR_INT：在主机分段配置传输下，CONF buffer 中的 Magic 值有误即触发此中断。
- SPI_MST_RX_AFIFO_WFULL_ERR_INT：GP-SPI2 用作主机时，如果发生 RX AFIFO write-full 错误，即触发此中断。
- SPI_MST_TX_AFIFO_REMPTY_ERR_INT：GP-SPI2 用作主机时，如果发生 TX AFIFO read-empty 错误即触发此中断。
- SPI_SLV_CMD_ERR_INT：GP-SPI2 用作从机时，如果接收到的命令值 GP-SPI2 不支持，即触发此中断。
- SPI_APP2_INT：用于软件，且由软件触发。仅用于用户自定义的功能。
- SPI_APP1_INT：用于软件，且由软件触发。仅用于用户自定义的功能。

GP-SPI2 用作主机和从机时分别用到的中断

表 26-18 和表 26-19 分别列出了 GP-SPI2 用作主机和用作从机时用到的中断。置位寄存器 SPI_DMA_INT_ENA_REG 中 SPI_*_INT_ENA 位，使能相应中断，并等待 SPI_INT 中断。传输结束时，将触发相关中断。注意，在下次传输之前，需软件清除中断。

表 26-18. GP-SPI2 用作主机时用到的中断

传输类型	通信模式	控制方式	中断
单次传输	全双工	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	半双工主机输出从机输入	DMA	SPI_TRANS_DONE_INT
		CPU	SPI_TRANS_DONE_INT
	半双工主机输入从机输出	DMA	GDMA_IN_SUC_EOF_CH*n*_INT
		CPU	SPI_TRANS_DONE_INT
分段配置传输	全双工	DMA	SPI_DMA_SEG_TRANS_DONE_INT [3]
		CPU	不支持
	半双工主机输出从机输入	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	不支持

接上页

表 26-18 – 见下页

传输类型	通信模式	控制方式	中断
	半双工主机输入从机输出	DMA	SPI_DMA_SEG_TRANS_DONE_INT
		CPU	不支持

[1] 如果触发了 GDMA_IN_SUC_EOF_CH*n*_INT 中断，则表示 GP-SPI2 的所有 RX 数据已保存至 RX buffer，且所有 TX 数据已发送至从机。

[2] CS 拉高，则将触发 SPI_TRANS_DONE_INT 中断，表明主机与从机已完成 SPI_W0_REG ~ SPI_W15_REG 的数据交换。

[3] 如果触发了 SPI_DMA_SEG_TRANS_DONE_INT 中断，则表明整个分段配置传输，包括若干个传输事务，已完成。即 RX 数据已全部存入 RX buffer 且所有 TX 数据已发送完毕。

表 26-19. GP-SPI2 用作从机时用到的中断

传输类型	通信模式	控制方式	中断
单次传输	全双工	DMA	GDMA_IN_SUC_EOF_CH*n*_INT [1]
		CPU	SPI_TRANS_DONE_INT [2]
	半双工主机输出从机输入	DMA (Wr_DMA)	GDMA_IN_SUC_EOF_CH*n*_INT[3]
		CPU (Wr_BUF)	SPI_TRANS_DONE_INT[4]
	半双工主机输入从机输出	DMA (Rd_DMA)	SPI_TRANS_DONE_INT[5]
		CPU (Rd_BUF)	SPI_TRANS_DONE_INT[6]
从机连续传输	全双工	DMA	GDMA_IN_SUC_EOF_CH*n*_INT[7]
		CPU	不支持[8]
	半双工主机输出从机输入	DMA (Wr_DMA)	SPI_DMA_SEG_TRANS_DONE_INT[9]
		CPU (Wr_BUF)	不支持[10]
	半双工主机输入从机输出	DMA (Rd_DMA)	SPI_DMA_SEG_TRANS_DONE_INT[11]
		CPU (Rd_BUF)	不支持[12]

[1] 如果触发了 GDMA_IN_SUC_EOF_CH*n*_INT 中断，则表示所有 RX 数据已保存至 RX buffer，且所有 TX 数据已发送至从机。

[2] CS 拉高，则将触发 SPI_TRANS_DONE_INT 中断，表明主机与从机已完成 SPI_W0_REG ~ SPI_W15_REG 的数据交换。

[3] 触发 SPI_SLV_WR_DMA_DONE_INT 中断仅表示 SPI 总线上的数据传输已完成，但并不能保证所有入栈数据已存至 RX buffer。因此，推荐使用 GDMA_IN_SUC_EOF_CH*n*_INT 中断。

[4] 或等待 SPI_SLV_WR_BUF_DONE_INT 中断。

[5] 或等待 SPI_SLV_RD_DMA_DONE_INT 中断。

[6] 或等待 SPI_SLV_RD_BUF_DONE_INT 中断。

[7] 传输开始前，从机应在 SPI_MS_DATA_BITLEN 中设置读数据的总长度。并在中断程序结束前，置位 SPI_RX_EOF_EN。

[8] 主机和从机需定义连续传输结束的方式，比如配置 GPIO 用作中断等。

[9] 主机发送 COM5 结束连续传输，或从机在 SPI_MS_DATA_BITLEN 中配置总的读数据长度，然后等待 GDMA_IN_SUC
_EOF_CH*n*_INT 中断。

[10] 半双工 Wr_BUF 单次传输也可用于 DMA 控制的从机连续传输中。

[11] 主机发送 End_SEG_TRAN 结束从机连续传输。

[12] 半双工 Rd_BUF 单次传输也可用于 DMA 控制的从机连续传输中。

26.9 寄存器列表

本小节的所有地址均为相对于 GP-SPI2 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

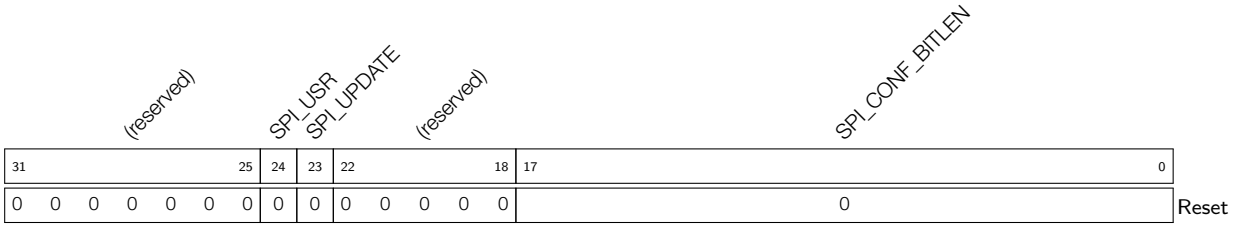
名称	描述	地址	访问
自定义控制寄存器			
SPI_CMD_REG	命令控制寄存器	0x0000	varies
SPI_ADDR_REG	地址值寄存器	0x0004	R/W
SPI_USER_REG	SPI 用户控制寄存器	0x0010	varies
SPI_USER1_REG	SPI 用户控制寄存器 1	0x0014	R/W
SPI_USER2_REG	SPI 用户控制寄存器 2	0x0018	R/W
控制和配置寄存器			
SPI_CTRL_REG	SPI 控制寄存器	0x0008	R/W
SPI_MS_DLEN_REG	SPI 数据位长控制寄存器	0x001C	R/W
SPI_MISC_REG	SPI MISC 寄存器	0x0020	R/W
SPI_DMA_CONF_REG	SPI DMA 控制寄存器	0x0030	varies
SPI_SLAVE_REG	SPI 从机控制寄存器	0x00E0	varies
SPI_SLAVE1_REG	SPI 从机控制寄存器 1	0x00E4	R/W/SS
时钟控制寄存器			
SPI_CLOCK_REG	SPI 时钟控制寄存器	0x000C	R/W
SPI_CLK_GATE_REG	SPI 模块时钟和寄存器时钟控制	0x00E8	R/W
时序寄存器			
SPI_DIN_MODE_REG	SPI 输入延迟模式配置	0x0024	R/W
SPI_DIN_NUM_REG	SPI 输入延迟周期配置	0x0028	R/W
SPI_DOUT_MODE_REG	SPI 输出延迟模式配置	0x002C	R/W
中断寄存器			
SPI_DMA_INT_ENA_REG	SPI DMA 中断使能寄存器	0x0034	R/W
SPI_DMA_INT_CLR_REG	SPI DMA 中断清除寄存器	0x0038	WT
SPI_DMA_INT_RAW_REG	SPI DMA 原始中断寄存器	0x003C	varies
SPI_DMA_INT_ST_REG	SPI DMA 中断状态寄存器	0x0040	RO
SPI_DMA_INT_SET_REG	SPI DMA 中断软件置位寄存器	0x0044	RO
CPU 数据 Buffer			
SPI_W0_REG	SPI CPU 控制的 buffer 0	0x0098	R/W/SS
SPI_W1_REG	SPI CPU 控制的 buffer 1	0x009C	R/W/SS
SPI_W2_REG	SPI CPU 控制的 buffer 2	0x00A0	R/W/SS
SPI_W3_REG	SPI CPU 控制的 buffer 3	0x00A4	R/W/SS
SPI_W4_REG	SPI CPU 控制的 buffer 4	0x00A8	R/W/SS
SPI_W5_REG	SPI CPU 控制的 buffer 5	0x00AC	R/W/SS
SPI_W6_REG	SPI CPU 控制的 buffer 6	0x00B0	R/W/SS
SPI_W7_REG	SPI CPU 控制的 buffer 7	0x00B4	R/W/SS
SPI_W8_REG	SPI CPU 控制的 buffer 8	0x00B8	R/W/SS
SPI_W9_REG	SPI CPU 控制的 buffer 9	0x00BC	R/W/SS
SPI_W10_REG	SPI CPU 控制的 buffer 10	0x00C0	R/W/SS

名称	描述	地址	访问
SPI_W11_REG	SPI CPU 控制的 buffer 11	0x00C4	R/W/SS
SPI_W12_REG	SPI CPU 控制的 buffer 12	0x00C8	R/W/SS
SPI_W13_REG	SPI CPU 控制的 buffer 13	0x00CC	R/W/SS
SPI_W14_REG	SPI CPU 控制的 buffer 14	0x00D0	R/W/SS
SPI_W15_REG	SPI CPU 控制的 buffer 15	0x00D4	R/W/SS
版本寄存器			
SPI_DATE_REG	版本控制寄存器	0x00F0	R/W

26.10 寄存器

本小节的所有地址均为相对于 GP-SPI2 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 26.1. SPI_CMD_REG (0x0000)



SPI_CONF_BITLEN 配置 SPI CONF 阶段的 SPI_CLK 周期。

单位：SPI_CLK 时钟周期。

可在 CONF 阶段配置。(R/W)

SPI_UPDATE 配置是否将 SPI 寄存器从 APB 时钟域同步到 SPI 模块时钟域。

0：不同步

1：同步

该位仅用于 SPI 主机。(WT)

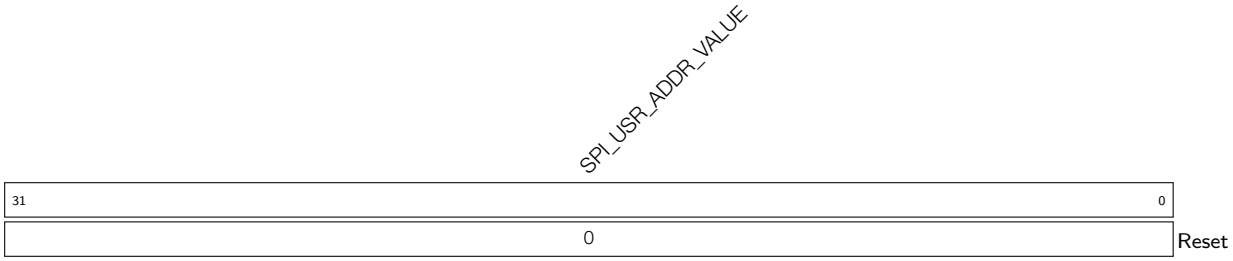
SPI_USR 配置是否使能用户自定义命令。

0：不使能

1：使能

置位此位将触发一次 SPI 操作。操作结束后此位被自动清零。CONF_buf 不可更改该配置。(R/W/SC)

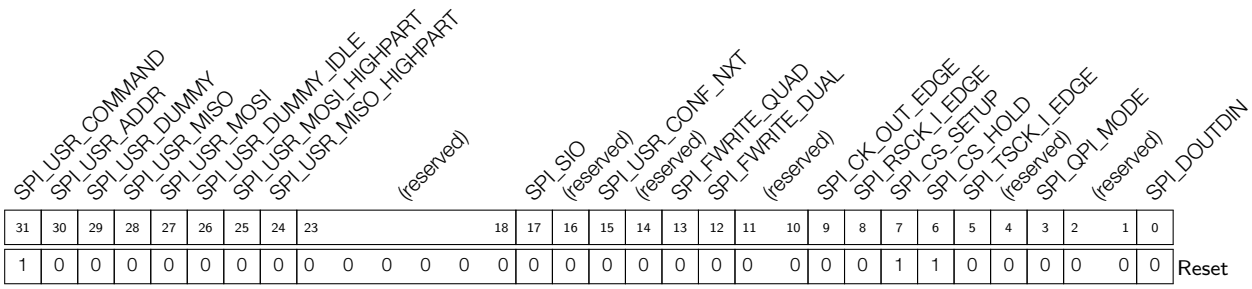
Register 26.2. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE 配置从机地址。

可在 CONF 阶段配置。(R/W)

Register 26.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN 配置是否使能全双工通信。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_QPI_MODE 配置是否使能 QPI 模式。

0：不使能

1：使能

SPI 主机和从机均支持该配置。可在 CONF 阶段配置。(R/W/SS/SC)

SPI_TSCK_I_EDGE 配置 SPI 从机是否更改 TSCK 极性。

0: TSCK = SPI_CK_I

1: TSCK = !SPI_CK_I

(R/W)

SPI_CS_HOLD 配置在 SPI 处于完成 (DONE) 阶段时是否保持 SPI CS 拉低。

0：不保持拉低

1：保持拉低

可在 CONF 阶段配置。(R/W)

SPI_CS_SETUP 配置在 SPI 处于准备 (PREP) 阶段时是否使能 SPI CS。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_RSCK_I_EDGE 配置 SPI 从机是否更改 RSCK 极性。

0: RSCK = !SPI_CK_I

1: RSCK = SPI_CK_I

(R/W)

SPI_CK_OUT_EDGE 配置 SPI 时钟模式，与 SPI_CK_IDLE_EDGE 搭配使用。

可在 CONF 阶段配置。更多信息见章节 26.7.2。(R/W)

SPI_FWRITE_DUAL 配置在写操作 (DOUT) 阶段时读数据的方式是否为 2-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

见下页...

Register 26.3. SPI_USER_REG (0x0010)

接上页...

SPI_FWRITE_QUAD 配置在写操作 (DOUT) 阶段时读数据的方式是否为 4-bit 方式。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_USR_CONF_NXT 配置是否使能下一次传输事务的 CONF 阶段。

0：当前传输事务结束后，本次分段配置传输结束。或者，当前的传输模式不是分段配置传输。

1：本次分段配置传输继续进行，开始下一次传输事务。

可在 CONF 阶段配置。(R/W)

SPI_SIO 配置是否使能 3 线半双工通信，其中 MOSI 和 MISO 信号共享一个管脚。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_USR_MISO_HIGHPART 配置在读数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_USR_MOSI_HIGHPART 配置在写数据阶段是否使能“高位模式”，即仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_USR_DUMMY_IDLE 配置是否在 DUMMY 阶段禁用 SPI 时钟。

0：不禁用

1：禁用

可在 CONF 阶段配置。(R/W)

SPI_USR_MOSI 配置是否使能一次操作的写数据 (DOUT) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_USR_MISO 配置是否使能一次操作的读数据 (DIN) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

SPI_USR_DUMMY 配置是否使能一次操作的 DUMMY 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

见下页...

Register 26.3. SPI_USER_REG (0x0010)

接上页...

SPI_USR_ADDR 配置是否使能一次操作的地址 (ADDR) 阶段。

0：不使能

1：使能

(R/W)

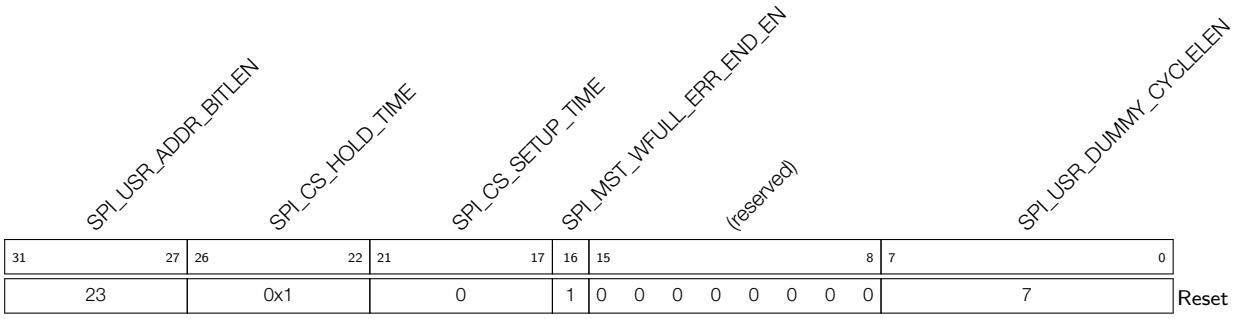
SPI_USR_COMMAND 配置是否使能一次操作的命令 (CMD) 阶段。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

Register 26.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN 配置 DUMMY 阶段的时长。

单位：SPI_CLK 时钟周期。

此值为（预期周期数 - 1）。可在 CONF 阶段配置。(R/W)

SPI_MST_WFULL_ERR_END_EN 配置在主机全双工或半双工模式下，如果发生 SPI RX AFIFO 满错误，是否终止 SPI 传输。

0：不终止

1：终止

(R/W)

SPI_CS_SETUP_TIME 配置准备 (PREP) 阶段的时长。

单位：SPI_CLK 时钟周期。

此值等于预期周期数 - 1。此字段与 SPI_CS_SETUP 搭配使用。可在 CONF 阶段配置。(R/W)

SPI_CS_HOLD_TIME 配置 CS 管脚的延迟周期。

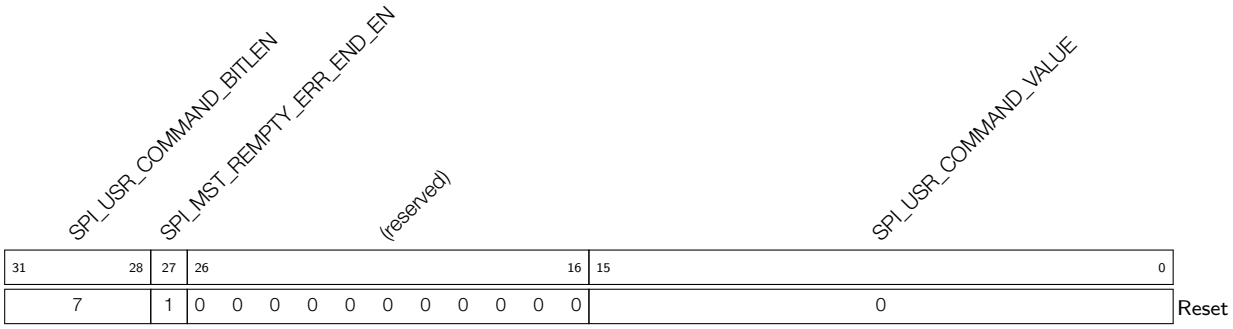
单位：SPI_CLK 时钟周期。

此字段与 SPI_CS_HOLD 搭配使用。可在 CONF 阶段配置。(R/W)

SPI_USR_ADDR_BITLEN 配置地址阶段的位长。

此值为（预期位数 - 1）。可在 CONF 阶段配置。(R/W)

Register 26.5. SPI_USER2_REG (0x0018)



SPI_USR_COMMAND_VALUE 配置命令值。

可在 CONF 阶段配置。(R/W)

SPI_MST_REMPTY_ERR_END_EN 配置在主机全双工或半双工模式下，如果发生 SPI TX AFIFO 空错误，是否终止 SPI 传输。

0：不终止

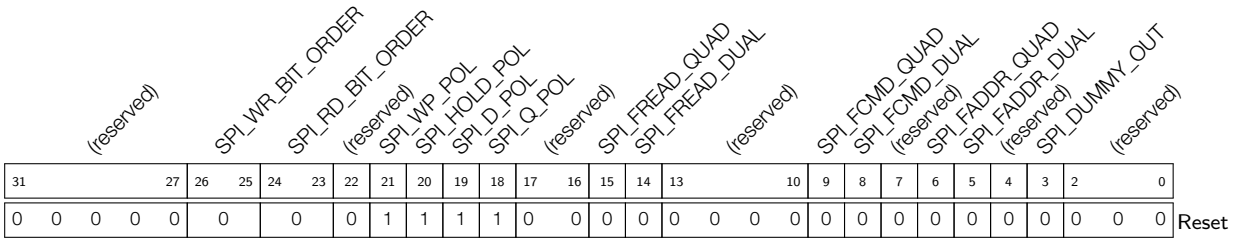
1：终止

(R/W)

SPI_USR_COMMAND_BITLEN 配置命令阶段的位长。

此值为（预期位数 - 1）。可在 CONF 阶段配置。(R/W)

Register 26.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT 配置在 DUMMY 阶段是否输出 FSPI 总线信号。

0：不输出

1：输出

可在 CONF 阶段配置。(R/W)

SPI_FADDR_DUAL 配置在地址 (ADDR) 阶段时是否采用 2-bit 模式。

0：不采用

1：采用

可在 CONF 阶段配置。(R/W)

SPI_FADDR_QUAD 配置在地址 (ADDR) 阶段时是否采用 4-bit 模式。

0：不采用

1：采用

可在 CONF 阶段配置。(R/W)

SPI_FCMD_DUAL 配置在命令 (CMD) 阶段时是否采用 2-bit 模式。

0：不采用

1：采用

可在 CONF 阶段配置。(R/W)

SPI_FCMD_QUAD 配置在命令 (CMD) 阶段是否采用 4-bit 模式。

0：不采用

1：采用

可在 CONF 阶段配置。(R/W)

SPI_FREAD_DUAL 配置在读数据 (DIN) 阶段是否采用 2-bit 模式。

0：不采用

1：采用

可在 CONF 阶段配置。(R/W)

SPI_FREAD_QUAD 配置在读数据 (DIN) 阶段是否采用 4-bit 模式。

0：不采用

1：采用

可在 CONF 阶段配置。(R/W)

SPI_Q_POL 配置 MISO 的极性。

0：低

1：高

可在 CONF 阶段配置。(R/W)

见下页...

Register 26.6. SPI_CTRL_REG (0x0008)

接上页...

SPI_D_POL 配置 MOSI 的极性。

0：低

1：高

可在 CONF 阶段配置。(R/W)

SPI_HOLD_POL 配置在 SPI 空闲状态下 SPI_HOLD 的输出值。

0：输出低电平

1：输出高电平

可在 CONF 阶段配置。(R/W)

SPI_WP_POL 配置在 SPI 空闲状态下 WP 信号的输出值。

0：输出低电平

1：输出高电平

可在 CONF 阶段配置。(R/W)

SPI_RD_BIT_ORDER 配置读数据 (MISO) 阶段的比特顺序。

0：先读高有效位

1：先读低有效位

可在 CONF 阶段配置。(R/W)

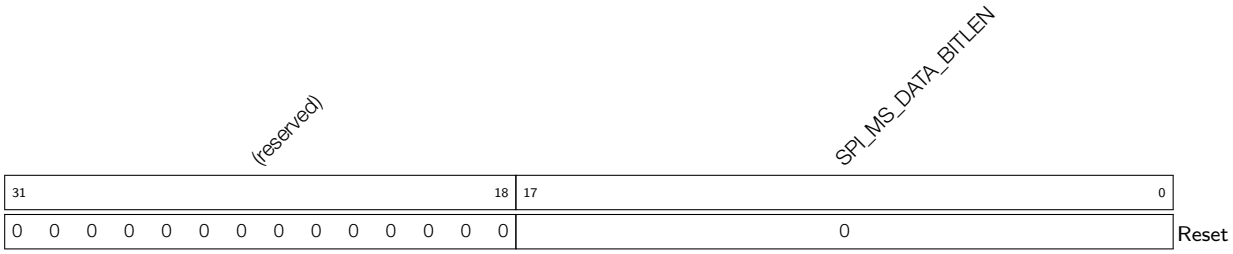
SPI_WR_BIT_ORDER 配置命令 (CMD)、地址 (ADDR) 和写数据 (MOSI) 阶段的比特顺序。

0：先写高有效位

1：先写低有效位

可在 CONF 阶段配置。(R/W)

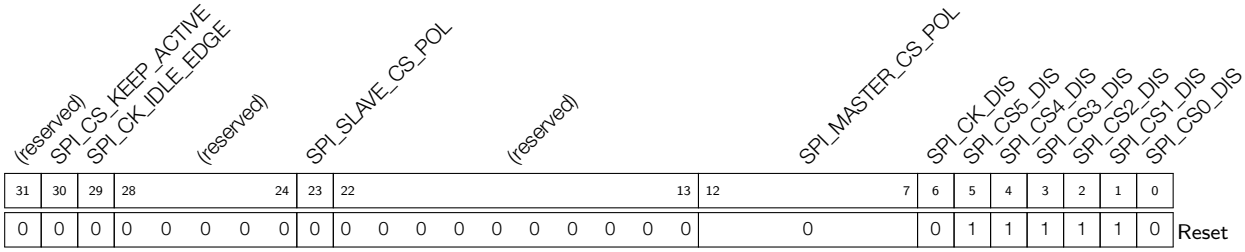
Register 26.7. SPI_MS_DLEN_REG (0x001C)



SPI_MS_DATA_BITLEN 配置主机 DMA 控制或 CPU 控制的 SPI 传输的数据位长。或配置从机 DMA 控制的传输中接收数据的位长。

该值等于需要的位长 - 1。可在 CONF 阶段配置。(R/W)

Register 26.8. SPI_MISC_REG (0x0020)



SPI_CS*n*_DIS (*n* = 0-5) 配置是否禁用 SPI_CS*n* 管脚。

0：SPI_CS*n* 信号来自 CS*n* 管脚或输出至 CS*n* 管脚

1：禁用 CS*n*

可在 CONF 阶段配置。(R/W)

SPI_CK_DIS 配置是否禁用 SPI_CK 输出信号。

0：使能 SPI_CLK 输出信号

1：停止 SPI_CLK 输出信号

可在 CONF 阶段配置。(R/W)

SPI_MASTER_CS_POL 配置主机 SPI_CS*n* (*n* = 0-5)。

0：SPI_CS*n* 低电平有效

1：SPI_CS*n* 高电平有效

(R/W)

SPI_SLAVE_CS_POL 配置 SPI 从机输入信号 CS 的极性。

0：保持不变

1：反相

可在 CONF 阶段配置。(R/W)

SPI_CK_IDLE_EDGE 配置 SPI_CLK 线在 GP-SPI2 空闲状态时是否保持高电平。

0：保持低电平

1：保持高电平

可在 CONF 阶段配置。(R/W)

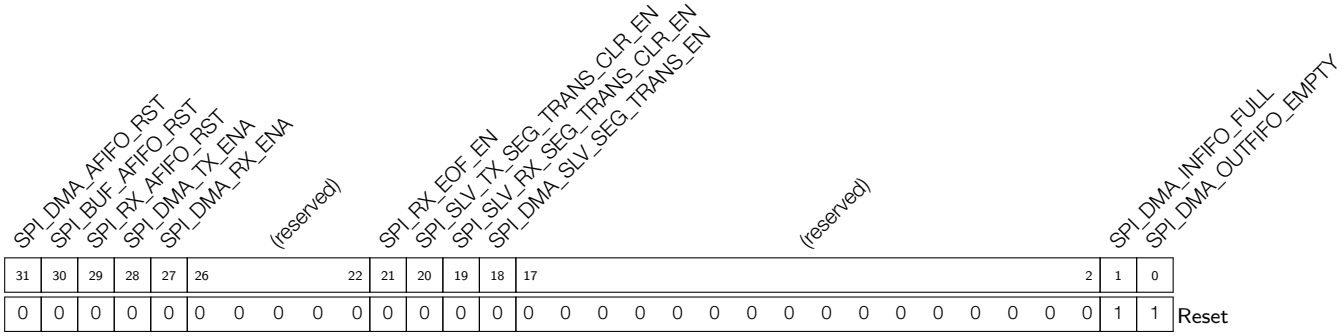
SPI_CS_KEEP_ACTIVE 配置 SPI_CS 是保持低电平。

0：不保持低电平

1：保持低电平

可在 CONF 阶段配置。(R/W)

Register 26.9. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY 表示 DMA TX FIFO 是否就绪。

0：DMA TX FIFO 已就绪，可以发送数据

1：DMA TX FIFO 尚未就绪，不能发送数据

(RO)

SPI_DMA_INFIFO_FULL 表示 DMA RX FIFO 是否就绪。

0：DMA RX FIFO 已就绪，可以接收数据

1：DMA RX FIFO 尚未就绪，不能接收数据

(RO)

SPI_DMA_SLV_SEG_TRANS_EN 配置是否使能半双工通信方式下，DMA 控制的从机连续传输。

0：不使能

1：使能

(R/W)

SPI_SLV_RX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA RX buffer 小于实际接收的数据长度：

1：后续 Wr_DMA 传输事务中传输的数据都不接收；

0: 当前 Wr_DMA 传输事务中传输的数据不接收，但在后续的 Wr_DMA 传输事务中：

- 如果 DMA RX buffer 长度不为 0，则后续 Wr_DMA 传输事务中传输的数据会被接收。
- 如果 DMA RX buffer 长度为 0，则后续 Wr_DMA 传输事务中传输的数据不会被接收。

(R/W)

SPI_SLV_TX_SEG_TRANS_CLR_EN 在从机连续传输中，如果 DMA TX buffer 小于实际发送的数据长度：

1：后续传输事务中传输的数据都不更新，即旧数据被重复发送；

0：当前传输事务中传输的数据不更新，但在后续的传输事务中：

- 如果有新的数据填充到 DMA TX FIFO，则将发送新数据。
- 如果没有新的数据填充到 DMA TX FIFO，则没有新数据被发送。

(R/W)

见下页...

Register 26.9. SPI_DMA_CONF_REG (0x0030)

接上页...

SPI_RX_EOF_EN 配置 SPI DMA 上报 eof 的中断模式

1：在 DMA 控制的数据传输过程中，如果 DMA 传输的数据比特数等于 (SPI_MS_DATA_BITLEN + 1)，则硬件会置位 GDMA_IN_SUC_EOF_CH*n*_INT_RAW。

0：在单次传输中，GDMA_IN_SUC_EOF_CH*n*_INT_RAW 由 SPI_TRANS_DONE_INT 事件置位；或在分段配置传输模式下，由 SPI_DMA_SEG_TRANS_DONE_INT 事件置位。

(R/W)

SPI_DMA_RX_ENA 配置是否使能 SPI DMA 控制的接收数据模式。

0：不使能

1：使能

(R/W)

SPI_DMA_TX_ENA 配置是否使能 SPI DMA 控制的发送数据模式。

0：不使能

1：使能

(R/W)

SPI_RX_AFIFO_RST 配置是否复位图 26-4 和图 26-5 中的 spi_rx_afifo。

0：不复位

1：复位

spi_rx_afifo 将在 SPI 主机和从机传输中用于接收数据。(WT)

SPI_BUF_AFIFO_RST 配置是否复位图 26-4 和图 26-5 中的 buf_tx_afifo。

0：不复位

1：复位

buf_tx_afifo 将在 CPU 控制的从机传输或主机传输中用于发送数据。(WT)

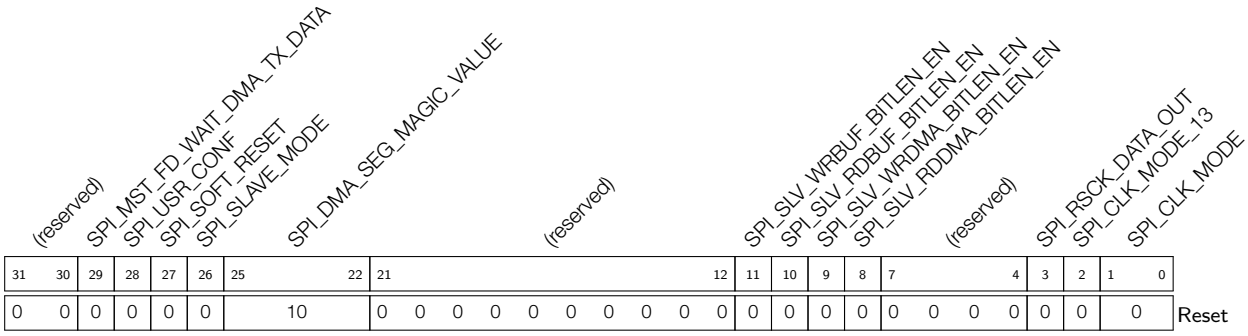
SPI_DMA_AFIFO_RST 配置是否复位图 26-4 和图 26-5 中的 dma_tx_afifo。

0：不复位

1：复位

dma_tx_afifo 在 DMA 控制的从机传输中用于发送数据。(WT)

Register 26.10. SPI_SLAVE_REG (0x00E0)



SPI_CLK_MODE 配置 SPI 时钟模式。

0：CS 信号无效时，SPI 时钟关闭；

1：CS 信号无效后，SPI 时钟延迟一个时钟周期；

2：CS 信号无效后，SPI 时钟延迟两个时钟周期；

3：SPI 时钟一直有效。

可在 CONF 阶段配置。(R/W)

SPI_CLK_MODE_13 配置时钟模式。

0：支持 SPI 时钟模式 0 或 2，见表 26-17。

1：支持 SPI 时钟模式 1 或 3，见表 26-17。

(R/W)

SPI_RSCK_DATA_OUT 配置输出数据的时钟沿。

0：在 TSCK 上升沿输出数据

1：在 RSCK 上升沿输出数据

(R/W)

SPI_SLV_RDDMA_BITLEN_EN 配置在 DMA 控制的 Rd_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

SPI_SLV_WRDMA_BITLEN_EN 配置在 DMA 控制的 Wr_DMA 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_SLV_RDBUF_BITLEN_EN 配置在 CPU 控制的 Rd_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机读取从机的数据位长。

0：不使用

1：使用

(R/W)

见下页...

Register 26.10. SPI_SLAVE_REG (0x00E0)

接上页...

SPI_SLV_WRBUF_BITLEN_EN 配置在 CPU 控制的 Wr_BUF 传输过程中，是否使用 SPI_SLV_DATA_BITLEN 存储主机向从机写数据的位长。

0：不使用

1：使用

(R/W)

SPI_DMA_SEG_MAGIC_VALUE 配置 DMA 控制的分段配置传输中位图表的 Magic 值。(R/W)

SPI_SLAVE_MODE 配置 SPI 工作模式。

0：主机

1：从机

(R/W)

SPI_SOFT_RESET 配置是否软件复位 SPI 时钟线、CS 线和数据线。

0：不复位

1：复位

可在 CONF 阶段配置。(WT)

SPI_USR_CONF 配置是否使能当前 DMA 控制分段配置传输的 CONF 阶段。

0：无效，表明当前传输不是分段配置传输

1：使能，开始分段配置传输

(R/W)

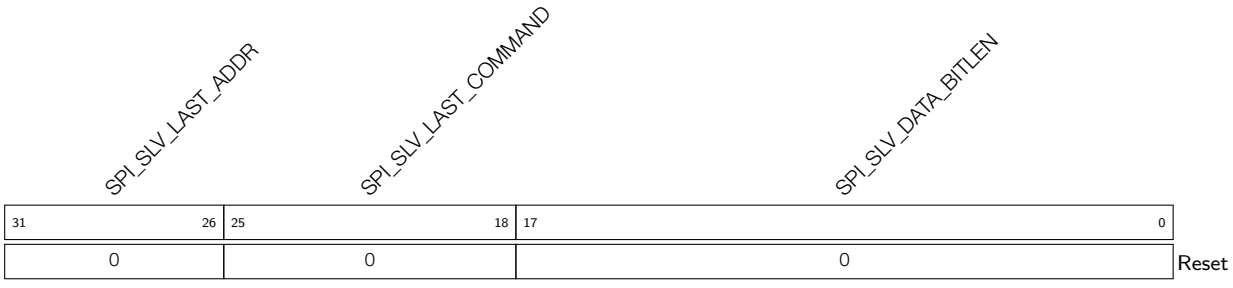
SPI_MST_FD_WAIT_DMA_TX_DATA 配置在主机全双工模式下 GP-SPI2 是否先等待 DMA TX 数据准备好之后，再开始 SPI 传输。

0：GP-SPI2 无需等待 DMA TX 数据即可开始 SPI 传输

1：等待

(R/W)

Register 26.11. SPI_SLAVE1_REG (0x00E4)

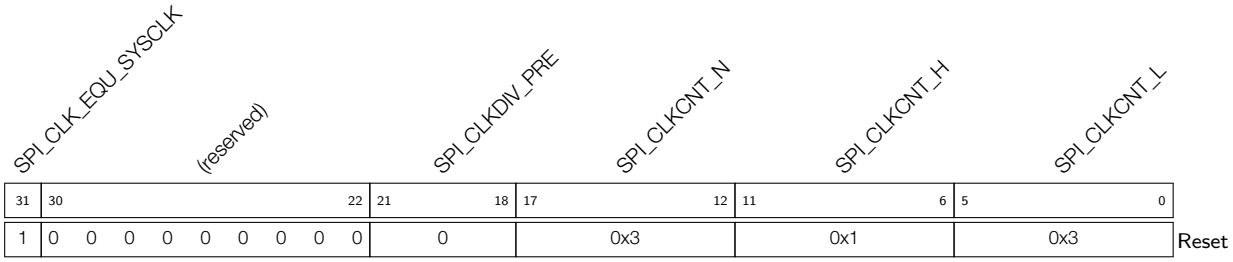


SPI_SLV_DATA_BITLEN 配置在 SPI 从机全双工和半双工传输中，传输的数据位长。(R/W/SS)

SPI_SLV_LAST_COMMAND 配置从机的命令值。(R/W/SS)

SPI_SLV_LAST_ADDR 配置从机的地址值。(R/W/SS)

Register 26.12. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L 用作主机时，必须与 SPI_CLKCNT_N 相等。在从机模式下，必须为 0。可在 CONF 阶段配置。(R/W)

SPI_CLKCNT_H 配置用作主机时 SPI_CLK（高电平）的占空比。

建议将此值配置为 floor ((SPI_CLKCNT_N + 1)/2 - 1)。floor() 表示向下取整值，例如 floor(2.2) = 2。

从机模式下，必须为 0。可在 CONF 阶段配置。(R/W)

SPI_CLKCNT_N 配置用作主机时 SPI_CLK 的分频系数。

SPI_CLK 频率为 $f_{clk\_spi\_mst}$/(SPI_CLKDIV_PRE + 1)/(SPI_CLKCNT_N + 1)。可在 CONF 阶段配置。(R/W)

SPI_CLKDIV_PRE 配置用作主机时 SPI_CLK 的预分频系数。

可在 CONF 阶段配置。(R/W)

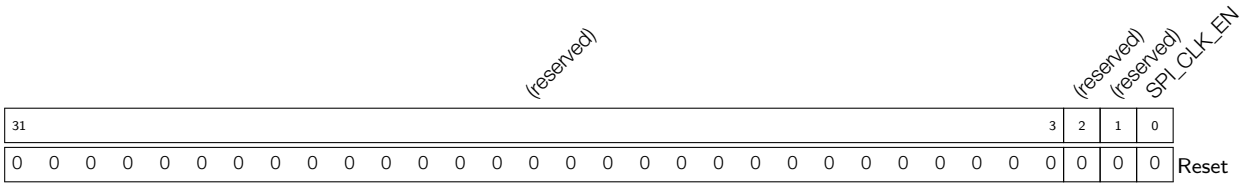
SPI_CLK_EQU_SYSCLK 配置用作主机时 SPI_CLK 频率是否与 APB_CLK 频率相同。

0：SPI_CLK 为 APB_CLK 的分频时钟

1：SPI_CLK 与 APB_CLK 频率相同

可在 CONF 阶段配置。(R/W)

Register 26.13. SPI_CLK_GATE_REG (0x00E8)



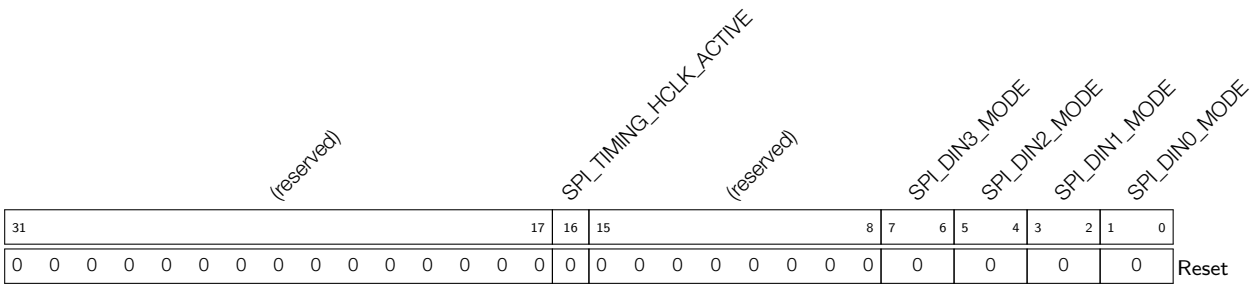
SPI_CLK_EN 配置是否使能时钟门控。

0：不使能

1：使能

(R/W)

Register 26.14. SPI_DIN_MODE_REG (0x0024)



SPI_DIN0_MODE 配置 FSPID 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN0_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN0_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。(R/W)

SPI_DIN1_MODE 配置 FSPIQ 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN1_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN1_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。(R/W)

SPI_DIN2_MODE 配置 FSPIWP 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN2_NUM + 1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN2_NUM + 1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。(R/W)

SPI_DIN3_MODE 配置 FSPIHD 信号的输入模式。

0：无输入延迟

1：在 clk_spi_mst 的第 (SPI_DIN3_NUM+1) 个下降沿输入

2：在 clk_hclk 的 (SPI_DIN3_NUM+1) 个上升沿延时加上一个 clk_spi_mst 上升沿延时的时刻输入

3：在 clk_hclk 的 (SPI_DIN3_NUM+1) 个上升沿延时加上一个 clk_spi_mst 下降沿延时的时刻输入

可在 CONF 阶段配置。(R/W)

见下页...

Register 26.14. SPI_DIN_MODE_REG (0x0024)

接上页...

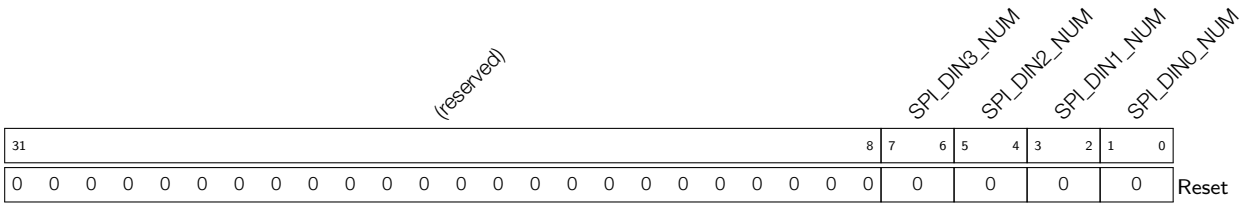
SPI_TIMING_HCLK_ACTIVE 配置是否使能 SPI 输入信号时序模块的高频时钟 HCLK。

0：不使能

1：使能

可在 CONF 阶段配置。(R/W)

Register 26.15. SPI_DIN_NUM_REG (0x0028)



SPI_DIN0_NUM 配置输入信号 FSPID 的延迟周期数。具体的延迟模式由 SPI_DIN0_MODE 配置。可在 CONF 阶段配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

SPI_DIN1_NUM 配置输入信号 FSPIQ 的延迟周期数。具体的延迟模式由 SPI_DIN1_MODE 配置。可在 CONF 阶段配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

SPI_DIN2_NUM 配置输入信号 FSPIWP 的延迟周期数。具体的延迟模式由 SPI_DIN2_MODE 配置。可在 CONF 阶段配置。

0：延迟 1 个时钟周期

1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

SPI_DIN3_NUM 配置输入信号 FSPIHD 的延迟周期数。具体的延迟模式由 SPI_DIN3_MODE 配置。可在 CONF 阶段配置。

0：延迟 1 个时钟周期

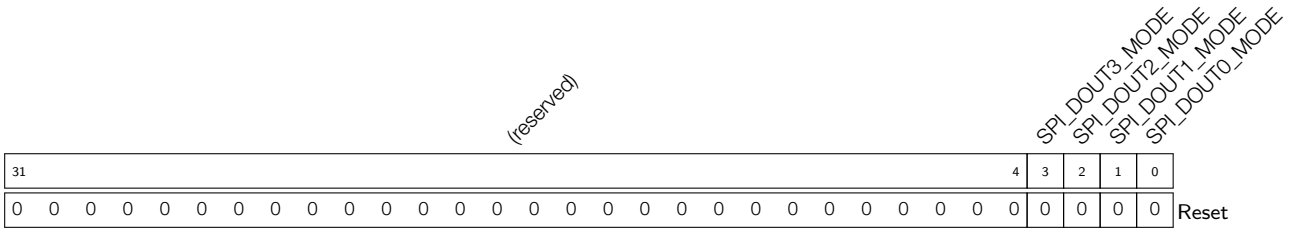
1：延迟 2 个时钟周期

2：延迟 3 个时钟周期

3：延迟 4 个时钟周期

(R/W)

Register 26.16. SPI_DOUT_MODE_REG (0x002C)



SPI_DOUT0_MODE 配置 FSPID 信号的输出模式。可在 CONF 阶段配置。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

SPI_DOUT1_MODE 配置 FSPIQ 信号的输出模式。可在 CONF 阶段配置。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

SPI_DOUT2_MODE 配置 FSPIWP 信号的输出模式。可在 CONF 阶段配置。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

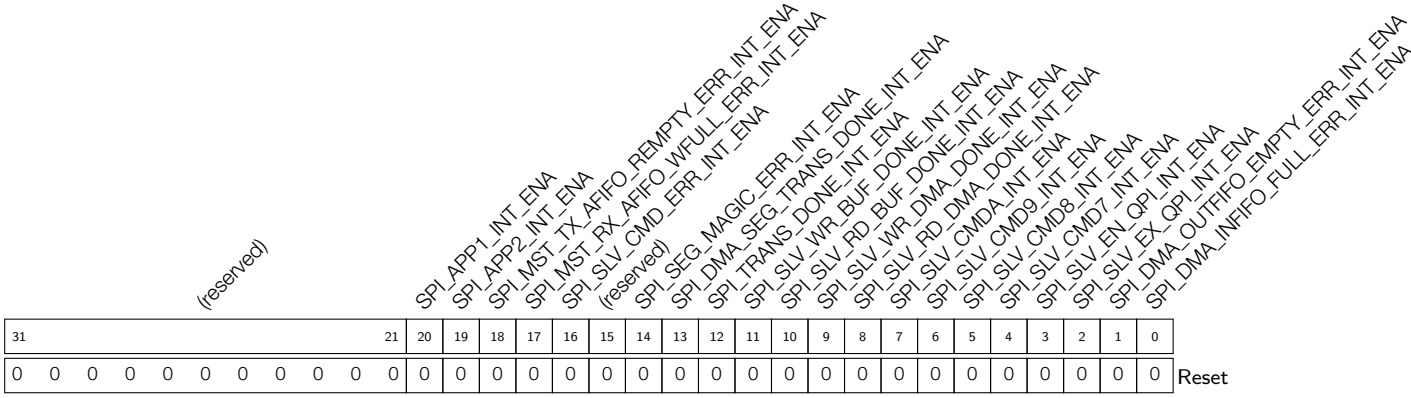
SPI_DOUT3_MODE 配置 FSPIHD 信号的输出模式。可在 CONF 阶段配置。

0：无输出延迟

1：在 SPI 模块时钟的下降沿，延迟一个时钟周期后输出

(R/W)

Register 26.17. SPI_DMA_INT_ENA_REG (0x0034)



SPI_DMA_INFIFO_FULL_ERR_INT_ENA 写 1 使能 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA 写 1 使能 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(R/W)

SPI_SLV_EX_QPI_INT_ENA 写 1 使能 SPI_SLV_EX_QPI_INT 中断。(R/W)

SPI_SLV_EN_QPI_INT_ENA 写 1 使能 SPI_SLV_EN_QPI_INT 中断。(R/W)

SPI_SLV_CMD7_INT_ENA 写 1 使能 SPI_SLV_CMD7_INT 中断。(R/W)

SPI_SLV_CMD8_INT_ENA 写 1 使能 SPI_SLV_CMD8_INT 中断。(R/W)

SPI_SLV_CMD9_INT_ENA 写 1 使能 SPI_SLV_CMD9_INT 中断。(R/W)

SPI_SLV_CMDA_INT_ENA 写 1 使能 SPI_SLV_CMDA_INT 中断。(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_DMA_DONE_INT 中断。(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_DMA_DONE_INT 中断。(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_RD_BUF_DONE_INT 中断。(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA 写 1 使能 SPI_SLV_WR_BUF_DONE_INT 中断。(R/W)

SPI_TRANS_DONE_INT_ENA 写 1 使能 SPI_TRANS_DONE_INT 中断。(R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA 写 1 使能 SPI_DMA_SEG_TRANS_DONE_INT 中断。(R/W)

SPI_SEG_MAGIC_ERR_INT_ENA 写 1 使能 SPI_SEG_MAGIC_ERR_INT 中断。(R/W)

SPI_SLV_CMD_ERR_INT_ENA 写 1 使能 SPI_SLV_CMD_ERR_INT 中断。(R/W)

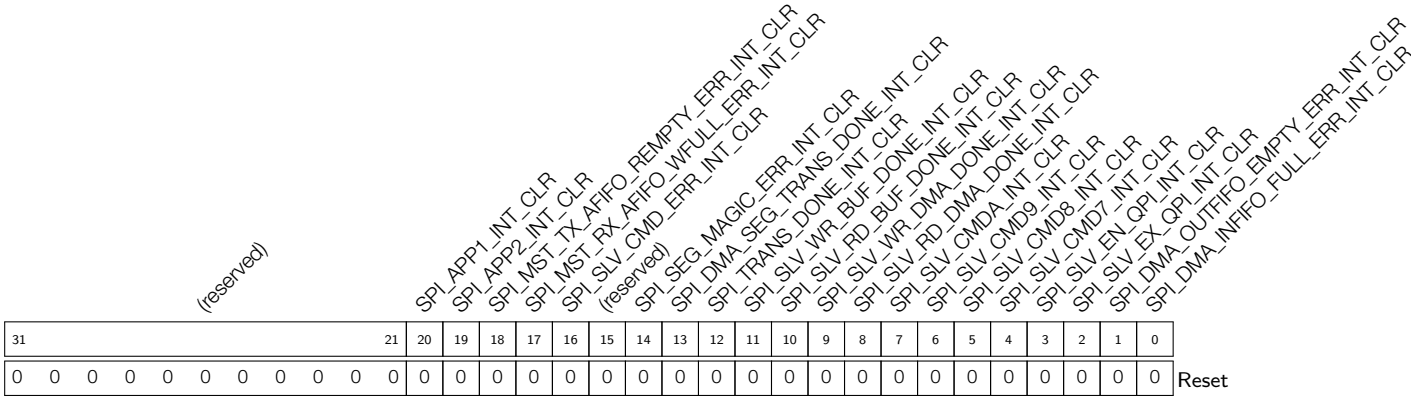
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA 写 1 使能 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。(R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA 写 1 使能 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。(R/W)

SPI_APP2_INT_ENA 写 1 使能 SPI_APP2_INT 中断。(R/W)

SPI_APP1_INT_ENA 写 1 使能 SPI_APP1_INT 中断。(R/W)

Register 26.18. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR 写 1 清除 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR 写 1 清除 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(WT)

SPI_SLV_EX_QPI_INT_CLR 写 1 清除 SPI_SLV_EX_QPI_INT 中断。(WT)

SPI_SLV_EN_QPI_INT_CLR 写 1 清除 SPI_SLV_EN_QPI_INT 中断。(WT)

SPI_SLV_CMD7_INT_CLR 写 1 清除 SPI_SLV_CMD7_INT 中断。(WT)

SPI_SLV_CMD8_INT_CLR 写 1 清除 SPI_SLV_CMD8_INT 中断。(WT)

SPI_SLV_CMD9_INT_CLR 写 1 清除 SPI_SLV_CMD9_INT 中断。(WT)

SPI_SLV_CMDA_INT_CLR 写 1 清除 SPI_SLV_CMDA_INT 中断。(WT)

SPI_SLV_RD_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_DMA_DONE_INT 中断。(WT)

SPI_SLV_WR_DMA_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_DMA_DONE_INT 中断。(WT)

SPI_SLV_RD_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_RD_BUF_DONE_INT 中断。(WT)

SPI_SLV_WR_BUF_DONE_INT_CLR 写 1 清除 SPI_SLV_WR_BUF_DONE_INT 中断。(WT)

SPI_TRANS_DONE_INT_CLR 写 1 清除 SPI_TRANS_DONE_INT 中断。(WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR 写 1 清除 SPI_DMA_SEG_TRANS_DONE_INT 中断。(WT)

SPI_SEG_MAGIC_ERR_INT_CLR 写 1 清除 SPI_SEG_MAGIC_ERR_INT 中断。(WT)

SPI_SLV_CMD_ERR_INT_CLR 写 1 清除 SPI_SLV_CMD_ERR_INT 中断。(WT)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR 写 1 清除 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR 写 1 清除 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。(WT)

SPI_APP2_INT_CLR 写 1 清除 SPI_APP2_INT 中断。(WT)

SPI_APP1_INT_CLR 写 1 清除 SPI_APP1_INT 中断。(WT)

Register 26.19. SPI_DMA_INT_RAW_REG (0x003C)

Bit(s)	Field	Reset
31:21	(reserved)	0 0 0 0 0 0 0 0 0 0 0
20	SPI_APP1_INT_RAW	0
19	SPI_APP2_INT_RAW	0
18	SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW	0
17	SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW	0
16	SPI_SLV_CMD_ERR_INT_RAW	0
15	(reserved)	0
14	SPI_SEG_MAGIC_ERR_INT_RAW	0
13	SPI_DMA_SEG_TRANS_DONE_INT_RAW	0
12	SPI_TRANS_DONE_INT_RAW	0
11	SPI_SLV_WR_BUF_DONE_INT_RAW	0
10	SPI_SLV_RD_BUF_DONE_INT_RAW	0
9	SPI_SLV_WR_DMA_DONE_INT_RAW	0
8	SPI_SLV_RD_DMA_DONE_INT_RAW	0
7	SPI_SLV_CMDA_INT_RAW	0
6	SPI_SLV_CMD9_INT_RAW	0
5	SPI_SLV_CMD8_INT_RAW	0
4	SPI_SLV_CMD7_INT_RAW	0
3	SPI_SLV_EN_QPI_INT_RAW	0
2	SPI_SLV_EX_QPI_INT_RAW	0
1	SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW	0
0	SPI_DMA_INFIFO_FULL_ERR_INT_RAW	0

SPI_DMA_INFIFO_FULL_ERR_INT_RAW SPI_DMA_INFIFO_FULL_ERR_INT 的原始中断状态。(R/W/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW SPI_SLV_EX_QPI_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW SPI_SLV_EN_QPI_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_CMD7_INT_RAW SPI_SLV_CMD7_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_CMD8_INT_RAW SPI_SLV_CMD8_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_CMD9_INT_RAW SPI_SLV_CMD9_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_CMDA_INT_RAW SPI_SLV_CMDA_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW SPI_SLV_RD_DMA_DONE_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW SPI_SLV_WR_DMA_DONE_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT 的原始中断状态。(R/W/WTC/SS)

SPI_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT 的原始中断状态。(R/W/WTC/SS)

SPI_DMA_SEG_TRANS_DONE_INT_RAW SPI_DMA_SEG_TRANS_DONE_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW SPI_SEG_MAGIC_ERR_INT 的原始中断状态。(R/W/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW SPI_SLV_CMD_ERR_INT 的原始中断状态。(R/W/WTC/SS)

SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT 的原始中断状态。(R/W/WTC/SS)

见下页...

Register 26.19. SPI_DMA_INT_RAW_REG (0x003C)

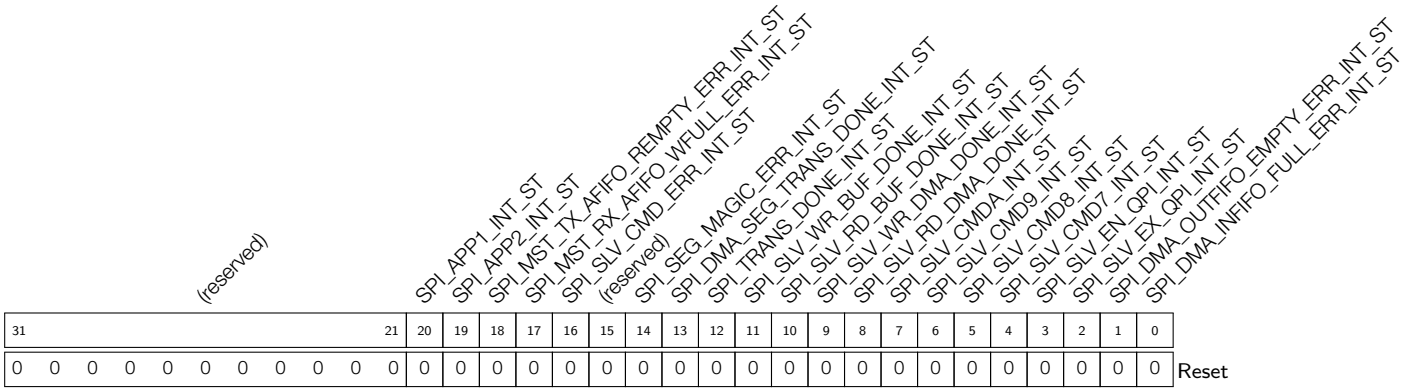
接上页...

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的原始中断状态。(R/W/WTC/SS)

SPI_APP2_INT_RAW SPI_APP2_INT 的原始中断状态。该值仅由应用控制。(R/W/WTC)

SPI_APP1_INT_RAW SPI_APP1_INT 的原始中断状态。该值仅由应用控制。(R/W/WTC)

Register 26.20. SPI_DMA_INT_ST_REG (0x0040)



SPI_DMA_INFIFO_FULL_ERR_INT_ST SPI_DMA_INFIFO_FULL_ERR_INT 的中断状态位。(RO)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ST SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断状态位。(RO)

SPI_SLV_EX_QPI_INT_ST SPI_SLV_EX_QPI_INT 的中断状态位。(RO)

SPI_SLV_EN_QPI_INT_ST SPI_SLV_EN_QPI_INT 的中断状态位。(RO)

SPI_SLV_CMD7_INT_ST SPI_SLV_CMD7_INT 的中断状态位。(RO)

SPI_SLV_CMD8_INT_ST SPI_SLV_CMD8_INT 的中断状态位。(RO)

SPI_SLV_CMD9_INT_ST SPI_SLV_CMD9_INT 的中断状态位。(RO)

SPI_SLV_CMDA_INT_ST SPI_SLV_CMDA_INT 的中断状态位。(RO)

SPI_SLV_RD_DMA_DONE_INT_ST SPI_SLV_RD_DMA_DONE_INT 的中断状态位。(RO)

SPI_SLV_WR_DMA_DONE_INT_ST SPI_SLV_WR_DMA_DONE_INT 的中断状态位。(RO)

SPI_SLV_RD_BUF_DONE_INT_ST SPI_SLV_RD_BUF_DONE_INT 的中断状态位。(RO)

SPI_SLV_WR_BUF_DONE_INT_ST SPI_SLV_WR_BUF_DONE_INT 的中断状态位。(RO)

SPI_TRANS_DONE_INT_ST SPI_TRANS_DONE_INT 的中断状态位。(RO)

SPI_DMA_SEG_TRANS_DONE_INT_ST SPI_DMA_SEG_TRANS_DONE_INT 的中断状态位。(RO)

SPI_SEG_MAGIC_ERR_INT_ST SPI_SEG_MAGIC_ERR_INT 的中断状态位。(RO)

SPI_SLV_CMD_ERR_INT_ST SPI_SLV_CMD_ERR_INT 的中断状态位。(RO)

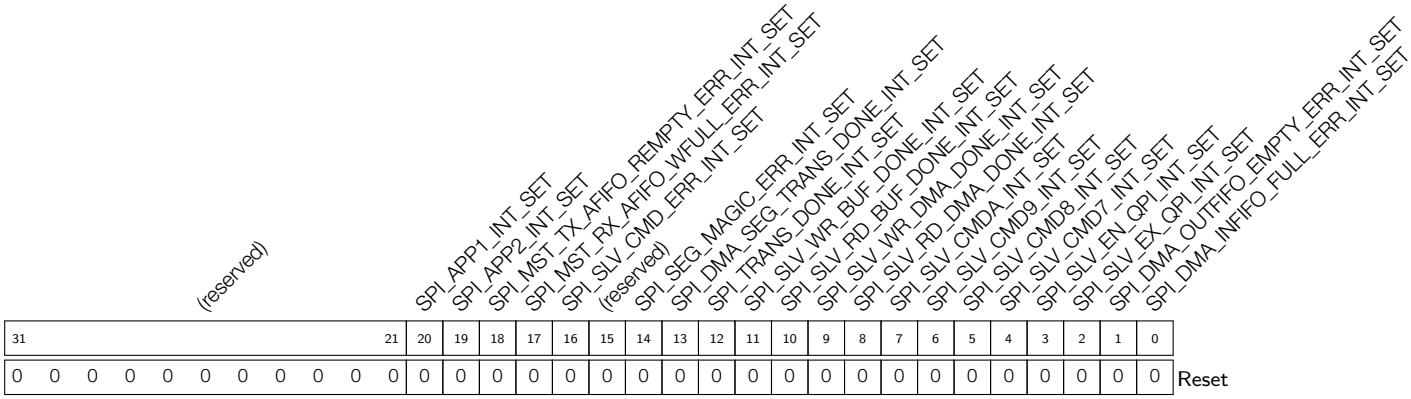
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断状态位。(RO)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ST SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断状态位。(RO)

SPI_APP2_INT_ST SPI_APP2_INT 的中断状态位。(RO)

SPI_APP1_INT_ST SPI_APP1_INT 的中断状态位。(RO)

Register 26.21. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET 写 1 置位 SPI_DMA_INFIFO_FULL_ERR_INT 中断。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET 写 1 置位 SPI_DMA_OUTFIFO_EMPTY_ERR_INT 中断。(WT)

SPI_SLV_EX_QPI_INT_SET 写 1 置位 SPI_SLV_EX_QPI_INT 中断。(WT)

SPI_SLV_EN_QPI_INT_SET 写 1 置位 SPI_SLV_EN_QPI_INT 中断。(WT)

SPI_SLV_CMD7_INT_SET 写 1 置位 SPI_SLV_CMD7_INT 中断。(WT)

SPI_SLV_CMD8_INT_SET 写 1 置位 SPI_SLV_CMD8_INT 中断。(WT)

SPI_SLV_CMD9_INT_SET 写 1 置位 SPI_SLV_CMD9_INT 中断。(WT)

SPI_SLV_CMDA_INT_SET 写 1 置位 SPI_SLV_CMDA_INT 中断。(WT)

SPI_SLV_RD_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_RD_DMA_DONE_INT 中断。(WT)

SPI_SLV_WR_DMA_DONE_INT_SET 写 1 置位 SPI_SLV_WR_DMA_DONE_INT 中断。(WT)

SPI_SLV_RD_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_RD_BUF_DONE_INT 中断。(WT)

SPI_SLV_WR_BUF_DONE_INT_SET 写 1 置位 SPI_SLV_WR_BUF_DONE_INT 中断。(WT)

SPI_TRANS_DONE_INT_SET 写 1 置位 SPI_TRANS_DONE_INT 中断。(WT)

SPI_DMA_SEG_TRANS_DONE_INT_SET SPI_DMA_SEG_TRANS_DONE_INT 中断。(WT)

SPI_SEG_MAGIC_ERR_INT_SET 写 1 置位 SPI_SEG_MAGIC_ERR_INT 中断。(WT)

SPI_SLV_CMD_ERR_INT_SET 写 1 置位 SPI_SLV_CMD_ERR_INT 中断。(WT)

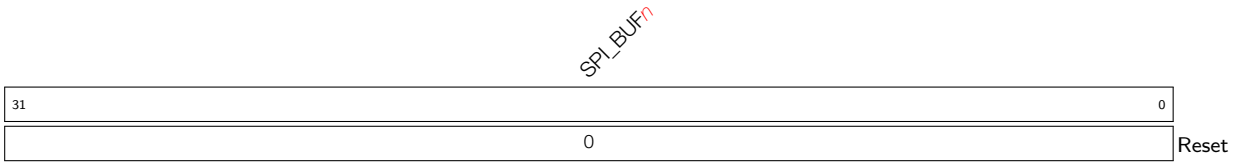
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET 写 1 置位 SPI_MST_RX_AFIFO_WFULL_ERR_INT 中断。(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET 写 1 置位 SPI_MST_TX_AFIFO_REMPTY_ERR_INT 中断。(WT)

SPI_APP2_INT_SET 写 1 置位 SPI_APP2_INT 中断。(WT)

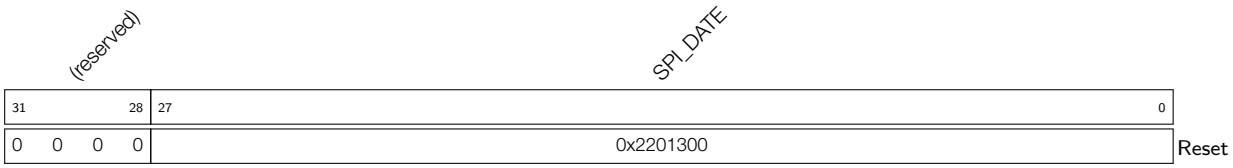
SPI_APP1_INT_SET 写 1 置位 SPI_APP1_INT 中断。(WT)

Register 26.22. SPI_W*n*_REG (*n*: 0-15) (0x0098 + 0x4n*)**



SPI_BUF*n* 数据 buffer *n*，32 位。(R/W/SS)

Register 26.23. SPI_DATE_REG (0x00F0)



SPI_DATE 版本寄存器。(R/W)

27 I2C 控制器 (I2C)

ESP32-H2 通过 I2C (Inter-Integrated Circuit) 总线和多个外部设备进行通信。多个外部设备可以共用一个 I2C 总线。ESP32-H2 共有两个既可作为主机又可作为从机的 I2C 控制器。

27.1 概述

I2C 是一个两线总线，由串行数据线 (SDA) 和串行时钟线 (SCL) 构成。这些线设置为漏极开漏 (open-drain) 输出。因此，I2C 总线上可以挂载多个外设，通常是和一个或多个主机以及一个或多个从机，但同一时刻只允许一个主机占用总线访问一个从机。

主机发出开始信号，即主机在 SCL 为高电平时拉低 SDA 线，代表通讯开始。随后主机通过 SCL 线发出 9 个时钟脉冲，前 8 个脉冲用于传输 7 位地址和 1 个读写位。如果从机地址与该 7 位地址一致，那么从机可以通过在第 9 个脉冲拉低 SDA 线来应答。接下来，根据读 / 写标志位，主机和从机之间可以传输更多的数据，根据应答位 (ACK) 的逻辑电平决定是否停止发送数据。在数据传输中，SDA 线仅在 SCL 线为低电平时才发生变化。当主机完成通讯，发送一个停止信号：主机在 SCL 为高电平时，拉高 SDA 线。如果一次通信中主机既有写操作又有读操作，则主机需在读写操作变化前，发送一个重新开始信号、从机地址和读写标志位。重新开始信号不仅用于一次通信中切换方向，也用于切换设备模式（主机或从机模式）。

27.2 主要特性

ESP32-H2 I2C 控制器具有以下特点：

- 支持主机模式和从机模式
- 支持多主机和从机通信
- 支持标准模式 (100 Kbit/s)
- 支持快速模式 (400 Kbit/s)
- 支持 7 位以及 10 位地址寻址
- 从机模式下支持拉低 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能
- 支持从机地址和从机内存或寄存器地址的双寻址模式

27.3 I2C 架构

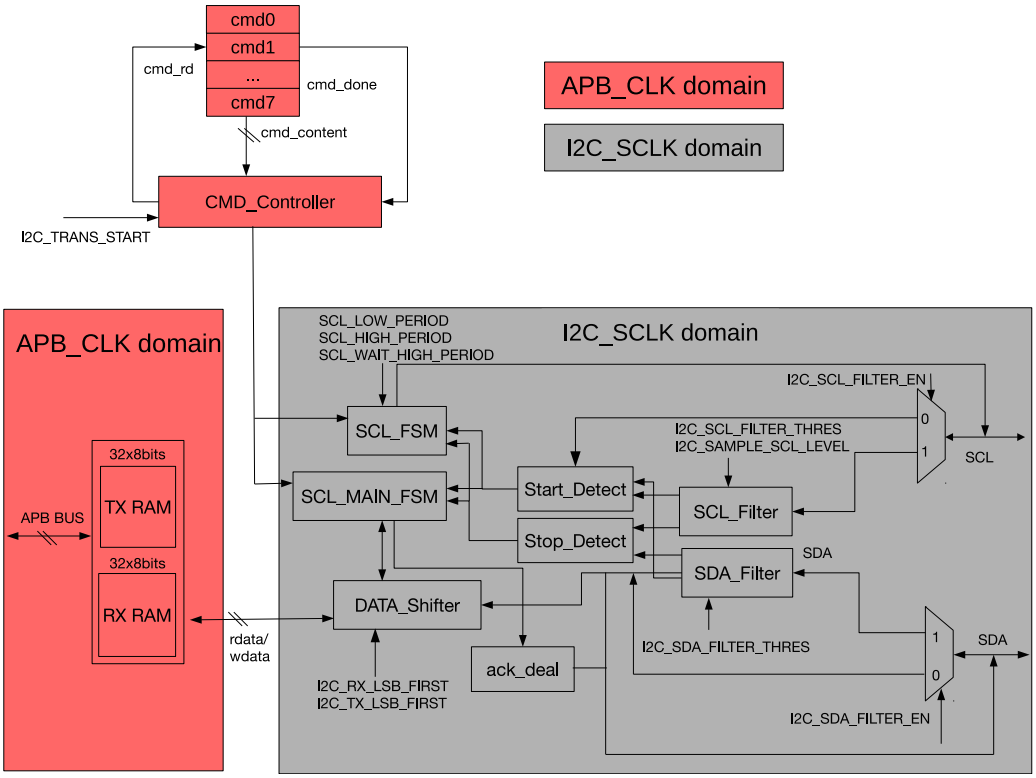


图 27-1. I2C 主机基本架构

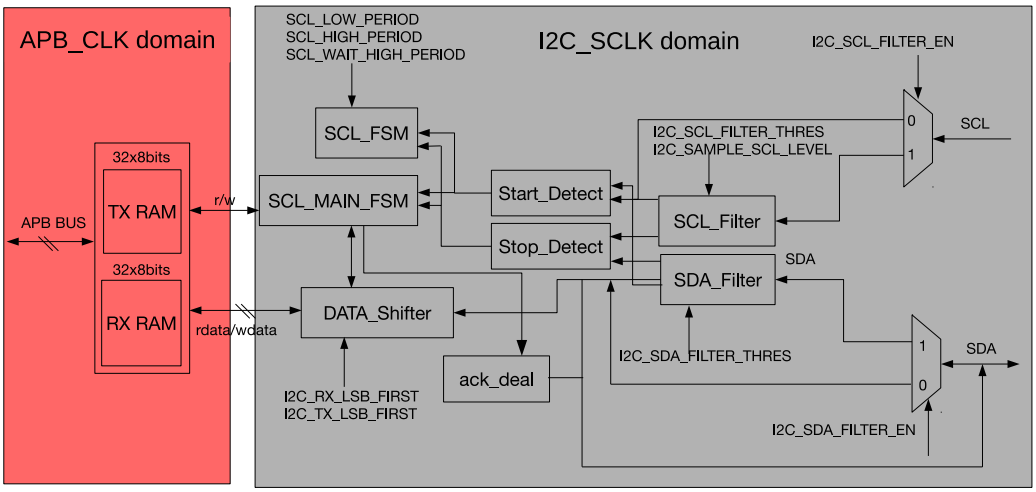


图 27-2. I2C 从机基本架构

I2C 控制器可以工作于主机模式或者从机模式，I2C_MS_MODE 寄存器用于模式选择。图 27-1 为 I2C 主机基本架构图，图 27-2 为 I2C 从机基本架构图。I2C 控制器内部包括的模块主要有：

- 接收和发送存储器 TX/RX RAM：分别用来存储 I2C 要发送和接收到的数据
- 命令控制器 CMD_Controller：产生 (R)START、STOP、WRITE、READ 和 END 指令
- SCL 时钟控制器 SCL_FSM：用来产生满足 I2C 协议的 SCL 时钟。图 27-3 和图 27-1 是 I2C 协议的时序图和对应的参数表。

- SDA 数据控制器 SCL_MAIN_FSM：用来控制 I2C 指令的执行，和 SDA 线的数据序列，还控制 ack_deal 模块生成 ACK 位并检测 SDA 线上 ACK 位的电平
- 串并转换器 DATA_Shifter：用来完成串行数据和并行数据之间的转换
- SCL 滤波器 SCL_Filter：用于消除 SCL 输入信号上的噪声
- SDA 滤波器 SDA_Filter：用于消除 SDA 输入信号上的噪声
- ACK 位控制器 ack_deal：在 SCL_MAIN_FSM 控制下生成 ACK 位并检测 SDA 线上 ACK 位的电平。

另外，还有产生 I2C 内部时钟的时钟模块，以及在 APB 总线和 I2C 模块之间同步的同步模块。

时钟模块的作用是进行时钟源选择、时钟开关和时钟分频。同步模块用来同步不同时钟域之间信号的传输。

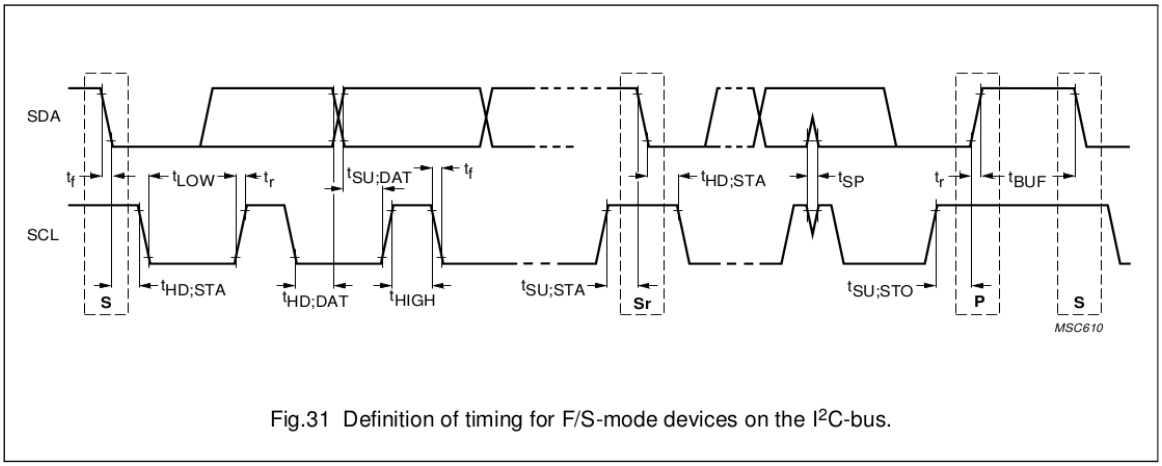


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

图 27-3. I2C 协议时序（引自 The I2C-bus specification Version 2.1 Fig.31）

PARAMETER	SYMBOL	STANDARD-MODE		FAST-MODE		UNIT
		MIN.	MAX.	MIN.	MAX.	
SCL clock frequency	f_{SCL}	0	100	0	400	kHz
Hold time (repeated) START condition. After this period, the first clock pulse is generated	$t_{HD;STA}$	4.0	–	0.6	–	µs
LOW period of the SCL clock	t_{LOW}	4.7	–	1.3	–	µs
HIGH period of the SCL clock	t_{HIGH}	4.0	–	0.6	–	µs
Set-up time for a repeated START condition	$t_{SU;STA}$	4.7	–	0.6	–	µs
Data hold time: for CBUS compatible masters (see NOTE, Section 10.1.3) for I²C-bus devices	$t_{HD;DAT}$	 5.0 0(2)	 – 3.45(3)	 – 0(2)	 – 0.9(3)	 µs µs
Data set-up time	$t_{SU;DAT}$	250	–	100(4)	–	ns
Rise time of both SDA and SCL signals	t_r	–	1000	$20 + 0.1C_b$(5)	300	ns
Fall time of both SDA and SCL signals	t_f	–	300	$20 + 0.1C_b$(5)	300	ns
Set-up time for STOP condition	$t_{SU;STO}$	4.0	–	0.6	–	µs
Bus free time between a STOP and START condition	t_{BUF}	4.7	–	1.3	–	µs

表 27-1. I2C 时序参数（引自 The I2C-bus specification Version 2.1 Table5）

27.4 功能描述

正如前文介绍，I2C 总线上可以挂载一个或多个主机以及一个或多个从机。下文介绍 ESP32-H2 I2C 控制器的操作方法。请注意，I2C 总线上其他主机或者从机的操作方法可能与 ESP32-H2 I2C 操作方法不同，具体请参考各个 I2C 控制器的技术规格书。

27.4.1 时钟配置

寄存器配置和 TX/RX RAM 部分的时钟域为 APB_CLK。I2C 主要逻辑部分，包括 SCL_FSM、SCL_MAIN_FSM、SCL_FILTER、SDA_FILTER 和 DATA_SHIFTER 的时钟域为 I2C_SCLK。

用户可以通过配置 PCR_I2C_SCLK_SEL 选择 I2C_SCLK 的时钟源：XTAL_CLK 或 RC_FAST_CLK。

- PCR_I2C_SCLK_EN 配置为 1 时打开 I2C_SCLK 的时钟源。
- PCR_I2C_SCLK_SEL 为 0 时选择时钟源 XTAL_CLK。
- PCR_I2C_SCLK_SEL 为 1 时选择时钟源 RC_FAST_CLK。

选择后的时钟经过小数分频得到 I2C 的工作时钟 I2C_SCLK，分频系数为：

$$PCR\_I2C\_SCLK\_DIV\_NUM + 1 + \frac{PCR\_I2C\_SCLK\_DIV\_A}{PCR\_I2C\_SCLK\_DIV\_B}$$

根据时序参数的限制，分频后的 I2C_SCLK 的频率要满足大于 SCL 频率的 20 倍的关系。

27.4.2 滤除 SCL 和 SDA 噪声

SCL_Filter 和 SDA_Filter 滤波器模块实现方式相同，用于滤除 SCL 及 SDA 输入信号上的噪声。通过配置 I2C_SCL_FILTER_EN 以及 I2C_SDA_FILTER_EN 寄存器可以开启或关闭滤波器。

以 SCL_Filter 为例，当使能 SCL_Filter 功能时，滤波器会连续采样输入信号 SCL，如果输入信号在连续 I2C_SCL_FILTER_THRES 个 I2C_SCLK 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，SCL_Filter 和 SDA_Filter 滤波器分别会过滤脉冲宽度小于 I2C_SCL_FILTER_THRES 和 I2C_SDA_FILTER_THRES 个 I2C_SCLK 时钟周期的线路毛刺。

27.4.3 SCL 时钟拉伸

从机模式下，可以通过拉低 SCL 线实现时钟拉伸，在此期间数据传输暂停，可以用来给软件足够的时间处理数据。置位 I2C_SLAVE_SCL_STRETCH_EN 位使能 SCL 时钟拉伸功能，配置 I2C_STRETCH_PROTECT_NUM 字段来控制 SCL 拉伸后释放的时长以防出现时序错误。出现以下四种情况时从机可以选择拉低 SCL 线实现时钟拉伸：

1. 地址命中：从机模式下，从机地址与主机发送到 SDA 线上的地址相匹配，且读写标志位为 1。
2. 写满：从机模式下，I2C 控制器的 RX RAM 为满。注意，从机在接收少于 FIFO 深度个字节时（ESP32-H2 I2C 的 FIFO 深度为 32 字节），可以不开启时钟拉伸功能；当要接收大于等于 FIFO 深度个字节时，可以通过 FIFO 阈值中断写 RAM 的乒乓操作，或者开始时钟拉伸功能，给软件提供处理时间。开启时钟拉伸功能时，必须将 I2C_RX_FULL_ACK_LEVEL 置 0，确保时钟拉伸功能正常使用，否则可能会出现不可预计的后果。
3. 读空：从机模式下，I2C 控制器要发送数据，但 TX RAM 为空。
4. 发送 ACK 时：从机模式下置位 I2C_SLAVE_BYTE_ACK_CTL_EN，从机会在发送 ACK 时拉低 SCL。软件在此阶段进行一些操作，如数据校验，并通过配置 I2C_SLAVE_BYTE_ACK_LVL 控制将要发送的 ACK 的

电平高低。要注意的是，当出现从机接收的 RX RAM 满时，要发送的 ACK 电平将由 I2C_RX_FULL_ACK_LEVEL 而不是 I2C_SLAVE_BYTE_ACK_LVL 决定。此时同样需要将 I2C_RX_FULL_ACK_LEVEL 置 0，以保证 SCL 时钟拉伸功能的正常产生。

产生时钟拉伸时，软件可读取 I2C_STRETCH_CAUSE 位获取 SCL 时钟拉伸的原因。置位 I2C_SLAVE_SCL_STRETCH_CLR 位清除 SCL 时钟拉伸。

27.4.4 SCL 空闲时产生 SCL 脉冲

通常情况下，在 I2C 总线空闲时，SCL 线一直为高。ESP32-H2 I2C 支持在 I2C 主机处于空闲状态时，可编程配置产生 SCL 脉冲的功能。这个功能仅在 I2C 控制器作为主机时有效。置位 I2C_SCL_RST_SLV_EN，硬件会发送 I2C_SCL_RST_SLV_NUM 个 SCL 脉冲，之后 I2C_SCL_RST_SLV_EN 位会自动清零。

27.4.5 同步

I2C 的寄存器配置用 APB 时钟，I2C 主模块用 I2C_SCLK，这之间存在异步处理，需要增加同步的步骤将配置寄存器的值更新进入 I2C 主模块。步骤为先写配置寄存器，再向 I2C_CONF_UPGATE 位写 1。需要通过这种方法更新的配置寄存器详见表 27-2。

表 27-2. I2C 同步寄存器

配置寄存器	寄存器域	地址
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	I2C_ADDR_10BIT_RW_CHECK_EN	
	I2C_ADDR_BROADCASTING_EN	
	I2C_SDA_FORCE_OUT	
	I2C_SCL_FORCE_OUT	
	I2C_SAMPLE_SCL_LEVEL	
	I2C_RX_FULL_ACK_LEVEL	
	I2C_MS_MODE	
	I2C_TX_LSB_FIRST	
	I2C_RX_LSB_FIRST	
	I2C_ARBITRATION_EN	
I2C_TO_REG	I2C_TIME_OUT_EN	0x000C
	I2C_TIME_OUT_VALUE	
I2C_SLAVE_ADDR_REG	I2C_ADDR_10BIT_EN	0x0010
	I2C_SLAVE_ADDR	
I2C_FIFO_CONF_REG	I2C_FIFO_ADDR_CFG_EN	0x0018
I2C_SCL_SP_CONF_REG	I2C_SDA_PD_EN	0x0080
	I2C_SCL_PD_EN	
	I2C_SCL_RST_SLV_NUM	
	I2C_SCL_RST_SLV_EN	
I2C_SCL_STRETCH_CONF_REG	I2C_SLAVE_BYTE_ACK_CTL_EN	0x0084
	I2C_SLAVE_BYTE_ACK_LVL	
	I2C_SLAVE_SCL_STRETCH_EN	
	I2C_STRETCH_PROTECT_NUM	
I2C_SCL_LOW_PERIOD_REG	I2C_SCL_LOW_PERIOD	0x0000
I2C_SCL_HIGH_PERIOD_REG	I2C_WAIT_HIGH_PERIOD	0x0038

	I2C_HIGH_PERIOD	
I2C_SDA_HOLD_REG	I2C_SDA_HOLD_TIME	0x0030
I2C_SDA_SAMPLE_REG	I2C_SDA_SAMPLE_TIME	0x0034
I2C_SCL_START_HOLD_REG	I2C_SCL_START_HOLD_TIME	0x0040
I2C_SCL_RSTART_SETUP_REG	I2C_SCL_RSTART_SETUP_TIME	0x0044
I2C_SCL_STOP_HOLD_REG	I2C_SCL_STOP_HOLD_TIME	0x0048
I2C_SCL_STOP_SETUP_REG	I2C_SCL_STOP_SETUP_TIME	0x004C
I2C_SCL_ST_TIME_OUT_REG	I2C_SCL_ST_TO_I2C	0x0078
I2C_SCL_MAIN_ST_TIME_OUT_REG	I2C_SCL_MAIN_ST_TO_I2C	0x007C
I2C_FILTER_CFG_REG	I2C_SCL_FILTER_EN	0x0050
	I2C_SCL_FILTER_THRES	
	I2C_SDA_FILTER_EN	
	I2C_SDA_FILTER_THRES	

27.4.6 漏级开路输出

SCL 及 SDA 线采用漏级开路的驱动方式。I2C 控制器有两种配置方式实现漏级开路驱动方式：

1. 置位 I2C_SCL_FORCE_OUT、I2C_SDA_FORCE_OUT 并配置相应 SCL 及 SDA PAD 的 GPIO_PIN*n*_PAD_DRIVER 寄存器为漏级开路驱动。
2. 清零 I2C_SCL_FORCE_OUT 以及 I2C_SDA_FORCE_OUT。

SCL 和 SDA 配置成开漏方式时，从低电平转向高电平的时间会较长，这个转变时间由线上的上拉电阻以及电容共同决定。开漏模式下，I2C 输出频率的占空比受限于 SCL 上拉速度，主要受 SCL 的速度限制。

另外，在 I2C_SCL_FORCE_OUT 和 I2C_SCL_PD_EN 置 1 时，可以强制拉低 SCL 线；在 I2C_SDA_FORCE_OUT 和 I2C_SDA_PD_EN 置 1 时，可以强制拉低 SDA 线。

27.4.7 时序参数配置

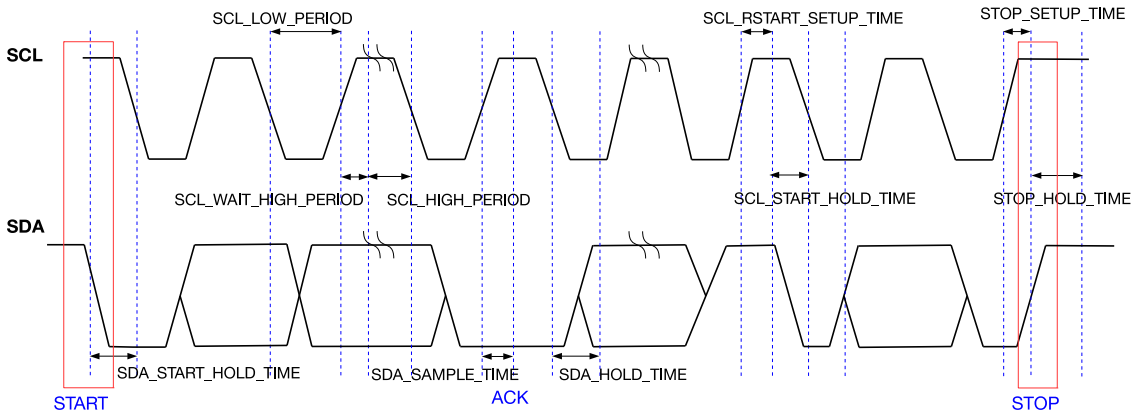


图 27-4. I2C 时序图

图 27-4 为实现 I2C 协议的 I2C 主机的时序图，图中的寄存器均用来配置时序参数。I2C 控制器的 START 位、STOP 位、数据保持时间、数据采样时间、SCL 上升沿等待时间等时序均可以通过图 27-4 中所示的寄存器进行配置。这些寄存器以模块时钟 (I2C_SCLK) 为单位，与各时序参数（见表 27-1）的对应关系为：

1. $t_{LOW} = (I2C\_SCL\_LOW\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
2. $t_{HIGH} = (I2C\_SCL\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
3. $t_{SU:STA} = (I2C\_SCL\_RSTART\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
4. $t_{HD:STA} = (I2C\_SCL\_START\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
5. $t_r = (I2C\_SCL\_WAIT\_HIGH\_PERIOD + 1) \cdot T_{I2C\_SCLK}$
6. $t_{SU:STO} = (I2C\_SCL\_STOP\_SETUP\_TIME + 1) \cdot T_{I2C\_SCLK}$
7. $t_{BUF} = (I2C\_SCL\_STOP\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
8. $t_{HD:DAT} = (I2C\_SDA\_HOLD\_TIME + 1) \cdot T_{I2C\_SCLK}$
9. $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

根据在何种模式下有意义，下列时序寄存器可分为两组：

- 主机模式：
  1. I2C_SCL_START_HOLD_TIME：生成 I2C 协议中的 start 位时，SDA 信号拉低到 SCL 信号拉低的时间间隔。该时间间隔为（I2C_SCL_START_HOLD_TIME +1）个模块时钟周期。仅控制器工作在主机模式时有意义。
  2. I2C_SCL_LOW_PERIOD：SCL 低电平持续时间。SCL 低电平时间为（I2C_SCL_LOW_PERIOD + 1）个模块时钟周期。该寄存器仅在控制器工作在主机模式时有意义。

     SCL 低电平时间会在以下情况下延长：
     - 当 I2C 为主机时，外设拉低 SCL
     - I2C 控制器执行 END 命令拉低 SCL
     - I2C 为从机时，发生 SCL 时钟拉伸
  3. I2C_SCL_WAIT_HIGH_PERIOD：等待 SCL 线拉高的模块时钟周期数。请确保在该时间内 SCL 线可以完成上拉。否则会导致 SCL 高电平持续时间不可预测。仅控制器工作在主机模式时有意义。
  4. I2C_SCL_HIGH_PERIOD：SCL 线拉高后维持高电平的模块时钟周期数。仅控制器工作在主机模式时有意义。当 SCL 线在 I2C_SCL_WAIT_HIGH_PERIOD + 1 个模块时钟内完成上拉，则 SCL 线的频率为：

     $$f_{scl} = \frac{f_{\text{I2C\_SCLK}}}{\text{I2C\_SCL\_LOW\_PERIOD} + \text{I2C\_SCL\_HIGH\_PERIOD} + \text{I2C\_SCL\_WAIT\_HIGH\_PERIOD} + 3 + \text{I2C\_SCL\_FILTER\_THRES}}$$

     其中 3 个时钟周期是对 SCL 进行同步所需的时间，如果打开了 SCL 滤波功能，则还需要加上滤波带来的延迟 I2C_SCL_FILTER_THRES。用 I2C_SCL_WAIT_HIGH_PERIOD + 1 个模块时钟模拟的 SCL 上拉时间受到上拉电阻、IO 驱动能力、SCL 线上电容等等的影响，可能出现测试的实际频率和理论频率有偏差的情况。此时可以通过调整 I2C_SCL_WAIT_HIGH_PERIOD 的值来使实际频率和理论频率接近。
- 主机模式和从机模式：
  1. I2C_SDA_SAMPLE_TIME：SCL 上升沿到采样 SDA 线电平值的时间间隔。推荐设置在 SCL 高电平持续时间的中间值，以保证能够正确采样到 SDA 线上电平。控制器工作在主机模式及从机模式时都有意义。
  2. I2C_SDA_HOLD_TIME：SDA 输出数据变化与 SCL 下降沿的时间间隔。控制器工作在主机模式及从机模式时都有意义。

根据时序参数的限制，对时序寄存器的配置范围也有约束。

1. $\frac{f_{I2C\_SCLK}}{f_{SCL}} > 20$
2. $3 \times f_{I2C\_SCLK} \leq (I2C\_SDA\_HOLD\_TIME - 4) \times f_{APB\_CLK}$
3. I2C_SDA_HOLD_TIME + I2C_SCL_START_HOLD_TIME > SDA_FILTER_THRES + 3
4. I2C_SCL_WAIT_HIGH_PERIOD < I2C_SDA_SAMPLE_TIME < I2C_SCL_HIGH_PERIOD
5. I2C_SDA_SAMPLE_TIME < I2C_SCL_WAIT_HIGH_PERIOD + I2C_SCL_START_HOLD_TIME + I2C_SCL_RSTART_SETUP_TIME
6. I2C_STRETCH_PROTECT_NUM + I2C_SDA_HOLD_TIME > I2C_SCL_LOW_PERIOD

27.4.8 超时控制

I2C 内部有三种超时控制，分别是对 SCL_FSM 状态、SCL_MAIN_FSM 状态以及 SCL 线状态的超时控制。其中前两种一直使能，第三种可编程配置。

当 SCL_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_ST\_TO\_I2C}$ 个时钟周期后，会触发 I2C_SCL_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_ST_TO_I2C 的配置值最大为 22，即最多在时间超过 2^{22} 个 I2C_SCLK 时钟周期后会产生超时中断。

当 SCL_MAIN_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_MAIN\_ST\_TO\_I2C}$ 个 I2C_SCLK 时钟周期后，会触发 I2C_SCL_MAIN_ST_TO_INT 中断，状态机会回到空闲状态。I2C_SCL_MAIN_ST_TO_I2C 的配置值最大为 22，即最大在时间超过 2^{22} 个模块时钟后会产生超时中断。

使能 I2C_TIME_OUT_EN 打开 SCL 线状态的超时控制。当 SCL 线状态长时间维持同一电平不变，且时间超过 $2^{I2C\_TIME\_OUT\_VALUE}$ 后，会触发 I2C_TIME_OUT_INT 中断，I2C 总线回到空闲状态。

27.4.9 指令配置

I2C 控制器工作于主机模式时，CMD_Controller 会依次从八个命令寄存器中读出命令并按照命令来控制 SCL_FSM 及 SCL_MAIN_FSM。

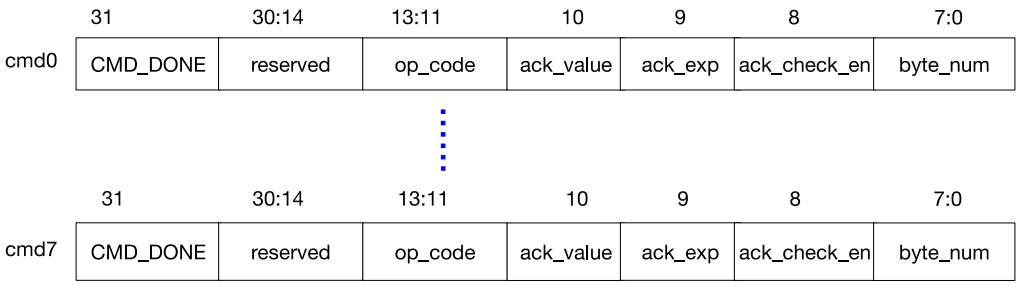


图 27-5. I2C 命令寄存器结构

命令寄存器只在 I2C 控制器工作于主机模式时才有效，其内部结构如图 27-5 所示。命令寄存器的参数为:

1. CMD_DONE: 命令执行完成标识。每条命令执行完硬件会将对应命令寄存器中的 CMD_DONE 置 1。软件可以通过读取每条命令的 CMD_DONE 位来判断该命令是否执行完毕。每次更新命令时，软件需要将 CMD_DONE 位清零。
2. op_code: 命令编码，共有五种命令：

- WRITE: op_code 等于 1 时为 WRITE 命令，该命令指示 I2C 控制器向从机发送从机地址、被访问的寄存器地址（仅限双寻址模式）、数据。
- STOP: op_code 等于 2 时为 STOP 命令，该命令指示 I2C 控制器发送 I2C 协议中的 STOP 位。此条命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件再次启动 CMD_Controller 后，会重新从命令寄存器 0 开始去取指令。
- READ: op_code 等于 3 时为 READ 命令，该命令指示 I2C 控制器从从机读取数据。
- END: op_code 等于 4 时为 END 命令，该命令指示 I2C 控制器将 SCL 信号拉低，暂停 I2C 通信。该命令也标识本次命令序列执行完成，CMD_Controller 将会停止取指令。软件在更新命令寄存器和 RAM 数据后可重新启动 CMD_Controller，继续进行 I2C 协议传输。再次启动后 CMD_Controller 会重新从命令寄存器 0 开始去取指令。
- RSTART: op_code 等于 6 时为 RSTART 命令，该命令指示 I2C 控制器发送 I2C 协议中的 START 位或 RESTART 位。

3. ack_value: 该位设置读操作时 I2C 控制器在 I2C 协议中的 ACK 位发送的电平值。RSTART、STOP、END、WRITE 命令中该位没有意义。
4. ack_exp: 该位用于设置写操作时 I2C 控制器在 I2C 协议中的 ACK 位期望接收的电平值。RSTART、STOP、END、READ 命令中该位没有意义。
5. ack_check_en: 该位使能写操作中 I2C 控制器检查从机发送的 ACK 位电平与命令中的 ack_exp 是否一致。如果接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致时，I2C 主机会产生 I2C_NACK_INT 中断，停止发送数据并产生 STOP。此位为 1 时，检测从机发送的 ACK 位电平；此位为 0 时，不检测从机发送的 ACK 位电平。RSTART、STOP、END、READ 命令中该位没有意义。
6. byte_num: 读写数据的长度 (单位字节)，最大为 255，最小为 1。RSTART、STOP、END 命令中 byte_num 无意义。

每次命令序列的执行都是从命令寄存器 0 开始，到 STOP 或 END 命令结束。所以需要保证每个命令序列中必须有 STOP 或 END 命令。

一次完整的 I2C 协议传输应该起始于 START 命令，结束于 STOP 命令。可通过 END 命令将一次 I2C 协议传输分为多个命令序列来完成。每个命令序列可以改变数据传输的方向、时钟频率、从机地址和数据长度等。这样可以弥补 RAM 大小不足的问题，也可以实现更灵活的 I2C 通信。

27.4.10 TX/RX RAM 数据存储

TX/RX RAM 大小均为 32 x 8 位。TX RAM 和 RX RAM 均可以通过 FIFO 和直接地址 (non-FIFO) 两种方式访问。将 I2C_NONFIFO_EN 位设置成 0，为 FIFO 方式；I2C_NONFIFO_EN 位设置成 1 时为直接地址方式。

TX RAM 用于存储 I2C 控制器需要发送的数据。在 I2C 通信的过程中，当 I2C 控制器需要发送数据时（不包括 ACK 位响应），会依次读出 TX RAM 中的数据并串行输出到 SDA 线上。当 I2C 控制器工作于主机模式时，所有需要发送给从机的数据都必须按照发送顺序依次存储在 TX RAM 中。包括被访问的从机地址、读写标志位、被访问的寄存器地址（仅限双地址寻址模式下）、写数据。当 I2C 控制器工作于从机模式时，TX RAM 中只存放写数据。

TX RAM 可被 CPU 读写。CPU 可通过两种方式写 TX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 I2C_DATA_REG 写 TX RAM，硬件自动进行 TX RAM 写地址自增。直接地址访问是通过地址段 (I2C 基地址 + 0x100) ~(I2C 基地址 + 0x17C) 直接访问 TX RAM。TX RAM 的每一个字节占据一个字 (word) 的地址。因此，第一个字节访问地址为 I2C 基地址 + 0x100，第二字节访问地址为 I2C 基地址 + 0x104，第三字节访问地址为

I2C 基地址 + 0x108，以此类推。CPU 只可通过直接地址访问方式读 TX RAM，且 CPU 可以通过直接地址读回写入到 TX RAM 的字节。读 TX RAM 的地址和写 TX RAM 的地址相同。

RX RAM 存储的是 I2C 通信过程中，I2C 控制器接收到的数据。当 I2C 控制器工作于从机模式时，主机发送的从机地址及被访问的寄存器地址（仅限双地址寻址模式下）都不会存储在 RX RAM 中。软件可以在 I2C 通信结束后，读出 RX RAM 的值。

RX RAM 只可被 CPU 读。CPU 可通过两种方式读 RX RAM: FIFO 访问和直接地址访问。FIFO 访问方式是通过固定地址 I2C_DATA_REG 读 RX RAM，硬件自动完成 RX RAM 读地址自增。直接地址访问是通过地址段 (I2C 基地址 + 0x180) ~(I2C 基地址 + 0x1FC) 直接访问 RX RAM。RX RAM 的每一个字节占据一个字的地址。因此，第一个字节访问地址为 I2C 基地址 + 0x180，第二字节访问地址为 I2C 基地址 + 0x184，第三字节访问地址为 I2C 基地址 + 0x188，以此类推。

在 FIFO 模式下可以对 TX RAM 进行乒乓操作，来实现发送大于 FIFO 深度的数据。置位 I2C_FIFO_PRT_EN，当 RAM 中剩下的待发送数据字节数小于 I2C_TXFIFO_WM_THRHD时，会产生 I2C_TXFIFO_WM_INT 中断。软件收到中断后，继续向 I2C_DATA_REG（主机）中写数。需要保证向 TX RAM 写数或更新数据的操作提前于 I2C 发送数据的动作，否则会产生不可预计的后果。

在 FIFO 模式下也可以对 RX RAM 进行乒乓操作，来实现接收大于 FIFO 深度的数据。置位 I2C_FIFO_PRT_EN，将 I2C_RX_FULL_ACK_LEVEL 置 0。当 RAM 中收到的数据字节数大于等于 I2C_RXFIFO_WM_THRHD（从机）时，会产生 I2C_RXFIFO_WM_INT 中断。软件收到中断后，从 I2C_DATA_REG （从机）中读数。

27.4.11 数据转换

DATA_Shifter 模块用于串并转换，当 I2C 发送数据时，将 TX RAM 中的字节数据转化成比特流；当 I2C 接收数据时，将比特流转化成字节数据并存入 RX RAM。I2C_RX_LSB_FIRST 和 I2C_TX_LSB_FIRST 用于配置最高有效位或最低有效位的优先储存或传输。

27.4.12 寻址模式

ESP32-H2 I2C 支持 7 位寻址模式和 10 位寻址模式，7 位寻址和 10 位寻址可结合使用。除此之外，ESP32-H2 I2C 还支持双地址寻址模式。

假设从机地址为 SLV_ADDR。ESP32-H2 I2C 控制器可以使用 7 位寻址（SLV_ADDR[6:0]），也可以使用 10 位寻址（SLV_ADDR[9:0]）。

对于主机模式而言，7 位寻址下只要发送一个字节地址段 SLV_ADDR[6:0] 和读写标志。7 位寻址模式下，有种特殊情况是广播寻址。在从机中，将 I2C_ADDR_BROADCASTING_EN 置 1，开启广播寻址模式。当接收到主机发送的地址为广播地址 (0x00) 且读写标志位为 0 时，此时无论从机自己的地址是多少，都会响应主机。

10 位寻址需要发送两字节地址段。第一个要发送的数是从机地址的第一个 7 位 slave_addr_first_7bits 和读写标志，slave_addr_first_7bits 的值应该配置为 (0x78 | SLV_ADDR[9:8])。第二个要发送的数是 slave_addr_second_byte，slave_addr_second_byte 的值为 SLV_ADDR[7:0]。在从机中，可以通过配置 I2C_ADDR_10BIT_EN 寄存器开启 10 位寻址模式。I2C_SLAVE_ADDR 用于配置 I2C 从机地址。I2C_SLAVE_ADDR[14:7] 的值应配置为 SLV_ADDR[7:0]，I2C_SLAVE_ADDR[6:0] 的值应配置为 (0x78 | SLV_ADDR[9:8]) 。由于 10 位从机地址比 7 位地址多一个字节，所以 WRITE 命令对应的 byte_num 以及 RAM 中数据数量都相应增加 1。详细说明可参考 编程示例。

控制器处于从机模式时，还支持双地址寻址模式。双地址的第一个地址是 I2C 从机地址，第二个地址是 I2C 从机的内存地址。双地址模式下，RAM 必须采用 non-FIFO 方式访问。通过置位 I2C_FIFO_ADDR_CFG_EN 来使能双地址访问功能。当从机接收到的从机地址和内部配置的从机地址不一致时，会产生 I2C_SLAVE_ADDR_UNMATCH 中断。

27.4.13 10 位寻址的读写标志位检查

在 10 位寻址模式下，将 I2C_ADDR_10BIT_RW_CHECK_EN 置 1，会对发送的第一个数 slave_addr_first_7bits 和读写标志做检查。当读写标志不是写，此时与协议不符合，会结束传输。若不打开此功能，当读写标志不是写，还能支持继续传输，但可能出现传输错误。

27.4.14 启动控制器

对于主机，配置成主机模式和命令寄存器等相关配置后，通过向 I2C_TRANS_START 写 1，启动主机解析，执行命令序列。一组命令总是从命令寄存器 0 开始，顺序执行到 STOP 或者 END 命令。当另一组命令需要从命令寄存器 0 开始执行时，需要重新向 I2C_TRANS_START 写 1 来更新命令。

对于从机，有两种启动方式：

- 置位 I2C_SLV_TX_AUTO_START_EN，则从机会在被主机寻址后自动启动传输。
- 清零 I2C_SLV_TX_AUTO_START_EN，且在每次接收传输前必须置位 I2C_TRANS_START。

27.5 编程示例

本节提供一些典型通信场景的编程示例。ESP32-H2 中有两个 I2C 控制器，为了便于描述，下文所有图示中的 I2C 主机和从机都假定为 ESP32-H2 I2C 外设控制器。

27.5.1 I2C 主机写入从机，7 位寻址，单次命令序列

27.5.1.1 场景介绍

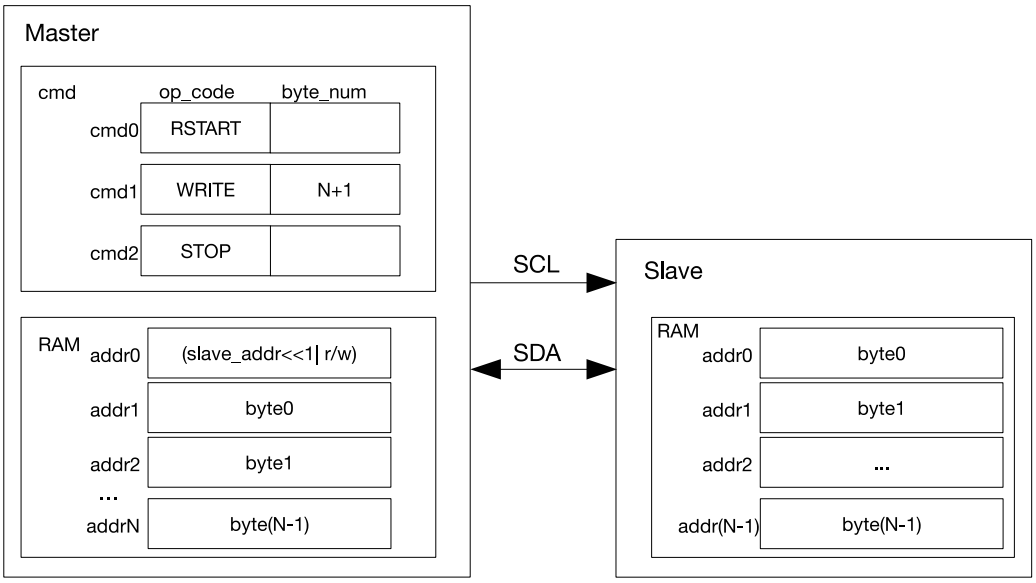


图 27-6. I2C 主机写 7 位寻址的从机

图 27-6 为 I2C 主机采用 7 位寻址写 N 个字节数据到 I2C 从机的寄存器或 RAM。如图 27-6 所示，主机 RAM 中第一个字节数据为 7 位从机地址 + 1 位读写标志位，其中读写标志位为 0 时表示写操作，接下来的连续空间存储待发送的数据。cmd 框中包含了相应的命令序列。

对于主机，在软件配置好命令序列以及 RAM 数据后，置位 I2C_TRANS_START 寄存器启动控制器进行数据传输。控制器的行为可分为四步：

1. 等待 SCL 线为高电平，以避免 SCL 线被其他主机或者从机占用。
2. 通过发送 START 位来执行 RSTART 命令。
3. 执行 WRITE 命令从 RAM 的首地址开始取出 N+1 个字节并依次发送给从机，其中第一个字节为地址。
4. 发送 STOP。当 I2C 主机完成 STOP 位的传输后，会产生 I2C_TRANS_COMPLETE_INT 中断。

27.5.1.2 配置示例

1. 参照章节 27.4.7，配置 I2C 主机和 I2C 从机的时序参数寄存器。
2. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2（主机）	STOP	—	—	—	—

5. 参考章节 27.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
6. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
10. I2C 主机发送数据，并根据 ack_check_en（主机）的配置决定是否对 ACK 信号进行检测。
11. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 27.4.10。
12. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 27.4.10。

    若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
13. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.2 I2C 主机写入从机，10 位寻址，单次命令序列

27.5.2.1 场景介绍

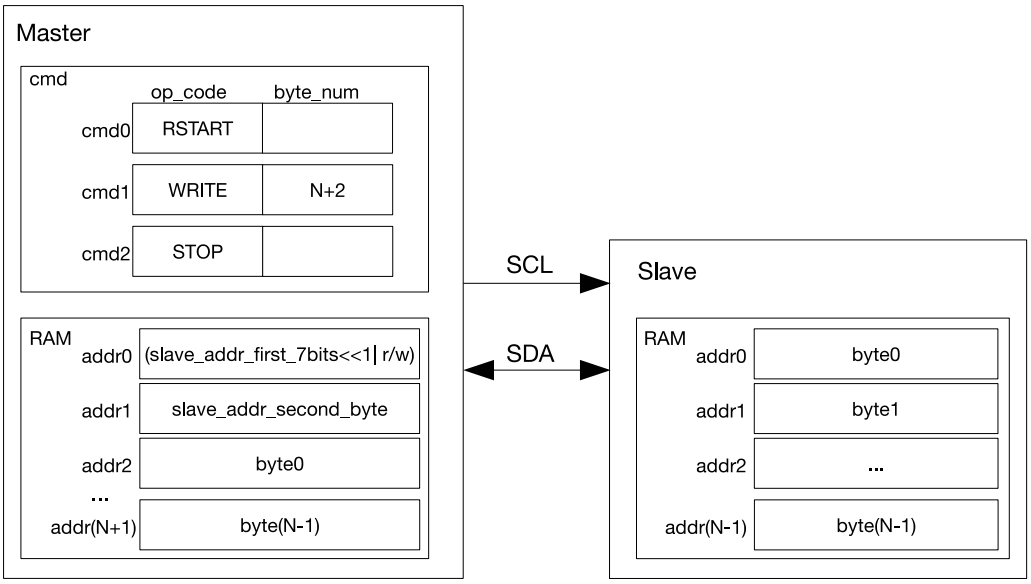


图 27-7. I2C 主机写 10 位寻址的从机

图 27-7 为 I2C 主机写 N 个字节到 10 位地址 I2C 从机的配置图。整个配置和传输过程都和 27.5.1中类似，除了在传输的开始主机在 10 位寻址模式下需要发送两字节地址段。由于 10 位从机地址比 7 位地址多一个字节，所以 WRITE 命令对应的 byte_num 以及 TX RAM 中数据数量都相应增加 1。

27.5.2.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

4. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的 10 位从机地址，并将 I2C_ADDR_10BIT_EN（从机）置 1 使能 10 位寻址模式。
5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W，第二个地址字节是I2C_SLAVE_ADDR[7:0]。之后就是要发送的数据，可选 FIFO 方式和直接访问方式。
6. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
7. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的

ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。

- 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
- 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。

9. I2C 主机发送数据，并根据 ack_check_en（主机）的配置判断是否进行 ACK 检测。
10. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 27.4.10。
11. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 从机的 RX RAM 进行乒乓操作，具体做法参照章节 27.4.10。

    若接收数据 N 大于 RX FIFO 深度，另一种处理方式是置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
12. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.3 I2C 主机写入从机，7 位双地址寻址，单次命令序列

27.5.3.1 场景介绍

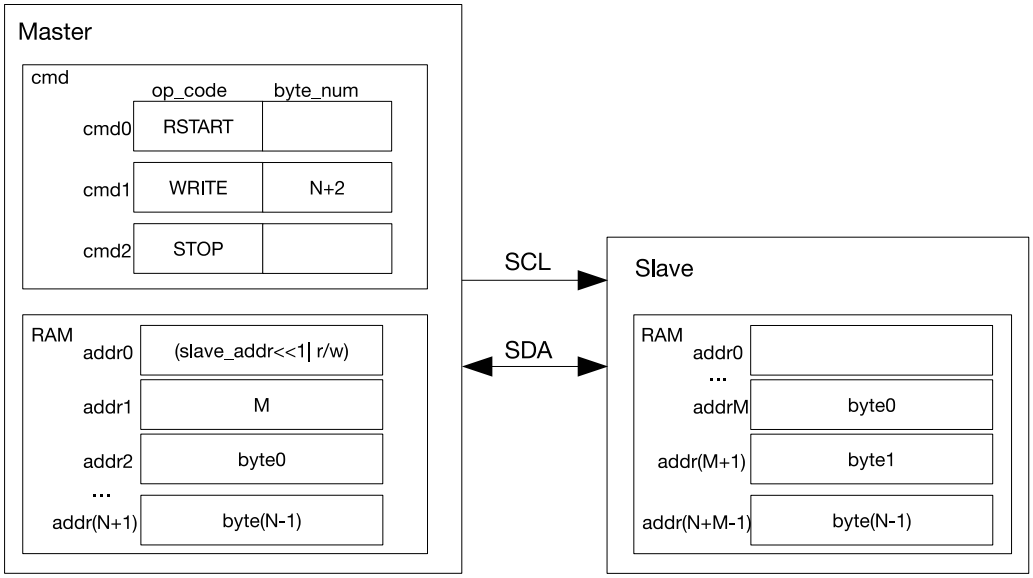


图 27-8. I2C 主机写 7 位双地址寻址从机

图27-8 为 I2C 主机采用 7 位双寻址模式写 N 个字节数据到 I2C 从机的寄存器或 RAM 的值。整个配置和传输过程都和章节 27.5.1 中类似，区别是传输的开始主机在 7 位双寻址模式下需要发送两个字节。双地址的第一个地址是 7 位从机地址，第二个地址是 I2C 从机的内存地址（即图 27-8 中的 addrM）。双地址模式下，RX RAM 必须采用 non-FIFO 方式访问。另一个区别是，I2C 从机将接收到的数据 byte0 ~ byte(N-1) 从 RX RAM 中的 addrM 开始依次存储，addrM 就是主机发送的 I2C 内存地址。当超出地址 31 后会从地址 0 开始继续存储。

27.5.3.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 设置I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2（主机）	STOP	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，可以用 FIFO 方式或直接访问方式。
6. 在 I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
10. I2C 从机接收到 I2C 主机发送的内存地址，完成 RX RAM 的地址偏移。
11. I2C 主机发送数据，并根据 ack_check_en（主机）的配置判断是否进行 ACK 检测。
12. 若发送数据 N 大于 TX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 TX RAM 进行乒乓操作，具体做法参照章节 27.4.10。
13. 若接收数据 N 大于 RX FIFO 深度，置位 I2C_SLAVE_SCL_STRETCH_EN（从机）使能 SCL 时钟拉伸，同时将 I2C_RX_FULL_ACK_LEVEL 置 0。RX RAM 满时会产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，并将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
14. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.4 I2C 主机写入从机，7 位寻址，多次命令序列

27.5.4.1 场景介绍

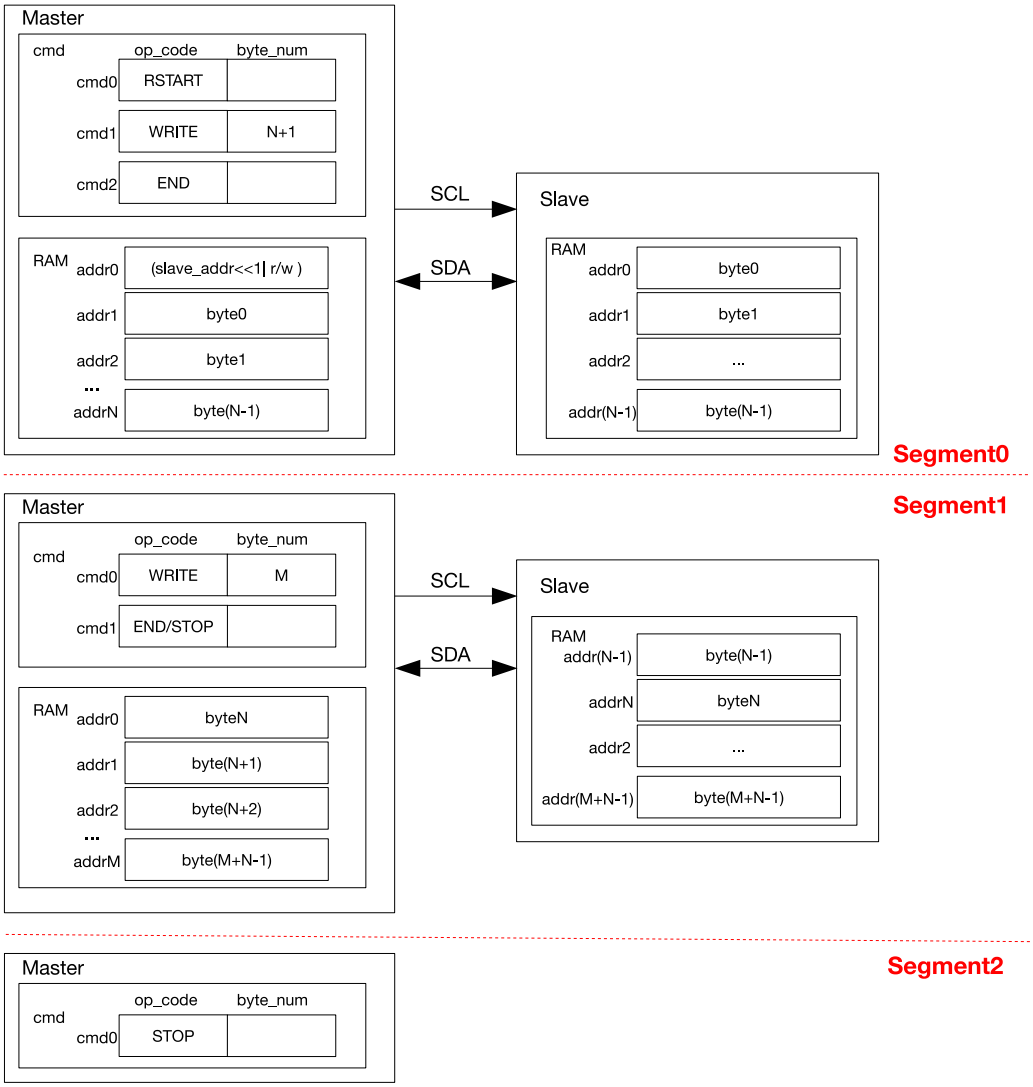


图 27-9. I2C 主机分段写 7 位寻址的从机

RAM 的大小只有 TX/RX FIFO 深度，当进行大量的数据传输时，建议使用多次命令序列进行分段传输。每次命令序列以 END 命令结束。当控制器执行 END 命令时，将拉低 SCL 线，软件此时可以更新命令序列寄存器和 RAM 的内容以用于下一次命令序列的传输。

以两段和三段传输为例，如图 27-9 所示为 I2C 主机分成三段或者两段写从机。配置 I2C 主机的命令序列如第一段所示，并且在主机的 RAM 中准备好数据，置位 I2C_TRANS_START，I2C 主机即开始数据传输。在执行到 END 命令后，I2C 主机会关闭 SCL 时钟，并将 SCL 线拉低来防止其他设备占用 I2C 总线。此时控制器产生 I2C_END_DETECT_INT 中断。

在检测到 I2C_END_DETECT_INT 中断后，软件可以更新命令序列以及 RAM 中的内容如第二段所示，并清除 I2C_END_DETECT_INT 中断。当第二段中 cmd1 为 STOP 时，不需要第三段，即为两段写从机。置位 I2C_TRANS_START 后，I2C 主机继续发送数据，并在最后发送 STOP 位。

当为三段写从机时，I2C 主机在第二段发送完数据，并检测到 I2C 主机的 I2C_END_DETECT_INT 中断后，即可配置 cmd 如第三段所示。置位 I2C_TRANS_START 后，I2C 主机即产生 STOP 位，从而停止传输。

请注意，在两个分段之间，I2C 总线上的其他主机设备不会占用总线。只有在发送了 STOP 命令后总线才会被

释放。任何情况下，置位 I2C_FSM_RST 可复位 I2C 控制器，该寄存器硬件自清 I2C_FSM_RST。

27.5.4.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
3. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2（主机）	END	—	—	—	—

4. 参考章节 27.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据。可选 FIFO 方式和直接访问方式。
5. 在 I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
6. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
7. 向 I2C_TRANS_START（主机）和 I2C_TRANS_START（从机）位写 1 开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的 I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
   - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
9. I2C 主机发送数据，并根据 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
10. 等到 I2C_END_DETECT_INT（主机）中断产生后，设置 I2C_END_DETECT_INT_CLR（主机）为 1 来清除中断。
11. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1（主机）	END/STOP	—	—	—	—

12. 向 I2C 主机的 TX RAM 写入 M 个要发送的数据，可以用 FIFO 方式或直接访问方式。
13. 向 I2C_TRANS_START（主机）位写 1 开始传输，并重复步骤 9 的流程。
14. 若指令为 STOP，I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。
15. 若 I2C_COMMAND1（主机）的指令为 END，则重复步骤 10。
16. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND1（主机）	STOP	—	—	—	—

17. 向 I2C_TRANS_START（主机）位写 1 开始传输。
18. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.5 I2C 主机读取从机，7 位寻址，单次命令序列

27.5.5.1 场景介绍

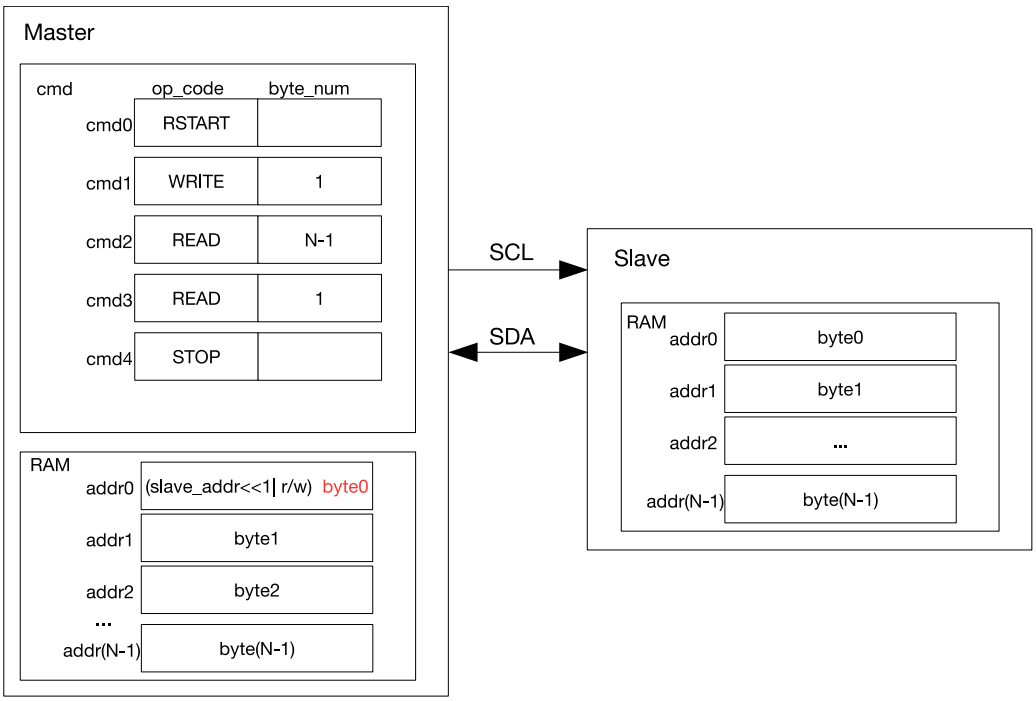


图 27-10. I2C 主机读 7 位寻址的从机

图 27-10 I2C 主机从 7 位寻址 I2C 从机读取 N 个字节数据的寄存器或 RAM 的值。cmd1 为 WRITE 命令，I2C 主机会将 I2C 从机的地址发送出去。该命令发送的字节是 7 位 I2C 从机地址以及读写标志位。读写标志位为 1 表示读操作。I2C 从机在地址匹配成功之后即开始发送数据给 I2C 主机。I2C 主机根据 READ 命令中设置的 ack_value，在接收完一个字节的数据之后回复 ACK。

图 27-10 中 READ 分成两次，I2C 主机对 cmd2 中 N-1 个数据均回复 ACK，对 cmd3 中的数据即传输的最后一个数据回复 NACK，实际使用时可以根据需要进行配置。在存储接收的数据时，I2C 主机从 RX RAM 的首地址开始存储，即为图中红色 byte0 存储的位置。

27.5.5.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num

I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	1
I2C_COMMAND2（主机）	READ	0	0	1	N-1
I2C_COMMAND3（主机）	READ	1	0	1	1
I2C_COMMAND4（主机）	STOP	—	—	—	—

5. 参考章节 27.4.10，向 I2C 主机的 TX RAM 写入从机地址。可选 FIFO 方式和直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 27.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
12. 参考章节 27.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
14. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
16. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。
17. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.6 I2C 主机读取从机，10 位寻址，单次命令序列

27.5.6.1 场景介绍

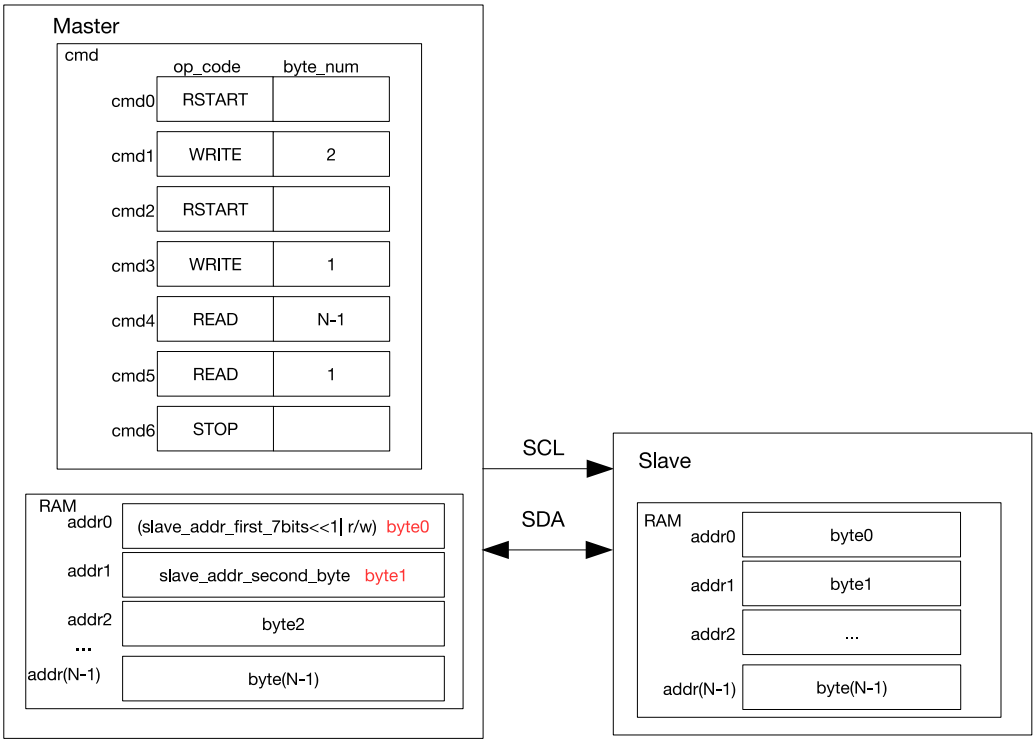


图 27-11. I2C 主机读 10 位寻址的从机

图 27-11 为 I2C 主机从 10 位寻址的 I2C 从机中读取数据的寄存器或 RAM 的值。相比于 7 位寻址，I2C 主机的第一写命令的字节数为两个字节，相应 TX RAM 中存储两个字节的 I2C 从机 10 位地址，且第一个地址字节的 R/W 位为 W（主机）。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位为 R（从机）。之后主机从从机中读取数据。两个字节地址的配置方式与章节 27.5.2 的相同。

27.5.6.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1
I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

5. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的 10 位从机地址，并将 I2C_ADDR_10BIT_EN（从机）置 1 使能 10 位寻址模式。
6. 向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 ((0x78 | I2C_SLAVE_ADDR[9:8])«1)|R/W，R/W 位为 W（主机）；第二个地址字节是I2C_SLAVE_ADDR[7:0]。第三个字节是重复的第一个地址字节加上 R/W 位，其中 R/W 位为 R（从机）。可选 FIFO 方式和直接访问方式。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 27.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. I2C 主机发送 RSTART 命令，并发送 TX RAM 里的第三个字节，即为重复的地址字节和 R 位。
12. I2C 从机重复执行步骤 10，若地址匹配，继续后面的步骤。
13. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
14. 参考章节 27.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
15. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
16. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
17. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生 I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将 I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
18. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 I2C_NACK_INT 中断，停止发送。
19. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.7 I2C 主机读取从机，7 位双寻址，单次命令序列

27.5.7.1 场景介绍

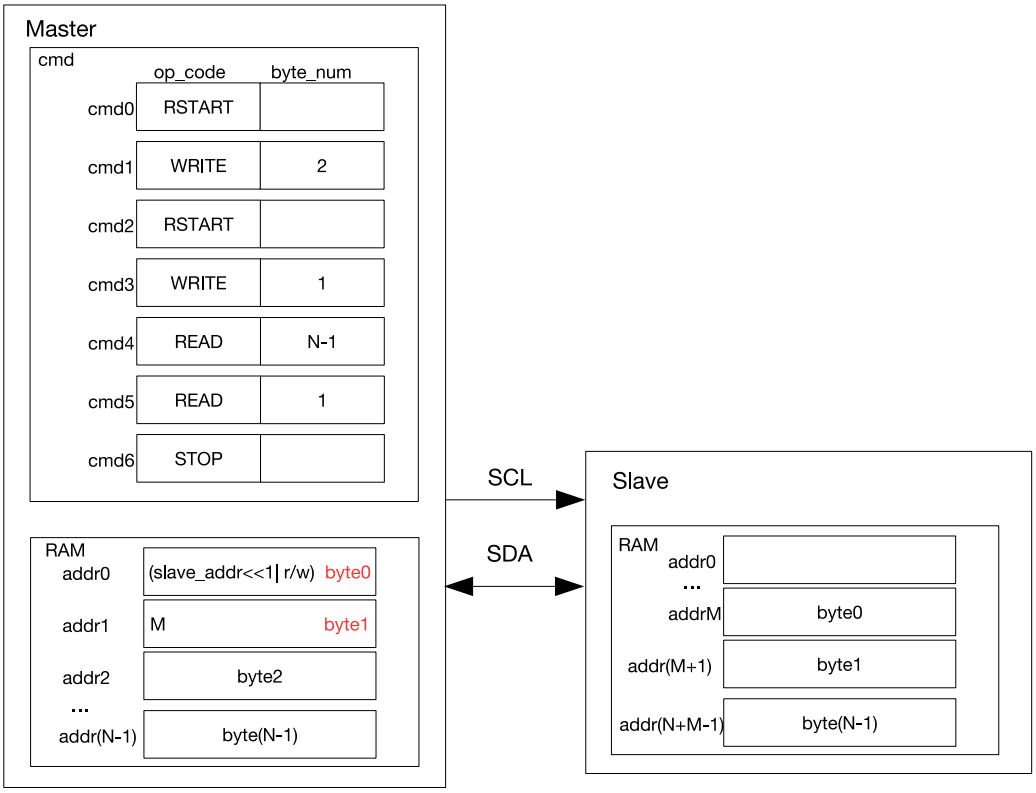


图 27-12. I2C 主机从 7 位寻址从机的 M 地址读取 N 个数据

图 27-12 为 I2C 主机从 I2C 从机中指定地址读取数据的寄存器或 RAM 的值。主机在传输开始发送 2 个地址字节，第一个地址字节是从机的 7 位地址加 R/W 位，R/W 位为 W（主机）；第二个地址字节是从机的内存地址 M。之后再次发送 RSTART，并重复发送第一个地址字节，R/W 位变为 R（从机）。之后主机从从机的 AddrM 地址开始读取数据。

27.5.7.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 设置I2C_FIFO_ADDR_CFG_EN（从机）为 1 来使能双寻址模式。
4. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
5. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	2
I2C_COMMAND2（主机）	RSTART	—	—	—	—
I2C_COMMAND3（主机）	WRITE	0	0	1	1
I2C_COMMAND4（主机）	READ	0	0	1	N-1

I2C_COMMAND5（主机）	READ	1	0	1	1
I2C_COMMAND6（主机）	STOP	—	—	—	—

6. 在I2C_SLAVE_ADDR_REG（从机）的 I2C_SLAVE_ADDR（从机）设置 I2C 从机的 7 位地址，I2C_ADDR_10BIT_EN（从机）置 0 使能 7 位寻址模式。
7. 参考章节 27.4.10，向 I2C 主机的 TX RAM 写入从机地址和要发送的数据，第一个地址字节是 (I2C_SLAVE_ADDR[6:0])«1)|R/W，R/W 位为 W（主机）；第二个地址字节是 I2C 从机的内存地址 M。第三个字节是重复的第一个地址字节加上 R/W 位，其中 R/W 位为 R（从机）。可选 FIFO 方式和直接访问方式。
8. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
9. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
10. 参考章节 27.4.14，启动 I2C 从机的传输。
11. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
12. I2C 从机接收到 I2C 主机发送的内存地址，完成 TX RAM 的地址偏移。
13. I2C 主机发送 RSTART 命令，并发送 TX RAM 里的第三个字节，即为重复的地址字节和 R 位。
14. I2C 从机重复执行步骤 11，若地址匹配，继续后面的步骤。
15. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
16. 参考章节 27.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
17. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
18. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
19. 若 I2C 主机要读取的数超过 I2C 从机的 TX FIFO 深度，当发送数据全部发完，在 I2C 从机的 TX RAM 空时产生I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C 从机的 TX RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将I2C_SLAVE_STRETCH_INT_CLR（从机）置 1，清除中断；I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
20. 当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 NACK 中断，停止发送。
21. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.5.8 I2C 主机读取从机，7 位寻址，多次命令序列

27.5.8.1 场景介绍

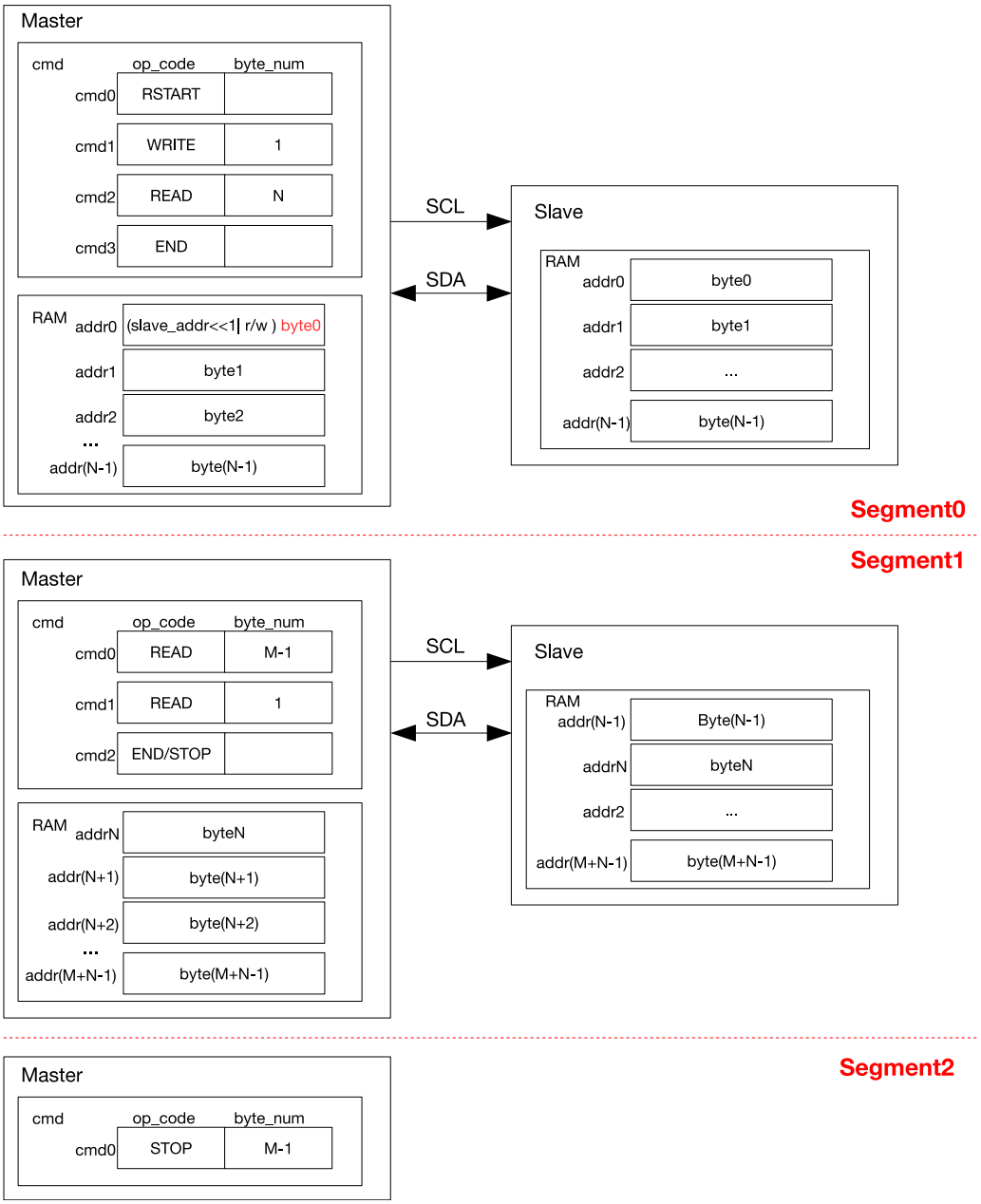


图 27-13. I2C 主机分段读 7 位寻址的从机

图 27-13 为 I2C 主机通过 END 命令分三段或者分两段，从 I2C 从机读取 N+M 个数据的流程图。

1. 第一段流程和 27-10类似，只是最后一个命令变为 END。
2. 接着在从机的 TX RAM 中准备好数据，置位 I2C_TRANS_START，当执行到 END 命令时，I2C 主机可以更新命令寄存器和 RAM 的内容，如第二段所示，并且清零其对应的 I2C_END_DETECT_INT 中断。当第二段中 cmd2 为 STOP 时，即两段读 I2C 从机，置位 I2C_TRANS_START，I2C 主机继续传输数据，最后发送 STOP 位来停止传输。
3. 当第二段中 cmd2 为 END 时，在 I2C 主机完成第二次数据传输，并检测到 I2C 主机的 I2C_END_DETECT_INT 中断后，配置 cmd 如第三段所示。置位 I2C_TRANS_START，I2C 主机发送 STOP 位停止传输。

27.5.8.2 配置示例

1. 设置 I2C_MS_MODE（主机）为 1，I2C_MS_MODE（从机）为 0。
2. 推荐将 I2C_SLAVE_SCL_STRETCH_EN（从机）置 1，以便 I2C 从机在需要发送数据时可以先把 SCL 拉低来给软件向 I2C 从机的 TX RAM 中写数提供时间，否则 I2C 从机需要在主机开始传输前提前准备好数据。以下配置流程均按照I2C_SLAVE_SCL_STRETCH_EN（从机）为 1 的情况进行。
3. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	RSTART	—	—	—	—
I2C_COMMAND1（主机）	WRITE	0	0	1	1
I2C_COMMAND2（主机）	READ	0	0	1	N
I2C_COMMAND3（主机）	END	—	—	—	—

5. 向 I2C 主机的 TX RAM 写入从机地址，可选 FIFO 方式和直接访问方式。
6. 在I2C_SLAVE_ADDR_REG（从机）的I2C_SLAVE_ADDR（从机）设置 I2C 从机的地址。
7. 向 I2C_CONF_UPGATE（主机）和 I2C_CONF_UPGATE（从机）写 1 来同步寄存器。
8. 向 I2C_TRANS_START（主机）写 1 开始主机的传输。
9. 参考章节 27.4.14，启动从机的传输。
10. I2C 从机比较 I2C 主机发送的从机地址和自己的I2C_SLAVE_ADDR（从机），当 I2C 主机 WRITE 命令中的 ack_check_en（主机）配置为 1 时，I2C 主机会在发送完每个字节之后进行 ACK 检测。若 ack_check_en 配置为 0，则不会对 ACK 检测，会默认为匹配。
    - 匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平一致，传输继续。
    - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp（主机）电平不一致，I2C 主机产生 I2C_NACK_INT（主机）中断，停止发送数据并且产生 STOP。
11. 等待I2C_SLAVE_STRETCH_INT（从机），读取 I2C_STRETCH_CAUSE 的值为 0，此时从机地址与 SDA 线上发送的地址相匹配，且从机要发送数据。
12. 参考章节 27.4.10，向 I2C 从机的 TX RAM 写入要发送的数据。可选 FIFO 方式和直接访问方式。
13. 将I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1，释放 SCL 总线。
14. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_check_en（主机）配置的不同进行或不进行 ACK 检测。
15. 若 I2C 主机一个 READ 指令要读取的数 N 或 M 超过 I2C 从机的 TX FIFO 深度个字节，当 I2C 从机的 TX RAM 中发送数据全部发完，为空时产生I2C_SLAVE_STRETCH_INT（从机）中断。此时 I2C 从机会将 SCL 拉低，软件在此期间可以继续向 I2C RAM 填充数据，也可以从 I2C 主机的 RX RAM 读取数据。等完成操作后再将I2C_SLAVE_STRETCH_INT_CLR（从机）置 1 清除中断，将 I2C_SLAVE_SCL_STRETCH_CLR（从机）置 1 释放 SCL 总线。
16. 等到一次 READ 指令完成，I2C 主机执行 END 指令，I2C_END_DETECT_INT（主机）中断产生后，设置 I2C_END_DETECT_INT_CLR（主机）为 1 来清除中断。

17. 更新 I2C 主机的指令寄存器，有两种设置方式：

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	READ	ack_value	ack_exp	1	M
I2C_COMMAND1（主机）	END	—	—	—	—

或者

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND0（主机）	READ	0	0	1	M-1
I2C_COMMAND0（主机）	READ	1	0	1	1
I2C_COMMAND1（主机）	STOP	—	—	—	—

18. 向 I2C 从机的 TX RAM 写入 M 个字节要发送的数据，若 M 大于 TX FIFO 深度，重复步骤 12，可以用 FIFO 方式或直接访问方式。
19. 向 I2C_TRANS_START（主机）位写 1 开始传输，并重复步骤 14 的流程。
20. 若最后一个指令为 STOP，则当 I2C 主机接收最后一个数据时，将 ack_value（主机）设成 1，I2C 从机接收到 NACK 中断，停止发送。I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。
21. 若最后一个指令为为 END，则重复步骤 16，并在完成后继续下面的步骤。
22. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_en	byte_num
I2C_COMMAND1（主机）	STOP	—	—	—	—

23. 向 I2C_TRANS_START（主机）位写 1 开始传输。
24. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT（主机）中断。

27.6 中断

- I2C_RXFIFO_WM_INT：I2C RX FIFO 水印中断。当I2C_FIFO_PRT_EN 为 1，且 RX FIFO 指针大于 I2C_RXFIFO_WM_THRHD[4:0] 时，即触发该中断。
- I2C_TXFIFO_WM_INT：I2C TX FIFO 水印中断。当I2C_FIFO_PRT_EN 为 1，且 TX FIFO 指针小于 I2C_TXFIFO_WM_THRHD[4:0] 时，即触发该中断。
- I2C_RXFIFO_OVF_INT：当 I2C RX FIFO 上溢时，即触发该中断。
- I2C_END_DETECT_INT：当 I2C 主机命令的 op_code 为 END，且检测到 I2C END 状态时，即触发该中断。
- I2C_BYTE_TRANS_DONE_INT：当 I2C 发送或接收一个字节，即触发该中断。
- I2C_ARBITRATION_LOST_INT：当 I2C 主机的 SCL 为高电平，SDA 输出值与输入值不相等时，即触发该中断。
- I2C_MST_TXFIFO_UDF_INT：当 I2C 主机的 TX FIFO 下溢时，即触发该中断。
- I2C_TRANS_COMPLETE_INT：当 I2C 检测到 STOP 位时，即触发该中断。

- I2C_TIME_OUT_INT：在传输过程中，当 I2C SCL 保持为高或为低电平的时间超过 $2^{I2C\_TIME\_OUT\_VALUE}$ 个模块时钟后，即触发该中断。
- I2C_TRANS_START_INT：当 I2C 发送一个 START 位时，即触发该中断。
- I2C_NACK_INT：当 I2C 配置为主机时，接收到的 ACK 与命令中期望的 ACK 值不一致时，即触发该中断；当 I2C 配置为从机时，接收到的 ACK 值为 1 时，即触发该中断。
- I2C_TXFIFO_OVF_INT：当 I2C 通过 APB 总线写 TX FIFO，但 TX FIFO 为满时，即触发该中断。
- I2C_RXFIFO_UDF_INT：当 I2C 通过 APB 总线读取 RX FIFO，但 RX FIFO 为空时，即触发该中断。
- I2C_SCL_ST_TO_INT：当 I2C 状态机 SCL_FSM 保持某个状态超过 I2C_SCL_ST_TO_I2C[23:0] 个模块时钟周期时，即触发该中断。
- I2C_SCL_MAIN_ST_TO_INT：当 I2C 主状态机 SCL_MAIN_FSM 保持某个状态超过 I2C_SCL_MAIN_ST_TO_I2C[23:0] 个模块时钟周期时，即触发该中断。
- I2C_DET_START_INT：主机或从机检测到 I2C START 信号，即触发该中断。
- I2C_SLAVE_STRETCH_INT：从机模式下，当从机时钟拉伸时，产生此中断。
- I2C_GENERAL_CALL_INT：当从机接收到通用广播地址 (general call address) 时，即触发该中断。
- I2C_SLAVE_ADDR_UNMATCH_INT：从机模式下，接收到的从机地址和内部配置的从机地址不一致时，即触发该中断。

27.7 寄存器列表

本小节的所有地址均为相对于 I2C 控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

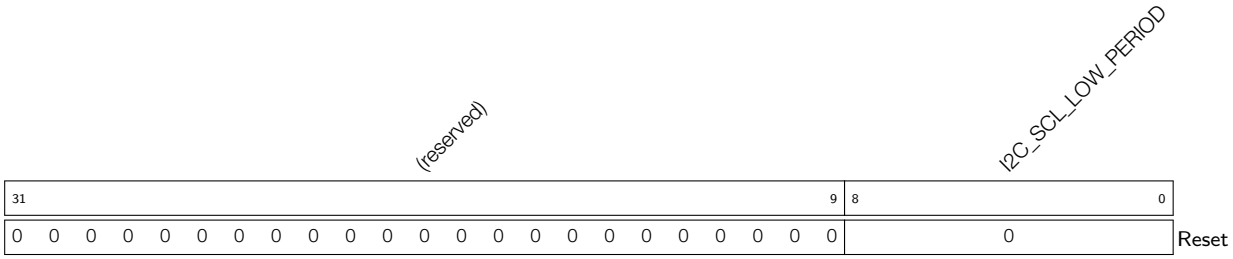
名称	描述	地址	访问
时序寄存器			
I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平的保持时间	0x0000	R/W
I2C_SDA_HOLD_REG	配置 SCL 下降沿后的数据保持时间	0x0030	R/W
I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平保持时间	0x0038	R/W
I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟时间	0x0044	R/W
I2C_SCL_STOP_HOLD_REG	配置 STOP 命令生成时 SCL 边沿的延迟	0x0048	R/W
I2C_SCL_STOP_SETUP_REG	配置 STOP 命令生成时 SDA 和 SCL 上升沿之间的间隔时间	0x004C	R/W
I2C_SCL_ST_TIME_OUT_REG	SCL 状态超时寄存器	0x0078	R/W
I2C_SCL_MAIN_ST_TIME_OUT_REG	SCL 主要状态超时寄存器	0x007C	R/W
配置寄存器			
I2C_CTR_REG	传输配置寄存器	0x0004	varies
I2C_TO_REG	数据接收超时控制寄存器	0x000C	R/W
I2C_SLAVE_ADDR_REG	从机地址配置寄存器	0x0010	R/W
I2C_FIFO_CONF_REG	FIFO 配置寄存器	0x0018	R/W
I2C_FILTER_CFG_REG	SCL 和 SDA 滤波配置寄存器	0x0050	R/W
I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
I2C_SCL_STRETCH_CONF_REG	配置 I2C 从机 SCL 时钟拉伸	0x0084	varies
状态寄存器			
I2C_SR_REG	I2C 的工作状态寄存器	0x0008	RO
I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
I2C_DATA_REG	Rx FIFO 读取数据寄存器	0x001C	HRO
Interrupt registers			
I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
I2C_INT_CLR_REG	中断清除位	0x0024	WT
I2C_INT_ENA_REG	中断使能位	0x0028	R/W
I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
I2C_COMD0_REG	I2C 命令寄存器 0	0x0058	varies
I2C_COMD1_REG	I2C 命令寄存器 1	0x005C	varies
I2C_COMD2_REG	I2C 命令寄存器 2	0x0060	varies
I2C_COMD3_REG	I2C 命令寄存器 3	0x0064	varies
I2C_COMD4_REG	I2C 命令寄存器 4	0x0068	varies

名称	描述	地址	访问
I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies
I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
版本寄存器			
I2C_DATE_REG	版本寄存器	0x00F8	R/W
Address register			
I2C_TXFIFO_START_ADDR_REG	I2C TXFIFO 基地址寄存器	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RXFIFO 基地址寄存器	0x0180	HRO

27.8 寄存器

本小节的所有地址均为相对于 I2C 控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 27.1. I2C_SCL_LOW_PERIOD_REG (0x0000)

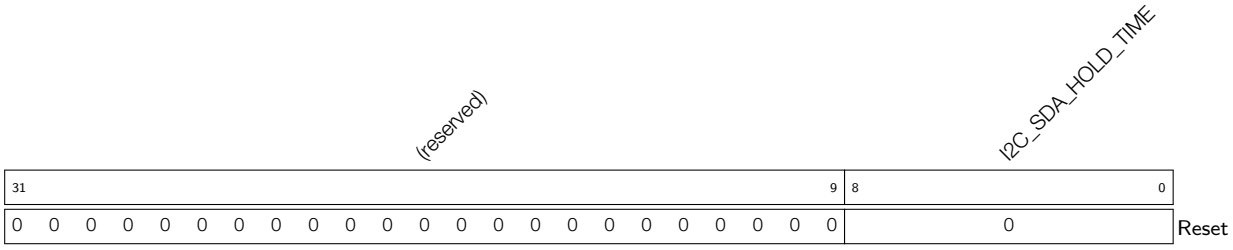


I2C_SCL_LOW_PERIOD 配置主机模式下 SCL 低电平的保持时间。

单位：I2C_SCLK

(R/W)

Register 27.2. I2C_SDA_HOLD_REG (0x0030)

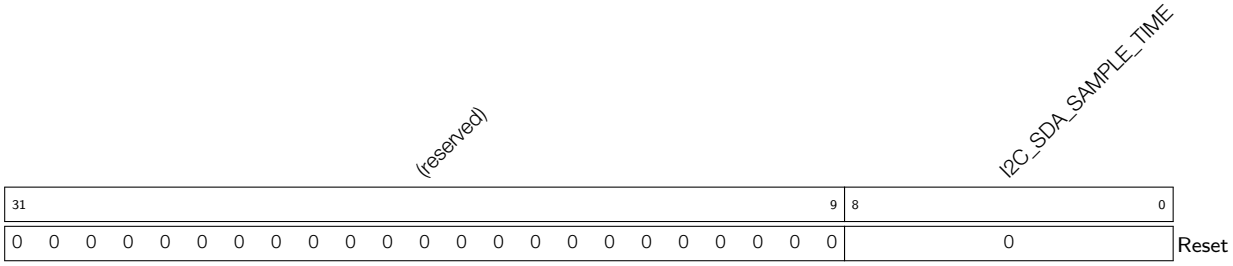


I2C_SDA_HOLD_TIME 配置 SCL 下降沿后的数据保持时间。

单位：I2C_SCLK

(R/W)

Register 27.3. I2C_SDA_SAMPLE_REG (0x0034)

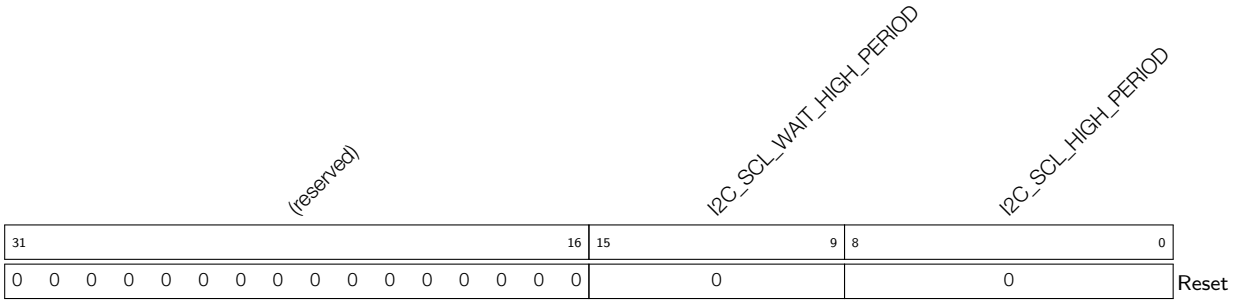


I2C_SDA_SAMPLE_TIME 配置采样 SDA 的时间。

单位：I2C_SCLK

(R/W)

Register 27.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD 配置 SCL 在主机模式下保持高电平的时间。

单位：I2C_SCLK

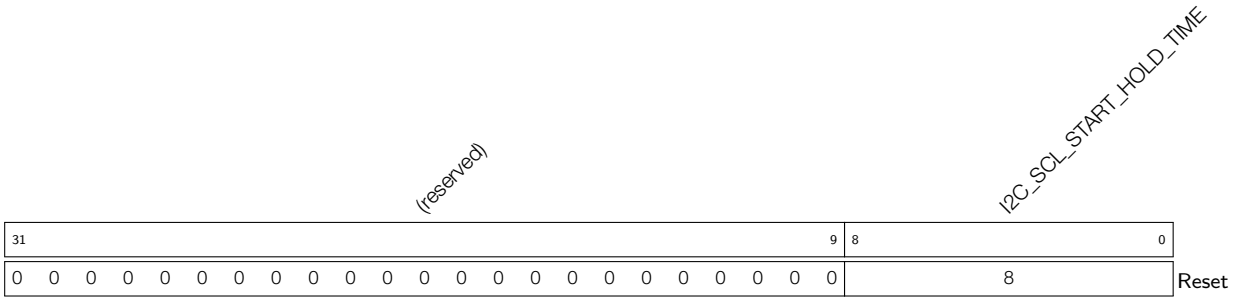
(R/W)

I2C_SCL_WAIT_HIGH_PERIOD 配置 SCL_FSM 等待 SCL 在主机模式下翻转至高电平的时间。

单位：I2C_SCLK

(R/W)

Register 27.5. I2C_SCL_START_HOLD_REG (0x0040)

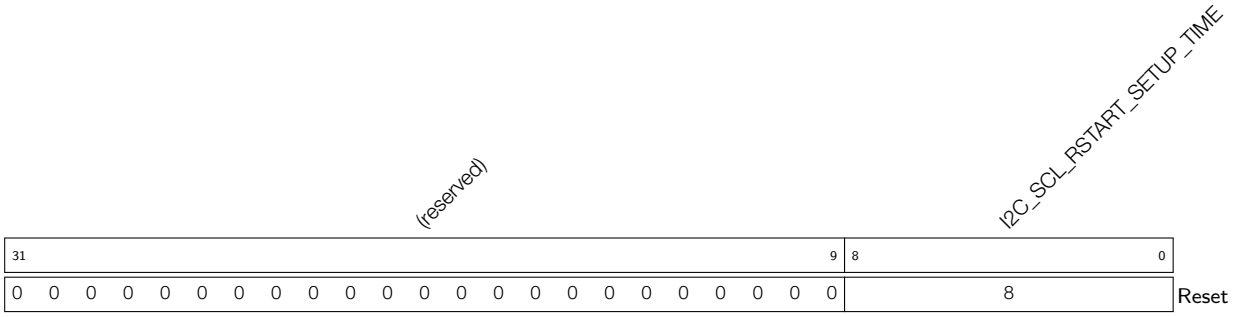


I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间。

单位：I2C_SCLK

(R/W)

Register 27.6. I2C_SCL_RSTART_SETUP_REG (0x0044)

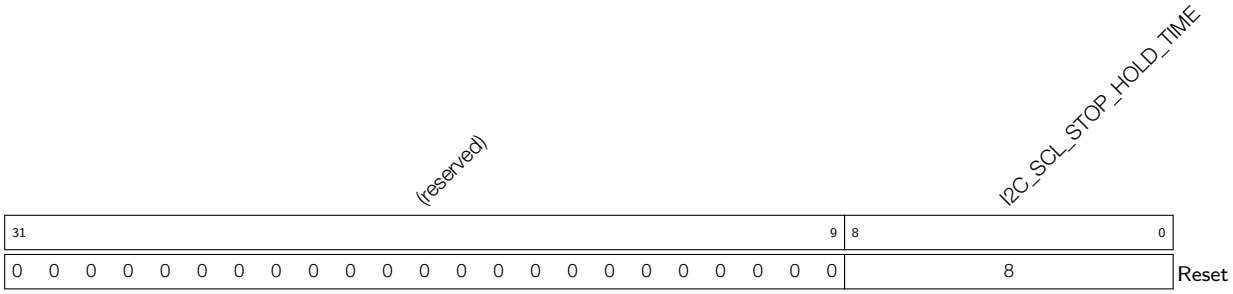


I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间。

单位：I2C_SCLK

(R/W)

Register 27.7. I2C_SCL_STOP_HOLD_REG (0x0048)

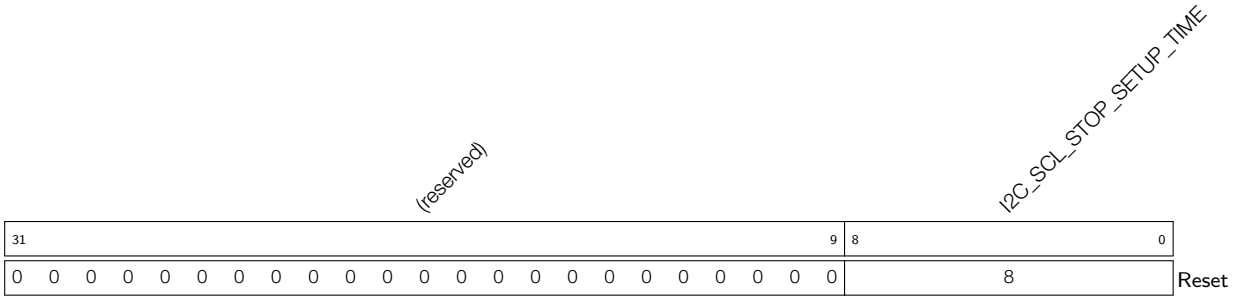


I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟时间。

单位：I2C_SCLK

(R/W)

Register 27.8. I2C_SCL_STOP_SETUP_REG (0x004C)

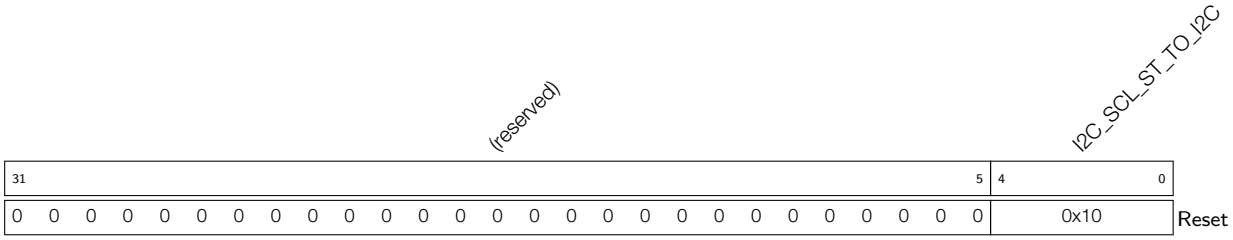


I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间。

单位：I2C_SCLK

(R/W)

Register 27.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)

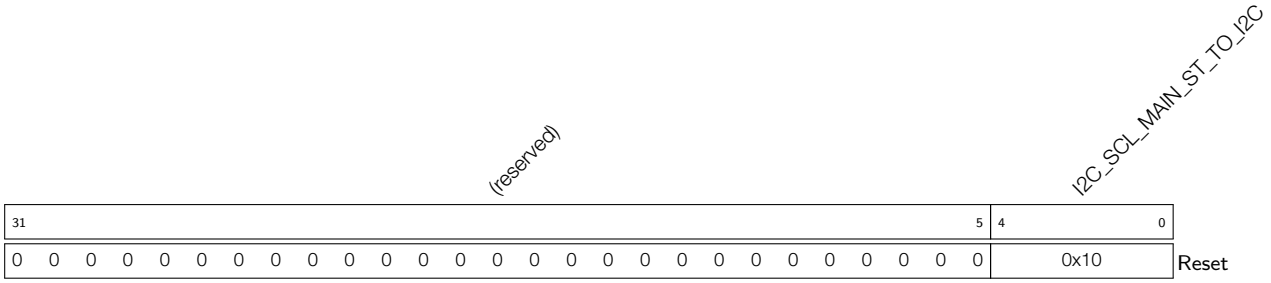


I2C_SCL_ST_TO_I2C 配置 SCL_FSM 状态保持不变的最大时间，不能大于 23。

单位：I2C_SCLK

(R/W)

Register 27.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)

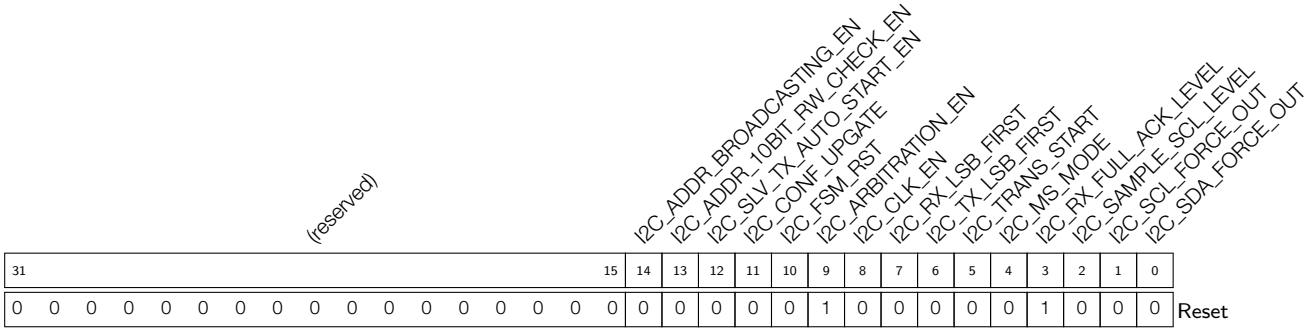


I2C_SCL_MAIN_ST_TO_I2C 配置 SCL_MAIN_FSM 状态保持不变的最大时间，不能大于 23。

单位：I2C_SCLK

(R/W)

Register 27.11. I2C_CTR_REG (0x0004)



I2C_SDA_FORCE_OUT 配置 SDA 输出模式。

0: 开漏输出

1: 直接输出

(R/W)

I2C_SCL_FORCE_OUT 配置 SCL 输出模式。

0：开漏输出

1：直接输出

(R/W)

I2C_SAMPLE_SCL_LEVEL 配置 SDA 的采样模式。

0: SCL 为高电平时采样 SDA 数据

1: SCL 为低电平时采样 SDA 数据

(R/W)

I2C_RX_FULL_ACK_LEVEL 配置主机在 rx_fifo_cnt 达到阈值时需发送的 ACK 电平值。(R/W)

I2C_MS_MODE 配置该模块为 I2C 主机或从机。

0：从机

1：主机

(R/W)

I2C_TRANS_START 配置从机是否开始发送 TX FIFO 中的数据。

0：无效

1：开始

(WT)

I2C_TX_LSB_FIRST 配置待发送数据的发送顺序。

0：从最高有效位开始发送数据

1：从最低有效位开始发送数据

(R/W)

I2C_RX_LSB_FIRST 配置接收数据的存储顺序。

0：从最高有效位开始接收数据

1：从最低有效位开始接收数据

(R/W)

Continued on the next page...

Register 27.11. I2C_CTR_REG (0x0004)

Continued from the previous page...

I2C_CLK_EN 门控寄存器的时钟信号。

0：仅当通过软件读写寄存器时支持时钟

1：强制寄存器时钟开启

(R/W)

I2C_ARBITRATION_EN 配置使能 I2C 总线仲裁检测。

0：无效

1：使能

(R/W)

I2C_FSM_RST 配置复位 SCL_FSM。

0：无效

1：复位

(WT)

I2C_CONF_UPGATE 配置此位实现同步。

0：无效

1：同步

(WT)

I2C_SLV_TX_AUTO_START_EN 配置从机自动发送数据的使能位。

0：禁用

1：使能

(R/W)

I2C_ADDR_10BIT_RW_CHECK_EN 配置使能 10 位寻址模式的读写标志位检查功能，检查读写标志是否符合协议。

0：不检查

1：检查

(R/W)

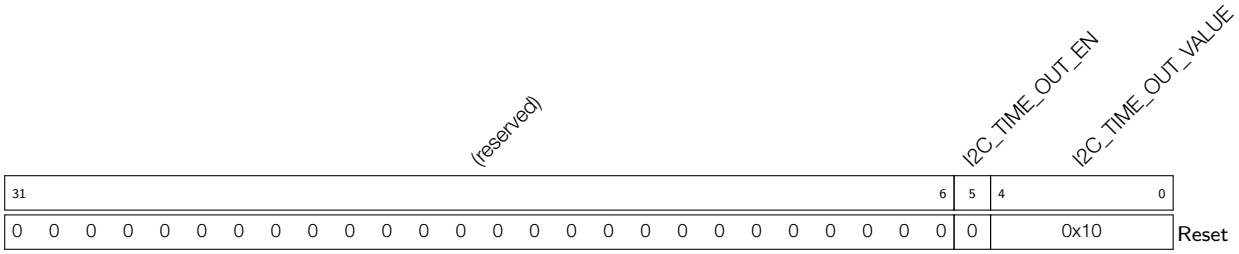
I2C_ADDR_BROADCASTING_EN 配置是否支持 7 位寻址模式的广播功能。

0：不支持

1：支持

(R/W)

Register 27.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE 配置 SCL 在高电平或低电平停滞的超时阈值周期。实际周期为 $2^{I2C\_TIME\_OUT\_VALUE}$。

单位：I2C_SCLK

(R/W)

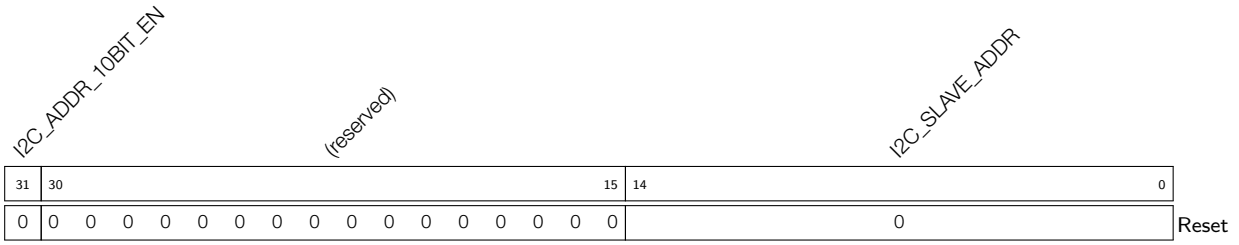
I2C_TIME_OUT_EN 配置使能超时控制。

0：无效

1：使能

(R/W)

Register 27.13. I2C_SLAVE_ADDR_REG (0x0010)



I2C_SLAVE_ADDR 配置 I2C 从机的从机地址。(R/W)

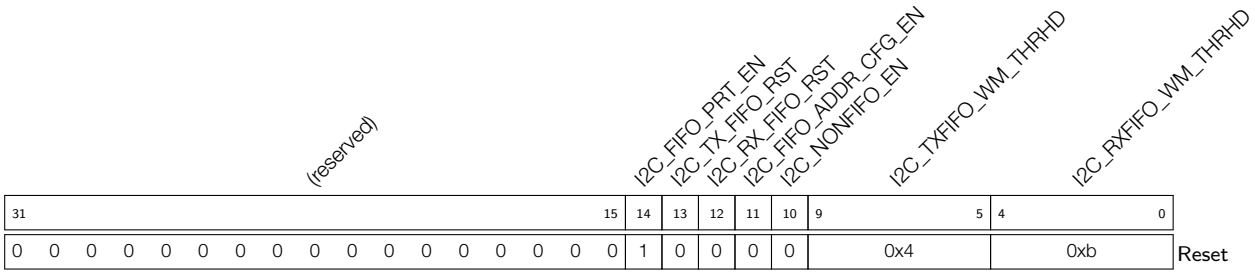
I2C_ADDR_10BIT_EN 配置在主机模式下使能从机的 10 位寻址模式。

0：无效

1：使能

(R/W)

Register 27.14. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD 配置在直接访问模式下，RX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 RX FIFO 计数值大于 I2C_RXFIFO_WM_THRHD[4:0] 时，I2C_RXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_TXFIFO_WM_THRHD 配置在直接访问模式下，TX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 TX FIFO 计数值大于 I2C_TXFIFO_WM_THRHD[4:0] 时，I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_NONFIFO_EN 配置使能 APB 直接访问。(R/W)

I2C_FIFO_ADDR_CFG_EN 配置从机是否使能双寻址模式。当该模式使能时，在 I2C 地址字节之后接收的字节代表 I2C 从机 RAM 中的偏移地址。

0：禁用

1：使能

(R/W)

I2C_RX_FIFO_RST 配置复位 RX FIFO。

0：无效

1：复位

(R/W)

I2C_TX_FIFO_RST 配置复位 TX FIFO。

0：无效

1：复位

(R/W)

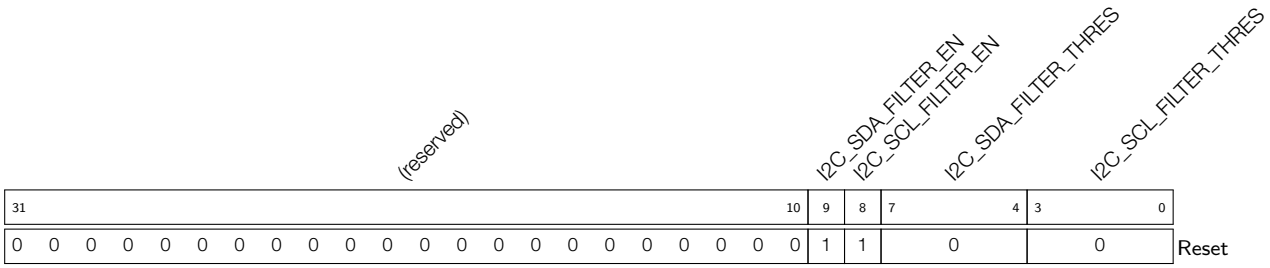
I2C_FIFO_PRT_EN 配置在直接访问模式下是否使能 FIFO 指针。该位控制有效位以及 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的中断。

0：无效

1：使能

(R/W)

Register 27.15. I2C_FILTER_CFG_REG (0x0050)



I2C_SCL_FILTER_THRES 配置 SCL 过滤脉冲宽度的阈值。当 SCL 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK

(R/W)

I2C_SDA_FILTER_THRES 配置 SDA 过滤脉冲宽度的阈值。当 SDA 输入信号的脉冲宽度小于该字段的值时，I2C 控制器忽略此脉冲。

单位：I2C_SCLK

(R/W)

I2C_SCL_FILTER_EN 配置使能 SCL 的滤波功能。

0：无效

1：复位

(R/W)

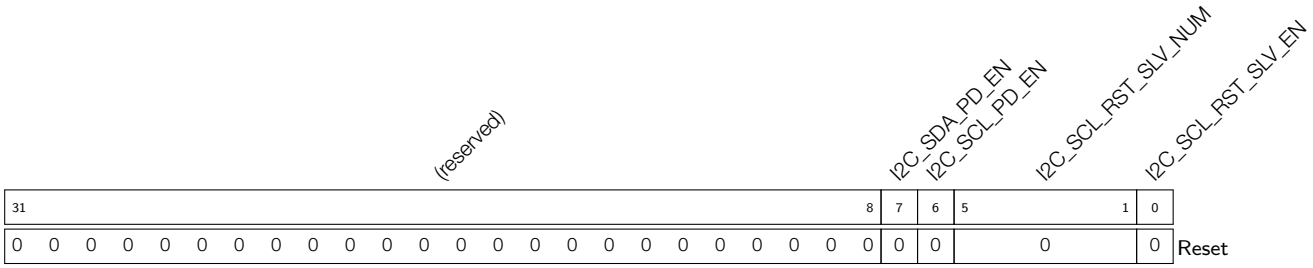
I2C_SDA_FILTER_EN 配置使能 SDA 的滤波功能。

0：无效

1：复位

(R/W)

Register 27.16. I2C_SCL_SP_CONF_REG (0x0080)



I2C_SCL_RST_SLV_EN 在 I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 I2C_SCL_RST_SLV_NUM[4:0]。(R/W/SC)

I2C_SCL_RST_SLV_NUM 配置 I2C 主机模式下生成的 SCL 脉冲。

仅在 I2C_SCL_RST_SLV_EN 为 1 时有效。

单位：I2C_SCLK

(R/W)

I2C_SCL_PD_EN 配置降低 I2C SCL 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 I2C_SCL_FORCE_OUT 为 1 时有效。

(R/W)

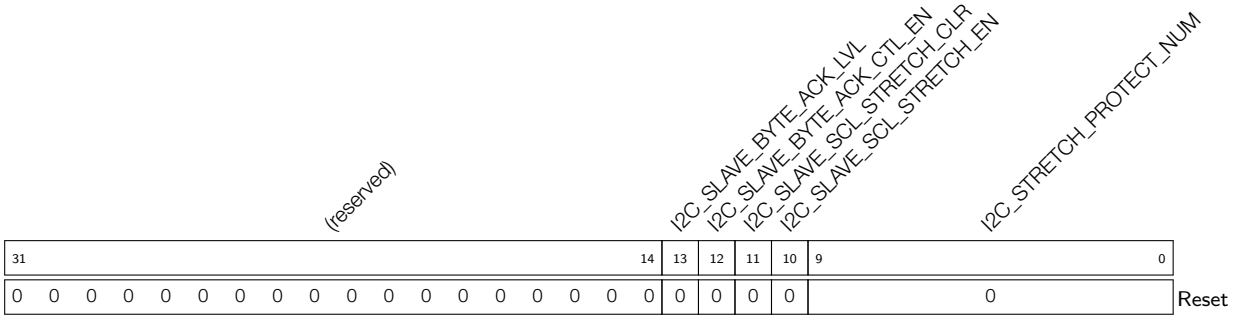
I2C_SDA_PD_EN 配置降低 I2C SDA 输出功耗。

0：正常功耗

1：不工作，降低功耗

仅在 I2C_SDA_FORCE_OUT 为 1 时有效。(R/W)

Register 27.17. I2C_SCL_STRETCH_CONF_REG (0x0084)



I2C_STRETCH_PROTECT_NUM 配置 SCL 时钟拉伸释放后的保护时间，通常设置为大于 SDA 的建立时间。

单位：I2C_SCLK

(R/W)

I2C_SLAVE_SCL_STRETCH_EN 配置使能从机的 SCL 时钟拉伸。I2C_SLAVE_SCL_STRETCH_EN 为 1，且出现可触发时钟拉伸的事件时，拉伸 SCL 时钟。SCL 时钟拉伸的原因可见 I2C_STRETCH_CAUSE。

0：禁用

1：使能

(R/W)

I2C_SLAVE_SCL_STRETCH_CLR 配置清除从机 SCL 时钟拉伸。

0：无效

1：清除

(WT)

I2C_SLAVE_BYTE_ACK_CTL_EN 配置使能从机控制 ACK 电平。

0：禁用

1：使能

(R/W)

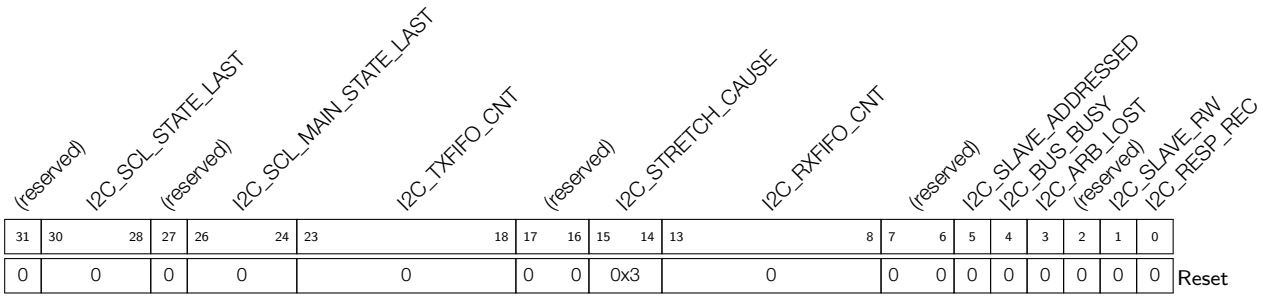
I2C_SLAVE_BYTE_ACK_LVL 从机控制 ACK 电平使能时，通过此位设置 ACK 的电平值。

0：低电平

1：高电平

(R/W)

Register 27.18. I2C_SR_REG (0x0008)



I2C_RESP_REC 表示主机模式或从机模式下接收的 ACK 电平值。

0: ACK

1: NACK

(RO)

I2C_SLAVE_RW 表示在从机模式下数据传输方向。

0：主机向从机写入数据

1：主机读取从机数据

(RO)

I2C_ARB_LOST 表示 I2C 控制器是否控制 SCL 线。

0: 仲裁未丢失

1: 仲裁丢失

(RO)

I2C_BUS_BUSY 表示 I2C 总线状态。

0：I2C 总线处于空闲状态

1：I2C 总线正在传输数据

(RO)

I2C_SLAVE_ADDRESSED 表示 master 发送的地址是否与 slave 的地址相同。

仅当模块被配置为 I2C 从机时有效。

0: 不相同

1: 相同

(RO)

I2C_RXFIFO_CNT 表示 RAM 接收数据的字节数。(RO)

I2C_STRETCH_CAUSE 表示从机模式下 SCL 时钟拉伸的原因。

0：主机开始读取数据时拉伸 SCL 时钟

1：从机模式下 I2C TX FIFO 读空时拉伸 SCL 时钟

2：从机模式下 I2C RX FIFO 写满时拉伸 SCL 时钟

(RO)

I2C_TXFIFO_CNT 表示需发送数据的字节数。(RO)

Continued on the next page...

Register 27.18. I2C_SR_REG (0x0008)

Continued from the previous page...

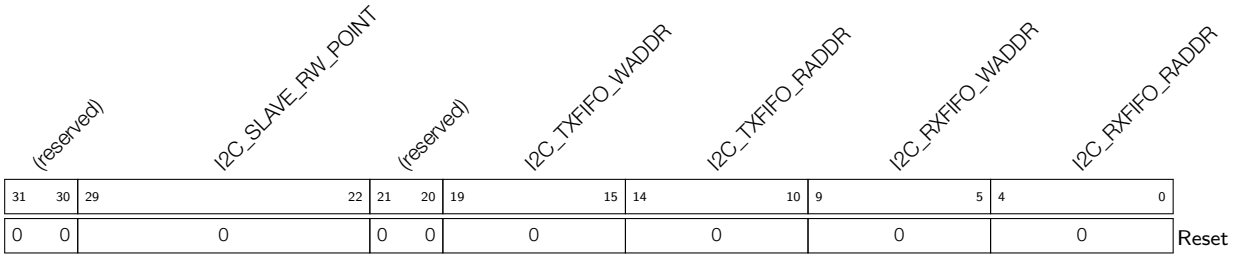
I2C_SCL_MAIN_STATE_LAST 表示 I2C 控制器状态机的状态。

0：空闲
1：地址偏移
2：ACK 地址
3：接收数据
4：发送数据
5：发送 ACK
6：等待 ACK
(RO)

I2C_SCL_STATE_LAST 表示用于生成 SCL 的状态机状态。

0：空闲
1：开始
2：下降沿
3：低电平
4：上升沿
5：高电平
6：停止
(RO)

Register 27.19. I2C_FIFO_ST_REG (0x0014)



I2C_RXFIFO_RADDR 表示 APB 总线读 RX FIFO 的偏移地址。(RO)

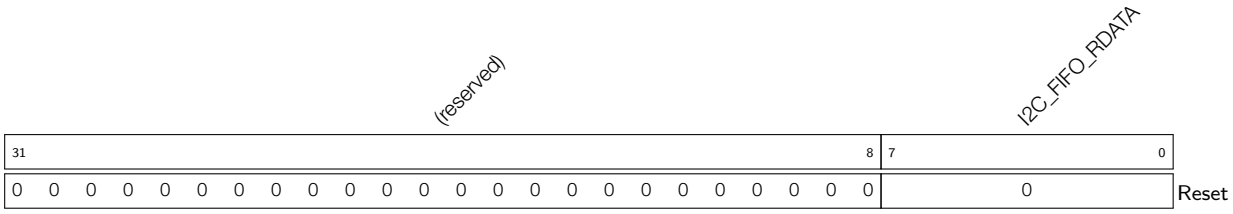
I2C_RXFIFO_WADDR 表示 I2C 控制器接收数据和写 RX FIFO 的偏移地址。(RO)

I2C_TXFIFO_RADDR 表示 I2C 控制器读 TX FIFO 的偏移地址。(RO)

I2C_TXFIFO_WADDR 表示 APB 总线写 TX FIFO 的偏移地址。(RO)

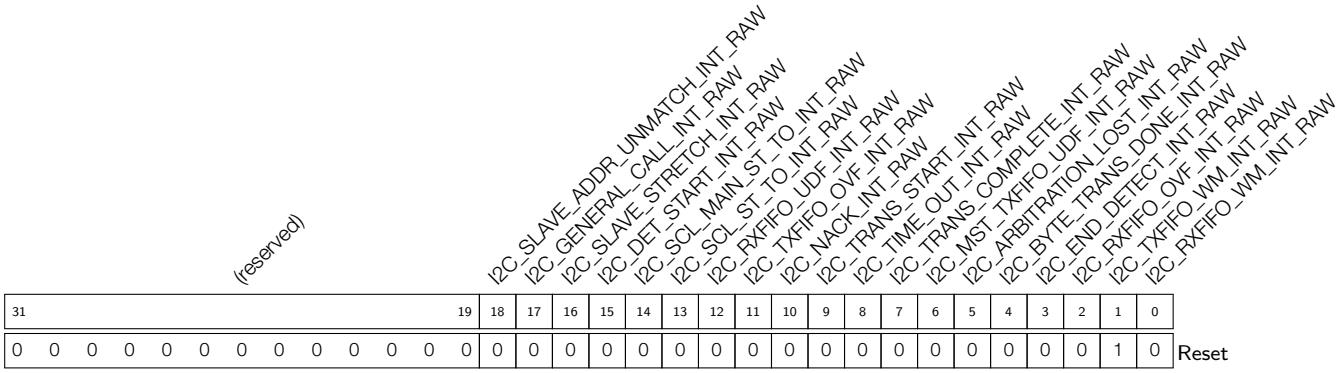
I2C_SLAVE_RW_POINT 表示在 I2C 从机模式下由 I2C 主机寻址的 I2C 从机 RAM 的偏移地址。(RO)

Register 27.20. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA 表示 RX FIFO 读取数据的值。(RO)

Register 27.21. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW I2C_RXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW I2C_TXFIFO_WM_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW I2C_RXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_END_DETECT_INT_RAW I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW I2C_END_DETECT_INT 的原始中断状态。(R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW I2C_ARBITRATION_LOST_INT 的原始中断状态。(R/SS/WTC)

Continued on the next page...

Register 27.21. I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

I2C_MST_TXFIFO_UDF_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断状态。(R/SS/WTC)

I2C_TIME_OUT_INT_RAW I2C_TIME_OUT_INT 的原始中断状态。(R/SS/WTC)

I2C_TRANS_START_INT_RAW I2C_TRANS_START_INT 的原始中断状态。(R/SS/WTC)

I2C_NACK_INT_RAW I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW I2C_TXFIFO_OVF_INT 的原始中断状态。(R/SS/WTC)

I2C_RXFIFO_UDF_INT_RAW I2C_RXFIFO_UDF_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW I2C_SCL_ST_TO_INT 的原始中断状态。(R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_MAIN_ST_TO_INT 的原始中断状态。(R/SS/WTC)

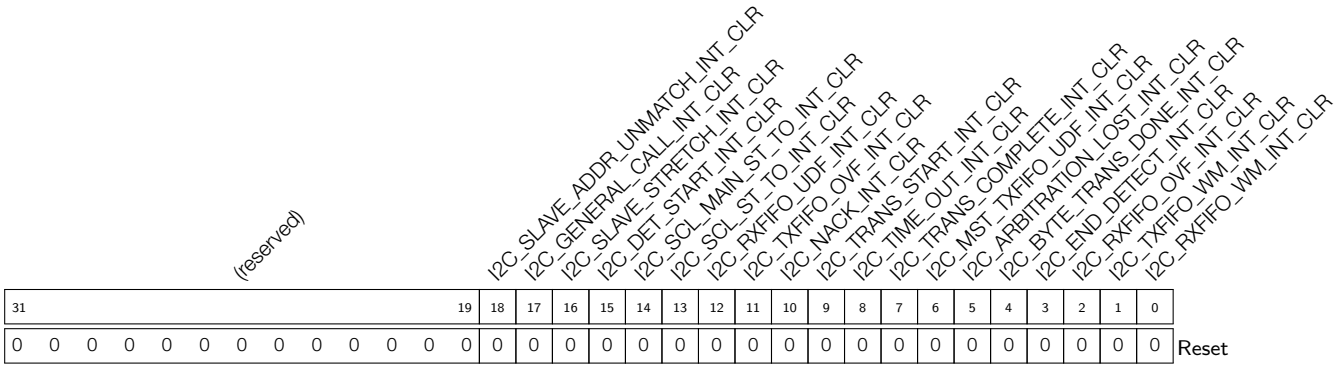
I2C_DET_START_INT_RAW I2C_DET_START_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_STRETCH_INT_RAW I2C_SLAVE_STRETCH_INT 的原始中断状态。(R/SS/WTC)

I2C_GENERAL_CALL_INT_RAW I2C_GENARAL_CALL_INT 的原始中断状态。(R/SS/WTC)

I2C_SLAVE_ADDR_UNMATCH_INT_RAW I2C_SLAVE_ADDR_UNMATCH_INT 的原始中断状态。(R/SS/WTC)

Register 27.22. I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR 写 1 清除 I2C_RXFIFO_WM_INT 中断。(WT)

I2C_TXFIFO_WM_INT_CLR 写 1 清除 I2C_TXFIFO_WM_INT 中断。(WT)

I2C_RXFIFO_OVF_INT_CLR 写 1 清除 I2C_RXFIFO_OVF_INT 中断。(WT)

I2C_END_DETECT_INT_CLR 写 1 清除 I2C_END_DETECT_INT 中断。(WT)

I2C_BYTE_TRANS_DONE_INT_CLR 写 1 清除 I2C_END_DETECT_INT 中断。(WT)

I2C_ARBITRATION_LOST_INT_CLR 写 1 清除 I2C_ARBITRATION_LOST_INT 中断。(WT)

I2C_MST_TXFIFO_UDF_INT_CLR 写 1 清除 I2C_TRANS_COMPLETE_INT 中断。(WT)

I2C_TRANS_COMPLETE_INT_CLR 写 1 清除 I2C_TRANS_COMPLETE_INT 中断。(WT)

I2C_TIME_OUT_INT_CLR 写 1 清除 I2C_TIME_OUT_INT 中断。(WT)

I2C_TRANS_START_INT_CLR 写 1 清除 I2C_TRANS_START_INT 中断。(WT)

I2C_NACK_INT_CLR 写 1 清除 I2C_SLAVE_STRETCH_INT 中断。(WT)

I2C_TXFIFO_OVF_INT_CLR 写 1 清除 I2C_TXFIFO_OVF_INT 中断。(WT)

I2C_RXFIFO_UDF_INT_CLR 写 1 清除 I2C_RXFIFO_UDF_INT 中断。(WT)

I2C_SCL_ST_TO_INT_CLR 写 1 清除 I2C_SCL_ST_TO_INT 中断。(WT)

I2C_SCL_MAIN_ST_TO_INT_CLR 写 1 清除 I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

I2C_DET_START_INT_CLR 写 1 清除 I2C_DET_START_INT 中断。(WT)

I2C_SLAVE_STRETCH_INT_CLR 写 1 清除 I2C_SLAVE_STRETCH_INT 中断。(WT)

I2C_GENERAL_CALL_INT_CLR 写 1 清除 I2C_GENARAL_CALL_INT 中断。(WT)

I2C_SLAVE_ADDR_UNMATCH_INT_CLR 写 1 清除 I2C_SLAVE_ADDR_UNMATCH_INT_RAW 中断。(WT)

Register 27.23. I2C_INT_ENA_REG (0x0028)

Bits	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_ENA	0
17	I2C_GENERAL_CALL_INT_ENA	0
16	I2C_SLAVE_STRETCH_INT_ENA	0
15	I2C_DET_START_INT_ENA	0
14	I2C_SCL_MAIN_ST_TO_INT_ENA	0
13	I2C_SCL_ST_TO_INT_ENA	0
12	I2C_RXFIFO_UDF_INT_ENA	0
11	I2C_TXFIFO_OVF_INT_ENA	0
10	I2C_NACK_INT_ENA	0
9	I2C_TRANS_START_INT_ENA	0
8	I2C_TIME_OUT_INT_ENA	0
7	I2C_TRANS_COMPLETE_INT_ENA	0
6	I2C_MST_TXFIFO_UDF_INT_ENA	0
5	I2C_ARBITRATION_LOST_INT_ENA	0
4	I2C_BYTE_TRANS_DONE_INT_ENA	0
3	I2C_END_DETECT_INT_ENA	0
2	I2C_RXFIFO_OVF_INT_ENA	0
1	I2C_TXFIFO_WM_INT_ENA	0
0	I2C_RXFIFO_WM_INT_ENA	0

I2C_RXFIFO_WM_INT_ENA 写 1 使能 I2C_RXFIFO_WM_INT 中断。(R/W)

I2C_TXFIFO_WM_INT_ENA 写 1 使能 I2C_TXFIFO_WM_INT 中断。(R/W)

I2C_RXFIFO_OVF_INT_ENA 写 1 使能 I2C_RXFIFO_OVF_INT 中断。(R/W)

I2C_END_DETECT_INT_ENA 写 1 使能 I2C_END_DETECT_INT 中断。(R/W)

I2C_BYTE_TRANS_DONE_INT_ENA 写 1 使能 I2C_END_DETECT_INT 中断。(R/W)

I2C_ARBITRATION_LOST_INT_ENA 写 1 使能 I2C_ARBITRATION_LOST_INT 中断。(R/W)

I2C_MST_TXFIFO_UDF_INT_ENA 写 1 使能 I2C_TRANS_COMPLETE_INT 中断。(R/W)

I2C_TRANS_COMPLETE_INT_ENA 写 1 使能 I2C_TRANS_COMPLETE_INT 中断。(R/W)

I2C_TIME_OUT_INT_ENA 写 1 使能 I2C_TIME_OUT_INT 中断。(R/W)

I2C_TRANS_START_INT_ENA 写 1 使能 I2C_TRANS_START_INT 中断。(R/W)

I2C_NACK_INT_ENA 写 1 使能 I2C_SLAVE_STRETCH_INT 中断。(R/W)

I2C_TXFIFO_OVF_INT_ENA 写 1 使能 I2C_TXFIFO_OVF_INT 中断。(R/W)

I2C_RXFIFO_UDF_INT_ENA 写 1 使能 I2C_RXFIFO_UDF_INT 中断。(R/W)

I2C_SCL_ST_TO_INT_ENA 写 1 使能 I2C_SCL_ST_TO_INT 中断。(R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA 写 1 使能 I2C_SCL_MAIN_ST_TO_INT 中断。(R/W)

I2C_DET_START_INT_ENA 写 1 使能 I2C_DET_START_INT 中断。(R/W)

I2C_SLAVE_STRETCH_INT_ENA 写 1 使能 I2C_SLAVE_STRETCH_INT 中断。(R/W)

I2C_GENERAL_CALL_INT_ENA 写 1 使能 I2C_GENARAL_CALL_INT 中断。(R/W)

I2C_SLAVE_ADDR_UNMATCH_INT_ENA 写 1 使能 I2C_SLAVE_ADDR_UNMATCH_INT 中断。(R/W)

Register 27.24. I2C_INT_STATUS_REG (0x002C)

Bits	Field	Reset
31:19	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0
18	I2C_SLAVE_ADDR_UNMATCH_INT_ST	0
17	I2C_GENERAL_CALL_INT_ST	0
16	I2C_SLAVE_STRETCH_INT_ST	0
15	I2C_DET_START_INT_ST	0
14	I2C_SCL_MAIN_ST_TO_INT_ST	0
13	I2C_SCL_ST_TO_INT_ST	0
12	I2C_RXFIFO_UDF_INT_ST	0
11	I2C_TXFIFO_OVF_INT_ST	0
10	I2C_NACK_INT_ST	0
9	I2C_TRANS_START_INT_ST	0
8	I2C_TIME_OUT_INT_ST	0
7	I2C_TRANS_COMPLETE_INT_ST	0
6	I2C_MST_TXFIFO_UDF_INT_ST	0
5	I2C_ARBITRATION_LOST_INT_ST	0
4	I2C_BYTE_TRANS_DONE_INT_ST	0
3	I2C_END_DETECT_INT_ST	0
2	I2C_RXFIFO_OVF_INT_ST	0
1	I2C_TXFIFO_WM_INT_ST	0
0	I2C_RXFIFO_WM_INT_ST	0

I2C_RXFIFO_WM_INT_ST I2C_RXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_WM_INT_ST I2C_TXFIFO_WM_INT 的屏蔽中断状态。(RO)

I2C_RXFIFO_OVF_INT_ST I2C_RXFIFO_OVF_INT 的屏蔽中断状态。(RO)

I2C_END_DETECT_INT_ST I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

I2C_BYTE_TRANS_DONE_INT_ST I2C_END_DETECT_INT 的屏蔽中断状态。(RO)

I2C_ARBITRATION_LOST_INT_ST I2C_ARBITRATION_LOST_INT 的屏蔽中断状态。(RO)

I2C_MST_TXFIFO_UDF_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

I2C_TRANS_COMPLETE_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽中断状态。(RO)

I2C_TIME_OUT_INT_ST I2C_TIME_OUT_INT 的屏蔽中断状态。(RO)

I2C_TRANS_START_INT_ST I2C_TRANS_START_INT 的屏蔽中断状态。(RO)

I2C_NACK_INT_ST I2C_SLAVE_STRETCH_INT 的屏蔽中断状态。(RO)

I2C_TXFIFO_OVF_INT_ST I2C_TXFIFO_OVF_INT 的屏蔽中断状态。(RO)

Continued on the next page...

Register 27.24. I2C_INT_RAW_REG (0x0020)

Continued from the previous page...

I2C_RXFIFO_UDF_INT_ST I2C_RXFIFO_UDF_INT 的屏蔽中断状态。(RO)

I2C_SCL_ST_TO_INT_ST I2C_SCL_ST_TO_INT 的屏蔽中断状态。(RO)

I2C_SCL_MAIN_ST_TO_INT_ST I2C_SCL_MAIN_ST_TO_INT 的屏蔽中断状态。(RO)

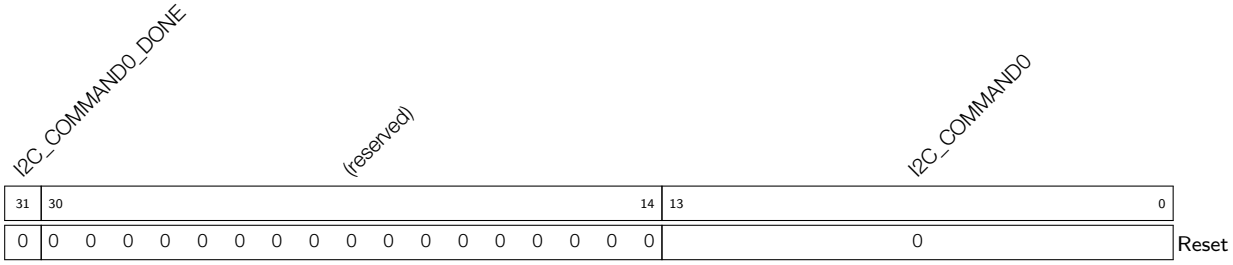
I2C_DET_START_INT_ST I2C_DET_START_INT 的屏蔽中断状态。(RO)

I2C_SLAVE_STRETCH_INT_ST I2C_SLAVE_STRETCH_INT 的屏蔽中断状态。(RO)

I2C_GENERAL_CALL_INT_ST I2C_GENARAL_CALL_INT 的屏蔽中断状态。(RO)

I2C_SLAVE_ADDR_UNMATCH_INT_ST I2C_SLAVE_ADDR_UNMATCH_INT 的屏蔽中断状态。(RO)

Register 27.25. I2C_COMD0_REG (0x0058)



I2C_COMMAND0 配置命令寄存器 0 的内容。

命令寄存器包含三个部分：

op_code 表示命令：

0: RSTART

1: WRITE

2: READ

3: STOP

4: END

Byte_num 表示需要发送或接收数据的字节数。

ack_check_en, ack_exp 和 ack 用于控制 ACK 位。详细信息，请参阅 I2C CMD 结构 27-5 。

(R/W)

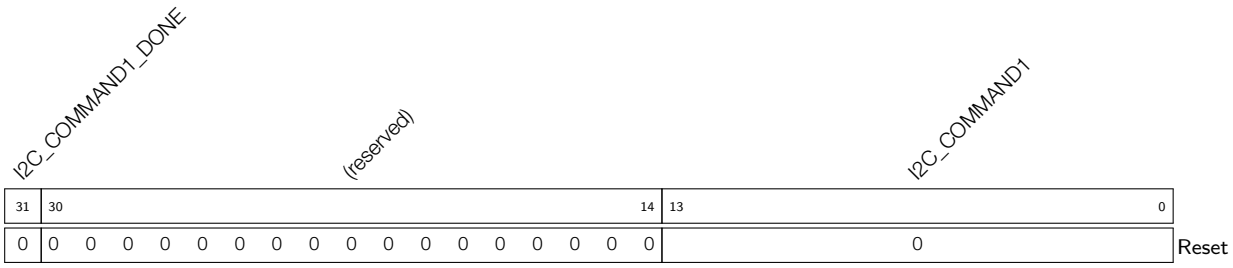
I2C_COMMAND0_DONE 表示在 I2C 主机模式下，命令 0 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 27.26. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 配置命令寄存器 1 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

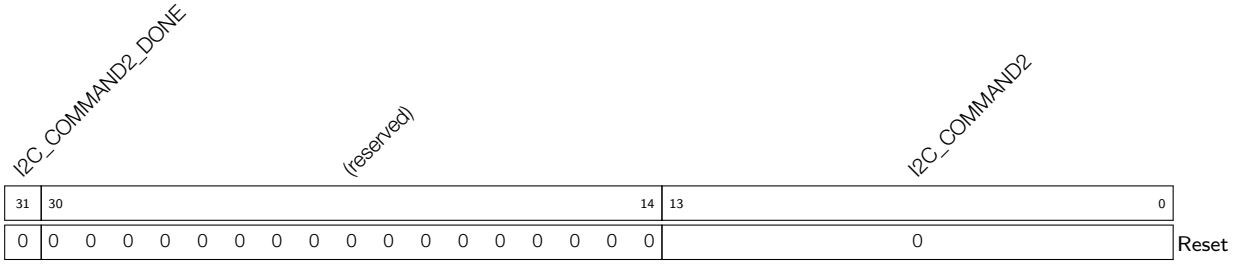
I2C_COMMAND1_DONE 表示在 I2C 主机模式下，命令 1 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 27.27. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 配置命令寄存器 2 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

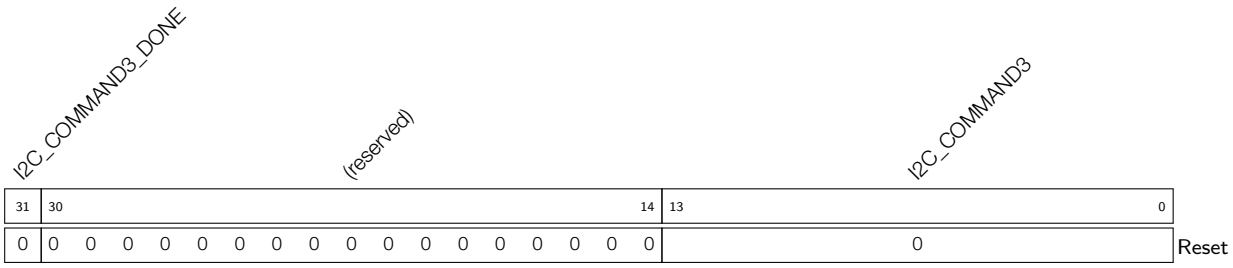
I2C_COMMAND2_DONE 表示在 I2C 主机模式下，命令 2 是否已完成。.

0: 未完成

1: 已完成

(R/W/SS)

Register 27.28. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 配置命令寄存器 3 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

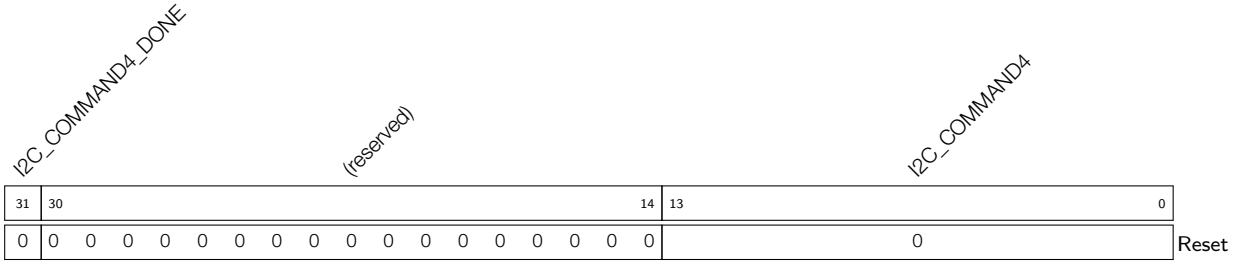
I2C_COMMAND3_DONE 表示在 I2C 主机模式下，命令 3 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 27.29. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 配置命令寄存器 4 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

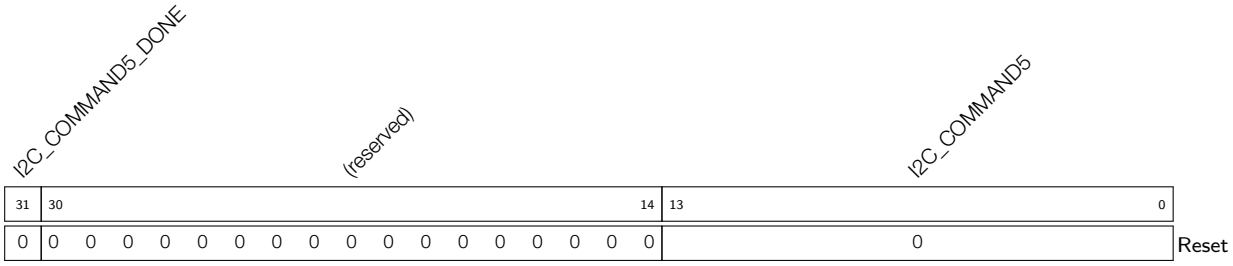
I2C_COMMAND4_DONE 表示在 I2C 主机模式下，命令 4 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 27.30. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 配置命令寄存器 5 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

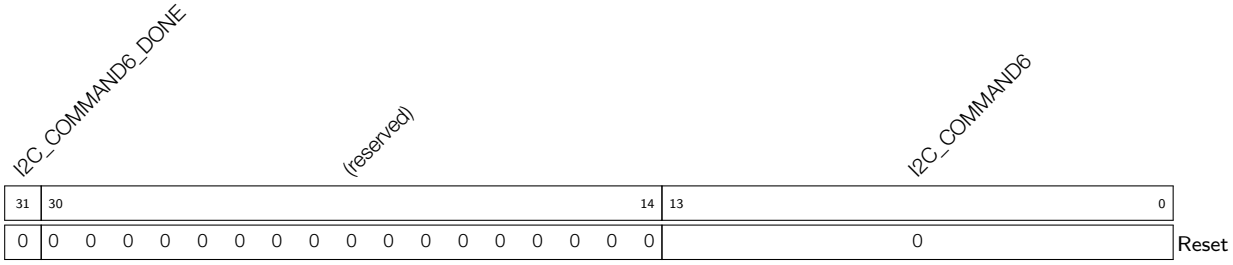
I2C_COMMAND5_DONE 表示在 I2C 主机模式下，命令 5 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 27.31. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 配置命令寄存器 6 的内容。具体可参考 I2C_CMD0_REG[13:0]。(R/W)

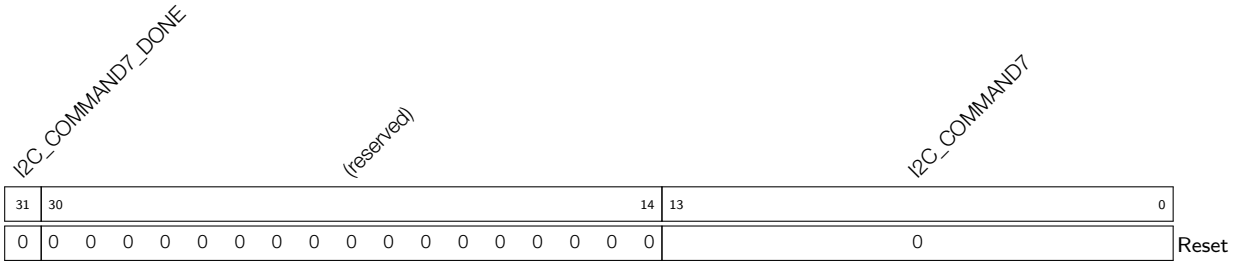
I2C_COMMAND6_DONE 表示在 I2C 主机模式下，命令 6 是否已完成。

0: 未完成

1: 已完成

(R/W/SS)

Register 27.32. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 配置命令寄存器 7 的内容。具体可参考 I2C_CMD0_REG[13:0]。 I2C_CMD0_REG[13:0]. (R/W)

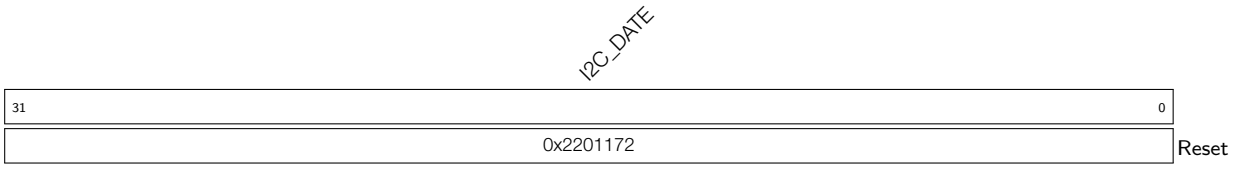
I2C_COMMAND7_DONE 表示在 I2C 主机模式下，命令 7 是否已完成。

0: 未完成

1: 已完成

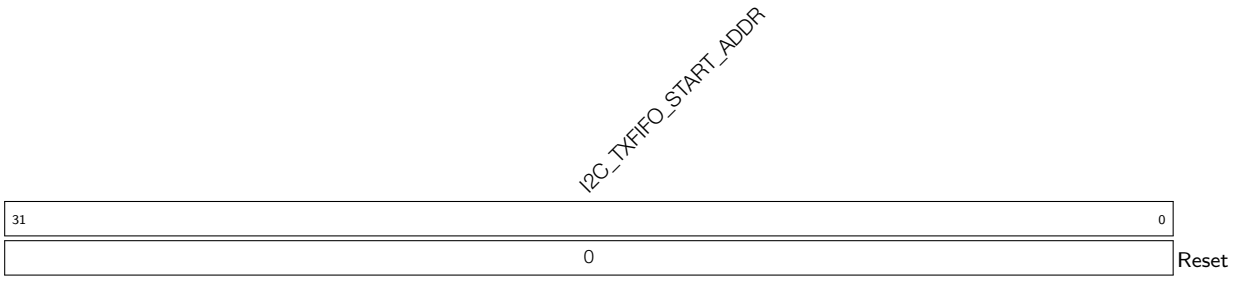
(R/W/SS)

Register 27.33. I2C_DATE_REG (0x00F8)



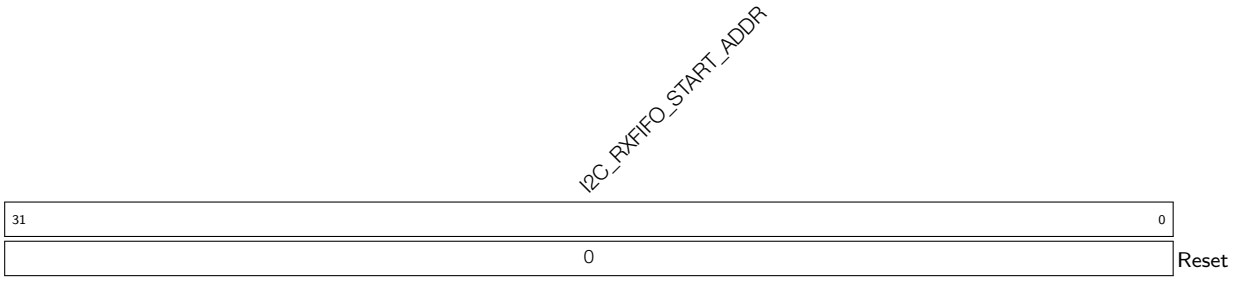
I2C_DATE 版本控制寄存器。(R/W)

Register 27.34. I2C_TXFIFO_START_ADDR_REG (0x0100)



I2C_TXFIFO_START_ADDR 表示 I2C TX FIFO 首地址。(RO)

Register 27.35. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR 表示 I2C RX FIFO 首地址。(RO)

28 I2S 控制器 (I2S)

28.1 概述

ESP32-H2 内置一个 I2S 接口，为多媒体应用，尤其是为数字音频应用提供了灵活的数据通信接口。

I2S 标准总线定义了三种信号：串行时钟信号 BCK、字选择信号 WS 和串行数据信号 SD。一个基础的 I2S 数据总线包含一个主机和一个从机。主机和从机的角色在通信过程中保持不变。ESP32-H2 的 I2S 模块包含独立的发送单元和接收单元，能够保证优良的通信性能。

28.2 术语

为了更好地说明 I2S 的功能，本章使用了以下术语。

主机模式 I2S 作为主机驱动 BCK/WS，向从机发送或从其接收数据。

从机模式 I2S 作为从机由 BCK/WS 驱动，从主机接收或向其发送数据。

全双工 主机与从机之间的发送线和接收线各自独立，发送数据和接收数据同时进行。

半双工 主机和从机只能有一方先发送数据，另一方接收数据。发送数据和接收数据不能同时进行。

A 律和 μ 律 A 律和 μ 律是数字脉冲编码调制 (PCM) 非均匀量化中的压缩/解压缩算法，可以有效地改善信号的量化信噪比。

TDM RX 模式 即 TDM 输入模式，利用时分复用方式接收脉冲编码调制 (PCM) 数据，并将其通过 DMA 存入储存器的模式。信号线包括 BCK、WS 和 DATA。可以接收最多 16 个通道的数据。通过用户配置，可支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。

普通 PDM RX 模式 即普通 PDM 输入模式，接收脉冲密度调制 (PDM) 数据，并将其通过 DMA 存入储存器的模式。信号线包括 WS 和 DATA。通过用户配置，可支持 PDM 标准格式等。

TDM TX 模式 即 TDM 输出模式，通过 DMA 从储存器中取得脉冲编码调制 (PCM) 数据，并利用时分复用方式将其发送的模式。信号线包括 BCK、WS 和 DATA，可以发送最多 16 个通道的数据。通过用户配置，可支持 TDM Philips 格式、TDM MSB 对齐格式、TDM PCM 格式等。

普通 PDM TX 模式 即普通 PDM 输出模式，通过 DMA 从储存器中取得脉冲密度调制 (PDM) 数据，并将其发送的模式。信号线包括 WS 和 DATA。通过用户配置，可支持 PDM 标准格式等。

PCM-to-PDM TX 模式 即 PCM-to-PDM 输出模式，通过 DMA 从储存器中取得脉冲编码调制 (PCM) 数据，将其转换为脉冲密度调制 (PDM) 数据，并将其发送的**主机**模式。信号线包括 WS 和 DATA。通过用户配置，可支持 PDM 标准格式等。

28.3 特性

I2S 模块具有以下特性：

- 支持主机模式和从机模式
- 支持全双工和半双工通信

- 支持 TX 模块和 RX 模块独立工作或同时工作
- 支持多种音频标准：
  - TDM Philips 标准
  - TDM MSB 对齐标准
  - TDM PCM 标准
  - PDM 标准
- 可配置高精度采样时钟，支持多种采样频率
- 支持 8/16/24/32 位数据通信
- 支持 DMA
- 支持 I2S 接口中断

说明：
从机模式下，由于时钟源的频率限制，ESP32-H2 I2S 支持的最高采样频率受到数据位宽和声道数的限制。例如，双声道 32 位宽时，支持 187.5 kHz 以下的采样频率，例如 8 kHz、16 kHz、32 kHz、44.1 kHz、48 kHz、88.2 kHz、96 kHz、128 kHz。具体请参考章节 28.6。

28.4 系统架构

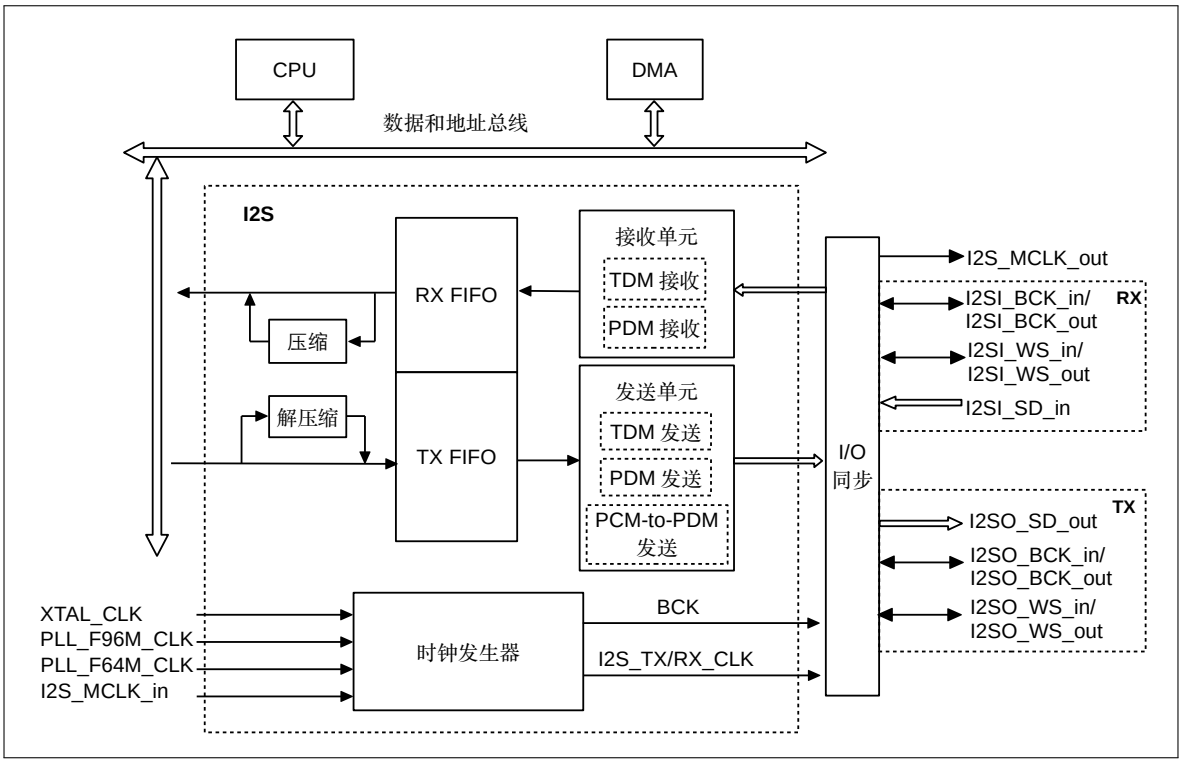


图 28-1. ESP32-H2 I2S 系统框图

图 28-1 为 ESP32-H2 I2S 模块的结构框图。ESP32-H2 I2S 模块包含：

- 独立的发送单元

- 独立的接收单元
- 输入输出时序调节单元（I/O 同步）
- 时钟分频器（时钟发生器）
- 64 x 32-bit TX FIFO
- 64 x 32-bit RX FIFO
- 压缩/解压缩模块

ESP32-H2 I2S 模块支持 DMA，可访问内部存储器，更多信息见章节 3 通用 DMA 控制器 (GDMA)。

发送单元和接收单元各自有一组三线接口，分别为串行时钟线 BCK、字选择线 WS 和串行数据线 SD。其中，发送单元的 SD 线固定为输出，接收单元的 SD 线固定为接收。发送单元和接收单元的 BCK 和 WS 信号线均可配置为主机输出模式或从机输入模式。

图 28-1 中的右半部分展示了 I2S 模块的信号总线。RX 和 TX 模块的信号命名规则为：I2SA_B_C，例如 I2SI_BCK_in。其中：

- “A” 表示 I2S 模块的数据总线的方向，包括：
  - “I” 表示输入
  - “O” 表示输出
- “B” 表示信号功能，包括：
  - BCK
  - WS
  - SD
- “C” 表示该信号的方向，包括：
  - “in” 表示该信号输入 I2S 模块
  - “out” 表示该信号自 I2S 模块输出

I2S 各信号的具体描述见表 28-2。

表 28-2. 模块信号描述

信号	方向	功能
I2SI_BCK_in	输入	I2S 从机模式下，输入 BCK 信号，用于 RX 模块
I2SI_BCK_out	输出	I2S 主机模式下，输出 BCK 信号，用于 RX 模块
I2SI_WS_in	输入	I2S 从机模式下，输入 WS 信号，用于 RX 模块
I2SI_WS_out	输出	I2S 主机模式下，输出 WS 信号，用于 RX 模块
I2SI_Data_in	输入	I2S RX 模块的串行输入数据线
I2SO_Data_out	输出	I2S TX 模块的串行输出数据线
I2SO_BCK_in	输入	I2S 从机模式下，输入 BCK 信号，用于 TX 模块
I2SO_BCK_out	输出	I2S 主机模式下，输出 BCK 信号，用于 TX 模块
I2SO_WS_in	输入	I2S 从机模式下，输入 WS 信号，用于 TX 模块
I2SO_WS_out	输出	I2S 主机模式下，输出 WS 信号，用于 TX 模块
I2S_MCLK_in	输入	I2S 从机模式下，来自外部芯片的时钟源
I2S_MCLK_out	输出	I2S 主机模式下，作为外部芯片的时钟源

说明：
I2S 的所有信号均需要经过 GPIO 交换矩阵映射到芯片的管脚。更多信息请参考章节 6 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)*。

28.5 I2S 模块支持的音频协议

ESP32-H2 I2S 模块支持多种音频标准，包括 TDM Philips 标准、TDM MSB 对齐标准、TDM PCM 标准以及 PDM 标准。

用户可通过配置以下寄存器，选择所需的音频标准：

- I2S_TX/RX_TDM_EN
  - 0：禁用 TDM 模式
  - 1：选择 TDM 模式
- I2S_TX/RX_PDM_EN
  - 0：禁用 PDM 模式
  - 1：选择 PDM 模式
- I2S_TX/RX_MSB_SHIFT
  - 0：配置 WS 信号和 SD 信号同时开始变化，即选择 MSB 对齐标准
  - 1：配置 WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即选择 Philips 标准或 PCM 标准

说明：
不可将 I2S_TX/RX_TDM_EN 和 I2S_TX/RX_PDM_EN 同时配置为 1 或同时配置为 0，否则会导致 ESP32-H2 I2S 以既非 TDM 模式也非 PDM 模式的状态错误发送数据。

28.5.1 TDM Philips 标准模式

在 Philips 标准下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并在当前通道数据发送结束前一个 BCK 时钟周期开始变化。SD 信号线上首先传输音频数据的最高位。

与 Philips 标准相比，TDM Philips 标准支持的通道更多，见图 28-2。

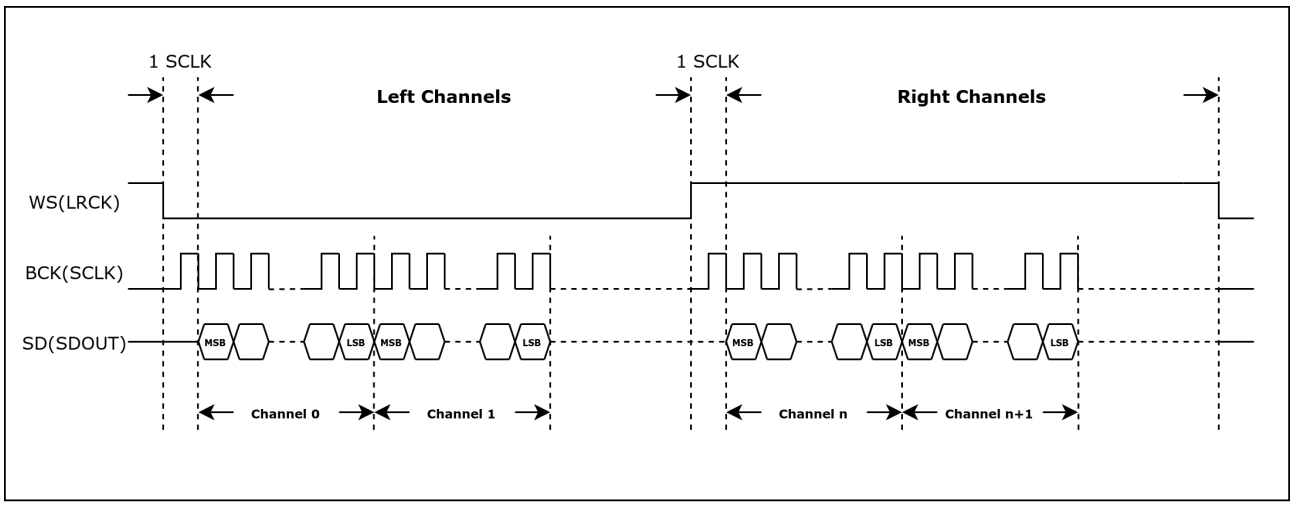


图 28-2. 时序图 – TDM Philips 标准

28.5.2 TDM MSB 对齐标准模式

MSB 对齐标准下，在 BCK 下降沿，WS 信号和 SD 信号同时变化。WS 持续到当前通道数据发送结束，SD 信号线上首先传输音频数据的最高位。

与 MSB 对齐标准相比，TDM MSB 对齐标准支持更多通道，见图 28-3。

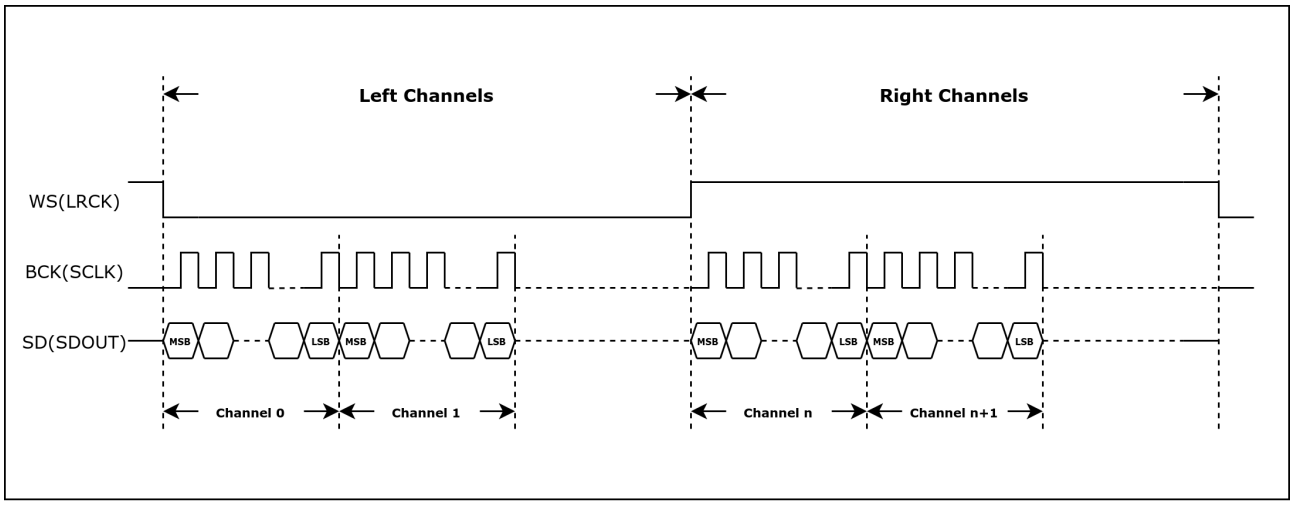


图 28-3. 时序图 – TDM MSB 对齐标准

28.5.3 TDM PCM 标准模式

在 PCM 标准的短帧同步模式下，在 BCK 的下降沿，WS 信号先于 SD 信号一个 BCK 时钟周期开始变化，即 WS 信号从当前通道数据的第一个位之前的一个时钟开始有效，并持续一个 BCK 时钟周期。SD 信号线上首先传输音频数据的最高位。

与 PCM 标准相比，TDM PCM 标准支持更多通道，见图 28-4 所示。

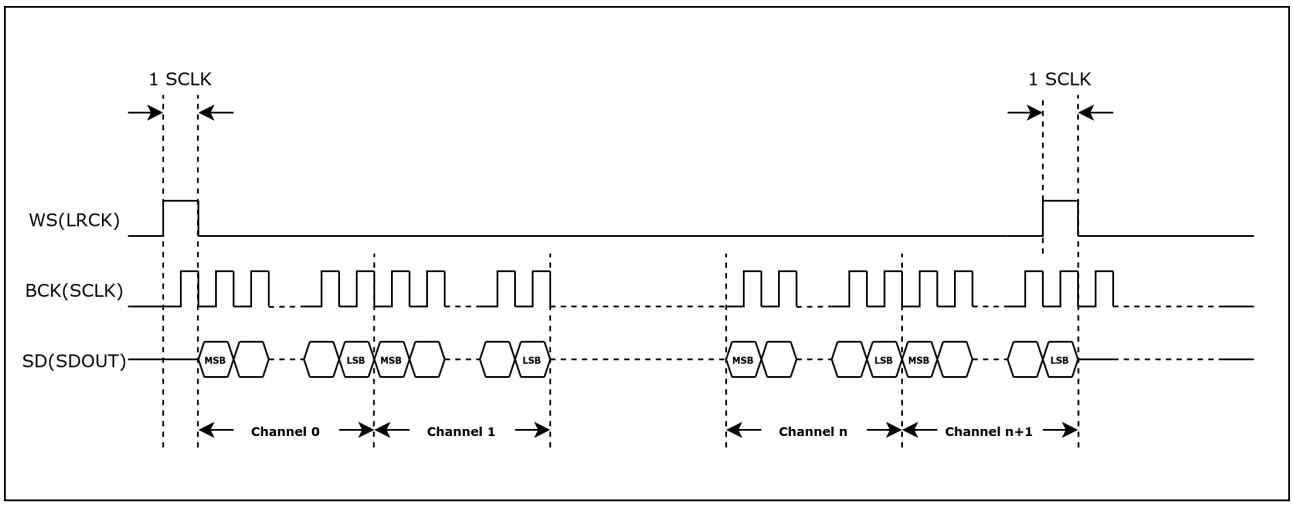


图 28-4. 时序图 – TDM PCM 标准

28.5.4 PDM 标准模式

如图 28-5 所示，在 PDM 标准下，WS 代表左/右声道，在 BCK 的下降沿，WS 与 SD 同时变化。WS 在数据发送过程中持续变化，WS 的高低对应两个声道。

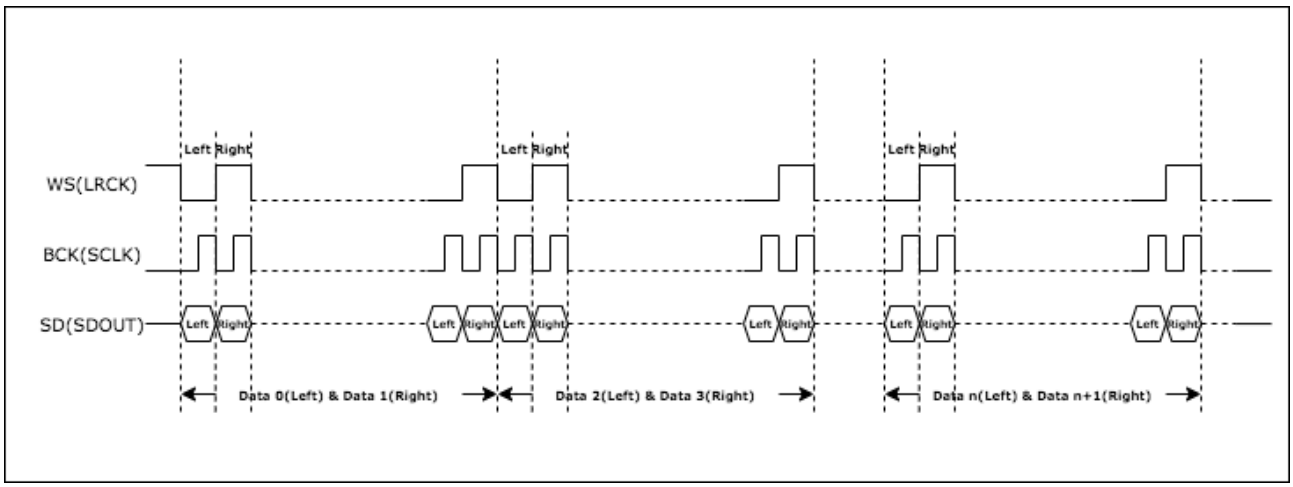


图 28-5. 时序图 – PDM 标准

28.6 I2S TX/RX 模块时钟

I2S_TX/RX_CLK 作为 I2S TX/RX 模块的主时钟，可由以下时钟分频所得：

- 40 MHz XTAL_CLK
- 96 MHz PLL_F96M_CLK
- 64 MHz PLL_F64M_CLK
- 外部输入时钟 I2S_MCLK_in

I2S TX/RX 模块的串行时钟 BCK 再由 I2S_TX/RX_CLK 分频获得，如图 28-6 所示。PCR_I2S_TX/RX_CLKM_SEL 用于选择 I2S TX/RX 的时钟源，PCR_I2S_TX/RX_CLKM_EN 用于使能或者关闭 I2S TX/RX 模块的时钟源。

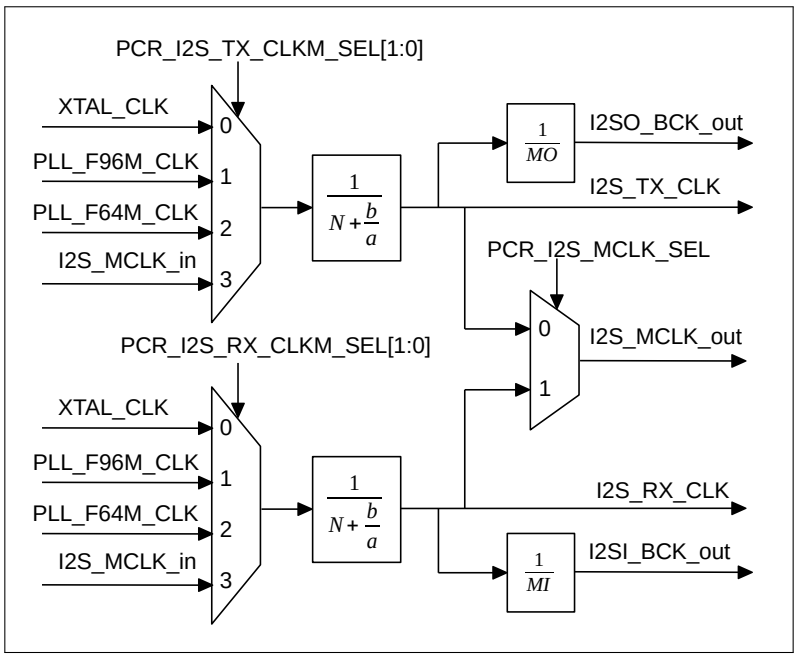


图 28-6. I2S 时钟分频器

I2S_TX/RX_CLK 的频率 $f_{\text{I2S\_TX/RX\_CLK}}$ 与分频器时钟源频率 $f_{\text{I2S\_CLK\_S}}$ 间的关系如下：

$$f_{\text{I2S\_TX/RX\_CLK}} = \frac{f_{\text{I2S\_CLK\_S}}}{\text{N} + \frac{\text{b}}{\text{a}}}$$

其中，2 =< N <= 256（N 为整数），N 对应为 PCR_I2S_TX/RX_CLKM_CONF_REG 寄存器中 PCR_I2S_TX/RX_CLKM_DIV_NUM 的值，具体为：

- PCR_I2S_TX/RX_CLKM_DIV_NUM = 0 时，N = 256；
- PCR_I2S_TX/RX_CLKM_DIV_NUM = 1 时，N = 2；
- PCR_I2S_TX/RX_CLKM_DIV_NUM 为其他值时，N = PCR_I2S_TX/RX_CLKM_DIV_NUM 的值。

分数部分分频系数 a、b 唯一对应系数 x、y、z、yn1。系数对应公式为：

- 当 b <= $\frac{a}{2}$ 时，yn1 = 0，x = $floor([\frac{a}{b}]) - 1$，y = a%b，z = b；
- 当 b > $\frac{a}{2}$ 时，yn1 = 1，x = $floor([\frac{a}{a-b}]) - 1$，y = a%(a - b)，z = a - b。

系数 x、y、z、yn1 在 PCR_I2S_TX/RX_CLKM_DIV_X、PCR_I2S_TX/RX_CLKM_DIV_Y、PCR_I2S_TX/RX_CLKM_DIV_Z、PCR_I2S_TX/RXCLKM_DIV_YN1 中配置。
对于整数分频，PCR_I2S_TX/RX_CLKM_DIV_X 和 PCR_I2S_TX/RX_CLKM_DIV_Z 清零，PCR_I2S_TX/RX_CLKM_DIV_Y 置为 1。

> **说明：**
> 使用小数分频功能可能会产生时钟抖动。

在主机发送模式下，I2S TX 模块的串行时钟 BCK 为 I2SO_BCK_out 信号，由 I2S_TX_CLK 分频获得。即：

$$f_{\text{I2SO\_BCK\_out}} = \frac{f_{\text{I2S\_TX\_CLK}}}{\text{MO}}$$

其中，MO 的值为 I2S_TX_BCK_DIV_NUM 的值 + 1，即：

$$MO = I2S\_TX\_BCK\_DIV\_NUM + 1$$

说明：
注意，I2S_TX_BCK_DIV_NUM 不可配置为 1。

在主机接收模式下，I2S RX 模块的串行时钟 BCK 为 I2SI_BCK_out 信号，由 I2S_RX_CLK 分频获得。即：

$$f_{I2SI\_BCK\_out} = \frac{f_{I2S\_RX\_CLK}}{MI}$$

其中 MI 对应 I2S_RX_BCK_DIV_NUM 的值 + 1，即：

$$MI = I2S\_RX\_BCK\_DIV\_NUM + 1$$

说明：

- I2S_RX_BCK_DIV_NUM 不可配置为 1；
- 当模块处于 I2S 从机模式时，必须保证 f_{I2S_TX/RX_CLK} >= 8 * f_{BCK}。另外模块可以输出 I2S_MCLK_out 作为外部设备的主时钟。

28.7 I2S 模块复位

I2S 模块中各个单元以及 FIFO 可通过配置相关位进行复位：

- I2S TX/RX 单元：可配置 I2S_TX_RESET 和 I2S_RX_RESET 进行复位；
- I2S TX/RX FIFO：可配置 I2S_TX_FIFO_RESET 和 I2S_RX_FIFO_RESET 进行复位。

说明：
在模块和 FIFO 复位之前，需要先配置 I2S 模块时钟。

28.8 I2S 主/从机模式

ESP32-H2 I2S 模块可作为主机或从机，两种模式均支持半双工通信或全双工通信。用户可配置 I2S_RX_SLAVE_MOD 和 I2S_TX_SLAVE_MOD 选择需要的模式。

- I2S_TX_SLAVE_MOD
  - 0：主机发送模式
  - 1：从机发送模式
- I2S_RX_SLAVE_MOD
  - 0：主机接收模式
  - 1：从机接收模式

28.8.1 主/从机发送模式

- 主机发送模式
  - 置位 I2S_TX_START 启动一次发送操作。
  - 置位该位，发送单元会一直输出时钟信号和串行数据。
  - 置位 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，则主机停止发送数据和时钟信号。
  - 清零 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，发送模块将一直发送最后一帧数据和时钟信号。
  - 当 I2S_TX_START 被清零时，主机停止发送数据。
- 从机发送模式
  - 置位 I2S_TX_START。
  - 发送单元等待主机 BCK 时钟，来启动发送操作。
  - 置位 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，则从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。
  - 清零 I2S_TX_STOP_EN 时，如果 FIFO 中的数据全部发送完毕，并且没有新数据填入，发送单元将一直发送最后一帧数据。
  - 当 I2S_TX_START 被清零时，从机发送数据一直为零，直到主机停止发送 BCK 时钟为止。

28.8.2 主/从机接收模式

- 主机接收模式
  - 置位 I2S_RX_START 启动一次接收操作。
  - 接收单元会一直输出时钟信号，并对输入数据进行采样。
  - 配置 I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    - 0：只有 I2S_RX_START 被清除，接收单元才会暂停接收；
    - 1：接收单元会在 I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收；
    - 2：接收单元会在 I2S_RX_START 被清除，或者 DMA RX FIFO 满时暂停接收。
  - 清零 I2S_RX_START，接收单元停止接收数据。
- 从机接收模式
  - 置位 I2S_RX_START。
  - 等待主机 BCK 时钟，来启动接收操作。
  - 配置 I2S_RX_STOP_MODE 来控制接收模式何时暂停接收：
    - 0：只有 I2S_RX_START 被清除，接收单元才会暂停接收；
    - 1：接收单元会在 I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收；
    - 2：接收单元会在 I2S_RX_START 被清除，或者 DMA RX FIFO 满时暂停接收。

– 清零 I2S_RX_START，接收单元停止接收数据。

28.9 发送数据

说明：
本小节以及后续小节所述的配置，均需要通过置位 I2S_TX_UPDATE 的方式来进行更新，从而将 I2S TX 寄存器数据从 APB 时钟域同步到 I2S TX 时钟域。详细配置见第 28.11.1 小节。

ESP32-H2 I2S 发送数据时，从 DMA 读取数据，经过数据格式控制和通道模式控制，从外设输出信号输出对应数据。

28.9.1 数据格式控制

数据格式控制分为三个阶段：

- 第一阶段：从内存中读出有效数据并写入 TX FIFO；
- 第二阶段：将待发送数据从 TX FIFO 中读出，并进行输出数据模式转换；
- 第三阶段：将待发送数据转换为串行数据流输出。

28.9.1.1 通道有效数据位宽

I2S_TX_BITS_MOD 和 I2S_TX_24_FILL_EN 决定了每个通道的有效数据位宽，其可取值和对应的有效数据位宽如下表：

表 28-3. 通道有效数据位宽控制

通道有效数据位宽	I2S_TX_BITS_MOD	I2S_TX_24_FILL_EN
32	31	x*
	23	1
24	23	0
16	15	x
8	7	x

* “x”表示该值被忽略。

28.9.1.2 通道有效数据字节序

I2S 通过 DMA 读取数据之后，I2S_TX_BIG_ENDIAN 用于控制从 DMA 读取数据的字节序。表 28-4 描述了不同通道有效数据位宽下，该寄存器对读取数据的控制。

表 28-4. 通道有效数据字节序控制

通道有效数据位宽	原始数据	控制后数据字节序	I2S_TX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

说明：
B0、B1、B2、B3 各代表一个 8 位数据，符号 {} 表示这些字节被组合到一起。例如，{B3, B2, B1, B0} 代表一个 32 位的数据，其中 B0 代表 0-7 位，B1 代表 8-15 位，B2 代表 16-23 位，B3 代表 24-31 位。

28.9.1.3 A 率/μ 率压缩/解压缩

ESP32-H2 I2S 按照 32-bit（缺省高位补 0*）的方式对排列好字节序的有效数据进行 A 率/μ 率压缩/解压缩。

说明：
缺省高位补 0，即如果有效数据位宽小于 32，则待压缩/解压缩数据的 [31: 有效位宽] 位自动填充 0。

配置 I2S_TX_PCM_BYPASS 为：

- 0：不进行压缩/解压缩
- 1：进行压缩/解压缩

配置 I2S_TX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2：μ 律解压缩
- 3：μ 律压缩

至此，数据格式控制的第一阶段完成。

28.9.1.4 通道发送数据位宽

ESP32-H2 I2S 中，I2S_TX_TDM_CHAN_BITS 决定了每个通道发送数据的位宽。

- 当每个通道发送数据的位宽大于有效数据位宽时，会在剩余的位置补充 0。此时，配置 I2S_TX_LEFT_ALIGN 为：
  - 0：有效数据位于发送数据低位，在数据高位补零；
  - 1：有效数据位于发送数据高位，在数据低位补零。

- 当每个通道发送数据的位宽小于有效数据位宽时，每次发送数据仅发送低位有效数据部分，高位部分将被舍弃。

至此，数据格式控制的第二阶段完成。

28.9.1.5 通道数据比特顺序

ESP32-H2 I2S 通道数据比特顺序由 I2S_TX_BIT_ORDER 控制：

- 0：数据从高位向低位依次发送；
- 1：数据从低位向高位依次发送。

至此，数据格式控制部分全部完成。图 28-7 所示即为一次完整的 TX 数据格式控制过程。

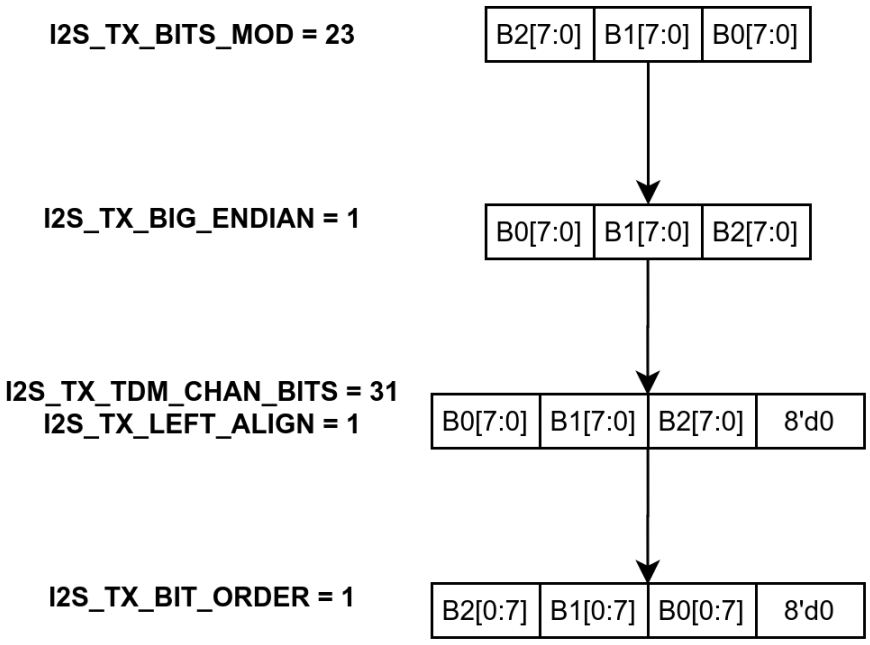


图 28-7. TX 数据格式控制

28.9.2 通道模式控制

ESP32-H2 I2S 支持 TDM 和 PDM 两种发送模式。置位 I2S_TX_TDM_EN 使能 TDM 发送模式，置位 I2S_TX_PDM_EN 使能 PDM 发送模式。

说明：

- I2S_TX_TDM_EN 和 I2S_TX_PDM_EN 不能同时置位或同时清零。
- 将 I2S 模块设置为 TDM 双通道模式，可实现控制大多数 I2S 双声道编解码器。

28.9.2.1 TDM 输出模式下 I2S 通道模式

在 TDM 输出模式下，I2S 最多支持 16 个发送通道。发送通道数由 I2S_TX_TDM_TOT_CHAN_NUM 控制。例如，配置 I2S_TX_TDM_TOT_CHAN_NUM 为 5，则六个通道（通道 0 ~ 5）将用于发送数据，见图 28-8。

在发送数据的通道中，如果其对应的 I2S_TX_TDM_CHANn_EN 为：

- 1：则该通道发送通道数据；
- 0：则该通道发送数据由 I2S_TX_CHAN_EQUAL 控制，当该值为：
  - 1：则发送上个通道的数据；
  - 0：则发送 I2S_SINGLE_DATA 的值。

当 I2S 处于主机 TDM 输出模式下，WS 信号由 I2S_TX_WS_IDLE_POL 和 I2S_TX_TDM_WS_WIDTH 控制：

- I2S_TX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- I2S_TX_TDM_WS_WIDTH 的值为在发送所有通道数据的过程中，WS 为默认电平的周期数。

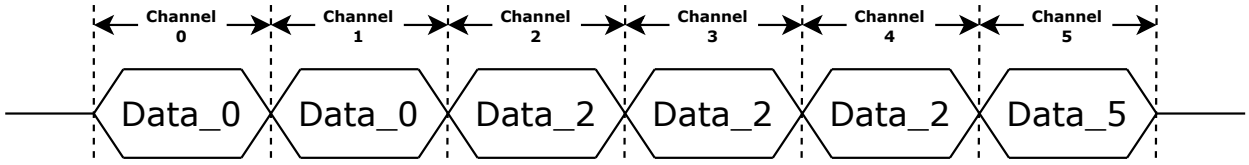
另外，I2S_TX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

TDM 通道配置示例

在本示例中，寄存器的配置如下：

- 配置 I2S_TX_TDM_CHAN_NUM 为 5，即选择使用通道 0 ~ 5 进行数据发送。
- 配置 I2S_TX_CHAN_EQUAL 为 1，即如果相应通道的 I2S_TX_TDM_CHAN*n*_EN 被置位，则该通道将发送上一通道的数据。*n* = 0 ~ 5。
- 配置 I2S_TX_TDM_CHAN0/2/5_EN 为 1，即这些通道将用于发送通道数据。
- 配置 I2S_TX_TDM_CHAN1/3/4_EN 为 0，则这些通道将发送上一个通道的数据。

配置完成后，则数据将按下图方式进行发送。



I2S_TX_TDM_CHAN_NUM = 5; I2S_TX_CHAN_EQUAL = 1;
I2S_TX_TDM_CHAN0_EN = 1; I2S_TX_TDM_CHAN1_EN = 0; I2S_TX_TDM_CHAN2_EN = 1;
I2S_TX_TDM_CHAN3_EN = 0; I2S_TX_TDM_CHAN4_EN = 0; I2S_TX_TDM_CHAN5_EN = 1;

图 28-8. TDM 通道控制

28.9.2.2 PDM 输出模式下 I2S 通道模式

ESP32-H2 I2S 支持两种 PDM 输出模式，即普通 PDM 输出模式和 PCM-to-PDM 输出模式。

在 PDM 输出模式下，从 DMA 取得数据的过程由 I2S_TX_MONO 和 I2S_TX_MONO_FST_VLD 控制，具体如下表。请根据内存中存储的数据为单/双通道数据来配置该寄存器。

表 28-5. PDM 输出模式下 I2S 取数逻辑

取数逻辑	模式	I2S_TX_MONO	I2S_TX_MONO_FST_VLD
每个 WS 沿都向 DMA 发起取数请求	双声道	0	x
只在 WS 后半周期向 DMA 发起取数请求	单声道	1	0
只在 WS 前半周期向 DMA 发起取数请求	单声道	1	1

当 I2S 处于主机 PDM 输出模式时，WS 信号的默认电平由 I2S_TX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半。请参考 28.6 小节配置 BCK 信号的方式来配置 WS 信号，见图 28-9。

在**普通 PDM 输出模式**下，I2S 通道数据由 I2S_TX_CHAN_MOD 和 I2S_TX_WS_IDLE_POL 控制，具体如下表。

表 28-6. 普通 PDM 输出模式下 I2S 通道模式控制

模式	左声道	右声道	模式控制字段[1]	声道选择位[2]
双声道	发送左通道数据	发送右通道数据	0	x
单声道	发送左通道数据	发送左通道数据	1	0
	发送右通道数据	发送右通道数据	1	1
	发送右通道数据	发送右通道数据	2	0
	发送左通道数据	发送左通道数据	2	1
	发送 “single” 值[3]	发送右通道数据	3	0
	发送左通道数据	发送 “single” 值	3	1
	发送左通道数据	发送 “single” 值	4	0
	发送 “single” 值	发送右通道数据	4	1

[1] I2S_TX_CHAN_MOD

[2] I2S_TX_WS_IDLE_POL

[3] single 值等于 I2S_SINGLE_DATA 的值。

在 **PCM-to-PDM 输出模式**下，可以将 DMA 中的 PCM 数据转为 PDM 数据，并按照 PDM 信号格式输出，配置 I2S_PCM2PDM_CONV_EN 启动该模式。PCM-to-PDM 输出模式的寄存器配置如下：

- 配置一线 PDM 输出格式或一/二线 DAC 输出格式，具体如下表。

表 28-7. PCM-to-PDM 输出模式

通道输出格式	I2S_TX_PDM_DAC_MODE_EN	I2S_TX_PDM_DAC_2OUT_EN
一线 PDM 输出格式[1]	0	x
一线 DAC 输出格式[2]	1	0
二线 DAC 输出格式	1	1

说明：

1. 此处定义的 PDM 输出格式是指一个 WS 周期发送两个通道的 SD 数据。
2. 此处定义的 DAC 输出格式是指一个 WS 周期发送一个通道的 SD 数据。

- 配置采样频率和上采样率。
  ESP32-H2 I2S PCM 转 PDM 输出模式下，PDM 时钟频率即为 BCK 时钟频率。采样频率 ($f_{Sampling}$) 与

BCK 时钟频率的关系如下：

$$f_{Sampling} = \frac{f_{BCK}}{OSR}$$

其中，上采样率 OSR 和 I2S_TX_PDM_SINC_OSR2 的关系为：

$$OSR = I2S\_TX\_PDM\_SINC\_OSR2 \times 64$$

采样频率 $f_{Sampling}$ 和 I2S_TX_PDM_FS 的对应关系为：

$$f_{Sampling} = I2S\_TX\_PDM\_FS \times 100$$

请根据需要的采样频率、上采样率以及 PDM 时钟频率进行寄存器配置。

PDM 通道配置示例

在本示例中，寄存器的配置如下：

- 配置 I2S_PCM2PDM_CONV_EN 为 0，即选择普通 PDM 输出模式。
- 配置 I2S_TX_MONO 为 0，即 I2S 在 WS 的高低电平均通过 DMA 从存储器取数。
- 配置 I2S_TX_CHAN_MOD 为 2，即选择单声道模式，右声道数据将被舍弃。
- 配置 I2S_TX_WS_IDLE_POL 为 1，则左声道和右声道均发送左通道数据。

配置完成后，假定存储器中的数据**经过数据格式控制之后**为：

Left	Right	Left	Right	...	Left	Right

说明：

1. 此处的数据并非存储器中的原始数据，而是经过数据格式控制之后的数据。
2. 表中的 “Left” 和 “Right” 代表通道数据，其位宽为通道有效数据位宽，请参考章节 28.9.1。

则数据经过通道模式控制后，将按照下图方式进行发送。

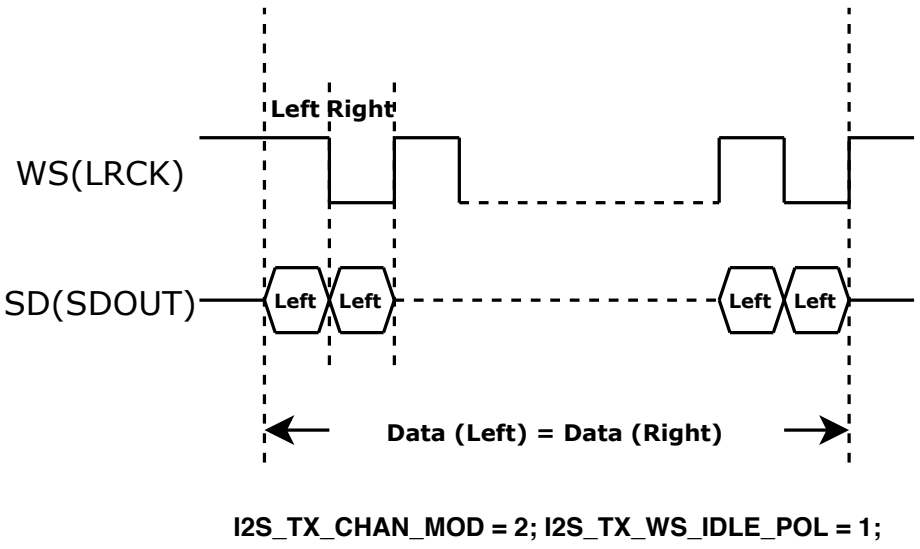


图 28-9. PDM 通道控制

28.10 接收数据

I2S 接收数据时，从外设接口读取数据，经过通道模式控制和数据格式控制，通过 DMA 输入数据至内存。

28.10.1 通道模式控制

ESP32-H2 I2S 支持 TDM 和 PDM 两种输入模式。置位 I2S_RX_TDM_EN 使能 TDM 输入模式，置位 I2S_RX_PDM_EN 使能 PDM 输入模式。

> **说明：**
> I2S_RX_TDM_EN 和 I2S_RX_PDM_EN 不能同时置位或同时清零。

28.10.1.1 TDM 输入模式下 I2S 通道模式

在 TDM 输入模式下，I2S 最多支持 16 个接收通道。接收通道数由 I2S_RX_TDM_TOT_CHAN_NUM 控制。例如，配置 I2S_RX_TDM_TOT_CHAN_NUM 为 5，则通道 0 ~ 5 接收数据。

在接收数据的通道中，如果其对应的 I2S_RX_TDM_CHANn_EN 为：

- 0：则该通道数据无效，不会存入 RX FIFO；
- 1：则该通道数据有效，会被存入 RX FIFO。

当 I2S 处于主机 TDM 输入模式下，WS 信号由 I2S_RX_WS_IDLE_POL 和 I2S_RX_TDM_WS_WIDTH 控制：

- I2S_RX_WS_IDLE_POL 的值为 WS 信号的默认电平；
- I2S_RX_TDM_WS_WIDTH 的值为在接收所有通道的过程中，WS 为默认电平的周期数。

另外，I2S_RX_HALF_SAMPLE_BITS 的值乘以 2，即为一个 WS 周期对应的 BCK 周期数。

28.10.1.2 PDM 输入模式下 I2S 通道模式

在 PDM 输入模式下，I2S 将通道中的串行数据转换为待输入数据。

当 I2S 处于主机 PDM 输入模式下，WS 信号的默认电平由 I2S_RX_WS_IDLE_POL 控制，WS 信号频率为 BCK 信号频率的一半，请参考 28.6 小节配置 BCK 信号的方式配置 WS 信号。注意，在 PDM 输入模式下，请配置 I2S_RX_HALF_SAMPLE_BITS 的值与 I2S_RX_BITS_MOD 的值相同。

28.10.2 数据格式控制

数据格式控制分为两个阶段：

- 第一阶段：从串行数据流输入转换为待输入数据，并存入 RX FIFO；
- 第二阶段：将待输入数据从 RX FIFO 中读出，并进行输入数据模式转换。

28.10.2.1 通道数据比特顺序

ESP32-H2 I2S 通道数据比特顺序由 I2S_RX_BIT_ORDER 控制，当该值为：

- 0：串行数据从高位向低位依次存入待输入数据；

- 1：串行数据从低位向高位依次存入待输入数据。

至此，数据格式控制的第一阶段完成。

28.10.2.2 通道储存数据位宽

I2S_RX_BITS_MOD 和 I2S_RX_24_FILL_EN 决定了每个通道的储存数据位宽，其可取值和对应的储存数据位宽如下表：

表 28-8. 通道储存数据位宽控制

通道储存数据位宽	I2S_RX_BITS_MOD	I2S_RX_24_FILL_EN
32	31	x
	23	1
24	23	0
16	15	x
8	7	x

28.10.2.3 通道接收数据位宽

ESP32-H2 I2S 中，I2S_RX_TDM_CHAN_BITS 决定了每个通道接收数据的位宽。

- 当每个通道储存数据位宽小于接收数据的位宽时，接收时仅会在接收数据中取储存位宽作为储存数据。此时，配置 I2S_RX_LEFT_ALIGN 为：
  - 0：储存数据位于接收数据低位；
  - 1：储存数据位于接收数据高位。
- 当每个通道接收数据的位宽小于储存数据位宽时，会将接收数据高位补 0，作为储存数据。

28.10.2.4 通道储存数据字节序

通道接收数据经过截取/补全后，成为了待储存数据，I2S_RX_BIG_ENDIAN 用于控制待储存数据的字节序。下表描述了不同通道储存数据位宽下，该寄存器对待储存数据的控制。

表 28-9. 通道储存数据字节序控制

通道储存数据位宽	原始数据	控制后数据	I2S_RX_BIG_ENDIAN
32	{B3, B2, B1, B0}	{B3, B2, B1, B0}	0
		{B0, B1, B2, B3}	1
24	{B2, B1, B0}	{B2, B1, B0}	0
		{B0, B1, B2}	1
16	{B1, B0}	{B1, B0}	0
		{B0, B1}	1
8	{B0}	{B0}	x

28.10.2.5 A 率/μ 率压缩/解压缩

ESP32-H2 I2S 对排列好字节序的待储存数据会按照 32-bit（缺省高位补 0）的方式进行 A 率/μ 率压缩/解压缩。

配置 I2S_RX_PCM_BYPASS 为：

- 0：不进行压缩/解压缩
- 1：进行压缩/解压缩

配置 I2S_RX_PCM_CONF 为：

- 0：A 律解压缩
- 1：A 律压缩
- 2，μ 律解压缩
- 3，μ 律压缩

至此，数据格式控制部分全部完成，数据通过 DMA 存入内存。

28.11 软件配置流程

28.11.1 软件配置 I2S 发送流程

软件配置 I2S 发送的流程如下：

1. 根据 28.6 小节的描述，配置时钟。
2. 根据表 28-2 的描述，配置信号管脚。
3. 根据主从机模式配置 I2S_TX_SLAVE_MOD：
   - 0：主机发送模式
   - 1：从机发送模式
4. 根据 28.9 小节的描述，配置正确的发送数据模式和发送通道模式，置位 I2S_TX_UPDATE。
5. 根据 28.7 小节的描述，复位发送单元和发送 FIFO。
6. 根据 28.12 小节的描述使能相应的中断。
7. 配置 DMA 发送链表。
8. 根据需要置位 I2S_TX_STOP_EN，更多信息见章节 28.8.1。
9. 开始发送数据：
   - 主机模式下，等待 I2S 从设备配置完成后，置位 I2S_TX_START 开始发送数据；
   - 从机模式下，置位 I2S_TX_START。I2S 主设备提供 BCK 和 WS 信号后，开始发送数据。
10. 等待步骤 6 设置的中断信号，或查询 I2S_TX_IDLE 检查传输是否结束：
    - 0：发送设备为工作状态；
    - 1：发送设备为空闲状态。
11. 清零 I2S_TX_START。

28.11.2 软件配置 I2S 接收流程

软件配置 I2S 接收模式的流程如下：

1. 根据 28.6 小节的描述，配置时钟。

2. 根据表 28-2 的描述，配置信号管脚。
3. 配置 I2S_RX_SLAVE_MOD 选择需要的模式：
    - 0：主机接收模式
    - 1：从机接收模式
4. 根据 28.10 小节的描述，配置正确的接收通道模式和接收数据模式，置位 I2S_RX_UPDATE。
5. 根据 28.7 小节的描述，复位接收单元和接收 FIFO。
6. 根据 28.12 小节的描述使能相应的中断。
7. 配置 DMA 接收链表，并在 I2S_RX_EOF_NUM_REG 中配置接收数据长度。
8. 开始接收数据：
    - 在主机模式下，等待从机准备好后，置位 I2S_RX_START 开始接收数据；
    - 在从机模式下，置位 I2S_RX_START，等待主机提供 BCK 和 WS 信号后开始接收数据。
9. 接收的数据由 DMA 根据配置，存到 ESP32-H2 存储器的指定地址。最终产生步骤 6 中设置的中断。

28.12 I2S 中断

- I2S_TX_HUNG_INT：当发送数据超时，触发此中断。例如，I2S 配置为从机发送模式，但主机长时间未提供 BCK 或 WS 信号，则将触发该中断。超时配置见寄存器 I2S_LC_HUNG_CONF_REG。
- I2S_RX_HUNG_INT：当接收数据超时，触发此中断。例如，I2S 配置为从机接收模式，但主机长时间未发送数据，则将触发该中断。超时配置见寄存器 I2S_LC_HUNG_CONF_REG。
- I2S_TX_DONE_INT：当发送数据完成，触发此中断。
- I2S_RX_DONE_INT：当接收数据完成，触发此中断。

28.12.1 事件任务矩阵功能

在 ESP32-H2 中，I2S 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 I2S 的 ETM 任务，或者通过 I2S 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与 I2S 相关的 ETM 任务和 ETM 事件。

I2S 可接收的 ETM 任务有：

- I2S_TASK_START_TX：触发时启动 I2S TX 数据传输。
- I2S_TASK_START_RX：触发时启动 I2S RX 数据传输。
- I2S_TASK_STOP_TX：触发时停止 I2S TX 数据传输。
- I2S_TASK_STOP_RX：触发时停止 I2S RX 数据传输。

GDMA 可产生的 ETM 事件有：

- I2S_EVT_TX_DONE：表示 I2S TX 发送完成。
- I2S_EVT_RX_DONE：表示 I2S RX 接收完成。
- I2S_EVT_X_WORDS_SENT：表示 I2S TX 发送字数大于等于 I2S_ETM_TX_SEND_WORD_NUM 的值。

- I2S_EVT_X_WORDS_RECEIVED：表示 I2S RX 接收字数大于等于 I2S_ETM_RX_RECEIVE_WORD_NUM 的值。

在具体应用中，可以配置 I2S 的 ETM 事件来触发 I2S 的 ETM 任务，例如配置 I2S_EVT_X_WORDS_SENT 事件触发 I2S_TASK_STOP_TX 任务，从而以 ETM 的方式完成停止 I2S 的操作。

28.13 寄存器列表

本小节的所有地址均为相对于 I2S 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

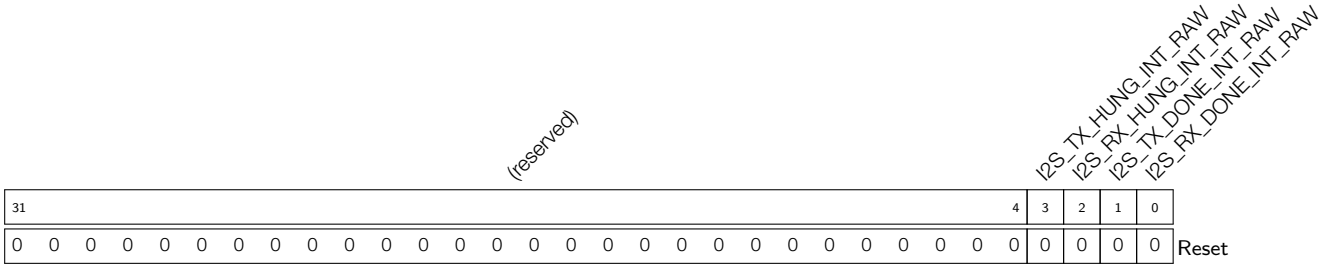
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
中断寄存器			
I2S_INT_RAW_REG	I2S 原始中断寄存器	0x000C	R/SS/WTC
I2S_INT_ST_REG	I2S 中断状态寄存器	0x0010	RO
I2S_INT_ENA_REG	I2S 中断使能寄存器	0x0014	R/W
I2S_INT_CLR_REG	I2S 中断清除寄存器	0x0018	WT
RX 控制和配置寄存器			
I2S_RX_CONF_REG	I2S RX 配置寄存器	0x0020	varies
I2S_RX_CONF1_REG	I2S RX 配置寄存器 1	0x0028	R/W
I2S_TX_PCM2PDM_CONF_REG	I2S TX PCM-to-PDM 配置寄存器	0x0040	R/W
I2S_TX_PCM2PDM_CONF1_REG	I2S TX PCM-to-PDM 配置寄存器 1	0x0044	R/W
I2S_RX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0050	R/W
I2S_RXEOF_NUM_REG	I2S RX 数据大小控制寄存器	0x0064	R/W
TX 控制和配置寄存器			
I2S_TX_CONF_REG	I2S TX 配置寄存器	0x0024	varies
I2S_TX_CONF1_REG	I2S TX 配置寄存器 1	0x002C	R/W
I2S_TX_TDM_CTRL_REG	I2S TX TDM 模式控制寄存器	0x0054	R/W
RX 时序寄存器			
I2S_RX_TIMING_REG	I2S RX 时序控制寄存器	0x0058	R/W
TX 时序寄存器			
I2S_TX_TIMING_REG	I2S TX 时序控制寄存器	0x005C	R/W
控制和配置寄存器			
I2S_LC_HUNG_CONF_REG	I2S 超时配置寄存器	0x0060	R/W
I2S_CONF_SIGLE_DATA_REG	I2S single 数据寄存器	0x0068	R/W
TX 状态寄存器			
I2S_STATE_REG	I2S TX 状态寄存器	0x006C	RO
ETM 寄存器			
I2S_ETM_CONF_REG	I2S ETM 配置寄存器	0x0070	R/W
版本寄存器			
I2S_DATE_REG	版本控制寄存器	0x0080	R/W

28.14 寄存器

本小节的所有地址均为相对 I2S 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 28.1. I2S_INT_RAW_REG (0x000C)



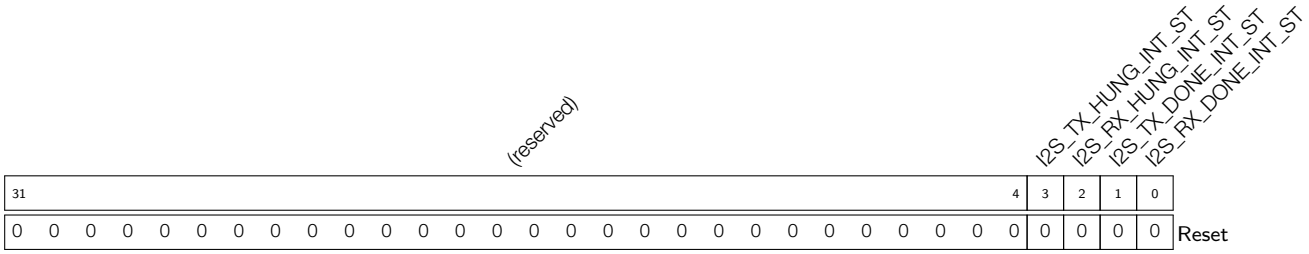
I2S_RX_DONE_INT_RAW I2S_RX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

I2S_TX_DONE_INT_RAW I2S_TX_DONE_INT 中断的原始中断状态。(R/SS/WTC)

I2S_RX_HUNG_INT_RAW I2S_RX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

I2S_TX_HUNG_INT_RAW I2S_TX_HUNG_INT 中断的原始中断状态。(R/SS/WTC)

Register 28.2. I2S_INT_ST_REG (0x0010)



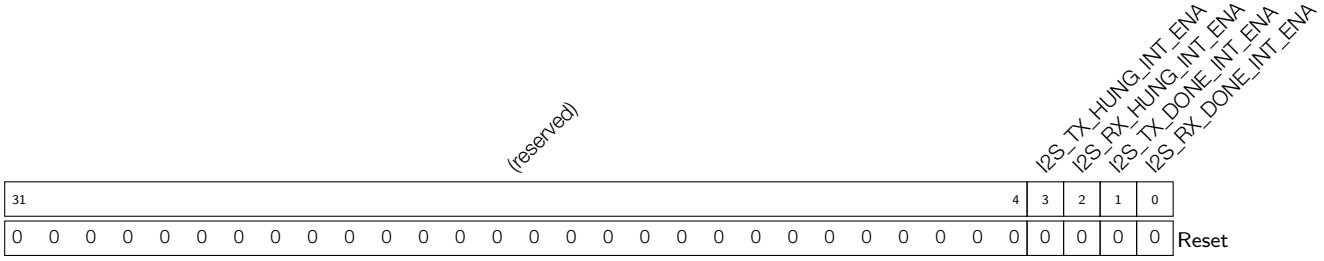
I2S_RX_DONE_INT_ST I2S_RX_DONE_INT 中断的屏蔽中断状态。(RO)

I2S_TX_DONE_INT_ST I2S_TX_DONE_INT 中断的屏蔽中断状态。(RO)

I2S_RX_HUNG_INT_ST I2S_RX_HUNG_INT 中断的屏蔽中断状态。(RO)

I2S_TX_HUNG_INT_ST I2S_TX_HUNG_INT 中断的屏蔽中断状态。(RO)

Register 28.3. I2S_INT_ENA_REG (0x0014)



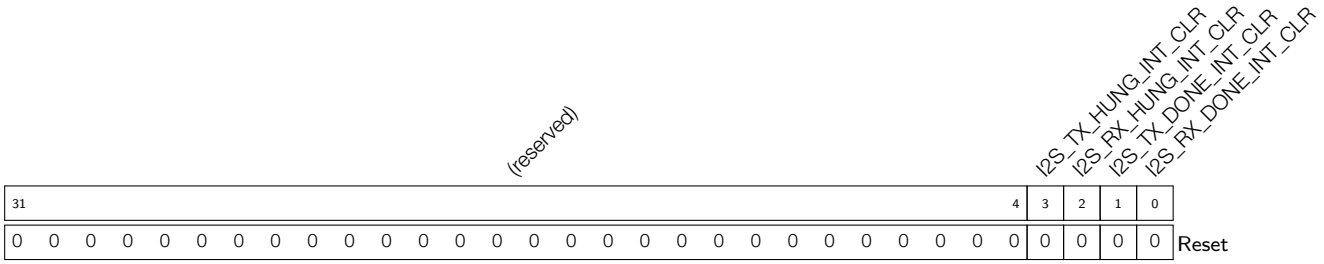
I2S_RX_DONE_INT_ENA 写 1 使能 I2S_RX_DONE_INT。(R/W)

I2S_TX_DONE_INT_ENA 写 1 使能 I2S_TX_DONE_INT。(R/W)

I2S_RX_HUNG_INT_ENA 写 1 使能 I2S_RX_HUNG_INT。(R/W)

I2S_TX_HUNG_INT_ENA 写 1 使能 I2S_TX_HUNG_INT。(R/W)

Register 28.4. I2S_INT_CLR_REG (0x0018)



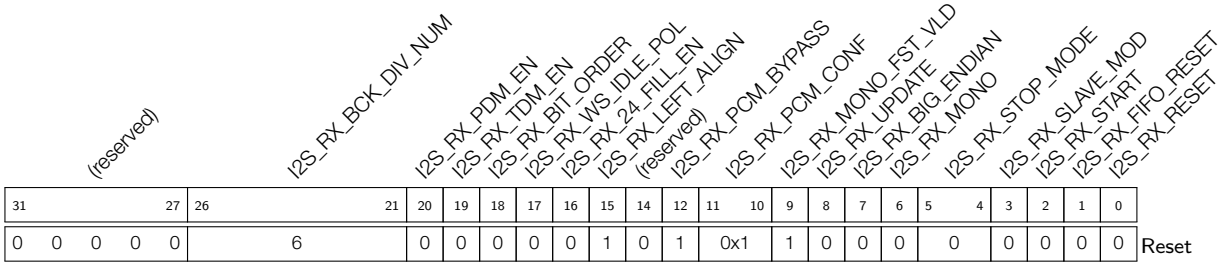
I2S_RX_DONE_INT_CLR 写 1 清除 I2S_RX_DONE_INT。(WT)

I2S_TX_DONE_INT_CLR 写 1 清除 I2S_TX_DONE_INT。(WT)

I2S_RX_HUNG_INT_CLR 写 1 清除 I2S_RX_HUNG_INT。(WT)

I2S_TX_HUNG_INT_CLR 写 1 清除 I2S_TX_HUNG_INT。(WT)

Register 28.5. I2S_RX_CONF_REG (0x0020)



I2S_RX_RESET 配置是否复位接收单元。

0：无效

1：复位

(WT)

I2S_RX_FIFO_RESET 配置是否复位 RX FIFO。

0：无效

1：复位

(WT)

I2S_RX_START 配置是否开始接收数据。

0：无效

1：开始

(R/W/SC)

I2S_RX_SLAVE_MOD 配置是否使能从机接收模式。

0：禁用

1：使能

(R/W)

I2S_RX_STOP_MODE 配置 I2S RX 何时暂停接收。

0：只有 I2S_RX_START 被清除时暂停接收

1：I2S_RX_START 被清除，或者接收字节数大于 I2S_RX_EOF_NUM_REG 配置的接收长度值时暂停接收

2：接收单元会在 I2S_RX_START 被清除，或者 DMA RX FIFO 满时暂停接收

(R/W)

I2S_RX_MONO 配置是否使能接收单元的单声道模式。

0：禁用

1：使能

(R/W)

I2S_RX_BIG_ENDIAN 配置 I2S RX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

见下页...

Register 28.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_UPDATE 配置是否将 I2S RX 寄存器从 APB 时钟域同步到 I2S RX 时钟域。

0：无效

1：同步

寄存器完成更新后，该位将由硬件清除。

(R/W/SC)

I2S_RX_MONO_FST_VLD 配置 I2S RX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

I2S_RX_PCM_CONF 配置 I2S RX 压缩/解压缩模式。

0 (atol)：A 率解压缩

1 (ltoa)：A 率压缩

2 (utol)：μ 率解压缩

3 (ltou)：μ 率压缩

(R/W)

I2S_RX_PCM_BYPASS 配置接收数据是否将绕过压缩/解压缩单元。

0：无效

1：绕过压缩/解压缩单元

(R/W)

I2S_RX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

I2S_RX_LEFT_ALIGN 配置 I2S RX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

I2S_RX_24_FILL_EN 配置 24 位通道数据位的储存数据位数。

0：将 24 位通道数据位保存到 24 位储存数据

1：将 24 位通道数据位保存到 32 位储存数据（多余的位填充为 0）

(R/W)

I2S_RX_WS_IDLE_POL 配置 WS 与接收数据通道的关系。

0：WS 为 0 时，接收左通道数据；WS 为 1 时，接收右通道数据

1：WS 为 1 时，接收左通道数据；WS 为 0 时，接收右通道数据

(R/W)

见下页...

Register 28.5. I2S_RX_CONF_REG (0x0020)

接上页...

I2S_RX_BIT_ORDER 配置 I2S RX 位顺序。

0：大端序，先接收最高位

1：小端序，先接收最低位

(R/W)

I2S_RX_TDM_EN 配置是否使能 I2S TDM RX 模式。

0：禁用

1：使能

(R/W)

I2S_RX_PDM_EN 配置是否使能 I2S PDM RX 模式。

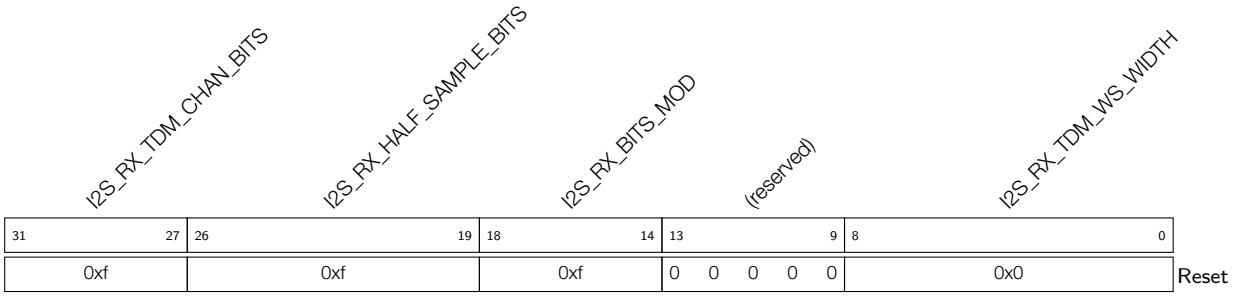
0：禁用

1：使能

(R/W)

I2S_RX_BCK_DIV_NUM 配置在 RX 模式下 BCK 时钟的分频系数。注意，不可配置为 1。(R/W)

Register 28.6. I2S_RX_CONF1_REG (0x0028)



I2S_RX_TDM_WS_WIDTH 配置 rx_ws_out（WS 默认电平）空闲时在 TDM 模式下的时长。rx_ws_out 空闲时在 TDM 模式下的时长 = (I2S_RX_TDM_WS_WIDTH[8:0] +1) x T_BCK。(R/W)

I2S_RX_BITS_MOD 配置在 RX 模式下接收通道的有效数据位长度。

7：所有有效的通道数据均为 8 位模式

15：所有有效的通道数据均为 16 位模式

23：所有有效的通道数据均为 24 位模式

31：所有有效的通道数据均为 32 位模式

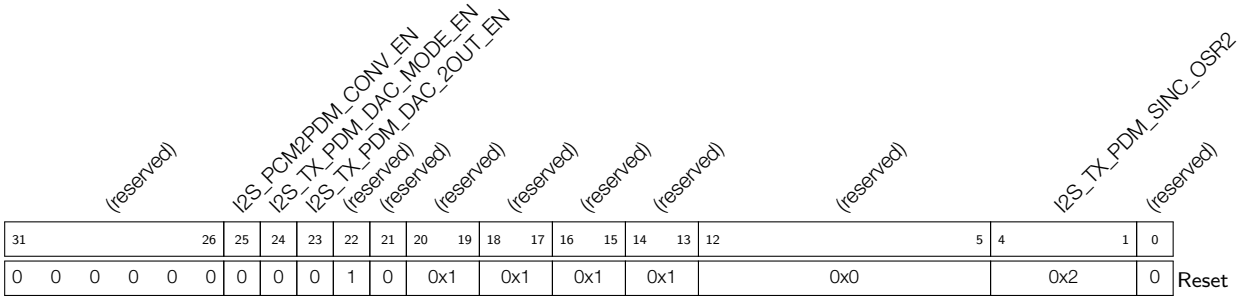
其他值无效。

(R/W)

I2S_RX_HALF_SAMPLE_BITS 配置 I2S RX 单次采样比特数。一个 WS 信号中 BCK 持续的周期长 = I2S_RX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_RX_TDM_CHAN_BITS 配置在 TDM RX 模式下每个通道的 RX 数据位长。RX 数据位长 = I2S_RX_TDM_CHAN_BITS + 1。(R/W)

Register 28.7. I2S_TX_PCM2PDM_CONF_REG (0x0040)



I2S_TX_PDM_SINC_OSR2 配置 I2S TX PDM OSR 的值。(R/W)

I2S_TX_PDM_DAC_2OUT_EN 配置 DAC 输出模式。

0：使能 1-line DAC 输出模式

1：使能 2-line DAC 输出模式

仅在 I2S_TX_PDM_DAC_MODE_EN 为 1 时有效。

(R/W)

I2S_TX_PDM_DAC_MODE_EN 配置使能 1-line PDM 输出模式或 DAC 输出模式。

0：使能 1-line PDM 输出模式

1：使能 DAC 输出模式

(R/W)

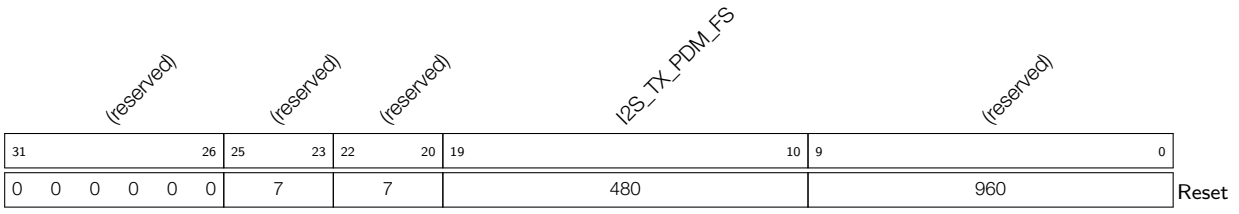
I2S_PCM2PDM_CONV_EN 配置是否使能 I2S TX PCM-to-PDM 转换器。

0: 禁用

1: 使能

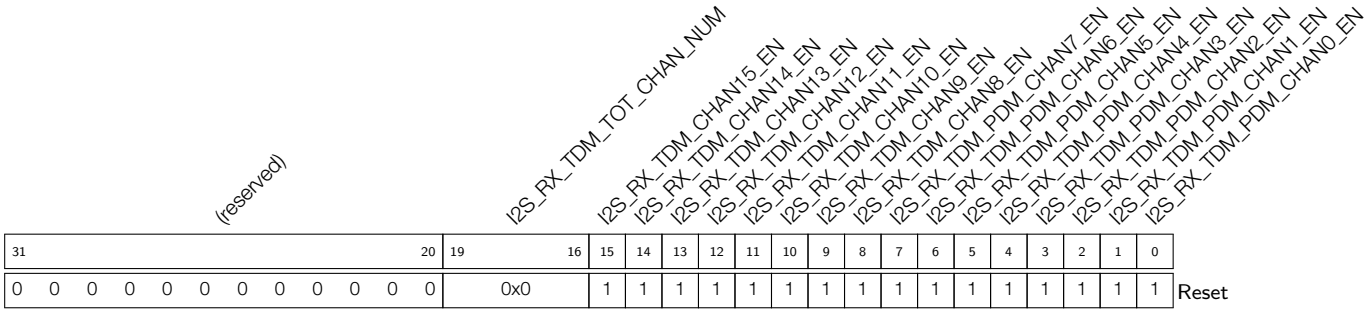
(R/W)

Register 28.8. I2S_TX_PCM2PDM_CONF1_REG (0x0044)



I2S_TX_PDM_FS 配置 I2S TX PDM 上采样率。(R/W)

Register 28.9. I2S_RX_TDM_CTRL_REG (0x0050)



I2S_RX_TDM_PDM_CHAN*n*_EN (*n*: 0-7) 配置是否使能 I2S TDM 或 PDM RX 通道 *n* 的有效输入数据。

0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

I2S_RX_TDM_CHAN*n*_EN (*n*: 8-15) 配置是否使能 I2S TDM RX 通道 *n* 的有效输入数据。

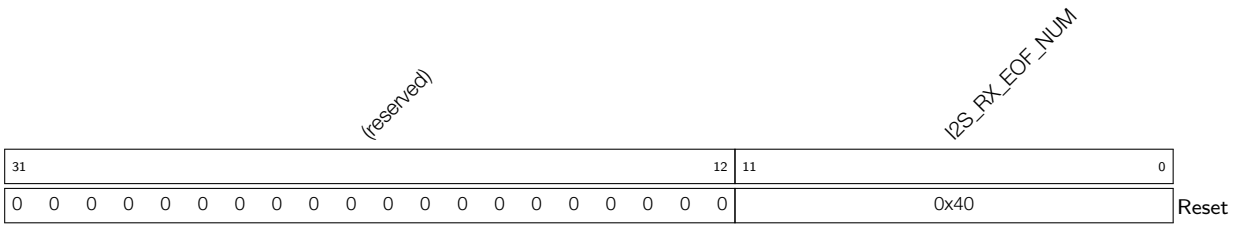
0：禁用该功能，在该通道中仅输入 0

1：使能

(R/W)

I2S_RX_TDM_TOT_CHAN_NUM 配置在 I2S TDM RX 模式下使用的通道总数。使用的通道总数 = I2S_RX_TDM_TOT_CHAN_NUM + 1。(R/W)

Register 28.10. I2S_RXEOF_NUM_REG (0x0064)



I2S_RX_EOF_NUM 配置接收数据的长度。接收数据长度 = (I2S_RX_BITS_MOD + 1) x (I2S_RX_EOF_NUM + 1)。如果接收数据的长度达到该值，则将在已配置的 DMA RX 通道中触发 GDMA_IN_SUC_EOF_CH*n*_INT 中断。(R/W)

Register 28.11. I2S_TX_CONF_REG (0x0024)

I2S_TX_RESET 配置是否复位发送单元。

0：无效

1：复位

(WT)

I2S_TX_FIFO_RESET 配置是否复位 TX FIFO。

0：无效

1：复位

(WT)

I2S_TX_START 配置是否开始发送数据。

0：无效

1：开始发送

(R/W/SC)

I2S_TX_SLAVE_MOD 配置是否使能从机发送模式。

0：禁用

1：使能

(R/W)

I2S_TX_STOP_EN 配置当 TX FIFO 为空时，发送单元是否停止输出 BCK 和 WS 信号。

0：无效

1：停止

(R/W)

I2S_TX_CHAN_EQUAL 配置在 I2S TX 单声道模式或 TDM 模式下，是否使能左声道数据等于右声道数据。

0：在 I2S TX 单声道模式或 TDM 模式下，I2S_SINGLE_DATA 为无效的通道数据

1：在 I2S TX 单声道模式或 TDM 模式下，左声道数据等于右声道数据

(R/W)

I2S_TX_MONO 配置是否使能发送单元的单声道模式。

0：禁用

1：使能

(R/W)

I2S_TX_BIG_ENDIAN 配置 I2S TX 字节序。

0：低字节数据写入低位地址

1：低字节数据写入高位地址

(R/W)

见下页...

Register 28.11. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_UPDATE 配置是否将 I2S TX 寄存器从 APB 时钟域同步到 I2S TX 时钟域。

0：无效

1：同步

寄存器更新完成后，此位将由硬件清除。

(R/W/SC)

I2S_TX_MONO_FST_VLD 配置在 I2S TX 单声道模式下的有效数据通道。

0：第二个通道数据有效

1：第一个通道数据有效

(R/W)

I2S_TX_PCM_CONF 配置 I2S TX 压缩/解压缩模式。

0 (atol)：A 率解压缩

1 (ltoa)：A 率压缩

2 (utol)：μ 率解压缩

3 (ltou)：u 率压缩

(R/W)

I2S_TX_PCM_BYPASS 配置是否发送数据将绕过压缩/解压缩单元。

0：无效

1：绕过

(R/W)

I2S_TX_MSB_SHIFT 配置 WS 和数据的 MSB 位之间的时序关系。

0：上升沿对齐

1：相隔一个周期

(R/W)

I2S_TX_BCK_NO_DLY 配置在主机模式下，BCK 上升沿和下降沿是否有延迟。

0：有延迟

1：无延迟

(R/W)

I2S_TX_LEFT_ALIGN 配置 I2S TX 对齐模式。

0：右对齐模式

1：左对齐模式

(R/W)

I2S_TX_24_FILL_EN 配置 24 位数据的发送格式。

0：将 24 位数据以 24 位的格式发送出去

1：将 24 位数据以 32 位的格式发送出去（不足的位用 0 填充）

(R/W)

见下页...

Register 28.11. I2S_TX_CONF_REG (0x0024)

接上页...

I2S_TX_WS_IDLE_POL 配置 WS 与发送数据通道的关系。

0：WS 为 0 时，发送左通道数据；WS 为 1 时，发送右通道数据

1：WS 为 1 时，发送左通道数据；WS 为 0 时，发送右通道数据

I2S_TX_BIT_ORDER 配置 I2S TX 位顺序。

0：大端序，先发送最高位

1：小端序，先发送最低位

(R/W)

I2S_TX_TDM_EN 配置是否使能 I2S TDM TX 模式。

0：禁用

1：使能

(R/W)

I2S_TX_PDM_EN 配置是否使能 I2S PDM TX 模式。

0：禁用

1：使能

(R/W)

I2S_TX_BCK_DIV_NUM 配置在 TX 模式下 BCK 时钟的分频系数。注意，不可配置为 1。(R/W)

I2S_TX_CHAN_MOD 配置 I2S 发送单元通道，更多信息见表 28-6。(R/W)

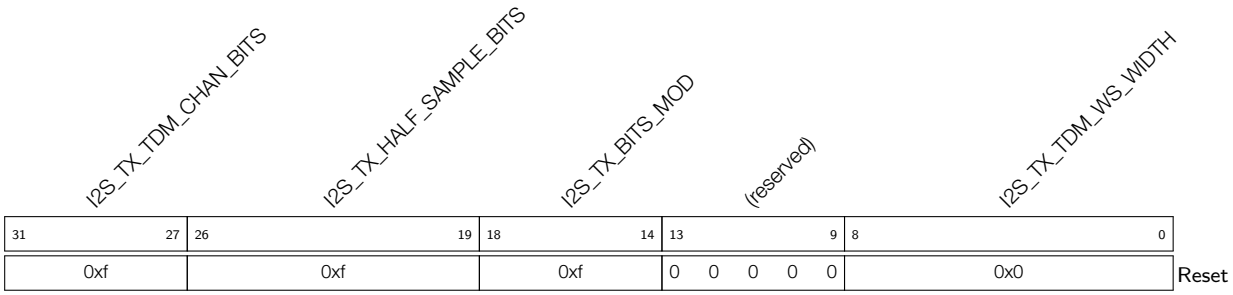
I2S_SIG_LOOPBACK 配置是否使能发送单元和接收单元共享 WS 和 BCK 信号。

0：禁用

1：使能

(R/W)

Register 28.12. I2S_TX_CONF1_REG (0x002C)



I2S_TX_TDM_WS_WIDTH 配置 tx_ws_out（WS 默认电平）空闲时在 TDM 模式下的时长。tx_ws_out 空闲时在 TDM 模式下的时长 = (I2S_TX_TDM_WS_WIDTH[8:0] +1) x T_BCK。(R/W)

I2S_TX_BITS_MOD 配置 TX 模式下发送通道的有效数据位长度。

7：所有有效的通道数据均为 8 位模式

15：所有有效的通道数据均为 16 位模式

23：所有有效的通道数据均为 24 位模式

31：所有有效的通道数据均为 32 位模式

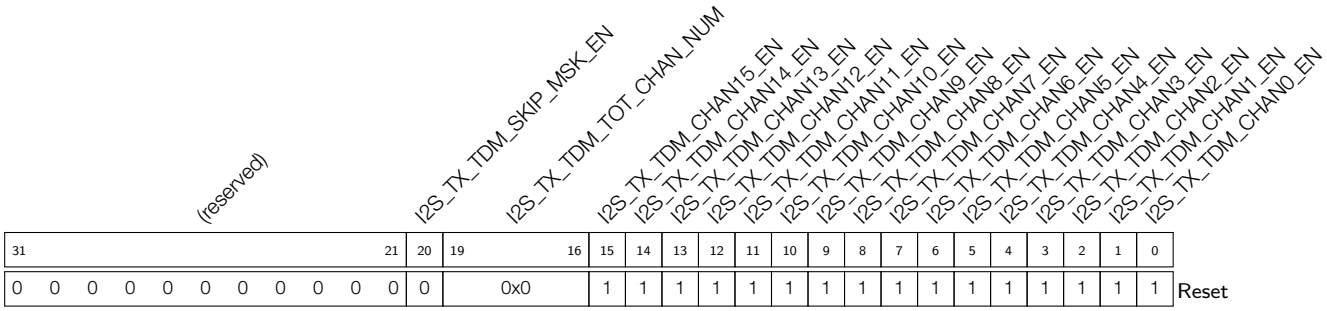
其他值无效。

(R/W)

I2S_TX_HALF_SAMPLE_BITS 配置 TX 单次采样比特数。一个 WS 信号中 BCK 持续的周期长 = I2S_TX_HALF_SAMPLE_BITS x 2。(R/W)

I2S_TX_TDM_CHAN_BITS 配置在 TDM TX 模式下每个通道的 TX 数据位。TX 数据位 = I2S_TX_TDM_CHAN_BITS + 1。(R/W)

Register 28.13. I2S_TX_TDM_CTRL_REG (0x0054)



I2S_TX_TDM_CHAN*n*_EN (*n*: 0-15) 配置是否使能 I2S TDM TX 通道 *n* 的有效输出数据。

0：通道发送数据由 I2S_TX_CHAN_EQUAL 和 I2S_SINGLE_DATA 控制，参考章节 28.9.2.1

1：使能

(R/W)

I2S_TX_TDM_TOT_CHAN_NUM 配置在 I2S TDM TX 模式下使用的通道总数。通道总数 = I2S_TX_TDM_TOT_CHAN_NUM + 1。(R/W)

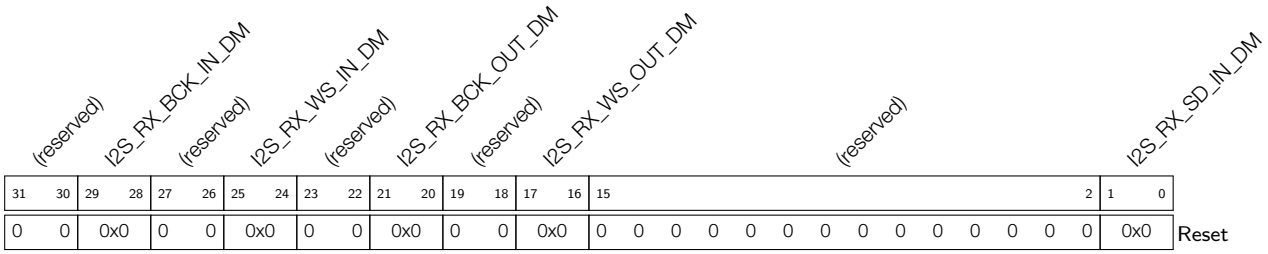
I2S_TX_TDM_SKIP_MSK_EN 配置 DMA TX buffer 的发送数据。

0：DMA TX buffer 中的数据都用于已启用的通道，未启用的通道发送时不会读取 DMA TX buffer 中的数据

1：DMA TX buffer 中的数据在所有通道发送时均会被读取，因此在未启用的通道发送时 DMA TX buffer 中的数据会被跳过

(R/W)

Register 28.14. I2S_RX_TIMING_REG (0x0058)



I2S_RX_SD_IN_DM 配置 I2S RX SD 输入信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：不使用该功能

(R/W)

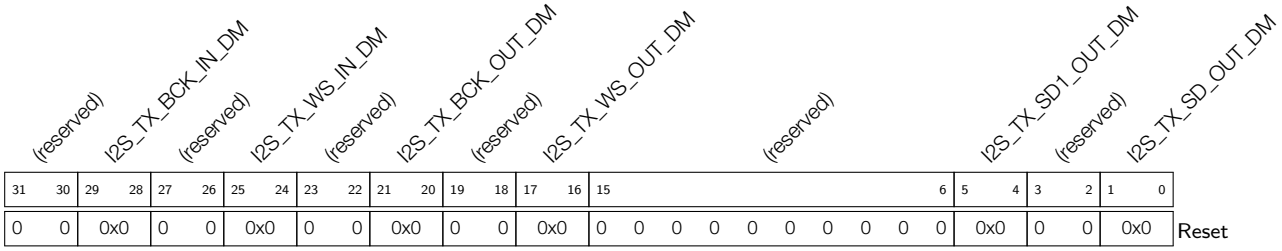
I2S_RX_WS_OUT_DM 配置 I2S RX WS 输出信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_BCK_OUT_DM 配置 I2S RX BCK 输出信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_WS_IN_DM 配置 I2S RX WS 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

I2S_RX_BCK_IN_DM 配置 I2S RX BCK 输入信号的延迟模式。具体配置值请参考 I2S_RX_SD_IN_DM。(R/W)

Register 28.15. I2S_TX_TIMING_REG (0x005C)



I2S_TX_SD_OUT_DM 配置 I2S TX SD 输出信号的延迟模式。

0：旁路

1：上升沿延迟

2：下降沿延迟

3：不使用该功能

(R/W)

I2S_TX_SD1_OUT_DM 配置 I2S TX SD1 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

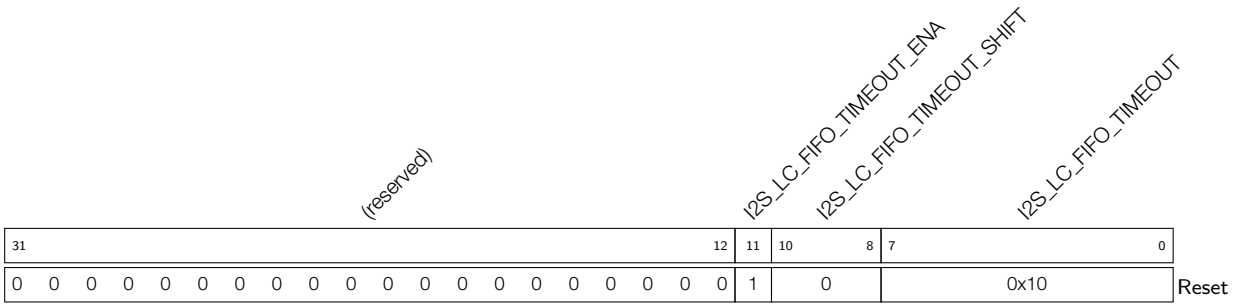
I2S_TX_WS_OUT_DM 配置 I2S TX WS 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_OUT_DM 配置 I2S TX BCK 输出信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_WS_IN_DM 配置 I2S TX WS 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

I2S_TX_BCK_IN_DM 配置 I2S TX BCK 输入信号的延迟模式。具体配置值请参考 I2S_TX_SD_OUT_DM。(R/W)

Register 28.16. I2S_LC_HUNG_CONF_REG (0x0060)



I2S_LC_FIFO_TIMEOUT 配置 FIFO 的超时阈值。FIFO Hung 计数器等于该值时，将触发 I2S_TX_HUNG_INT 中断或 I2S_RX_HUNG_INT 中断。(R/W)

I2S_LC_FIFO_TIMEOUT_SHIFT 配置分频滴答计数器的阈值。计数器值大于等于 $88000/2^{I2S\_LC\_FIFO\_TIMEOUT\_SHIFT}$ 时，复位滴答计数器。(R/W)

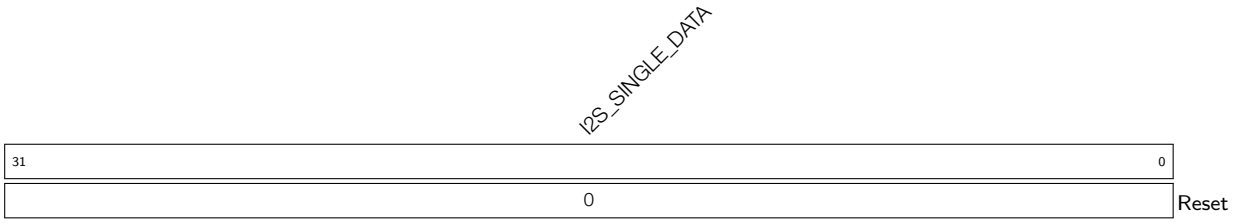
I2S_LC_FIFO_TIMEOUT_ENA 配置是否使能 FIFO 超时。

0：禁用

1：使能

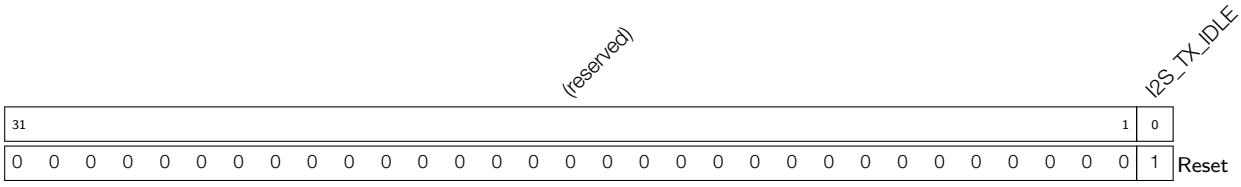
(R/W)

Register 28.17. I2S_CONF_SIGLE_DATA_REG (0x0068)



I2S_SINGLE_DATA 配置用于发送的通道常量数据。(R/W)

Register 28.18. I2S_STATE_REG (0x006C)



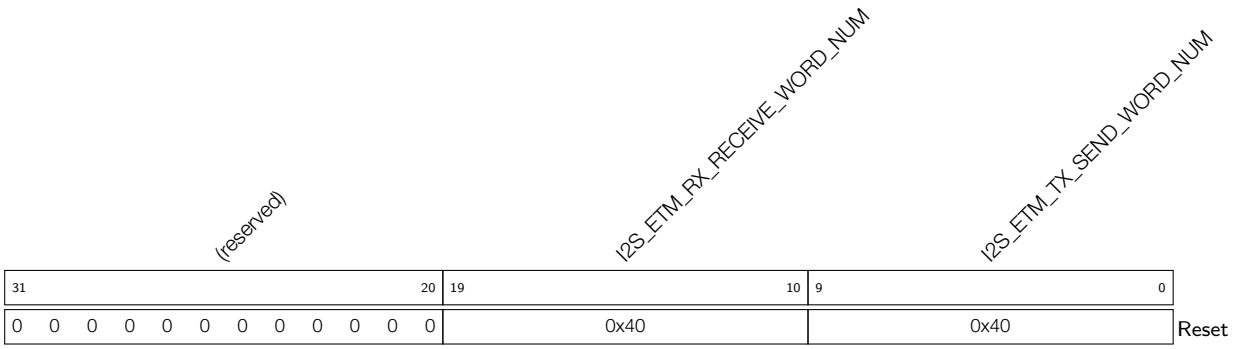
I2S_TX_IDLE 表示 I2S TX 单元的工作状态。

0：I2S TX 单元处于工作状态

1：I2S TX 单元处于空闲状态

(RO)

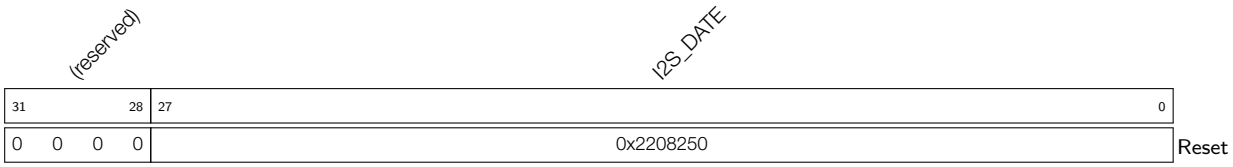
Register 28.19. I2S_ETM_CONF_REG (0x0070)



I2S_ETM_TX_SEND_WORD_NUM 配置触发 I2S_TX_X_WORDS_SENT 事件的阈值。当发送字数为 I2S_ETM_TX_SEND_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

I2S_ETM_RX_RECEIVE_WORD_NUM 配置触发 ETM I2S_RX_X_WORDS_RECEIVED 事件的阈值。当接收字数为 I2S_ETM_RX_RECEIVE_WORD_NUM [9:0] 时，I2S 将触发对应的 ETM 事件。(R/W)

Register 28.20. I2S_DATE_REG (0x0080)



I2S_DATE 版本控制寄存器。(R/W)

29 脉冲计数控制器 (PCNT)

脉冲计数控制器 (Pulse Count Controller, PCNT) 用于对输入脉冲计数，通过记录输入脉冲信号的上升沿或下降沿进行递增或递减计数。PCNT 有四个称为“单元”的独立脉冲计数控制器，这些单元拥有自己的寄存器。PCNT 模块仅有一个时钟，为 APB_CLK。下文描述中 *n* 表示单元编号 0 ~ 3。

每个单元有两个通道（ch0 和 ch1），可以独立配置为递增或递减计数。两个通道功能相同，下文以通道 0 (ch0) 为例进行介绍。

如图 29-1 所示，每个通道有两个输入信号：

1. 一个脉冲输入信号（如 sig_ch0_u*n* 为单元 *n* ch0 的脉冲输入信号）
2. 一个控制信号（如 ctrl_ch0_u*n* 为单元 *n* ch0 的控制信号）

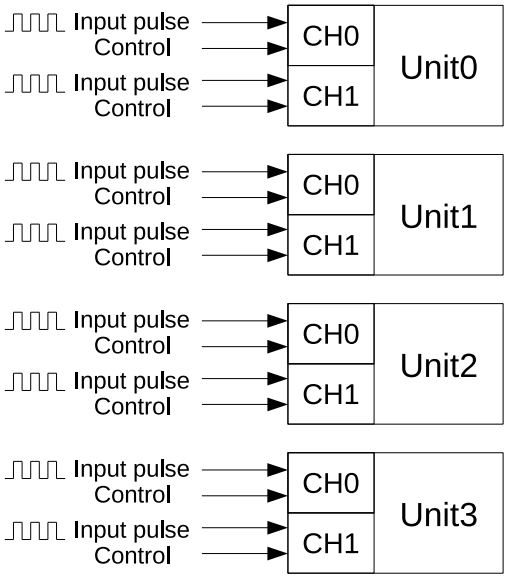


图 29-1. PCNT 框图

29.1 主要特性

PCNT 有如下特性：

- 四个脉冲计数控制器（单元），各自独立工作，计数范围是 1 ~ 65535
- 每个单元有两个独立的通道，共用一个脉冲计数控制器
- 所有通道均有输入脉冲信号（如 sig_ch0_u*n*）和相应的控制信号（如 ctrl_ch0_u*n*）
- 滤波器独立工作，过滤每个单元输入脉冲信号（sig_ch0_u*n* 和 sig_ch1_u*n*）控制信号（ctrl_ch0_u*n* 和 ctrl_ch1_u*n*）的毛刺
- 每个通道参数如下：
  1. 选择在输入脉冲信号的上升沿或下降沿计数
  2. 在控制信号为高电平或低电平时可将计数模式配置为递增、递减或停止计数
  3. 五个计数器观察点
- 最大脉冲频率：$\frac{f_{APB\_CLK}}{2}$

29.2 功能描述

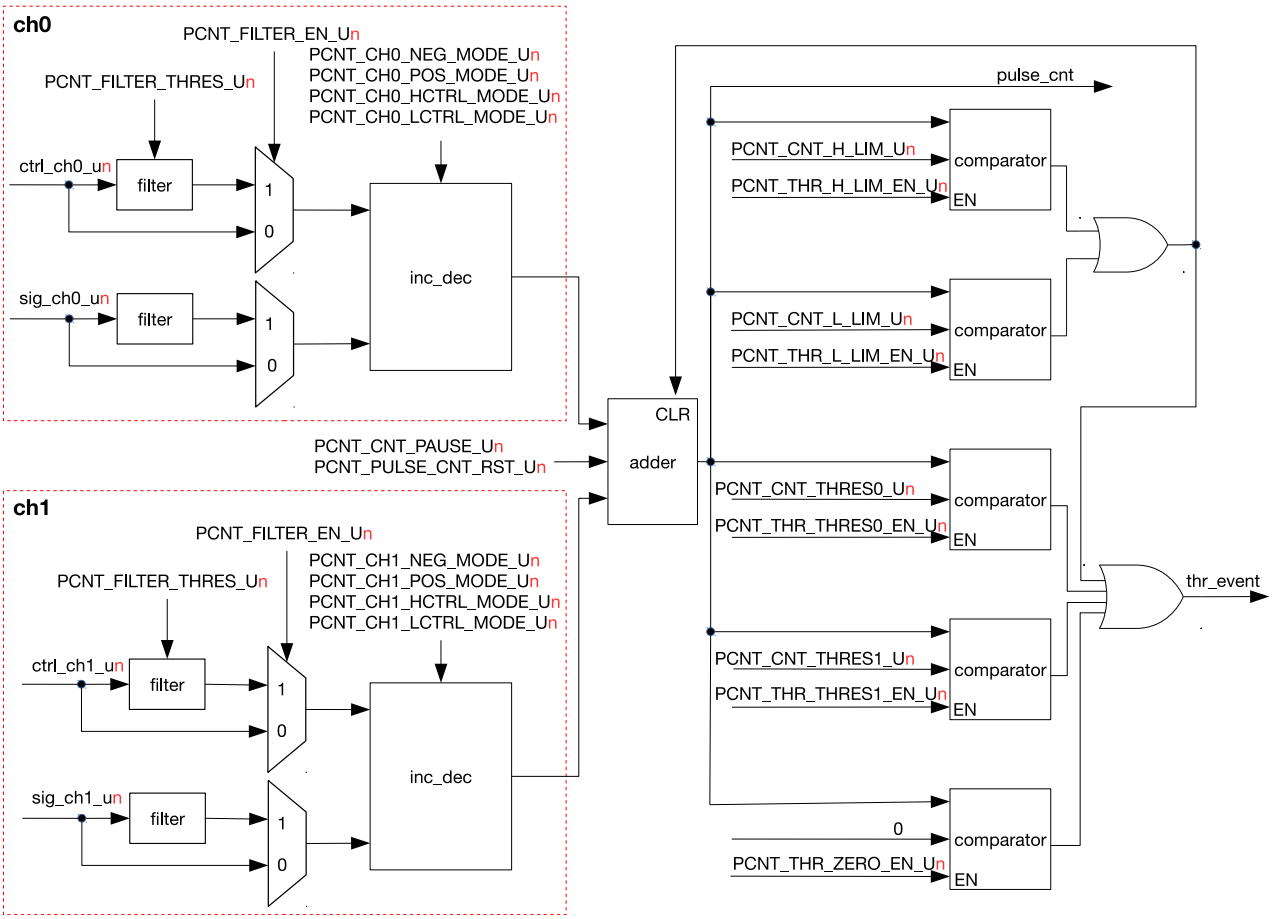


图 29-2. PCNT 单元基本架构图

图 29-2 为 PCNT 单元的基本架构图。如上所述，ctrl_ch0_u*n* 为单元 *n* ch0 的控制信号，控制信号 ctrl_ch0_u*n* 为高电平或低电平时可配置不同的计数模式，在输入脉冲信号 sig_ch0_u*n* 的上升沿和下降沿计数。可选计数模式如下：

- 递增模式：通道检测到 sig_ch0_u*n* 的有效边沿（有效边沿软件可配）时，计数器的值 pulse_cnt 加 1。pulse_cnt 的值达到 PCNT_CNT_H_LIM_U*n* 时被清零。如果在 pulse_cnt 达到 PCNT_CNT_H_LIM_U*n* 前，该通道的计数模式改变或 PCNT_CNT_PAUSE_U*n* 置 1，则 pulse_cnt 停止计数，计数模式改变。
- 递减模式：通道检测到 sig_ch0_u*n* 的有效边沿（有效边沿软件可配）时，计数器的值 pulse_cnt 减 1。pulse_cnt 的值达到 PCNT_CNT_L_LIM_U*n* 时被清零。如果在 pulse_cnt 达到 PCNT_CNT_L_LIM_U*n* 前，该通道的计数模式改变或 PCNT_CNT_PAUSE_U*n* 置 1，则 pulse_cnt 停止计数，计数模式改变。
- 停止计数：计数停止，计数器的值 pulse_cnt 保持不变。

表 29-1 至表 29-4 说明了如何配置通道 0 的计数模式。

表 29-1. 控制信号为低电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 29-2. 控制信号为高电平时输入脉冲信号上升沿的计数模式

PCNT_CH0_POS_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 29-3. 控制信号为低电平时输入脉冲信号下降沿的计数模式

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_LCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

表 29-4. 控制信号为高电平时输入脉冲信号下降沿的计数模式

PCNT_CH0_NEG_MODE_U*n*	PCNT_CH0_HCTRL_MODE_U*n*	计数模式
1	0	递增模式
	1	递减模式
	其他	停止计数
2	0	递减模式
	1	递增模式
	其他	停止计数
其他	N/A	停止计数

每个单元均有一个滤波器，用于该单元的所有控制信号和输入脉冲信号。置位 PCNT_FILTER_EN_U*n* 位使能滤

波器。滤波器监测信号，滤除脉冲宽度小于 PCNT_FILTER_THRES_U*n* 个 APB 时钟周期的线路毛刺。

如前文所述，每个单元有通道 0 和通道 1 两个通道，处理不同的输入脉冲信号，并通过各自的 inc_dec 模块递增或递减计数值。之后，两个通道将计数值发送给加法器模块，该模块有一个带符号位的 16 位宽寄存器。软件可以通过置位 PCNT_CNT_PAUSE_U*n* 暂停加法器，也可以通过置位 PCNT_PULSE_CNT_RST_U*n* 清零加法器。

PCNT 可以设置五个观察点，五个观察点共用一个中断，可以通过每个观察点各自的中断使能信号开启或屏蔽中断。

- 最大计数值：当 pulse_cnt 大于等于 PCNT_CNT_H_LIM_U*n* 时，产生上限中断，同时 PCNT_CNT_THR_H_LIM_LAT_U*n* 为高。
- 最小计数值：当 pulse_cnt 小于等于 PCNT_CNT_L_LIM_U*n* 时，产生下限中断，同时 PCNT_CNT_THR_L_LIM_LAT_U*n* 为高。
- 两个中间阈值：当 pulse_cnt 等于 PCNT_CNT_THRES0_U*n* 或者 PCNT_CNT_THRES1_U*n* 时，产生中断，同时 PCNT_CNT_THR_THRES0_LAT_U*n* 或 PCNT_CNT_THR_THRES1_LAT_U*n* 为高。
- 零：当 pulse_cnt 等于 0 时，产生中断，同时 PCNT_CNT_THR_ZERO_LAT_U*n* 有效。

在 PCNT 模块工作过程中，软件如果重新配置 PCNT_CNT_H_LIM_U*n* 和/或 PCNT_CNT_L_LIM_U*n*，则新的配置值会在 pluse_cnt 达到上述五个观察点的任意一个后再生效；软件如果重新配置 PCNT_CNT_THRES0_U*n* 和/或 PCNT_CNT_THRES1_U*n*，则新的配置值会立即生效。

29.3 应用实例

每个单元的通道 0 和通道 1 可配置为独立工作或一起工作。下文详细说明了通道 0 独自递增计数、通道 0 独自递减计数和两个通道一起递增计数的应用实例。本节中未详述的通道工作模式（如通道 1 独自递减或递增、双通道一增一减），可参考这三种模式。

29.3.1 通道 0 独自递增计数

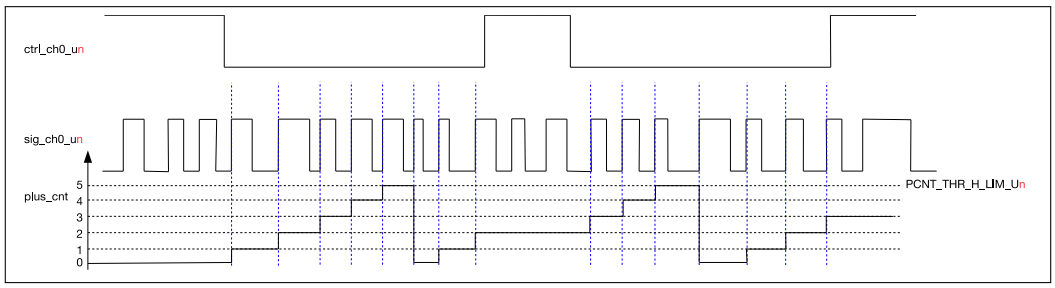


图 29-3. 通道 0 递增计数图

图 29-3 为通道 0 在 sig_ch0_u*n* 上升沿独立递增计数的示意图，此时通道 1 关闭（请参阅 29.2 一节查看如何关闭通道 1）。通道 0 的配置如下所示。

- PCNT_CH0_LCTRL_MODE_U*n*=0：当 ctrl_ch0_u*n* 为低电平时，递增计数。
- PCNT_CH0_HCTRL_MODE_U*n*=2：当 ctrl_ch0_u*n* 为高电平时，停止计数。
- PCNT_CH0_POS_MODE_U*n*=1：在 sig_ch0_u*n* 的上升沿递增计数。
- PCNT_CH0_NEG_MODE_U*n*=0：在 sig_ch0_u*n* 的下降沿不计数。
- PCNT_CNT_H_LIM_U*n*=5：pulse_cnt 的值递增至 PCNT_CNT_H_LIM_U*n* 时被清零。

29.3.2 通道 0 独自递减计数

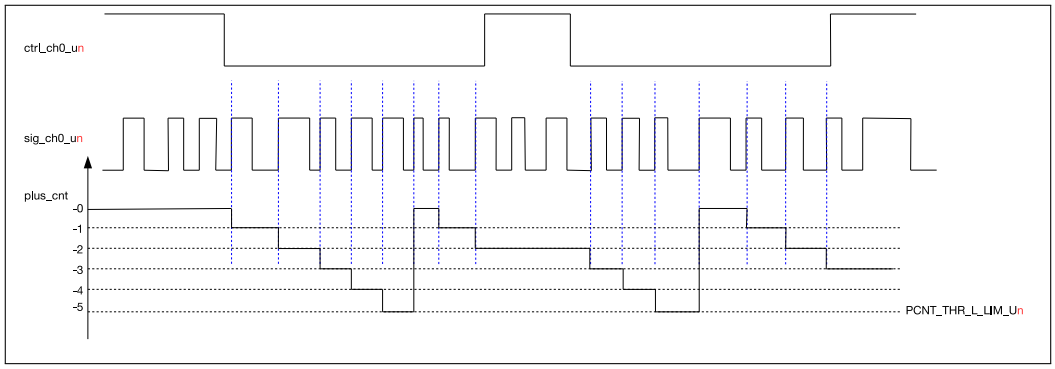


图 29-4. 通道 0 递减计数图

图 29-4 为通道 0 在 sig_ch0_u*n* 上升沿独立递减计数的示意图，此时通道 1 关闭。此时通道 0 的配置与图 29-3 相比有如下区别：

- PCNT_CH0_POS_MODE_U*n*=2：即在 sig_ch0_u*n* 的上升沿递减计数。
- PCNT_CNT_L_LIM_U*n*=-5：pulse_cnt 的值递减到 PCNT_CNT_L_LIM_U*n* 时被清零。

29.3.3 通道 0 和通道 1 同时递增计数

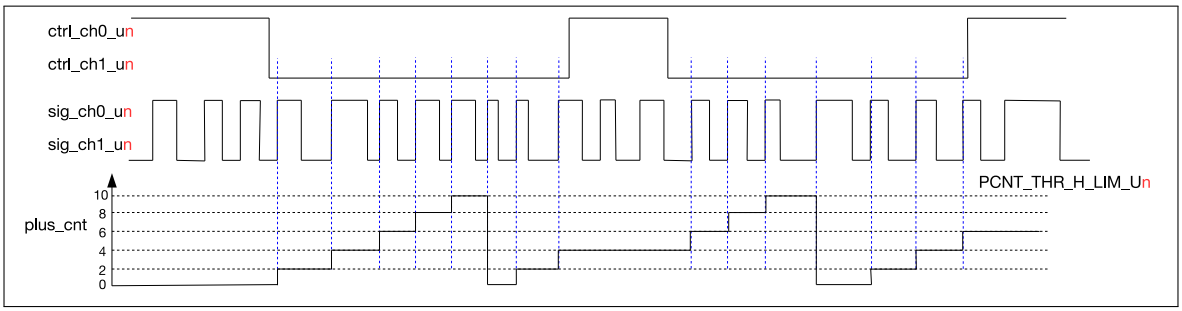


图 29-5. 双通道递增计数图

图 29-5 为通道 0 和通道 1 分别在 sig_ch0_u*n* 和 sig_ch1_u*n* 上升沿同时递增计数的示意图。如图 29-5 所示，控制信号 ctrl_ch0_u*n* 与 ctrl_ch1_u*n* 的波形一致，输入脉冲信号 sig_ch0_u*n* 和 sig_ch1_u*n* 波形一致。具体配置如下：

- 通道 0：
  - PCNT_CH0_LCTRL_MODE_U*n*=0：当 ctrl_ch0_u*n* 为低电平时，递增计数。
  - PCNT_CH0_HCTRL_MODE_U*n*=2：当 ctrl_ch0_u*n* 为高电平时，停止计数。
  - PCNT_CH0_POS_MODE_U*n*=1：在 sig_ch0_u*n* 的上升沿递增计数。
  - PCNT_CH0_NEG_MODE_U*n*=0：在 sig_ch0_u*n* 的下降沿不计数。
- 通道 1：
  - PCNT_CH1_LCTRL_MODE_U*n*=0：当 ctrl_ch1_u*n* 为低电平时，递增计数。
  - PCNT_CH1_HCTRL_MODE_U*n*=2：当 ctrl_ch1_u*n* 为高电平时，停止计数。
  - PCNT_CH1_POS_MODE_U*n*=1：在 sig_ch1_u*n* 的上升沿递增计数。

  - PCNT_CH1_NEG_MODE_U*n*=0：在 sig_ch1_u*n* 的下降沿不计数。

- PCNT_CNT_H_LIM_U*n*=10：pulse_cnt 递增至 PCNT_CNT_H_LIM_U*n* 时被清零。

29.4 寄存器列表

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

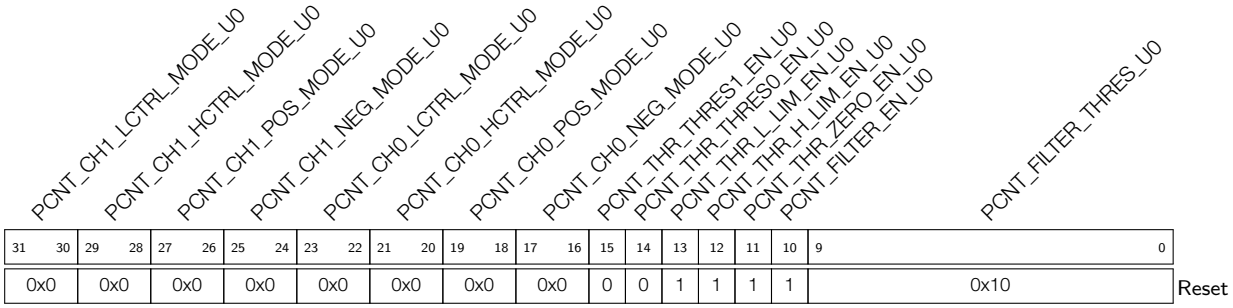
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
PCNT_U0_CONF0_REG	单元 0 的配置寄存器 0	0x0000	R/W
PCNT_U0_CONF1_REG	单元 0 的配置寄存器 1	0x0004	R/W
PCNT_U0_CONF2_REG	单元 0 的配置寄存器 2	0x0008	R/W
PCNT_U1_CONF0_REG	单元 1 的配置寄存器 0	0x000C	R/W
PCNT_U1_CONF1_REG	单元 1 的配置寄存器 1	0x0010	R/W
PCNT_U1_CONF2_REG	单元 1 的配置寄存器 2	0x0014	R/W
PCNT_U2_CONF0_REG	单元 2 的配置寄存器 0	0x0018	R/W
PCNT_U2_CONF1_REG	单元 2 的配置寄存器 1	0x001C	R/W
PCNT_U2_CONF2_REG	单元 2 的配置寄存器 2	0x0020	R/W
PCNT_U3_CONF0_REG	单元 3 的配置寄存器 0	0x0024	R/W
PCNT_U3_CONF1_REG	单元 3 的配置寄存器 1	0x0028	R/W
PCNT_U3_CONF2_REG	单元 3 的配置寄存器 2	0x002C	R/W
PCNT_CTRL_REG	所有计数器的控制寄存器	0x0060	R/W
状态寄存器			
PCNT_U0_CNT_REG	单元 0 的计数器值	0x0030	RO
PCNT_U1_CNT_REG	单元 1 的计数器值	0x0034	RO
PCNT_U2_CNT_REG	单元 2 的计数器值	0x0038	RO
PCNT_U3_CNT_REG	单元 3 的计数器值	0x003C	RO
PCNT_U0_STATUS_REG	脉冲计数器单元 0 的状态寄存器	0x0050	RO
PCNT_U1_STATUS_REG	脉冲计数器单元 1 的状态寄存器	0x0054	RO
PCNT_U2_STATUS_REG	脉冲计数器单元 2 的状态寄存器	0x0058	RO
PCNT_U3_STATUS_REG	脉冲计数器单元 3 的状态寄存器	0x005C	RO
中断寄存器			
PCNT_INT_RAW_REG	原始中断状态寄存器	0x0040	RO
PCNT_INT_ST_REG	中断状态寄存器	0x0044	RO
PCNT_INT_ENA_REG	中断使能寄存器	0x0048	R/W
PCNT_INT_CLR_REG	中断清除寄存器	0x004C	WO
版本寄存器			
PCNT_DATE_REG	脉冲计数器的版本控制寄存器	0x00FC	R/W

29.5 寄存器

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

Register 29.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xCn*)**



PCNT_FILTER_THRES_U*n* 配置滤波器的最大阈值。滤波器启动时，任何小于该值的脉冲都会被过滤。

单位：APB_CLK 时钟周期。

(R/W)

PCNT_FILTER_EN_U*n* 单元 *n* 输入滤波器的使能位。(R/W)

PCNT_THR_ZERO_EN_U*n* 单元 *n* 过零比较器的使能位。(R/W)

PCNT_THR_H_LIM_EN_U*n* 单元 *n* 上限比较器的使能位，控制上限中断事件的产生。(R/W)

PCNT_THR_L_LIM_EN_U*n* 单元 *n* 下限比较器的使能位，控制下限中断事件的产生。(R/W)

PCNT_THR_THRES0_EN_U*n* 单元 *n* 阈值 0 比较器的使能位。(R/W)

PCNT_THR_THRES1_EN_U*n* 单元 *n* 阈值 1 比较器的使能位。(R/W)

PCNT_CH0_NEG_MODE_U*n* 配置通道 0 输入信号检测下降沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH0_POS_MODE_U*n* 配置通道 0 输入信号检测上升沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

见下页...

Register 29.1. PCNT_U*n*_CONF0_REG (*n*: 0-3) (0x0000+0xCn*)**

接上页...

PCNT_CH0_HCTRL_MODE_U*n* 控制信号为高电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH0_LCTRL_MODE_U*n* 控制信号为低电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH1_NEG_MODE_U*n* 配置通道 1 输入信号检测下降沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH1_POS_MODE_U*n* 配置通道 1 输入信号检测上升沿的工作模式。

1：计数器递增

2：计数器递减

0、3：对计数器无任何影响。

(R/W)

PCNT_CH1_HCTRL_MODE_U*n* 控制信号为高电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

PCNT_CH1_LCTRL_MODE_U*n* 控制信号为低电平时，用于改变 CH*n*_POS_MODE 和 CH*n*_NEG_MODE 的设置。

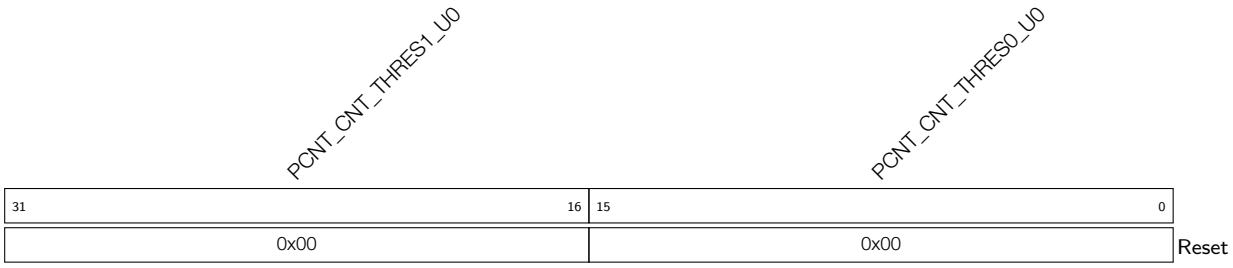
0：不做修改

1：反转（增加转为减少，减少转为增加）

2、3：禁止计数器修改。

(R/W)

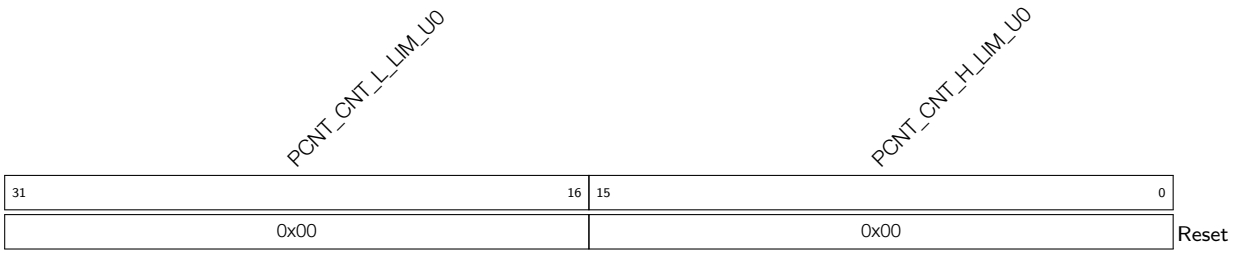
Register 29.2. PCNT_U*n*_CONF1_REG (*n*: 0-3) (0x0004+0xCn*)**



PCNT_CNT_THRES0_U*n* 配置单元 *n* 阈值 0 的值。(R/W)

PCNT_CNT_THRES1_U*n* 配置单元 *n* 阈值 1 的值。(R/W)

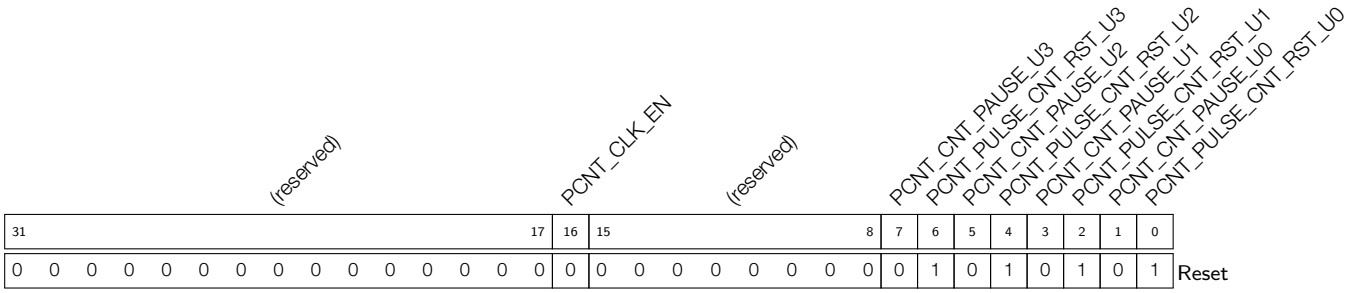
Register 29.3. PCNT_U*n*_CONF2_REG (*n*: 0-3) (0x0008+0xCn*)**



PCNT_CNT_H_LIM_U*n* 配置单元 *n* 的计数上限阈值。当脉冲计数达到上限阈值，计数会被清零。(R/W)

PCNT_CNT_L_LIM_U*n* 配置单元 *n* 的计数下限阈值。当脉冲计数达到下限阈值，计数会被清零。(R/W)

Register 29.4. PCNT_CTRL_REG (0x0060)



PCNT_PULSE_CNT_RST_U*n* 写 1 清零单元 *n* 的计数器。

0：无效

1：清零

(R/W)

PCNT_CNT_PAUSE_U*n* 写 1 暂停单元 *n* 的计数器。

0：无效

1：暂停

(R/W)

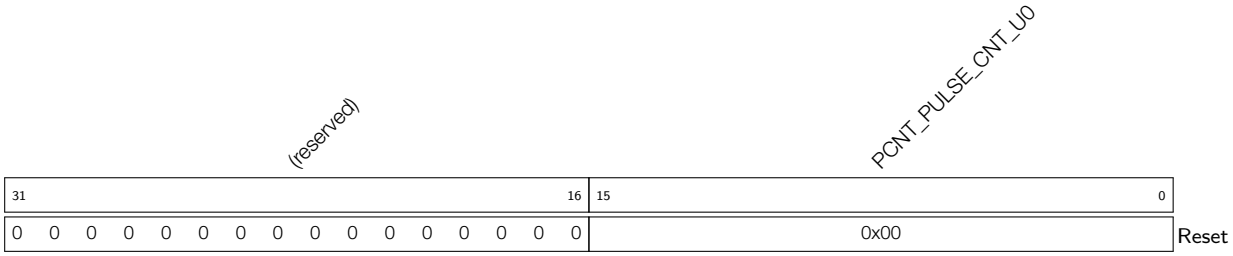
PCNT_CLK_EN 配置是否使能脉冲计数器模块寄存器时钟门控。

0：寄存器时钟仅在有读写操作时开启

1：寄存器时钟一直开启

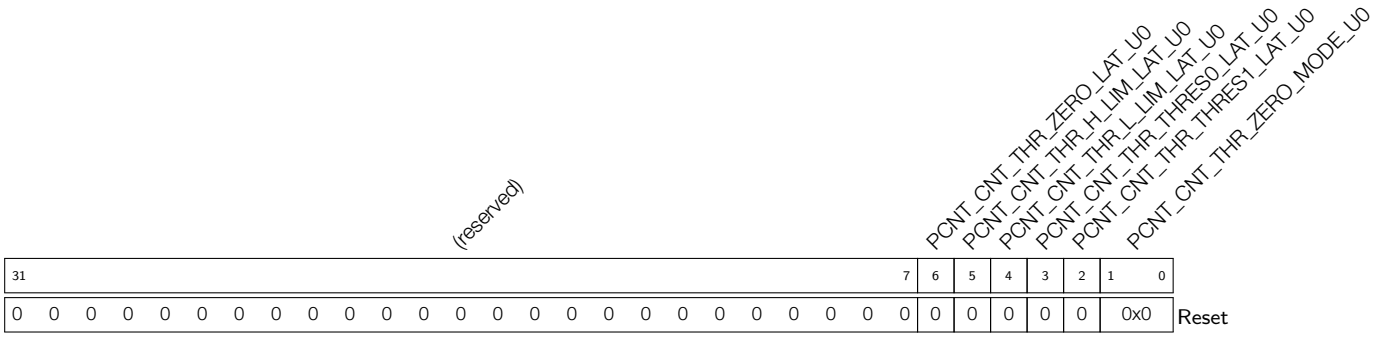
(R/W)

Register 29.5. PCNT_U*n*_CNT_REG (*n*: 0-3) (0x0030+0x4*n)



PCNT_PULSE_CNT_U*n* 表示单元 *n* 脉冲计数器的当前值。(RO)

Register 29.6. PCNT_U*n*_STATUS_REG (*n*: 0-3) (0x0050+0x4*n)



PCNT_CNT_THR_ZERO_MODE_U*n* 表示 PCNT_U*n* 为 0 时的脉冲计数器状态。

0：脉冲计数器的值由正数减至 0

1：脉冲计数器的值由负数增至 0

2：脉冲计数器为负

3：脉冲计数器为正

(RO)

PCNT_CNT_THR_THRES1_LAT_U*n* 表示阈值中断有效时，PCNT_U*n* 阈值 1 的锁存值。

0：其他

1：脉冲计数器的当前值与阈值 1 相等，阈值 1 有效

(RO)

PCNT_CNT_THR_THRES0_LAT_U*n* 表示阈值中断有效时，PCNT_U*n* 阈值 0 的锁存值。

0：其他

1：脉冲计数器的当前值与阈值 0 相等，阈值 0 有效

(RO)

PCNT_CNT_THR_L_LIM_LAT_U*n* 表示下限中断有效时，PCNT_U*n* 下限的锁存值。

0：其他

1：脉冲计数器的当前值与下限阈值相等，下限有效

(RO)

PCNT_CNT_THR_H_LIM_LAT_U*n* 表示上限中断有效时，PCNT_U*n* 上限的锁存值。

0：其他

1：脉冲计数器的当前值与上限阈值相等，上限有效

(RO)

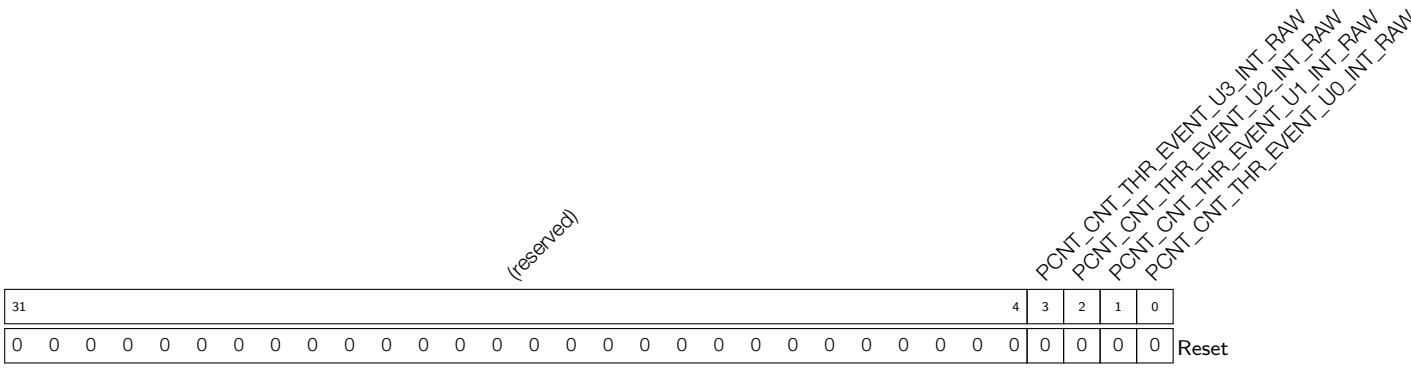
PCNT_CNT_THR_ZERO_LAT_U*n* 表示阈值中断有效时，PCNT_U*n* 阈值 0 的锁存值。

0：其他

1：脉冲计数器的当前值为 0，阈值 0 有效

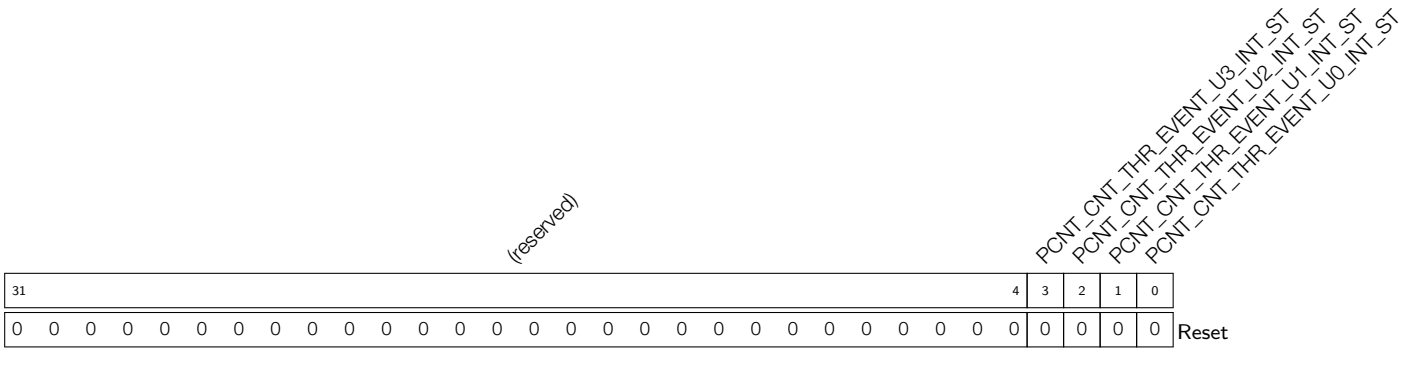
(RO)

Register 29.7. PCNT_INT_RAW_REG (0x0040)



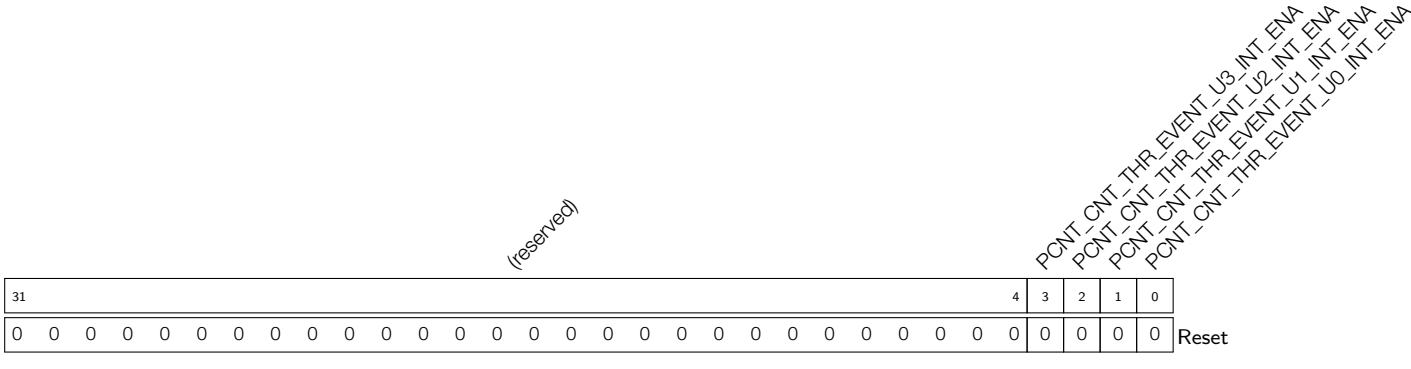
PCNT_CNT_THR_EVENT_U*n*_INT_RAW 单元 *n* 事件中断的原始中断状态。(RO)

Register 29.8. PCNT_INT_ST_REG (0x0044)



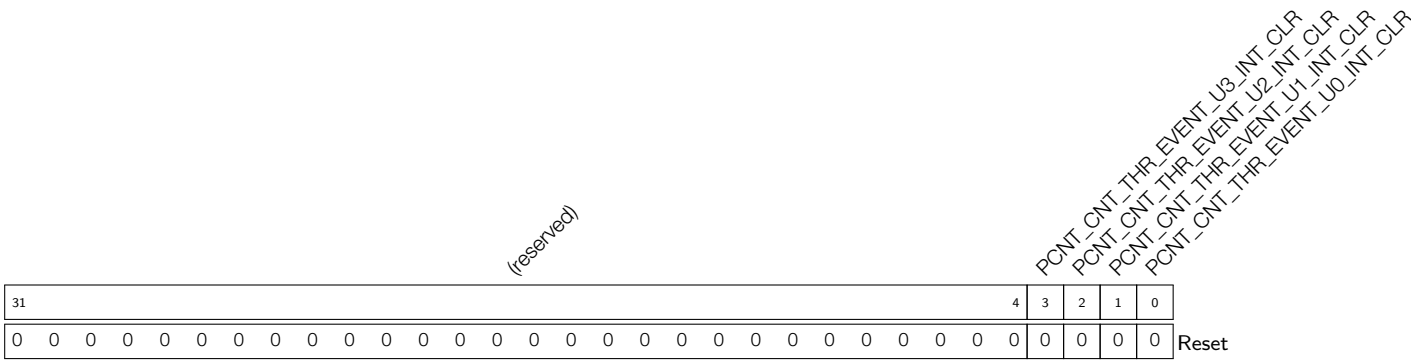
PCNT_CNT_THR_EVENT_U*n*_INT_ST 单元 *n* 事件中断的屏蔽中断状态。(RO)

Register 29.9. PCNT_INT_ENA_REG (0x0048)



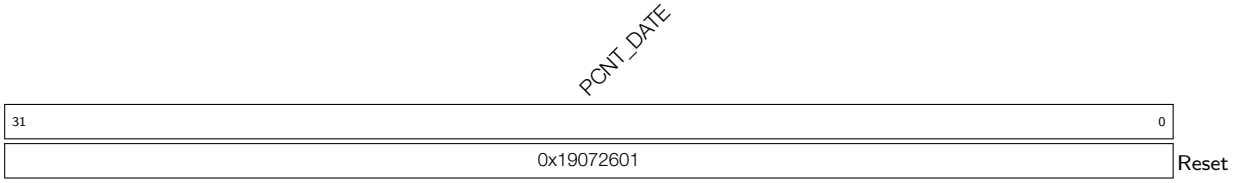
PCNT_CNT_THR_EVENT_U*n*_INT_ENA 写 1 使能单元 *n* 事件中断。(R/W)

Register 29.10. PCNT_INT_CLR_REG (0x004C)



PCNT_CNT_THR_EVENT_U*n*_INT_CLR 写 1 清除单元 *n* 事件中断。(WO)

Register 29.11. PCNT_DATE_REG (0x00FC)



PCNT_DATE 版本控制寄存器。(R/W)

30 USB 串口/JTAG 控制器 (USB_SERIAL_JTAG)

ESP32-H2 中包含一个 USB 串口/JTAG 控制器，可用于烧录芯片的外部 flash、读取程序输出的数据以及将调试器连接到正在运行的程序中。任何带有 USB 主机（下文将简称为“主机”）的计算机都可以实现上述功能，无需其他外部组件辅助。

30.1 概述

使用串口通信和 JTAG 调试端口通信来编程和调试 ESP32-H2 项目都是可行的，但也有其限制。首先，串口通信和 JTAG 调试端口通信都占用 IO 管脚，减少了软件中可用于控制外部信号的管脚数量。此外，串口通信和 JTAG 调试端口通信都需借助外部芯片或适配器与主机连接，因此，如果需要整合这两个功能，则需额外使用外部芯片或调试适配器。

为解决上述问题，ESP32-H2 提供了一个 USB 串口/JTAG 控制器，同时集成 USB-串口转换器和 USB-JTAG 适配器功能。由于该模块仅使用 USB2.0 所需的两条数据线直接连接外部 USB 主机，因此 ESP32-H2 仅需占用两个管脚用于调试。

30.2 特性

USB 串口/JTAG 控制器具有如下特性：

- 支持 USB 全速标准；集成 CDC-ACM （通信设备类抽象控制模型）和 JTAG 适配器固定功能
- CDC-ACM：
  - 配置虚拟串行功能，在大多数现代操作系统上可实现即插即用
  - 支持主机控制芯片复位和进入下载模式
- JTAG 适配器：
  - 支持使用紧凑的 JTAG 指令实现与 CPU 调试内核的快速通信
- 共配置两个 OUT 端点、三个 IN 端点和一个控制端点 EP_0，可实现最大 64 字节的数据载荷
- 集成内部 PHY：基本无需其他外部组件连接主机计算机
- USB D+ 与 USB D- 管脚可以互换

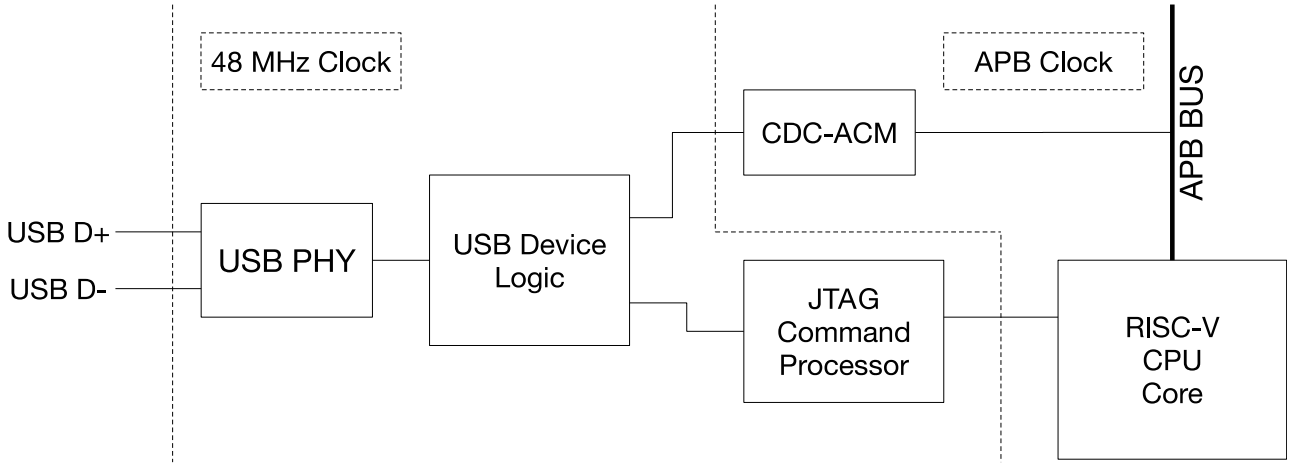


图 30-1. USB 串口/JTAG 控制器高层框图

如图 30-1 所示，USB 串口/JTAG 控制器包含 USB PHY、USB 设备接口、集成了响应捕捉单元的 JTAG 命令处理器，以及若干个 CDC-ACM 寄存器。PHY 和设备接口使用主 PLL (BBPLL) 时钟产生的 48 MHz 时钟作为时钟源，CDC-ACM 块中软件可访问的部分则使用 APB_CLK 作为时钟源。JTAG 命令处理器与 ESP32-H2 主处理器中的 JTAG 调试单元相连；CDC-ACM 寄存器则连接至 APB 总线，主 CPU 上运行的软件可对其进行读写访问。

请注意，USB 串口/JTAG 控制器仅支持 USB 2.0 全速标准 (12 Mbps)，不支持 USB 2.0 标准的其他模式（如 480 Mbps 的高速模式）。

图 30-2 展示了 USB 串口/JTAG 控制器中 USB 部分的内部详细结构。USB 串口/JTAG 控制器由一个 USB 2.0 全速设备组成，包含一个控制端点、一个虚拟中断端点、两个批量输入端点和两个批量输出端点。这些端点共同组成该复合型 USB 设备，具体可分为 CDC-ACM USB 设备和实现 JTAG 接口功能的供应商特定设备。JTAG 接口直接与芯片的 RISC-V CPU 的调试接口相连，可对运行在该 CPU 上的程序进行调试。同时，CDC-ACM 中包含一组寄存器，CPU 上运行的程序可对其进行读写操作。此外，芯片上的 ROM 启动代码可允许用户通过使用该接口重新烧录 flash。

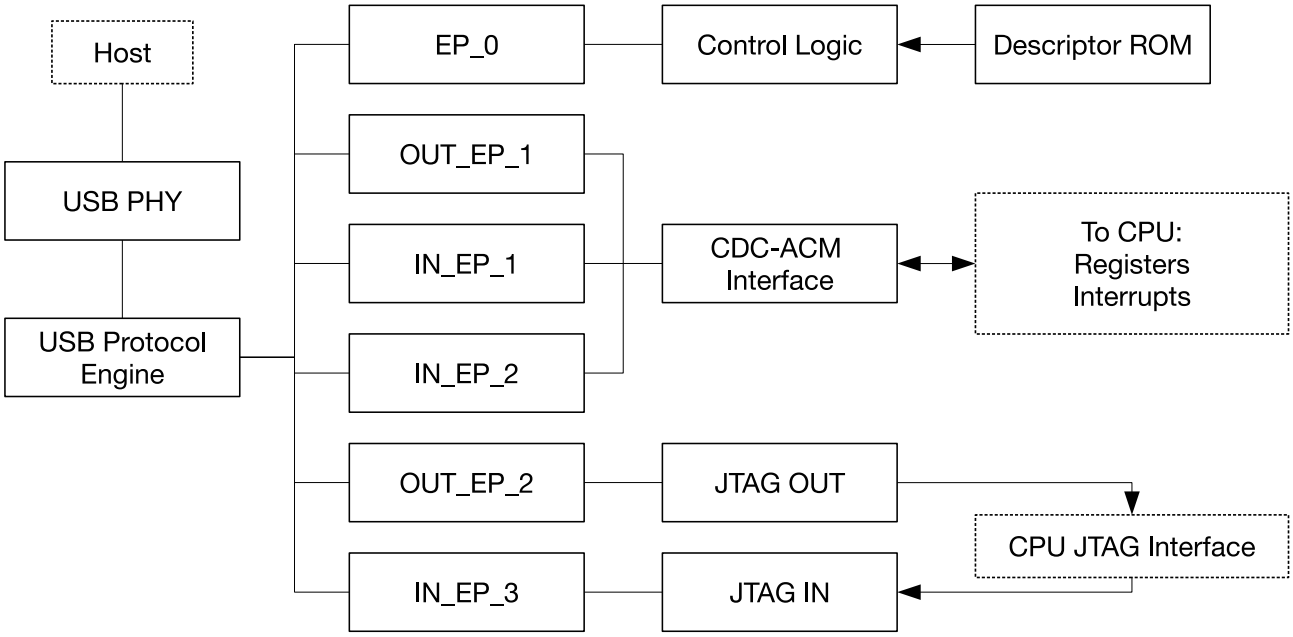


图 30-2. USB 串口/JTAG 控制器框图

30.3 功能描述

USB 串口/JTAG 控制器一边与 USB 主机处理器连接，另一边与 CPU 调试硬件以及通过 CDC-ACM 端口通信的软件连接。

30.3.1 CDC-ACM USB 接口描述

CDC-ACM 接口遵循标准 USB CDC-ACM 类别进行虚拟串口通信，包含一个虚拟中断端点（不会发送任何事件，无使用需求）以及一个批量输入端点 (Bulk IN) 和一个批量输出端点 (Bulk OUT) 进行数据接收和发送。这些端点一次可以处理最高 64 字节的数据包，实现高吞吐量。CDC-ACM 为标准的 USB 设备类型，主机一般无需任何特殊安装程序就能正常工作，也就是说，当 USB 调试设备正确连接至主机时，操作系统应能在片刻后显示新的串口信息。

CDC-ACM 接口可以接收以下标准 CDC-ACM 控制请求：

表 30-1. 标准 CDC-ACM 控制请求

命令	操作
SEND_BREAK	接收但忽略（虚拟命令）
SET_LINE_CODING	接收，发送值在软件中可读
GET_LINE_CODING	默认返回 9600 baud，无奇偶校验，8 个数据位，1 个停止位（可通过软件更改）
SET_CONTROL_LINE_STATE	设置 RTC/DTR 线的状态，如表 30-2 所示

除了通用的通信之外，CDC-ACM 接口还可以复位 ESP32-H2 并选择使其进入下载模式，从而烧录新的固件。这一功能可通过设置虚拟串口的 RTS 和 DTR 线来实现。

表 30-2. CDC-ACM 中 RTS 和 DTR 的设置

RTS	DTR	操作
0	0	清除下载模式标志
0	1	置位下载模式标志
1	0	复位 ESP32-H2
1	1	无操作

请注意，当 ESP32-H2 复位时，如果下载模式标志已置位，则 ESP32-H2 重启时将直接进入下载模式；如果下载模式标志已清除，则 ESP32-H2 将从 flash 启动。具体操作流程，请参见章节 30.4。除此之外，也可以通过烧写相应 eFuse 来禁用上述功能，详细信息请参见章节 5 *eFuse 控制器 (EFUSE)*。

30.3.2 CDC-ACM 固件接口描述

由于 USB 串口/JTAG 控制器与 ESP32-H2 的内部 APB 总线相连，因此 CPU 可直接与该模块交互，主要用于对连接主机上的虚拟串口进行读写操作。

CPU 向主机发送并从主机接收 0 ~ 64 字节大小的 USB CDC-ACM 串口数据包。主机已积累足够多的 CDC-ACM 数据时，将向 CDC-ACM 的接收端点发送一个数据包。如果 USB 串行/JTAG 控制器中有空闲缓冲区，该缓冲区将接收这一数据包。反之，主机会定期检查 USB 串口/JTAG 控制器内是否有待向主机发送的数据包，如果有，主机将接收这个数据包。

固件可通过以下两种方式之一获知是否有来自主机的新数据：第一，只要缓冲区中还有来自主机的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位将保持为 1；第二，如果有新的未读数据，USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断将被触发。当新数据可用时，固件可通过重复从 USB_SERIAL_JTAG_EP1_REG 读取字节来获取该数据。读取每个字节后，可通过检查 USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 位来看是否还有其他可读取的数据，从而确定需要读取的总字节数。读取完所有数据后，USB 调试设备会自动做好准备，以接收来自主机的新数据包。

当固件需要发送数据时，可将待发送数据置于发送缓冲区并触发刷写，从而使主机以 USB 数据包的形式接收该数据。在此之前，需确保发送缓冲区有可用空间以存储待发送数据。固件可通过读取 USB_REG_SERIAL_IN_EP_DATA_FREE 位检查发送缓冲区是否有可用空间；当该值为 1 时，发送缓冲区有可用空间。此时，固件可通过向 USB_SERIAL_JTAG_EP1_REG 寄存器中写入字节，从而向缓冲区中写入数据。但是，数据写入后并不会立即触发向主机发送数据，需首先对缓冲区执行刷写操作。刷写操作后，整个缓冲区将准备好被 USB 主机立即接收。可通过两种方式触发刷写：1) 将第 64 个字节写入缓冲区后，USB 硬件会自动将缓冲区刷写到主机；2) 固件可通过向 USB_SERIAL_JTAG_WR_DONE 写入 1 来触发刷写。

不论以何种方式触发刷写操作，在缓冲区中的所有数据都被主机读取完成之前，固件无法向缓冲区写入数据。主机读取完成后，将触发 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断，此时可向缓冲区内写入新的 64 字节数据。

在软件中可以处理一些带外串行请求，例如主机设置 DTR 和 RTS，以及改变线路状态。如果 CDC-ACM 接口收到 SET_LINE_CODING 请求，可将外设配置为触发 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断，此时可以从 USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG 寄存器读取线路编码。类似的，如果 SET_CONTROL_LINE_STATE 请求导致线路状态改变，将触发 USB_SERIAL_JTAG_RTS_CHG_INT and USB_SERIAL_JTAG_DTR_CHG_INT 中断。此后，软件可以通过 USB_SERIAL_JTAG_RTS 和 USB_SERIAL_JTAG_DTR 位来读取具体状态。注意，如前所述，某些 RTS/DTR 序列会导致 ESP32-H2 硬件复位。软件可以通过设置 USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 位来禁止硬件识别这些 DTR/RTS 序列，从而允许软件自由解读信号。

最后，主机可以通过 GET_LINE_CODING 读取当前线路状态。此事件将回送 USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG 寄存器中的数据，并触发 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。

30.3.3 USB-JTAG 接口：JTAG 命令处理器

USB-JTAG 接口使用一种供应商特定接口类型实现命令解析的功能。它由两个端点组成：一个用于接收命令，一个用于发送响应。此外，一些对时效性要求不高的命令也可以作为控制请求发出。

JTAG 命令处理器负责解析从主机到 JTAG 接口的命令。JTAG 命令处理器内部包含一个全四线 JTAG 总线，包括发送信号到 RISC-V CPU 的 TCK、TMS 和 TDI 输出线，以及从 CPU 返回信号至 JTAG 响应捕捉单元的 TDO 线。这些信号都符合 IEEE 1149.1 JTAG 标准。此外，还有一条 SRST 线用于复位 ESP32-H2。

另外，软件也可以置位 USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN，将这些信号通过 GPIO 交换矩阵连接至 ESP32-H2 的 IO 焊盘上，此操作也将允许通过 USB 串口/JTAG 控制器对外部设备进行调试。

JTAG 命令处理器会将每个接收到的半字节 (4 位) 解析为一条命令。由于 USB 以 8 位为一个字节接收数据，这就意味着每个字节中都包含两条命令。USB 命令处理器将先解析高 4 位字节，然后再解析低 4 位字节。这些命令用于控制内部 JTAG 总线的 TCK、TMS、TDI、SRST 线，以及向 JTAG 响应捕捉单元发出信号，说明需要捕捉 TDO 线（由 CPU 调试逻辑驱动）的状态。

JTAG 总线中，TCK、TMS、TDI 和 TDO 线直接与 RISC-V CPU 的 JTAG 调试逻辑相连。上文提到的 SRST 线则与 ESP32-H2 数字电路中的复位逻辑相连，该线电平拉高，芯片将进行芯片复位。请注意，SRST 线并不会对 USB 串口/JTAG 控制器模块产生影响。

1 个半字节中可包含以下命令：

表 30-3. 半字节中的命令

位	3	2	1	0
CMD_CLK	0	cap	tms	tdi
CMD_RST	1	0	0	srst
CMD_FLUSH	1	0	1	0
CMD_RSV	1	0	1	1
CMD_REP	1	1	R1	R0

- CMD_CLK：将 TDI 和 TMS 设置为指示值，并在 TCK 上发出一个时钟脉冲。如果 CAP 位为 1，将指示 JTAG 响应捕捉单元捕捉 TDO 线的状态。该指令构成了 JTAG 通信的基础。
- CMD_RST：将 SRST 线的状态设置为指示值。该命令可用于复位 ESP32-H2。

- CMD_FLUSH：指示 JTAG 响应捕捉单元对接收到的所有位的缓冲区进行刷写操作，以便主机读取这些位。请注意在某些情况下，一次 JTAG 通信会结束于第奇数个命令，即结束于第奇数个半字节。此时，可重复执行该命令直到获得偶数个半字节，使其组成整数个字节。
- CMD_RSV：该版本中保留。ESP32-H2 在接收到该命令时会自动忽略。
- CMD_REP：将上一条指令（非 CMD_REP）重复一定次数，以压缩多次重复 CMD_CLK 的命令流。因此，CMD_CLK 命令后可能跟随着多个 CMD_REP。一次 CMD_REP 命令产生的重复次数可表示为 $repetition\_count = (R1 \times 2 + R0) \times (4^{cmd\_rep\_count})$，其中 cmd_rep_count 表示该命令之前的 CMD_REP 数量。一条 CMD_REP 命令最多可重复 5 次，原始命令最多可重复 1023 次。请注意，CMD_REP 仅用于重复 CMD_CLK 命令。也就是说，如果在 CMD_FLUSH 后使用该命令，USB 设备将无法响应，需进行 USB 复位后才可恢复正常。

30.3.4 USB-JTAG 接口：CMD_REP 使用示例

下列命令用于演示如何使用 CMD_REP 命令。请注意，该示例中每个命令为半字节，命令流的每个字节为 0x0D 0x5E 0xCF。

1. 0x0 (CMD_CLK: cap=0, tdi=0, tms=0)
2. 0xD (CMD_REP: R1=0, R0=1)
3. 0x5 (CMD_CLK: cap=1, tdi=0, tms=1)
4. 0xE (CMD_REP: R1=1, R0=0)
5. 0xC (CMD_REP: R1=0, R0=0)
6. 0xF (CMD_REP: R1=1, R0=1)

每一步骤的具体操作为：

1. TCK 上发出一个时钟脉冲，TDI 和 TMS 设置为 0。未捕捉到任何数据。
2. TCK 上再次发出 $(0 \times 2 + 1) \times (4^0) = 1$ 个时钟脉冲，其他设置与步骤 1 相同。
3. TCK 上发出一个时钟脉冲，TDI 设置为 0，TMS 设置为 1。捕捉到 TDO 线上的数据。
4. TCK 上再次发出 $(1 \times 2 + 0) \times (4^0) = 2$ 个时钟脉冲，其他设置与步骤 3 相同。
5. 未发生任何动作：$(0 \times 2 + 0) \times (4^1) = 0$。请注意，该操作将增加下一步骤中的 cmd_rep_count 数值。
6. TCK 上再次发出 $(1 \times 2 + 1) \times (4^2) = 48$ 个时钟脉冲，其他设置与步骤 3 相同。

换言之，该命令流示例的操作结果等同于执行两次命令 1，然后执行 51 次命令 3 的效果。

30.3.5 USB-JTAG 接口：响应捕捉单元

响应捕捉单元首先读取内部 JTAG 总线的 TDO 线，并在命令处理器执行 CMD_CLK (cap=1) 命令时捕捉 TDO 线的值，然后将这个值放入内部移位寄存器中，且在接收到 8 位时向 USB 缓冲区写入 1 个字节。这 8 位中的最低有效位即为最先从 TDO 线读取的值。

一旦接收到 64 字节 (512 位) 数据或执行 CMD_FLUSH 命令后，响应捕捉单元将使缓冲区可被主机接收。请注意，USB 逻辑的接口为双缓冲。因此，只要 USB 的吞吐量充足，响应捕捉单元就可以随时接收更多数据，即当一个缓冲区等待发送给主机时，另一个缓冲区可以继续接收数据。当主机从缓冲区成功接收数据且响应捕捉单元对缓冲区执行刷写操作后，这两个缓冲区便可以交换位置。

同时，这也意味着一个命令流可导致最多 128 字节（若该命令流中有刷写命令，则该数字会减小）的捕捉数据生成，而无需主机主动接收这些数据。如果还是生成了超过该阈值数量的捕捉数据，则命令流将暂停，且设备在这些数据被读取之前不会接收其他命令。

另需注意，一般情况下，响应捕捉单元会尽量不发送 0 字节响应。例如，当发送一系列 CMD_FLUSH 命令后，并不会产生一系列 0 字节 USB 响应。但在当前版本中，一些特殊情况下也可能产生 0 字节响应。建议直接忽略这些响应信息。

30.3.6 USB-JTAG 接口：控制传输请求

除命令处理器和响应捕捉单元之外，USB-JTAG 接口也可接收一些控制请求，具体如下表所示：

表 30-4. USB-JTAG 控制请求

bmRequestType	bRequest	wValue	wIndex	wLength	Data
01000000b	0 (VEND_JTAG_SETDIV)	[divider]	接口	0	None
01000000b	1 (VEND_JTAG_SETIO)	[iobits]	接口	0	None
11000000b	2 (VEND_JTAG_GETTDO)	0	接口	1	[iostate]
10000000b	6 (GET_DESCRIPTOR)	0x2000	0	256	[jtag cap desc]

- VEND_JTAG_SETDIV：设置使用的分频器。该命令直接影响 TCK 时钟脉冲的持续时间。TCK 时钟脉冲来自于由内部分频器向下分频得到的 48 MHz 基准时钟。该控制请求允许主机设置此内部分频器。请注意，该分频器在启动时的初始值为 2，即 TCK 时钟速率一般为 40 MHz。
- VEND_JTAG_SETIO：跳过 JTAG 命令处理器，直接将内部 TDI、TDO、TMS 和 SRST 线设置为指定值。这些指定值在 wValue 字段中以 11'b0, srst, trst, tck, tms, tdi 的格式编码。
- VEND_JTAG_GETTDO：跳过 JTAG 响应捕捉单元，直接读取内部 TDO 信号。该请求返回 1 个字节，其中的最低有效位代表 TDO 线的状态。
- GET_DESCRIPTOR：为标准 USB 请求，该请求也可使用供应商专用的 0x2000 wValue 获取 JTAG 功能描述符。该请求返回一定字节，具体字节数所代表的 USB-JTAG 适配器功能如表 30-5 所示。这一固定结构允许主机软件自动支持未来的硬件新版本，无需再次更新。

ESP32-H2 包含的 JTAG 功能描述符如下表所示。请注意，所有 16 位值都为小端序存储。

表 30-5. JTAG 功能描述符

字节	数值	描述
0	1	JTAG 协议功能结构的版本
1	10	JTAG 协议功能长度
2	1	结构类型：1 代表高速功能结构类型
3	8	该高速功能结构长度
4 ~ 5	4800	以 10 KHz 为增量的 JTAG 基准时钟速度，其最大速度为该值的一半
6 ~ 7	1	最小分频系数，可通过 VEND_JTAG_SETDIV 设置
8 ~ 9	255	最大分频系数，可通过 VEND_JTAG_SETDIV 设置

30.4 操作建议

使用 USB 串口/JTAG 控制器之前，几乎不需要多余的配置。除了主机操作系统已经完成的标准 USB 初始化之外，USB-JTAG 硬件本身不需要进行任何配置。此外，CDC-ACM 虚拟串口在主机端也支持即插即用。

固件方面也几乎不需要初始化。USB 硬件是自初始化的，在其启动后，如果固件连接了一台主机并在 CDC-ACM 接口上监听，除非固件设置了中断处理程序，否则无需任何特定设置就可以实现前文所述的数据交换。

需要注意的是，可能会出现主机未连接或 CDC-ACM 虚拟串口未打开的情况。在这种情况下，发送至主机的数据包永远无法被接收，发送缓冲区也永远不会为空。因而，必须对这种情况进行检测并执行超时操作，以便清楚地检测主机侧的端口是否关闭。

其次，需知 USB 设备依赖于产生 48 MHz USB PHY 时钟的 BBPLL。如果此 PLL 被禁用，USB 通信将停止。

上述情况可能发生在 Deep-sleep 期间。在 Deep-sleep 模式下，USB 串口/JTAG 控制器（及其连接的 RISC-V CPU）将完全断电。如果有设备需要在这两种模式下进行调试，最好使用外部 JTAG 调试器和串行接口。

CDC-ACM 接口还可用于复位芯片，使其进入或退出下载模式。产生正确的握手信号序列则有些复杂，因为大多数操作系统仅支持分别设置或清零 DTR 和 RTS，无法同时进行。此外，一些驱动程序（如 Windows 系统上的标准 CDC-ACM 驱动程序）须先设置 RTS 后才可设置 DTR，此时用户必须先明确设置 RTS，才能传播 DTR 的值。推荐遵循以下步骤进行设置。

复位芯片使其进入下载模式：

表 30-6. 复位芯片进入下载模式

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	初始化以获取数值
清除 RTS	RTS=0, DTR=0	-
设置 DTR	RTS=0, DTR=1	设置下载模式标志
清除 RTS	RTS=0, DTR=1	传播 DTR
设置 RTS	RTS=1, DTR=1	-
清除 DTR	RTS=1, DTR=0	复位芯片
设置 RTS	RTS=1, DTR=0	传播 DTR
清除 RTS	RTS=0, DTR=0	清除下载标志

复位 SoC 使其从 flash 启动：

表 30-7. 复位 SoC 从 flash 启动

操作	内部状态	备注
清除 DTR	RTS=?, DTR=0	-
清除 RTS	RTS=0, DTR=0	清除下载标志
设置 RTS	RTS=1, DTR=0	复位 SoC
清除 RTS	RTS=0, DTR=0	退出复位

30.5 中断

- USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT：JTAG 输入端口 2 接收到刷写命令时触发。
- USB_SERIAL_JTAG_SOF_INT：接收到 SOF 帧时触发。
- USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT：串口输出端点接收到 1 个数据包时触发。

- USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT：串口输入端点为空时触发。
- USB_SERIAL_JTAG_PID_ERR_INT：检测到 PID 错误时触发。
- USB_SERIAL_JTAG_CRC5_ERR_INT：检测到 CRC5 错误时触发。
- USB_SERIAL_JTAG_CRC16_ERR_INT：检测到 CRC16 错误时触发。
- USB_SERIAL_JTAG_STUFF_ERR_INT：检测到位填充错误时触发。
- USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT：输入端点 1 接收到一个 IN 令牌时触发。
- USB_SERIAL_JTAG_USB_BUS_RESET_INT：检测到 USB 总线复位时触发。
- USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT：输出端点 1 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT：输出端点 2 接收到有效载荷为 0 的数据包时触发。
- USB_SERIAL_JTAG_RTS_CHG_INT：USB 串行通道的 RTS 电平改变时触发。
- USB_SERIAL_JTAG_DTR_CHG_INT：USB 串行通道的 DTR 电平改变时触发。
- USB_SERIAL_JTAG_GET_LINE_CODE_INT：收到 GET LINE CODING 请求的电平时触发。
- USB_SERIAL_JTAG_SET_LINE_CODE_INT：收到 SET LINE CODING 请求的电平时触发。

30.6 寄存器列表

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

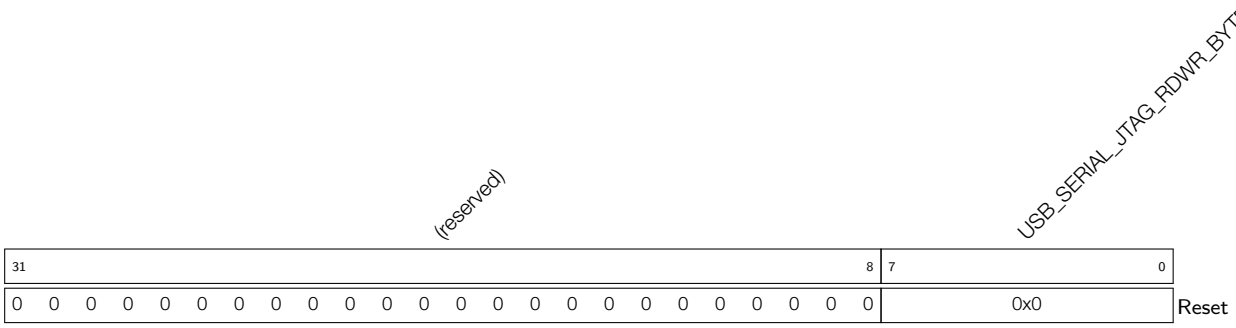
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
USB_SERIAL_JTAG_EP1_REG	CDC-ACM 输入/输出端点 FIFO 访问配置寄存器	0x0000	R/W
USB_SERIAL_JTAG_EP1_CONF_REG	CDC-ACM FIFO 配置寄存器	0x0004	varies
USB_SERIAL_JTAG_CONF0_REG	PHY 硬件配置寄存器	0x0018	R/W
USB_SERIAL_JTAG_TEST_REG	PHY 调试寄存器	0x001C	varies
USB_SERIAL_JTAG_MISC_CONF_REG	时钟使能控制寄存器	0x0044	R/W
USB_SERIAL_JTAG_MEM_CONF_REG	存储器配置寄存器	0x0048	R/W
USB_SERIAL_JTAG_CHIP_RST_REG	CDC-ACM 芯片重置寄存器	0x004C	varies
USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG	GET_LINE_CODING 命令的 W0	0x0058	R/W
USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG	GET_LINE_CODING 命令的 W1	0x005C	R/W
USB_SERIAL_JTAG_CONFIG_UPDATE_REG	配置寄存器更新寄存器	0x0060	WT
USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG	Serial AFIFO 配置寄存器	0x0064	varies
中断寄存器			
USB_SERIAL_JTAG_INT_RAW_REG	中断原始状态寄存器	0x0008	R/WTC/SS
USB_SERIAL_JTAG_INT_ST_REG	中断状态寄存器	0x000C	RO
USB_SERIAL_JTAG_INT_ENA_REG	中断使能状态寄存器	0x0010	R/W
USB_SERIAL_JTAG_INT_CLR_REG	中断清除状态寄存器	0x0014	WT
状态寄存器			
USB_SERIAL_JTAG_JFIFO_ST_REG	JTAG FIFO 状态与控制寄存器	0x0020	varies
USB_SERIAL_JTAG_FRAM_NUM_REG	接收 SOF 帧索引寄存器	0x0024	RO
USB_SERIAL_JTAG_IN_EP0_ST_REG	输入端点状态信息寄存器	0x0028	RO
USB_SERIAL_JTAG_IN_EP1_ST_REG	CDC-ACM 输入端点状态信息寄存器	0x002C	RO
USB_SERIAL_JTAG_IN_EP2_ST_REG	CDC-ACM 中断输入端点状态信息寄存器	0x0030	RO
USB_SERIAL_JTAG_IN_EP3_ST_REG	JTAG 输入端点状态信息寄存器	0x0034	RO
USB_SERIAL_JTAG_OUT_EP0_ST_REG	输出端点状态信息寄存器	0x0038	RO
USB_SERIAL_JTAG_OUT_EP1_ST_REG	CDC-ACM 输出端点状态信息寄存器	0x003C	RO
USB_SERIAL_JTAG_OUT_EP2_ST_REG	JTAG 输出端点状态信息寄存器	0x0040	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG	SET_LINE_CODING 命令的 W0	0x0050	RO
USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG	SET_LINE_CODING 命令的 W1	0x0054	RO
USB_SERIAL_JTAG_BUS_RESET_ST_REG	USB 总线重置状态寄存器	0x0068	RO
版本寄存器			
USB_SERIAL_JTAG_DATE_REG	版本寄存器	0x0080	R/W

30.7 寄存器

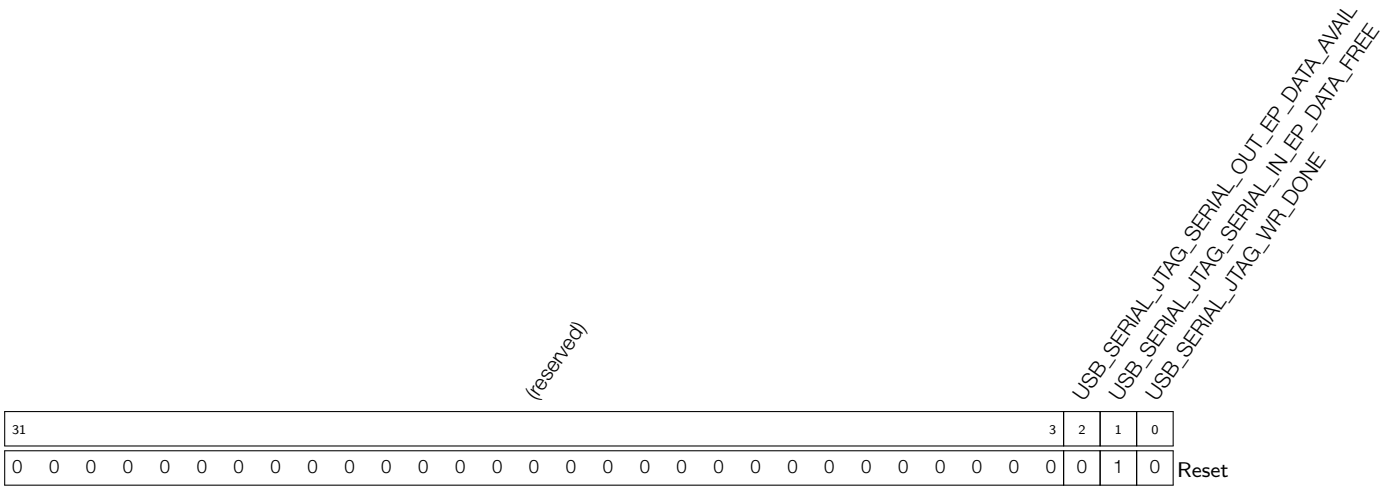
本小节的所有地址均为相对于 USB 串口/JTAG 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 30.1. USB_SERIAL_JTAG_EP1_REG (0x0000)



USB_SERIAL_JTAG_RDWR_BYTE 向 UART TX FIFO 中写入数据或从 UART RX FIFO 中读取数据。USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 置位时，用户可向 UART TX FIFO 中写入数据（最大 64 字节）。USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 置位时，用户可通过查看 USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 的值获知接收到的数据量，然后从 UART RX FIFO 中读取这些数据。(R/W)

Register 30.2. USB_SERIAL_JTAG_EP1_CONF_REG (0x0004)



USB_SERIAL_JTAG_WR_DONE 配置是否表明已完成向 UART TX FIFO 写入字节数据。

0: 无效

1: 表明已完成向 UART TX FIFO 写入字节数据

该位将保持为 0，直到 USB 主机读取 UART TX FIFO 中的数据。

(WT)

USB_SERIAL_JTAG_SERIAL_IN_EP_DATA_FREE 表示 UART TX FIFO 是否可以写入数据。

0：UART TX FIFO 已写满，不可写入数据

1：UART TX FIFO 不为空且可以写入数据

写入 USB_SERIAL_JTAG_WR_DONE 后，该位将置为 0，直到其中的数据已发送至 USB 主机。

(RO)

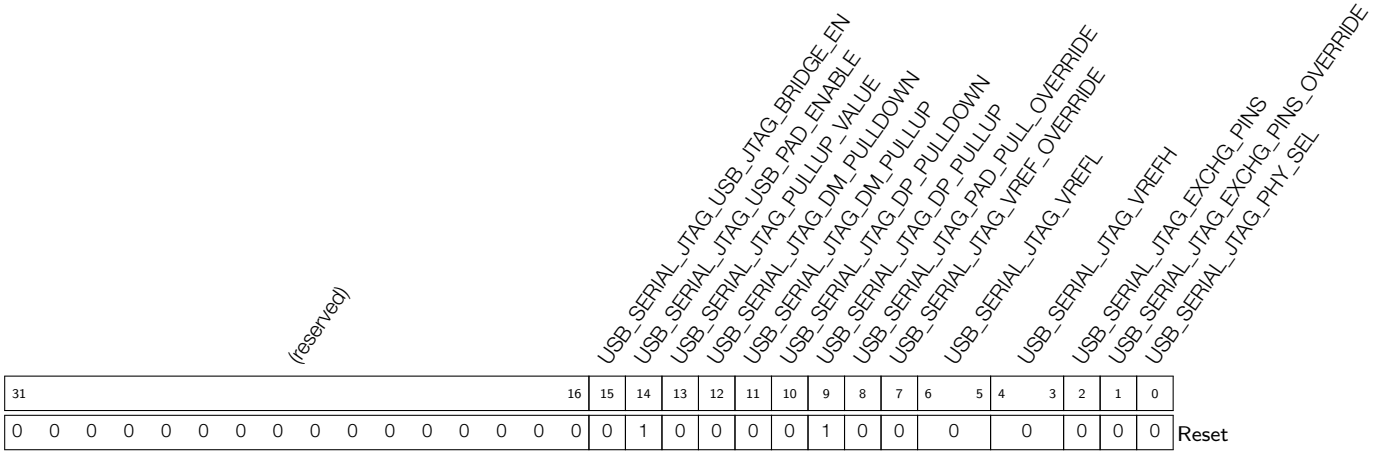
USB_SERIAL_JTAG_SERIAL_OUT_EP_DATA_AVAIL 表示 UART RX FIFO 中是否有数据。

0: UART RX FIFO 中没有数据

1: UART RX FIFO 中有数据

(RO)

Register 30.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)



USB_SERIAL_JTAG_PHY_SEL 配置选择使用内部 PHY 或外部 PHY。

0：内部 PHY

1：外部 PHY

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚交换。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_EXCHG_PINS 配置是否交换 USB D+ 和 D- 管脚。

0：无效

1：交换

(R/W)

USB_SERIAL_JTAG_VREFH 配置单端输入高阈值。

0：1.76 V

1：1.84 V

2：1.92 V

3：2.00 V

(R/W)

USB_SERIAL_JTAG_VREFL 配置单端输入低阈值。

0：0.80 V

1：0.88 V

2：0.96 V

3：1.04 V

(R/W)

USB_SERIAL_JTAG_VREF_OVERRIDE 配置是否使能软件控制输入阈值。

0：不使能

1：使能

(R/W)

见下页...

Register 30.3. USB_SERIAL_JTAG_CONF0_REG (0x0018)

接上页...

USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 配置是否使能软件控制 USB D+ 和 D- 管脚的上下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DP_PULLUP 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D+ 管脚的上拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DP_PULLDOWN 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D+ 管脚的下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_DM_PULLDOWN 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时，是否使能 USB D- 管脚的下拉电阻。

0：不使能

1：使能

(R/W)

USB_SERIAL_JTAG_PULLUP_VALUE 配置当 USB_SERIAL_JTAG_PAD_PULL_OVERRIDE 为 1 时的上拉数值。

0：2.2 K

1：1.1 K

(R/W)

USB_SERIAL_JTAG_USB_PAD_ENABLE 配置是否使能 USB 填充功能。

0：不使能

1：使能

(R/W)

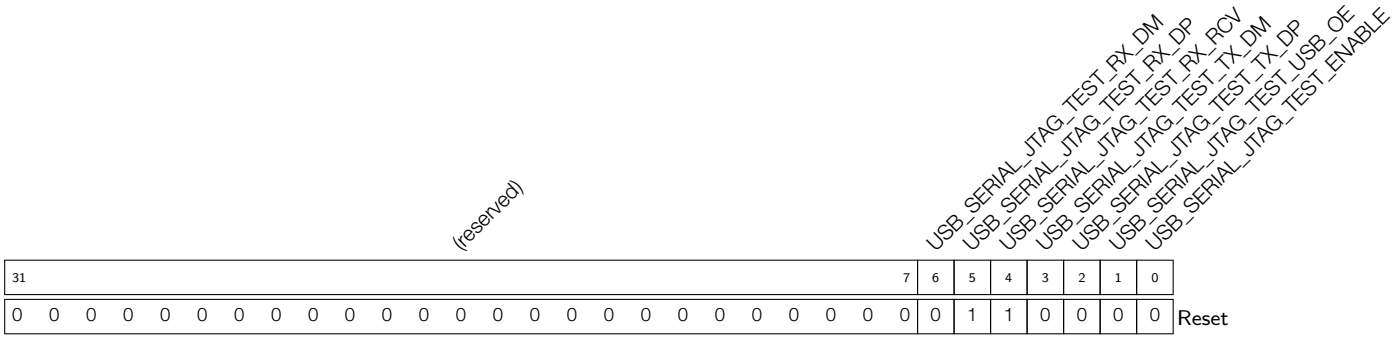
USB_SERIAL_JTAG_USB_JTAG_BRIDGE_EN 配置是否断开 usb_jtag 和内部 JTAG 的连接。

0：usb_jtag 连接到 CPU 的内部 JTAG 端口

1：usb_jtag 和内部 JTAG 之间断开连接，MTMS、MTDI 和 MTCK 为通过 GPIO 交换矩阵的输出，MTDO 为通过 GPIO 交换矩阵的输入

(R/W)

Register 30.4. USB_SERIAL_JTAG_TEST_REG (0x001C)



USB_SERIAL_JTAG_TEST_ENABLE 配置是否使能 USB 填充测试模式。

0：不使能，恢复正常模式

1：使能 USB 填充测试模式

使能 USB 填充测试模式将允许使用该寄存器的其他位来控制/读取 USB 填充。

(R/W)

USB_SERIAL_JTAG_TEST_USB_OE 配置是否使能 USB 填充输出。

0：将 D+ 和 D- 设为高阻抗

1：在 D+ 和 D- 管脚上输出 USB_SERIAL_JTAG_TEST_TX_DP 和 USB_SERIAL_JTAG_TEST_TX_DM 中设置的值

(R/W)

USB_SERIAL_JTAG_TEST_TX_DP 当 USB_SERIAL_JTAG_TEST_USB_OE 为 1 时，设置测试模式下 USB D+ 管脚的值。(R/W)

USB_SERIAL_JTAG_TEST_TX_DM 当 USB_SERIAL_JTAG_TEST_USB_OE 为 1 时，设置测试模式下 USB D- 管脚的值。(R/W)

USB_SERIAL_JTAG_TEST_RX_RCV 表示在测试模式下 USB D- 和 USB D+ 管脚之间电压差的当前逻辑水平。

0：USB D- 电压高于 USB D+

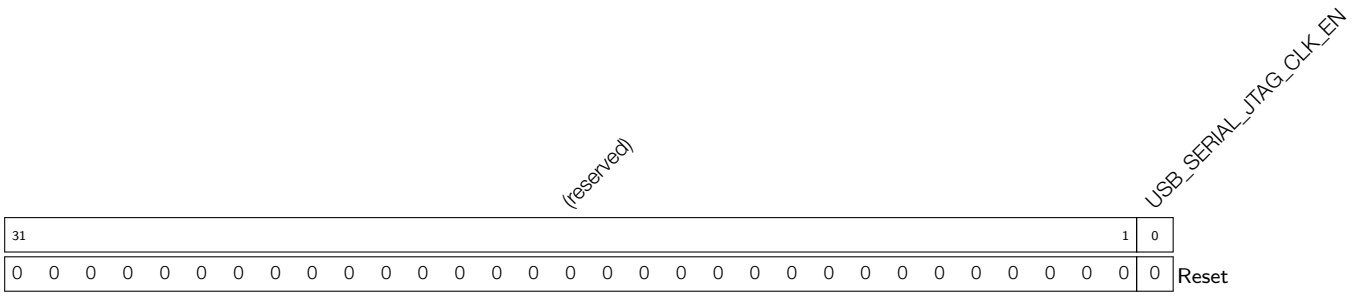
1：USB D- 电压低于 USB D+

(RO)

USB_SERIAL_JTAG_TEST_RX_DP 表示在测试模式下 USB D+ 管脚的逻辑电平。(RO)

USB_SERIAL_JTAG_TEST_RX_DM 表示在测试模式下 USB D- 管脚的逻辑电平。(RO)

Register 30.5. USB_SERIAL_JTAG_MISC_CONF_REG (0x0044)



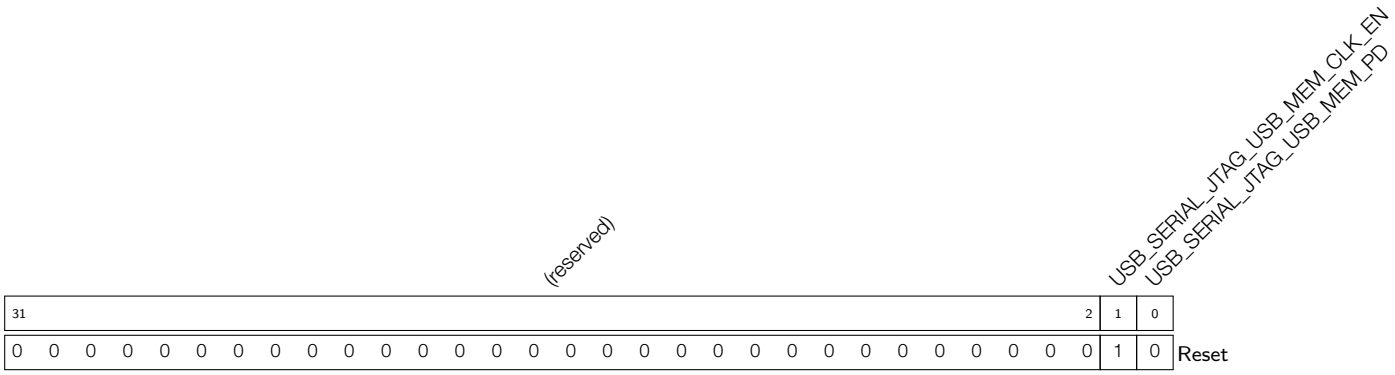
USB_SERIAL_JTAG_CLK_EN 配置是否强制打开寄存器的时钟。

0：支持仅当应用程序向寄存器写入数据时打开时钟

1：强制打开寄存器的时钟

(R/W)

Register 30.6. USB_SERIAL_JTAG_MEM_CONF_REG (0x0048)



USB_SERIAL_JTAG_USB_MEM_PD 配置是否关闭 USB 存储器。

0：无效

1：关闭

(R/W)

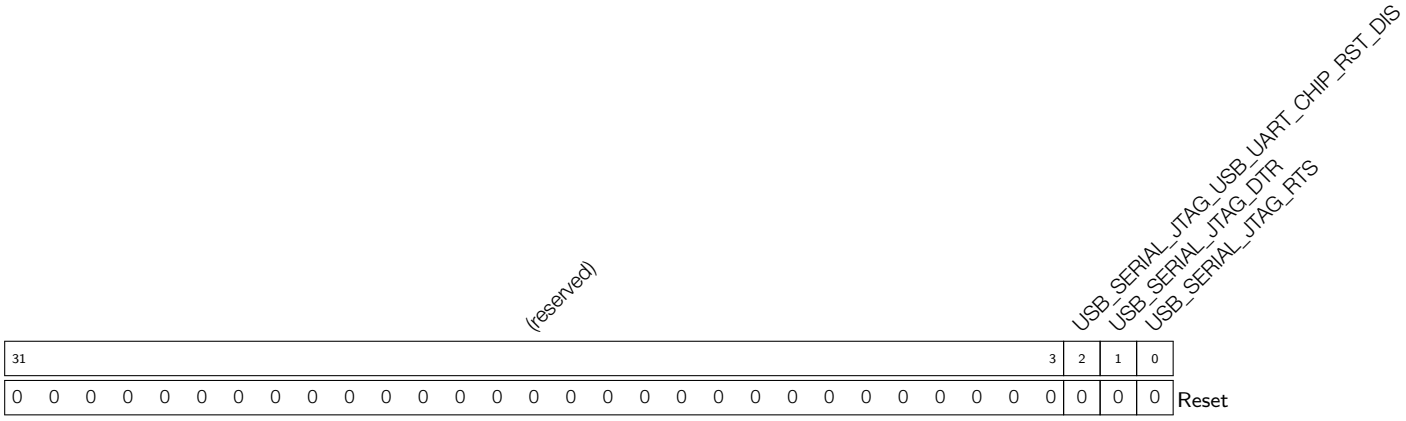
USB_SERIAL_JTAG_USB_MEM_CLK_EN 配置是否强制对 USB 存储器进行时钟分频。

0：无效

1：强制

(R/W)

Register 30.7. USB_SERIAL_JTAG_CHIP_RST_REG (0x004C)



USB_SERIAL_JTAG_RTS 表示最近的 SET_LINE_CODING 命令所设定的 RTS 信号的状态。(RO)

USB_SERIAL_JTAG_DTR 表示最近的 SET_LINE_CODING 命令所设定的 DTR 信号的状态。(RO)

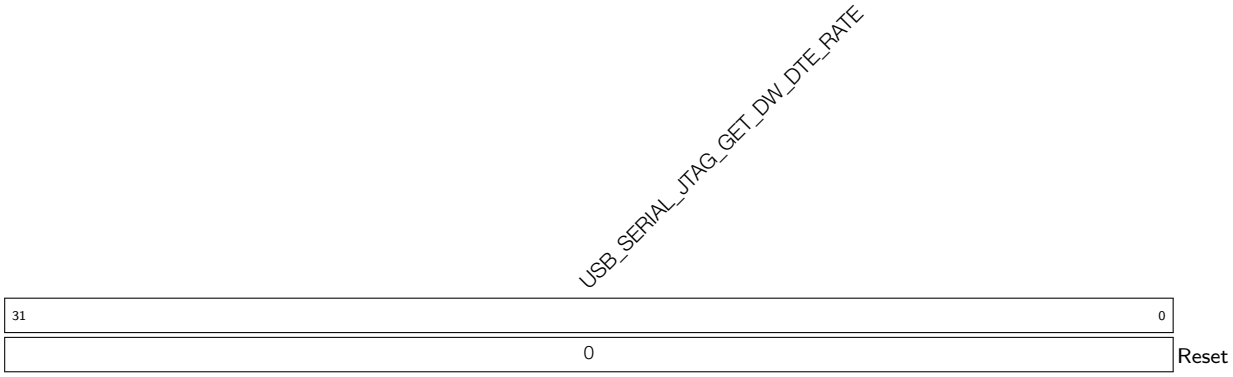
USB_SERIAL_JTAG_USB_UART_CHIP_RST_DIS 配置是否禁用来自 USB 串行通道的芯片复位。

0: 无效

1: 禁用

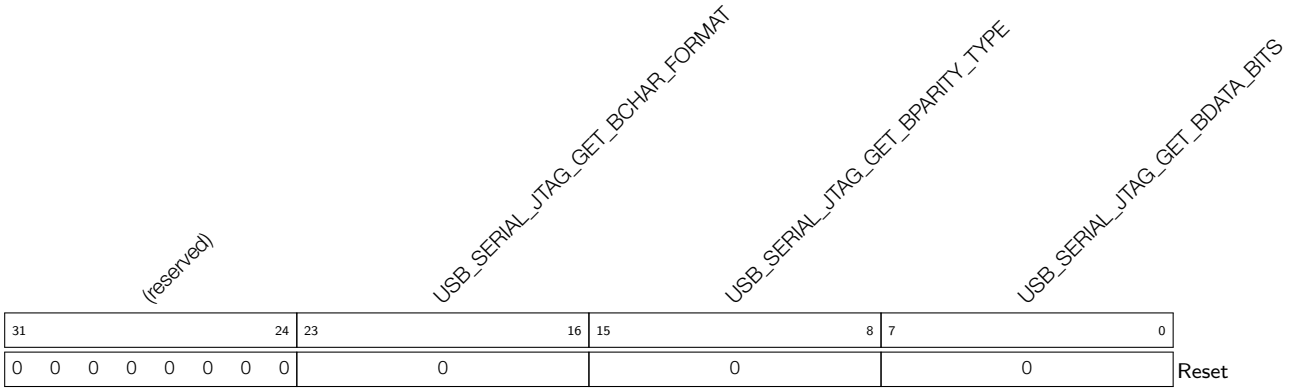
(R/W)

Register 30.8. USB_SERIAL_JTAG_GET_LINE_CODE_W0_REG (0x0058)



USB_SERIAL_JTAG_GET_DW_DTE_RATE 配置软件设置的 dwDTERate 值（单位：比特每秒），该值由 GET_LINE_CODING 命令请求，它不会影响实际通信速度。(R/W)

Register 30.9. USB_SERIAL_JTAG_GET_LINE_CODE_W1_REG (0x005C)

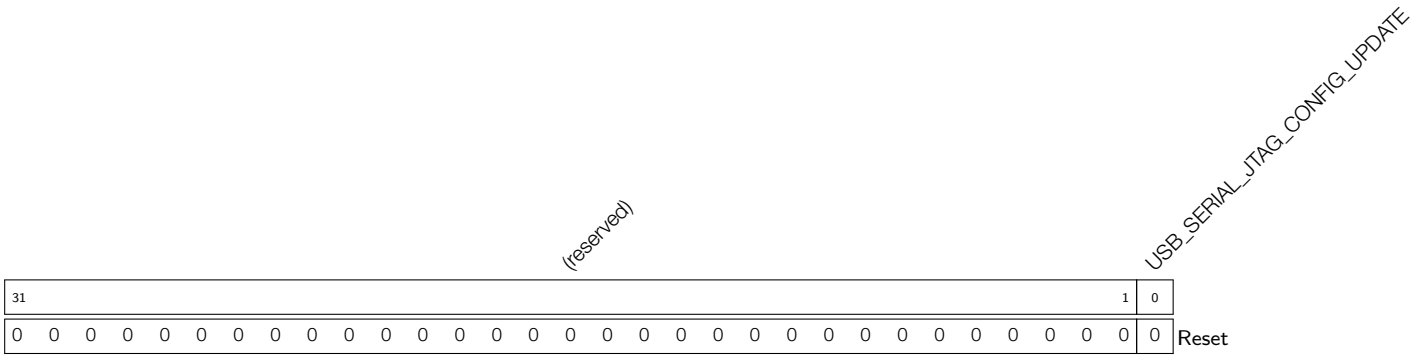


USB_SERIAL_JTAG_GET_BDATA_BITS 配置软件设置的 bDataBits 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BPARITY_TYPE 配置软件设置的 bParityType 值，该值由 GET_LINE_CODING 命令请求。(R/W)

USB_SERIAL_JTAG_GET_BCHAR_FORMAT 配置软件设置的 bCharFormat 值，该值由 GET_LINE_CODING 命令请求。(R/W)

Register 30.10. USB_SERIAL_JTAG_CONFIG_UPDATE_REG (0x0060)



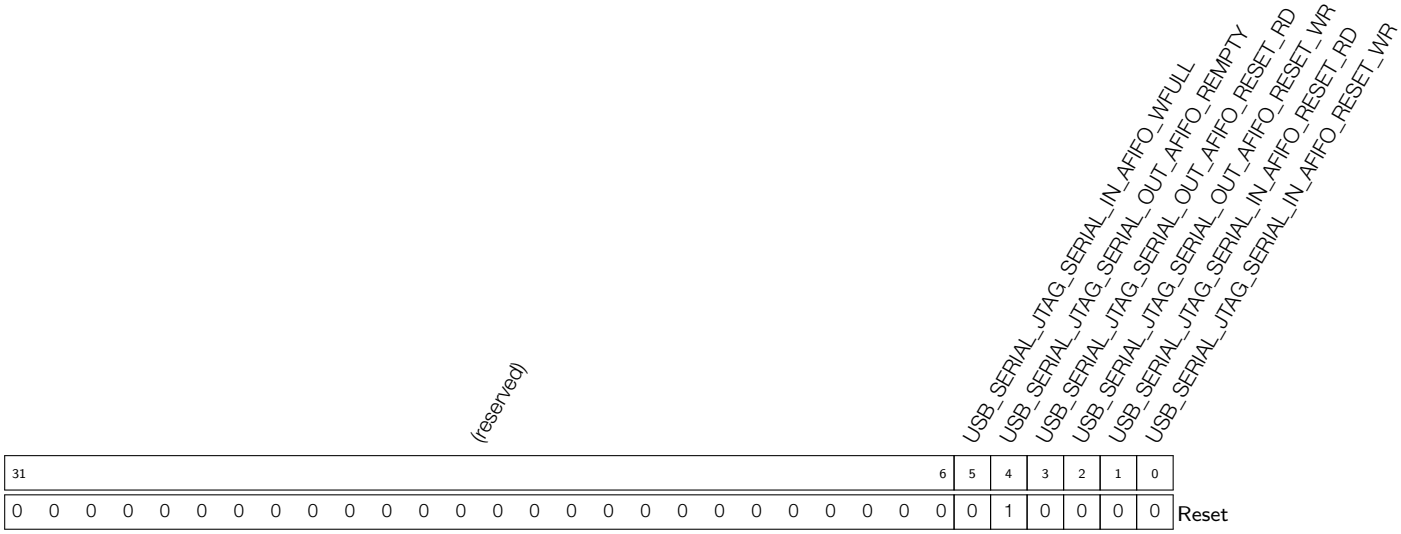
USB_SERIAL_JTAG_CONFIG_UPDATE 配置是否将配置寄存器的值从 APB 时钟域更新到 48 MHz 时钟域。

0: 无效

1: 更新

(WT)

Register 30.11. USB_SERIAL_JTAG_SER_AFIFO_CONFIG_REG (0x0064)



USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_WR 配置是否复位 CDC_ACM IN 异步 FIFO 写入时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_RESET_RD 配置是否复位 CDC_ACM IN 异步 FIFO 读取时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_WR 配置是否复位 CDC_ACM OUT 异步 FIFO 写入时钟域。

0: 无效

1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_RESET_RD 配置是否复位 CDC_ACM OUT 异步 FIFO 读取时钟域。

0: 无效

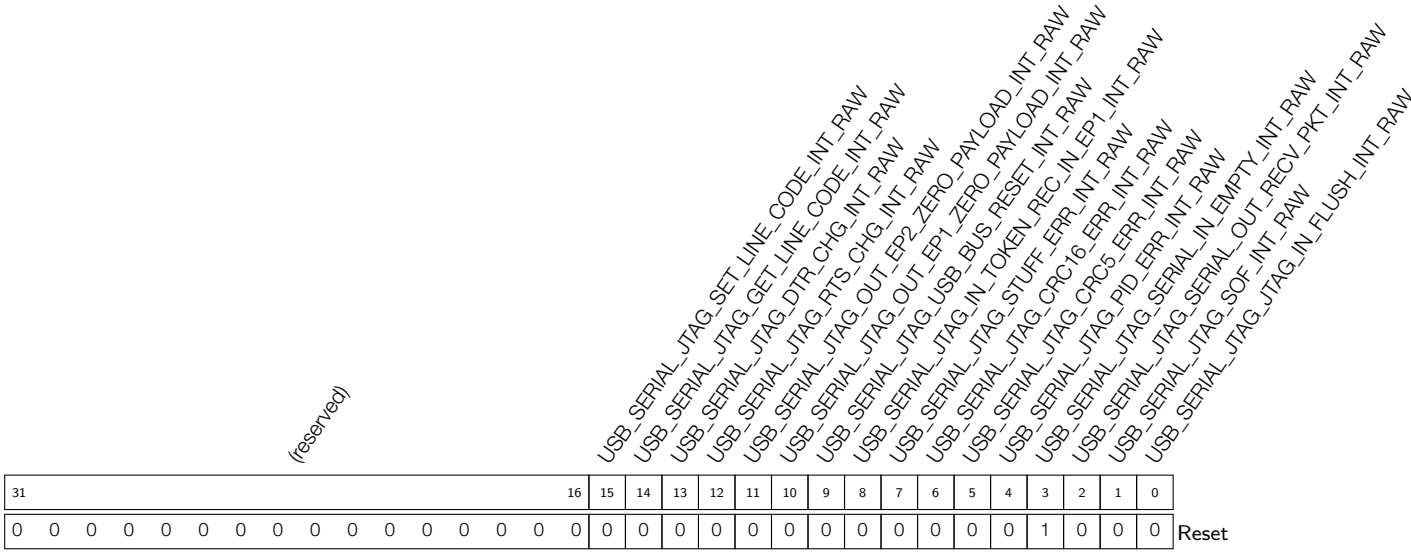
1: 复位

(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_AFIFO_REMPTY 表示 CDC_ACM OUT 读取时钟域的异步 FIFO 空信号。(RO)

USB_SERIAL_JTAG_SERIAL_IN_AFIFO_WFULL 表示 CDC_ACM IN 写入时钟域的异步 FIFO 满信号。(RO)

Register 30.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_RAW USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SOF_INT_RAW USB_SERIAL_JTAG_SOF_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_RAW USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_RAW USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_PID_ERR_INT_RAW USB_SERIAL_JTAG_PID_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC5_ERR_INT_RAW USB_SERIAL_JTAG_CRC5_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_CRC16_ERR_INT_RAW USB_SERIAL_JTAG_CRC16_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_STUFF_ERR_INT_RAW USB_SERIAL_JTAG_STUFF_ERR_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_RAW USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_RAW USB_SERIAL_JTAG_USB_BUS_RESET_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_RAW USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的原始中断状态。(R/WTC/SS)

见下页...

Register 30.12. USB_SERIAL_JTAG_INT_RAW_REG (0x0008)

接上页...

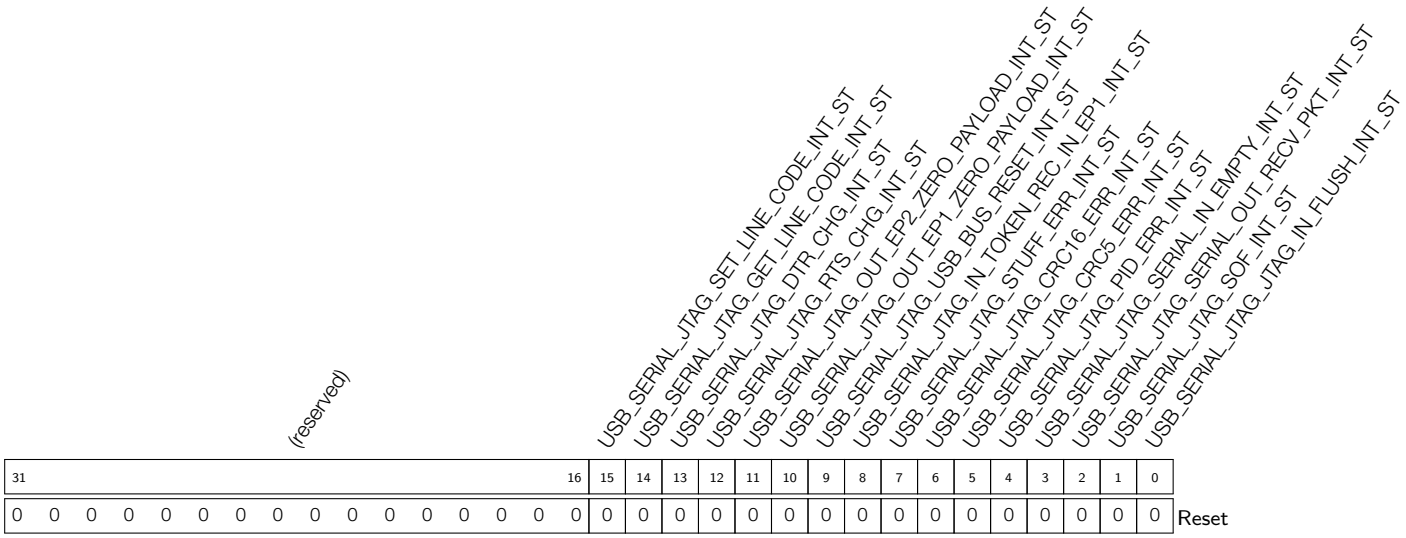
USB_SERIAL_JTAG_RTS_CHG_INT_RAW USB_SERIAL_JTAG_RTS_CHG_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_DTR_CHG_INT_RAW USB_SERIAL_JTAG_DTR_CHG_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_GET_LINE_CODE_INT 的原始中断状态。(R/WTC/SS)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_RAW USB_SERIAL_JTAG_SET_LINE_CODE_INT 的原始中断状态。(R/WTC/SS)

Register 30.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ST USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SOF_INT_ST USB_SERIAL_JTAG_SOF_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ST USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ST USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_PID_ERR_INT_ST USB_SERIAL_JTAG_PID_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC5_ERR_INT_ST USB_SERIAL_JTAG_CRC5_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_CRC16_ERR_INT_ST USB_SERIAL_JTAG_CRC16_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_STUFF_ERR_INT_ST USB_SERIAL_JTAG_STUFF_ERR_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ST USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ST USB_SERIAL_JTAG_USB_BUS_RESET_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ST USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 的屏蔽中断状态。(RO)

见下页...

Register 30.13. USB_SERIAL_JTAG_INT_ST_REG (0x000C)

接上页...

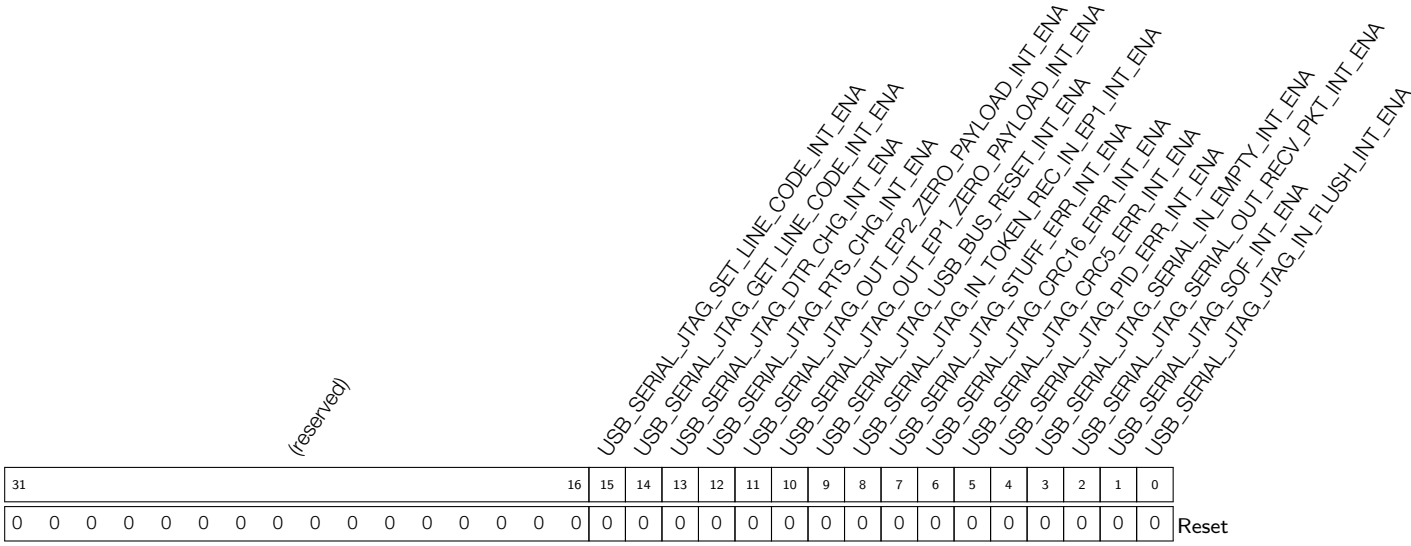
USB_SERIAL_JTAG_RTS_CHG_INT_ST USB_SERIAL_JTAG_RTS_CHG_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_DTR_CHG_INT_ST USB_SERIAL_JTAG_DTR_CHG_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ST USB_SERIAL_JTAG_GET_LINE_CODE_INT 的屏蔽中断状态。(RO)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ST USB_SERIAL_JTAG_SET_LINE_CODE_INT 的屏蔽中断状态。(RO)

Register 30.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_ENA 写 1 使能 USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 中断。(R/W)

USB_SERIAL_JTAG_SOF_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SOF_INT 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断。(R/W)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断。(R/W)

USB_SERIAL_JTAG_PID_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_PID_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_CRC5_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_CRC5_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_CRC16_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_CRC16_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_STUFF_ERR_INT_ENA 写 1 使能 USB_SERIAL_JTAG_STUFF_ERR_INT 中断。(R/W)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_ENA 写 1 使能 USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 中断。(R/W)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_ENA 写 1 使能 USB_SERIAL_JTAG_USB_BUS_RESET_INT 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_ENA 写 1 使能 USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 中断。(R/W)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_ENA 写 1 使能 USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 中断。(R/W)

见下页...

Register 30.14. USB_SERIAL_JTAG_INT_ENA_REG (0x0010)

接上页...

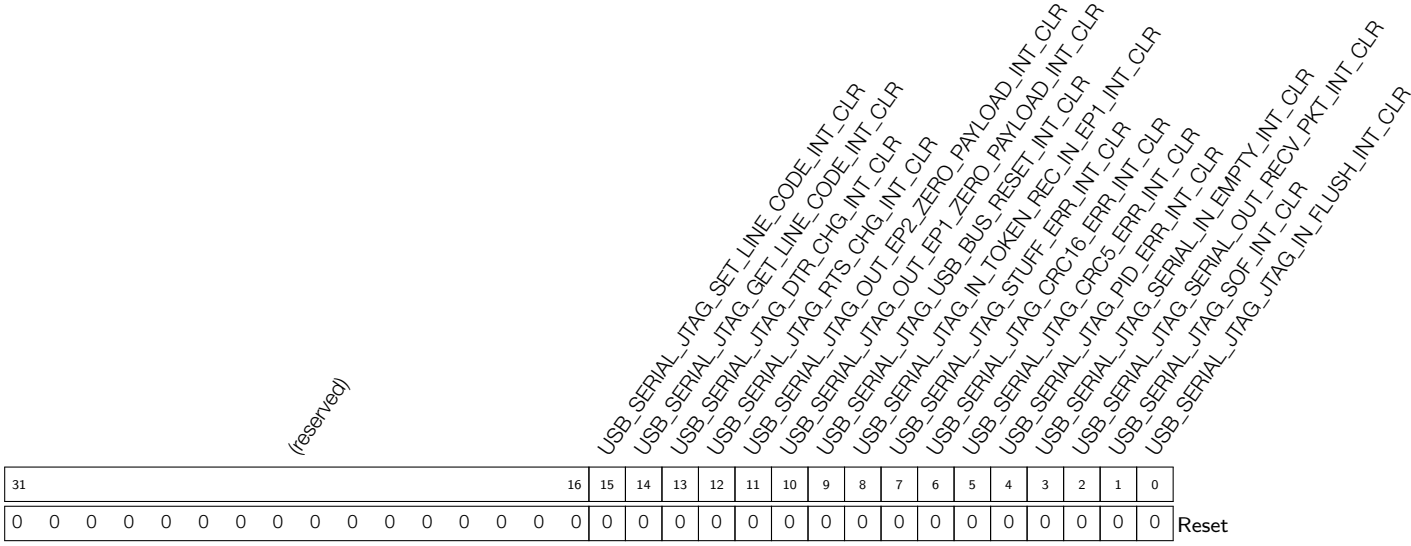
USB_SERIAL_JTAG_RTS_CHG_INT_ENA 写 1 使能 USB_SERIAL_JTAG_RTS_CHG_INT 中断。(R/W)

USB_SERIAL_JTAG_DTR_CHG_INT_ENA 写 1 使能 USB_SERIAL_JTAG_DTR_CHG_INT 中断。(R/W)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_ENA 写 1 使能 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。(R/W)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_ENA 写 1 使能 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断。(R/W)

Register 30.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)



USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT_CLR 写 1 清除 USB_SERIAL_JTAG_JTAG_IN_FLUSH_INT 中断。(WT)

USB_SERIAL_JTAG_SOF_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SOF_INT 中断。(WT)

USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断。(WT)

USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SERIAL_IN_EMPTY_INT 中断。(WT)

USB_SERIAL_JTAG_PID_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_PID_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_CRC5_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_CRC5_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_CRC16_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_CRC16_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_STUFF_ERR_INT_CLR 写 1 清除 USB_SERIAL_JTAG_STUFF_ERR_INT 中断。(WT)

USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT_CLR 写 1 清除 USB_SERIAL_JTAG_IN_TOKEN_REC_IN_EP1_INT 中断。(WT)

USB_SERIAL_JTAG_USB_BUS_RESET_INT_CLR 写 1 清除 USB_SERIAL_JTAG_USB_BUS_RESET_INT 中断。(WT)

USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT_CLR 写 1 清除 USB_SERIAL_JTAG_OUT_EP1_ZERO_PAYLOAD_INT 中断。(WT)

USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT_CLR 写 1 清除 USB_SERIAL_JTAG_OUT_EP2_ZERO_PAYLOAD_INT 中断。(WT)

见下页...

Register 30.15. USB_SERIAL_JTAG_INT_CLR_REG (0x0014)

接上页...

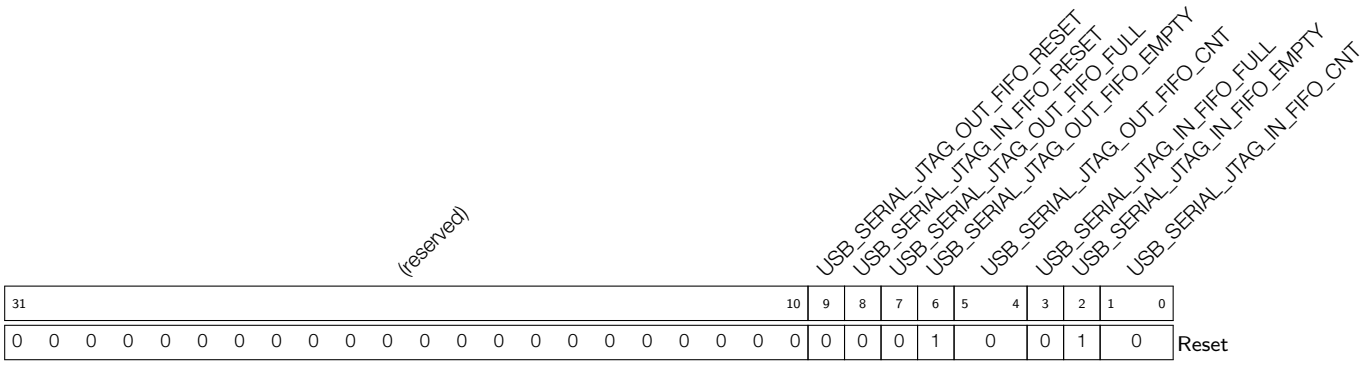
USB_SERIAL_JTAG_RTS_CHG_INT_CLR 写 1 清除 USB_SERIAL_JTAG_RTS_CHG_INT 中断。(WT)

USB_SERIAL_JTAG_DTR_CHG_INT_CLR 写 1 清除 USB_SERIAL_JTAG_DTR_CHG_INT 中断。(WT)

USB_SERIAL_JTAG_GET_LINE_CODE_INT_CLR 写 1 清除 USB_SERIAL_JTAG_GET_LINE_CODE_INT 中断。(WT)

USB_SERIAL_JTAG_SET_LINE_CODE_INT_CLR 写 1 清除 USB_SERIAL_JTAG_SET_LINE_CODE_INT 中断。(WT)

Register 30.16. USB_SERIAL_JTAG_JFIFO_ST_REG (0x0020)



USB_SERIAL_JTAG_IN_FIFO_CNT 表示 JTAG IN FIFO 计数器。(RO)

USB_SERIAL_JTAG_IN_FIFO_EMPTY 表示 JTAG IN FIFO 是否为空。

0：JTAG IN FIFO 不为空

1：JTAG IN FIFO 为空

(RO)

USB_SERIAL_JTAG_IN_FIFO_FULL 表示 JTAG IN FIFO 是否为满。

0：JTAG IN FIFO 不为满

1：JTAG IN FIFO 为满

(RO)

USB_SERIAL_JTAG_OUT_FIFO_CNT 表示 JTAG OUT FIFO 计数器。(RO)

USB_SERIAL_JTAG_OUT_FIFO_EMPTY 表示 JTAG OUT FIFO 是否为空。

0：JTAG OUT FIFO 不为空

1：JTAG OUT FIFO 为空

(RO)

USB_SERIAL_JTAG_OUT_FIFO_FULL 表示 JTAG OUT FIFO 是否为满。

0：JTAG OUT FIFO 不为满

1：JTAG OUT FIFO 为满

(RO)

USB_SERIAL_JTAG_IN_FIFO_RESET 配置是否复位 JTAG IN FIFO。

0：无效

1：复位

(R/W)

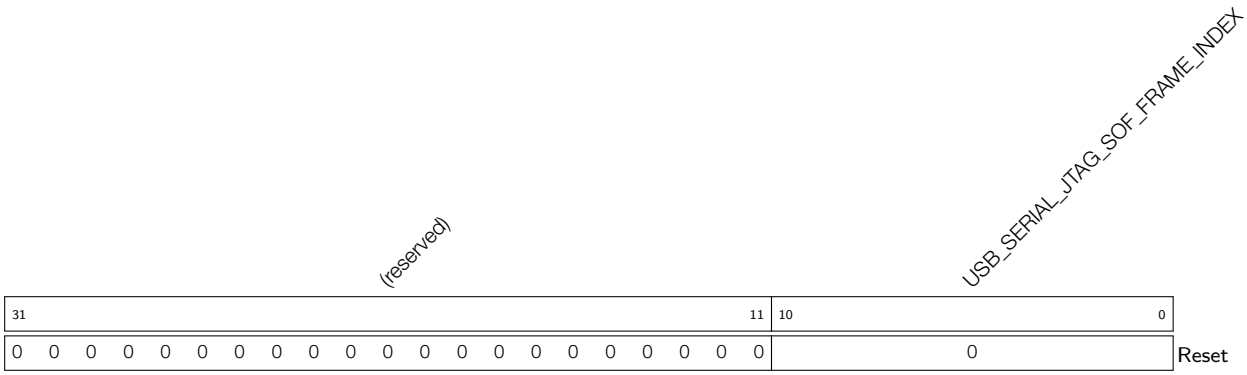
USB_SERIAL_JTAG_OUT_FIFO_RESET 配置是否复位 JTAG OUT FIFO。

0：无效

1：复位

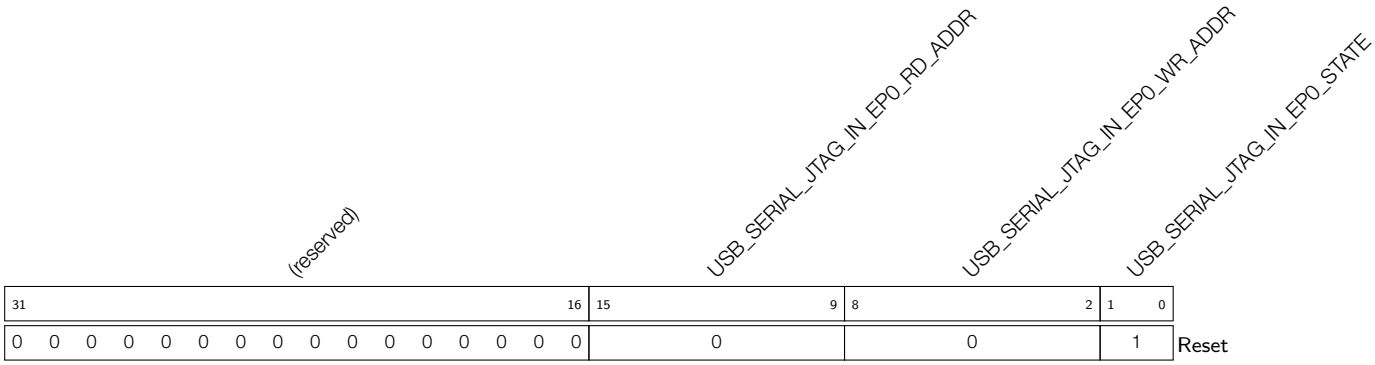
(R/W)

Register 30.17. USB_SERIAL_JTAG_FRAM_NUM_REG (0x0024)



USB_SERIAL_JTAG_SOF_FRAME_INDEX 接收 SOF 帧的帧索引。(RO)

Register 30.18. USB_SERIAL_JTAG_IN_EP0_ST_REG (0x0028)

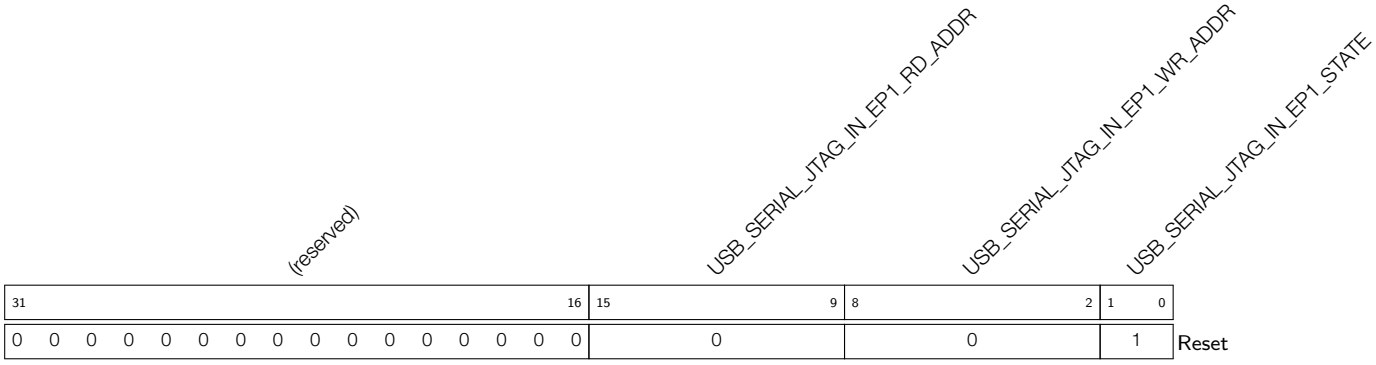


USB_SERIAL_JTAG_IN_EP0_STATE 表示输入端点 0 的状态。(RO)

USB_SERIAL_JTAG_IN_EP0_WR_ADDR 表示输入端点 0 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP0_RD_ADDR 表示输入端点 0 的读取数据地址。(RO)

Register 30.19. USB_SERIAL_JTAG_IN_EP1_ST_REG (0x002C)

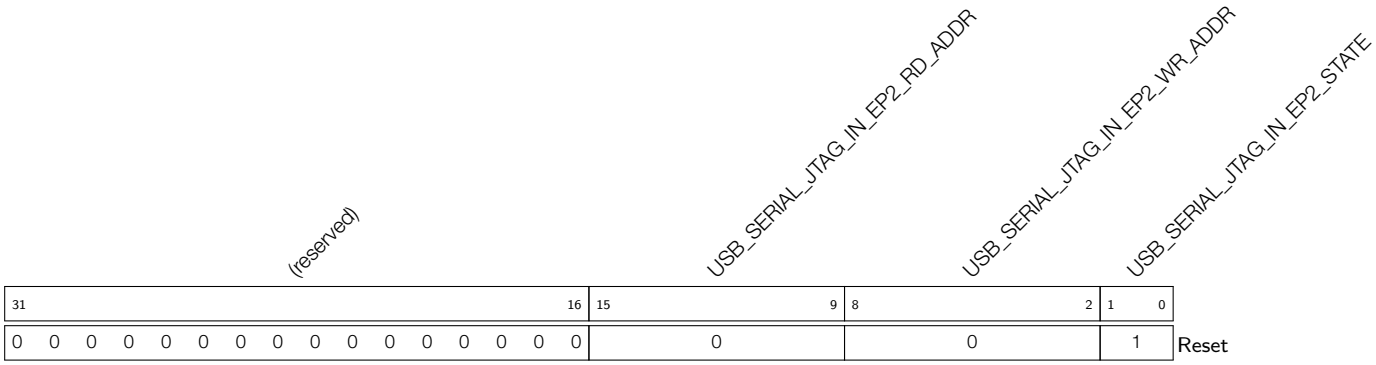


USB_SERIAL_JTAG_IN_EP1_STATE 表示输入端点 1 的状态。(RO)

USB_SERIAL_JTAG_IN_EP1_WR_ADDR 表示输入端点 1 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP1_RD_ADDR 表示输入端点 1 的读取数据地址。(RO)

Register 30.20. USB_SERIAL_JTAG_IN_EP2_ST_REG (0x0030)

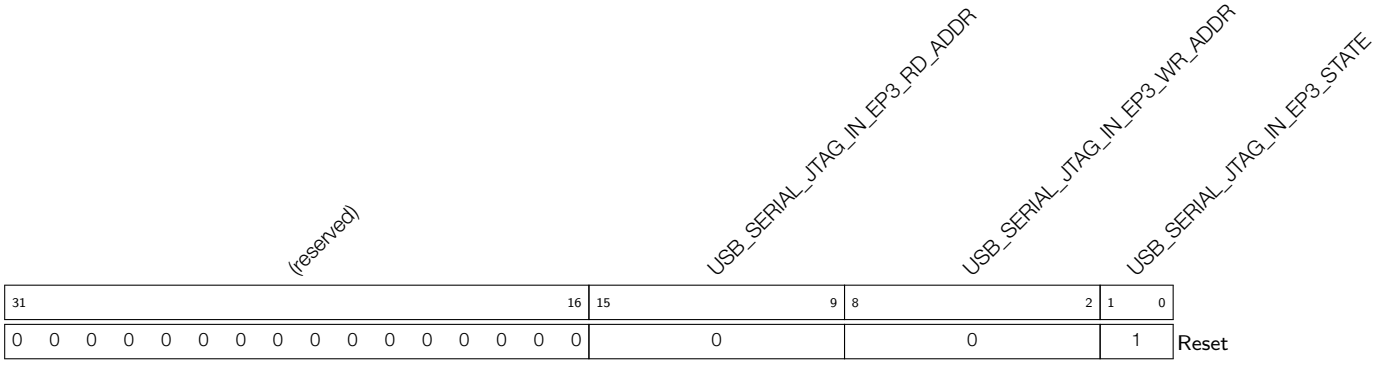


USB_SERIAL_JTAG_IN_EP2_STATE 表示输入端点 2 的状态。(RO)

USB_SERIAL_JTAG_IN_EP2_WR_ADDR 表示输入端点 2 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP2_RD_ADDR 表示输入端点 2 的读取数据地址。(RO)

Register 30.21. USB_SERIAL_JTAG_IN_EP3_ST_REG (0x0034)

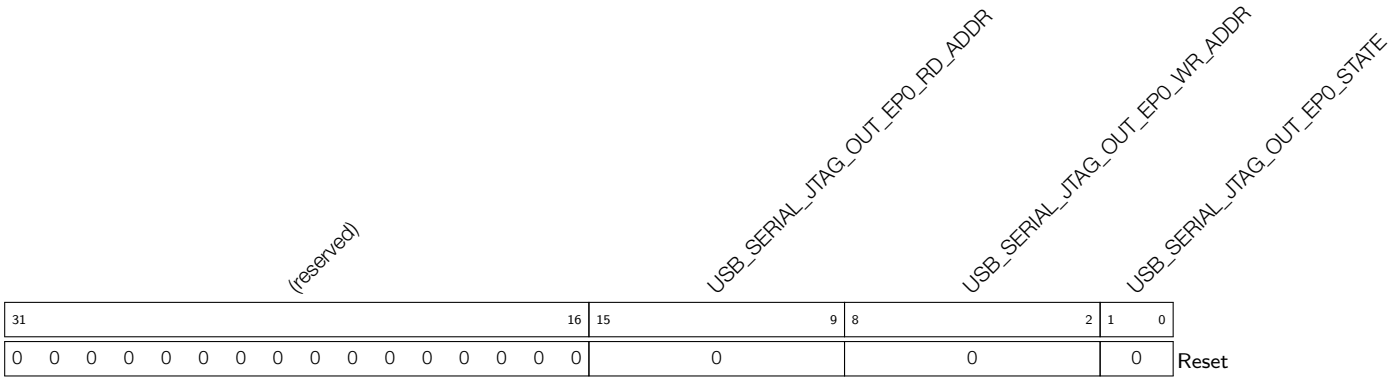


USB_SERIAL_JTAG_IN_EP3_STATE 表示输入端点 3 的状态。(RO)

USB_SERIAL_JTAG_IN_EP3_WR_ADDR 表示输入端点 3 的写入数据地址。(RO)

USB_SERIAL_JTAG_IN_EP3_RD_ADDR 表示输入端点 3 的读取数据地址。(RO)

Register 30.22. USB_SERIAL_JTAG_OUT_EP0_ST_REG (0x0038)



USB_SERIAL_JTAG_OUT_EP0_STATE 表示输出端点 0 的状态。(RO)

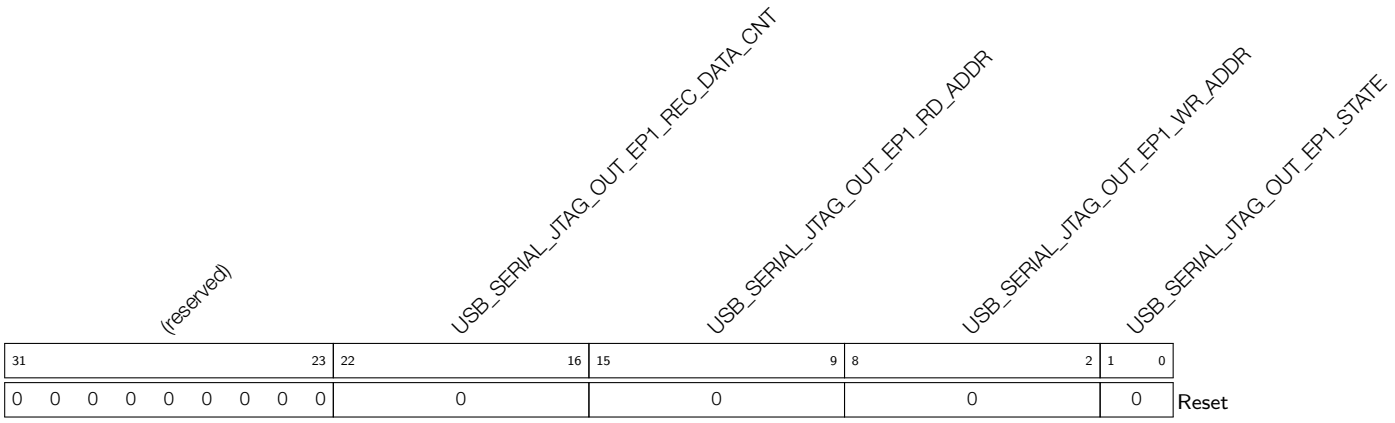
USB_SERIAL_JTAG_OUT_EP0_WR_ADDR 表示输出端点 0 的写入数据地址。

当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 0 中有 (USB_SERIAL_JTAG_OUT_EP0_WR_ADDR – 2) 个字节数据。

(RO)

USB_SERIAL_JTAG_OUT_EP0_RD_ADDR 表示输出端点 0 的读取数据地址。(RO)

Register 30.23. USB_SERIAL_JTAG_OUT_EP1_ST_REG (0x003C)



USB_SERIAL_JTAG_OUT_EP1_STATE 表示输出端点 1 的状态。(RO)

USB_SERIAL_JTAG_OUT_EP1_WR_ADDR 表示输出端点 1 的写入数据地址。

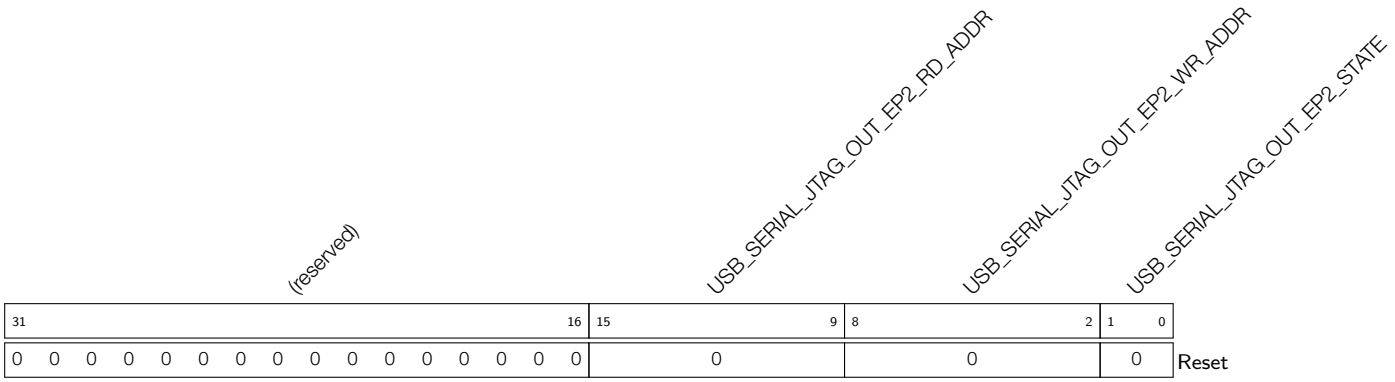
当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 1 中有 (USB_SERIAL_JTAG_OUT_EP1_WR_ADDR – 2) 个字节数据。

(RO)

USB_SERIAL_JTAG_OUT_EP1_RD_ADDR 表示输出端点 1 的读取数据地址。(RO)

USB_SERIAL_JTAG_OUT_EP1_REC_DATA_CNT 当接收到一个数据包时输出端点 1 中的数据计数器。(RO)

Register 30.24. USB_SERIAL_JTAG_OUT_EP2_ST_REG (0x0040)



USB_SERIAL_JTAG_OUT_EP2_STATE 表示输出端点 2 的状态。(RO)

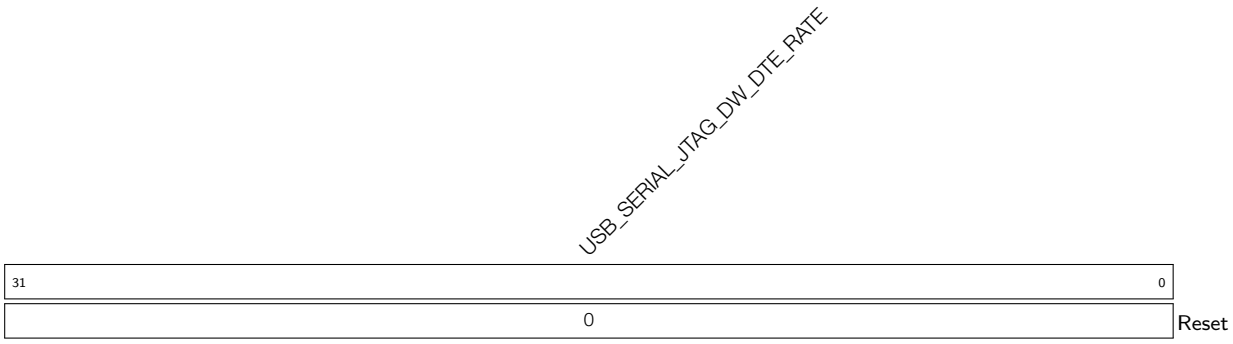
USB_SERIAL_JTAG_OUT_EP2_WR_ADDR 表示输出端点 2 的写入数据地址。

当检测到 USB_SERIAL_JTAG_SERIAL_OUT_RECV_PKT_INT 中断时，输出端点 2 中有 (USB_SERIAL_JTAG_OUT_EP2_WR_ADDR – 2) 个字节数据。

(RO)

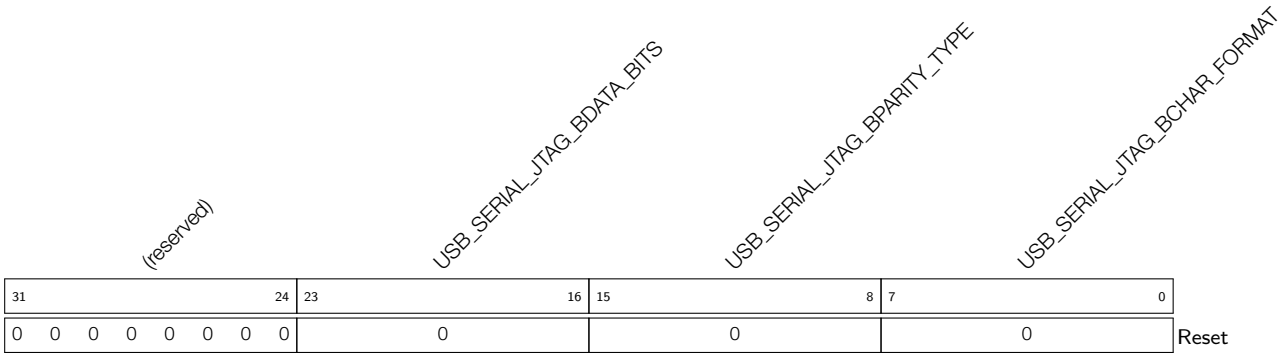
USB_SERIAL_JTAG_OUT_EP2_RD_ADDR 表示输出端点 2 的读取数据地址。(RO)

Register 30.25. USB_SERIAL_JTAG_SET_LINE_CODE_W0_REG (0x0050)



USB_SERIAL_JTAG_DW_DTE_RATE 表示主机通过 SET_LINE_CODING 命令设置的 dwDTERate 值，其单位为比特每秒），它不会影响实际通信速度。(RO)

Register 30.26. USB_SERIAL_JTAG_SET_LINE_CODE_W1_REG (0x0054)

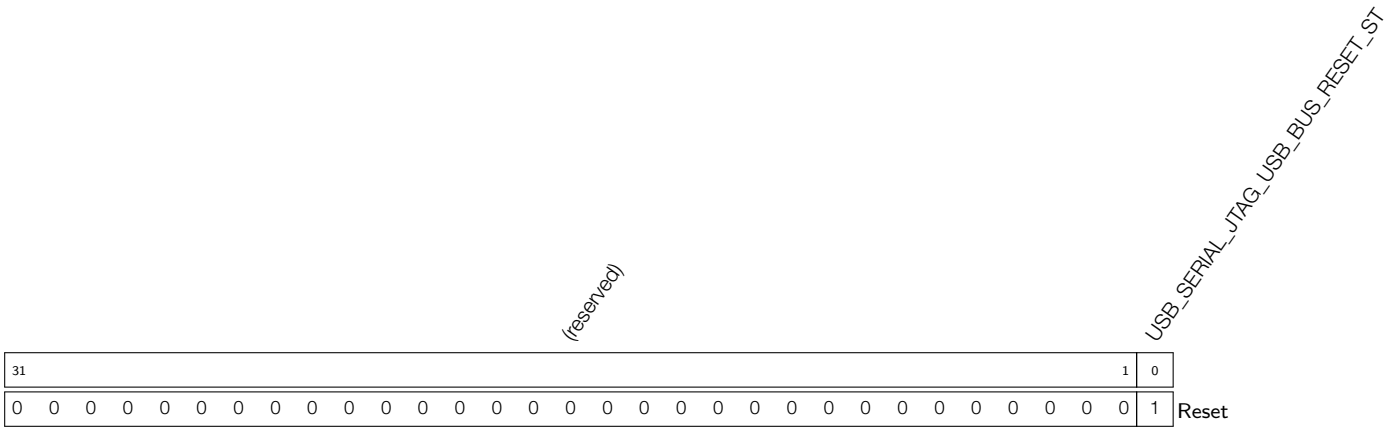


USB_SERIAL_JTAG_BCHAR_FORMAT 表示主机通过 SET_LINE_CODING 命令设置的 bCharFormat 值。(RO)

USB_SERIAL_JTAG_BPARITY_TYPE 表示主机通过 SET_LINE_CODING 命令设置的 bParityTpye 值。(RO)

USB_SERIAL_JTAG_BDATA_BITS 表示主机通过 SET_LINE_CODING 命令设置的 bDataBits 值。(RO)

Register 30.27. USB_SERIAL_JTAG_BUS_RESET_ST_REG (0x0068)



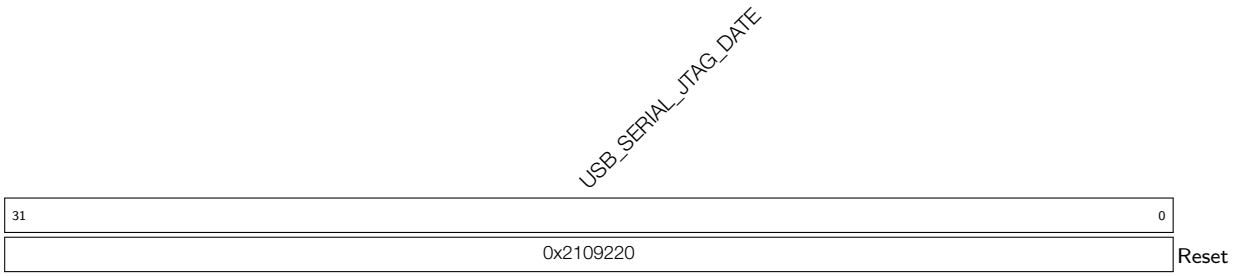
USB_SERIAL_JTAG_USB_BUS_RESET_ST 表示 USB 总线复位是否被释放。

0：USB 串口/JTAG 控制器处于 USB 总线复位状态

1：USB 总线复位被释放

(RO)

Register 30.28. USB_SERIAL_JTAG_DATE_REG (0x0080)



USB_SERIAL_JTAG_DATE 版本控制寄存器。(R/W)

31 双线汽车接口 (TWAI)

双线车载串口 (Two-wire Automotive Interface, TWAI®) 协议是一种多主机、多播的通信协议，具有错误检测、发送错误信号以及内置报文优先仲裁等功能。TWAI 协议适用于汽车和工业应用（参见章节 31.2）。

ESP32-H2 包含一个 TWAI 控制器，控制器可通过外部收发器连接到 TWAI 总线。TWAI 控制器包含一系列先进的功能，用途广泛，可用于如汽车产品、工业自动化控制、楼宇自动化等。

31.1 主要特性

ESP32-H2 TWAI 控制器具有以下特性：

- 兼容 ISO 11898-1 协议（CAN 规范 2.0）
- 支持标准格式（11 位标识符）和扩展格式（29 位标识符）两种帧格式
- 支持 1 Kbit/s ~ 1 Mbit/s 位速率
- 支持多种操作模式：
  - 正常模式
  - 只听模式（不影响总线）
  - 自测模式（发送数据时无需应答）
- 配置 64 字节接收 FIFO
- 支持特殊发送：
  - 单次发送（发生错误时不会自动重新发送）
  - 自发自收（TWAI 控制器同时发送和接收报文）
- 配置数据接收滤波器（支持单滤波器和双滤波器模式）
- 支持错误检测与处理
  - 配置错误计数器
  - 可配置错误报警阈值
  - 内置错误代码记录
  - 内置仲裁丢失记录
  - 支持收发器自动待机功能

31.2 协议概述

31.2.1 TWAI 性能

TWAI 协议连接网络中的两个或多个节点，并允许各节点以确定性延迟的方式进行报文交互。TWAI 总线具有以下性能：

单通道通信与不归零编码： TWAI 总线只有一根传输线进行单通道通信，为半双工通信。同步调整也在单通道中进行，无需其他通道（如时钟通道和使能通道）。TWAI 上报文的位流采用不归零编码 (NRZ) 方式。

位值：单通道可处于显性状态或隐性状态，显性状态的逻辑值为 0，隐性状态的逻辑值为 1。发送显性状态数据的节点优先级高于发送隐性状态数据的节点。总线上的其他物理功能（如差分电平、单线）由其各自应用实现。

位填充： TWAI 报文的某些域已经过位填充。发送器在发送连续五个位的相同值（如显性数值或隐性数值）后，会自动插入一个互补位。同理，在接收五个相同值的连续位后，接收器应将下一个位视为填充位。位填充应用于以下域：SOF、仲裁域、控制域、数据域和 CRC 序列（参见章节 31.2.2）。

多播：当各节点连接到同个总线上时，这些节点都将接收到相同的位。如无总线错误（参见章节 31.2.3），各节点上的数据将保持一致。

多主机：任意节点都可发起数据传输。如果当前已有正在进行的数据传输，则节点将在当前传输结束后再发起新的数据传输。

报文优先级与仲裁：若两个或多个节点同时发起数据传输，TWAI 协议将确保其中一个节点获得总线的优先仲裁权。优先仲裁权由各节点所发送报文的仲裁域决定。

错误检测：各节点将积极检测总线上的错误，并向 TWAI 总线发送错误帧来广播检测到的错误。

故障限制：各节点都配置有错误计数器，计数器数量依据 TWAI 协议规则增加或减少。当错误计数超过一定阈值时，对应节点将自动关闭并退出网络。

可配置位速率：单个 TWAI 总线的位速率是可配置的。但是，同一总线中的所有节点须以相同位速率工作。

发送器与接收器：不论何时，任意 TWAI 节点都可作为发送器和接收器。

- 产生报文的节点即为发送器。在总线空闲或该节点失去仲裁之前，该节点将一直作为发送器。请注意，仲裁期间存在多个作为发送器的节点。
- 所有不是发送器的节点都可作为接收器。

31.2.2 TWAI 报文

TWAI 节点使用报文发送数据，并在监测到总线上存在错误时，向其他节点发送错误信号。报文有多种帧类型，不同的帧类型具有不同的帧格式。

TWAI 协议支持以下帧类型：

- 数据帧
- 远程帧
- 错误帧
- 过载帧
- 帧间距

TWAI 协议支持以下帧格式：

- 标准格式 (SFF)，由 11 位标识符组成
- 扩展格式 (EFF)，由 29 位标识符组成

31.2.2.1 数据帧和远程帧

节点使用数据帧向其他节点发送数据，可负载 0 ~ 8 字节数据。节点使用远程帧向其他节点请求具有相同标识符的数据帧，因此，远程帧中不包含任何数据字节。但是，数据帧和远程帧中包含许多相同域。下图 31-1 所示为不同帧类型和不同帧格式中包含的域和子域。

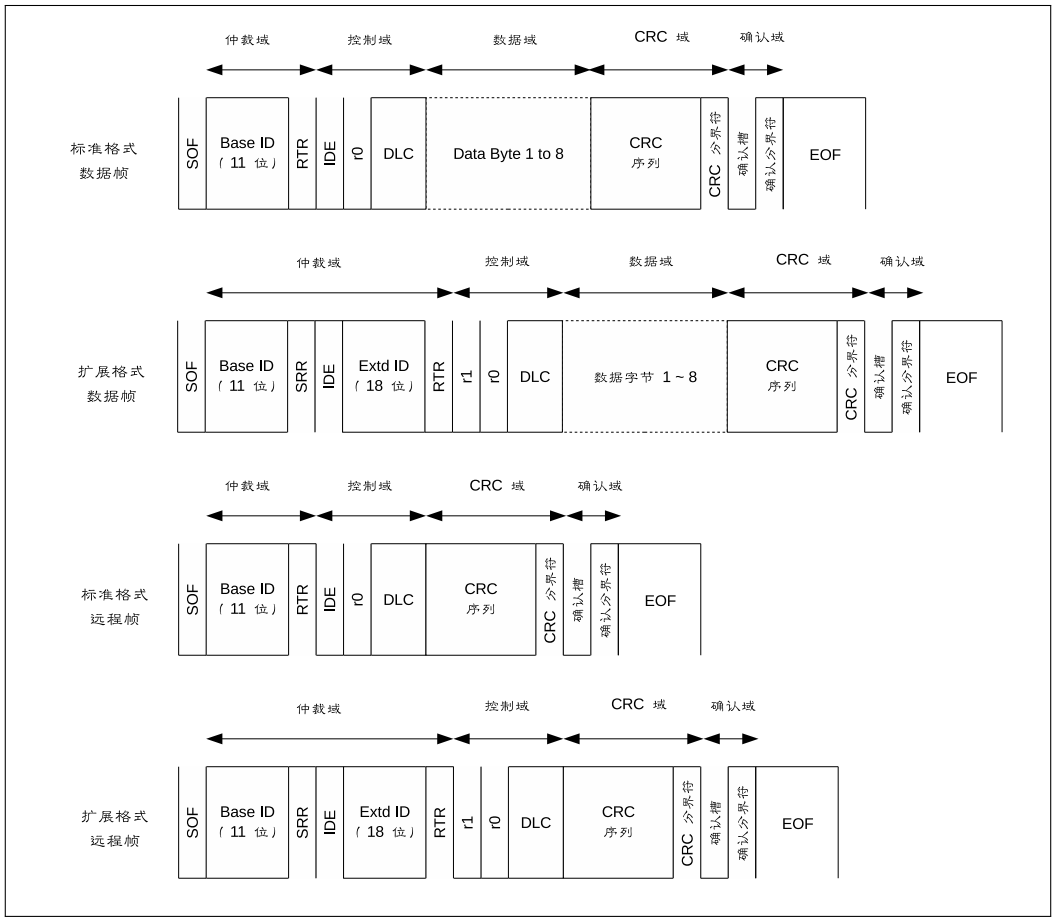


图 31-1. 数据帧和远程帧中的位域

仲裁域

当两个或多个节点同时发送数据帧和远程帧时，将根据仲裁域的位信息来决定总线上获得优先仲裁的节点。在发送仲裁域位信息时，如果一个节点在发送隐性位的同时检测到了一个显性位，表示有其他节点优先于这个隐性位。那么，这个发送隐性位的节点将丢失总线仲裁，应立即转为接收器。

仲裁域主要由获得最高优先发送权的帧标识符的有效位组成。根据显性位代表的逻辑值为 0，隐性位代表的逻辑值为 1，有以下规律:

- ID 值最小的帧将总是获得仲裁。
- 如果 ID 数值相同，由于数据帧的 RTR 位为显性位，数据帧将优先于远程帧。
- 如果 ID 的前 11 位相同，由于扩展帧的 SRR 位为隐形位，标准格式帧将优先于扩展格式帧。

控制域

控制域主要由数据长度代码 (DLC) 组成。DLC 表示一个数据帧中负载的数据字节长度，或一个远程帧请求的数据字节长度。DLC 优先发送长度数值的最高有效位。

数据域

数据域中包含一个数据帧真实负载的数据字节。远程帧中不包含数据域。

CRC 域

CRC 域主要由 CRC 序列组成。CRC 序列是一个 15 位的循环冗余校验编码，根据数据帧或远程帧中位填充前的内容（从 SOF 到数据域末尾的所有内容）计算而来。

确认域

确认 (ACK) 域由确认槽和确认分界符组成，主要功能在于接收器向发送器报告已正确接收到有效报文。

表 31-1. 不同帧类型、帧格式下的域及子域信息

数据/远程帧	描述
SOF	帧起始 (SOF) 是一个用于同步总线上节点的单个显性位。
Base ID	基标识符 (ID.28 ~ ID.18) 是 SFF 的 11 位标识符，或者是 EFF 中 29 位标识符的前 11 位。
RTR	远程发送请求位 (RTR) 显示当前报文是数据帧（显性）还是远程帧（隐性）。这意味着，当某个数据帧和一个远程帧有相同标识符时，数据帧始终优先于远程帧仲裁。
SRR	在 EFF 中发送替代远程请求位 (SRR)，以替代 SFF 中相同位置的 RTR 位。
IDE	标识符扩展位 (IED) 显示当前报文是 SFF （显性）还是 EFF（隐性）。这意味着，当某 SFF 帧和 EFF 帧有相同基标识符时，SFF 帧将始终优先于 EFF 帧仲裁。
Extd ID	扩展标识符 (ID.17 ~ ID.0) 是 EFF 中 29 位标识符的剩余 18 位。
r1	r1（保留位 1）始终是显性位。
r0	r0（保留位 0）始终是显性位。
DLC	数据长度代码 (DLC) 为 4 位，且为 0 ~ 8 中任一数值。数据帧使用 DLC 表示自身包含的数据字节长度。远程帧使用 DLC 表示从其他节点请求的数据字节长度。
数据字节	表示数据帧的数据负载量。该字节长度应与 DLC 的值匹配。首先发送数据字节的最高有效位 (MSB)。
CRC 序列	CRC 序列是一个 15 位的循环冗余校验编码。
CRC 分界符	CRC 分界符是紧随 CRC 序列的 1 位隐性位。
确认槽	确认槽用于接收器节点表示是否已成功接收数据帧或远程帧。发送器节点将在确认槽中发送一个隐性位，如果接收到的帧没有错误，则接收器节点将发送 1 位显性位替代隐性位。
确认分界符	确认分界符是紧随确认槽的 1 位隐性位。
EOF	帧结束 (EOF) 标志着数据帧或远程帧的结束，由 7 个隐性位组成。

31.2.2.2 错误帧和过载帧

错误帧

当某节点检测到总线错误时，将发送一个错误帧。错误帧由一个特殊的错误标志构成，该标志由某相同值的六个连续位组成，因而违反了位填充的规则。所以，当某节点检测到总线错误并发送错误帧时，其余节点也将相应地检测到一个填充错误并各自发送错误帧。也就是说，当发生总线错误时，通过上述过程，可将该报文传递至总线上的所有节点。

当某节点检测到总线错误时，该节点将于下一个位发送错误帧。不过，当总线错误类型为 CRC 错误时，错误帧将从确认分界符的下一个位开始（参见章节 31.2.3）。下图 31-2 所示为一个错误帧所包含的不同域：

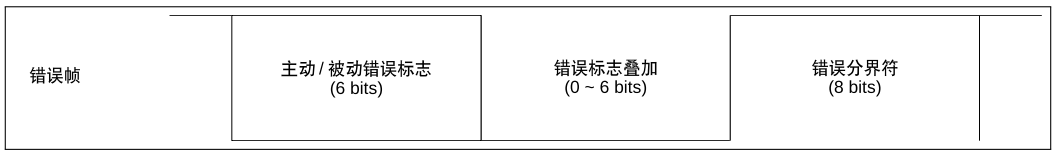


图 31-2. 错误帧中的位域

表 31-2. 错误帧中的位域信息

错误帧	描述
错误标志	错误标志包括两种形式: 主动错误标志和被动错误标志，主动错误标志由 6 个显性位组成，被动错误标志由 6 个隐性位组成（被其他节点的显性位优先仲裁时除外）。处于主动错误状态的节点发送主动错误标志，处于被动错误状态的节点发送被动错误标志。
错误标志叠加	错误标志叠加域的主要目的是允许总线上的其他节点发送各自的主动错误标志。叠加域的范围是 0 ~ 6 位，结束标志是检测到第一个隐性位（如检测到分界符上的第一个位时）。
错误分界符	分界符域标志着错误/过载帧结束，由 8 个隐性位构成。

过载帧

过载帧与包含主动错误标志的错误帧有着相同的位信息。二者区别在于触发发送过载帧的条件。下图 31-3 所示为过载帧中包含的位域:

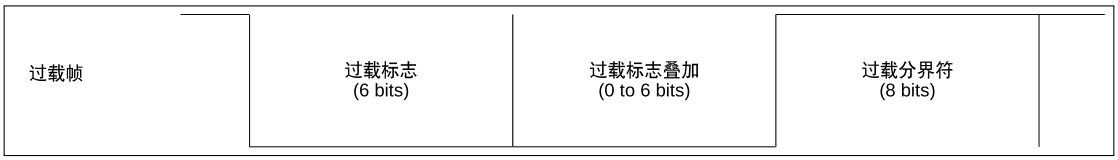


图 31-3. 过载帧中的位域

表 31-3. 过载帧中的位域信息

过载帧	描述
过载标志	由 6 个显性位构成，与主动错误标志相同。
过载标志叠加	允许其他节点发送过载标志的叠加，与错误标志叠加相似。
过载分界符	由 8 个隐性位构成，与错误分界符相同。

下列情况将触发发送过载帧:

1. 接收器内部要求延迟发送下一个数据帧或远程帧。
2. 在间歇域中的首个和第二个位上检测到显性位。
3. 如果在错误分界符的第八个（最后一个）位上检测到显性位。请注意，在这种情况下，TEC 和 REC 的值不会增加（参见章节 31.2.3）。

由于上述情况发送过载帧时，须满足以下规定：

- 第 1 条情况下，发送的过载帧只能在间歇域的第一个位开始。
- 第 2、3 条情况下，发送的过载帧须从检测到显性位的后一个位开始。
- 针对第 1 条情况，最多可生成两个过载帧。

31.2.2.3 帧间距

帧间距充当各帧之间的分隔符。数据帧和远程帧必须与前一帧用一个帧间距分隔开，不论前面的帧是何类型（数据帧、远程帧、错误帧、过载帧）。错误帧和过载帧则无需与前一个帧分隔开。

下图 31-4 所示为帧间距中包含的域：

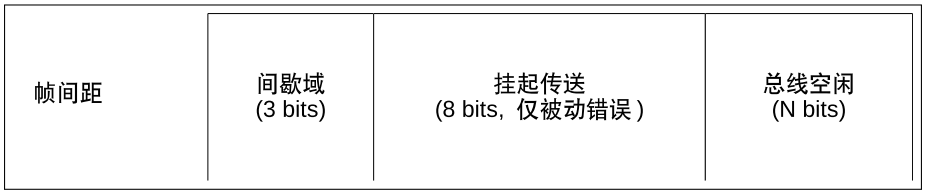


图 31-4. 帧间距中的域

表 31-4. 帧间距中的域信息

帧间距	描述
间歇域	间歇域由 3 个隐性位构成。
挂起传送	被动错误节点发送报文后，必须在帧间距中包含一个挂起传送域，由 8 个隐性位构成。主动错误节点中不含这个域。
总线空闲	总线空闲域长度任意。发送 SOF 时，总线空闲结束。若节点中有挂起传送，则 SOF 应在间歇域后的第一位发送。

31.2.3 TWAI 错误

31.2.3.1 错误类型

TWAI 中的总线错误包括以下类型：

位错误
当节点发送一个位值（显性位或隐性位）但检测到相反的位时（如发送显性位时检测到了隐性位），就会发生位错误。但是，如果发送的位是隐性位，且位于仲裁域、确认槽或被动错误标志中，那么此时检测到显性位则不会认定为位错误。

填充错误
当检测到相同值的六个连续位时（违反位填充的编码规则），发生填充错误。

CRC 错误
接收器将根据接收到的数据帧和远程帧的有效位（无填充位流）计算 CRC 值。当接收器计算的值与接收到的数据帧和远程帧中的 CRC 序列不匹配时，会发生 CRC 错误。

格式错误
当某个报文中的固定格式位中包含非法位时，可检测到格式错误。比如，r1 和 r0 域必须固定为显性。

确认错误
当发送器无法在确认槽中检测到显性位时，会发生确认错误。

31.2.3.2 错误状态

每个节点 TWAI 控制器通过维护两个错误计数器来实现故障界定，计数数值决定错误状态。这两个错误计数器分别为：发送错误计数 (TEC) 和接收错误计数 (REC)。TWAI 包含以下错误状态：

主动错误
处于主动错误状态的节点可参与到总线交互中，且在检测到错误时发送主动错误标志。

被动错误
处于被动错误状态的节点可参与到总线交互中，且在检测到错误时发送被动错误标志。被动错误节点发送数据帧或远程帧后，需在后续的帧间距中增加挂起传送域。

离线
禁止处于离线状态的节点以任意方式干扰总线（如不允许离线节点进行数据传输）。

31.2.3.3 错误计数

TEC 和 REC 根据以下规则递增/递减。**请注意，一条报文传输中可应用多个规则**。

1. 当接收器检测到错误时，REC 数值增加 1，检测到的错误为发送主动错误标志或过载标志期间的位错误除外。
2. 发送错误标志后，当接收器第一个检测到的位是显性位时，REC 数值增加 8。
3. 当发送器发送错误标志时，TEC 数值增加 8。但是，以下情况不适用于该规则：
   - 发送器处于被动错误状态，在检测到应答错误且发送被动错误标志时检测不到显性位时，TEC 数值不应增加。
   - 发送器在仲裁期间因填充错误而发送错误标志，且填充位本该是隐性位但是检测到显性位，TEC 数值不应增加。
4. 若发送器在发送主动错误标志和过载标志时检测到位错误，TEC 数值增加 8。
5. 若接收器在发送主动错误标志和过载标志时检测到位错误，REC 数值增加 8。
6. 任意节点在发送主动/被动错误标志或过载标志后，节点仅能承载最多 7 个连续显性位。在发送主动错误标志或过载标志时检测到第 14 个连续显性位，或在被动错误标志后检测到第 8 个连续显性位后，发送器的 TEC 数值增加 8，而接收器的 REC 数值增加 8。每一附加的 8 个连续显性位都会使发送器的 TEC 和接收器的 REC 数值增加 8。
7. 每当发送器成功发送报文后，即接收到 ACK，且直到 EOF 完成都未发生错误时，TEC 数值减小 1，除非 TEC 的数值已经为 0。
8. 当接收器成功接收报文后，即确认槽前未检测到错误，且已成功发送 ACK 时，REC 数值相应减小。
   - 若 REC 数值位于 1 ~ 127 之间，则其值减小 1。
   - 若 REC 数值大于 127，则其值减小到 127。
   - 若 REC 数值为 0，则仍保持为 0。

9. 当一个节点的 TEC 和/或 REC 数值大于等于 128 时，该节点变为被动错误节点。对于导致节点发生上述状态切换的错误，该节点仍发送主动错误标志。请注意，一旦 REC 数值到达 128，后续任何增加该值的动作都是无效的，直到 REC 数值返回到 128 以下。
10. 当某节点的 TEC 数值大于等于 256 时，该节点进入离线状态。
11. 当某被动错误节点的 TEC 和 REC 数值都小于等于 127 时，该节点变为主动错误节点。
12. 当离线节点在总线上检测到 128 次 11 个连续隐性位后，该节点可变为主动错误节点（TEC 和 REC 数值都重设为 0）。

31.2.4 TWAI 位时序

31.2.4.1 标称位

TWAI 协议允许 TWAI 总线以特定的位速率运行。但是，总线内的所有节点必须以统一位速率运行。

- **标称位速率**为每秒发送比特数量。
- **标称位时间**为 **1/标称位速率**。

每个标称位时间中含多个段，每段由多个时间定额 (Time Quanta) 组成。**时间定额**为最小时间单位，作为一种预分频时钟信号应用于各个节点中。下图 31-5 所示为一个标称位时间内所包含的段。

TWAI 控制器将在以一个时间定额的步长内进行操作，每个时间定额中都会分析 TWAI 的总线状态。如果两个连续的时间定额中总线状态不同（隐性-显性，或反之），意味着有边沿产生。PBS1 和 PBS2 的交点将被视为采样点，采样的总线状态即为这个位的数值。

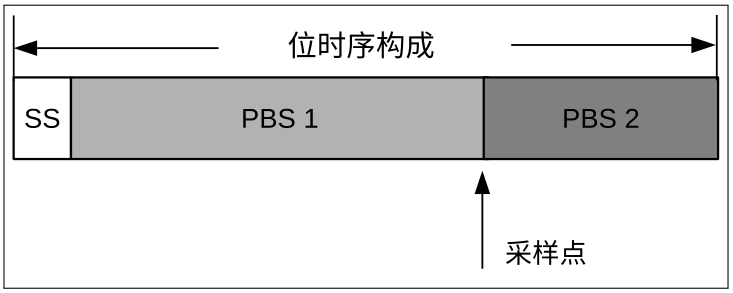


图 31-5. 标称位构成

表 31-5. 名义位时序中包含的段

段	描述
同步段 (SS)	SS（同步段）的长度为 1 个时间定额。若所有节点都同步，则位边沿发生时应位于该段内。
缓冲时期段 1 (PBS1)	PBS1 的长度可为 1 ~ 16 个时间定额，用于补偿网络中的物理延迟时间。可通过增加 PBS1 的长度来实现同步。
缓冲时期段 2 (PBS2)	PBS2 的长度可为 1 ~ 8 个时间定额，用于补偿节点中的信息处理时间。可通过缩短 PBS2 的长度来实现同步。

31.2.4.2 硬同步与再同步

由于时钟偏移和抖动，同一总线上节点的位时序可能会脱离相位段。因而，位边沿可能会偏移到同步段的前后。针对上述位边沿偏移的问题，TWAI 提供多种同步方式。设发生相位错误时位边沿偏移的时间定额数量为 **“e”**，

该值与 SS 相关。

- 主动相位错误 (e > 0)：位边沿位于同步段之后采样点之前，即边沿向后偏移。
- 被动相位错误 (e < 0)：位边沿位于前个位的采样点之后同步段之前，即边沿向前偏移。

为解决相位错误，有两种同步方式，**硬同步**与**再同步**。**硬同步**与**再同步**遵守以下规则：

- 单个位时序中仅可执行一次同步。
- 同步仅可发生在隐性位到显性位的边沿上。

硬同步

总线空闲期间，硬同步发生在隐性位到显性位的变化边沿上（如总线空闲后的第一个 SOF 位）。此时，所有节点都将重启其内部位时序，从而使该变化边沿位于重启位时序的同步段内。

再同步

非总线空闲期间，再同步发生在隐性位到显性位的变化边沿上。如果边沿上有主动相位错误 (e > 0)，则增加 PBS1 段的长度。如果边沿上有被动相位错误 (e < 0)，则减小 PBS2 段的长度。

PBS1/PBS2 具体增加和减小的时间定额取决于相位错误的绝对值，同时也受可配置的同步跳宽 (SJW) 数值限制。

- 当相位错误的绝对值小于等于 SJW 数值时，PBS1/PBS2 将增加/减小 **e** 个时间定额。该过程与硬同步具有相同效果。
- 当相位错误的绝对值大于 SJW 数值时，PBS1/PBS2 将增加/减小与 SJW 相同数值的时间定额。这意味着，在完全解决相位错误之前，可能需要执行多次位的再同步。

31.3 结构概述

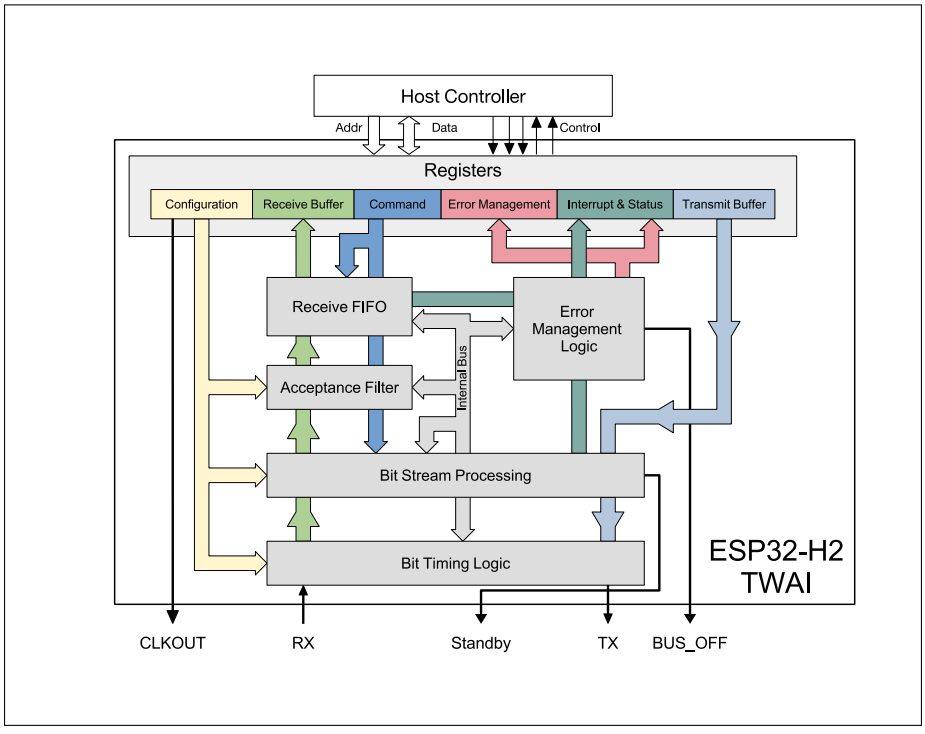


图 31-6. TWAI 概略图

TWAI 控制器的主要功能模块如图 31-6。

TWAI 控制器仅工作在 CORE 时钟域，CORE 时钟域的时钟源选择为 RC_FAST_CLK 或晶振时钟 XTAL_CLK。用户可以通过配置 PCR_TWAI0_FUNC_CLK_CONF_REG 选择时钟源。

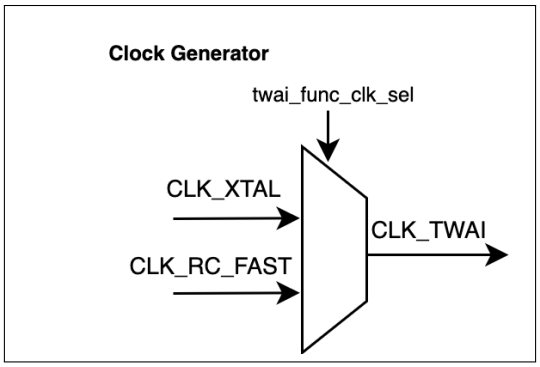


图 31-7. TWAI 时钟生成图

31.3.1 寄存器模块

ESP32-H2 的 CPU 使用 32 位字对齐地址访问外设。但对于 TWAI 控制器中的大部分寄存器，仅最低有效字节 (bits [7:0]) 数据有效。在这些寄存器中，bits [31:8] 数值在写入时被忽略，在读取时返回 0。

配置寄存器

配置寄存器存储 TWAI 控制器的各配置项，如位速率、操作模式、接收滤波器等。只有在 TWAI 控制器处于复位模式时，才可修改配置寄存器（参见章节 31.4.1）。

指令寄存器

CPU 通过指令寄存器驱动 TWAI 控制器执行任务，如发送报文或清除接收缓冲器。只有在 TWAI 控制器处于操作模式时，才可修改指令寄存器（参见章节 31.4.1）。

中断 & 状态寄存器

中断寄存器显示 TWAI 控制器中发生的事件（每个事件由一个单独的位表示）。状态寄存器显示 TWAI 控制器的当前状态。

错误管理寄存器

错误管理寄存器包括错误计数和捕捉寄存器。错误计数寄存器表示 TEC 和 REC 的数值。捕捉寄存器负责记录相关信息，如 TWAI 控制器在何处检测到总线错误或何时丢失仲裁。

发送缓冲寄存器

发送缓冲器大小为 13 字节，用于存储 TWAI 的待发送报文。

接收缓冲寄存器

接收缓冲器大小为 13 字节，用于存储单个报文。接收缓冲器是读取接收 FIFO 的窗口，接收 FIFO 中的第一个报文将被映射到接收缓冲器中。

请注意，发送缓冲寄存器、接收缓冲寄存器和接收滤波配置寄存器共享同一段地址偏移 0x0040 ~ 0x0070。CPU 访问上述地址范围内寄存器遵循以下规则：

- 当 TWAI 控制器处于复位模式时，该地址范围被映射到接收滤波配置寄存器。
- TWAI 控制器处于操作模式时：
  - 对地址范围的所有读取都映射到接收缓冲寄存器中。
  - 对地址范围的所有写入都映射到发送缓冲寄存器中。

31.3.2 位流处理器

位流处理 (BSP) 模块负责对发送缓冲器的数据进行帧处理（如位填充和附加 CRC 域），并为位时序逻辑 (BTL) 模块生成位流。同时，BSP 模块还负责处理从 BTL 模块中接收的位流（如去填充和验证 CRC），并将处理后报文写入接收 FIFO。BSP 还负责检测 TWAI 总线上的错误并将此类错误报告给错误管理逻辑 (EML)。

31.3.3 错误管理逻辑

错误管理逻辑 (EML) 模块负责更新 TEC 和 REC 数值、记录错误信息（如错误类型和错误位置）、更新控制器的错误状态，确保 BSP 模块发送正确的错误标志。此外，该模块还负责记录 TWAI 控制器丢失仲裁时的 bit 位置。

31.3.4 位时序逻辑

位时序逻辑 (BTL) 模块负责以预先配置的位速率发送和接受报文。BTL 模块还负责同步位时序，确保数据传输的稳定性。位速率由多个可编程的段组成，且用户可设置每个段的时间定额长度，来调整传播延迟、控制器处理时间等因素。

31.3.5 接收滤波器

接收滤波器是一个可编程的报文过滤单元，允许 TWAI 控制器根据报文的标识符域接收或拒绝该报文。通过接收滤波器的报文才能被存储到接收 FIFO 中。用户可配置接收滤波器的模式：单滤波器、双滤波器。

31.3.6 接收 FIFO

接收 FIFO 位于 TWAI 控制器内部，是大小为 64 字节的缓冲器，负责存储通过接收滤波器的接收报文。接收 FIFO 中存储的报文大小可以不同（范围在 3 ~ 13 byte 之间）。当接收 FIFO 为满或剩余空间不足以完全存储下当前的接收报文时，将触发溢出中断，后续的接收报文将丢失，直到接收 FIFO 中清除出足够的存储空间。接收 FIFO 中的第一条报文将被映射到 13 字节的接收缓冲器中，直到该报文被清除（通过释放接收缓冲器指令）。清除后，接收 FIFO 将释放上一条已清除报文的空间，同时将窗口映射到接收 FIFO 中的下一条报文。

31.4 功能描述

31.4.1 模式

ESP32-H2 TWAI 控制器有两种工作模式：复位模式和操作模式。将 TWAI_RESET_MODE 位置 1，进入复位模式；置 0，进入操作模式。

31.4.1.1 复位模式

要修改 TWAI 控制器的各种配置寄存器，需进入复位模式。进入复位模式时，TWAI 控制器彻底与 TWAI 总线断开连接。复位模式下，TWAI 控制器将无法发送任何报文（包括错误信号）。任何正在进行的报文传输将立即被终止。同样的，TWAI 控制器在该模式下也将无法接收任何报文。

31.4.1.2 操作模式

进入操作模式后，TWAI 控制器与总线相连，并且写保护各配置寄存器，以确保控制器的配置在运行期间保持一致。操作模式下，TWAI 控制器可以发送和接收报文（包括错误信号），但具体取决于 TWAI 控制器配置于哪种运行子模式。TWAI 控制器支持以下三种子模式：

- **正常模式：** TWAI 控制器可以发送和接收包含错误信号在内的报文（如，错误帧和过载帧）。
- **自测模式：** 与正常模式相同，但在该模式下，TWAI 控制器发送报文时，即使在 CRC 域之后没有接收到应答信号，也不会产生应答错误。通常在单个 TWAI 控制器自测时使用该模式。
- **只听模式：** TWAI 控制器可以接收报文，但在 TWAI 总线上保持完全被动。因此，TWAI 控制器将无法发送任何报文、应答或错误信号。错误计数寄存器将保持冻结状态。该模式用于 TWAI 总线监控。

请注意，退出复位模式后（如，进入操作模式时），TWAI 控制器需等待 11 个连续隐性位出现，才能完全连接上 TWAI 总线（即，可以发送或接收报文）。

31.4.2 位时序

TWAI 控制器的工作位速率必须在控制器处于复位模式时进行配置。在寄存器 TWAI_BUS_TIMING_0_REG 和 TWAI_BUS_TIMING_1_REG 中配置位速率，这两个寄存器包含以下域：

下表 31-6 所示为 TWAI_BUS_TIMING_0_REG 包含的位域。TWAI 工作时钟有多个时钟源，用户可以根据需要进行配置，详细配置说明请参见章节 7 复位和时钟。

表 31-6. TWAI_BUS_TIMING_0_REG 的位信息 (0x18)

Bit 31-16	Bit 15	Bit 14	Bit 13	Bit 12	.....	Bit 1	Bit 0
保留	SJW.1	SJW.0	BRP.13	BRP.12	.....	BRP.1	BRP.0

说明：

- 预分频值 (BRP)：TWAI 时间定额时钟由 XTAL 时钟（数值可配，默认是 40 Mhz）分频得到。可通过以下公式计算分频数值，其中 t_{Tq} 为时间定额的时钟周期，t_{CLK} 为 TWAI 工作时钟周期：
  $t_{Tq} = 2 \times t_{CLK} \times (2^{13} \times BRP.13 + 2^{12} \times BRP.12 + 2^{11} \times BRP.11 + ... + 2^{1} \times BRP.1 + 2^{0} \times BRP.0 + 1)$
- 同步跳宽 (SJW)：SJW 数值在 SJW.0 和 SJW.1 中配置，计算公式为：SJW = (2 x SJW.1 + SJW.0 + 1)。

下表 31-7 所示为 TWAI_BUS_TIMING_1_REG 包含的位域。

表 31-7. TWAI_BUS_TIMING_1_REG 的位 (0x1c)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	SAM	PBS2.2	PBS2.1	PBS2.0	PBS1.3	PBS1.2	PBS1.1	PBS1.0

说明：

- PBS1: 根据以下公式计算缓冲时期段 1 中的时间定额数量：(8 x PBS1.3 + 4 x PBS1.2 + 2 x PBS1.1 + PBS1.0 + 1)。
- PBS2: 根据以下公式计算缓冲时期段 2 中的时间定额数量：(4 x PBS2.2 + 2 x PBS2.1 + PBS2.0 + 1)。
- SAM: 该值置 1 启动三点采样。用于低/中速总线，有利于过滤总线上的尖峰信号。

31.4.3 中断管理

ESP32-H2 TWAI 控制器提供了八种中断，每种中断由寄存器 TWAI_INT_ST_REG 中的一个位表示。要触发某个特定的中断，须设置 TWAI_INT_ENA_REG 中相应的使能位。

TWAI 控制器提供了以下八种中断：

- 接收中断
- 发送中断

- 错误报警中断
- 数据溢出中断
- 被动错误中断
- 仲裁丢失中断
- 总线错误中断
- 总线空闲状态中断

只要在 TWAI_INT_ST_REG 一个或多个中断位为 1，TWAI 控制器中的中断信号即为有效，当 TWAI_INT_ST_REG 中的所有位都被清除时，TWAI 控制器中的中断信号则失效。读操作访问寄存器 TWAI_INT_ST_REG 后，其中的大多数中断位将自动清除。但是，只有通过 TWAI_RELEASE_BUF 指令位清除所有接收报文后，接收中断位才能被清除。

31.4.3.1 接收中断 (RXI)

当 TWAI 接收 FIFO 中有待读取报文时（TWAI_RX_MESSAGE_CNT_REG > 0），都会触发 RXI。TWAI_RX_MESSAGE_CNT_REG 中记录的报文数量包括接收 FIFO 中的有效报文和溢出报文。直到通过 TWAI_RELEASE_BUF 指令位清除所有挂起接收报文后，RXI 才会失效。

31.4.3.2 发送中断 (TXI)

当发送缓冲器由锁定状态变为为空闲状态，将会触发 TXI。此时软件可以将其他报文加载到发送缓冲器中等待发送。以下几种情况都会触发该中断：

- 报文发送已成功完成（如，应答未发现错误）。任何发送失败将自动重发。
- 单次发送已完成（TWAI_TX_COMPLETE 位指示发送成功与否）。
- 使用 TWAI_ABORT_TX 指令位终止报文发送。

31.4.3.3 错误报警中断 (EWI)

每当寄存器 TWAI_STATUS_REG 中 TWAI_ERR_ST 和 TWAI_BUS_OFF_ST 的位值改变时（如，从 0 变为 1 或反之），都会触发 EWI。根据 EWI 触发时 TWAI_ERR_ST 和 TWAI_BUS_OFF_ST 的值分成以下几种情况：

- 如果 TWAI_ERR_ST = 0 且 TWAI_BUS_OFF_ST = 0：
  - 如果 TWAI 控制器处于主动错误状态，则表示 TEC 和 REC 的值都返回到了 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值之下。
  - 如果 TWAI 控制器此前正处于总线恢复状态，则表示此时总线恢复已成功完成。
- 如果 TWAI_ERR_ST = 1 且 TWAI_BUS_OFF_ST = 0：表示 TEC 或 REC 数值已超过 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值。
- 如果 TWAI_ERR_ST = 1 且 TWAI_BUS_OFF_ST = 1：表示 TWAI 控制器已进入 BUS_OFF 状态（因 TEC >= 256）。
- 如果 TWAI_ERR_ST = 0 且 TWAI_BUS_OFF_ST = 1：表示 BUS_OFF 恢复期间，TWAI 控制器的 TEC 数值已低于 TWAI_ERR_WARNING_LIMIT_REG 所设的阈值。

31.4.3.4 数据溢出中断 (DOI)

每当接收 FIFO 中有溢出发生时，都会触发 DOI。DOI 表示接收 FIFO 已满且应立即进行读取，以防出现更多溢出报文。

只有导致接收 FIFO 溢出的第一条报文可触发 DOI（如，当接收 FIFO 从未满变为开始溢出时）。任意后续的溢出报文将不会再次重复触发 DOI。只有当所有接收的（有效报文或溢出）报文都被读取后，才能再次触发 DOI。

31.4.3.5 被动错误中断 (EPI)

每当 TWAI 控制器从主动错误变为被动错误，或反之时，都会触发 EPI。

31.4.3.6 仲裁丢失中断 (ALI)

每当 TWAI 控制器尝试发送报文且丢失仲裁时，都会触发 ALI。TWAI 控制器丢失仲裁的 bit 位置将自动记录在仲裁丢失捕捉寄存器 (TWAI_ARB_LOST_CAP_REG) 中。仲裁丢失捕捉寄存器被清除（通过 CPU 读取该寄存器）之前，将不会再记录新发生的仲裁失败时的 bit 位置。

31.4.3.7 总线错误中断 (BEI)

每当 TWAI 控制器在 TWAI 总线上检测到错误时，都会触发 BEI。发生总线错误时，总线错误的类型和发生错误时的 bit 位置都将自动记录在错误捕捉寄存器 (TWAI_ERR_CODE_CAP_REG) 中。错误捕捉寄存器被清除（通过 CPU 的读取）之前，将不会再记录新的总线错误信息。

31.4.3.8 总线空闲状态中断 (BSI)

当 TWAI 控制器进入空闲状态的时钟周期数超过预先设配置的 TWAI_IDLE_INTR_CNT_REG 寄存器内的数值时，将产生总线空闲状态中断。用户可以通过配置该中断来获知 TWAI 控制器空闲状态，进一步决定是否关闭外部 TWAI 接收器来降低整体功耗（参见31.4.10）。

31.4.4 发送缓冲器与接收缓冲器

31.4.4.1 缓冲器概述

表 31-8. SFF 与 EFF 的缓冲器布局

标准格式 (SFF)		扩展格式 (EFF)	
偏移地址	内容	偏移地址	内容
0x40	TX/RX 帧信息	0x40	TX/RX 帧信息
0x44	TX/RX identifier 1	0x44	TX/RX identifier 1
0x48	TX/RX identifier 2	0x48	TX/RX identifier 2
0x4c	TX/RX data byte 1	0x4c	TX/RX identifier 3
0x50	TX/RX data byte 2	0x50	TX/RX identifier 4
0x54	TX/RX data byte 3	0x54	TX/RX data byte 1
0x58	TX/RX data byte 4	0x58	TX/RX data byte 2
0x5c	TX/RX data byte 5	0x5c	TX/RX data byte 3

见下页

表 31-8 – 接上页

标准格式 (SFF)		扩展格式 (EFF)	
偏移地址	内容	偏移地址	内容
0x60	TX/RX data byte 6	0x60	TX/RX data byte 4
0x64	TX/RX data byte 7	0x64	TX/RX data byte 5
0x68	TX/RX data byte 8	0x68	TX/RX data byte 6
0x6c	保留	0x6c	TX/RX data byte 7
0x70	保留	0x70	TX/RX data byte 8

表 31-8 所示为发送缓冲器和接收缓冲器的寄存器布局。发送和接收缓冲寄存器的访问地址范围相同，且只有当 TWAI 控制器处于操作模式时才可访问。CPU 的写入操作将访问发送缓冲寄存器，CPU 的读取操作将访问接收缓冲寄存器。发送缓冲器和接受缓冲器中存储报文（接收报文或待发送报文）的寄存器布局和域完全一致。

发送缓冲寄存器用于配置 TWAI 的待发送报文。CPU 可以在发送缓冲寄存器进行写入操作，指定报文的帧类型、帧格式、帧 ID 和帧数据（有效载荷）。一旦发送缓冲器配置完成后，CPU 可以将 TWAI_CMD_REG 中的 TWAI_TX_REQ 位置 1，以开始报文发送。

- 若是自发自收请求，变更为将 TWAI_SELF_RX_REQ 置 1。
- 若是单次发送，需要同时将 TWAI_TX_REQ 和 TWAI_ABORT_TX 置 1。

接收缓冲寄存器将映射接收 FIFO 中的第一条报文。CPU 可以对接收缓冲寄存器执行读取操作，获取第一条报文的帧类型、帧格式、帧 ID 和帧数据（有效载荷）。读取完接收缓冲寄存器中的报文后，CPU 通过将 TWAI_CMD_REG 中的 TWAI_RELEASE_BUF 位置 1 来清除接收缓冲寄存器，若接收 FIFO 中仍有待处理的报文，按照接收报文的先后次序依次将接收到下一个的报文映射到接收缓冲寄存器。

31.4.4.2 帧信息

帧信息的长度为 1-byte，主要用于明确报文的帧类型、帧格式以及数据长度。下表 31-9 所示为帧信息域。

表 31-9. TX/RX 帧信息 (SFF/EFF)；TWAI 地址 0x40

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	FF	RTR	X	X	DLC.3	DLC.2	DLC.1	DLC.0

说明：

1. FF: 主要明确某报文属于 EFF 还是 SFF。当 FF 位为 1 时，该报文为 EFF，当 FF 位为 0 时，该报文为 SFF。
2. RTR: 主要明确某报文是数据帧还是远程帧。当 RTR 位为 1 时，该报文为远程帧，当 RTR 位为 0 时，该报文为数据帧。
3. X: 无关位，可以是任意值。
4. DLC: 主要明确数据帧中的数据字节数量，或从远程帧中请求的数据字节数量。TWAI 数据帧的最大载荷为 8 个数据字节，因此 DLC 的数值范围应是 0 ~ 8。

31.4.4.3 帧标识符

若报文为 SFF，则对应的帧标识符域占据 2 字节（11 位）；若报文为 EFF，则对应的帧标识符域占据 4 字节（29 位）。

下表 Table 31-10 ~ 31-11 所示为 SFF（11 位）报文的帧标识符域。

表 31-10. TX/RX 标识符 1 (SFF); TWAI 地址 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3

表 31-11. TX/RX 标识符 2 (SFF); TWAI 地址 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]	X[2]	X[2]

说明：

1. 无关项。建议设置为与接收缓冲器兼容（设为 RTR），以防需使用自接收功能（或与自接收功能一起使用）。
2. 无关项。建议设置为与接收缓冲器兼容（设为 0），以防需使用自接收功能（或与自接收功能一起使用）。

下表 31-12 ~ 31-15 所示为 EFF（29 位）报文的帧标识符域。

表 31-12. TX/RX 标识符 1 (EFF); TWAI 地址 0x44

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21

表 31-13. TX/RX 标识符 2 (EFF); TWAI 地址 0x48

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13

表 31-14. TX/RX 标识符 3 (EFF); TWAI 地址 0x4c

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5

表 31-15. TX/RX 标识符 4 (EFF); TWAI 地址 0x50

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ID.4	ID.3	ID.2	ID.1	ID.0	X[1]	X[2]	X[2]

说明：

1. 无关项。建议设置为与接收缓冲器兼容（设为 RTR），以防需使用自接收功能（或与自接收功能一起使用）。
2. 无关项。建议设置为与接收缓冲器兼容（设为 0），以防需使用自接收功能（或与自接收功能一起使用）。

31.4.4.4 帧数据

帧数据域包含发送或接收的数据帧，范围为 0 ~ 8 bytes。其中的有效字节数应与 DLC 相同。但是，如果 DLC 数值大于 8，则帧数据域的有效字节数仍为 8。远程帧中不包含数据载荷，因此不存在帧数据域。

比如，当发送 5 个字节的数据帧时，CPU 应在 DLC 域中写入数值 5，并将数据写入数据域 1 ~ 5 字节对应的寄存器。同样，当接收 DLC 为 5 的数据帧时，只有 1 ~ 5 数据字节中包含 CPU 可以读取的有效载荷数据。

31.4.5 接收 FIFO 和数据溢出

接收 FIFO 是一个 64-byte 的内部缓冲器，用于以先进先出的原则存储接收到的报文。一条接收报文可在接收 FIFO 中占 3 ~ 13 bytes 空间，且其中字节序与接收缓冲器的寄存器地址顺序相同。接收缓冲寄存器将被映射到接收 FIFO 中第一条报文。

当 TWAI 控制器接收到一条报文时，TWAI_RX_MESSAGE_COUNTER 的值将增加 1，最大值为 64。如果接收 FIFO 中有足够的剩余空间，报文内容将被写入到接收 FIFO 中。读取接收缓冲器中的消息后，通过将 TWAI_RELEASE_BUF 的位置 1，释放接收 FIFO 第一条报文所占的空间，TWAI_RX_MESSAGE_COUNTER 的值也将减小 1。然后，接收缓冲器将映射接收 FIFO 中的下一条报文。

当 TWAI 控制器接收到一条报文，但接收 FIFO 没有足够空间完整地存储这条接收报文时（不论是因为报文内容大小大于接收 FIFO 中的空闲空间，还是因为接收 FIFO 已满），便会发生数据溢出。

数据溢出发生时：

- 接收 FIFO 中剩余的空间将填满溢出报文的内容。如果接收 FIFO 已满，则无法存储溢出报文的任何内容。
- 接收 FIFO 首次发生数据溢出时，将触发数据溢出中断。
- 溢出报文仍将增加 TWAI_RX_MESSAGE_COUNTER 的值到最大值 64。
- 接收 FIFO 将在内部将溢出报文标记为无效。可使用 TWAI_MISS_ST 位，确认目前接收缓冲器映射的报文是有效报文还是溢出报文。

为了清除接收 FIFO 中的溢出报文，应重复调用 TWAI_RELEASE_BUF，直到 TWAI_RX_MESSAGE_COUNTER 为 0。这就要求用户读取接收 FIFO 中的所有有效报文，并清除所有溢出报文。

31.4.6 接收滤波器

接收滤波器允许 TWAI 控制器根据报文 ID 过滤接收报文（甚至可以过滤报文的第一个数据字节和帧类型）。只有通过过滤的报文才能存储到接收 FIFO 中。接收滤波器的使用可以一定程度地减轻 TWAI 控制器的运行负荷（如，可减少使用接收 FIFO 和发生接收中断的次数），因为 TWAI 控制器将只需要操作过滤后的一小部分报文。

接收滤波的配置寄存器和发送/接收缓冲寄存器使用相同的访问地址空间，只有当 TWAI 控制器处于复位模式时，该地址段才被映射到接收滤波器的配置寄存器。

接收滤波器的配置寄存器由 32 位的 Code 值和 32 位的 Mask 值组成。Code 值将指定一种位排列模式，每条过滤报文中的位都必须匹配该模式，才能使该报文通过过滤。Mask 值可屏蔽 Code 值中的某些位（屏蔽位将被设置为“不相关”位）。如图 31-8 所示，为了使报文通过过滤，每条过滤报文的 ID 位都必须匹配 Code 值所设模式或者被 Mask 值屏蔽。

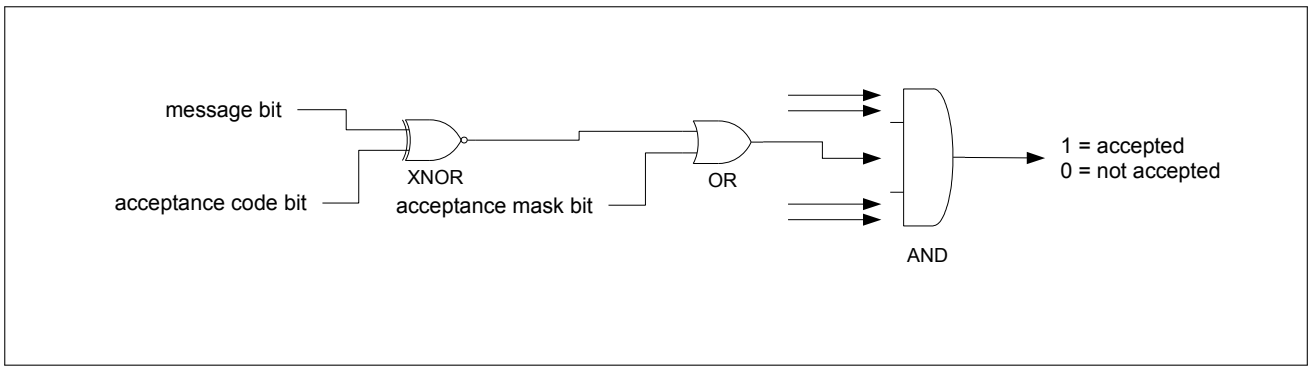


图 31-8. 接收滤波器

TWAI 控制器的接收滤波器允许 32 位的 Code 值和 Mask 值定义单个滤波器（单滤波模式），或两个滤波器（双滤波模式）。接收滤波器如何解析 32 位的 Code 值和 Mask 值，取决于滤波模式以及接收报文的格式（如，SFF 还是 EFF）。

31.4.6.1 单滤波模式

将 TWAI_RX_FILTER_MODE 的位置 1，可启动单滤波模式。此后，32 位 Code/Mask 的值将定义单个滤波器。

单个滤波器可过滤数据帧和远程帧中的以下位：

- SFF
  - 11 位 ID 整体
  - RTR 位
  - 数据字节 1 和数据字节 2
- EFF
  - 29 位 ID 整体
  - RTR 位

下图 31-9 所示为单滤波模式下如何解析 32 位 Code/Mask 的值。

31.4.6.2 双滤波模式

将 TWAI_RX_FILTER_MODE 的位置 0，可启动双滤波模式。此后，32 位 Code/Mask 的值将定义两个滤波器之一，即滤波器 1 或滤波器 2。双滤波模式下，如果报文通过这两个滤波器中的至少一个，则表示该报文已成功通过过滤。

这两个滤波器可以过滤数据帧和远程帧中的以下位：

- SFF
  - 11 位 ID 整体
  - RTR 位
  - 数据字节 1 (仅适用于滤波器 1)
- EFF

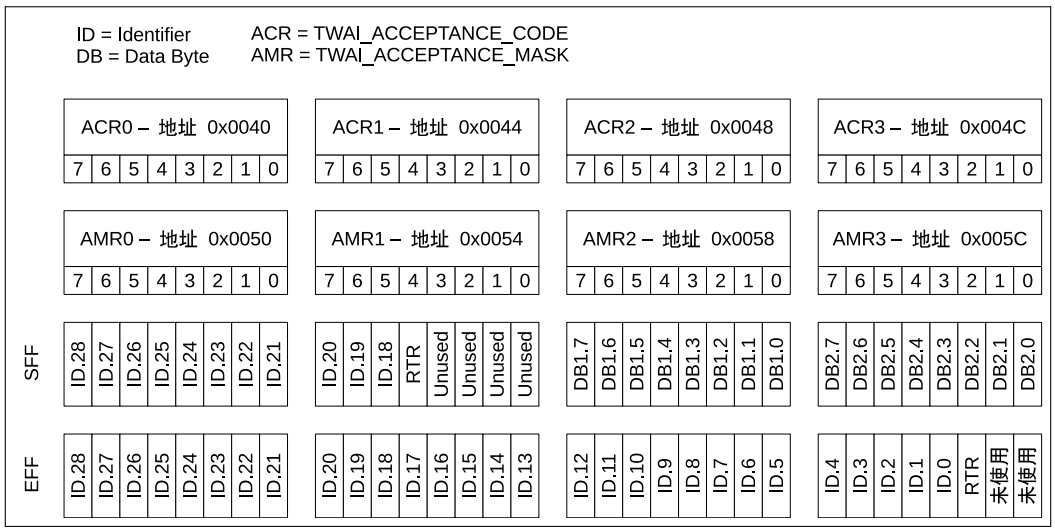


图 31-9. 单滤波模式

– 29 位 ID 的前 16 位

下图 31-10 所示为双滤波模式下如何解析 32 位 Code/Mask 的值。

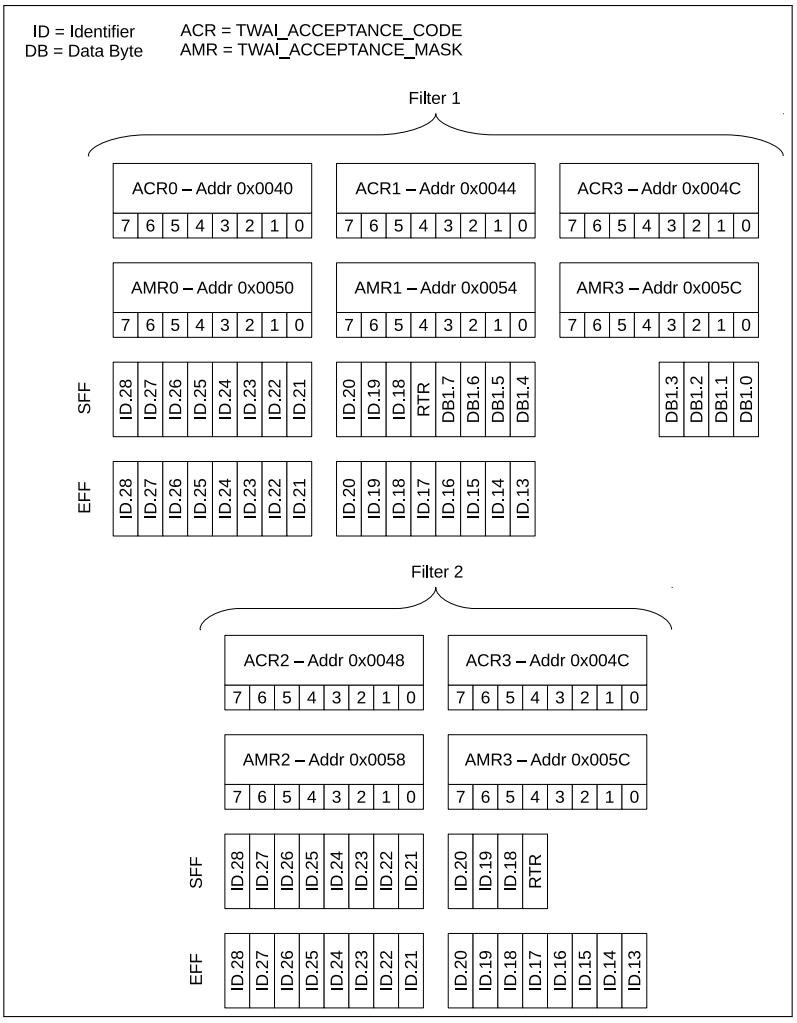


图 31-10. 双滤波模式

31.4.7 错误管理

TWAI 协议要求每个 TWAI 节点中都包含发送错误计数器 (TEC) 和接收错误计数器 (REC)。这两个错误计数的数值决定了 TWAI 控制器当前的错误状态（如，主动错误、被动错误、离线）。TWAI 控制器将 TEC 和 REC 的数值分别存储在 TWAI_TX_ERR_CNT_REG 和 TWAI_RX_ERR_CNT_REG 中，CPU 可随时进行读取。除了错误状态之外，TWAI 控制器还提供错误报警限制 (EWL) 的功能，这个功能可在 TWAI 控制器进入被动错误状态之前，提醒用户当前发生的严重总线错误。

TWAI 控制器的当前错误状态通过以下各数值和状态位体现，即：TEC、REC、TWAI_ERR_ST 和 TWAI_BUS_OFF_ST。这些数值和状态位的变化也将触发中断，从而提醒用户当前的错误状态变化（可参见第 31.4.3 章）。下图 31-11 所示为错误状态、上述数值和状态位以及错误状态相关中断之间的关系。

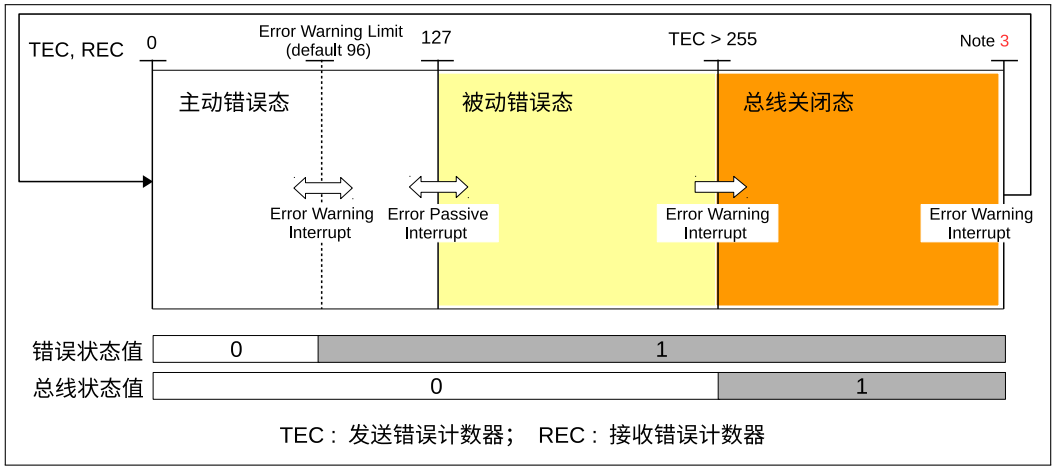


图 31-11. 错误状态变化

31.4.7.1 错误报警限制

错误报警限制 (EWL) 为 TEC 和 REC 的可配置阈值，若错误计数数值超过该阈值，将触发 EWI 中断。EWL 将作为一个报警功能提示当前发生的严重 TWAI 总线错误，且在 TWAI 控制器进入被动错误状态之前被触发。EWL 数值应在寄存器 TWAI_ERR_WARNING_LIMIT_REG 中进行配置，配置同时 TWAI 控制器必须处于复位模式下。TWAI_ERR_WARNING_LIMIT_REG 默认数值为 96。

当 TEC 和/或 REC 数值大于等于 EWL 数值时，TWAI_ERR_ST 位将立即被置 1。同理，当 TEC 和 REC 数值都小于 EWL 数值时，TWAI_ERR_ST 位将立即复位为 0。只要 TWAI_ERR_ST（或 TWAI_BUS_OFF_ST）位值发生变化，便会触发错误报警中断。

31.4.7.2 被动错误

当 TEC 或 REC 数值大于 127 时，TWAI 控制器处于被动错误状态。同理，当 TEC 和 REC 数值都小于等于 127 时，TWAI 控制器进入主动错误状态。每当 TWAI 控制器从主动错误状态变为被动错误状态，或反之时，都将触发被动错误中断。

31.4.7.3 离线状态与离线恢复

当 TEC 数值大于 255 时，TWAI 控制器将进入离线状态。进入离线状态后，TWAI 控制器将自动进行以下动作：

- REC 数值置为 0

- TEC 数值置为 127
- TWAI_BUS_OFF_ST 位置 1
- 进入复位模式

每当 TWAI_BUS_OFF_ST 位（或 TWAI_ERR_ST 位）数值发生变化时，都将触发错误报警中断。

为了返回主动错误状态，TWAI 控制器必须进行离线恢复。要启动离线恢复，首先需要退出复位模式，进入操作模式。然后要求 TWAI 控制器在总线上检测到 128 次 11 个连续隐性位。

每一次 TWAI 控制器检测到 11 个连续隐性位时，TEC 数值都将减小，以追踪离线恢复进程。当离线恢复完成后（TEC 数值从 127 减小到 0），TWAI_BUS_OFF_ST 位将自动复位为 0，从而触发错误报警中断。

31.4.8 错误捕捉

错误捕捉 (ECC) 功能允许 TWAI 控制器以错误代码的形式记录 TWAI 总线错误的错误类型和 bit 位置。当检测到一个 TWAI 总线错误时，总线错误中断将被触发，相应的错误代码将记录在 TWAI_ERR_CODE_CAP_REG 中。寄存器 TWAI_ERR_CODE_CAP_REG 中存储的当前错误代码被读取之前，后续的总线错误中断触发时，将不会再记录错误代码。

下表 31-16 所示为寄存器 TWAI_ERR_CODE_CAP_REG 中的域：

表 31-16. TWAI_ERR_CODE_CAP_REG 中的位信息 (0x30)

Bit 31-8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	ERRC.1	ERRC.0	DIR	SEG.4	SEG.3	SEG.2	SEG.1	SEG.0

说明：

- 错误代码 (ERRC)：表示总线错误的类型。00 代表位错误，01 代表格式错误，10 代表填充错误，11 代表其他错误类型。
- 传输方向 (DIR)：表示总线错误发生时，TWAI 控制器处于发送器状态还是接收器状态。0 代表发送器，1 代表接收器。
- 错误段 (SEG)：表示总线错误发生在 TWAI 报文的哪个段。

下表 31-17 所示为 SEG.0 ~ SEG.4 的位信息。

表 31-17. SEG.4 - SEG.0 的位信息

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	描述
0	0	0	1	1	帧起始
0	0	0	1	0	ID.28 ~ ID.21
0	0	1	1	0	ID.20 ~ ID.18
0	0	1	0	0	bit SRTR
0	0	1	0	1	bit IDE
0	0	1	1	1	ID.17 ~ ID.13
0	1	1	1	1	ID.12 ~ ID.5
0	1	1	1	0	ID.4 ~ ID.0
0	1	1	0	0	bit RTR
0	1	1	0	1	保留位 1
0	1	0	0	1	保留位 0

见下页

表 31-17 – 接上页

Bit SEG.4	Bit SEG.3	Bit SEG.2	Bit SEG.1	Bit SEG.0	描述
0	1	0	1	1	数据长度代码
0	1	0	1	0	数据域
0	1	0	0	0	CRC 序列
1	1	0	0	0	CRC 分界符
1	1	0	0	1	确认槽
1	1	0	1	1	确认分界符
1	1	0	1	0	帧结束
1	0	0	1	0	间歇域
1	0	0	0	1	主动错误标志
1	0	1	1	0	被动错误标志
1	0	0	1	1	兼容显性位
1	0	1	1	1	错误分界符
1	1	1	0	0	过载标志

说明：

- Bit SRTR: 标准格式 RTR bit。
- Bit IDE: 标识符扩展位。0 表示标准格式。

31.4.9 仲裁丢失捕捉

仲裁丢失捕捉 (ALC) 功能允许 TWAI 控制器记录丢失仲裁的 bit 位置。当 TWAI 控制器丢失仲裁时，bit 位置将被记录在寄存器 TWAI_ARB_LOST_CAP_REG 中，同时触发仲裁丢失中断。

后续的仲裁丢失中断触发时，bit 位置将不会被记录在 TWAI_ARB_LOST_CAP_REG 中，直到 TWAI_ERR_CODE_CAP_REG 中的当前仲裁丢失捕捉被读取。

下表 31-18 所示为 TWAI_ERR_CODE_CAP_REG 中的位域；下图 31-12 所示为一条 TWAI 报文的 bit 位置。

表 31-18. TWAI_ARB_LOST_CAP_REG 中的位信息 (0x2c)

Bit 31-5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
保留	BITNO.4	BITNO.3	BITNO.2	BITNO.1	BITNO.0

说明：

- 位号 (BITNO)：表示丢失仲裁的 TWAI 报文的第 n 个位。

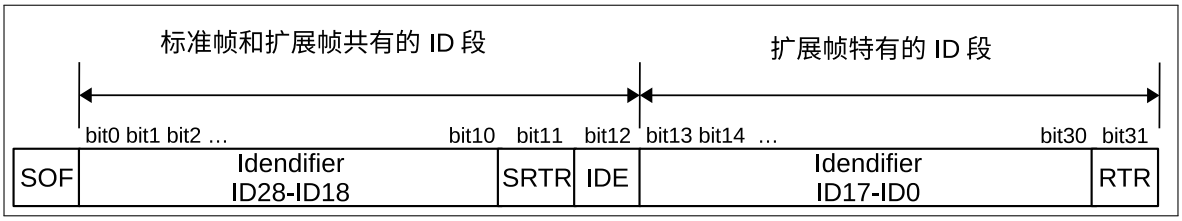


图 31-12. 丢失仲裁的 bit 位置

31.4.10 收发器自动待机

为降低功耗，TWAI 收发器通常支持自动待机模式。TWAI 控制器可以在适当的时候（例如，当总线将长时间空闲时）产生待机信号，将收发器置于待机状态。如果控制器停止发出待机信号，或在收发器检测到总线活动（也称为唤醒功能）时，收发器将退出待机模式。

ESP32-H2 的 TWAI 控制器支持通过硬件（自动）和软件（手动）两种方式输出待机信号，来控制芯片外部连接的 TWAI 收发器的开关。在硬件控制下，当总线保持空闲的时间超过可配置的时间时，TWAI 控制器将自动输出待机信号。在软件控制下，待机信号可由软件直接手动产生/关闭。

- 硬件方式：
  1. 通过将 TWAI_HW_CFG_REG 寄存器中的 TWAI_HW_STANDBY_EN 域置位，使能硬件待机功能。
  2. 配置 TWAI_HW_STANDB_CNT_REG 寄存器。该寄存器表明 TWAI 控制器进入空闲状态后多久会产生待机信号，数值为 TWAI 控制器工作时钟（默认为 32 MHz）的周期数。
- 软件方式：
  1. 直接置位 TWAI_SW_STANDBY_CFG_REG 寄存器中的 TWAI_SW_STANDBY_EN 域，便可使 TWAI 控制器产生待机信号。

上述两种方式产生的待机信号，在满足以下任一条件后，会被拉低（清零）。

1. TWAI 控制器退出空闲状态时，待机信号将被自动清零。
2. 用户通过置位 TWAI_SW_STANDBY_CFG_REG 寄存器中的 TWAI_SW_STANDBY_CLR 域，也可以将待机信号拉低。

31.5 寄存器列表

请注意，“访问权限”一栏中，“|”用于区分第 31.4.1 中描述的工作模式，其中左侧为操作模式下的访问权限，右侧标红字体为复位模式下的访问权限。本小节的所有地址均为相对于双线汽车接口基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

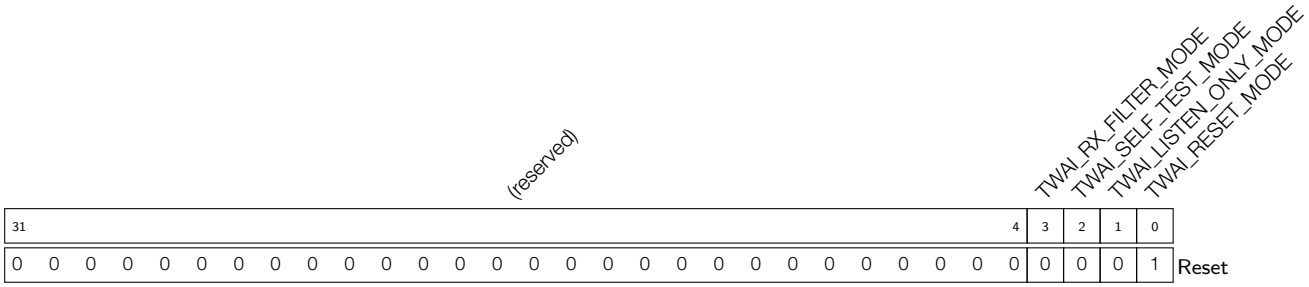
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
TWAI_MODE_REG	模式寄存器	0x0000	R/W
TWAI_BUS_TIMING_0_REG	时序配置寄存器 0	0x0018	RO | R/W
TWAI_BUS_TIMING_1_REG	时序配置寄存器 1	0x001C	RO | R/W
TWAI_ERR_WARNING_LIMIT_REG	错误寄存器	0x0034	RO | R/W
TWAI_DATA_0_REG	数据寄存器 0	0x0040	WO | R/W
TWAI_DATA_1_REG	数据寄存器 1	0x0044	WO | R/W
TWAI_DATA_2_REG	数据寄存器 2	0x0048	WO | R/W
TWAI_DATA_3_REG	数据寄存器 3	0x004C	WO | R/W
TWAI_DATA_4_REG	数据寄存器 4	0x0050	WO | R/W
TWAI_DATA_5_REG	数据寄存器 5	0x0054	WO | R/W
TWAI_DATA_6_REG	数据寄存器 6	0x0058	WO | R/W
TWAI_DATA_7_REG	数据寄存器 7	0x005C	WO | R/W
TWAI_DATA_8_REG	数据寄存器 8	0x0060	WO | RO
TWAI_DATA_9_REG	数据寄存器 9	0x0064	WO | RO
TWAI_DATA_10_REG	数据寄存器 10	0x0068	WO | RO
TWAI_DATA_11_REG	数据寄存器 11	0x006C	WO | RO
TWAI_DATA_12_REG	数据寄存器 12	0x0070	WO | RO
TWAI_CLOCK_DIVIDER_REG	时钟分频寄存器	0x007C	不定
TWAI_SW_STANDBY_CFG_REG	软件 standby 寄存器	0x0080	R/W | R/W
TWAI_HW_CFG_REG	硬件 standby 寄存器	0x0084	R/W | R/W
TWAI_HW_STANDBY_CNT_REG	standby 计数长度寄存器	0x0088	R/W | R/W
TWAI_IDLE_INTR_CNT_REG	空闲状态时长寄存器	0x008C	R/W | R/W
控制寄存器			
TWAI_CMD_REG	指令寄存器	0x0004	WO
状态寄存器			
TWAI_STATUS_REG	状态寄存器	0x0008	RO
TWAI_ARB_LOST_CAP_REG	仲裁丢失寄存器	0x002C	RO
TWAI_ERR_CODE_CAP_REG	错误捕获寄存器	0x0030	RO
TWAI_RX_ERR_CNT_REG	接收错误寄存器	0x0038	RO | R/W
TWAI_TX_ERR_CNT_REG	发送错误寄存器	0x003C	RO | R/W
TWAI_RX_MESSAGE_CNT_REG	接收数据寄存器	0x0074	RO
中断寄存器			
TWAI_INT_ST_REG	中断屏蔽寄存器	0x000C	RO
TWAI_INT_ENA_REG	中断使能寄存器	0x0010	R/W

31.6 寄存器

请注意“访问权限”一栏中，“I”左侧为操作模式下的访问权限，右侧标红字体为复位模式下的访问权限。本小节的所有地址均为相对于双线汽车接口基地址的地址偏移量（相对地址）。其中，TWAI 0 和 TWAI 1 都具有各自的基地址。具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 31.1. TWAI_MODE_REG (0x0000)



TWAI_RESET_MODE 配置 TWAI 控制器操作模式。

0：操作模式

1：复位模式

(R/W)

TWAI_LISTEN_ONLY_MODE 配置只听模式。

0：无效

1：只听模式。在该模式下，节点只接收总线上数据，不产生应答信号，也不更新接收错误计数。

(R/W)

TWAI_SELF_TEST_MODE 配置自测模式。

0：无效

1：自测模式。在该模式下，发送节点发送完数据后无需应答信号反馈。该模式常配合自接自收指令测试某个节点。

(R/W)

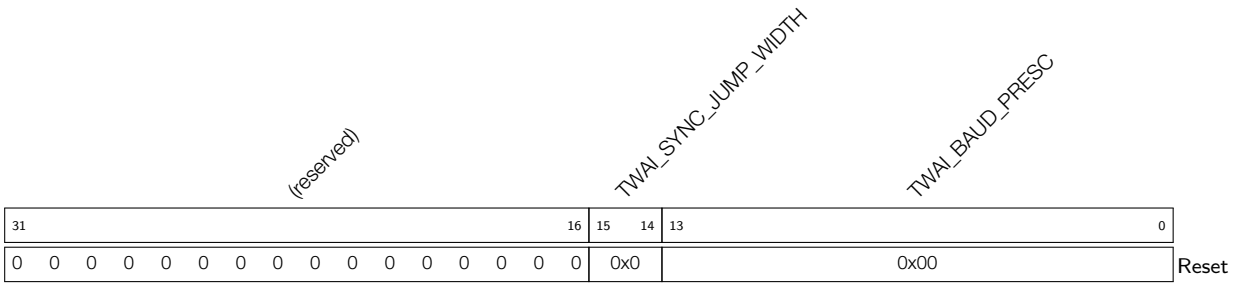
TWAI_RX_FILTER_MODE 配置滤波模式。

0：双滤波模式

1：单滤波模式

(R/W)

Register 31.2. TWAI_BUS_TIMING_0_REG (0x0018)



TWAI_BAUD_PRESC 配置预分频值，决定分频比例。

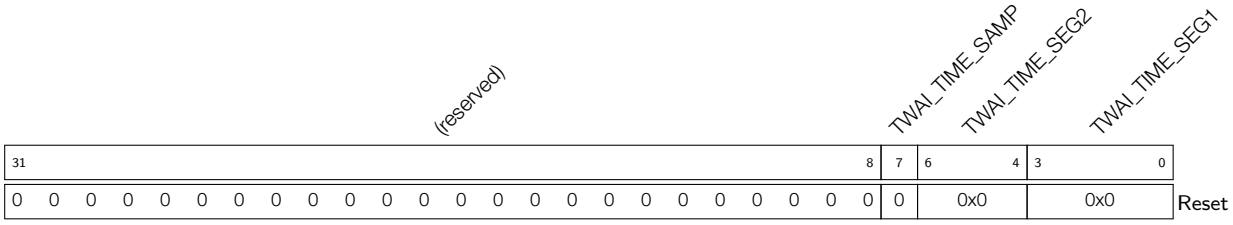
0：低比例

1：高比例

(RO | R/W)

TWAI_SYNC_JUMP_WIDTH 配置同步跳宽 (SJW)，范围为 1 ~ 4 个时间定额。(RO | R/W)

Register 31.3. TWAI_BUS_TIMING_1_REG (0x001C)



TWAI_TIME_SEG1 配置缓冲时期段 1 的宽度。(RO | R/W)

TWAI_TIME_SEG2 配置缓冲时期段 2 的宽度。(RO | R/W)

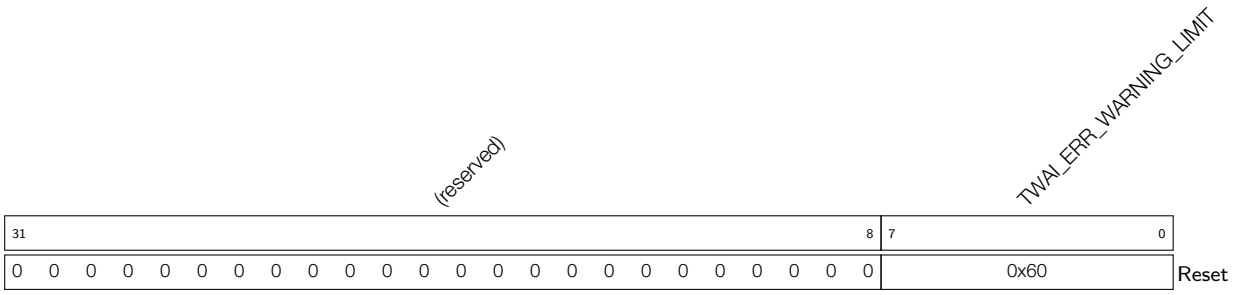
TWAI_TIME_SAMP 配置采样点数目。

0：采样一次

1：采样三次

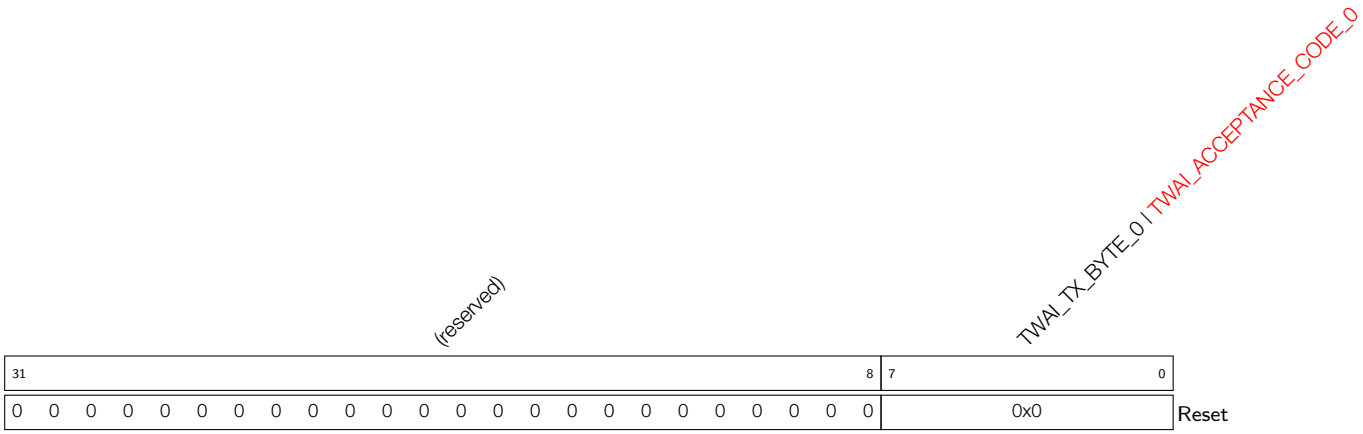
(RO | R/W)

Register 31.4. TWAI_ERR_WARNING_LIMIT_REG (0x0034)



TWAI_ERR_WARNING_LIMIT 配置错误报警阈值。当任一错误计数数值超过该阈值或者所有错误计数数值都小于该阈值时，将触发错误报警中断。仅在使能信号为 1 时有效。(RO | R/W)

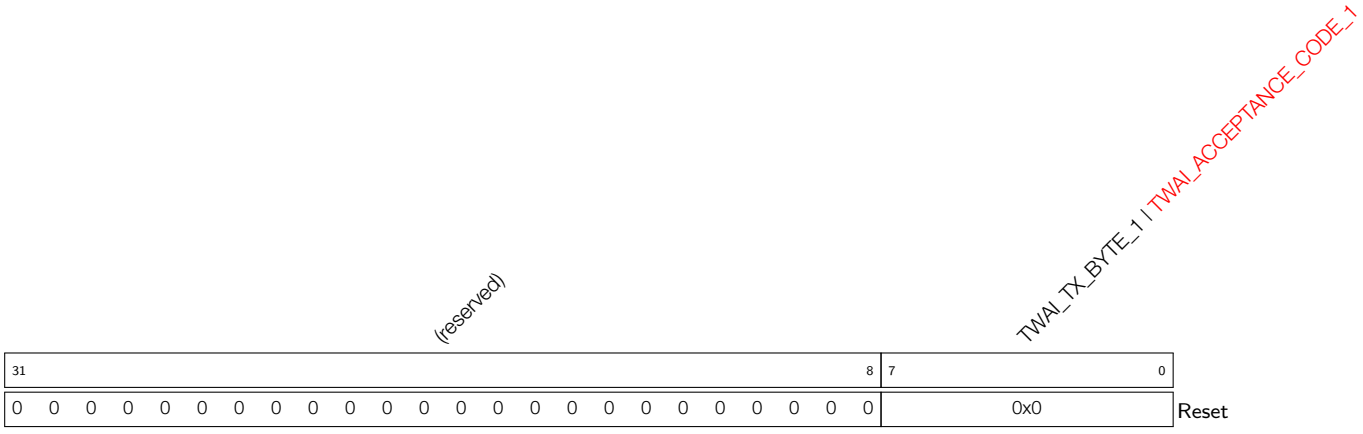
Register 31.5. TWAI_DATA_0_REG (0x0040)



TWAI_TX_BYTE_0 操作模式下，配置待发送数据的第 0 个字节内容。(WO)

TWAI_ACCEPTANCE_CODE_0 复位模式下，配置滤波编码的第 0 个字节。(R/W)

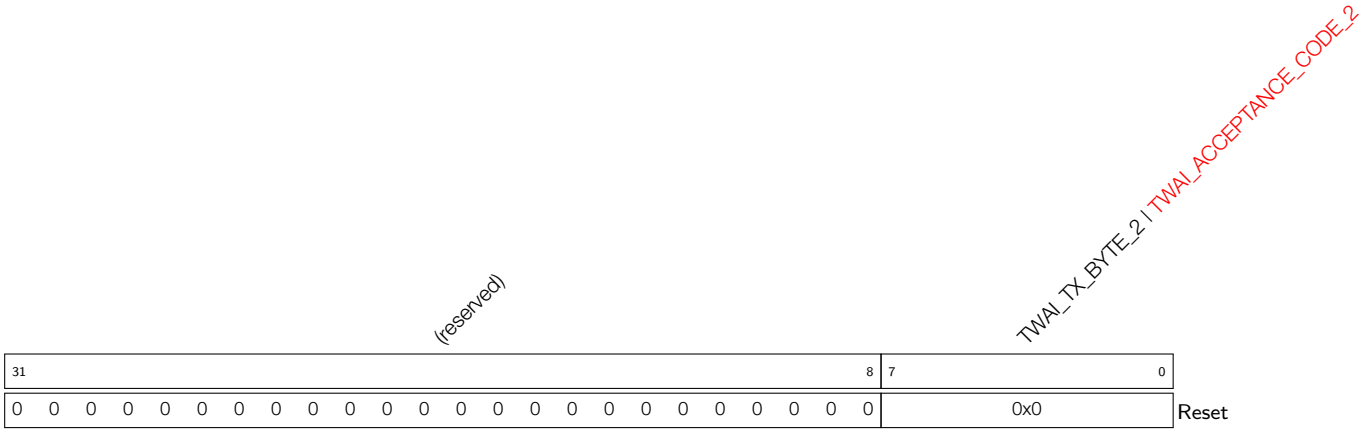
Register 31.6. TWAI_DATA_1_REG (0x0044)



TWAI_TX_BYTE_1 操作模式下，配置待发送数据的第 1 个字节内容。(WO)

TWAI_ACCEPTANCE_CODE_1 复位模式下，配置滤波编码的第 1 个字节。(R/W)

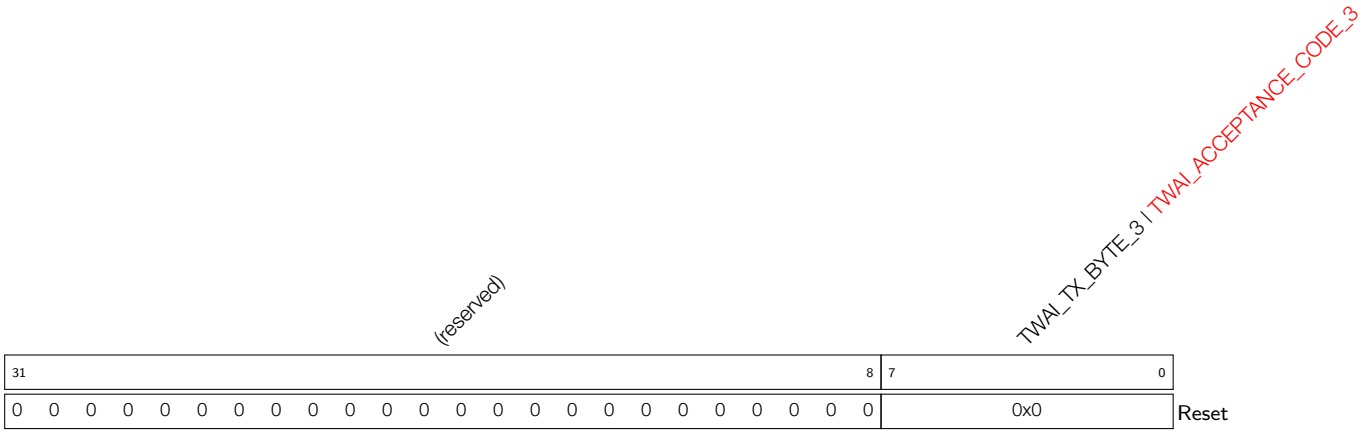
Register 31.7. TWAI_DATA_2_REG (0x0048)



TWAI_TX_BYTE_2 操作模式下，配置待发送数据的第 2 个字节内容。(WO)

TWAI_ACCEPTANCE_CODE_2 复位模式下，配置滤波编码的第 2 个字节。(R/W)

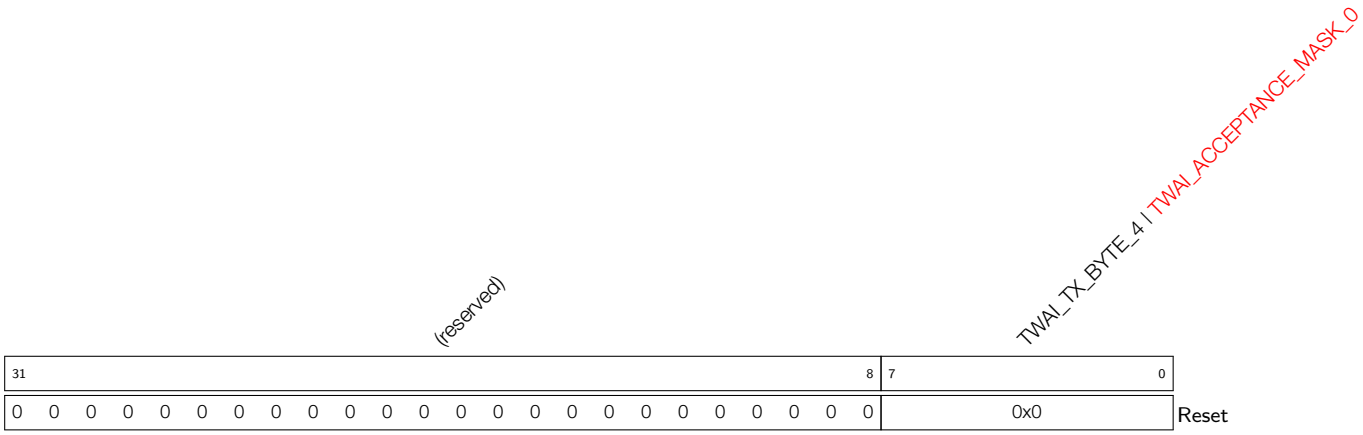
Register 31.8. TWAI_DATA_3_REG (0x004C)



TWAI_TX_BYTE_3 操作模式下，配置待发送数据的第 3 个字节内容。(WO)

TWAI_ACCEPTANCE_CODE_3 复位模式下，配置滤波编码的第 3 个字节。(R/W)

Register 31.9. TWAI_DATA_4_REG (0x0050)



TWAI_TX_BYTE_4 操作模式下，配置待发送数据的第 4 个字节内容。(WO)

TWAI_ACCEPTANCE_MASK_0 复位模式下，配置滤波编码的第 0 个字节。

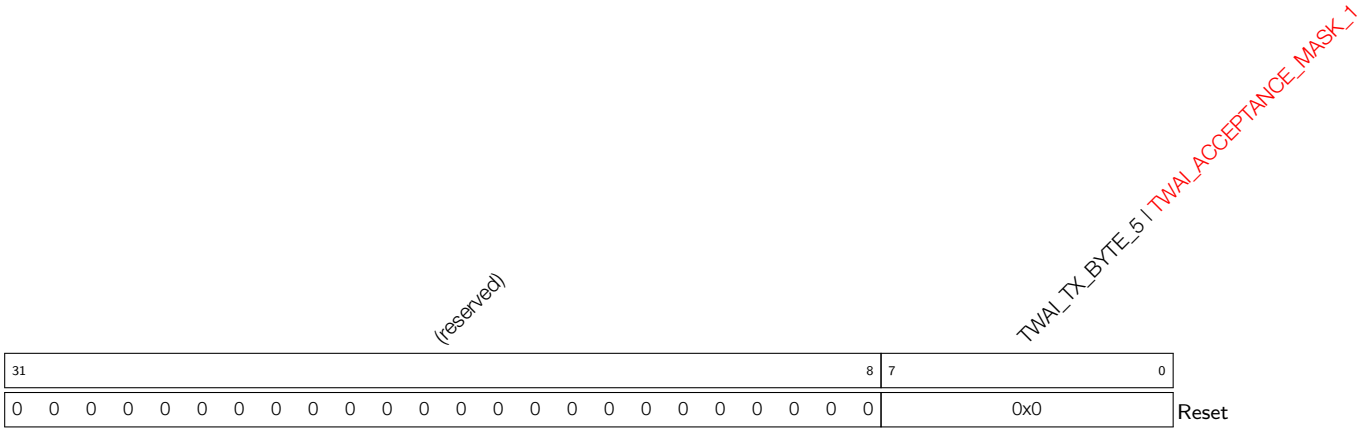
1：nihao

2：world

3：success

(R/W)

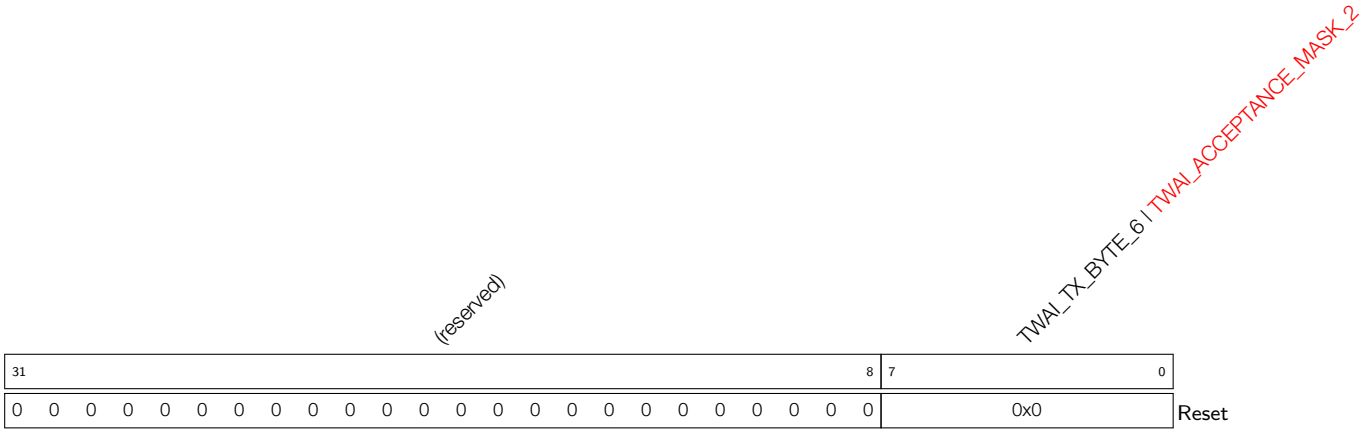
Register 31.10. TWAI_DATA_5_REG (0x0054)



TWAI_TX_BYTE_5 操作模式下，配置待发送数据的第 5 个字节内容。(WO)

TWAI_ACCEPTANCE_MASK_1 复位模式下，配置滤波编码的第 1 个字节。(R/W)

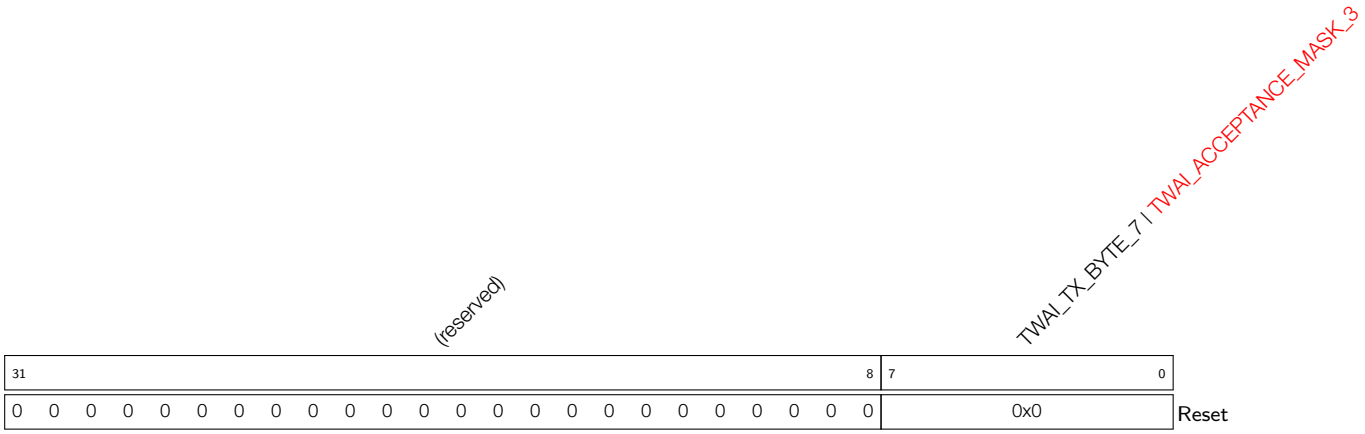
Register 31.11. TWAI_DATA_6_REG (0x0058)



TWAI_TX_BYTE_6 操作模式下，配置待发送数据的第 6 个字节内容。(WO)

TWAI_ACCEPTANCE_MASK_2 复位模式下，配置滤波编码的第 2 个字节。(R/W)

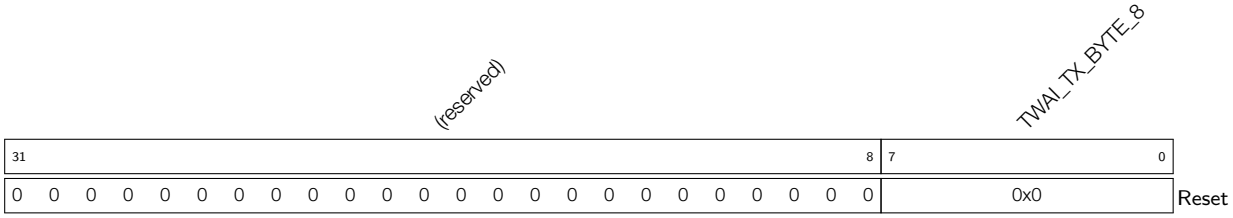
Register 31.12. TWAI_DATA_7_REG (0x005C)



TWAI_TX_BYTE_7 操作模式下，配置待发送数据的第 7 个字节内容。(WO)

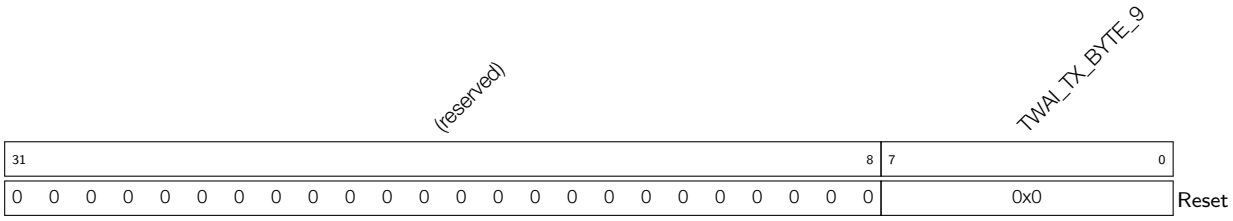
TWAI_ACCEPTANCE_MASK_3 复位模式下，配置滤波编码的第 3 个字节。(R/W)

Register 31.13. TWAI_DATA_8_REG (0x0060)



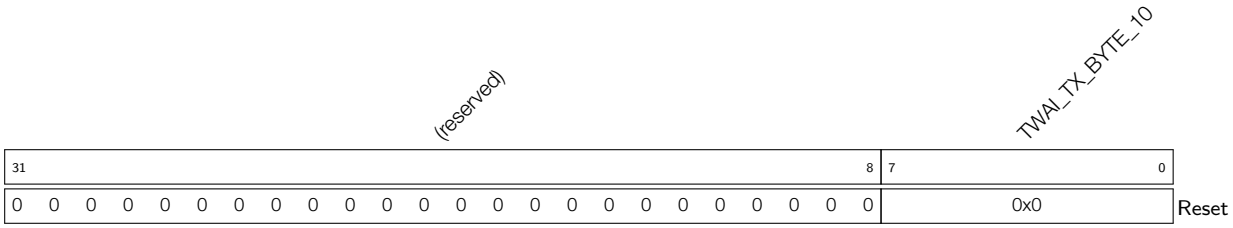
TWAI_TX_BYTE_8 操作模式下，配置待发送数据的第 8 个字节内容。(WO)

Register 31.14. TWAI_DATA_9_REG (0x0064)



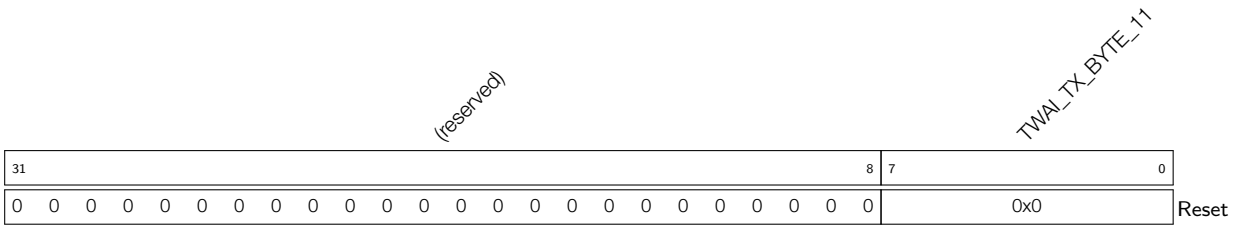
TWAI_TX_BYTE_9 操作模式下，配置待发送数据的第 9 个字节内容。(WO)

Register 31.15. TWAI_DATA_10_REG (0x0068)



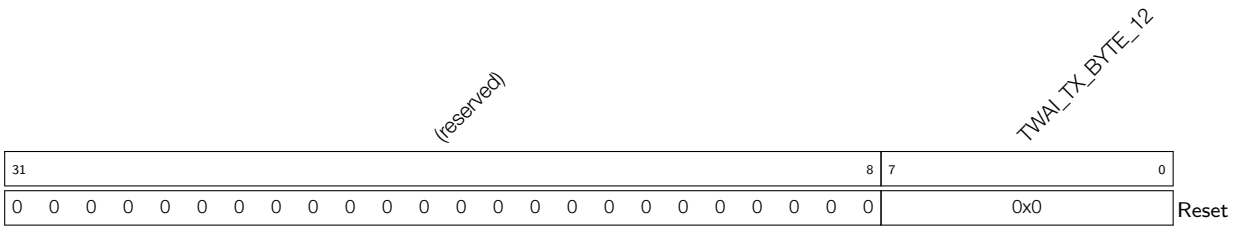
TWAI_TX_BYTE_10 操作模式下，配置待发送数据的第 10 个字节内容。(WO)

Register 31.16. TWAI_DATA_11_REG (0x006C)



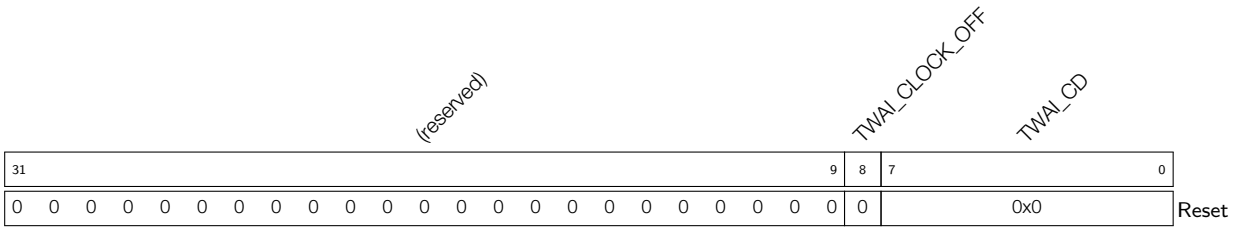
TWAI_TX_BYTE_11 操作模式下，配置待发送数据的第 11 个字节内容。(WO)

Register 31.17. TWAI_DATA_12_REG (0x0070)



TWAI_TX_BYTE_12 操作模式下，配置待发送数据的第 12 个字节内容。(WO)

Register 31.18. TWAI_CLOCK_DIVIDER_REG (0x007C)



TWAI_CD 配置输出时钟 CLKOUT 的分频系数。(R/W)

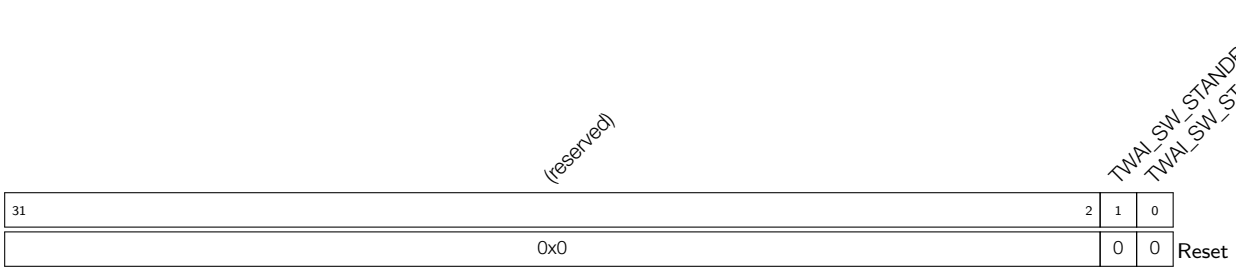
TWAI_CLOCK_OFF 复位模式下，配置是否打开 CLKOUT 时钟。

0：打开 CLKOUT 时钟

1：关闭 CLKOUT 时钟

(RO | R/W)

Register 31.19. TWAI_SW_STANDBY_CFG_REG (0x0080)



TWAI_SW_STANDBY_CLR 用于软件清除待机信号。

0：无效

1：清除待机信号

(R/W | R/W)

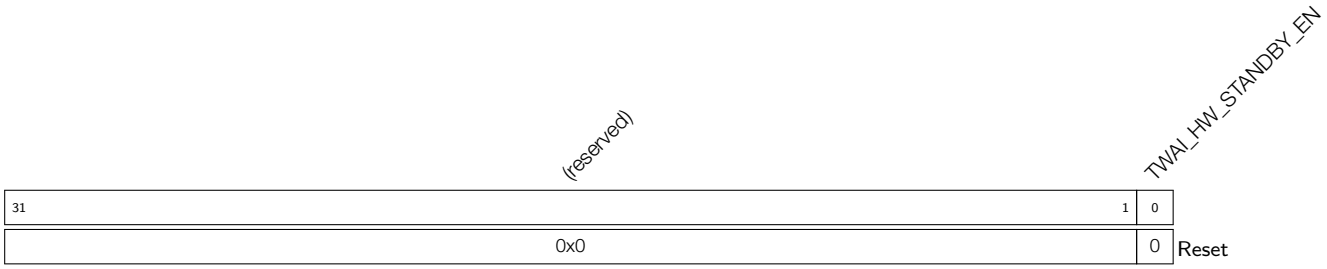
TWAI_SW_STANDBY_EN 用于软件置位待机信号。

0：无效

1：置位待机信号

(R/W | R/W)

Register 31.20. TWAI_HW_CFG_REG (0x0084)



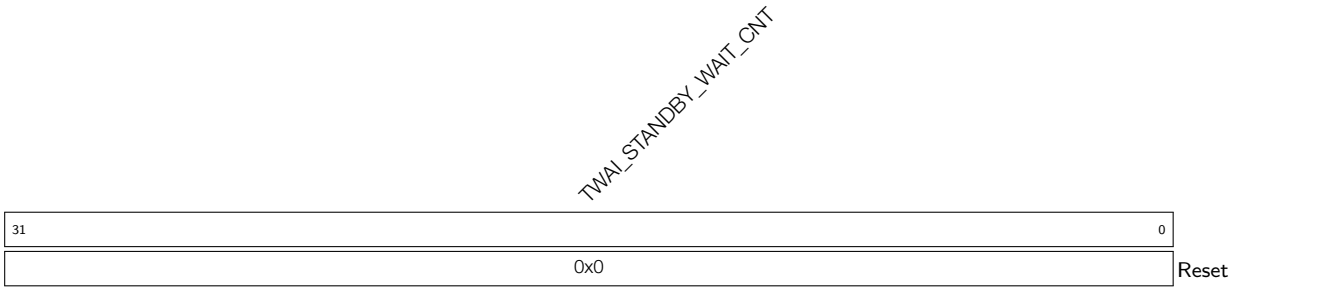
TWAI_HW_STANDBY_EN 用于使能硬件待机功能。

0：无效

1：使能硬件待机功能

(R/W | R/W)

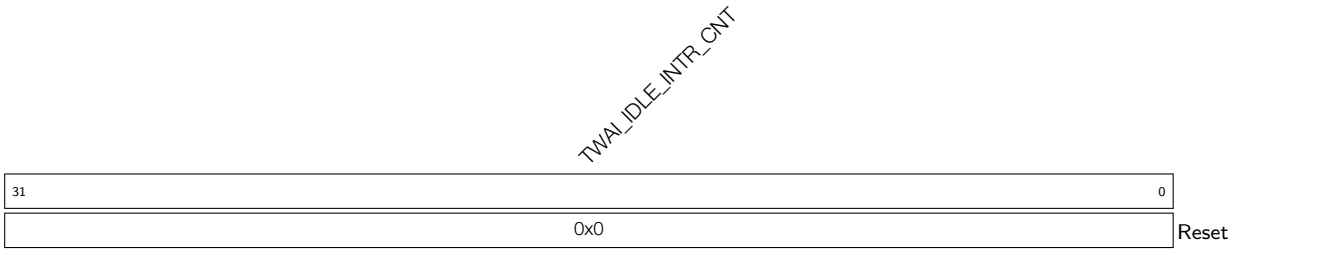
Register 31.21. TWAI_HW_STANDBY_CNT_REG (0x0070)



TWAI_STANDBY_WAIT_CNT 用于配置空闲状态下硬件触发待机信号的时间。(R/W | R/W)

单位：TWAI 控制器时钟周期数。

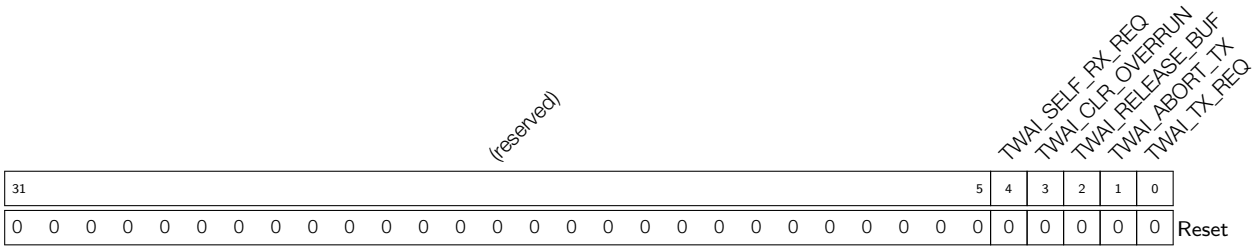
Register 31.22. TWAI_IDLE_INTR_CNT_REG (0x0070)



TWAI_IDLE_INTR_CNT 用于配置空闲状态下硬件产生总线空闲中断信号的时间。(R/W | R/W)

单位：TWAI 控制器时钟周期数。

Register 31.23. TWAI_CMD_REG (0x0004)



TWAI_TX_REQ 配置是否驱动节点开始发送数据任务。

0：无效

1：驱动节点开始发送数据任务

(WO)

TWAI_ABORT_TX 配置是否取消当前还未开始的发送任务。

0：无效

1：取消当前还未开始的发送任务

(WO)

TWAI_RELEASE_BUF 配置是否释放接收缓冲器。

0：无效

1：释放接收缓冲器

(WO)

TWAI_CLR_OVERRUN 配置是否清除数据溢出状态。

0：无效

1：清除数据溢出状态

(WO)

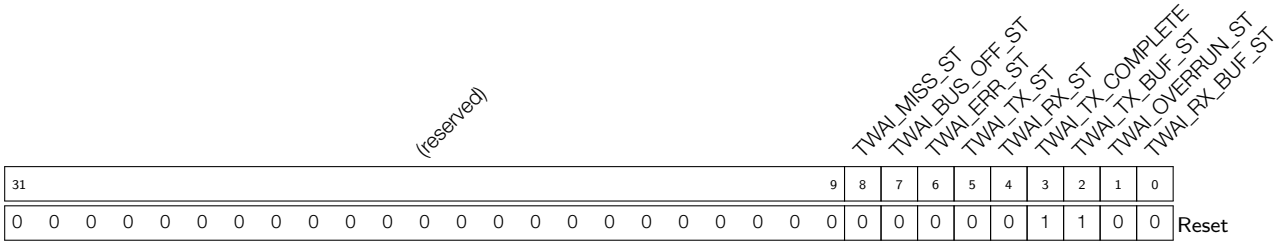
TWAI_SELF_RX_REQ 配置是否允许发送节点发送数据的同时接收总线上的数据。

0：无效

1：允许发送节点发送数据的同时接收总线上的数据

(WO)

Register 31.24. TWAI_STATUS_REG (0x0008)



TWAI_RX_BUF_ST 表示接收缓冲器中数据是否为空。

0：为空

1：不为空，至少有一个已经接收到的数据包。

(RO)

TWAI_OVERRUN_ST 表示接收 FIFO 中存储的数据是否已满。

0：未满

1：已满，产生了溢出。

(RO)

TWAI_TX_BUF_ST 表示发送缓冲器是否为空。

0：不为空

1：为空，允许写入待发送数据。

(RO)

TWAI_TX_COMPLETE 表示是否从总线上接收到一个数据包。

0：未接收

1：成功接收

(RO)

TWAI_RX_ST 表示节点是否正从总线上接收数据。

0：未接收

1：正在接收

(RO)

TWAI_TX_ST 表示节点是否正在往总线上发送数据。

0：未接收

1：正在接收

(RO)

TWAI_ERR_ST 表示接收错误计数和发送错误计数中至少有一个数值大于等于寄存器 TWAI_ERR_WARNING_LIMIT_REG 中配置的数值。(RO)

TWAI_BUS_OFF_ST 表示节点是否处于离线状态。

0：非离线状态

1：离线状态，不再响应总线上的数据传输。

(RO)

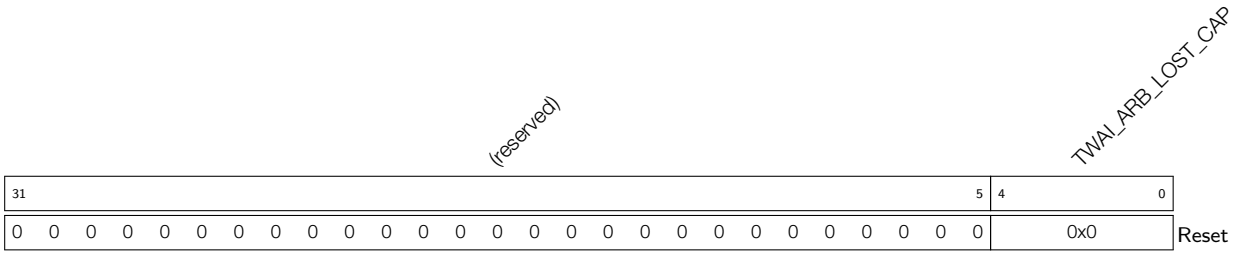
TWAI_MISS_ST 表示从接收 FIFO 中取出数据包的完整状态。

0：当前数据包是完整的。

1：当前数据包是缺失的。

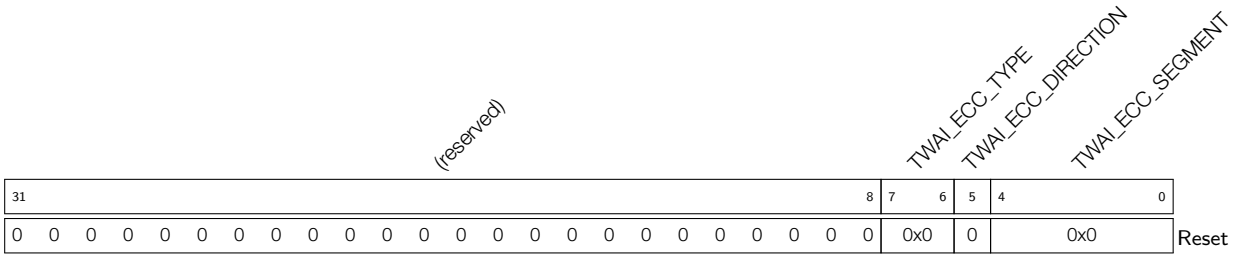
(RO)

Register 31.25. TWAI_ARB_LOST_CAP_REG (0x002C)



TWAI_ARB_LOST_CAP 表示发送节点仲裁丢失的 bit 位置。(RO)

Register 31.26. TWAI_ERR_CODE_CAP_REG (0x0030)



TWAI_ECC_SEGMENT 表示错误发生的位置，详见表 31-16。(RO)

TWAI_ECC_DIRECTION 表示错误时节点的数据传输方向。

0：发送数据时发生错误。

1：接收数据时发生错误。

(RO)

TWAI_ECC_TYPE 表示错误类别。

0：位错误

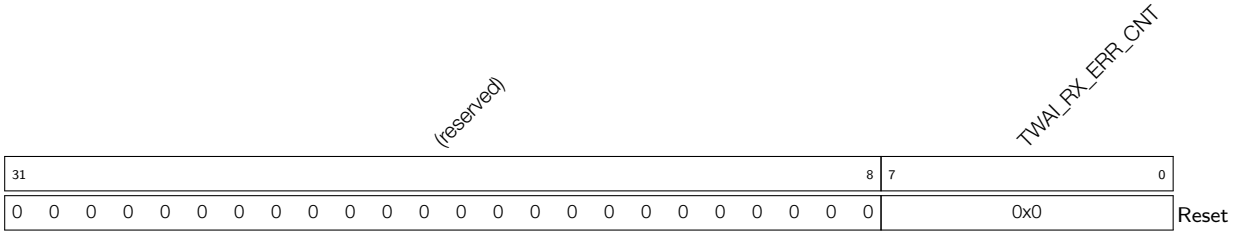
1：格式错误

2：填充错误

3：其他错误

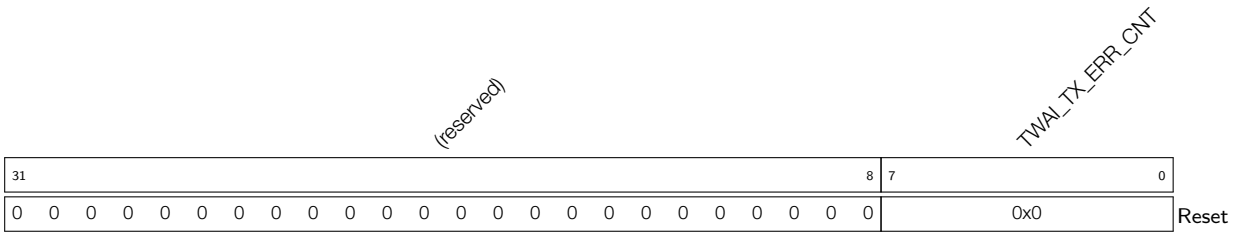
(RO)

Register 31.27. TWAI_RX_ERR_CNT_REG (0x0038)



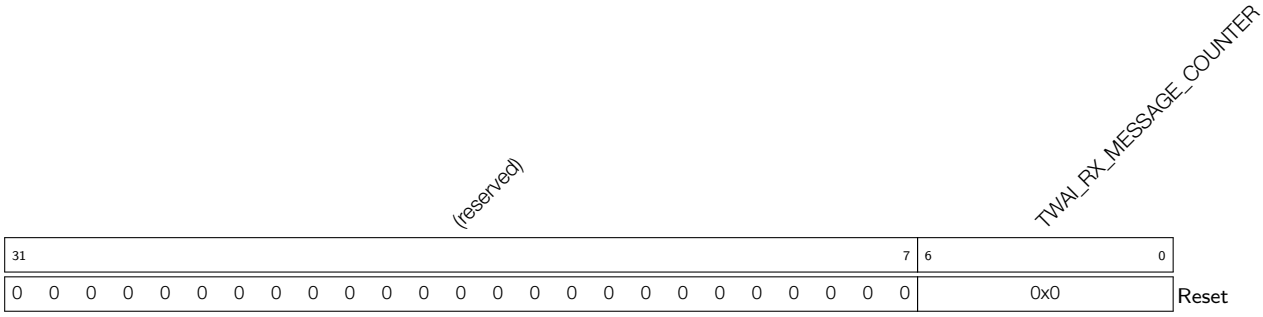
TWAI_RX_ERR_CNT 接收错误计数，数值变化发生在接收状态下。(RO | R/W)

Register 31.28. TWAI_TX_ERR_CNT_REG (0x003C)



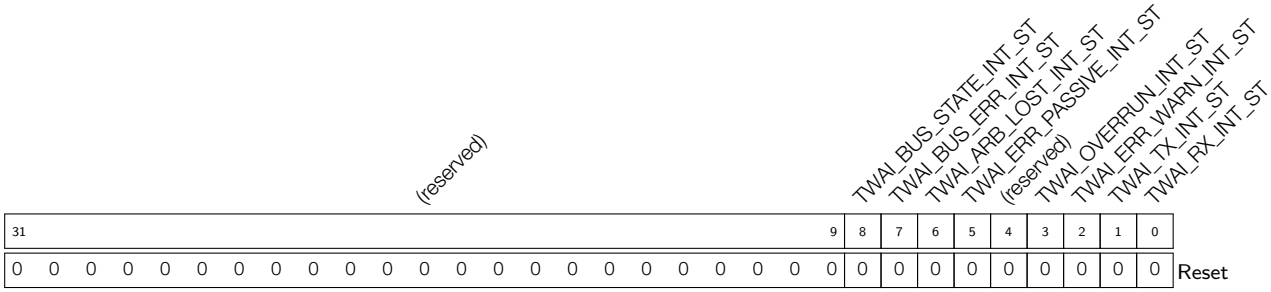
TWAI_TX_ERR_CNT 发送错误计数，数值变化发生在发送状态下。(RO | R/W)

Register 31.29. TWAI_RX_MESSAGE_CNT_REG (0x0074)



TWAI_RX_MESSAGE_COUNTER 表示接收 FIFO 中数据包的个数。(RO)

Register 31.30. TWAI_INT_ST_REG (0x000C)



TWAI_RX_INT_ST TWAI_RX_INT 的屏蔽中断状态。(RO)

TWAI_TX_INT_ST TWAI_TX_INT 的屏蔽中断状态。(RO)

TWAI_ERR_WARN_INT_ST TWAI_ERR_WARN_INT 的屏蔽中断状态。(RO)

TWAI_OVERRUN_INT_ST TWAI_OVERRUN_INT 的屏蔽中断状态。(RO)

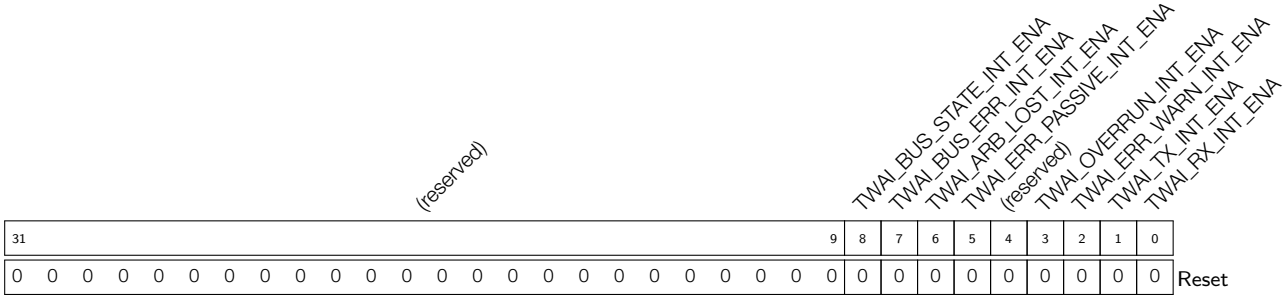
TWAI_ERR_PASSIVE_INT_ST TWAI_ERR_PASSIVE_INT 的屏蔽中断状态。(RO)

TWAI_ARB_LOST_INT_ST TWAI_ARB_LOST_INT 的屏蔽中断状态。(RO)

TWAI_BUS_ERR_INT_ST TWAI_BUS_ERR_INT 的屏蔽中断状态。(RO)

TWAI_BUS_STATE_INT_ST TWAI_BUS_STATE_INT 的屏蔽中断状态。(RO)

Register 31.31. TWAI_INT_ENA_REG (0x0010)



TWAI_RX_INT_ENA 置 1 使能接收中断。(R/W)

TWAI_TX_INT_ENA 置 1 使能发送中断。(R/W)

TWAI_ERR_WARN_INT_ENA 置 1 使能错误报警中断。(R/W)

TWAI_OVERRUN_INT_ENA 置 1 使能数据溢出中断。(R/W)

TWAI_ERR_PASSIVE_INT_ENA 置 1 使能被动错误中断。(R/W)

TWAI_ARB_LOST_INT_ENA 置 1 使能仲裁丢失中断。(R/W)

TWAI_BUS_ERR_INT_ENA 置 1 使能总线错误中断。(R/W)

TWAI_BUS_STATE_INT_ENA 置 1 使能总线状态中断。(R/W)

32 LED PWM 控制器 (LEDC)

32.1 概述

LED PWM 控制器用于生成控制 LED 的脉冲宽度调制信号 (PWM)，具有占空比自动渐变等专门功能。该外设也可生成 PWM 信号用作其他用途。

32.2 特性

LED PWM 控制器具有如下特性:

- 六个独立的 PWM 生成器（即六个通道）
- PWM 占空比最大精度为 20 位
- 四个独立定时器，可实现小数分频
- PWM 输出信号相位可调节
- PWM 占空比微调
- 占空比自动渐变—即 PWM 信号占空比可逐渐增加或减小，无须处理器干预，渐变完成时产生中断
- 每个 PWM 生成器包含 16 个占空比渐变区间，用于生成占空比伽玛曲线渐变的信号。每个区间都可以独立配置占空比变化方向（增加或减少）、变化步长、变化次数以及变化频率
- 低功耗模式 (Light-sleep mode) 下可输出 PWM 信号
- 可以生成 ETM（事件任务矩阵）外设相关的事件，可以接收 ETM 外设相关的任务

由于四个定时器具有相同的功能和运行方式，下文将四个定时器统称为定时器*x*（*x* 的范围是 0 到 3）。同样的，六个 PWM 生成器的功能和运行方式也相同，因此下文将统称为 PWM *n*（*n* 的范围是 0 到 5）。

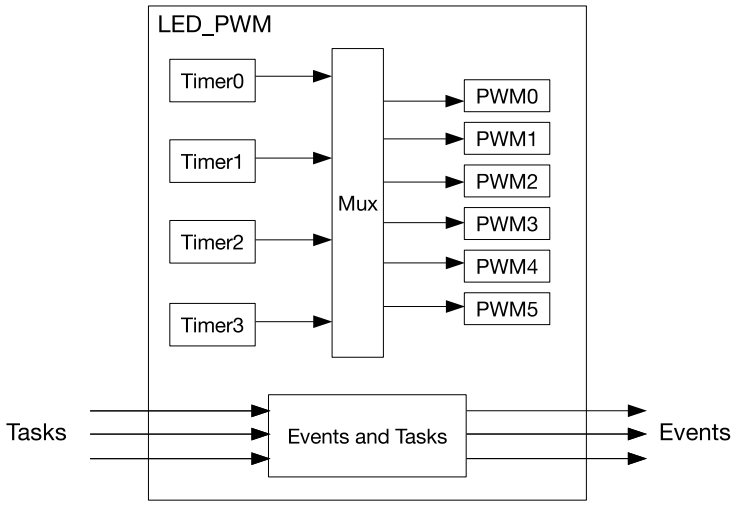


图 32-1. LED PWM 控制器架构

32.3 功能描述

32.3.1 架构

图 32-1为 LED PWM 控制器的架构。

四个定时器每个内部都有一个时基计数器（即基于基准时钟周期计数的计数器），因此每个定时器都可以独立配置（可配置时钟分频器和计数器最大值）。每个 PWM 生成器通过配置 LEDC_TIMER_SEL_CH*n* 可以在四个定时器中择一，并以该定时器的计数值 timer*x*_cnt 为基准生成 PWM 信号。

图 32-2 为定时器和 PWM 生成器的主要功能块。

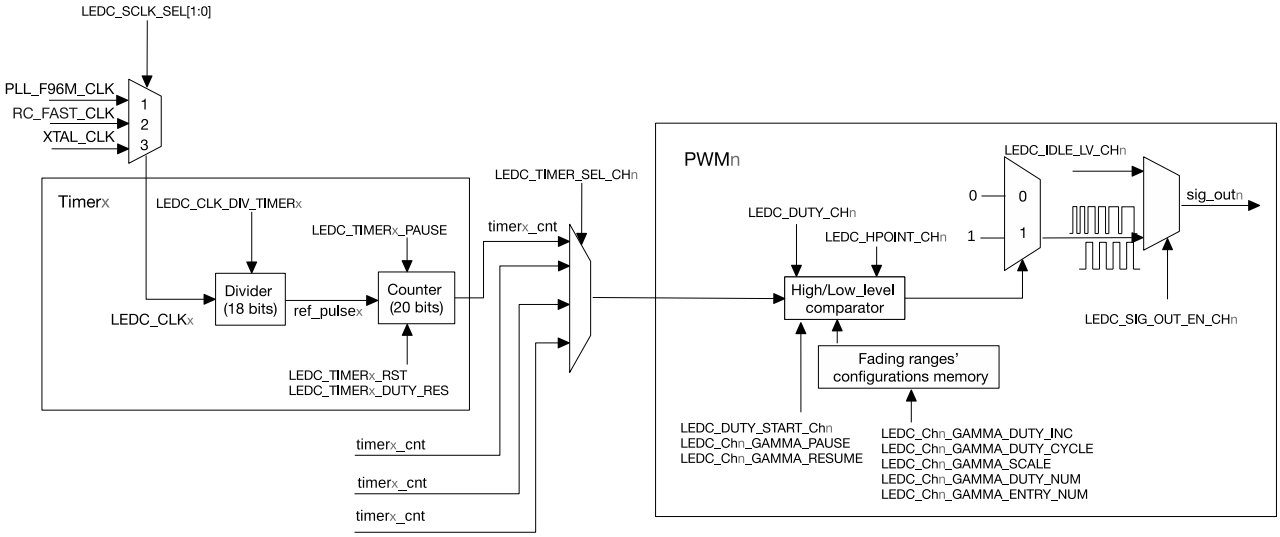


图 32-2. 定时器和 PWM 生成器功能框图

32.3.2 定时器

LED PWM 控制器的每个定时器内部都有一个时基计数器。图 32-2 中时基计数器使用的时钟信号称为 ref_pulse*x*。所有定时器使用同一个时钟源信号 LEDC_CLK*x*，该时钟源信号经分频器分频后产生 ref_pulse*x* 供计数器使用。

32.3.2.1 时钟源

软件可配置的 LED PWM 寄存器由 APB_CLK 时钟驱动。要使用 LED PWM 控制器，需使能 LED PWM 的 APB_CLK 时钟信号，该时钟信号可通过置位 PCR_LEDC_CONF_REG 寄存器的 PCR_LEDC_CLK_EN 字段使能，通过软件置位 PCR_LEDC_CONF_REG 寄存器的 PCR_LEDC_RST_EN 字段复位。

LED PWM 控制器的定时器从下述时钟信号中选择一个作为其时钟源信号：PLL_F96M_CLK、RC_FAST_CLK 和 XTAL_CLK。为 LEDC_CLK*x* 选择时钟源信号的配置如下：

- PLL_F96M_CLK：将 LEDC_SCLK_SEL[1:0] 置 1
- RC_FAST_CLK：将 LEDC_SCLK_SEL[1:0] 置 2
- XTAL_CLK：将 LEDC_SCLK_SEL[1:0] 置 3

之后，LEDC_CLK*x* 信号会进入时钟分频器。

如果 LEDC_SCLK_SEL[1:0] 配置为无效值，即 0，则 LED PWM 控制器无时钟源，所有功能无法正常工作。

更多信息，请参阅章节 7 复位和时钟。

32.3.2.2 时钟分频器配置

LEDC_CLK*x* 信号传输到时钟分频器，产生 ref_pulse*x* 信号供计数器使用。ref_pulse*x* 的频率等于 LEDC_CLK*x* 的频率经分频系数 LEDC_CLK_DIV 分频后的结果（见图 32-2）。

分频系数 LEDC_CLK_DIV 可为非整数，其值可根据下列等式配置：

$$\text{LEDC\_CLK\_DIV} = A + \frac{B}{256}$$

- 整数部分 *A* 为 LEDC_CLK_DIV_TIMER*x* 字段的高 10 位（即 LEDC_TIMER*x*_CONF_REG[22:13]）
- 小数部分 *B* 为 LEDC_CLK_DIV_TIMER*x* 字段的低 8 位（即 LEDC_TIMER*x*_CONF_REG[12:5]）

小数部分 *B* 为 0 时，LEDC_CLK_DIV 的值为整数（整数分频）。也就是说，每 *A* 个 LEDC_CLK*x* 时钟周期产生一个 ref_pulse*x* 时钟脉冲。

小数部分 *B* 不为 0 时，LEDC_CLK_DIV 的值非整数。时钟分频器按照 *A* 个 LEDC_CLK*x* 时钟周期和 (*A*+1) 个 LEDC_CLK*x* 时钟周期轮流产生 ref_pulse*x* 时钟脉冲，实现非整数分频。这样一来，ref_pulse*x* 时钟脉冲的平均频率便会是理想值（非整数分频的频率）。每 256 个 ref_pulse*x* 时钟脉冲中：

- 有 *B* 个 ref_pulse*x* 时钟脉冲以 (*A*+1) 个 LEDC_CLK*x* 时钟周期分频
- 有 (256-*B*) 个 ref_pulse*x* 时钟脉冲以 *A* 个 LEDC_CLK*x* 时钟周期分频
- 以 (*A*+1) 个 LEDC_CLK*x* 时钟周期分频的 ref_pulse*x* 时钟脉冲均匀分布在以 *A* 个 LEDC_CLK*x* 时钟周期分频的 ref_pulse*x* 时钟脉冲中

图 32-3 展示了分频系数 LEDC_CLK_DIV 非整数时，LEDC_CLK*x* 时钟脉冲和 ref_pulse*x* 时钟脉冲的关系。

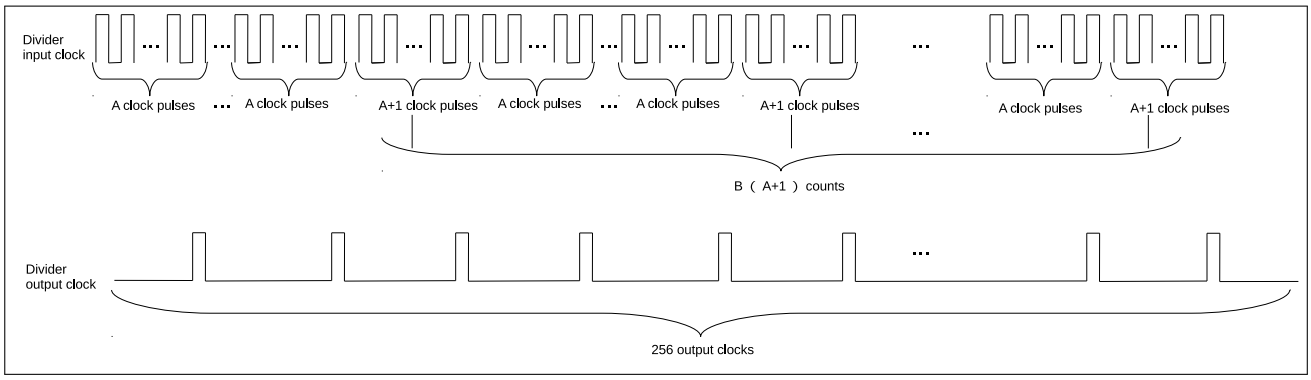


图 32-3. LEDC_CLK_DIV 非整数时的分频

在运行时改变定时器时钟的分频系数，需先配置 LEDC_CLK_DIV_TIMER*x* 字段，然后置位 LEDC_TIMER*x*_PARA_UP 字段应用新配置。新配置会在计数器下次溢出时生效。LEDC_TIMER*x*_PARA_UP 字段由硬件自动清除。

32.3.2.3 20 位计数器

每个定时器有一个以 ref_pulse*x* 为基准时钟的 20 位时基计数器（见图 32-2）。LEDC_TIMER*x*_DUTY_RES 字段用于配置 20 位计数器的最大值。因此，PWM 信号的最大精度为 20 位。计数器最大可计数至 $2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}} - 1$，然后溢出并重新从 0 开始计数。软件可以读取、复位、暂停计数器。计数器和 PWM 信号精度的关系如图 32-4 所示。

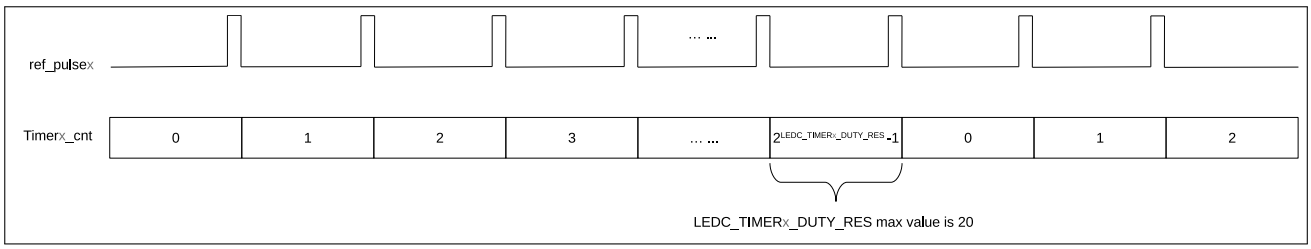


图 32-4. 计数器和 PWM 信号精度关系

计数器可在每次溢出时触发 LEDC_TIMER*x*_OVF_INT 中断（这个中断为硬件自动产生，不需要配置）。计数器也可配置为在溢出 LEDC_OVF_NUM_CH*n* + 1 次时触发 LEDC_OVF_CNT_CH*n*_INT 中断，该中断配置步骤如下：

1. 配置 LEDC_TIMER_SEL_CH*n* 为 PWM 生成器选择该定时器
2. 置位 LEDC_OVF_CNT_EN_CH*n* 使能溢出次数计数器
3. 把 LEDC_OVF_NUM_CH*n* 的值设为触发中断的计数器溢出次数减 1
4. 置位 LEDC_OVF_CNT_CH*n*_INT_ENA 使能溢出中断
5. 置位 LEDC_TIMER*x*_DUTY_RES 使能定时器，等待 LEDC_OVF_CNT_CH*n*_INT 中断产生

在运行时改变计数器的最大值，需先配置 LEDC_TIMER*x*_DUTY_RES 字段，然后置位 LEDC_TIMER*x*_PARA_UP 字段。新的配置在计数器下一次溢出时生效。如果重新配置 LEDC_OVF_CNT_EN_CH*n* 字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。总之，更改配置时需置位 LEDC_TIMER*x*_PARA_UP 或 LEDC_PARA_UP_CH*n* 应用新配置。
LEDC_TIMER*x*_PARA_UP 和 LEDC_PARA_UP_CH*n*字段由硬件自动清除。

如图 32-2 所示，PWM 生成器输出信号 sig_out*n* 的频率取决于定时器时钟源 LEDC_CLK*x* 的频率、时钟分频系数 LEDC_CLK_DIV 以及占空比精度（计数器位宽）LEDC_TIMER*x*_DUTY_RES：

$$f_{\text{PWM}} = \frac{f_{\text{LEDC\_CLK}x}}{\text{LEDC\_CLK\_DIV} \cdot 2^{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}}$$

上述公式变形后，可得到以下公式计算预期的占空比精度：

$$\text{LEDC\_TIMER}x\text{\_DUTY\_RES} = \log_2\left(\frac{f_{\text{LEDC\_CLK}x}}{f_{\text{PWM}} \cdot \text{LEDC\_CLK\_DIV}}\right)$$

表 32-1 列出了常用配置频率及其对应精度。

表 32-1. 常用配置频率及精度

LEDC_CLK*x*	PWM 频率	最高精度（位）[1]	最低精度（位）[2]
PLL_F96M_CLK (96 MHz)	1 kHz	16	6
PLL_F96M_CLK (96 MHz)	5 kHz	14	4
PLL_F96M_CLK (96 MHz)	10 kHz	13	3
XTAL_CLK (32 MHz)	1 kHz	14	4
XTAL_CLK (32 MHz)	4 kHz	12	2
RC_FAST_CLK (8 MHz)	1 kHz	12	2
RC_FAST_CLK (8 MHz)	2 kHz	11	1

[1] 最高精度指时钟分频系数 LEDC_CLK_DIV 为 1 时的精度。如果经公式计算出的最高精度超过了计数器位宽 20 位，则最高精度为 20。

[2] 最低精度指时钟分频系数 LEDC_CLK_DIV 为 $1023 + \frac{255}{256}$ 时的精度。如果经公式计算出的最低精度小于 0，则最低精度为 1。

32.3.3 PWM 生成器

要生成 PWM 信号，PWM 生成器 (PWM*n*) 需选择一个定时器 (Timer*x*)。每个 PWM 生成器均可通过置位 LEDC_TIMER_SEL_CH*n* 单独配置，在四个定时器中选择一个用于生成 PWM 信号。

如图 32-2 所示，每个 PWM 生成器主要包括一个高低电平比较器和两个选择器。PWM 生成器将定时器的 20 位计数值 (Timer*x*_cnt) 与高低电平比较器的值 Hpoint*n* 和 Lpoint*n* 比较。如果定时器的计数值等于 Hpoint*n* 或 Lpoint*n*，PWM 信号可以输出高低电平：

- 如果 Timer*x*_cnt == Hpoint*n*，则 sig_out*n* 为 1。
- 如果 Timer*x*_cnt == Lpoint*n*，则 sig_out*n* 为 0。

图 32-5 展示了如何使用 Hpoint*n* 和 Lpoint*n* 生成占空比固定的 PWM 信号。

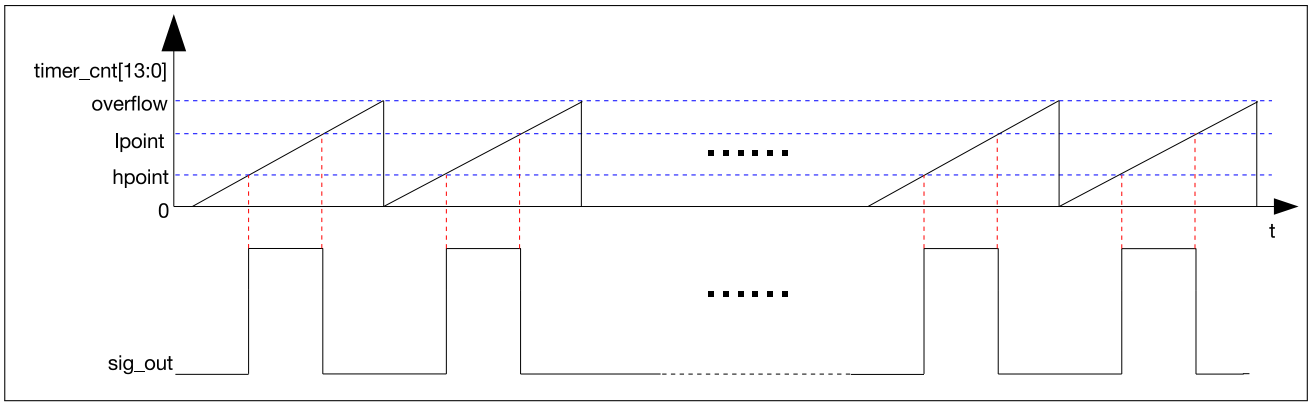


图 32-5. LED PWM 输出信号图

每当所选定时器的计数器溢出时，PWM 生成器 (PWM*n*) 的 Hpoint*n* 值更新为 LEDC_HPOINT_CH*n*。Lpoint*n* 的值同样在计数器每次溢出时更新，为 LEDC_DUTY_CH*n*[24:4] 和 LEDC_HPOINT_CH*n* 的和。通过配置 LEDC_DUTY_CH*n*[24:4] 和 LEDC_HPOINT_CH*n* 两个字段，可设置 PWM 输出信号的相对相位和占空比。

置位 LEDC_SIG_OUT_EN_CH*n*，开启 PWM 信号（sig_out*n*）输出；清除 LEDC_SIG_OUT_EN_CH*n*，关闭 PWM 信号输出，输出信号 (sig_out*n*) 输出恒定电平，电平值为 LEDC_IDLE_LV_CH*n*。

LEDC_DUTY_CH*n*[3:0] 通过周期性改变 PWM 输出信号 sig_out*n* 的占空比实现微调。如 LEDC_DUTY_CH*n*[3:0] 不为 0，那么 sig_out*n* 每 16 个周期中，有 LEDC_DUTY_CH*n*[3:0] 个周期的 PWM 脉冲占空比要比 (16 - LEDC_DUTY_CH*n*[3:0]) 个周期的脉冲占空比多一个定时器的计数周期。比如，如果 LEDC_DUTY_CH*n*[24:4] 设为 10，LEDC_DUTY_CH*n*[3:0] 设为 5，则 16 个周期中，有 5 个周期的 PWM 脉冲占空比为 11，剩余 11 个周期的 PWM 脉冲占空比为 10。16 个周期的平均占空比为 10.3125。

如果重新配置 LEDC_TIMER_SEL_CH*n*、LEDC_HPOINT_CH*n*、LEDC_DUTY_CH*n*[24:4] 和 LEDC_SIG_OUT_EN_CH*n* 字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。新配置在计数器下次溢出时生效。LEDC_TIMER*x*_PARA_UP 字段由硬件自动清除。

32.3.4 占空比渐变

PWM 生成器可以渐变 PWM 输出信号的占空比（即由一种占空比逐渐变为为另一种占空比）。每个 PWM 生成器最多可以包含 16 个占空比渐变区间，每个区间可以独立配置占空比渐变方向（增加或减少）、渐变步长、渐变次数和渐变频率。如果开启占空比渐变功能，每个区间的 Lpoint*n* 的值会根据该区间的渐变配置进行变化。

32.3.4.1 线性占空比渐变

通过配置第一个占空比渐变区间的渐变方向、渐变步长、渐变次数和渐变频率，可以产生线性占空比渐变 PWM 信号。

线性占空比渐变 PWM 信号通过以下步骤配置产生：

1. 配置寄存器字段 LEDC_DUTY_CH*n*。该字段用来设置 Lpoint*n* 的初始值。
2. 置位寄存器字段 LEDC_DUTY_START_CH*n*。当该字段置位/清除，占空比渐变会使能/关闭。
3. 配置寄存器 LEDC_CH*n*_GAMMA_WR_REG 的 LEDC_CH*n*_GAMMA_DUTY_INC 字段。当该字段配置为 1/0 时，当前所配置的占空比渐变区间的 Lpoint*n* 会增加/减少。
4. 配置寄存器 LEDC_CH*n*_GAMMA_WR_REG 的 LEDC_CH*n*_GAMMA_DUTY_CYCLE 字段。该字段用来设置当前所配置的占空比渐变区间的 Lpoint*n* 每增加/减少一次，计数器溢出的次数。即当计数器溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次，Lpoint*n*将会增加/减少一次。
5. 配置寄存器 LEDC_CH*n*_GAMMA_WR_REG 的 LEDC_CH*n*_GAMMA_SCALE 字段。该字段用来设置当前所配置的占空比渐变区间的 Lpoint*n*每次增加/减少的量。
6. 配置寄存器 LEDC_CH*n*_GAMMA_WR_REG 的 LEDC_CH*n*_GAMMA_DUTY_NUM 字段。该字段用来设置当前所配置的占空比渐变区间的渐变次数。
7. 配置寄存器 LEDC_CH*n*_GAMMA_WR_ADDR_REG 的 LEDC_CH*n*_GAMMA_WR_ADDR 字段。该字段用来指定步骤 3、4、5、6 所配置的属性属于哪个占空比渐变区间（区间号为 0 到 15）。由于线性占空比渐变只需要配置第一个占空比渐变区间即可，所以 LEDC_CH*n*_GAMMA_WR_ADDR 在这里配置为 0。
8. 将寄存器 LEDC_CH*n*_GAMMA_CONF_REG 的 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为 1。该字段用来指定一次渐变过程中有效的占空比渐变区间的个数（当指定数目的区间都渐变完成后，PWM 信号会停止占空比渐变，PWM 生成器会触发 LEDC_DUTY_CHNG_END_CH*n*_INT 中断）。由于线性占空比渐变只包含一个渐变区间（第一个占空比渐变区间），所以 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为 1。
9. 置位寄存器字段 LEDC_PARA_UP_CH*n* 应用上述步骤的配置。当该字段被置位后，上述的占空比渐变配置会在下一次计数器溢出时生效，PWM 生成器会输出指定配置的线性占空比渐变 PWM 信号（LEDC_PARA_UP_CH*n* 字段会被硬件自动清除）。

当上述配置步骤完成后，每当计数器溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次后，PWM 生成器将 PWM 信号的占空比渐变一次。PWM 信号每次渐变时，其 Lpoint*n* 会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 字段配置）LEDC_CH*n*_GAMMA_SCALE，相应地占空比会增加 /减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 字段配置）

$$\frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}$$

PWM 信号的占空比会渐变 LEDC_CH*n*_GAMMA_DUTY_NUM 次。完成该次数的渐变后，PWM 信号的占空比停止变化并保持以该占空比输出。由于每次渐变时占空比增加/减少的值都相同，所以产生的 PWM 信号为线性占空比渐变信号。

图32-6展示了一个 PWM 信号占空比线性渐变的例子。

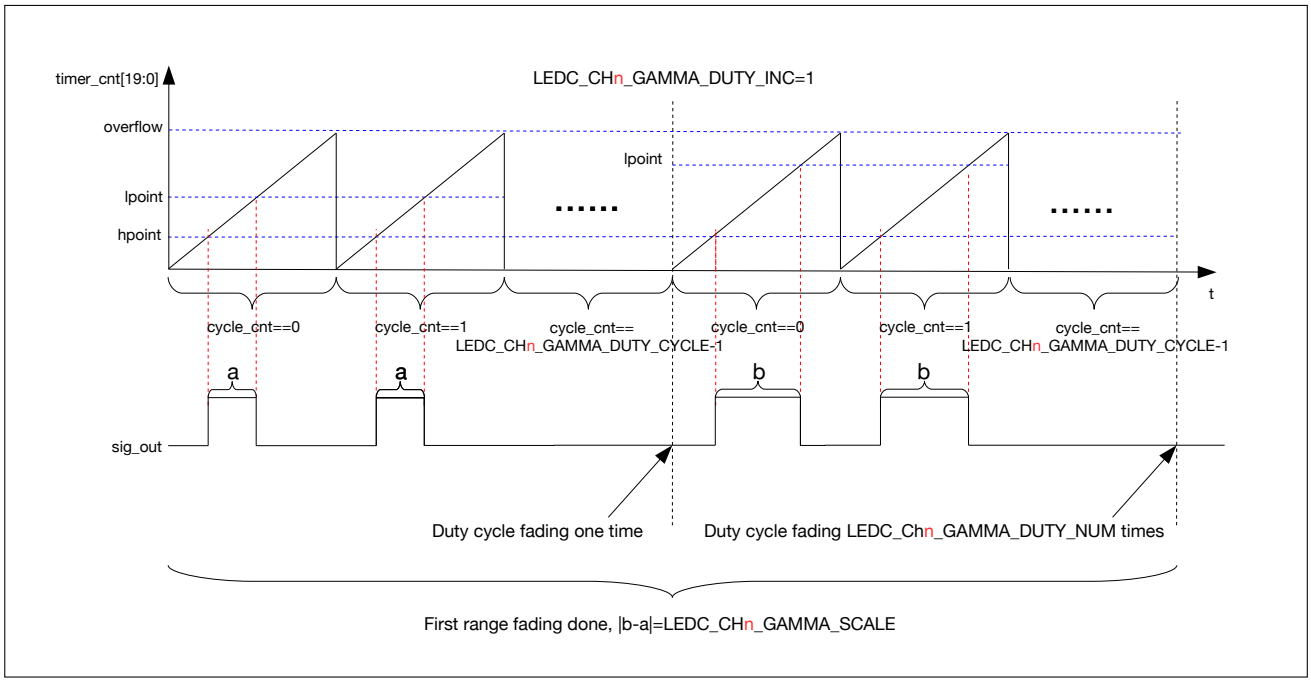


图 32-6. 输出信号占空比线性渐变图

32.3.4.2 伽马曲线渐变

通过配置多个占空比渐变区间的占空比渐变方向，渐变步长，渐变次数和渐变频率，PWM 生成器可以产生伽玛曲线渐变的 PWM 信号。

伽玛曲线渐变可以通过下述步骤配置：

1. 和章节 32.3.4.1 的步骤 1 相同。
2. 和章节 32.3.4.1 的步骤 2 相同。
3. 配置多个占空比渐变区间：
   (a) 为当前所配置的占空比渐变区间配置 LEDC_CH*n*_GAMMA_WR_REG 寄存器的 LEDC_CH*n*_GAMMA_DUTY_INC 字段。
   (b) 为当前所配置的占空比渐变区间配置 LEDC_CH*n*_GAMMA_WR_REG 寄存器的 LEDC_CH*n*_GAMMA_DUTY_CYCLE 字段。

(c) 为当前所配置的占空比渐变区间配置 LEDC_CH*n*_GAMMA_WR_REG 寄存器的 LEDC_CH*n*_GAMMA_SCALE 字段。

(d) 为当前所配置的占空比渐变区间配置 LEDC_CH*n*_GAMMA_WR_REG 寄存器的 LEDC_CH*n*_GAMMA_DUTY_NUM 字段。

(e) 配置 LEDC_CH*n*_GAMMA_WR_ADDR_REG 寄存器的 LEDC_CH*n*_GAMMA_WR_ADDR 字段，指定上述步骤的配置所属的占空比渐变区间的序号（区间序号为 0 到 15）。所配置的占空比渐变区间的序号必须从 0 开始，每配置完一个占空比渐变区间，下一个要配置的占空比渐变区间的序号需要加 1。

(f) 上述步骤配置完毕后，一个占空比渐变区间的配置视为完成，然后可以重复上述的步骤配置其他的占空比渐变区间的参数。每个占空比渐变区间的配置都是独立的，拥有有效配置的区间个数可以是任意的（不能大于 16 个区间）。

4. 当所需的占空比渐变区间都配置完毕后，将 LEDC_CH*n*_GAMMA_CONF_REG 寄存器的 LEDC_CH*n*_GAMMA_ENTRY_NUM 字段配置为步骤 3 中配置过的占空比渐变区间的总个数。

5. 置位寄存器的 LEDC_PARA_UP_CH*n* 字段，应用以上步骤的配置。当该字段被置位后，上述步骤的占空比渐变配置会在下一次计数器溢出时生效，PWM 生成器会按照配置输出伽玛曲线渐变的 PWM 信号（LEDC_PARA_UP_CH*n* 字段会自动被硬件清除）。

经过上述所有的配置步骤，PWM 生成器会产生包含 LEDC_CH*n*_GAMMA_ENTRY_NUM 个占空比渐变区间的 PWM 信号。首先，PWM 信号会根据序号为 0 的占空比渐变区间的配置进行占空比渐变，渐变完成后，按照序号为 1 的占空比渐变区间的配置进行占空比渐变，直到序号为 (LEDC_CH*n*_GAMMA_ENTRY_NUM – 1) 的占空比渐变区间（即最后一个区间）完成渐变。在每个区间中，PWM 信号都是独立变化的。对于每个由 LEDC_CH*n*_GAMMA_WR_ADDR 字段指定序号的占空比渐变区间，PWM 信号会在计数器每溢出 LEDC_CH*n*_GAMMA_DUTY_CYCLE 次后，进行一次占空比渐变，对于每次渐变，其 Lpoint*n* 会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 字段配置）LEDC_CH*n*_GAMMA_SCALE, 相对的，占空比会增加/减少（由 LEDC_CH*n*_GAMMA_DUTY_INC 字段配置）

$$\frac{\text{LEDC\_CH}n\text{\_GAMMA\_SCALE}}{\text{LEDC\_TIMER}x\text{\_DUTY\_RES}}$$

在该区间内，占空比会渐变 LEDC_CH*n*_GAMMA_DUTY_NUM 次，当指定次数的占空比渐变完成后，视为当前区间的占空比渐变完成。

当 LEDC_CH*n*_GAMMA_ENTRY_NUM 个区间的占空比渐变都完成后，PWM 信号会停止占空比渐变，其占空比会保持最后一次渐变的占空比。由于每个占空比渐变区间的渐变过程都是独立不同的线性渐变，因此多个区间的线性占空比渐变就可以拟合成一个特定的伽玛曲线占空比渐变。

图32-7展示了一个伽玛曲线渐变 PWM 信号。

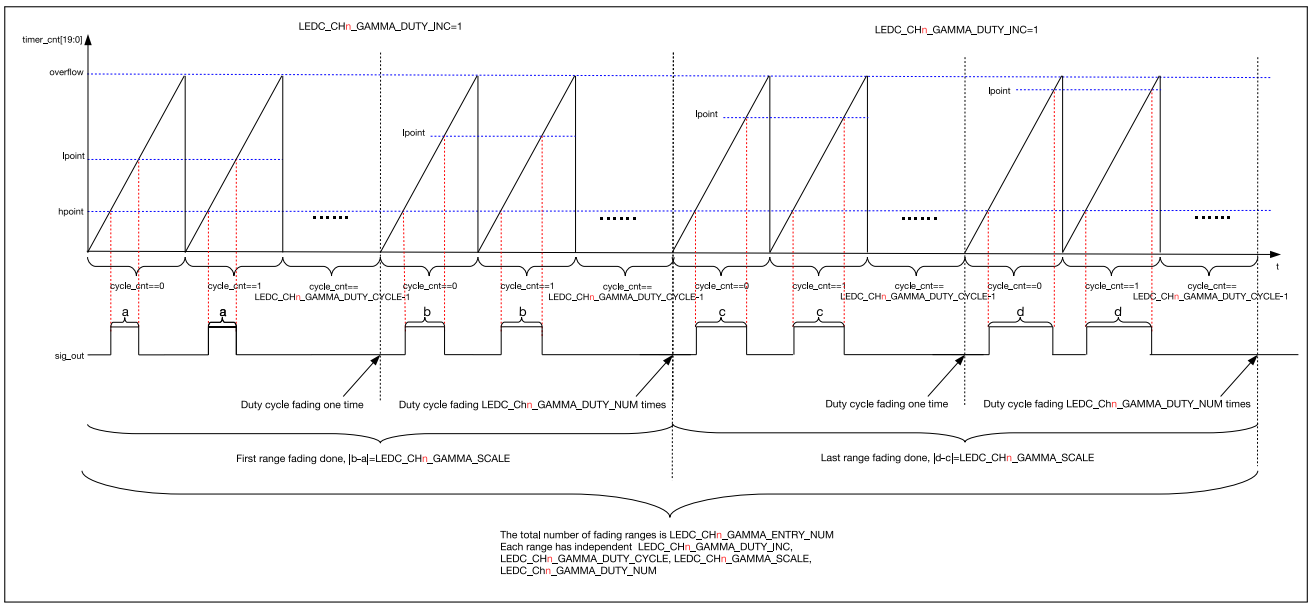


图 32-7. 输出信号占空比伽玛曲线渐变图

32.3.4.3 占空比渐变暂停和恢复

向 LEDC_CH*n*_GAMMA_CONF_REG 寄存器的 LEDC_CH*n*_GAMMA_PAUSE 字段写 1 可以暂停已经开始的占空比渐变过程。一旦将 LEDC_CH*n*_GAMMA_PAUSE 字段置 1，PWM 信号将会保持最近一次渐变后的占空比。

向 LEDC_CH*n*_GAMMA_CONF_REG 寄存器的 LEDC_CH*n*_GAMMA_RESUME 字段写 1 可以恢复被暂停的占空比渐变过程。一旦将 LEDC_CH*n*_GAMMA_RESUME 字段置 1，PWM 信号将会按照暂停时所在的占空比渐变区间的配置，从暂停时的占空比继续进行占空比渐变，直到所有有效的区间都完成指定的占空比渐变（LEDC_CH*n*_GAMMA_RESUME 字段置 1 后，硬件会自动清除 LEDC_CH*n*_GAMMA_PAUSE 字段）。

32.3.5 事件任务矩阵功能

在 ESP32-H2 中，LEDC 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 LEDC 的 ETM 任务，或者通过 LEDC 的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与 LEDC 相关的 ETM 任务和 ETM 事件。

ETM 外设相关的事件和任务的使能由寄存器 LEDC_EVT_TASK_EN0_REG、LEDC_EVT_TASK_EN1_REG 和 LEDC_EVT_TASK_EN2_REG 的相关字段所配置。事件和任务与寄存器 LEDC_EVT_TASK_EN0_REG、LEDC_EVT_TASK_EN1_REG、LEDC_EVT_TASK_EN2_REG 的每个字段的对应关系参照章节 32.5。

LEDC 可接收的 ETM 任务有：

- LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*: 如果 LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN 字段使能，当接收到 LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* 任务后，相应的 LEDC 通道（PWM*n*）将会根据最新配置的 LEDC_CH*n*_GAMMA_SCALE 字段的值，重新开始生成占空比渐变 PWM 信号。
- LEDC_TASK_TIMER*x*_RES_UPDATE: 如果 LEDC_TASK_TIMER*x*_RES_UPDATE_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RES_UPDATE 任务后，相应的定时器（Timer*x*）将会根据最新配置的 LEDC_TIMER*x*_DUTY_RES 字段的值，在下一次计数器溢出时，更新计数器的最大计数值为该值。
- LEDC_TASK_TIMER*x*_CAP: 如果 LEDC_TASK_TIMER*x*_CAP_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_CAP 任务时，相应的定时器（Timer*x*）将会抓取当前计数器的计数值。该计数值将

会存储在寄存器 LEDC_TIMER*x*_CNT_CAP_REG 的 LEDC_TIMER*x*_CNT_CAP 字段。

- LEDC_TASK_SIG_OUT_DIS_CH*n*: 如果 LEDC_TASK_SIG_OUT_DIS_CH*n*_EN 字段使能，当接收到 LEDC_TASK_SIG_OUT_DIS_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）的 PWM 信号将停止输出，该通道的输出信号 (sig_out*n*) 将会变为 LEDC_IDLE_LV_CH*n* 字段所配置电平的固定电平信号，如图 32-2 所示。
- LEDC_TASK_OVF_CNT_RST_CH*n*: 如果 LEDC_TASK_OVF_CNT_RST_CH*n*_EN 字段使能，当接收到 LEDC_TASK_OVF_CNT_RST_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）定时器的计数器溢出次数的计数器值将会被复位为 0。
- LEDC_TASK_TIMER*x*_RST: 如果 LEDC_TASK_TIMER*x*_RST_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RST 任务时，相应的定时器（Timer*x*）的计数器值将会被复位为 0。
- LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE: 如果 LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN 字段使能，当接收到 LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE 任务时，相应的定时器（Timer*x*）将会被交替暂停或恢复。即，当该任务被接收一次时，定时器（Timer*x*）将会暂停，当该任务再被接收一次时，定时器（Timer*x*）将会恢复。
- LEDC_TASK_GAMMA_RESTART_CH*n*: 如果 LEDC_TASK_GAMMA_RESTART_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_RESTART_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会重新开始产生占空比渐变 PWM 信号。
- LEDC_TASK_GAMMA_PAUSE_CH*n*: 如果 LEDC_TASK_GAMMA_PAUSE_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_PAUSE_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会在计数器下一次溢出时停止占空比的渐变。即，当该任务被接收后，相应的 LEDC 通道（PWM*n*）的 PWM 信号的占空比将不会再进行渐变。
- LEDC_TASK_GAMMA_RESUME_CH*n*: 如果 LEDC_TASK_GAMMA_RESUME_CH*n*_EN 字段使能，当接收到 LEDC_TASK_GAMMA_RESUME_CH*n* 任务时，相应的 LEDC 通道（PWM*n*）将会在计数器下一次溢出时恢复占空比的渐变。即，当该任务被接收后，相应的 LEDC 通道（PWM*n*）的 PWM 信号将会按照暂停时的占空比渐变配置继续进行占空比的渐变。

LEDC 可产生的 ETM 事件有：

- LEDC_EVT_DUTY_CHNG_END_CH*n*: 如果 LEDC_EVT_DUTY_CHNG_END_CH*n*_EN 字段使能，当相应的 LEDC 通道（PWM*n*）占空比渐变完成后，将会产生 LEDC_EVT_DUTY_CHNG_END_CH*n* 事件。
- LEDC_EVT_OVF_CNT_PLS_CH*n*: 如果 LEDC_EVT_OVF_CNT_PLS_CH*n*_EN 字段使能，当相应的 LEDC 通道（PWM*n*）的定时器的计数器溢出 LEDC_OVF_NUM_CH*n* + 1 次后，将会产生 LEDC_EVT_OVF_CNT_PLS_CH*n* 事件。
- LEDC_EVT_TIME_OVF_TIMER*x*: 如果 LEDC_EVT_TIME_OVF_TIMER*x*_EN 字段使能，当相应的定时器（Timer*x*）的计数器溢出时，将会产生 LEDC_EVT_TIME_OVF_TIMER*x* 事件。
- LEDC_EVT_TIMER*x*_CMP: 如果 LEDC_EVT_TIME*x*_CMP_EN 字段使能，当相应的定时器（Timer*x*）的计数器值等于寄存器 LEDC_TIMER*x*_CMP_REG 的 LEDC_TIMER*x*_CMP 字段的值时，将会产生 LEDC_EVT_TIMER*x*_CMP 事件。

在具体应用中，LEDC 的 ETM 事件可以用来触发 LEDC 的 ETM 任务，例如，LEDC_EVT_DUTY_CHNG_END_CH*n* 事件可以触发 LEDC_TASK_GAMMA_RESTART_CH*n* 任务，从而在一次渐变完成后直接开始下一次渐变。

32.3.6 中断

- LEDC_OVF_CNT_CH*n*_INT：定时器计数器溢出 LEDC_OVF_NUM_CH*n* + 1 次且寄存器 LEDC_OVF_CNT_EN_CH*n* 置 1 时触发中断。当寄存器 LEDC_INT_ENA_REG 的 LEDC_OVF_CNT_CH*n*_INT_ENA 字段置位后，才能触发该中断。
- LEDC_DUTY_CHNG_END_CH*n*_INT：PWM 生成器渐变完成后触发中断。当寄存器 LEDC_INT_ENA_REG 的 LEDC_DUTY_CHNG_END_CH*n*_INT_ENA 字段置位后，才能触发该中断。
- LEDC_TIMER*x*_OVF_INT：定时器达到最大计数值时触发中断。当寄存器 LEDC_INT_ENA_REG 的 LEDC_TIMER*x*_OVF_INT_ENA 字段置位后，才能触发该中断。

32.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
LEDC_CH0_CONF0_REG	通道 0 的配置寄存器 0	0x0000	varies
LEDC_CH0_CONF1_REG	通道 0 的配置寄存器 1	0x000C	R/ W/ SC
LEDC_CH1_CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
LEDC_CH1_CONF1_REG	通道 1 的配置寄存器 1	0x0020	R/ W/ SC
LEDC_CH2_CONF0_REG	通道 2 的配置寄存器 0	0x0028	varies
LEDC_CH2_CONF1_REG	通道 2 的配置寄存器 1	0x0034	R/ W/ SC
LEDC_CH3_CONF0_REG	通道 3 的配置寄存器 0	0x003C	varies
LEDC_CH3_CONF1_REG	通道 3 的配置寄存器 1	0x0048	R/ W/ SC
LEDC_CH4_CONF0_REG	通道 4 的配置寄存器 0	0x0050	varies
LEDC_CH4_CONF1_REG	通道 4 的配置寄存器 1	0x005C	R/ W/ SC
LEDC_CH5_CONF0_REG	通道 5 的配置寄存器 0	0x0064	varies
LEDC_CH5_CONF1_REG	通道 5 的配置寄存器 1	0x0070	R/ W/ SC
LEDC_EVT_TASK_EN0_REG	LEDC 事件任务使能寄存器 0	0x01A0	R/W
LEDC_EVT_TASK_EN1_REG	LEDC 事件任务使能寄存器 1	0x01A4	R/W
LEDC_EVT_TASK_EN2_REG	LEDC 事件任务使能寄存器 2	0x01A8	R/W
LEDC_TIMER0_CMP_REG	LEDC 定时器 0 的比较值	0x01B0	R/W
LEDC_TIMER1_CMP_REG	LEDC 定时器 1 的比较值	0x01B4	R/W
LEDC_TIMER2_CMP_REG	LEDC 定时器 2 的比较值	0x01B8	R/W
LEDC_TIMER3_CMP_REG	LEDC 定时器 3 的比较值	0x01BC	R/W
LEDC_TIMER0_CNT_CAP_REG	LEDC 定时器 0 计数值抓取寄存器	0x01C0	RO
LEDC_TIMER1_CNT_CAP_REG	LEDC 定时器 1 计数值抓取寄存器	0x01C4	RO
LEDC_TIMER2_CNT_CAP_REG	LEDC 定时器 2 计数值抓取寄存器	0x01C8	RO
LEDC_TIMER3_CNT_CAP_REG	LEDC 定时器 3 计数值抓取寄存器	0x01CC	RO
LEDC_CONF_REG	LEDC 全局配置寄存器	0x01F0	R/W
Hpoint 寄存器			
LEDC_CH0_HPOINT_REG	通道 0 的 Hpoint 寄存器	0x0004	R/W
LEDC_CH1_HPOINT_REG	通道 1 的 Hpoint 寄存器	0x0018	R/W
LEDC_CH2_HPOINT_REG	通道 2 的 Hpoint 寄存器	0x002C	R/W
LEDC_CH3_HPOINT_REG	通道 3 的 Hpoint 寄存器	0x0040	R/W
LEDC_CH4_HPOINT_REG	通道 4 的 Hpoint 寄存器	0x0054	R/W
LEDC_CH5_HPOINT_REG	通道 5 的 Hpoint 寄存器	0x0068	R/W

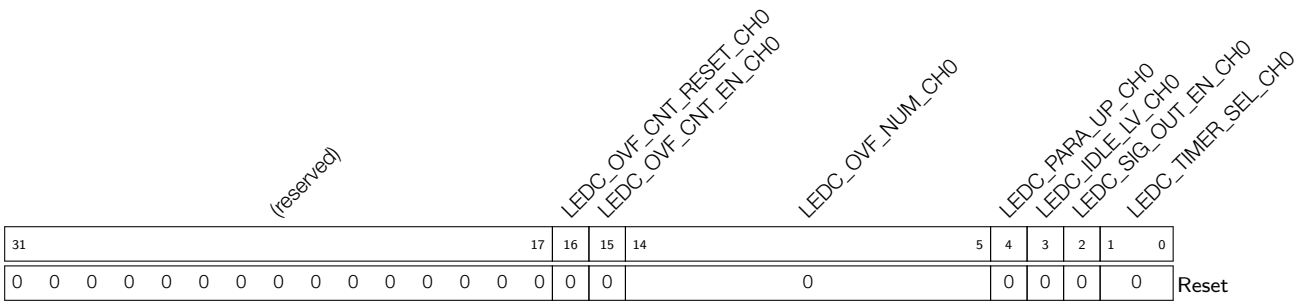
名称	描述	地址	访问
占空比寄存器			
LEDC_CH0_DUTY_REG	通道 0 的初始占空比	0x0008	R/W
LEDC_CH0_DUTY_R_REG	通道 0 的当前占空比	0x0010	RO
LEDC_CH1_DUTY_REG	通道 1 的初始占空比	0x001C	R/W
LEDC_CH1_DUTY_R_REG	通道 1 的当前占空比	0x0024	RO
LEDC_CH2_DUTY_REG	通道 2 的初始占空比	0x0030	R/W
LEDC_CH2_DUTY_R_REG	通道 2 的当前占空比	0x0038	RO
LEDC_CH3_DUTY_REG	通道 3 的初始占空比	0x0044	R/W
LEDC_CH3_DUTY_R_REG	通道 3 的当前占空比	0x004C	RO
LEDC_CH4_DUTY_REG	通道 4 的初始占空比	0x0058	R/W
LEDC_CH4_DUTY_R_REG	通道 4 的当前占空比	0x0060	RO
LEDC_CH5_DUTY_REG	通道 5 的初始占空比	0x006C	R/W
LEDC_CH5_DUTY_R_REG	通道 5 的当前占空比	0x0074	RO
定时器寄存器			
LEDC_TIMER0_CONF_REG	配置定时器 0	0x00A0	varies
LEDC_TIMER0_VALUE_REG	配置定时器 0 的当前计数器数值	0x00A4	RO
LEDC_TIMER1_CONF_REG	配置定时器 1	0x00A8	varies
LEDC_TIMER1_VALUE_REG	配置定时器 1 的当前计数器数值	0x00AC	RO
LEDC_TIMER2_CONF_REG	配置定时器 2	0x00B0	varies
LEDC_TIMER2_VALUE_REG	配置定时器 2 的当前计数器数值	0x00B4	RO
LEDC_TIMER3_CONF_REG	配置定时器 3	0x00B8	varies
LEDC_TIMER3_VALUE_REG	配置定时器 3 的当前计数器数值	0x00BC	RO
中断寄存器			
LEDC_INT_RAW_REG	原始中断寄存器	0x00C0	R/ WTC/ SS
LEDC_INT_ST_REG	屏蔽中断寄存器	0x00C4	RO
LEDC_INT_ENA_REG	中断使能寄存器	0x00C8	R/W
LEDC_INT_CLR_REG	中断清除寄存器	0x00CC	WT
伽马 RAM 寄存器			
LEDC_CH0_GAMMA_WR_REG	LEDC 通道 0 写伽马 RAM 寄存器	0x0100	R/W
LEDC_CH0_GAMMA_WR_ADDR_REG	LEDC 通道 0 伽马 RAM 写地址寄存器	0x0104	R/W
LEDC_CH0_GAMMA_RD_ADDR_REG	LEDC 通道 0 伽马 RAM 读地址寄存器	0x0108	R/W
LEDC_CH0_GAMMA_RD_DATA_REG	LEDC 通道 0 读伽马 RAM 寄存器	0x010C	RO
LEDC_CH1_GAMMA_WR_REG	LEDC 通道 1 写伽马 RAM 寄存器	0x0110	R/W
LEDC_CH1_GAMMA_WR_ADDR_REG	LEDC 通道 1 伽马 RAM 写地址寄存器	0x0114	R/W
LEDC_CH1_GAMMA_RD_ADDR_REG	LEDC 通道 1 伽马 RAM 读地址寄存器	0x0118	R/W
LEDC_CH1_GAMMA_RD_DATA_REG	LEDC 通道 1 读伽马 RAM 寄存器	0x011C	RO
LEDC_CH2_GAMMA_WR_REG	LEDC 通道 2 写伽马 RAM 寄存器	0x0120	R/W
LEDC_CH2_GAMMA_WR_ADDR_REG	LEDC 通道 2 伽马 RAM 写地址寄存器	0x0124	R/W
LEDC_CH2_GAMMA_RD_ADDR_REG	LEDC 通道 2 伽马 RAM 读地址寄存器	0x0128	R/W
LEDC_CH2_GAMMA_RD_DATA_REG	LEDC 通道 2 读伽马 RAM 寄存器	0x012C	RO
LEDC_CH3_GAMMA_WR_REG	LEDC 通道 3 写伽马 RAM 寄存器	0x0130	R/W

名称	描述	地址	访问
LEDC_CH3_GAMMA_WR_ADDR_REG	LEDC 通道 3 伽马 RAM 写地址寄存器	0x0134	R/W
LEDC_CH3_GAMMA_RD_ADDR_REG	LEDC 通道 3 伽马 RAM 读地址寄存器	0x0138	R/W
LEDC_CH3_GAMMA_RD_DATA_REG	LEDC 通道 3 读伽马 RAM 寄存器	0x013C	RO
LEDC_CH4_GAMMA_WR_REG	LEDC 通道 4 写伽马 RAM 寄存器	0x0140	R/W
LEDC_CH4_GAMMA_WR_ADDR_REG	LEDC 通道 4 伽马 RAM 写地址寄存器	0x0144	R/W
LEDC_CH4_GAMMA_RD_ADDR_REG	LEDC 通道 4 伽马 RAM 读地址寄存器	0x0148	R/W
LEDC_CH4_GAMMA_RD_DATA_REG	LEDC 通道 4 读伽马 RAM 寄存器	0x014C	RO
LEDC_CH5_GAMMA_WR_REG	LEDC 通道 5 写伽马 RAM 寄存器	0x0150	R/W
LEDC_CH5_GAMMA_WR_ADDR_REG	LEDC 通道 5 伽马 RAM 写地址寄存器	0x0154	R/W
LEDC_CH5_GAMMA_RD_ADDR_REG	LEDC 通道 5 伽马 RAM 读地址寄存器	0x0158	R/W
LEDC_CH5_GAMMA_RD_DATA_REG	LEDC 通道 5 读伽马 RAM 寄存器	0x015C	RO
伽马曲线配置寄存器			
LEDC_CH0_GAMMA_CONF_REG	LEDC 通道 0 伽马曲线配置寄存器	0x0180	varies
LEDC_CH1_GAMMA_CONF_REG	LEDC 通道 1 伽马曲线配置寄存器	0x0184	varies
LEDC_CH2_GAMMA_CONF_REG	LEDC 通道 2 伽马曲线配置寄存器	0x0188	varies
LEDC_CH3_GAMMA_CONF_REG	LEDC 通道 3 伽马曲线配置寄存器	0x018C	varies
LEDC_CH4_GAMMA_CONF_REG	LEDC 通道 4 伽马曲线配置寄存器	0x0190	varies
LEDC_CH5_GAMMA_CONF_REG	LEDC 通道 5 伽马曲线配置寄存器	0x0194	varies
版本寄存器			
LEDC_DATE_REG	版本控制寄存器	0x01FC	R/W

32.5 寄存器

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

Register 32.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14n*)**



LEDC_TIMER_SEL_CH*n* 配置通道 *n* 选取的定时器。

0：定时器 0

1：定时器 1

2：定时器 2

3：定时器 3

(R/W)

LEDC_SIG_OUT_EN_CH*n* 配置是否开启通道 *n* 的信号输出。

0：关闭

1：开启

(R/W)

LEDC_IDLE_LV_CH*n* 配置通道 *n* 关闭（即 LEDC_SIG_OUT_EN_CH*n* 为 0）时的输出电平值。

(R/W)

LEDC_PARA_UP_CH*n* 配置是否更新以下字段：LEDC_HPOINT_CH*n*、LEDC_DUTY_START_CH*n*、LEDC_SIG_OUT_EN_CH*n*、LEDC_TIMER_SEL_CH*n*、LEDC_OVF_CNT_EN_CH*n*

0：无效值，没有作用

1：更新

该字段由硬件自清。

(WT/SC)

LEDC_OVF_NUM_CH*n* 配置定时器的最大溢出次数减 1。

通道 *n* 的定时器溢出次数达到 (LEDC_OVF_NUM_CH*n* +1) 次时，触发 LEDC_OVF_CNT_CH*n*_INT 中断。(R/W)

LEDC_OVF_CNT_EN_CH*n* 配置是否开启通道 *n* 的溢出次数计数器。

0：关闭

1：开启

(R/W)

见下页...

Register 32.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)

接上页...

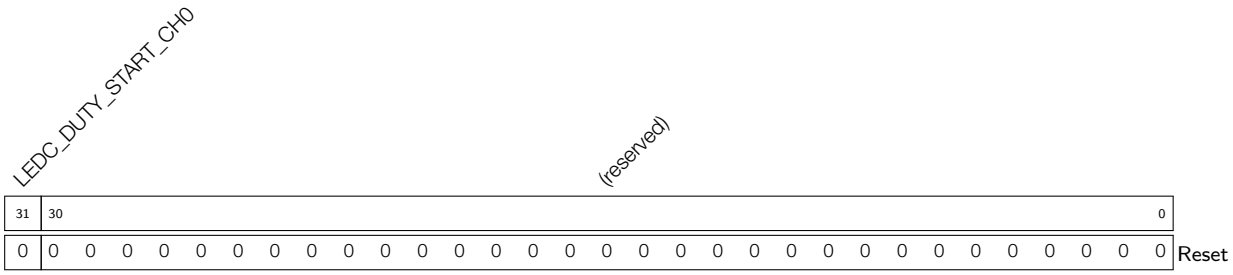
LEDC_OVF_CNT_RESET_CH*n* 配置是否复位通道 *n* 的溢出次数计数器。

0：无效值，没有作用

1：复位

(WT)

Register 32.2. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+0x14**n*)



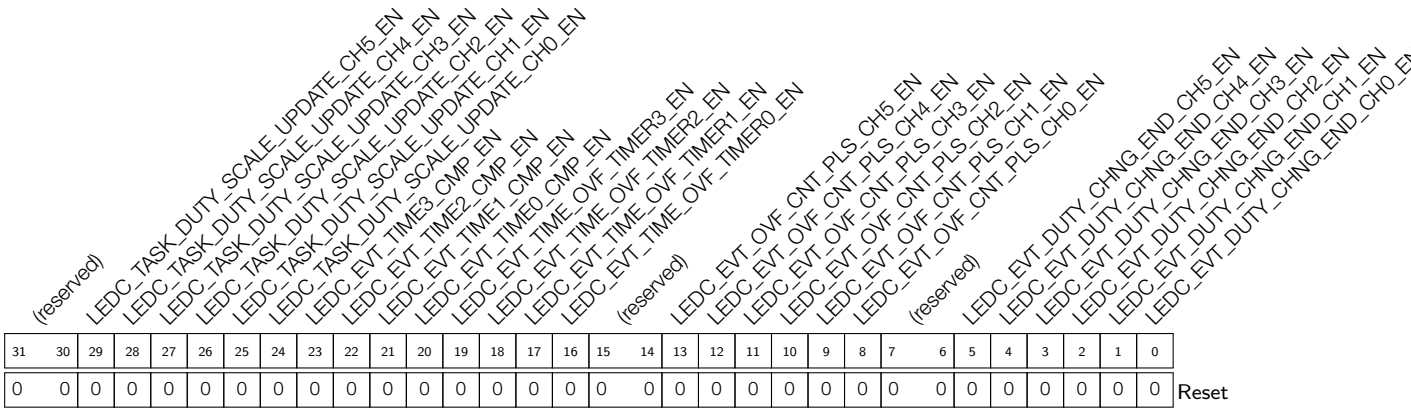
LEDC_DUTY_START_CH*n* 配置是否开启占空比渐变。

0：关闭

1：开启

(R/W/SC)

Register 32.3. LEDC_EVT_TASK_EN0_REG (0x01A0)



LEDC_EVT_DUTY_CHNG_END_CH*n*_EN 配置是否使能 LEDC_EVT_DUTY_CHNG_END_CH*n* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_OVF_CNT_PLS_CH*n*_EN 配置是否使能 LEDC_EVT_OVF_CNT_PLS_CH*n* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_TIME_OVF_TIMER*x*_EN 配置是否使能 LEDC_EVT_TIME_OVF_TIMER*x* 事件。

0：关闭

1：使能

(R/W)

LEDC_EVT_TIME*n*_CMP_EN 配置是否使能 LEDC_EVT_TIME*n*_CMP 事件。

0：关闭

1：使能

(R/W)

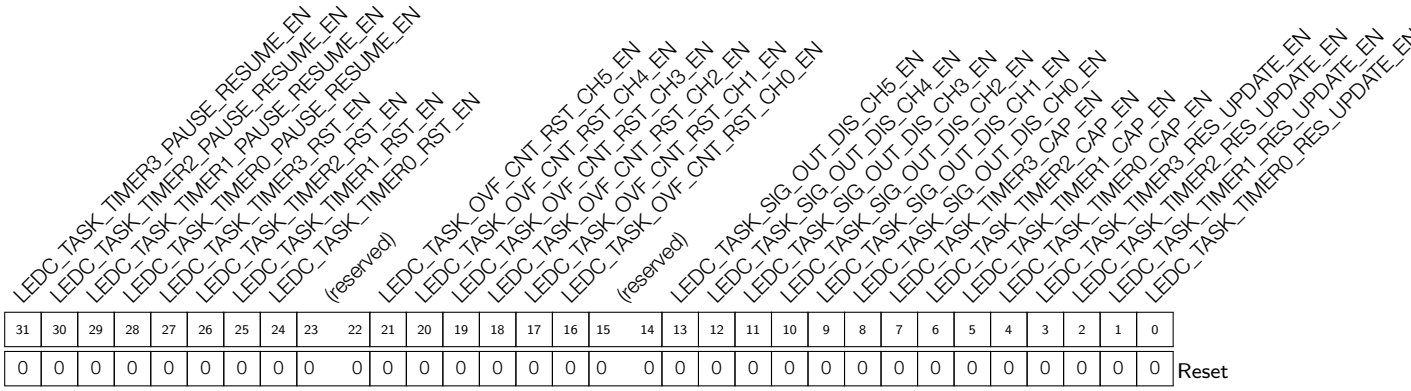
LEDC_TASK_DUTY_SCALE_UPDATE_CH*n*_EN 配置是否使能 LEDC_TASK_DUTY_SCALE_UPDATE_CH*n* 任务。

0：关闭

1：使能

(R/W)

Register 32.4. LEDC_EVT_TASK_EN1_REG (0x01A4)



LEDC_TASK_TIMER*x*_RES_UPDATE_EN 配置是否使能 LEDC_TASK_TIMER*x*_RES_UPDATE 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_TIMER*x*_CAP_EN 配置是否使能 LEDC_TASK_TIMER*x*_CAP 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_SIG_OUT_DIS_CH*n*_EN 配置是否使能 LEDC_TASK_SIG_OUT_DIS_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_OVF_CNT_RST_CH*n*_EN 配置是否使能 LEDC_TASK_OVF_CNT_RST_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_TIMER*x*_RST_EN 配置是否使能 LEDC_TASK_TIMER*x*_RST 任务。

0：关闭

1：使能

(R/W)

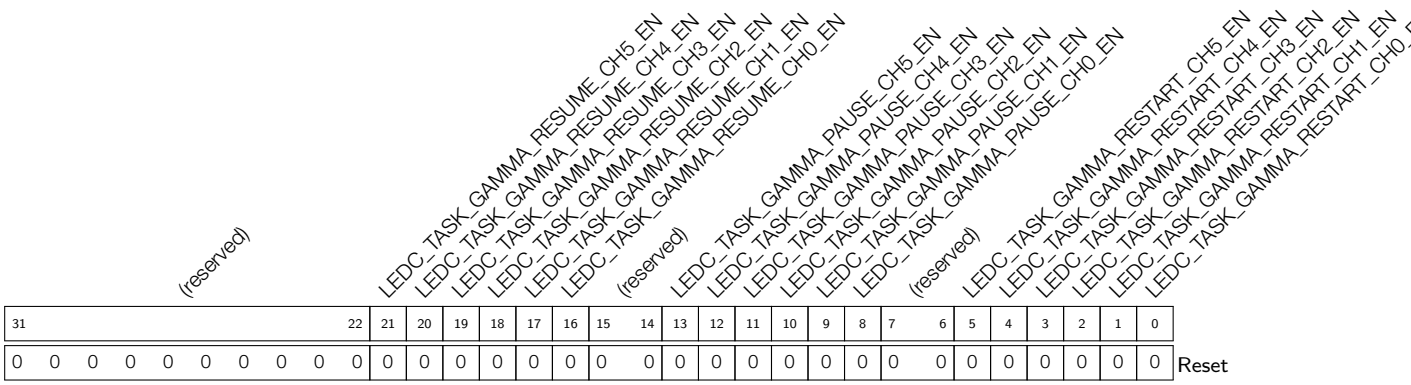
LEDC_TASK_TIMER*x*_PAUSE_RESUME_EN 配置是否使能 LEDC_TASK_TIMER*x*_RESUME 和 LEDC_TASK_TIMER*x*_PAUSE 任务。

0：关闭

1：使能

(R/W)

Register 32.5. LEDC_EVT_TASK_EN2_REG (0x01A8)



LEDC_TASK_GAMMA_RESTART_CH*n*_EN 配置是否使能 LEDC_TASK_GAMMA_RESTART_CH*n* 任务。

0：关闭

1：使能

(R/W)

LEDC_TASK_GAMMA_PAUSE_CH*n*_EN 配置是否使能 LEDC_TASK_GAMMA_PAUSE_CH*n* 任务。

0：关闭

1：使能

(R/W)

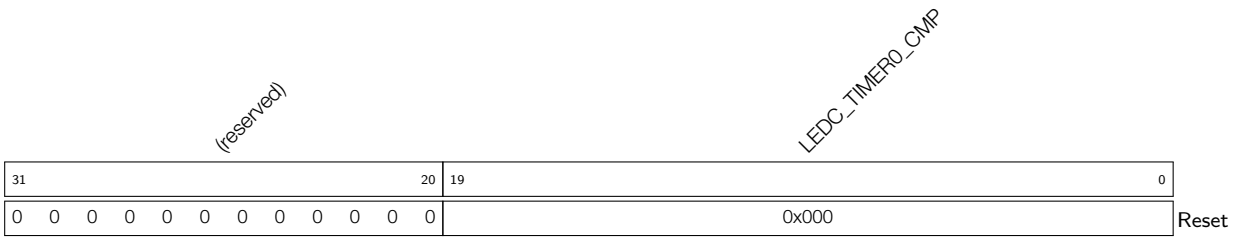
LEDC_TASK_GAMMA_RESUME_CH*n*_EN 配置是否使能 LEDC_TASK_GAMMA_RESUME_CH*n* 任务。

0：关闭

1：使能

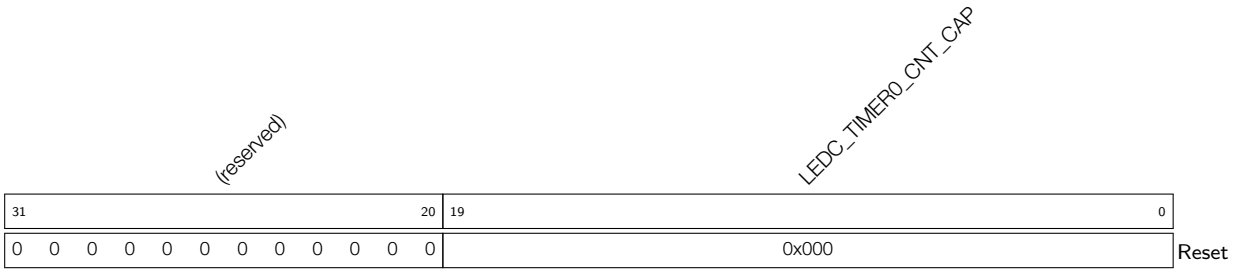
(R/W)

Register 32.6. LEDC_TIMER*x*_CMP_REG (*x*: 0-3) (0x01B0+0x4n*)**



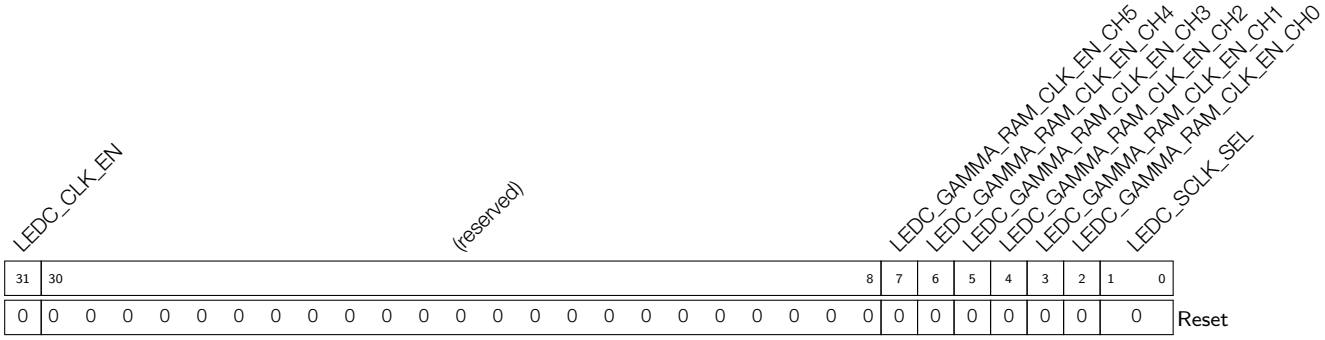
LEDC_TIMER*x*_CMP 配置 LEDC 定时器 *n* 的比较值。(R/W)

Register 32.7. LEDC_TIMERx_CNT_CAP_REG (x: 0-3) (0x01C0+0x4*n)



LEDC_TIMERx_CNT_CAP 表示捕捉到的 LEDC 定时器 *n* 的计数器值。(RO)

Register 32.8. LEDC_CONF_REG (0x01F0)



LEDC_SCLK_SEL 配置四个定时器的时钟源。

0：PLL_F96M_CLK

1：RC_FAST_CLK

2：XTAL_CLK

3：无效值，没有作用

(R/W)

LEDC_GAMMA_RAM_CLK_EN_CHn 配置伽马 RAM 时钟开启的时间。

0：仅在应用读或写伽马 RAM 时开启时钟

1：强制为伽马 RAM 开启时钟

(R/W)

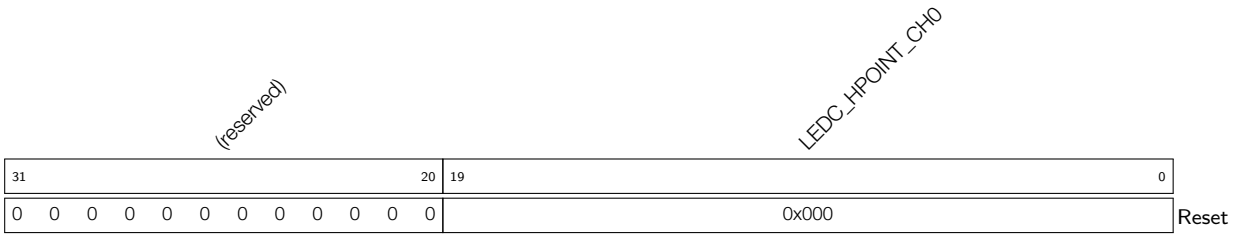
LEDC_CLK_EN 配置寄存器时钟开启的时间。

0：仅在应用写寄存器时开启时钟

1：强制为寄存器开启时钟

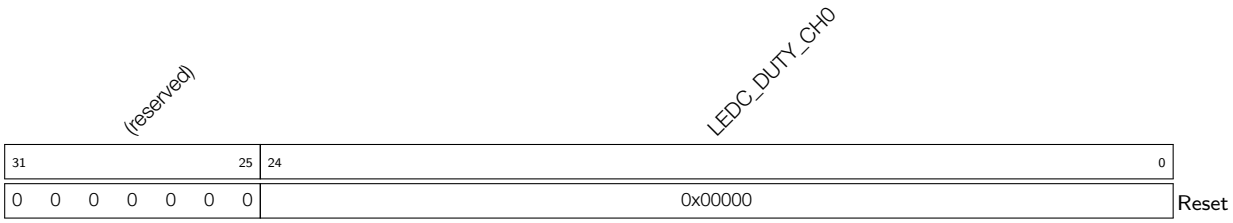
(R/W)

Register 32.9. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+0x14*n*)



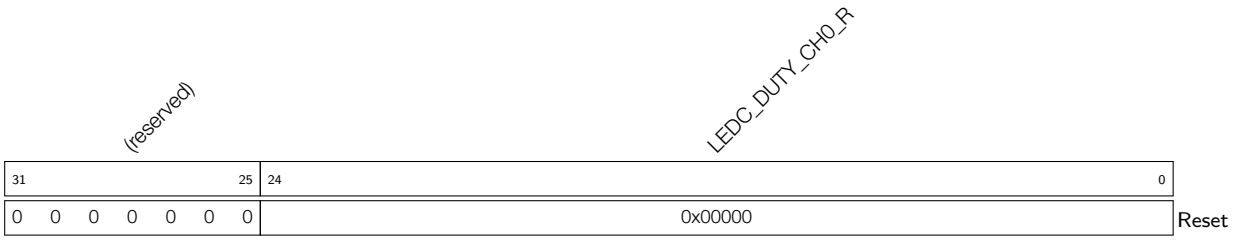
LEDC_HPOINT_CH*n* 配置 Hpoint 的值。(R/W)

Register 32.10. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+0x14*n*)



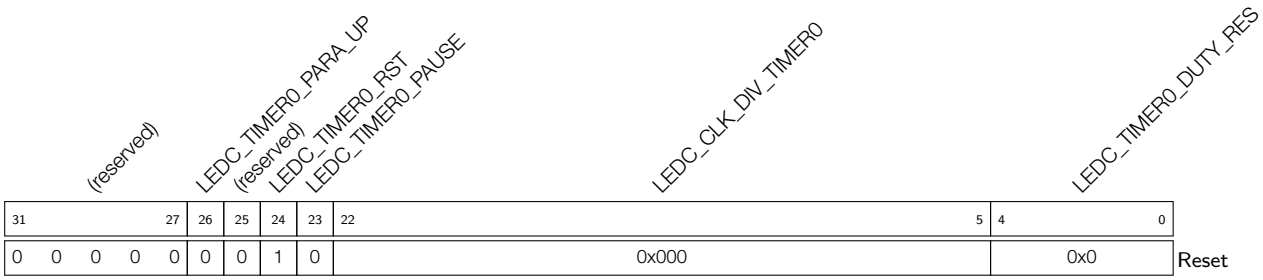
LEDC_DUTY_CH*n* 配置 Lpoint 的初始值。(R/W)

Register 32.11. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+0x14*n*)



LEDC_DUTY_CH*n*_R 表示通道 *n* 输出信号的当前占空比。(RO)

Register 32.12. LEDC_TIMER*x*_CONF_REG (*x*: 0-3) (0x00A0+0x8n*)**



LEDC_TIMER*x*_DUTY_RES 配置占空比精度（定时器 *n* 计数器的位宽）。(R/W)

LEDC_CLK_DIV_TIMER*x* 配置定时器 *n* 中分频器的分频系数。

低 8 位为小数部分，高 10 位为整数部分。(R/W)

LEDC_TIMER*x*_PAUSE 配置是否暂停定时器 *n* 的计数器。

0：不暂停

1：暂停

(R/W)

LEDC_TIMER*x*_RST 配置是否复位定时器 *n* 的计数器（复位后，计数器的值为 0。）。

0：不复位

1：复位

(R/W)

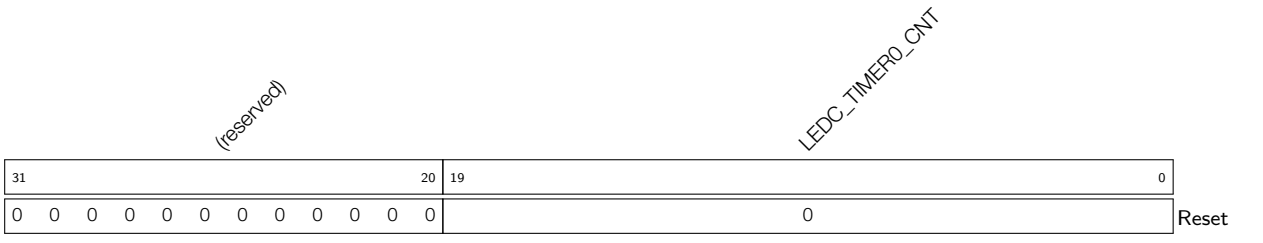
LEDC_TIMER*x*_PARA_UP 配置是否更新 LEDC_CLK_DIV_TIMER*x* 和 LEDC_TIMER*x*_DUTY_RES 字段。

0：无效值，没有作用

1：更新

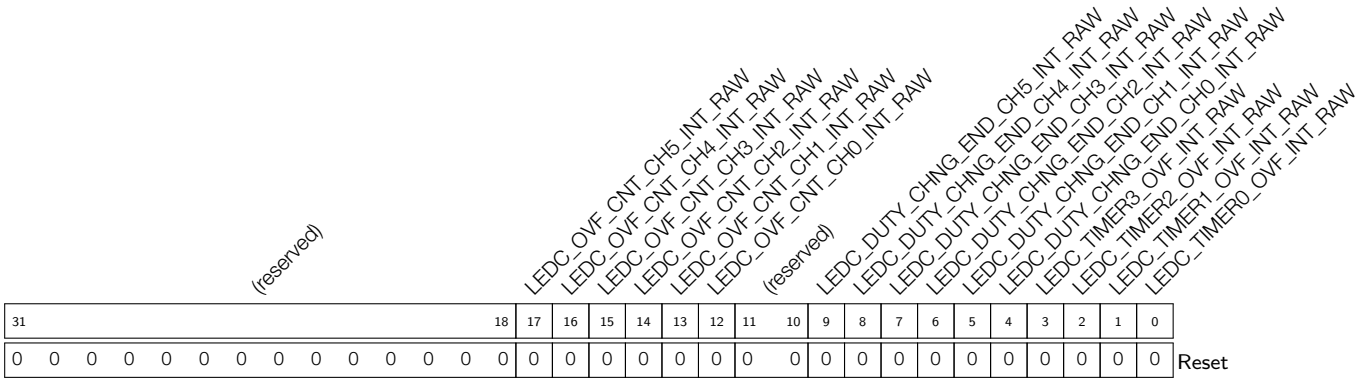
(WT)

Register 32.13. LEDC_TIMER*x*_VALUE_REG (*x*: 0-3) (0x00A4+0x8n*)**



LEDC_TIMER*x*_CNT 表示定时器 *n* 计数器的当前值。(RO)

Register 32.14. LEDC_INT_RAW_REG (0x00C0)

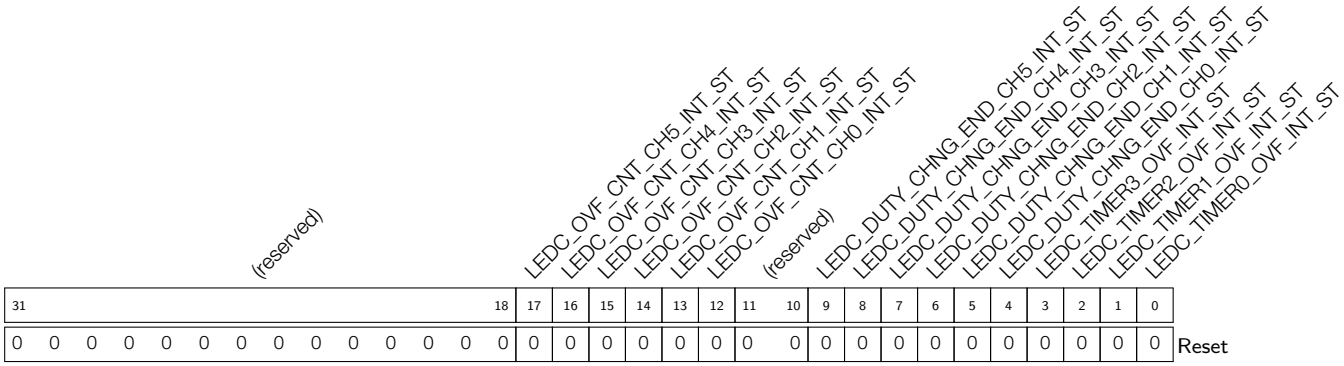


LEDC_TIMER*x*_OVF_INT_RAW LEDC_TIMER*x*_OVF_INT 的原始中断状态。(R/WTC/SS)

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

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

Register 32.15. LEDC_INT_ST_REG (0x00C4)

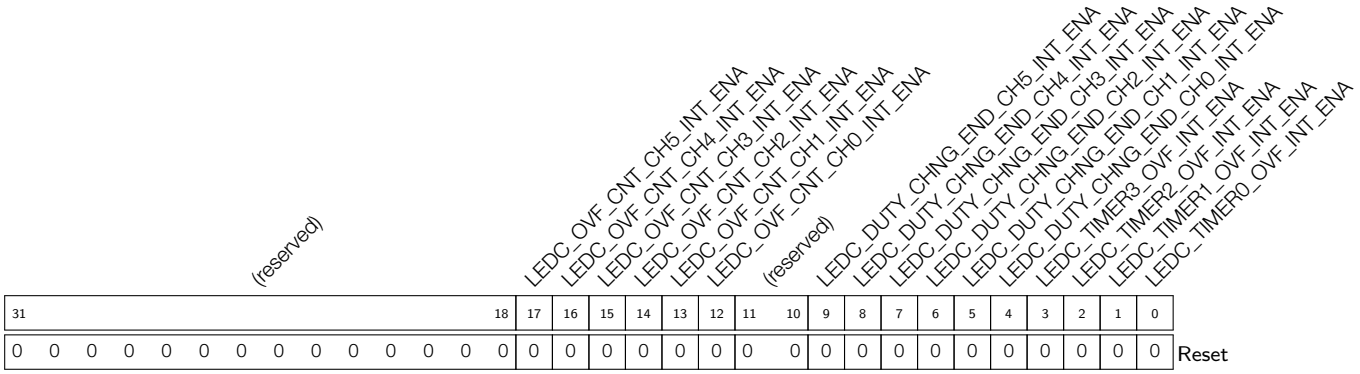


LEDC_TIMER*x*_OVF_INT_ST LEDC_TIMER*x*_OVF_INT 的屏蔽中断状态。仅在 LEDC_TIMER*x*_OVF_INT_ENA 为 1 时有效。(RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST LEDC_DUTY_CHNG_END_CH*n*_INT 的屏蔽中断状态。仅在 LEDC_DUTY_CHNG_END_CH*n*_INT_ENA 为 1 时有效。(RO)

LEDC_OVF_CNT_CH*n*_INT_ST LEDC_OVF_CNT_CH*n*_INT 的屏蔽中断状态。仅在 LEDC_OVF_CNT_CH*n*_INT_ENA 为 1 时有效。(RO)

Register 32.16. LEDC_INT_ENA_REG (0x00C8)

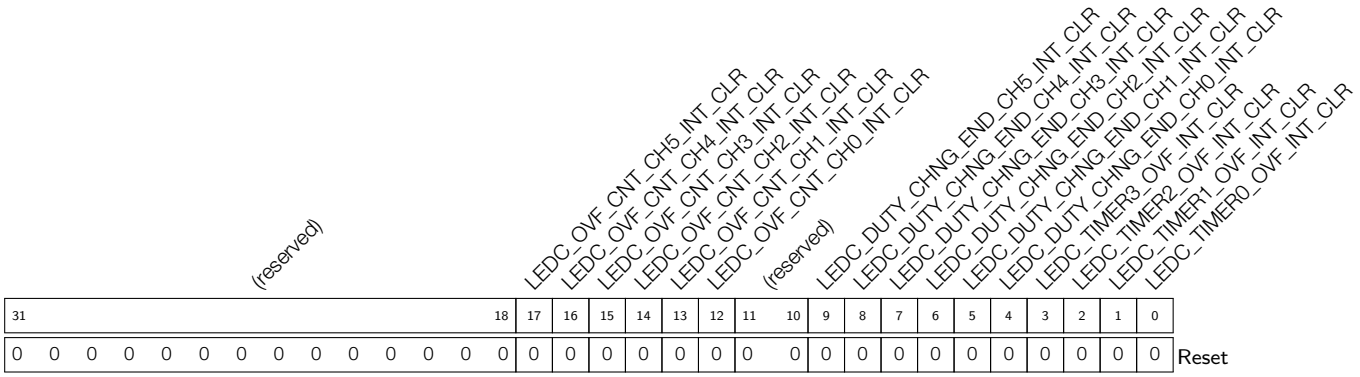


LEDC_TIMER*x*_OVF_INT_ENA 写 1 使能 LEDC_TIMER*x*_OVF_INT。(R/W)

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

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

Register 32.17. LEDC_INT_CLR_REG (0x00CC)

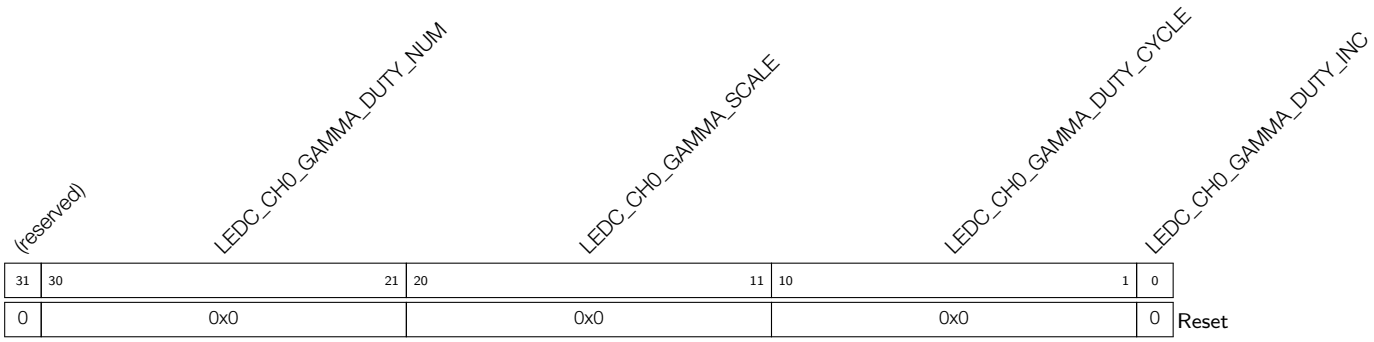


LEDC_TIMER*x*_OVF_INT_CLR 写 1 清除 LEDC_TIMER*x*_OVF_INT。(WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR 写 1 清除 LEDC_DUTY_CHNG_END_CH*n*_INT。(WT)

LEDC_OVF_CNT_CH*n*_INT_CLR 写 1 清除 LEDC_OVF_CNT_CH*n*_INT。(WT)

Register 32.18. LEDC_CH*n*_GAMMA_WR_REG (*n*: 0-5) (0x0100+0x10*n)



LEDC_CH*n*_GAMMA_DUTY_INC 配置通道 *n* 上 PWM 信号的占空比渐变的方向。

0：减少

1：增加

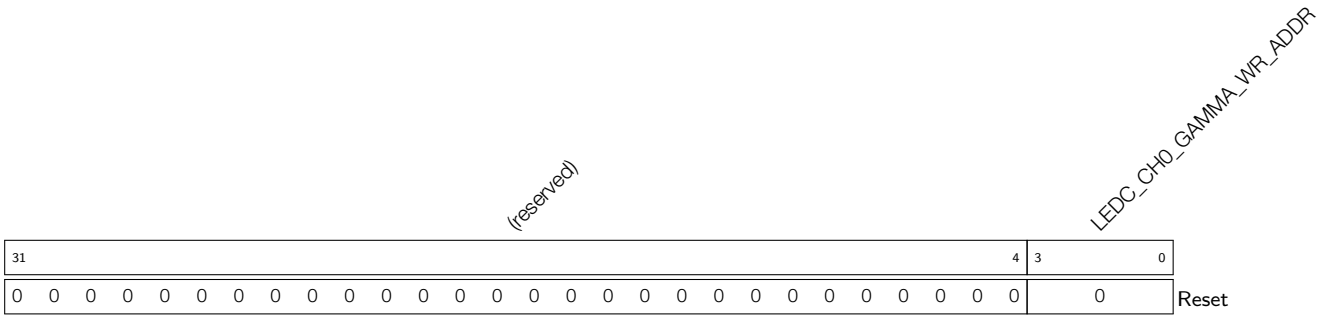
(R/W)

LEDC_CH*n*_GAMMA_DUTY_CYCLE 配置触发一次占空比变化所需的计数器溢出次数。(R/W)

LEDC_CH*n*_GAMMA_SCALE 配置 Lpoint*n* 每次增加/减少的量。(R/W)

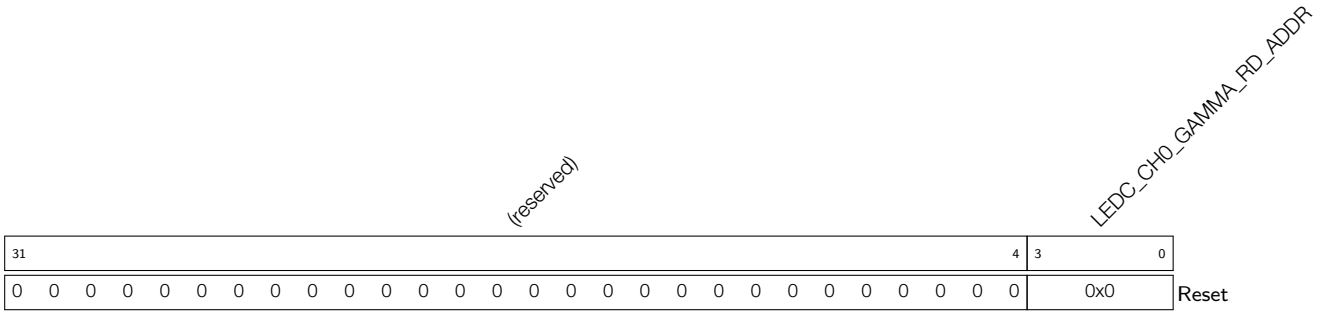
LEDC_CH*n*_GAMMA_DUTY_NUM 配置渐变区间内的渐变次数。(R/W)

Register 32.19. LEDC_CH*n*_GAMMA_WR_ADDR_REG (*n*: 0-5) (0x0104+0x10*n)



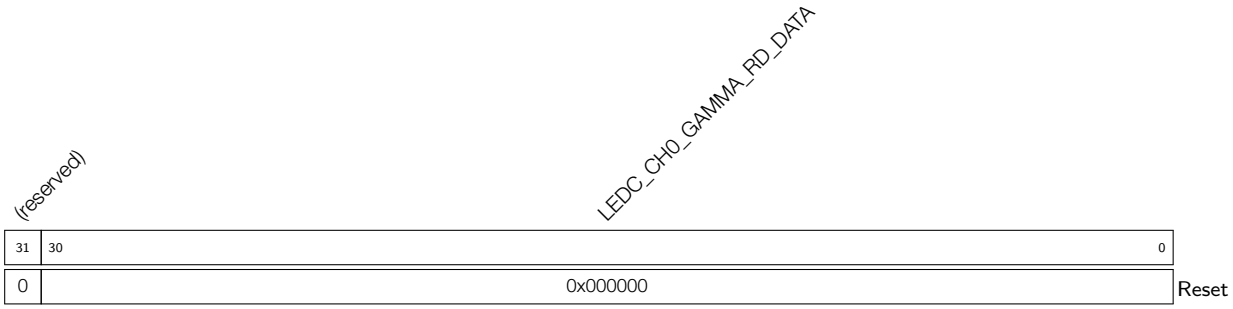
LEDC_CH*n*_GAMMA_WR_ADDR 配置通道 *n* 伽马 RAM 的写地址。(R/W)

Register 32.20. LEDC_CH*n*_GAMMA_RD_ADDR_REG (*n*: 0-5) (0x0108+0x10n*)**



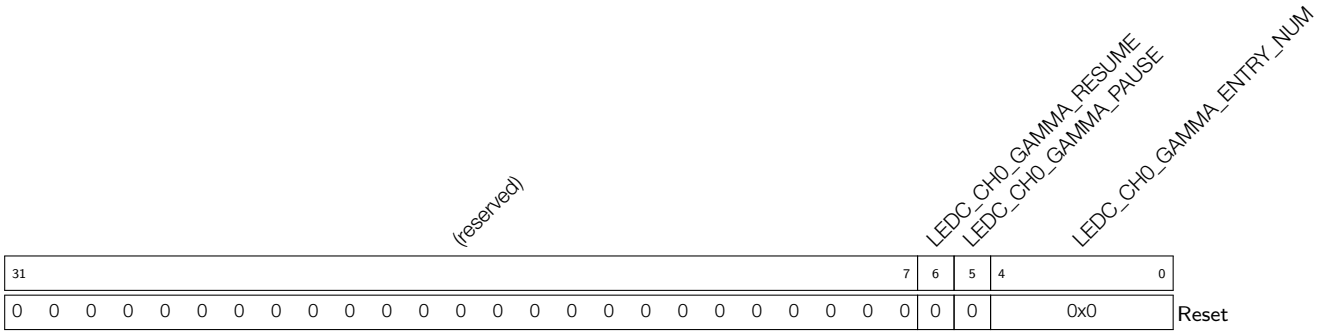
LEDC_CH*n*_GAMMA_RD_ADDR 配置通道 *n* 伽马 RAM 的读地址。(R/W)

Register 32.21. LEDC_CH*n*_GAMMA_RD_DATA_REG (*n*: 0-5) (0x010C+0x10n*)**



LEDC_CH*n*_GAMMA_RD_DATA 表示从通道 *n* 伽马 RAM 读取的数据。(RO)

Register 32.22. LEDC_CH*n*_GAMMA_CONF_REG (*n*: 0-5) (0x0180+0x4*n)



LEDC_CH*n*_GAMMA_ENTRY_NUM 配置占空比渐变区间的数量，最大值为 16。(R/W)

LEDC_CH*n*_GAMMA_PAUSE 配置是否暂停占空比渐变。

0：无效值，没有作用

1：暂停

(WT)

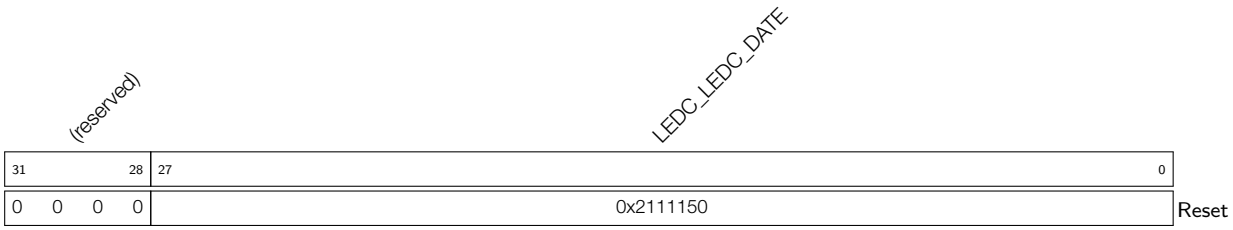
LEDC_CH*n*_GAMMA_RESUME 配置是否恢复已暂停的占空比渐变。

0：无效值，没有作用

1：恢复

(WT)

Register 32.23. LEDC_DATE_REG (0x01FC)



LEDC_LEDC_DATE 版本控制寄存器。(R/W)

33 电机控制脉宽调制器 (MCPWM)

33.1 概述

电机控制脉宽调制器 (MCPWM) 外设用于电机和电源控制。该外设提供了 6 个 PWM 输出，可在几种拓扑结构中运行。常见的拓扑结构之一是用一对 PWM 输出来驱动 H 桥以控制电机旋转速度和旋转方向。

MCPWM 外设主要分为五个子模块：PWM 定时器、PWM 操作器、捕获模块、事件矩阵 (Event Task Matrix, ETM) 模块、故障检测模块。PWM 定时器提供参考时序，可以自由运行，或同步到其他定时器或外部源。PWM 操作器包含为一个 PWM 通道生成波形对的所有控制资源。捕获模块用于需要精确定时外部事件的系统。ETM 模块对输入到 MCPWM 的任务做出相应的响应动作，根据不同的运动状态产生相应的事件。故障检测模块用于捕获外部故障，使得系统可以选择做出响应。

ESP32-H2 有一个 MCPWM 外设，为 MCPWM0。

33.2 主要特性

MCPWM 外设有一个时钟分频器（预分频器）、三个 PWM 定时器、三个 PWM 操作器、一个捕获模块、一个 ETM 模块和一个故障检测模块。MCPWM 的主时钟可以从三个时钟源中任选其一，分别为：PLL_F96M_CLK, XTAL_CLK 和 RC_FAST_CLK（通过 PCR 寄存器的 PWM_CLKM_SEL 字段配置）。图 33-1 描述了 MCPWM 内的模块和接口上的信号。PWM 定时器用于生成定时参考，PWM 操作器将根据生成的定时参考生成所需的波形。通过配置，任一 PWM 操作器可以使用任一 PWM 定时器的定时参考。不同的 PWM 操作器可以使用相同的 PWM 定时器的定时参考产生 PWM 信号，也可以使用不同的 PWM 定时器的值来生成单独的 PWM 信号。不同的 PWM 定时器可以进行同步。

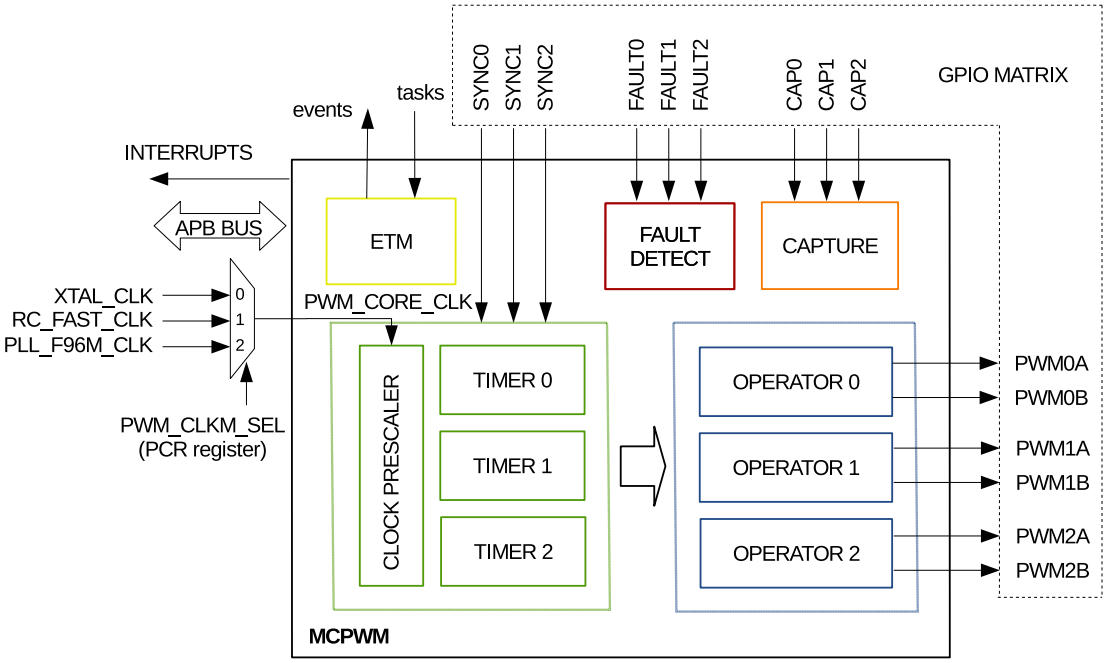


图 33-1. MCPWM 外设概览

以下是图 33-1 中模块的功能概述：

- PWM 定时器 0、1 和 2：

  - 每个 PWM 定时器都有一个专用的 8 位时钟预分频器
  - PWM 定时器中的 16 位计数器的工作模式包括：递增计数模式，递减计数模式，递增递减循环计数模式
  - 硬件/软件同步可以触发 PWM 定时器重载，重载值位于相位寄存器中；同时触发预分频的重启，从而同步定时器的时钟。硬件同步源可以来自任何 GPIO 或任何其他 PWM 定时器的 sync_out 信号。软件同步源通过向 MCPWM_TIMER*x*_SYNC_SW 位写入取反值获取。
- PWM 操作器 0、1 和 2：
  - 每个 PWM 操作器有两个 PWM 输出（PWM*x*A 和 PWM*x*B），可以在对称和非对称配置中独立工作
  - 可以通过异步方式更新对 PWM 信号的控制
  - 死区时间在上升沿和下降沿可配置，并可分别设置
  - 所有事件都可触发 CPU 中断
  - 通过高频载波信号调制 PWM 输出，在使用变压器隔离栅极驱动器时可发挥巨大作用
  - 周期、时间戳寄存器和其他主要的控制寄存器有影子寄存器，具有灵活的更新方式
- 故障检测模块：
  - 出现故障时，可选择在逐周期模式或一次性模式下处理
  - 故障条件可强制 PWM 输出高或低电平
- 捕获模块：
  - 捕获模块的时钟为 MCPWM 的主时钟
  - 旋转电机的速度测量
  - 位置传感器脉冲之间的间隔时间测量
  - 脉冲序列信号的周期和占空比测量
  - 从电流/电压传感器的占空比编码信号导出的解码电流或电压振幅
  - 3 个独立的捕获通道，各具备一个 32 位的时间戳寄存器
  - 输入捕获信号可以预分频，边沿极性可选
  - 捕获定时器可以与 PWM 定时器或外部信号同步
  - 3 个捕获通道上都可以产生中断
- ETM 模块：
  - 根据每个定时器和操作器不同的运行状态，产生不同的事件
  - 每个定时器和操作器响应其对应的任务，自动进行相应的操作
  - 每个事件和任务都可以独立使能。某个事件不使能时，不产生对应的事件；某个任务不使能时，不响应对应的任务

33.3 模块

33.3.1 概述

本节提供 MCPWM 关键模块的主要配置参数。调整特定参数，例如 PWM 定时器的同步源，请参考章节 33.3.2。

33.3.1.1 预分频器模块

图 33-2. 预分频器模块

配置参数：

- 对 PWM_CORE_CLK 进行分频。

33.3.1.2 定时器模块

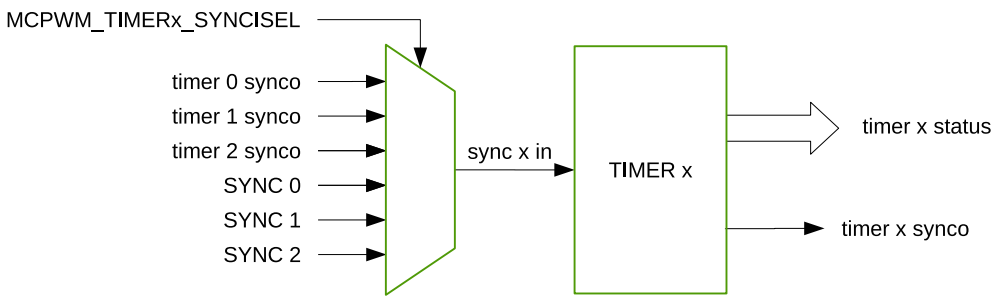


图 33-3. 定时器模块

配置参数：

- 配置 PWM 定时器的频率或周期
- 配置定时器的工作模式：
  - 递增计数模式：用于非对称 PWM 输出
  - 递减计数模式：用于非对称 PWM 输出
  - 递增递减循环计数模式：用于对称 PWM 输出
- 配置软件或硬件同步发生时的重载相位，包括值和方向
- 通过硬件或软件同步使 PWM 定时器彼此同步
- 配置 PWM 定时器的同步输入源，共 7 个可选输入源：
  - 3 个 PWM 定时器的同步输出
  - 来自 GPIO 交换矩阵的 3 个同步信号：PWMn_SYNC0_IN、PWMn_SYNC1_IN、PWMn_SYNC2_IN
  - 未选择同步输入信号

- 配置 PWM 定时器的同步输出源，共 4 个可选输出源：
  - 同步输入信号
  - PWM 定时器的值为 0 时生成的事件
  - PMW 定时器的值与时钟周期的值相同时生成的事件
  - 向 MCPWM_TIMERx_SYNC_SW 位写入取反值时生成的事件
- 配置周期更新方式

33.3.1.3 操作器模块

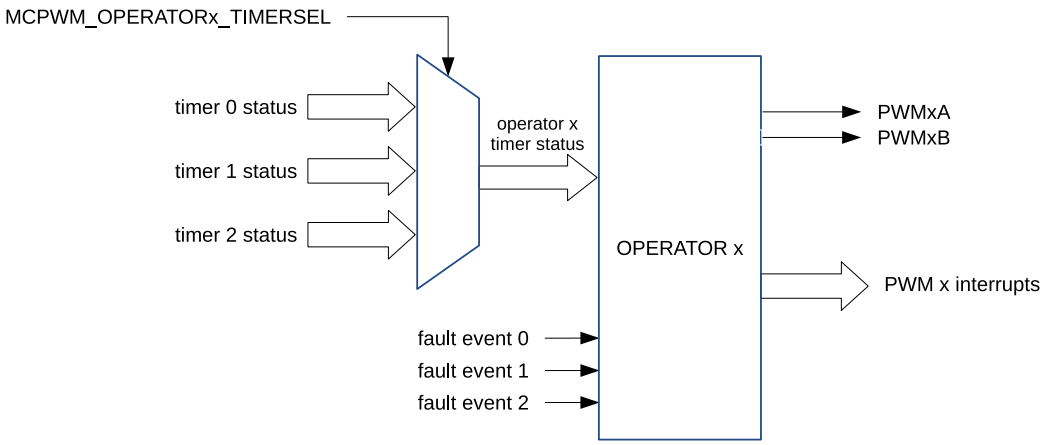


图 33-4. 操作器模块

表 33-1 列举操作器模块的主要配置参数。

表 33-1. 操作器模块的配置参数

模块	配置参数/选项
PWM 生成器	• 配置 PWMxA 和/或 PWMxB 输出的 PWM 占空比 • 配置定时事件发生的时间 • 配置发生定时事件时执行的操作： – 改变 PWMxA 和/或 PWMxB 输出为高或低 – 将 PWMxA 和/或 PWMxB 取反 – 不对输出执行任何操作 • 通过直接软件控制强制 PWM 输出的状态 • 在 PWM 输出的上升和/或下降边沿上增加死区 • 配置该模块的更新方式
死区生成器	• 控制高侧和低侧开关之间的互补死区关系 • 指定上升沿死区 • 指定下降沿死区 • 绕过死区发生器模块，PWM 波形不插入死区 • 可根据 PWMxA 输出进行 PWMxB 相移 • 配置该模块的更新方式

模块	配置参数/选项
PWM 载波	• 使能载波，设置载波频率 • 配置载波波形中第一个脉冲的持续时间 • 配置第二个以及之后的脉冲的占空比 • 绕过 PWM 载波模块，PWM 波形无变动
故障处理器	• 配置 PWM 模块是否以及如何响应故障事件信号 • 指定发生故障事件时采取的操作： – 强制 PWMxA 和/或 PWMxB 为高电平 – 强制 PWMxA 和/或 PWMxB 为低电平 – 配置 PWMxA 和/或 PWMxB 忽略任何故障事件 • 配置 PWM 应对故障事件的间隔模式： – 一次性模式 – 逐周期模式 • 生成中断 • 绕过故障处理器模块 • 配置逐周期操作清除的方式 • 当时基计数器采取向上和向下时，可采取不同操作

33.3.1.4 故障检测模块

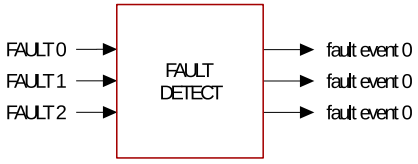


图 33-5. 故障检测模块

配置参数：

- 启用生成故障事件，并为每个故障信号配置故障事件生成的极性
- 中断生成故障事件

33.3.1.5 捕获模块

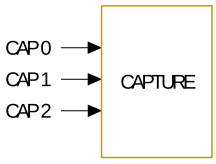


图 33-6. 捕获模块

配置参数：

- 选择捕获模块输入的边沿极性和预分频
- 配置软件触发捕获
- 配置捕获定时器同步触发和同步相位
- 软件同步捕获定时器

33.3.1.6 ETM 模块

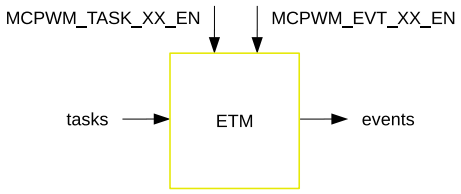


图 33-7. ETM 模块

配置参数：

- 独立配置每个事件和任务的使能。某个事件不使能时，不产生对应的事件；某个任务不使能时，不响应对应的任务。

33.3.2 PWM 定时器模块

MCPWM 外设有三个 PWM 定时器模块。它们中的任何一个都可以决定三个 PWM 操作器模块中任意一个的必要事件时序。通过使用 GPIO 交换矩阵的同步信号，内置同步逻辑允许该 MCPWM 外设中的多个 PWM 定时器模块作为一个系统协同工作。

33.3.2.1 PWM 定时器模块的配置

用户可配置 PWM 定时器模块的以下功能：

- 通过指定 PWM 定时器频率或周期来控制事件发生的频率
- 配置某个 PWM 定时器与其他 PWM 定时器或模块同步
- 使 PWM 定时器与其他 PWM 定时器或模块同相
- 配置定时器计数模式：递增，递减，或递增递减循环计数模式
- 使用预分频器更改 PWM 定时器时钟 (PT_CLK) 的速率。每个定时器都有自己的预分频器，通过寄存器 MCPWM_TIMER0_CFG0_REG 的 MCPWM_TIMER*x*_PRESCALE 配置。PWM 定时器根据该寄存器的设置以较慢的速度递增或递减。当定时器停止后并再次开始计数时，新的MCPWM_TIMER*x*_PRESCALE 配置值才会生效。

33.3.2.2 PWM 定时器工作模式和定时事件生成

PWM 定时器有三种工作模式，由 MCPWM_TIMER*x*_MOD (*x* = 0, 1, 2) 定时器模式字段配置：

- 递增计数模式：

定时器从零增加到周期字段中配置的值。达到周期值后，PWM 定时器清零，并再次开始递增。PWM 周期 = 周期寄存器中的周期值 + 1。

说明：周期字段为 MCPWM_TIMER*x*_PERIOD (*x* = 0, 1, 2)，对应 MCPWM_TIMER0_PERIOD、MCPWM_TIMER1_PERIOD、MCPWM_TIMER2_PERIOD。

- 递减计数模式：
  PWM 定时器从周期寄存器中的值开始递减到零。达到零后，将恢复为周期值，并再次开始递减。在这种情况下，PWM 周期 = 周期寄存器中的周期值 + 1。
- 递增-递减循环模式：
  此模式结合了上述两种模式。PWM 定时器从零开始递增，直到达到周期值，再次递减为零。PWM 定时器按照此模式循环递增递减。PWM 周期 = 周期寄存器的周期值 × 2。

图 33-8 至 33-11 显示不同模式下的 PWM 定时器波形，包括同步事件期间的定时器行为。递增计数模式中，同步后永远保持递增计数；递减计数模式中，同步后永远保持递减计数；递增-递减循环模式中，同步后由 MCPWM_TIMER*x*_PHASE_DIRECTION 配置计数方向。

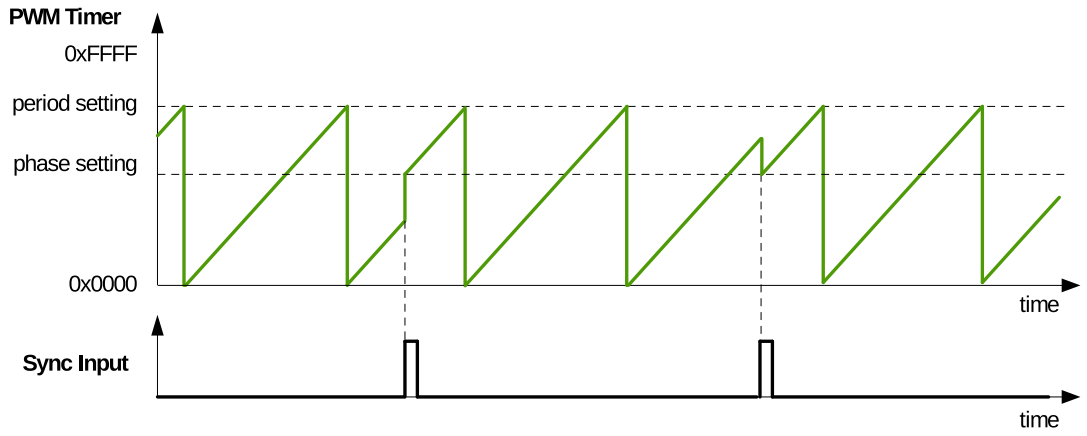


图 33-8. 递增计数模式波形

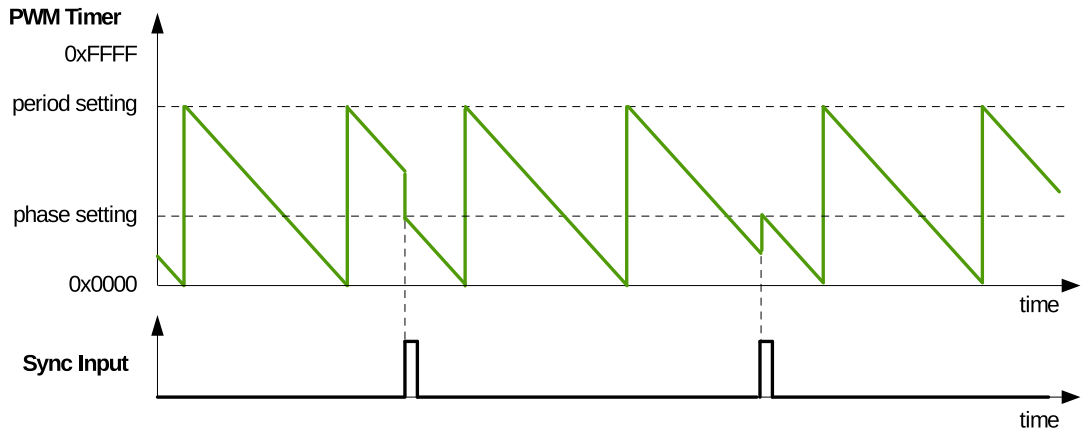


图 33-9. 递减计数模式波形

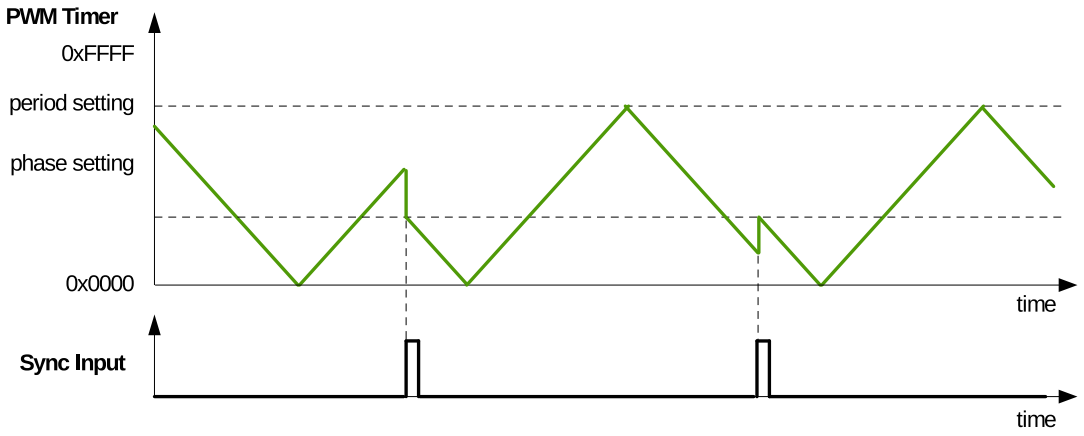


图 33-10. 递增递减循环模式波形，同步事件后递减

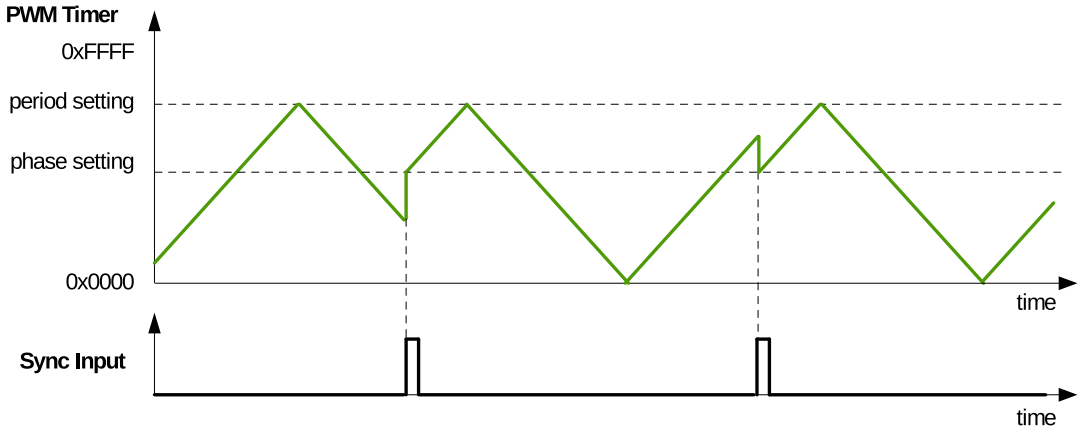


图 33-11. 递增递减循环模式波形，同步事件后递增

PWM 定时器运行时，定期自动生成以下定时事件：

- UTEP：当 PWM 定时器等于周期字段的值 (MCPWM_TIMER*x*_PERIOD) 且 PWM 定时器递增计数时，生成的定时事件
- UTEZ：当 PWM 定时器等于零且 PWM 定时器递增计数时，生成的定时事件
- DTEP：当 PWM 定时器等于周期字段的值 (MCPWM_TIMER*x*_PERIOD) 且 PWM 定时器递减时，生成的定时事件
- DTEZ：当 PWM 定时器等于零且 PWM 定时器递减时，生成的定时事件

图 33-12 至 33-14 为 U/DTEP 和 U/DTEZ 定时事件的时序波形。

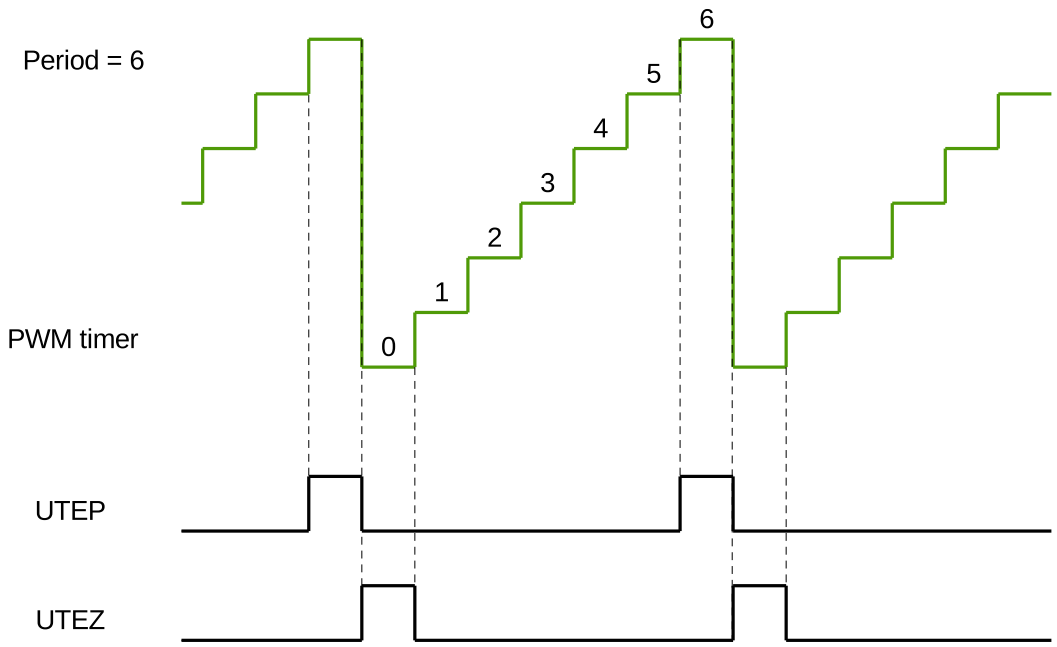


图 33-12. 递增模式中生成的 UTEP 和 UTEZ

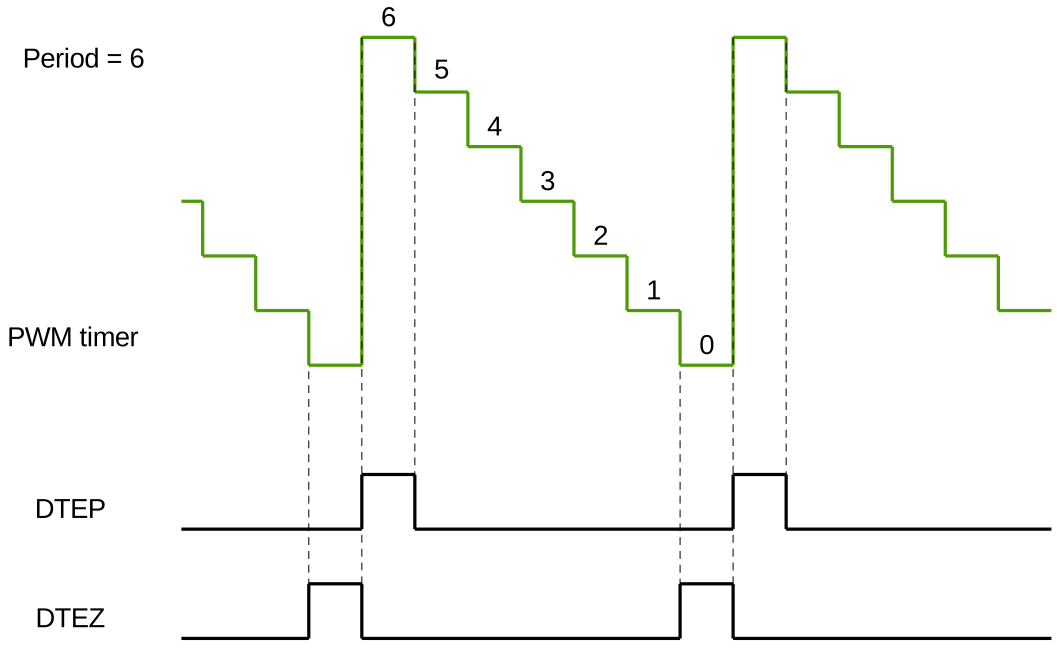


图 33-13. 递减模式中生成的 UTEP 和 UTEZ

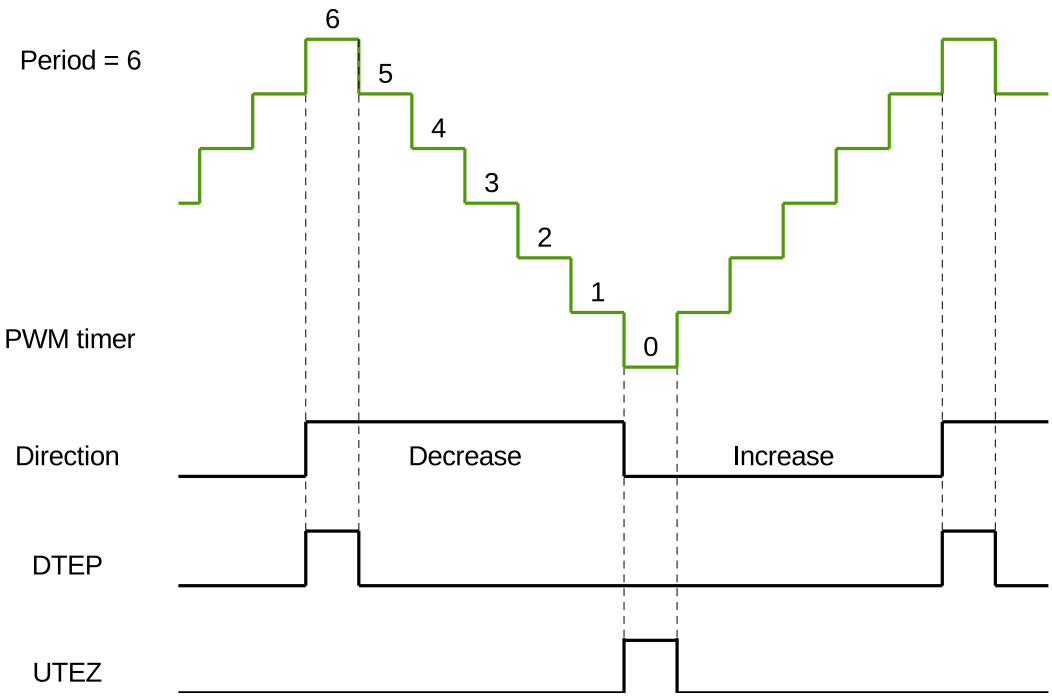


图 33-14. 递增递减模式中生成的 UTEP 和 UTEZ

33.3.2.3 PWM 定时器影子寄存器

PWM 定时器周期寄存器和 PWM 定时器时钟预分频器寄存器配有影子寄存器。影子寄存器能够备份即将写入有效寄存器的值，并在硬件同步的特定时刻写入有效寄存器。有效寄存器和影子寄存器的功能如下：

- 有效寄存器：直接控制硬件执行的所有操作。
- 影子寄存器：作为临时缓冲区，存储即将写入有效寄存器的值。在用户配置的某个时间点，影子寄存器中的值被写入有效寄存器。在此之前，影子寄存器的内容对受控硬件没有任何直接影响。这有助于防止寄存器由软件异步修改时可能发生的错误硬件操作。影子寄存器和有效寄存器具有相同的存储器地址。软件写入或读取影子寄存器。

  当定时器开始工作时，时钟预分频器的有效寄存器更新。当 MCPWM_GLOBAL_UP_EN 置 1 时，可通过以下方式选择更新有效寄存器的时间点：

  - 将 MCPWM_TIMER*x*_PERIOD_UPMETHOD 置 0，立即更新
  - 将 MCPWM_TIMER*x*_PERIOD_UPMETHOD 置 1，当 PWM 定时器值为 0 时，开始更新
  - 将 MCPWM_TIMER*x*_PERIOD_UPMETHOD 置 2，当 PWM 定时器同步时，开始更新
  - 将 MCPWM_TIMER*x*_PERIOD_UPMETHOD 置 3，当 PWM 定时器值为 0 时或同步时，开始更新
  - 软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新

33.3.2.4 PWM 定时器同步和锁相

PWM 模块采用灵活的同步方法。每个 PWM 定时器都有一个同步输入和一个同步输出。同步输入可以从 GPIO 矩阵的三个同步输出和三个同步信号中选择。同步输出可以在 PWM 定时器等于周期、PWM 定时器等于零或软件同步时，根据同步输入信号生成。因此，PWM 定时器可以通过相位锁定而相连。在同步期间，PWM 定时器时钟预分频器将复位其计数器，以同步 PWM 定时器时钟。

33.3.3 PWM 操作器模块

PWM 操作器模块具备以下功能：

- 根据相应 PWM 定时器的定时参考生成 PWM 信号对
- PWM 信号对的每个信号都可以独立设置特定的死区时间
- 可通过配置将载波叠加到 PWM 信号上
- 故障条件下处理响应

图 33-15 为 PWM 操作器的框图。

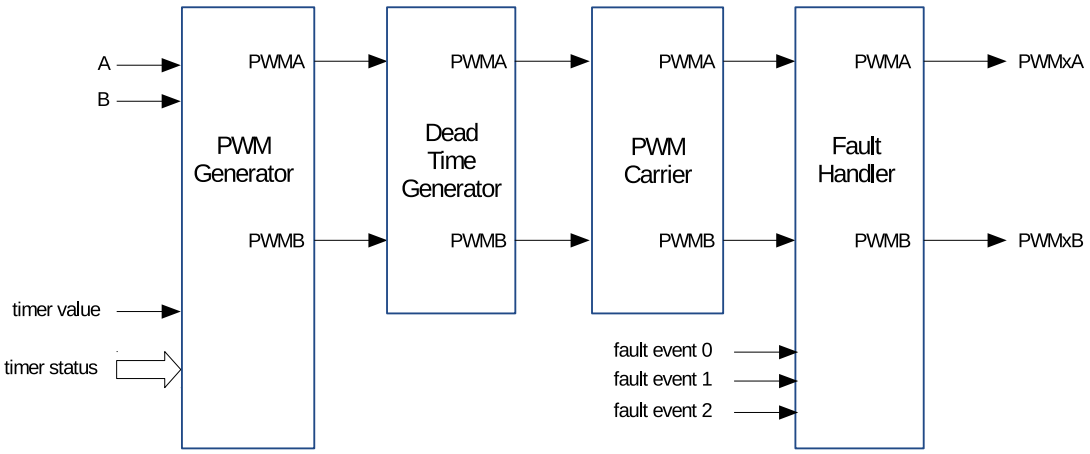


图 33-15. PWM 操作器的子模块

33.3.3.1 PWM 生成器模块

PWM 生成器模块的作用

此模块可以生成或导入重要的时序事件，转化为特定操作后，在 PWM*x*A 和 PWM*x*B 输出处生成所需的波形。

PWM 生成器模块执行以下操作：

- 基于使用寄存器 A 和 B 配置的时间戳生成定时事件。满足以下条件时发生定时事件：
  - UTEA：PWM 定时器递增计数并且其值等于寄存器 A
  - UTEB：PWM 定时器递增计数并且其值等于寄存器 B
  - DTEA：PWM 定时器递减计数并且其值等于寄存器 A

  - DTEB：PWM 定时器递减计数并且其值等于寄存器 B

- 基于故障或同步事件生成 U/DT1、U/DT2 定时事件
  - UT0：定时器递增计数并且检测到 FAULT0 事件（字段 MCPWM_GENx_T0_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENx_T0_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENx_T0_SEL 置 2）或同步（字段 MCPWM_GENx_T0_SEL 置 3）。
  - UT1：定时器递增计数并且检测到 FAULT0 事件（字段 MCPWM_GENx_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENx_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENx_T1_SEL 置 2）或同步（字段 MCPWM_GENx_T1_SEL 置 3）。
  - DT0：定时器递减计数并且检测到 FAULT0 事件（字段 MCPWM_GENx_T0_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENx_T0_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENx_T0_SEL 置 2）或同步（字段 MCPWM_GENx_T0_SEL 置 3）。
  - DT1：定时器递减计数并且检测到 FAULT0 事件（字段 MCPWM_GENx_T1_SEL 置 0）或检测到 FAULT1 事件（字段 MCPWM_GENx_T1_SEL 置 1）或检测到 FAULT2 事件（字段 MCPWM_GENx_T1_SEL 置 2）或同步（字段 MCPWM_GENx_T1_SEL 置 3）。
- 当这些定时事件同时发生时管理优先级
- 基于定时事件产生置 1、置 0 和取反操作
- 根据 PWM 生成器模块的配置来控制 PWM 占空比
- 使用影子寄存器处理新的时间戳值，以防止 PWM 波形中的毛刺干扰

PWM 操作器影子寄存器

时间戳寄存器 A 和 B，以及操作配置寄存器 MCPWM_GENx_A_REG 和 MCPWM_GENx_B_REG 都有影子寄存器。影子寄存器提供了一种与硬件同步更新寄存器的方法。

当 MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。MCPWM_GENx_A_REG 和 MCPWM_GENx_B_REG 的更新方式字段为 MCPWM_GENx_CFG_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。更多关于影子寄存器的描述，请参考章节 33.3.2.3。

定时事件

表 33-2 概括了所有定时信号和事件。

表 33-2. PWM 生成器中的所有定时事件

信号	事件描述	PWM 定时器操作
DTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递减计数
DTEZ	PWM 定时器的值等于 0	
DTEA	PWM 定时器的值等于寄存器 A	
DTEB	PWM 定时器的值等于寄存器 B	
DT0 事件	基于故障或同步事件	
DT1 事件	基于故障或同步事件	
UTEP	PWM 定时器的值等于周期寄存器的值	PWM 定时器递增计数
UTEZ	定时器的值等于 0	
UTEA	PWM 定时器的值等于寄存器 A	

信号	事件描述	PWM 定时器操作
UTEB	PWM 定时器的值等于寄存器 B	
UT0 事件	基于故障或同步事件	
UT1 事件	基于故障或同步事件	
软件强制事件	软件触发的异步事件	-

软件强制事件用于在 PWM*x*A 和 PWM*x*B 的输出上施加非连续或连续的强制电平。此更改是异步完成的。软件强制由寄存器 MCPWM_GEN*x*_FORCE_REG 控制。

PWM 生成器模块中 T0/T1 的选择和配置独立于故障检测模块中的故障事件的配置。跳闸事件可以不被配置为在故障检测模块中引起跳闸动作，但相同的事件可以由 PWM 生成器用于触发 T0/T1 以控制 PWM 波形。

需要注意的是，当 PWM 定时器处于递增递减循环计数模式时，它将在 TEP 事件后递减，在 TEZ 事件后递增。因此，在此模式下，将出现 DTEP 和 UTEZ，但不会出现 UTEP 和 DTEZ。

PWM 生成器可以同时处理多个事件。事件优先级由硬件决定，详见表 33-3 和表 33-4。优先级从 1（最高）到 7（最低）排列。需要注意的是，TEP 和 TEZ 事件的优先级取决于 PWM 定时器的计数模式。

如果 A 或 B 的值设置为大于周期，则 U/DTEA 和 U/DTEB 将永远不会发生。

表 33-3. PWM 定时器递增计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	UTEP
3	UT0
4	UT1
5	UTEB
6	UTEA
7（最低）	UTEZ

表 33-4. PWM 定时器递减计数时，定时事件的优先级

优先级	事件
1（最高）	软件强制事件
2	DTEZ
3	DT0
4	DT1
5	DTEB
6	DTEA
7（最低）	DTEP

说明：

1. UTEP 和 UTEZ 不同时发生。当 PWM 定时器处于递增计数模式，UTEP 将始终比 UTEZ 提前一个周期发生，如图 33-12 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，UTEP 不会发生。
2. DTEP 和 DTEZ 不同时发生。当 PWM 定时器处于递减计数模式时，DTEZ 始终比 DTEP 早一个周期发生，

如图 33-13 所示。因此，它们对 PWM 信号的作用不会彼此干扰。当 PWM 定时器处于递增递减循环模式时，DTEZ 不会发生。

PWM 信号生成

当某个定时事件发生时，PWM 生成器控制输出 PWM*x*A 和 PWM*x*B 的电平。定时事件通过 PWM 定时器计数模式（递增或递减）进一步限定。根据定时器计数方向，模块可以对 PWM 定时器递增或递减计数的阶段执行不同的操作。

可以在 PWM*x*A 和 PWM*x*B 输出上配置以下操作：

- 置为高电平：将 PWM*x*A 或 PWM*x*B 的输出配置为高电平。
- 置为低电平：通过将 PWM*x*A 或 PWM*x*B 的输出配置为低电平，来清除 PWM*x*A 或 PWM*x*B 的输出。
- 取反：将 PWM*x*A 或 PWM*x*B 的当前输出电平配置为相反的值。如果它当前被拉高，则拉低，或反之。
- 不进行操作：保持 PWM*x*A 和 PWM*x*B 输出电平不变。在这种状态下，仍然可以触发中断。

输出上的操作通过寄存器 MCPWM_GEN*x*_A_REG 和 MCPWM_GEN*x*_B_REG 配置。每一次输出的操作都独立配置。此外，根据事件在某个输出上，灵活地执行不同的操作。表 33-2 中列举的任何事件都可以作用于 PWM*x*A 或 PWM*x*B 输出上。关于生成器 0，1 或 2 的寄存器信息，请参考第 33.4 章。

常见配置的波形

图 33-16 为 PWM 定时器在递增递减循环计数时生成的对称 PWM 波形。该模式下的直流 0%-100% 调制可由以下公式获得：

$$Duty = (Period - A) \div Period$$

如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递增，则 PWM 输出被拉高。如果 A 的值等于 PWM 定时器的值，并且 PWM 定时器递减，则 PWM 输出被拉低。

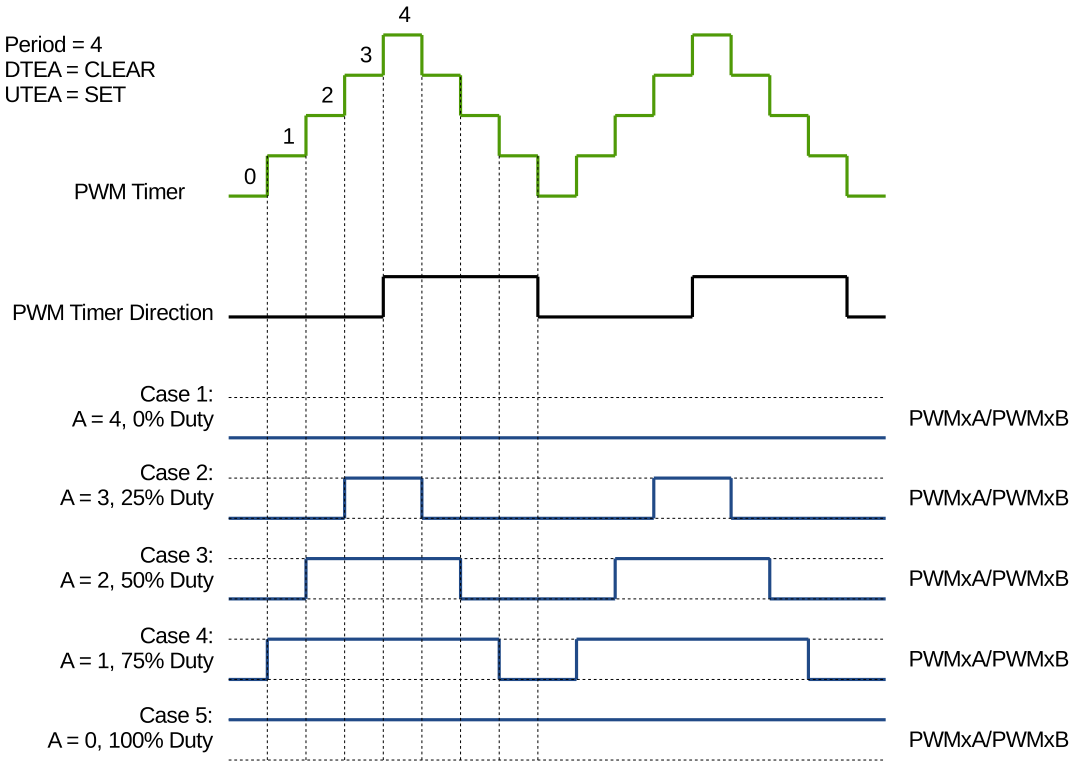


图 33-16. 递增递减模式下的对称波形

图 33-17 至图 33-20 的 PWM 波形描述了常见的 PWM 操作器配置。图中数据说明如下：

- Period A 和 B 分别表示写入周期寄存器 A 和 B 的值。
- PWM*x*A 和 PWM*x*B 是 PWM 操作器 *x* 的输出信号。

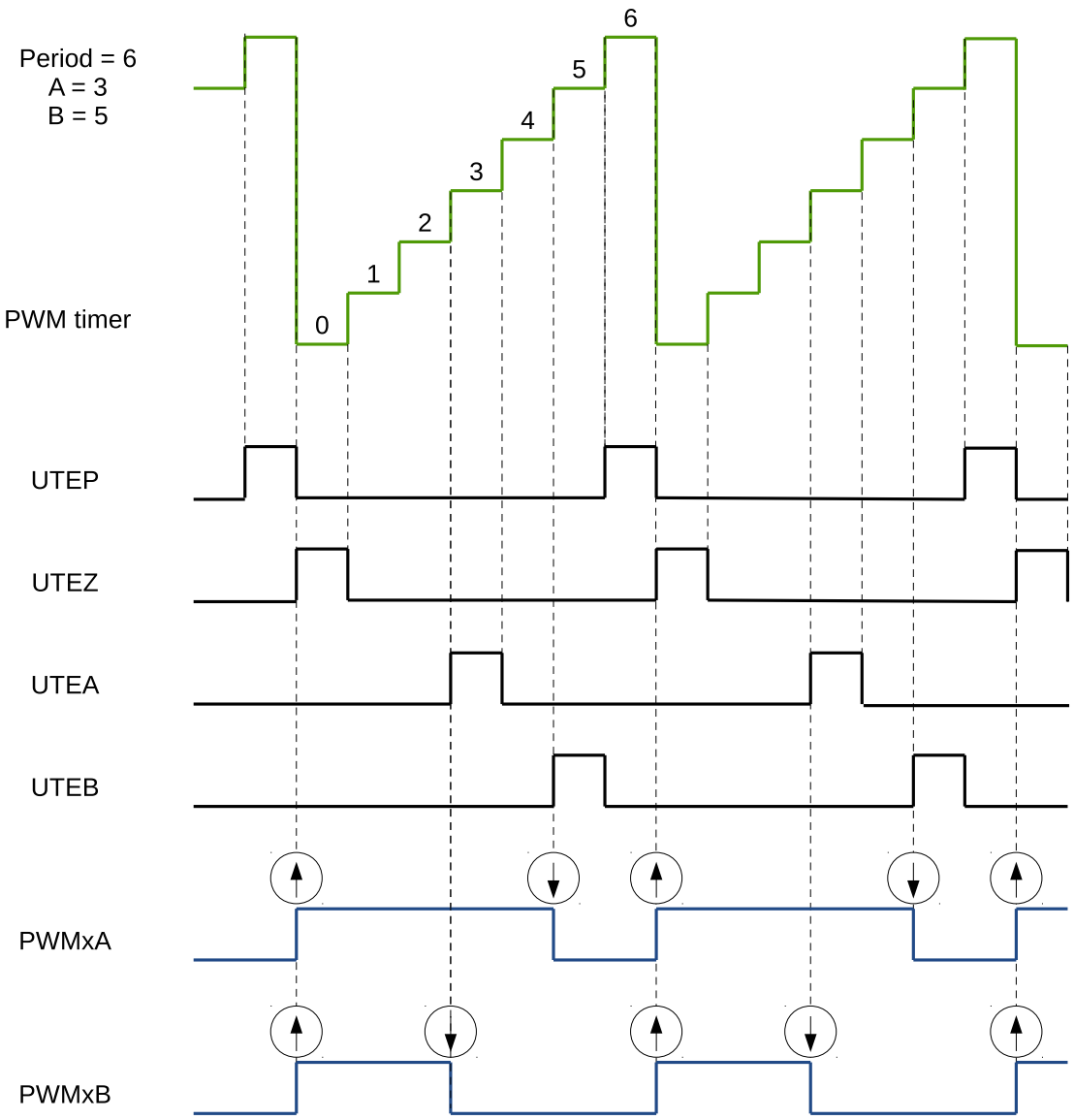


图 33-17. 递增计数模式，单边不对称波形，PWM*x*A 和 PWM*x*B 独立调制–高电平

PWM*x*A 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWM*x*B 的占空比调制由 A 设置，高电平有效，与 A 成正比。

$$Period = (MCPWM\_TIMERx\_PERIOD + 1) \times T_{PT\_CLK}$$

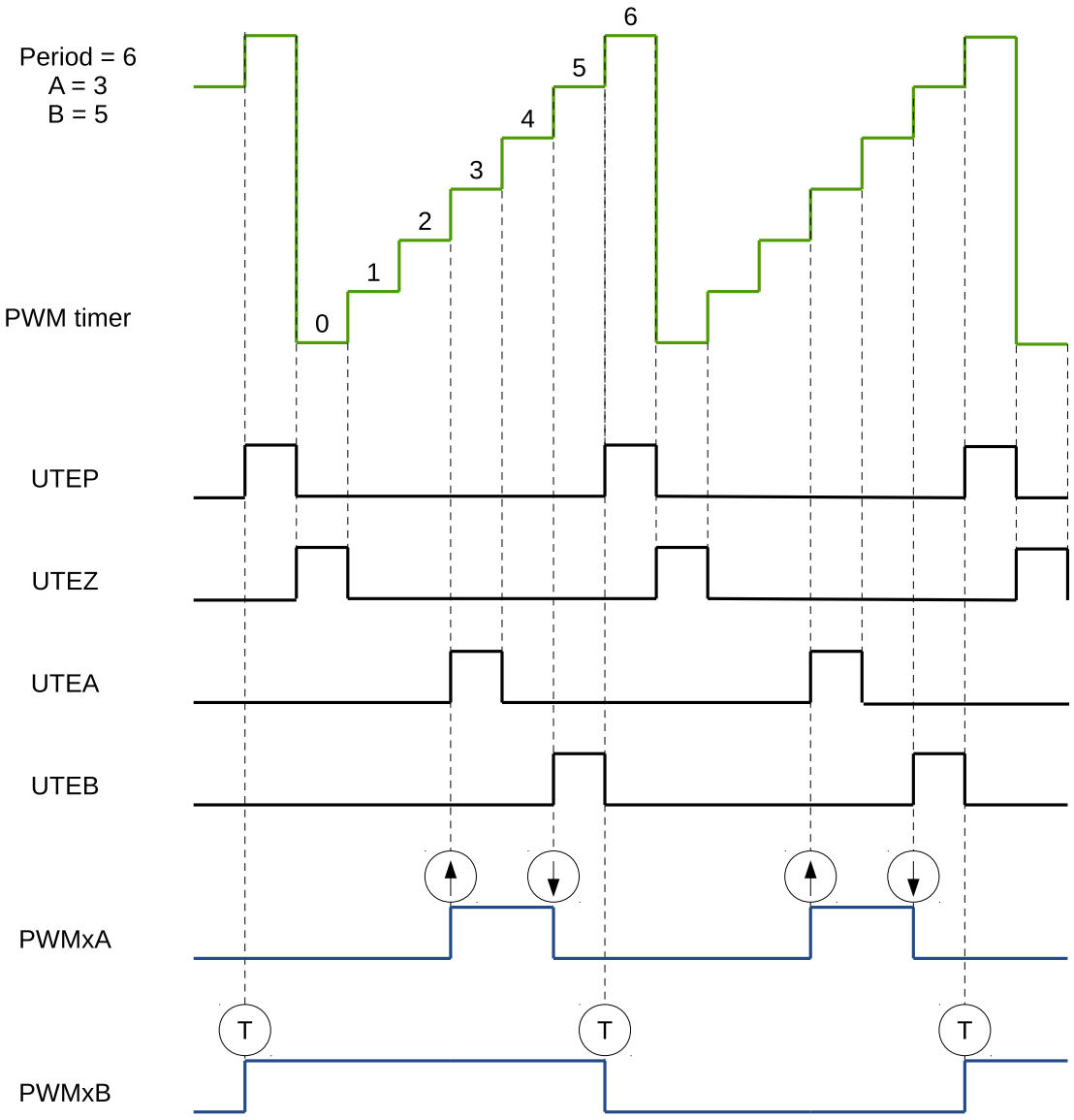


图 33-18. 递增计数模式，脉冲位置不对称波形，PWMxA 独立调制

脉冲可以在 PWM 波形内（零至周期值之间）的任何地方生成。
PWMxA 占空比与 (B – A) 成正比。

$$Period = (MCPWM\_TIMERx\_PERIOD + 1) \times T_{PT\_CLK}$$

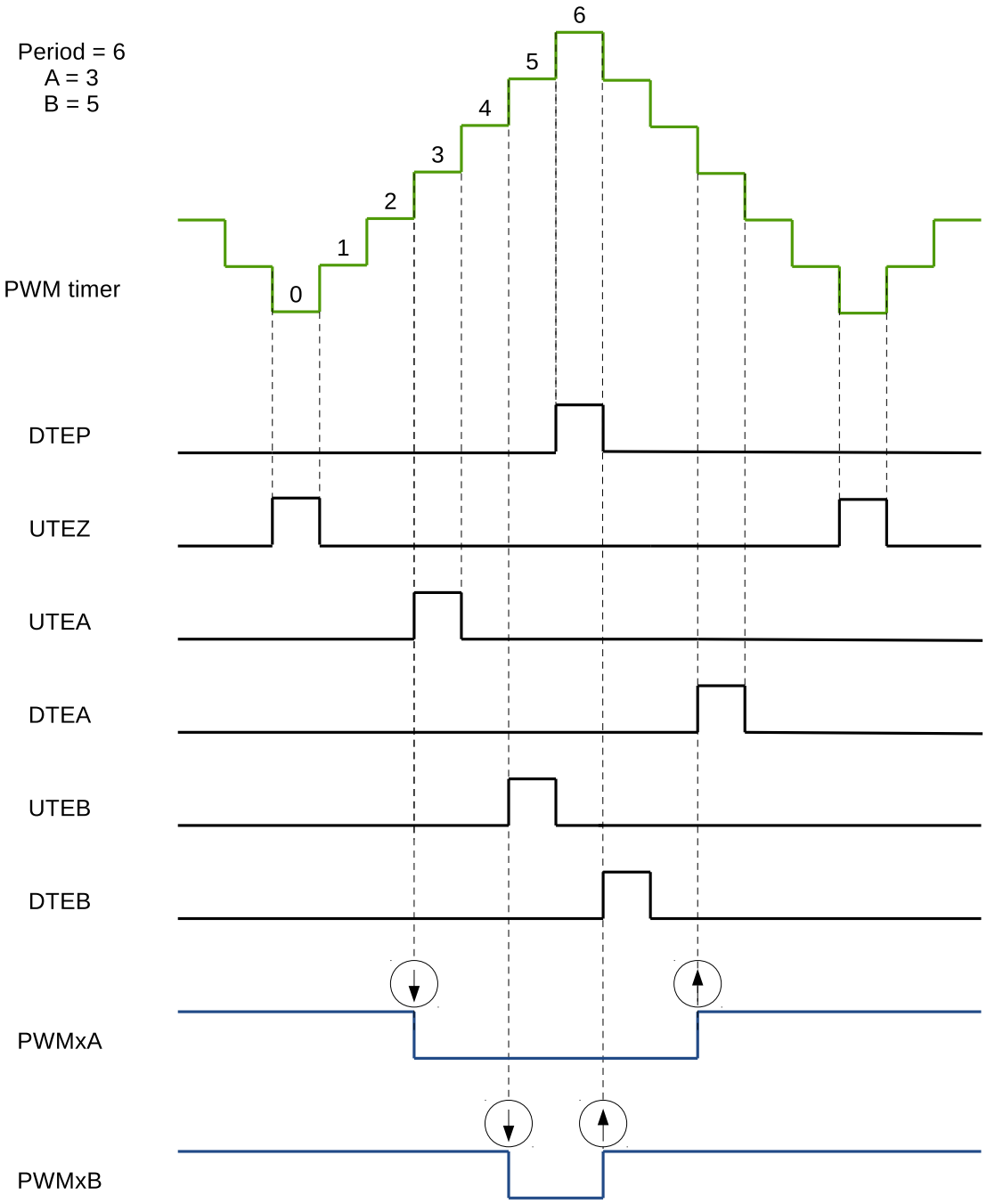


图 33-19. 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–高电平有效

PWM*x*A 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWM*x*B 的占空比调制由 B 设置，高电平有效，与 B 成正比。
输出 PWM*x*A 和 PWM*x*B 可驱动不同开关。

$$Period = (2 \times MCPWM\_TIMERx\_PERIOD) \times T_{PT\_CLK}$$

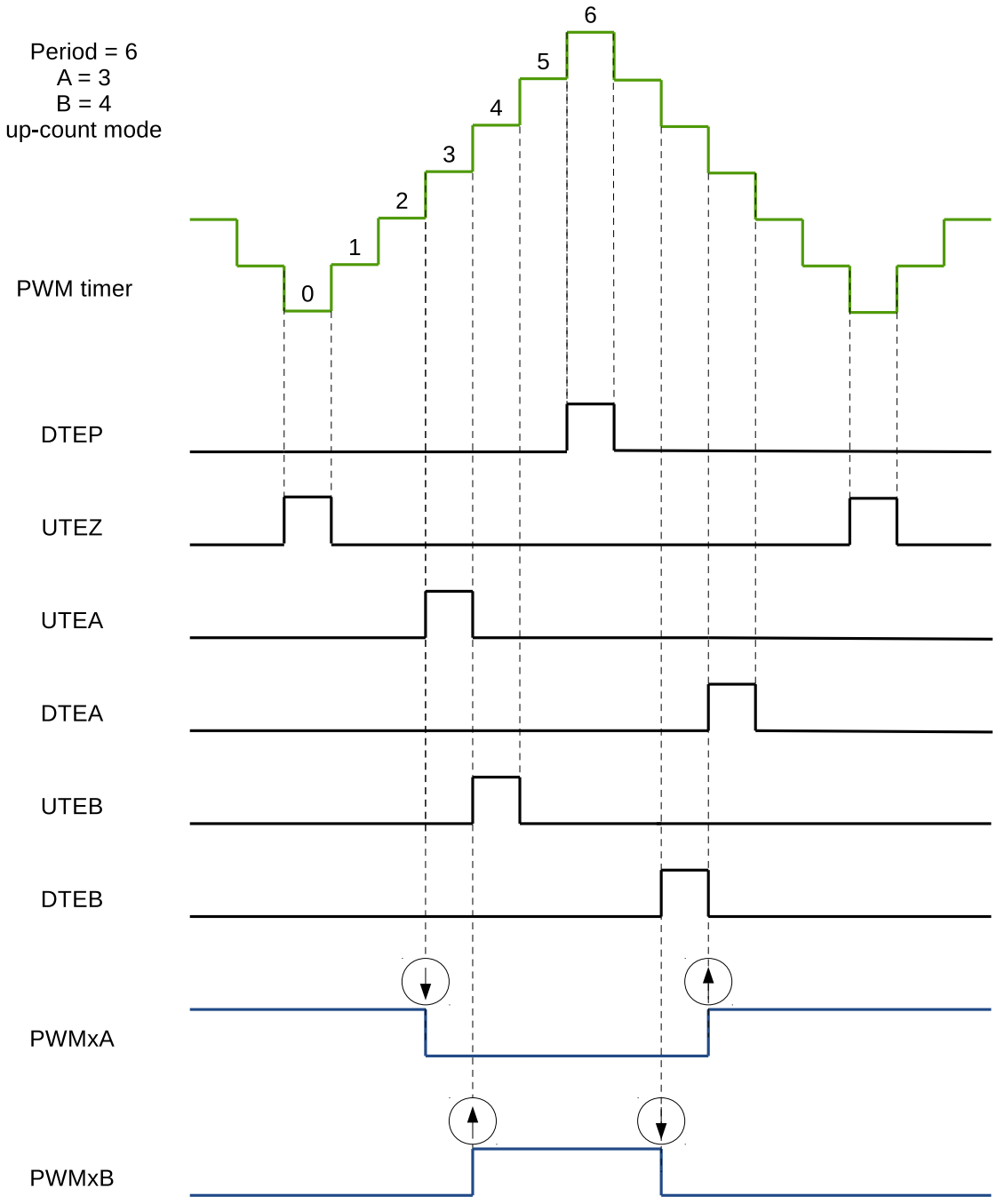


图 33-20. 递增递减循环计数模式，双沿对称波形，在 PWM*x*A 和 PWM*x*B 上独立调制–互补

PWM*x*A 的占空比调制由 A 设置，高电平有效，与 A 成正比。
PWM*x*B 的占空比调制由 B 设置，高电平有效，与 B 成正比。
PWM*x*A/B 输出可驱动上/下（互补）开关。
死区 = B – A，边沿位置完全可由软件配置。必要时，可使用死区生成器模块设置其他边沿延迟方式。

$$Period = (2 \times MCPWM\_TIMERx\_PERIOD) \times T_{PT\_CLK}$$

图 33-21 为当 UT0/1 和 DT0/1 事件产生时的波形。在本例中，T0 选择 FAULT0 ，T1 选择 FAULT1。T0 和 T1 所选择的事件可以独立配置，这些事件可以是 FAULT0，FAULT1，FAULT2 或同步事件。具体配置方法

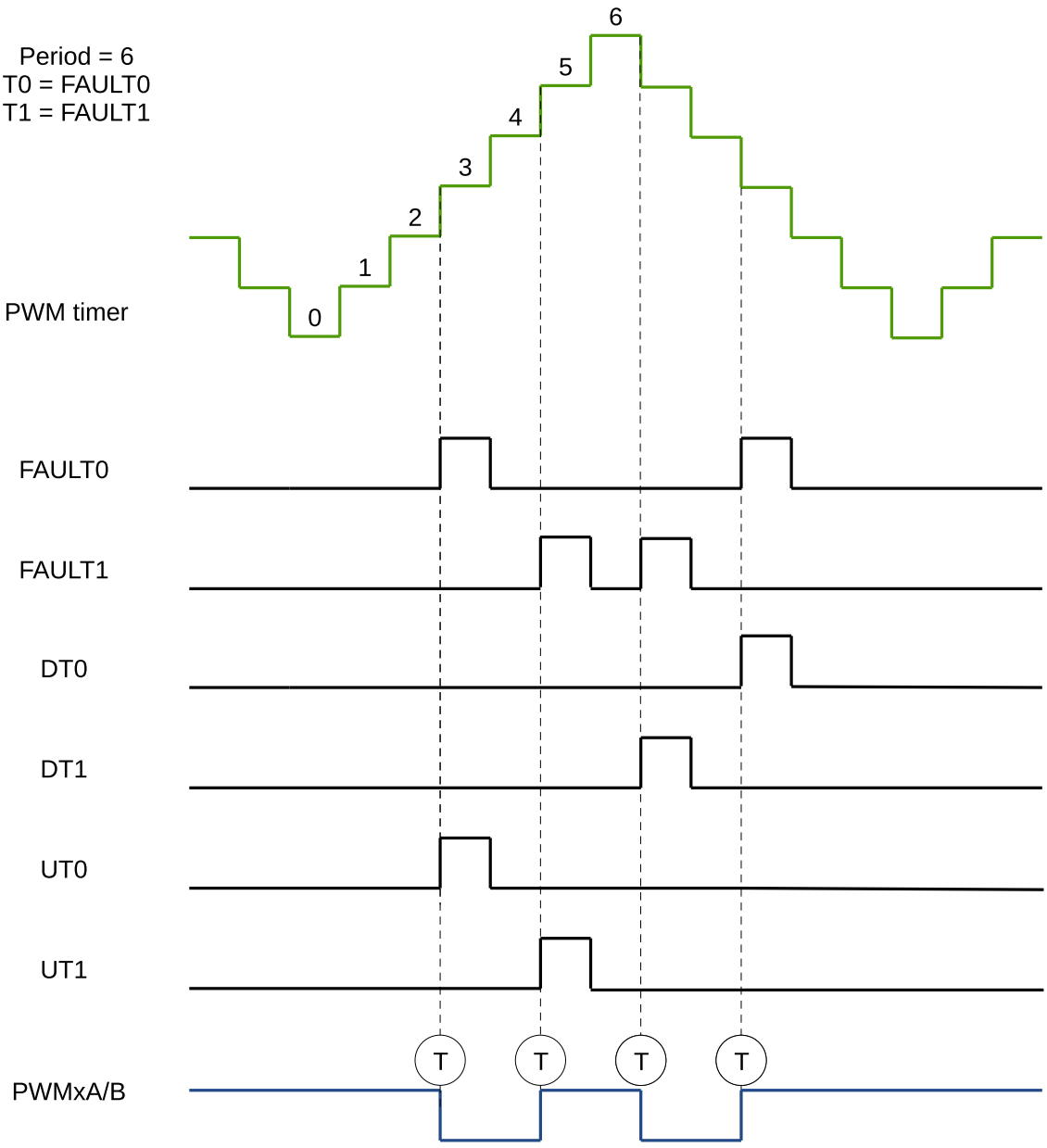


图 33-21. 递增递减循环计数模式，错误或同步事件，在 PWM*x*A 和 PWM*x*B 上相同调制

见33.3.3.1。

软件强制事件

在 PWM 生成器内有 2 种软件强制事件：

- 非连续即时 (NCI) 软件强制事件：当由软件触发时，这类事件在 PWM 输出上立即生效。强制是不连续的，意味着下一个激活的定时事件能够改变 PWM 输出。
- 连续 (CNTU) 软件强制事件：这类事件是连续的。强制 PWM 将持续输出，直到通过软件释放。事件触发器可配置。这类事件可配置为定时或即时发生。

图 33-22 为 NCI 软件强制事件的一种波形。NCI 用于单独强制 PWM*x*A 输出为低电平，PWM*x*B 不受强制。

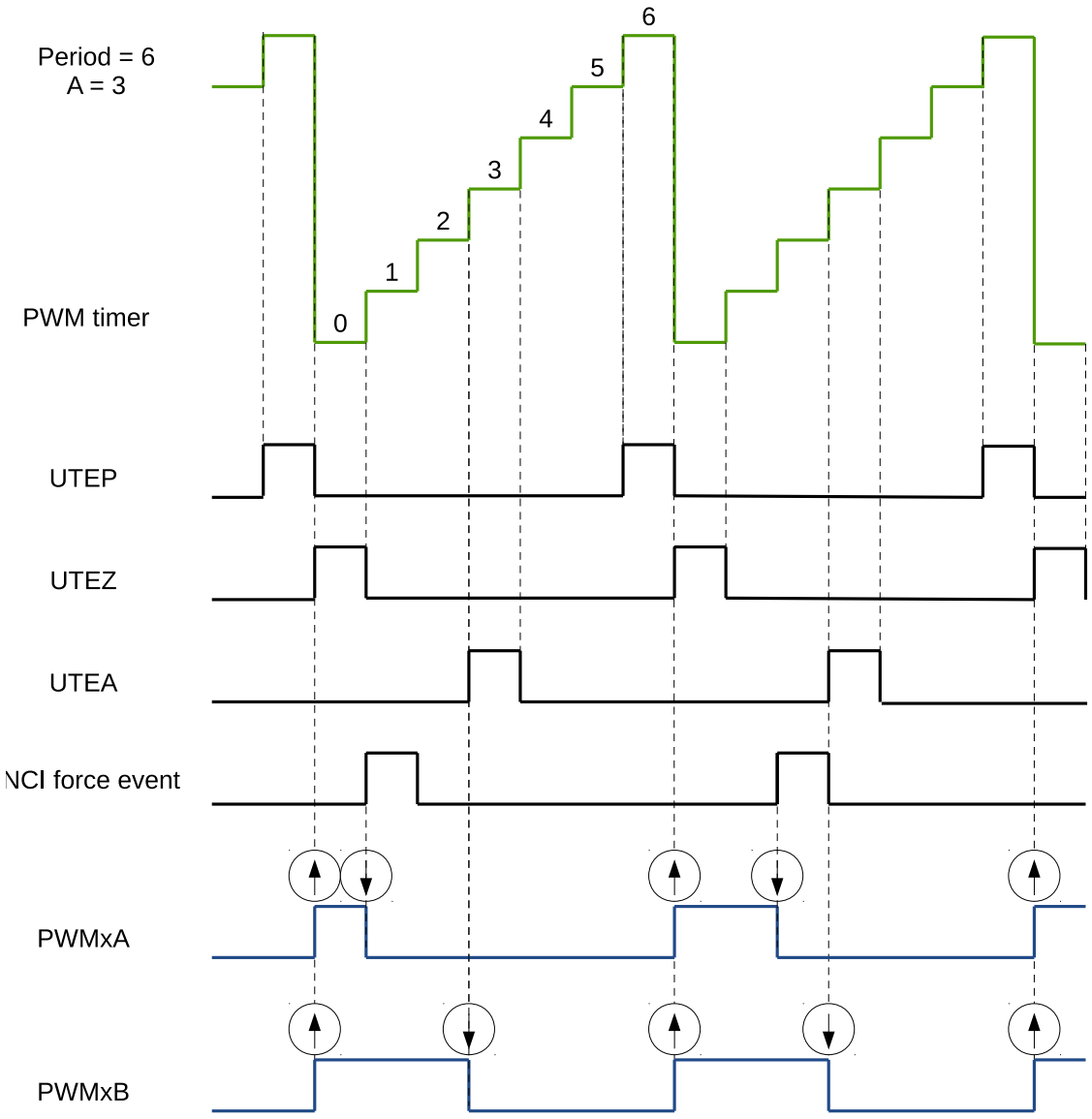


图 33-22. NCI 在 PWMxA 输出上软件强制事件示例

图 33-23 为 CNTU 软件强制事件的波形。UTEZ 事件被选为 CNTU 软件强制事件的触发器。CNTU 用于单独强制 PWMxB 输出为低电平，但 PWMxA 不受强制。

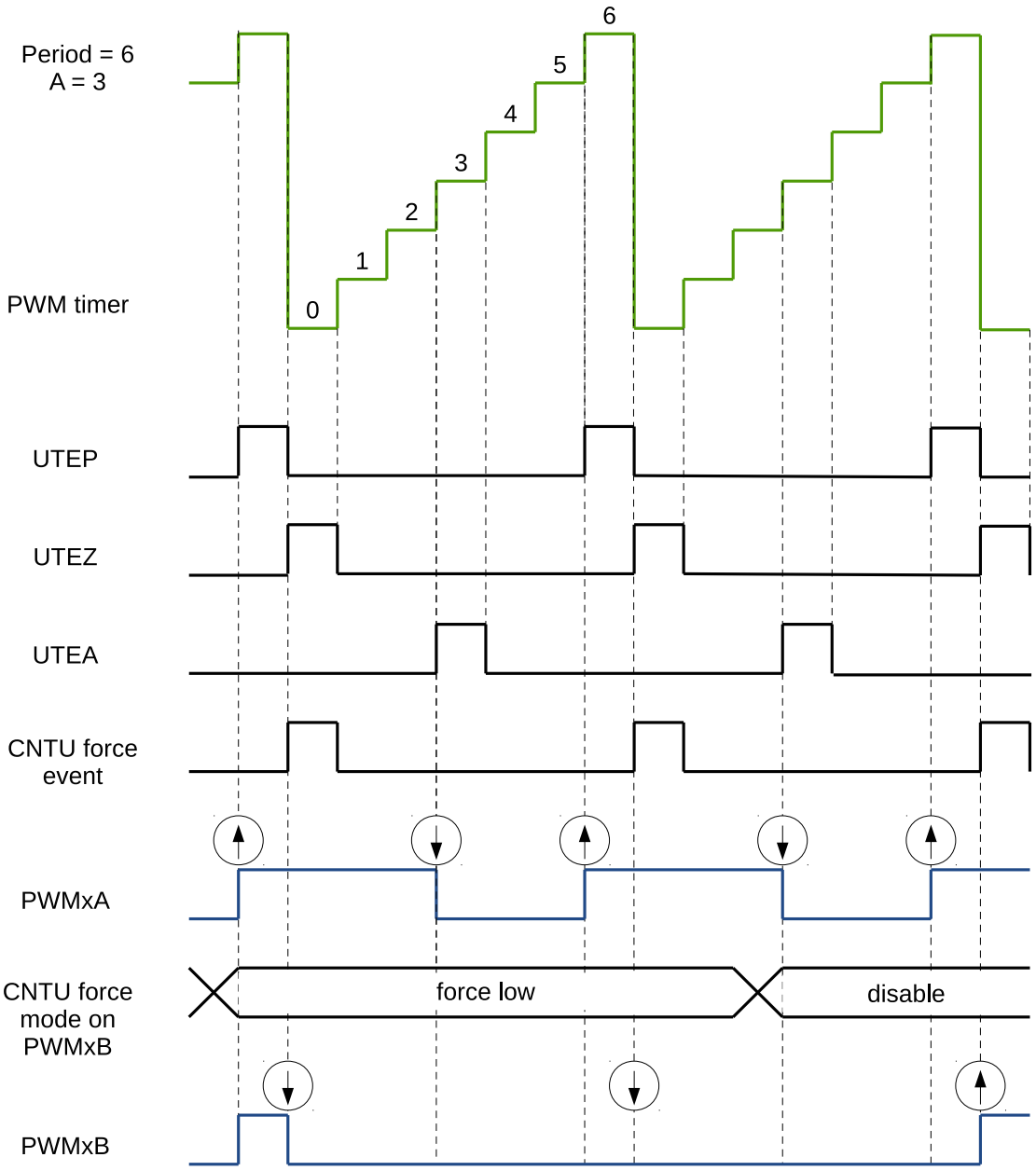


图 33-23. CNTU 在 PWMxB 输出上软件强制事件示例

33.3.3.2 死区生成器模块

死区生成器模块作用

章节 33.3.3.1 介绍了在 PWM*x*A 和 PWM*x*B 输出上生成信号的几种方式/事件，包括信号边沿的特定位置。通过改变信号之间的边沿位置以及设置信号的占空比，可获得所需的死区。此外，也可以使用专门的死区生成器模块控制死区。

死区生成器模块的主要功能如下：

- 根据单个 PWM*x*A 输入的死区生成信号对（PWM*x*A 和 PWM*x*B）
- 通过在信号边沿增加延迟来生成死区：
  - 上升沿延迟 (RED)
  - 下降沿延迟 (FED)
- 配置信号对为：
  - 高电平有效互补 (AHC)
  - 低电平有效互补 (ALC)
  - 高电平有效 (AH)
  - 低电平有效 (AL)
- 如果死区已经在生成器中配置，死区发生器也可以不进行配置

死区模块生成器影子寄存器

延迟寄存器 RED 和 FED 的影子寄存器为 MCPWM_DT*x*_RED_CFG_REG 和 MCPWM_DT*x*_FED_CFG_REG。
MCPWM_GLOBAL_UP_EN 置 1 时，影子寄存器中的值可在某个特定时间写入有效寄存器中。
MCPWM_DT*x*_RED_CFG_REG 的更新方式寄存器为 MCPWM_DT*x*_RED_UPMETHOD；
MCPWM_DT*x*_FED_CFG_REG 的更新方式寄存器为 MCPWM_DT*x*_FED_UPMETHOD。软件也可以触发全局强制更新位 MCPWM_GLOBAL_FORCE_UP，该位将触发模块中的所有寄存器根据影子寄存器进行更新。寄存器描述详见 33.3.2.3。

死区生成器模块的操作要点

图 33-24 描述了创建死区模块的开关拓扑。

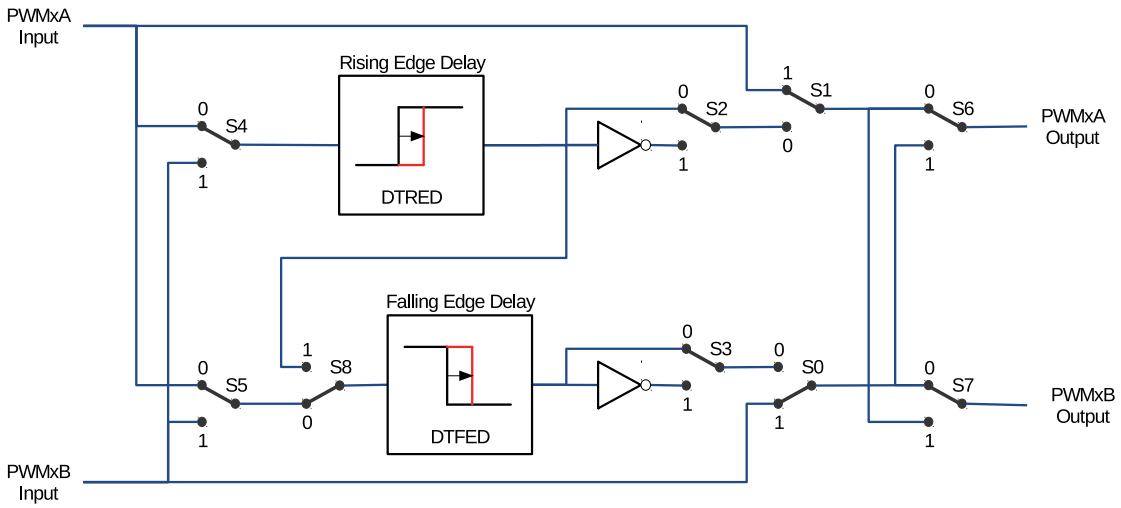


图 33-24. 死区模块的开关拓扑

上图中的 S0 - S8 开关由 MCPWM_DT*x*_CFG_REG 寄存器中的字段控制，详细信息见表 33-5。

表 33-5. 控制死区时间生成器开关的字段

开关	字段
S0	MCPWM_DT*x*_B_OUTBYPASS
S1	MCPWM_DT*x*_A_OUTBYPASS
S2	MCPWM_DT*x*_RED_OUTINVERT
S3	MCPWM_DT*x*_FED_OUTINVERT
S4	MCPWM_DT*x*_RED_INSEL
S5	MCPWM_DT*x*_FED_INSEL
S6	MCPWM_DT*x*_A_OUTSWAP
S7	MCPWM_DT*x*_B_OUTSWAP
S8	MCPWM_DT*x*_DEB_MODE

所有开关组合皆受支持，但不是所有的开关模式都是典型的使用模式。表 33-6 列举了一些典型的死区配置。在这些配置中，S4 和 S5 的开关位置将 PWM*x*A 设置为下降沿和上升沿延迟的公共源。表 33-6 中的模式可分为以下几类：

- **模式 1：绕过下降沿 (FED) 和上升沿 (RED) 的延迟**
  该模式下，死区模块被关闭。PWM*x*A 和 PWM*x*B 信号的波形无变化。
- **模式 2-5：经典死区极性设置**
  这些模式为典型极性配置，涵盖工业电源栅极驱动器中的高 / 低电平有效模式。图 33-25 至 33-28为典型波形。
- **模式 6 和 7：绕过下降沿 (FED) 或上升沿 (RED) 的延迟**
  这两种模式下，绕过上升沿延迟 (RED) 或下降沿延迟 (FED) 。因此，无需使用对应延迟。

表 33-6. 死区生成器的典型操作模式

模式	描述	S0	S1	S2	S3
1	PWM*x*A 和 PWM*x*B 波形无变化	1	1	X	X
2	高电平有效互补 (AHC)，参见图 33-25	0	0	0	1
3	低电平有效互补 (ALC)，参见图 33-26	0	0	1	0
4	高电平有效 (AH)，参见图 33-27	0	0	0	0
5	低电平有效 (AL)，参见图 33-28	0	0	1	1
6	PWM*x*A 输出 = PWM*x*A 输入（无延迟） PWM*x*B 输出 = PWM*x*A 输入，下降沿延迟	0	1	0 或 1	0 或 1
7	PWM*x*A 输出 = PWM*x*A 输入，上升沿延迟 PWM*x*B 输出 = PWM*x*B 输入（无延迟）	1	0	0 或 1	0 或 1

说明：

以上所有模式中，S4 - S8 的开关位置都置 0。

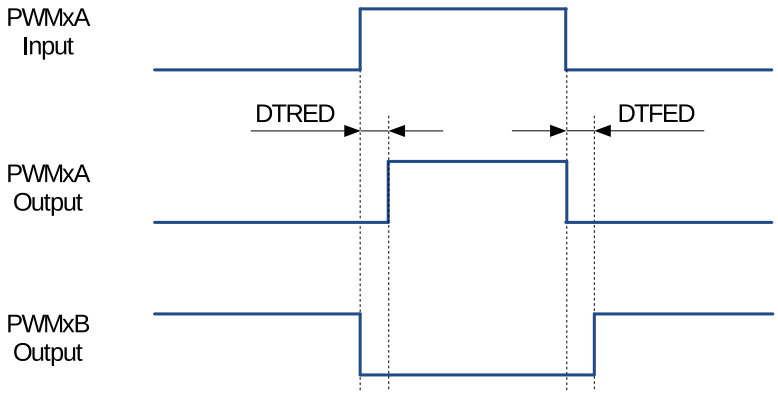


图 33-25. 高电平有效互补 (AHC) 死区波形

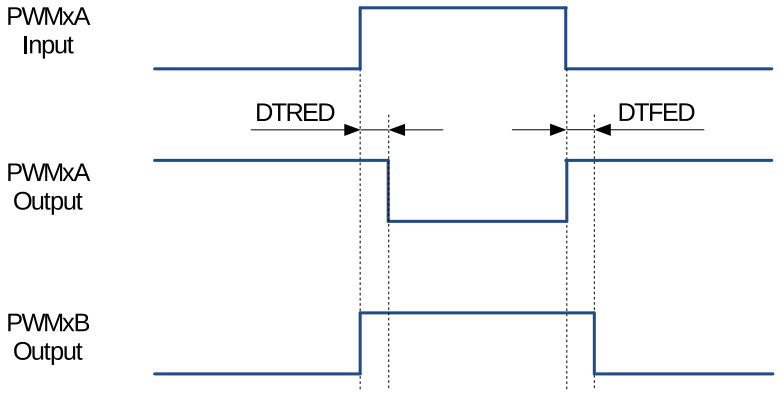


图 33-26. 低电平有效互补 (ALC) 死区波形

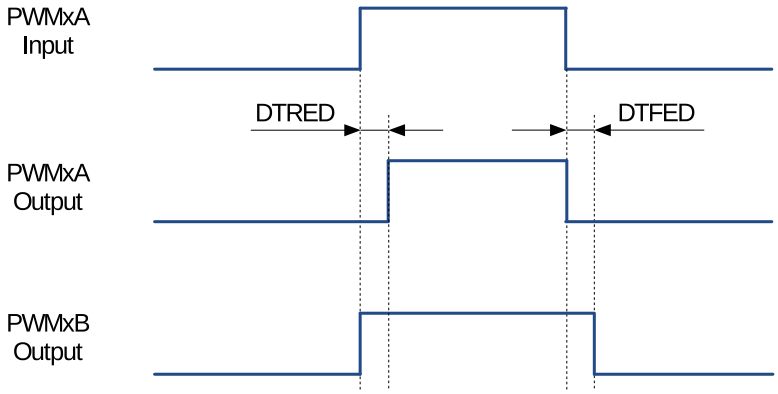


图 33-27. 高电平有效 (AH) 死区波形

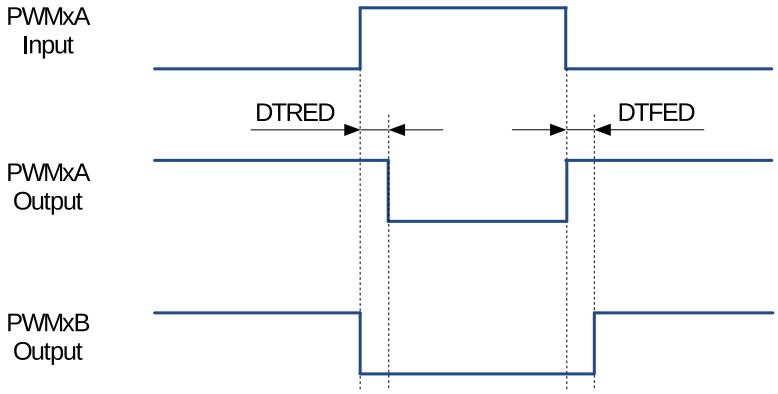


图 33-28. 低电平有效 (AL) 死区波形

上升沿延迟 (RED) 和下降沿延迟 (FED) 可分别设置。延迟的值通过 16 位寄存器 MCPWM_DT*x*_*RED* 和 MCPWM_DT*x*_*FED* 配置。寄存器值表示一个信号边沿可以延迟的 DT_CLK 时钟周期值。DT_CLK 可通过寄存器 MCPWM_DT*x*_CLK_SEL 从 PWM_CLK 或 PT_CLK 中选择。

通过以下公式计算下降沿延迟 (FED) 和上升沿延迟 (RED) 的值:

$$FED = MCPWM\_DTx\_FED \times T_{DT\_CLK}$$

$$RED = MCPWM\_DTx\_RED \times T_{DT\_CLK}$$

33.3.3.3 PWM 载波模块

将 PWM 输出耦合到电机驱动器可能需要使用变压器隔离。变压器只提供交流信号，而 PWM 信号的占空比可能在 0% 到 100% 之间变化。PWM 载波模块可以通过使用高频载波对其进行调制，将该信号传递给变压器。

功能概述

此模块的以下关键功能可配置:

- 载波频率
- 第一个脉冲的脉宽
- 第二个以及之后的脉冲的占空比
- 开启/关闭载波

操作要点

PWM 载波时钟 (PC_CLK) 来自于 PWM_CLK。通过寄存器 MCPWM_CARRIER*x*_CFG_REG 的 MCPWM_CARRIER*x*_PRESCALE 和 MCPWM_CARRIER*x*_DUTY 位配置频率和占空比。一次性脉冲的功能在于提供高能量脉冲以接通电源开关。随后的脉冲用于保持上电的状态。一次性脉冲宽度可通过 MCPWM_CARRIER*x*_OSHTWTH 字段进行配置。通过 MCPWM_CARRIER*x*_EN 位来使能/禁用载波模块。

载波示例

图 33-29 描述了载波叠加在原始 PWM 脉冲上的示例波形。图中未显示第一个脉冲和占空比控制，相关详细信息将在后两节中介绍。

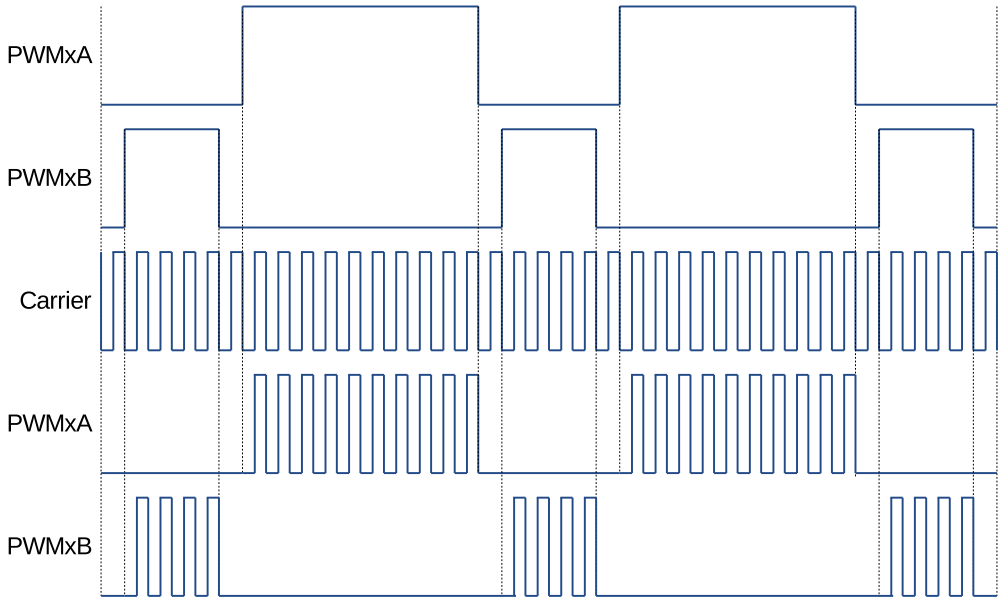


图 33-29. PWM 载波操作的波形示例

第一个脉冲

第一个脉冲的宽度可配置，其值有 16 种可能，可通过下公式计算：

$$T_{1stpulse} = T_{PWM\_CLK} \times 8 \times (MCPWM\_CARRIERx\_PRESCALE + 1) \times (MCPWM\_CARRIERx\_OSHTWTH + 1)$$

其中：

- T_{PMW_CLK} 为 PWM 时钟周期 (PWM_CLK)
- $(MCPWM\_CARRIERx\_OSHTWTH + 1)$ 为一次性脉冲宽度值（取值范围：1-16）
- $(MCPWM\_CARRIERx\_PRESCALE + 1)$ PWM 载波时钟 (PC_CLK) 预分频值

图 33-30 展示了第一个脉冲和之后持续的脉冲。

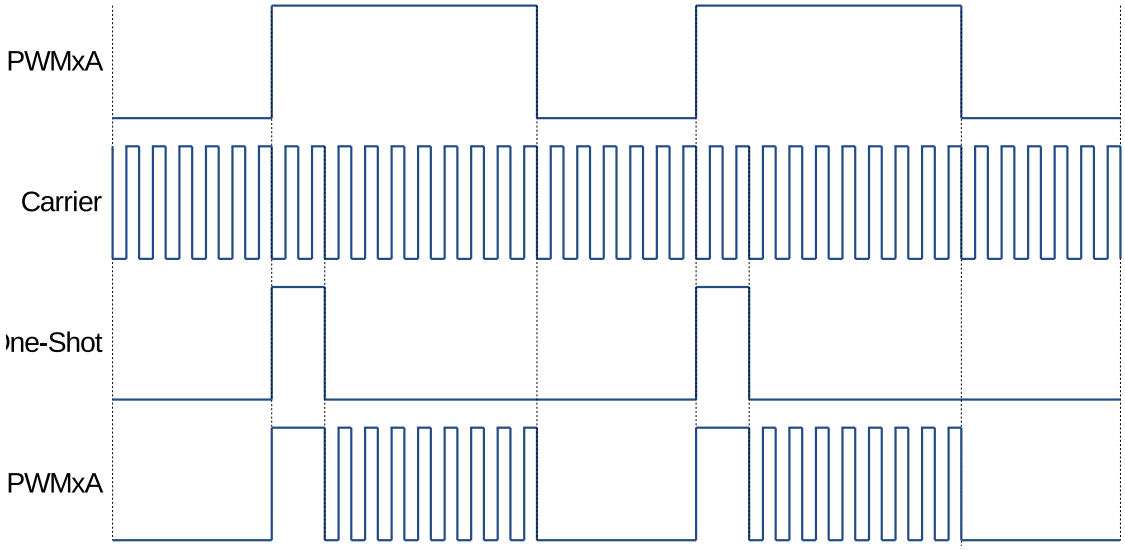


图 33-30. 载波模块的第一个脉冲和之后持续的脉冲示例

占空比控制

在发出第一个一次性脉冲之后，根据载波频率调制剩余的 PWM 信号。用户可配置该信号的占空比。在一定情况下，调整占空比可使信号通过隔离变压器后仍然可以开启或关闭电动机驱动器，改变电机旋转速度和方向。

占空比通过 MCPWM_CARRIERx_DUTY 或寄存器 MCPWM_CARRIERx_CFG_REG 的 [7:5] 位设置，其值有 7 种可能性。

占空比的值可通过以下方式计算：

$$Duty = MCPWM\_CARRIERx\_DUTY \div 8$$

图 33-31 为所有 7 种占空比设置。

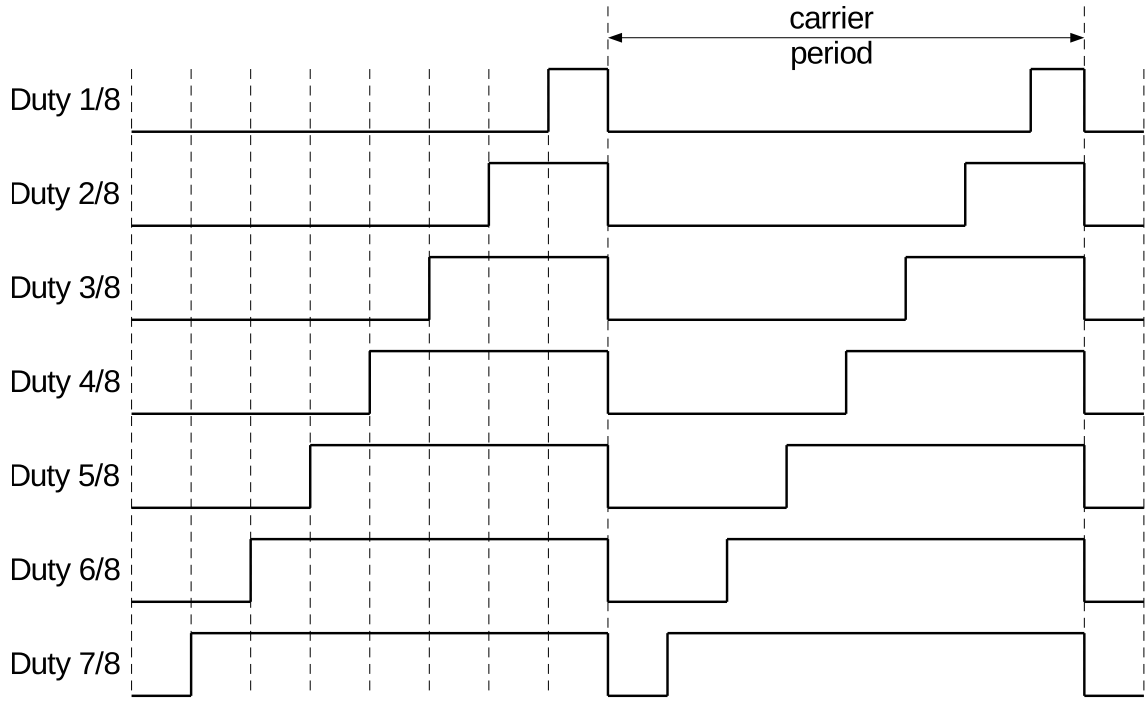


图 33-31. PWM 载波模块中持续脉冲的 7 种占空比设置

33.3.3.4 故障检测模块

每个 MCPWM 外设都连接来自 GPIO 矩阵的 3 个故障信号（FAULT0、FAULT1 和 FAULT2）。这些信号用于指示外部故障状况，并且可由故障检测模块预处理后生成故障事件。故障事件通过执行用户代码，针对特定故障调整 MCPWM 输出。

故障检测模块功能

故障检测模块的主要功能为：

- 在检测到故障时强制 PWM*x*A 和 PWM*x*B 输出信号进入以下状态之一：
  - 高
  - 低
  - 取反
  - 无
- 在检测到过电流/短路时执行一次性跳闸（OST）
- 逐周期跳闸（CBC）以提供限流操作
- 每个故障信号单独分配一次性或逐周期操作
- 每个故障输入都生成中断
- 支持软件强制跳闸
- 根据需要开启或关闭模块功能

操作与配置要点

本节介绍故障检测模块的操作要点和配置选项。

来自管脚的故障信号在 GPIO 矩阵中采样和同步。为了保证故障脉冲成功采样，每个脉冲持续时间必须至少为 2 个 APB 时钟周期。故障检测模块使用 PWM_CLK 对故障信号进行采样，因此，来自 GPIO 矩阵的故障脉冲持续时间必须至少为 1 个 PWM_CLK 周期。换句话说，无论 APB 时钟周期和 PWM_CLK 周期的大小关系如何，管脚上的故障信号脉冲的宽度必须至少等于两个 APB 时钟周期与一个 PWM_CLK 周期的和。

故障检测模块可以使用故障信号 FAULT0 至 FAULT2 中的高电平或低电平来生成故障事件 fault_event0 至 fault_event2。每个故障事件可以单独配置为进行 CBC 操作、OST 操作或无操作。

- **逐周期 (CBC) 操作：**
  触发 CBC 操作时，PWM*x*A 和 PWM*x*B 的状态立即根据字段 MCPWM_FH*x*_A_CBC_U/D 和 MCPWM_FH*x*_B_CBC_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 CBC 操作中断。通过状态字段 MCPWM_FH*x*_CBC_ON 开启或关闭 CBC 操作。在没有故障事件时，将在指定时间点，即发生 D/UTEP 或 D/UTEZ 事件时，清除 PWM*x*A/B 上的 CBC 操作。由字段 MCPWM_FH*x*_CBCPULSE 决定 PWM*x*A 和 PWM*x*B 恢复正常的事件。因此，在此模式下，每个 PWM 循环后，CBC 操作都将清除或刷新。
- **一次性（OST）操作：**
  触发 OST 操作时，PWM*x*A 和 PWM*x*B 的状态立即根据字段 MCPWM_FH*x*_A_OST_U/D 和 MCPWM_FH*x*_B_OST_U/D 改变。PWM 定时器递增或递减计数时，可配置不同的操作。不同的故障事件可触发不同的 OST 操作中断。通过状态字段 MCPWM_FH*x*_OST_ON 开启或关闭 OST 操作。在没有故障事件时，PWM*x*A/B 上的 OST 操作无法自动清除。须通过将 MCPWM_FH*x*_CLR_OST 位置 1 来清除一次性操作。

33.3.4 捕获模块

33.3.4.1 介绍

捕获模块包含 3 个完整的捕获通道。通道输入信号 CAP0、CAP1 和 CAP2 来自于 GPIO 矩阵。由于 GPIO 矩阵的灵活性，CAP0、CAP1 和 CAP2 可以通过任一管脚输入配置。多个捕获通道可以来自同一管脚输入，每个通道的预分频可以分别设置。同时，每个捕获通道也可以来自不同的管脚输入。因此，可以通过后台硬件用多种方式处理捕获信号，而不直接由 CPU 处理。

捕捉模块有以下独立资源：

- 一个 32 位定时器（计数器），可与 PWM 定时器、另一个模块或软件同步
- 三个捕获通道，每个通道配有一个 32 位时间戳和一个捕获预分频器
- 任何捕获通道的边沿极性（上升/下降沿）可独立选择
- 输入捕获信号预分频（分频取值范围：1 ~ 256）
- 三个捕获事件都有中断功能

33.3.4.2 捕获定时器

捕获定时器是一个 32 位计数器，使能时不断递增计数。将 MCPWM_CAP_TIMER_EN 置 1 使能捕获寄存器。其操作时钟源为 MCPWM 模块的主时钟。配置 MCPWM_CAP_SYNCI_EN 后，发生同步事件时，存储在

MCPWM_CAP_TIMER_PHASE_REG 寄存器中的相位将被加载至计数器中。同步事件可来自 PWM 定时器同步输出或 PWM 模块同步输入，通过 MCPWM_CAP_SYNCI_SEL 进行配置。也可将 MCPWM_CAP_SYNC_SW 置 1 生成同步事件。捕获定时器为所有三个捕获通道提供定时参考。

33.3.4.3 捕获通道

必要时，到达捕获通道的捕获信号可先被反相，然后预分频。每个捕获通道都有一个预分频寄存器 MCPWM_CAP*x*_PRESCALE。最后，预处理后的捕获信号的指定边沿将触发捕获事件。将 MCPWM_CAP*x*_EN 置 1 使能捕获通道。捕获事件将在 MCPWM_CAP*x*_MODE 配置的时间发生。在捕获事件发生时，捕获定时器的值存储在时间戳寄存器 MCPWM_CAP_CH*x*_REG 中。捕获事件中的不同捕获通道可生成不同的中断。触发捕获事件的边沿储存在寄存器 MCPWM_CAP*x*_EDGE 中。捕获事件可通过将 MCPWM_CAP*x*_SW 置 1 由软件强制发生。

33.3.5 ETM 模块

33.3.5.1 介绍

在 ESP32-H2 中，MCPWM 支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 MCPWM 的 ETM 任务，或者通过 MCPWM 的 ETM 事件触发任意外设的 ETM 任务。MCPWM 的捕获模块、故障检测模块、三个定时器以及三个操作器都可以独立产生事件或响应任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与 MCPWM 相关的 ETM 任务和 ETM 事件。

33.3.5.2 MCPWM 可产生的 ETM 事件

将使能字段置 1 后，满足对应生成条件时，将产生对应 ETM 事件，详见表 33-7：

表 33-7. MCPWM 可产生的 ETM 事件

使能字段	生成条件	产生事件
MCPWM_EVT_CAP*x*_EN	发生 CAP*x* 捕获事件	MCPWM_EVT_CAP*x*
MCPWM_EVT_TZ*x*_OST_EN	PWM 操作器 *x* 执行一次性跳闸 (OST) 操作	MCPWM_EVT_TZ*x*_OST
MCPWM_EVT_TZ*x*_CBC_EN	PWM 操作器 *x* 执行逐周期跳闸 (CBC) 操作	MCPWM_EVT_TZ*x*_CBC
MCPWM_EVT_F*x*_CLR_EN	清除故障事件 fault_event*x*	MCPWM_EVT_F*x*_CLR
MCPWM_EVT_F*x*_EN	产生故障事件 fault_event*x*	MCPWM_EVT_F*x*
MCPWM_EVT_OP*x*_TEB_EN	PWM 操作器 *x* 选择的定时器计数值等于时间戳 B 的值	MCPWM_EVT_OP*x*_TEB
MCPWM_EVT_OP*x*_TEA_EN	PWM 操作器 *x* 选择的定时器计数值等于时间戳 A 的值[1]	MCPWM_EVT_OP*x*_TEA
MCPWM_EVT_TIMER*x*_TEP_EN	定时器 *x* 的计数值等于周期值 MCPWM_TIMER*x*_PERIOD	MCPWM_EVT_TIMER*x*_TEP
MCPWM_EVT_TIMER*x*_TEZ_EN	定时器 *x* 的计数值等于 0	MCPWM_EVT_TIMER*x*_TEZ
MCPWM_EVT_TIMER*x*_STOP_EN	定时器 *x* 停止计数	MCPWM_EVT_OP*x*_TEA

[1] 有关时间戳 A 和时间戳 B 的描述，详见章节 33.3.3.1。

33.3.5.3 MCPWM 可接收的 ETM 任务

将使能字段置 1 后，接收对应有效任务后，将产生对应的响应操作，详见表 33-8：

表 33-8. MCPWM 可接收的 ETM 任务

使能字段	接收的有效任务	响应操作
MCPWM_TASK_CAP*x*_EN	MCPWM_TASK_CAP*x*	CAP*x* 通道进行捕获操作
MCPWM_TASK_CLR*x*_OST_EN	MCPWM_TASK_CLR*x*_OST	清除 PWM 操作器*x* 的一次性跳闸 (OST) 操作
MCPWM_TASK_TZ*x*_OST_EN	MCPWM_TASK_TZ*x*_OST	PWM 操作器*x* 执行一次性跳闸 (OST) 操作
MCPWM_TASK_TIMER*x*_PERIOD_UP_EN	MCPWM_TASK_TIMER*x*_PERIOD_UP	定时器 *x* 的周期更新为周期寄存器 MCPWM_TIMER*x*_PERIOD 配置的值
MCPWM_TASK_TIMER*x*_SYNC_EN	MCPWM_TASK_TIMER*x*_SYN	定时器*x* 执行一次同步操作
MCPWM_TASK_GEN_STOP_EN	MCPWM_TASK_GEN_STOP	所有定时器停止计数，所有的 PWM 操作器输出的 PWM 信号保持不变
MCPWM_TASK_CMPR*x*_B_UP_EN	MCPWM_TASK_CMPR*x*_B_UP	将 PWM 操作器*x* 的时间戳 B 更新为时间戳 B 影子寄存器 MCPWM_GEN*x*_B 的值
MCPWM_TASK_CMPR*x*_A_UP_EN	MCPWM_TASK_CMPR*x*_A_UP	将 PWM 操作器*x* 的时间戳 A 更新为时间戳 A 影子寄存器 MCPWM_GEN*x*_A 的值。

33.3.6 中断

- MCPWM_TIMER*x*_STOP_INT：定时器*x*停止后触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPMW_TIMER*x*_STOP_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_TIMER*x*_TEZ_INT：PWM 定时器*x* TEZ 事件触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_TIMER*x*_TEZ_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_TIMER*x*_TEP_INT：PWM 定时器*x* TEP 事件触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_TIMER*x*_TEP_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_FAULT*x*_INT：fault_event*x* 开始后触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_FAULT*x*_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_FAULT*x*_CLR_INT：fault_event*x* 结束后触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_FAULT*x*_CLR_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_CMPR*x*_TEA_INT：PWM 操作器*x* TEA 事件触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_CMPR*x*_TEA_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_CMPR*x*_TEB_INT：PWM 操作器*x* TEB 事件触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_CMPR*x*_TEB_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_TZ*x*_CBC_INT：由 PWM*x* 逐周期操作触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_TZ*x*_CBC_INT_ENA 字段置位后，才能触发该中断。
- MCPWM_TZ*x*_OST_INT：由 PWM*x* 一次性操作触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_TZ*x*_OST_INT_ENA 字段置位后，才能触发该中断。

- MCPWM_CAP*x*_INT：由信道*x* 上捕获事件触发的中断。当寄存器 MCPMW_INT_ENA_REG 的 MCPWM_CAP*x*_INT_ENA 字段置位后，才能触发该中断。

33.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
预分频器配置			
MCPWM_CLK_CFG_REG	配置预分频器	0x0000	R/W
MCPWM 定时器 0 配置与状态			
MCPWM_TIMER0_CFG0_REG	定时器周期与更新方法	0x0004	R/W
MCPWM_TIMER0_CFG1_REG	工作模式与开始/停止控制	0x0008	varies
MCPWM_TIMER0_SYNC_REG	PWM 定时器 0 同步功能配置寄存器	0x000C	R/W
MCPWM_TIMER0_STATUS_REG	PWM 定时器 0 状态	0x0010	RO
MCPWM 定时器 1 配置与状态			
MCPWM_TIMER1_CFG0_REG	定时器更新方式与周期	0x0014	R/W
MCPWM_TIMER1_CFG1_REG	工作模式与开始/停止控制	0x0018	varies
MCPWM_TIMER1_SYNC_REG	同步设置	0x001C	R/W
MCPWM_TIMER1_STATUS_REG	PWM 定时器 1 状态	0x0020	RO
MCPWM 定时器 2 配置与状态			
MCPWM_TIMER2_CFG0_REG	定时器方式与周期	0x0024	R/W
MCPWM_TIMER2_CFG1_REG	工作模式与开始/停止控制	0x0028	varies
MCPWM_TIMER2_SYNC_REG	同步设置	0x002C	R/W
MCPWM_TIMER2_STATUS_REG	PWM 定时器 2 状态	0x0030	RO
MCPWM 定时器常见配置			
MCPWM_TIMER_SYNCI_CFG_REG	定时器同步输入选择	0x0034	R/W
MCPWM_OPERATOR_TIMERSEL_REG	为 PWM 操作器选择特定的计时器	0x0038	R/W
MCPWM 操作器 0 配置与状态			
MCPWM_GEN0_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x003C	varies
MCPWM_GEN0_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0040	R/W
MCPWM_GEN0_TSTMP_B_REG	寄存器 B 的影子寄存器	0x0044	R/W
MCPWM_GEN0_CFG0_REG	故障事件 T0 和 T1 处理	0x0048	R/W
MCPWM_GEN0_FORCE_REG	软件强制 PWM0A 和 PWM0B 输出	0x004C	R/W
MCPWM_GEN0_A_REG	PWM0A 输出上事件触发的操作	0x0050	R/W
MCPWM_GEN0_B_REG	PWM0B 输出上事件触发的操作	0x0054	R/W
MCPWM_DT0_CFG_REG	死区与类型的选择与配置	0x0058	R/W
MCPWM_DT0_FED_CFG_REG	FED 的影子寄存器	0x005C	R/W
MCPWM_DT0_RED_CFG_REG	RED 的影子寄存器	0x0060	R/W
MCPWM_CARRIER0_CFG_REG	载波使能与配置	0x0064	R/W
MCPWM_FH0_CFG0_REG	故障事件中 PWM0A 和 PWM0B 上的操作	0x0068	R/W
MCPWM_FH0_CFG1_REG	故障处理的软件触发	0x006C	R/W
MCPWM_FH0_STATUS_REG	故障事件状态	0x0070	RO
MCPWM 操作器 1 配置与状态			
MCPWM_GEN1_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x0074	varies
MCPWM_GEN1_TSTMP_A_REG	寄存器 A 的影子寄存器	0x0078	R/W

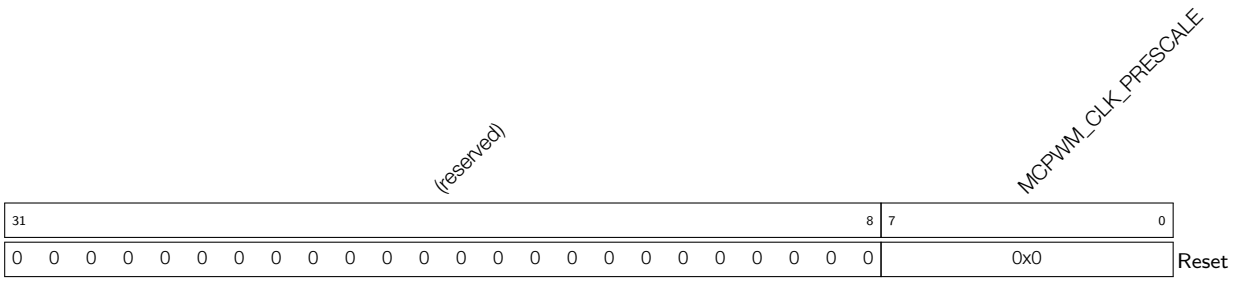
名称	描述	地址	访问
MCPWM_GEN1_TSTMP_B_REG	寄存器 B 的影子寄存器	0x007C	R/W
MCPWM_GEN1_CFG0_REG	故障事件 T0 和 T1 处理	0x0080	R/W
MCPWM_GEN1_FORCE_REG	软件强制 PWM1A 和 PWM1B 输出	0x0084	R/W
MCPWM_GEN1_A_REG	PWM1A 输出上的事件触发的操作	0x0088	R/W
MCPWM_GEN1_B_REG	PWM1B 输出上的事件触发的操作	0x008C	R/W
MCPWM_DT1_CFG_REG	死区类型的选择与配置	0x0090	R/W
MCPWM_DT1_FED_CFG_REG	FED 的影子寄存器	0x0094	R/W
MCPWM_DT1_RED_CFG_REG	RED 的影子寄存器	0x0098	R/W
MCPWM_CARRIER1_CFG_REG	使能与配置载波	0x009C	R/W
MCPWM_FH1_CFG0_REG	故障事件中 PWM1A 和 PWM1B 输出上的操作	0x00A0	R/W
MCPWM_FH1_CFG1_REG	故障处理的软件触发	0x00A4	R/W
MCPWM_FH1_STATUS_REG	故障事件状态	0x00A8	RO
MCPWM 操作器 2 的配置与状态			
MCPWM_GEN2_STMP_CFG_REG	时间戳寄存器 A 和 B 的传输状态和更新方式	0x00AC	varies
MCPWM_GEN2_TSTMP_A_REG	寄存器 A 的影子寄存器	0x00B0	R/W
MCPWM_GEN2_TSTMP_B_REG	寄存器 B 的影子寄存器	0x00B4	R/W
MCPWM_GEN2_CFG0_REG	故障事件 T0 和 T1 处理	0x00B8	R/W
MCPWM_GEN2_FORCE_REG	软件强制 PWM2A 和 PWM2B 输出	0x00BC	R/W
MCPWM_GEN2_A_REG	PWM2A 输出上的事件触发的操作	0x00C0	R/W
MCPWM_GEN2_B_REG	PWM2B 输出上的事件触发的操作	0x00C4	R/W
MCPWM_DT2_CFG_REG	死区类型的选择与配置	0x00C8	R/W
MCPWM_DT2_FED_CFG_REG	FED 影子寄存器	0x00CC	R/W
MCPWM_DT2_RED_CFG_REG	RED 影子寄存器	0x00D0	R/W
MCPWM_CARRIER2_CFG_REG	使能与配置载波	0x00D4	R/W
MCPWM_FH2_CFG0_REG	故障事件中 PWM2A 和 PWM2B 输出上的操作	0x00D8	R/W
MCPWM_FH2_CFG1_REG	故障处理的软件触发	0x00DC	R/W
MCPWM_FH2_STATUS_REG	故障事件状态	0x00E0	RO
故障检测与配置			
MCPWM_FAULT_DETECT_REG	故障检测与配置	0x00E4	varies
捕获配置与状态			
MCPWM_CAP_TIMER_CFG_REG	配置捕获定时器	0x00E8	varies
MCPWM_CAP_TIMER_PHASE_REG	捕获定时器同步相位	0x00EC	R/W
MCPWM_CAP_CH0_CFG_REG	捕获通道 0 的配置与使能	0x00F0	varies
MCPWM_CAP_CH1_CFG_REG	捕获通道 1 的配置与使能	0x00F4	varies
MCPWM_CAP_CH2_CFG_REG	捕获通道 2 的配置与使能	0x00F8	varies
MCPWM_CAP_CH0_REG	捕获通道 0 值的状态	0x00FC	RO
MCPWM_CAP_CH1_REG	捕获通道 1 值的状态	0x0100	RO
MCPWM_CAP_CH2_REG	捕获通道 2 值的状态	0x0104	RO
MCPWM_CAP_STATUS_REG	上一次捕获触发器的边沿	0x0108	RO
使能有效寄存器的更新			
MCPWM_UPDATE_CFG_REG	使能更新	0x010C	R/W
管理中断			
MCPWM_INT_ENA_REG	中断使能位	0x0110	R/W

名称	描述	地址	访问
MCPWM_INT_RAW_REG	原始中断状态	0x0114	R/ WTC / SS
MCPWM_INT_ST_REG	屏蔽中断状态	0x0118	RO
MCPWM_INT_CLR_REG	中断清除位	0x011C	WT
使能 MCPWM 事件			
MCPWM_EVT_EN_REG	使能 MCPWM 事件	0x0120	R/W
使能 MCPWM 任务			
MCPWM_TASK_EN_REG	使能 MCPWM 任务	0x0124	R/W
MCPWM APB 配置			
MCPWM_CLK_REG	MCPWM APB 配置	0x0128	R/W
版本寄存器			
MCPWM_VERSION_REG	版本控制寄存器	0x012C	R/W

33.5 寄存器

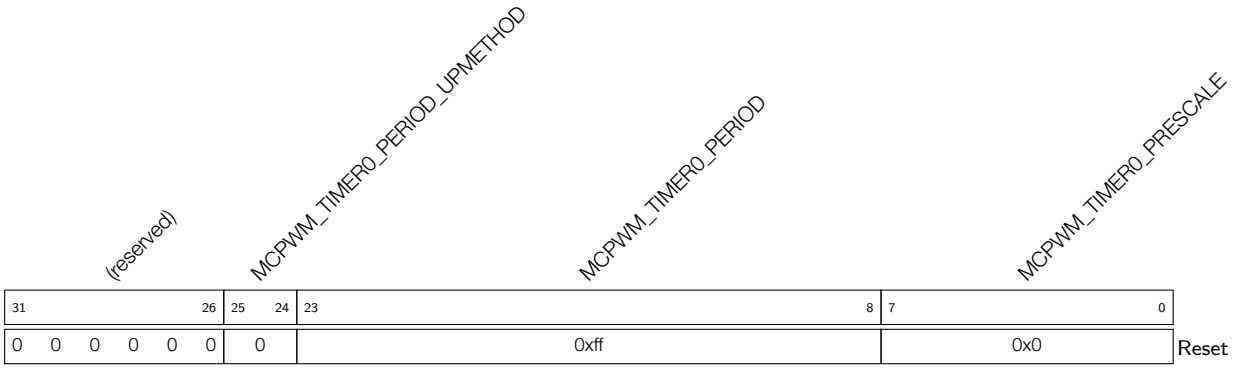
本小节的所有地址均为相对于电机控制器基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 33.1. MCPWM_CLK_CFG_REG (0x0000)



MCPWM_CLK_PRESCALE 配置时钟预分频系数，使得 PWM_CLK 的周期为 6.25ns * (PWM_CLK_PRESCALE + 1)。(R/W)

Register 33.2. MCPWM_TIMER0_CFG0_REG (0x0004)



MCPWM_TIMER0_PRESCALE 配置定时器 0 的预分频系数，使得 PT0_CLK 的周期 = PWM_CLK 的周期 * (PWM_TIMER0_PRESCALE + 1)。(R/W)

MCPWM_TIMER0_PERIOD 配置定时器 0 的影子周期。(R/W)

MCPWM_TIMER0_PERIOD_UPMETHOD 配置 PWM 定时器 0 周期有效寄存器的更新方式。

0：立即更新

1：发生 TEZ 事件时更新

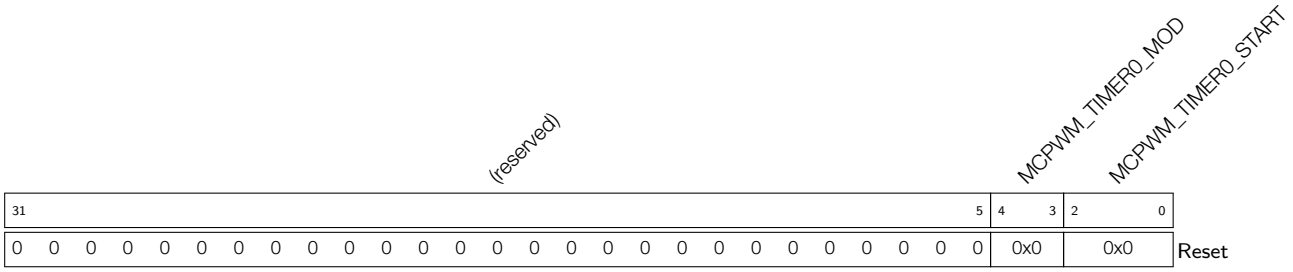
2：同步时更新

3：发生 TEZ 事件或同步时更新

本文档中，TEZ 指定时器为 0 时的事件。

(R/W)

Register 33.3. MCPWM_TIMER0_CFG1_REG (0x0008)



MCPWM_TIMER0_START 控制 PWM 定时器的开启与关闭。

0：如果开启，在 TEZ 事件发生时停止

1：如果开启，在 TEP 事件发生时停止

2：开启

3：开启，并在下一个 TEZ 事件发生时停止

4：开启，并在下一个 TEP 事件发生时停止

其他值：无效值，不产生影响

本文档中，TEP 指定时器为周期值时发生的事件。

(R/W/SC)

MCPWM_TIMER0_MOD 配置 PWM 定时器 0 的工作模式。

0：暂停

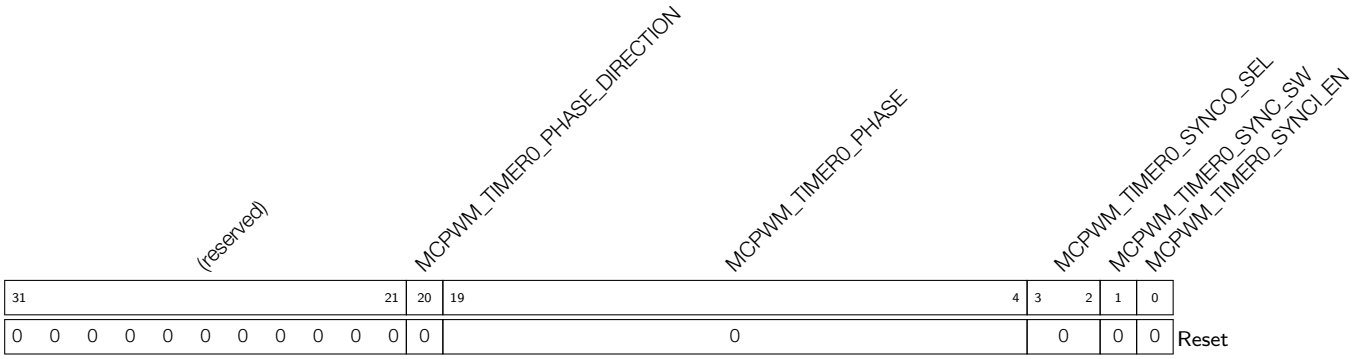
1：递增模式

2：递减模式

3：递增递减循环模式

(R/W)

Register 33.4. MCPWM_TIMER0_SYNC_REG (0x000C)



MCPWM_TIMER0_SYNCI_EN 置 1 时，使能在同步输入事件发生时的定时器相位重载。(R/W)

MCPWM_TIMER0_SYNC_SW 此位取反，触发软件同步。(R/W)

MCPWM_TIMER0_SYNCO_SEL 选择 PWM 定时器 0 的同步输出来源。

0：同步。取反 MCPWM_TIMER0_SYNC_SW 位时始终生成同步输出。

1：TEZ

2：TEP

3：无效

(R/W)

MCPWM_TIMER0_PHASE 同步事件中定时器重载的相位。(R/W)

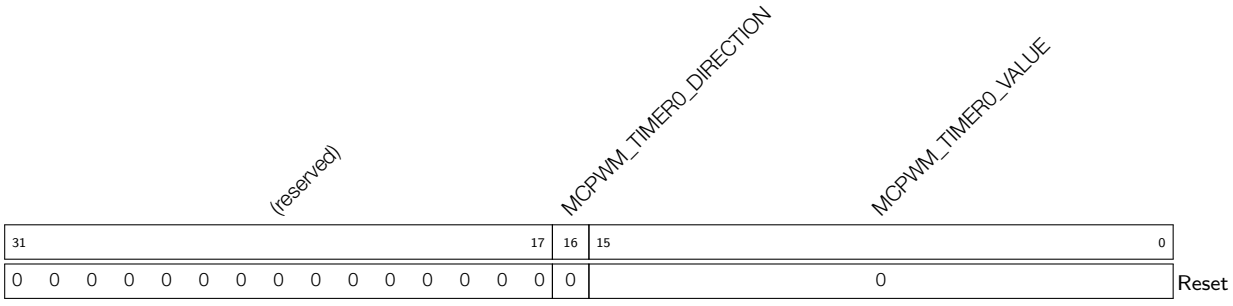
MCPWM_TIMER0_PHASE_DIRECTION 当定时器 0 为递增-递减循环模式时，配置该定时器方向。

0：递增

1：递减

(R/W)

Register 33.5. MCPWM_TIMER0_STATUS_REG (0x0010)



MCPWM_TIMER0_VALUE 表示当前 PWM 定时器 0 计数器的值。(RO)

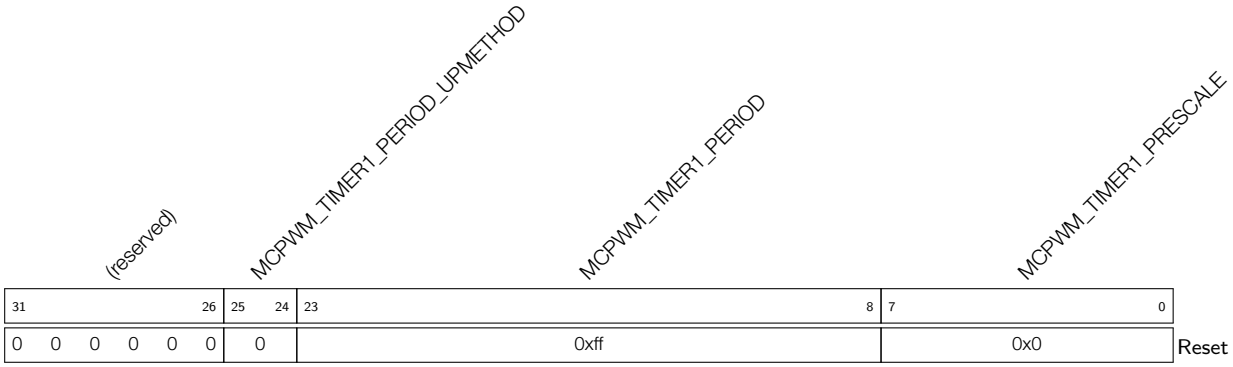
MCPWM_TIMER0_DIRECTION 配置当前 PWM 定时器 0 的计数器模式。

0：递增模式

1：递减模式

(RO)

Register 33.6. MCPWM_TIMER1_CFG0_REG (0x0014)



MCPWM_TIMER1_PRESCALE 配置 timer1 预分频系数，使得 PT0_CLK 周期 = PWM_CLK 周期 * (PWM_timer1_PRESCALE + 1)。(R/W)

MCPWM_TIMER1_PERIOD 定时器 1 的影子周期寄存器。(R/W)

MCPWM_TIMER1_PERIOD_UPMETHOD 配置 PWM 定时器 1 周期有效寄存器的更新方式。

0：立即更新

1：发生 TEZ 事件时更新

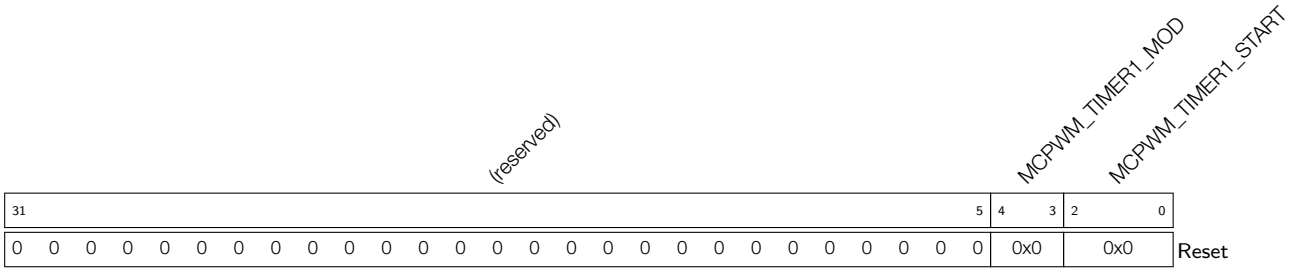
2：同步时更新

3：发生 TEZ 事件或同步时更新

本文档中，TEZ 指定时器为 0 时的事件。

(R/W)

Register 33.7. MCPWM_TIMER1_CFG1_REG (0x0018)



MCPWM_TIMER1_START PWM 控制定时器 1 的开启与停止。

0：如果开启，在发生 TEZ 事件时停止

1：如果开启，在发生 TEP 事件时停止

2：开启

3：开启并在下一次 TEZ 事件中停止

4：开启并在下一次 TEP 事件中停止

其他值：无效值，不产生影响。

本文档中，TEP 指定时器为周期数时的事件。

(R/W/SC)

MCPWM_TIMER1_MOD PWM 定时器 1 的工作模式。

0：暂停

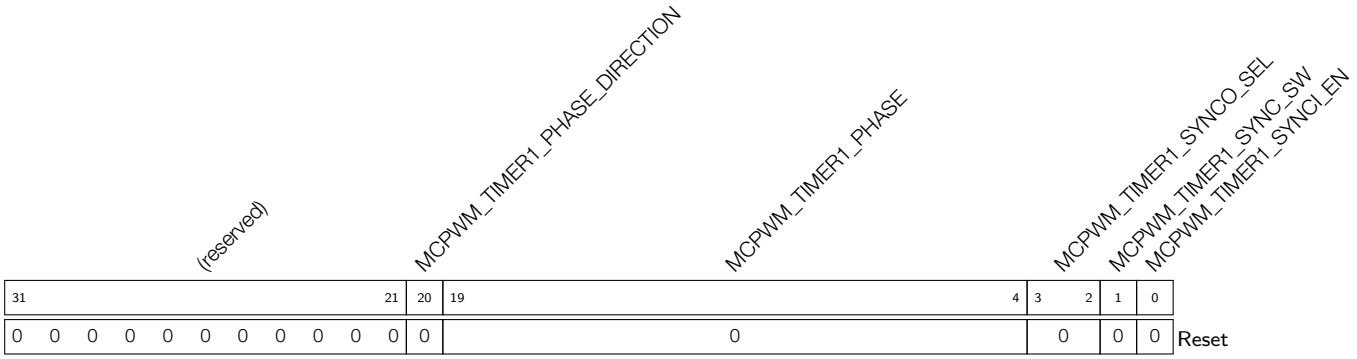
1：递增模式

2：递减模式

3：递增-递减循环模式

(R/W)

Register 33.8. MCPWM_TIMER1_SYNC_REG (0x001C)



MCPWM_TIMER1_SYNCI_EN 置 1 时，使能在同步输入事件时的定时器相位重载。(R/W)

MCPWM_TIMER1_SYNC_SW 此位取反，触发软件同步事件。(R/W)

MCPWM_TIMER1_SYNCO_SEL 选择 PWM 定时器 1 的同步输出来源。

0：同步

1：TEZ

2：TEP。取反 reg_timer1_sw 位时始终生成同步输出

3：无效

(R/W)

MCPWM_TIMER1_PHASE 同步时间中定时器重载的相位。(R/W)

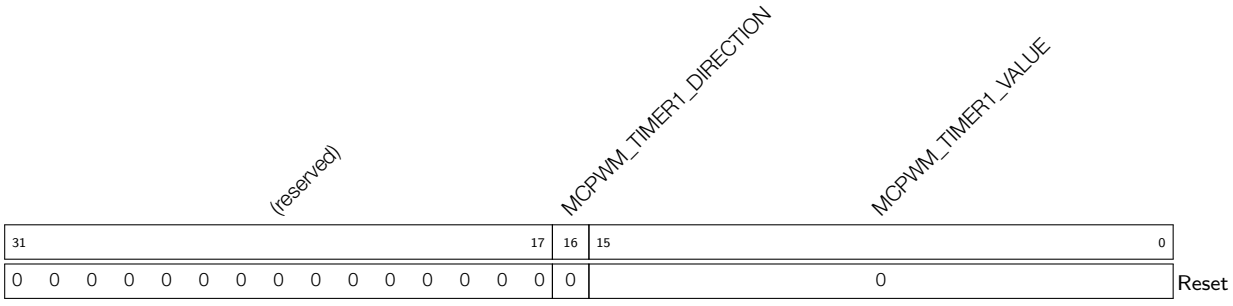
MCPWM_TIMER1_PHASE_DIRECTION 当定时器 1 为递增-递减循环模式时，配置该定时器方向。

0：递增

1：递减

(R/W)

Register 33.9. MCPWM_TIMER1_STATUS_REG (0x0020)



MCPWM_TIMER1_VALUE 配置当前 PWM 定时器 1 的计数器值。(RO)

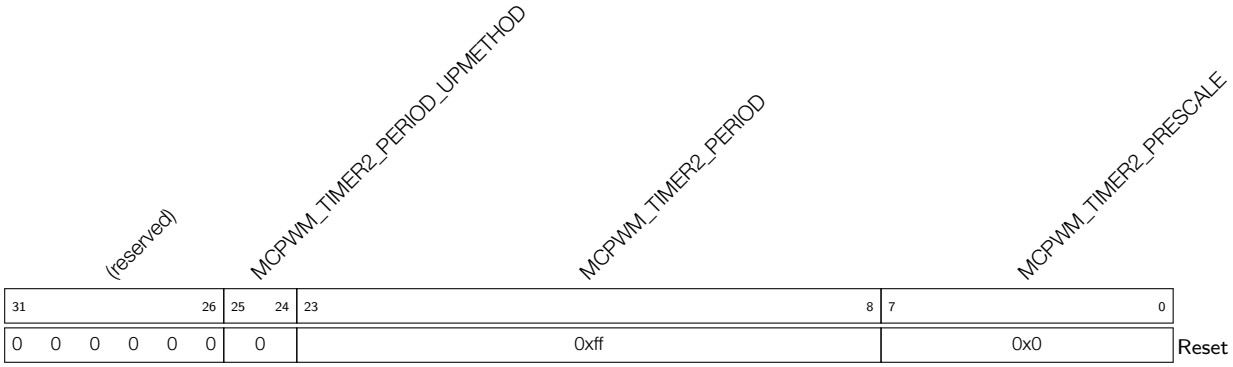
MCPWM_TIMER1_DIRECTION 配置当前 PWM 定时器 1 的计数器模式。

0：递增

1：递减

(RO)

Register 33.10. MCPWM_TIMER2_CFG0_REG (0x0024)



MCPWM_TIMER2_PRESCALE 配置定时器 2 的预分频系数，使得 PT0_CLK 周期 = PWM_CLK 周期 * (PWM_timer2_PRESCALE + 1)。(R/W)

MCPWM_TIMER2_PERIOD PWM 定时器 2 的影子周期寄存器。(R/W)

MCPWM_TIMER2_PERIOD_UPMETHOD 配置 PWM 定时器 2 周期有效寄存器的更新方式。

0：立即更新

1：发生 TEZ 事件时更新

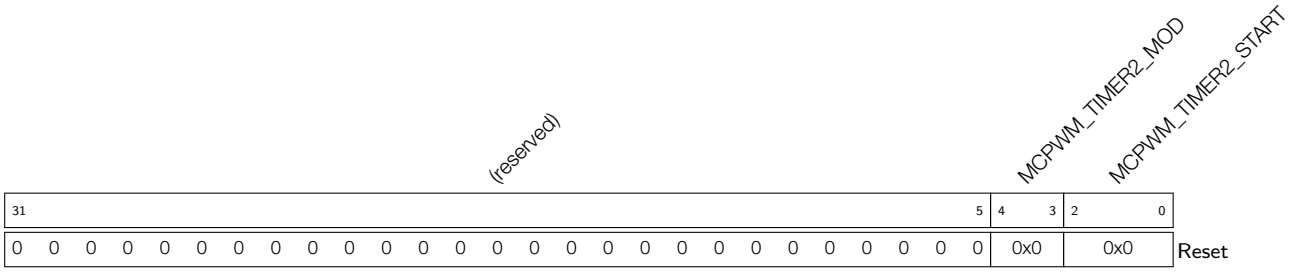
2：同步时更新

3：发生 TEZ 事件或同步时更新

本文档中，TEZ 指定时器为 0 时的事件。

(R/W)

Register 33.11. MCPWM_TIMER2_CFG1_REG (0x0028)



MCPWM_TIMER2_START 配置 PWM 控制定时器 2 的开启与停止。

0：如果开启，在发生 TEZ 事件时停止

1：如果开启，在发生 TEP 事件时停止

2：开启

3：开启并在下一次 TEZ 事件中停止

4：开启并在下一次 TEP 事件中停止

其他值：无效值，不产生影响

本文档中，TEP 指定时器为周期数时的事件。

(R/W/SC)

MCPWM_TIMER2_MOD 配置 PWM 定时器 1 的工作模式。

0：暂停

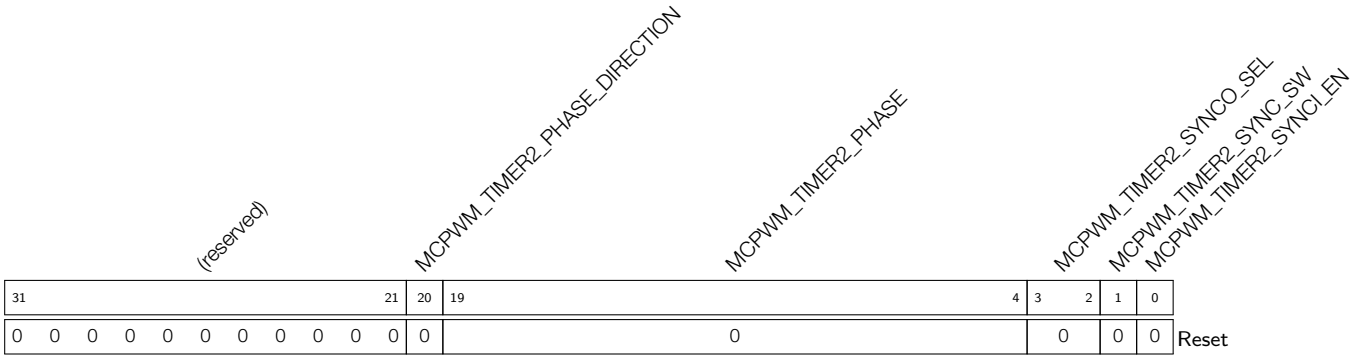
1：递增模式

2：递减模式

3：递增-递减循环模式

(R/W)

Register 33.12. MCPWM_TIMER2_SYNC_REG (0x002C)



MCPWM_TIMER2_SYNCI_EN 置 1 时使能在同步输入事件时的定时器相位重载。(R/W)

MCPWM_TIMER2_SYNC_SW 此位取反，触发软件同步事件。(R/W)

MCPWM_TIMER2_SYNCO_SEL 选择 PWM 定时器 2 的同步输出来源。

0：同步

1：TEZ

2：TEP。取反 reg_timer2_sw 位时始终生成同步输出

3：无效

(R/W)

MCPWM_TIMER2_PHASE 同步事件中定时器重载相位。(R/W)

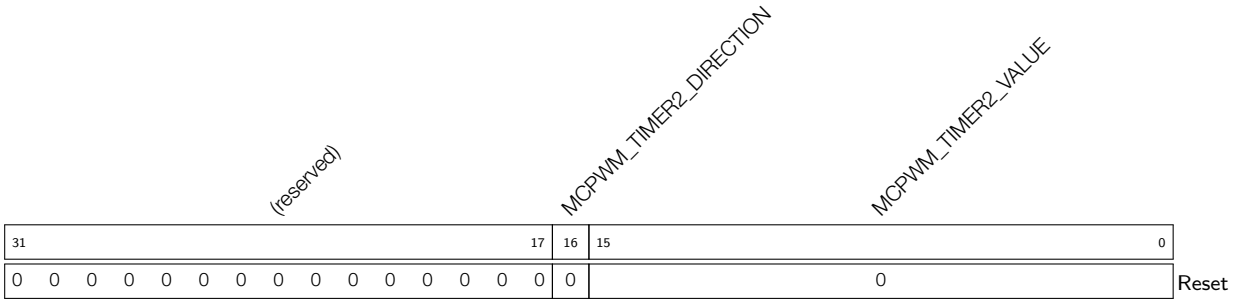
MCPWM_TIMER2_PHASE_DIRECTION 当定时器 2 为递增-递减循环模式时，配置该定时器方向。

0：递增

1：递减

(R/W)

Register 33.13. MCPWM_TIMER2_STATUS_REG (0x0030)



MCPWM_TIMER2_VALUE 表示当前 PWM 定时器 2 计数器的值。(RO)

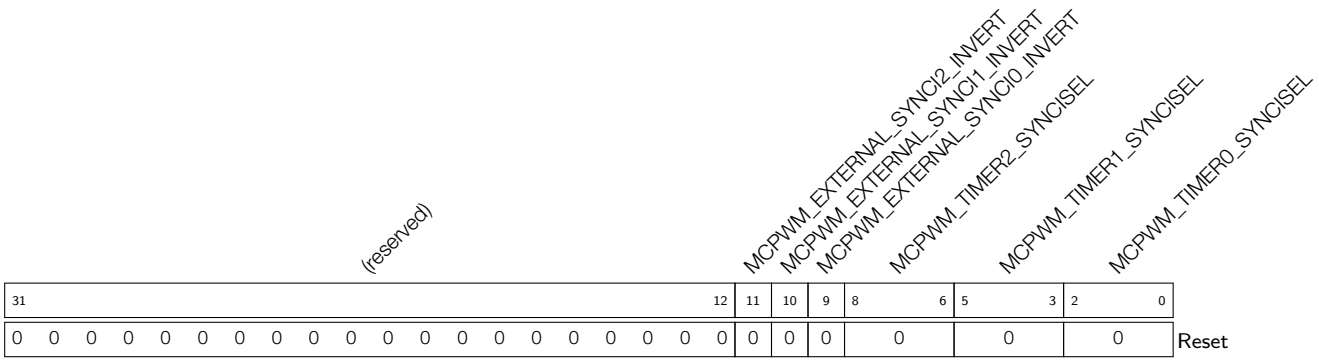
MCPWM_TIMER2_DIRECTION 表示当前 PWM 定时器 2 的计数器模式。

0：递增

1：递减

(RO)

Register 33.14. MCPWM_TIMER_SYNCI_CFG_REG (0x0034)



MCPWM_TIMER0_SYNCISEL 选择 PWM 定时器 0 的同步输入来源。

1：PWM 定时器 0 同步输出
2：PWM 定时器 1 同步输出
3：PWM 定时器 2 同步输出
4：来自 GPIO 矩阵的 SYNC0
5：来自 GPIO 矩阵的 SYNC1
6：来自 GPIO 矩阵的 SYNC2
其他值：未选择任何同步输入
(R/W)

MCPWM_TIMER1_SYNCISEL 选择 PWM 定时器 1 的同步输入来源。

1：PWM 定时器 0 同步输出
2：PWM 定时器 1 同步输出
3：PWM 定时器 2 同步输出
4：来自 GPIO 矩阵的 SYNC0
5：来自 GPIO 矩阵的 SYNC1
6：来自 GPIO 矩阵的 SYNC2
其他值：未选择任何同步输入
(R/W)

MCPWM_TIMER2_SYNCISEL 选择 PWM 定时器 2 的同步输入来源。

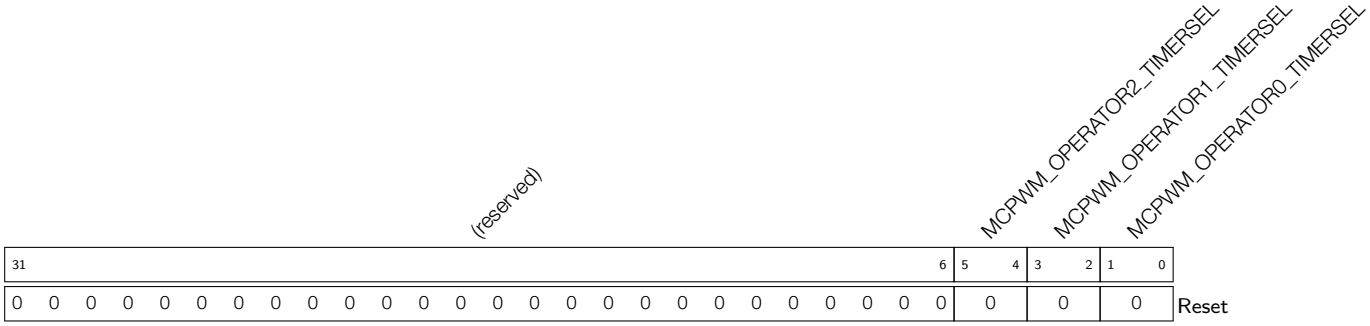
1：PWM 定时器 0 同步输出
2：PWM 定时器 1 同步输出
3：PWM 定时器 2 同步输出
4：来自 GPIO 矩阵的 SYNC0
5：来自 GPIO 矩阵的 SYNC1
6：来自 GPIO 矩阵的 SYNC2
其他值：未选择任何同步输入
(R/W)

MCPWM_EXTERNAL_SYNCI0_INVERT 将来自 GPIO 矩阵的 SYNC0 反相。(R/W)

MCPWM_EXTERNAL_SYNCI1_INVERT 将来自 GPIO 矩阵的 SYNC1 反相。(R/W)

MCPWM_EXTERNAL_SYNCI2_INVERT 将来自 GPIO 矩阵的 SYNC2 反相。(R/W)

Register 33.15. MCPWM_OPERATOR_TIMERSEL_REG (0x0038)



MCPWM_OPERATOR0_TIMERSEL 选择 PWM 操作器 0 的定时参考来源。

0：定时器 0

1：定时器 1

2：定时器 2

3：无效

(R/W)

MCPWM_OPERATOR1_TIMERSEL 选择 PWM 操作器 1 的定时参考来源。

0：定时器 0

1：定时器 1

2：定时器 2

3：无效

(R/W)

MCPWM_OPERATOR2_TIMERSEL 选择 PWM 操作器 2 的定时参考来源。

0：定时器 0

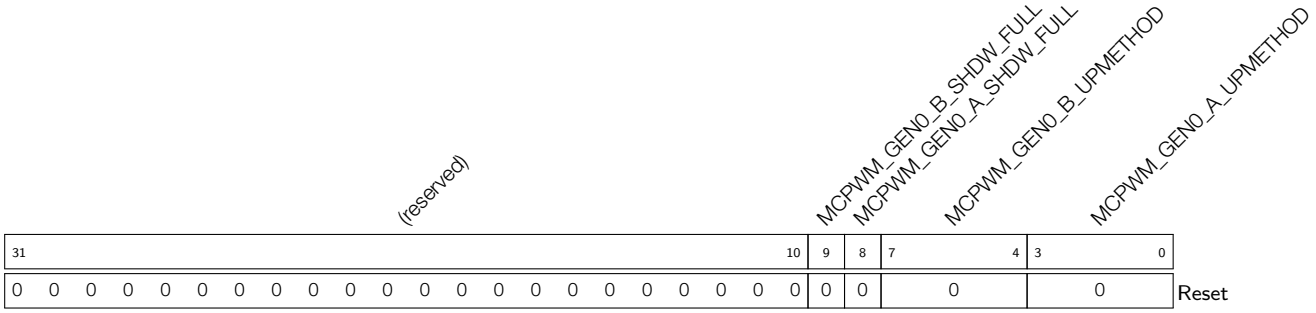
1：定时器 1

2：定时器 2

3：无效

(R/W)

Register 33.16. MCPWM_GEN0_STMP_CFG_REG (0x003C)



MCPWM_GEN0_A_UPMETHOD 配置 PWM 生成器 0 时间戳寄存器 A 的有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN0_B_UPMETHOD 配置 PWM 生成器 0 时间戳寄存器 B 有效寄存器的更新方式。

所有位值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN0_A_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 A 中

1：将值写入 PWM 生成器 0 时间戳寄存器 A 的影子寄存器中，等待传输给有效寄存器 A。

(R/SC/WTC)

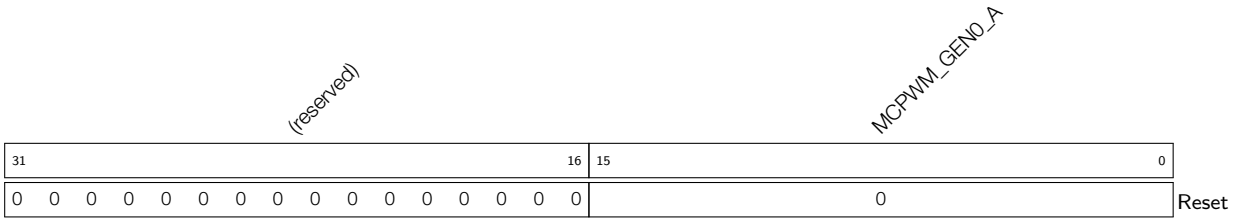
MCPWM_GEN0_B_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 B 中

1：将值写入 PWM 生成器 0 时间戳寄存器 B 的影子寄存器中，等待传输给有效寄存器 B。

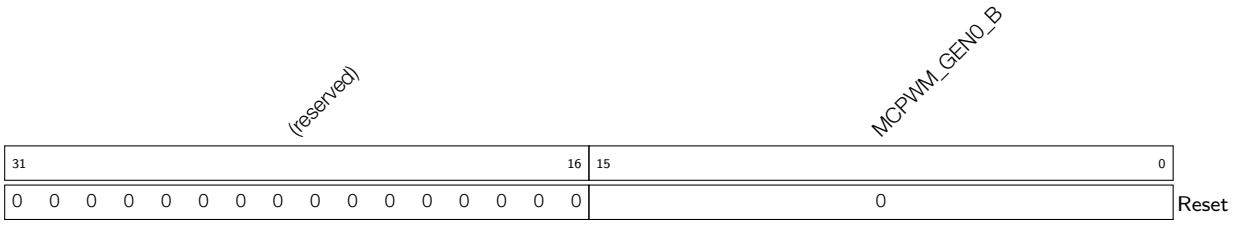
(R/SC/WTC)

Register 33.17. MCPWM_GEN0_TSTMP_A_REG (0x0040)



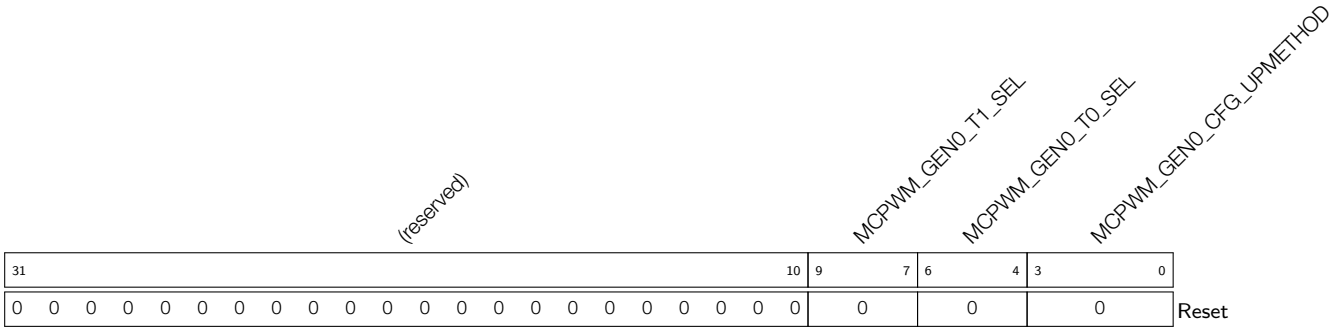
MCPWM_GEN0_A PWM 生成器 0 时间戳寄存器 A 的影子寄存器。(R/W)

Register 33.18. MCPWM_GEN0_TSTMP_B_REG (0x0044)



MCPWM_GEN0_B PWM 生成器 0 时间戳寄存器 B 的影子寄存器。(R/W)

Register 33.19. MCPWM_GEN0_CFG0_REG (0x0048)



MCPWM_GEN0_CFG_UPMETHOD PWM 生成器 0 有效配置寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN0_T0_SEL 选择 PWM 生成器 0 event_t0 的信号源，立即生效。

0：fault_event0

1：fault_event1

2：fault_event2

3：sync_taken

4：无

(R/W)

MCPWM_GEN0_T1_SEL 选择 PWM 生成器 0 event_t1 的信号源，立即生效。

0：fault_event0

1：fault_event1

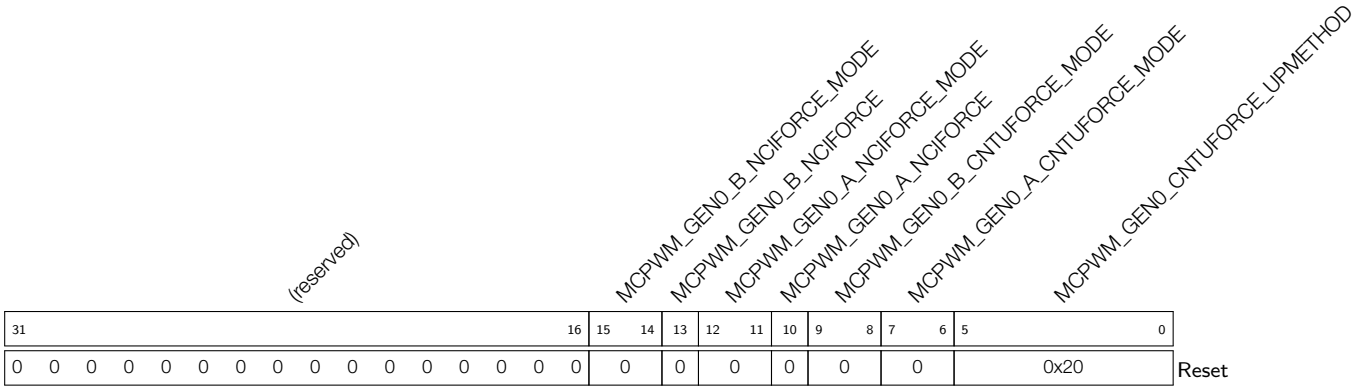
2：fault_event2

3：sync_taken

4：无

(R/W)

Register 33.20. MCPWM_GEN0_FORCE_REG (0x004C)



MCPWM_GEN0_CNTUFORCE_UPMETHOD 生成器 0 的连续软件强制事件更新方式。

所有 bit 为 0 时：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生 TEA 事件时更新

bit3 为 1：发生 TEB 事件时更新

bit4：发生同步事件时更新

bit5 为 1：关闭更新

本文档中，TEA/B 指定时器值为寄存器 A/B 的值时生成的事件。

(R/W)

MCPWM_GEN0_A_CNTUFORCE_MODE 设置 PWM0A 的连续软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN0_B_CNTUFORCE_MODE 设置 PWM0B 的连续软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

MCPWM_GEN0_A_NCIFORCE 该位的值取反时将触发 PWM0A 上的非连续即时软件强制事件。(R/W)

MCPWM_GEN0_A_NCIFORCE_MODE 设置用于 PWM0A 的非连续即时软件强制模式。

0：关闭

1：低电平

2：高电平

3：关闭

(R/W)

见下页...

Register 33.20. MCPWM_GEN0_FORCE_REG (0x0034)

... 接上页

MCPWM_GEN0_B_NCIFORCE 该位的值取反时将触发 PWM0B 上的非连续即时软件强制事件。(R/W)

MCPWM_GEN0_B_NCIFORCE_MODE 设置用于 PWM0B 的非连续即时软件强制模式。

0：关闭

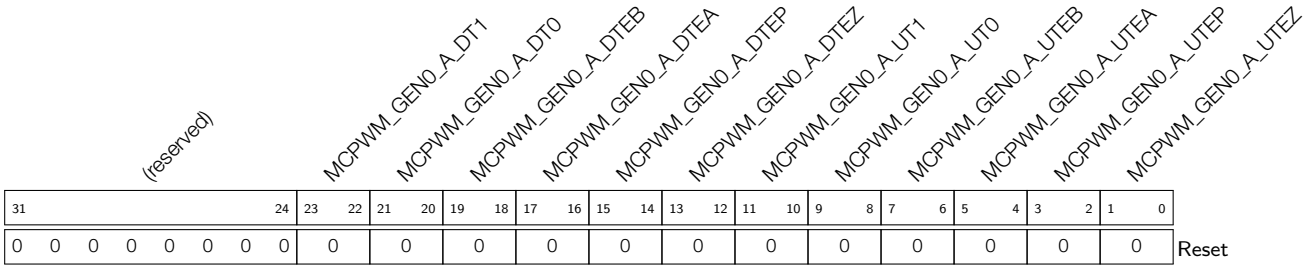
1：低电平

2：高电平

3：关闭

(R/W)

Register 33.21. MCPWM_GEN0_A_REG (0x0050)



MCPWM_GEN0_A_UTEZ 定时器递增时，TEZ 事件在 PWM0A 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN0_A_UTEP 定时器递增时，TEP 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_UTEA 定时器递增时，TEA 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_UTEB 定时器递增时，TEB 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_UT0 定时器递增时，event_t0 在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_UT1 定时器递增时，event_t1 在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_DTEZ 定时器递减时，TEZ 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_DTEP 定时器递减时，TEP 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

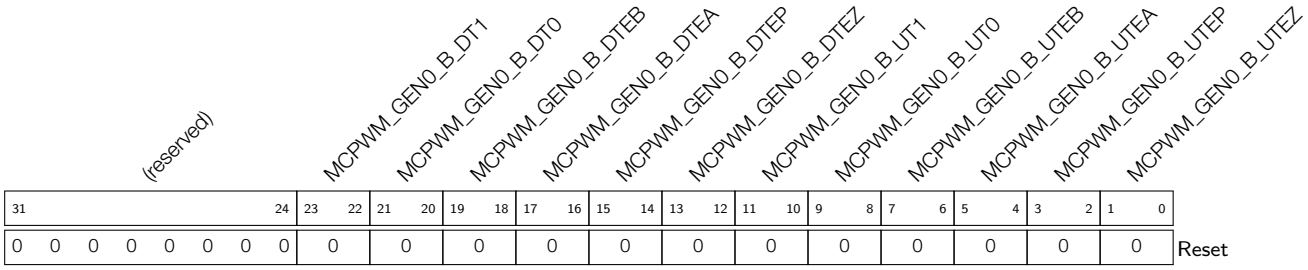
MCPWM_GEN0_A_DTEA 定时器递减时，TEA 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_DTEB 定时器递减时，TEB 事件在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_DT0 定时器递减时，event_t0 在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

MCPWM_GEN0_A_DT1 定时器递减时，event_t1 在 PWM0A 上触发的操作。具体配置值请参考 MCPWM_GEN0_A_UTEZ。(R/W)

Register 33.22. MCPWM_GEN0_B_REG (0x0054)



MCPWM_GEN0_B_UTEZ 定时器递增时，TEZ 在 PWM0B 上触发的操作。

0：波形无改变

1：拉低

2：拉高

3：取反

(R/W)

MCPWM_GEN0_B_UTEP 定时器递增时，TEP 在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_UTEA 定时器递增时，TEA 在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_UTEB 定时器递增时，TEB 在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_UT0 定时器递增时，event_t0 在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_UT1 定时器递增时，event_t1 在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_DTEZ 定时器递减时，TEZ 事件在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_DTEP 定时器递减时，TEP 事件在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

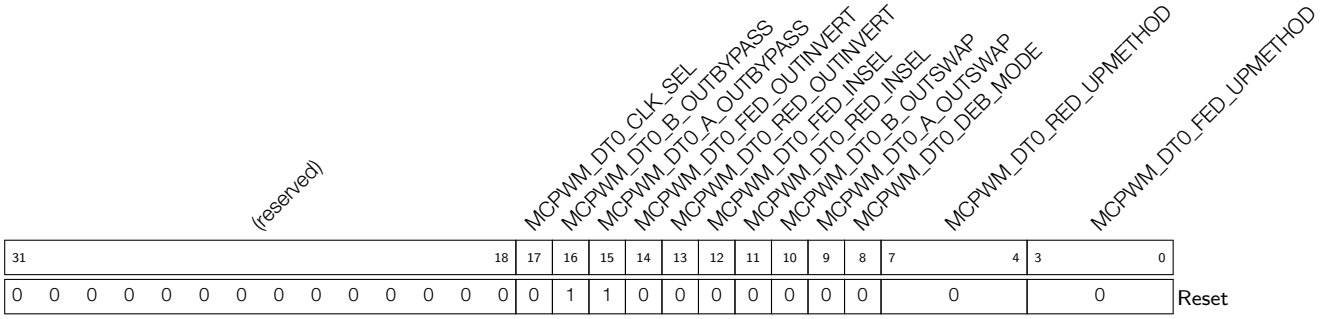
MCPWM_GEN0_B_DTEA 定时器递减时，TEA 事件在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_DTEB 定时器递减时，TEB 事件在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_DT0 定时器递减时，event_t0 事件在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

MCPWM_GEN0_B_DT1 定时器递减时，event_t1 在 PWM0B 上触发的操作。具体配置值请参考 MCPWM_GEN0_B_UTEZ。(R/W)

Register 33.23. MCPWM_DT0_CFG_REG (0x0058)



MCPWM_DT0_FED_UPMETHOD 下降沿延迟有效寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DT0_RED_UPMETHOD 上升沿延迟有效寄存器的更新方式。具体配置值请参考 MCPWM_DT0_FED_UPMETHOD。(R/W)

MCPWM_DT0_DEB_MODE 配置表 33-5中的 S8 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_A_OUTSWAP 配置表 33-5 中的 S6 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_B_OUTSWAP 配置表 33-5 中的 S7 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_RED_INSEL 配置表 33-5 中的 S4 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_FED_INSEL 配置表 33-5 中的 S5 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_RED_OUTINVERT 配置表 33-5 中的 S2 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_FED_OUTINVERT 配置表 33-5 中的 S3 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_A_OUTBYPASS 配置表 33-5 中的 S1 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT0_B_OUTBYPASS 配置表 33-5 中的 S0 开关，详细配置信息请参考表 33-6。(R/W)

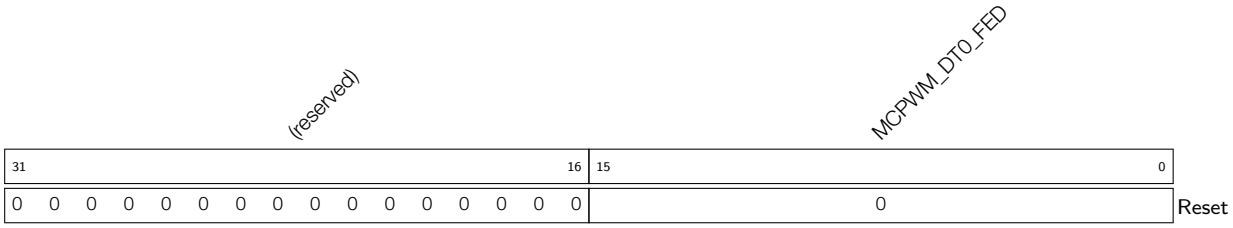
MCPWM_DT0_CLK_SEL 选择死区时间生成器 0 的时钟。

0：PWM_CLK

1：PT_CLK

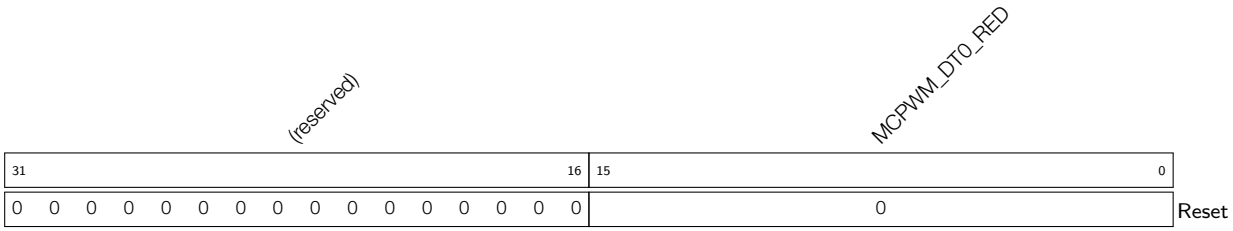
(R/W)

Register 33.24. MCPWM_DT0_FED_CFG_REG (0x005C)



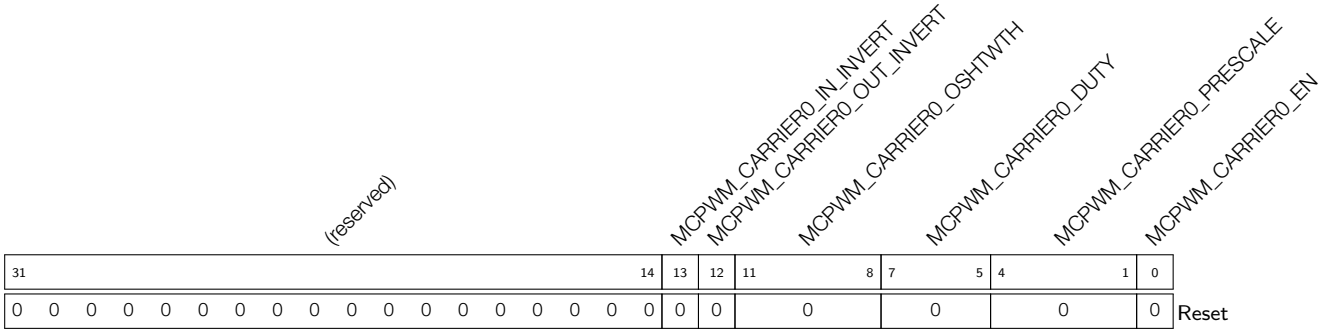
MCPWM_DT0_FED FED 的影子寄存器。(R/W)

Register 33.25. MCPWM_DT0_RED_CFG_REG (0x0060)



MCPWM_DT0_RED RED 的影子寄存器。(R/W)

Register 33.26. MCPWM_CARRIER0_CFG_REG (0x0064)



MCPWM_CARRIER0_EN 置 1 时，使能载波 0 的功能。清零时，载波 0 被绕过。(R/W)

MCPWM_CARRIER0_PRESCALE PWM 载波 0 时钟 (PC_CLK) 的预分频值。PC_CLK 周期 = PWM_CLK 周期 * (PWM_CARRIER0_PRESCALE + 1)。(R/W)

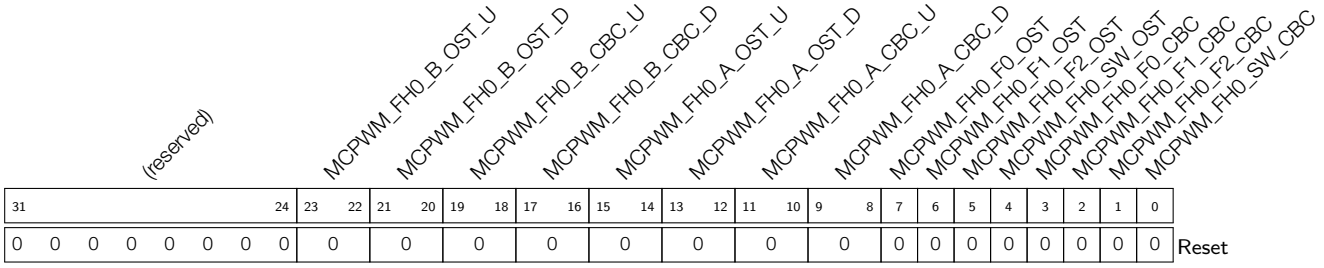
MCPWM_CARRIER0_DUTY 选择载波占空比。占空比 = PWM_CARRIER0_DUTY/8。(R/W)

MCPWM_CARRIER0_OSHTWTH 载波第一个脉冲的宽度，单位为载波周期。(R/W)

MCPWM_CARRIER0_OUT_INVERT 置 1 时，将此模块的 PWM0A 和 PWM0B 输出反相。(R/W)

MCPWM_CARRIER0_IN_INVERT 置 1 时，将此模块的 PWM0A 和 PWM0B 输入反相。(R/W)

Register 33.27. MCPWM_FH0_CFG0_REG (0x0068)



MCPWM_FH0_SW_CBC 使能软件强制逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH0_F2_CBC 设置 fault_event2 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH0_F1_CBC 设置 fault_event1 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH0_F0_CBC 设置 fault_event0 是否触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH0_SW_OST 软件强制一次性模式操作的使能寄存器。

0：关闭

1：使能

(R/W)

MCPWM_FH0_F2_OST 设置 fault_event2 是否触发一次性模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH0_F1_OST 设置 fault_event1 是否触发一次性模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH0_F0_OST 设置 fault_event0 是否触发一次性模式操作。

0：关闭

1：使能

(R/W)

见下页...

Register 33.27. MCPWM_FH0_CFG0_REG (0x0034)

... 接上页

MCPWM_FH0_A_CBC_D 定时器递减计数并且发生故障事件时，PWM0A 上的逐周期模式操作。

0：无

1：强制拉低

2：强制拉高

3：取反

(R/W)

MCPWM_FH0_A_CBC_U 定时器递增计数并且发生故障事件时，PWM0A 上的逐周期模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

MCPWM_FH0_A_OST_D 定时器递减计数并且发生故障事件时，PWM0A 上的一次性模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

MCPWM_FH0_A_OST_U 定时器递增计数并且发生故障事件时，PWM0A 上的一次性模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

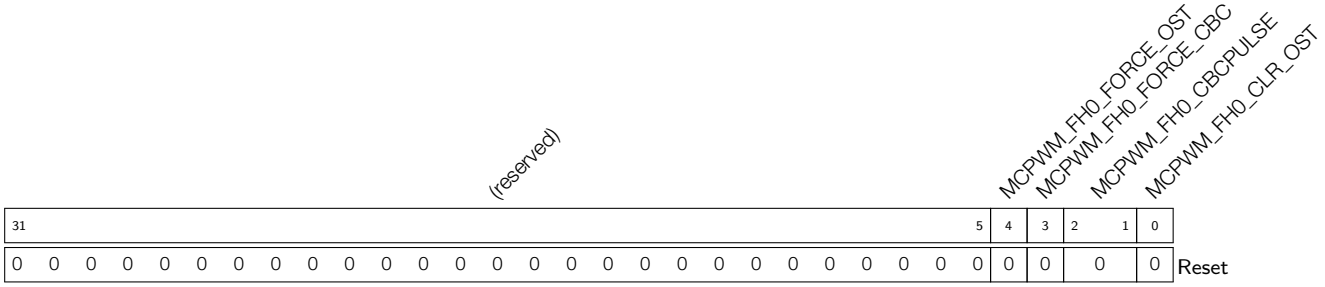
MCPWM_FH0_B_CBC_D 定时器递减计数并且发生故障事件时，PWM0B 上的逐周期模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

MCPWM_FH0_B_CBC_U 定时器递增计数并且发生故障事件时，PWM0B 上的逐周期模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

MCPWM_FH0_B_OST_D 定时器递减计数并且发生故障事件时，PWM0B 上的一次性模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

MCPWM_FH0_B_OST_U 定时器递增计数并且发生故障事件时，PWM0B 上的一次性模式操作。具体配置值请参考 MCPWM_FH0_A_CBC_D。(R/W)

Register 33.28. MCPWM_FH0_CFG1_REG (0x006C)



MCPWM_FH0_CLR_OST 上升沿时将清除持续一次性模式操作。(R/W)

MCPWM_FH0_CBCPULSE 设置逐周期模式操作更新的时间点。

bit0 为 1：发生 TEZ 事件时

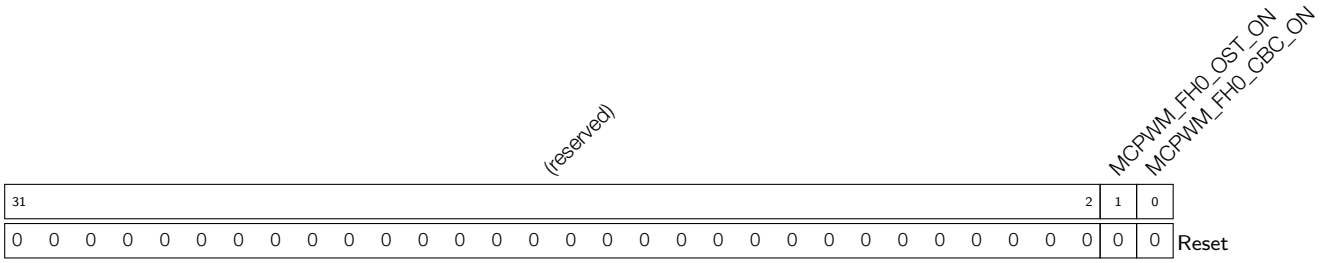
bit1 为 1：发生 TEP 事件时

(R/W)

MCPWM_FH0_FORCE_CBC 通过软件将此位的值取反，可触发逐周期模式的操作。(R/W)

MCPWM_FH0_FORCE_OST 通过软件将此位的值取反，可触发一次性模式的操作。(R/W)

Register 33.29. MCPWM_FH0_STATUS_REG (0x0070)



MCPWM_FH0_CBC_ON 表示是否正在进行逐周期模式的操作。此字段由硬件置 1 和清零。

0：不存在正在进行的逐周期模式操作

1：逐周期模式的操作正在进行

(RO)

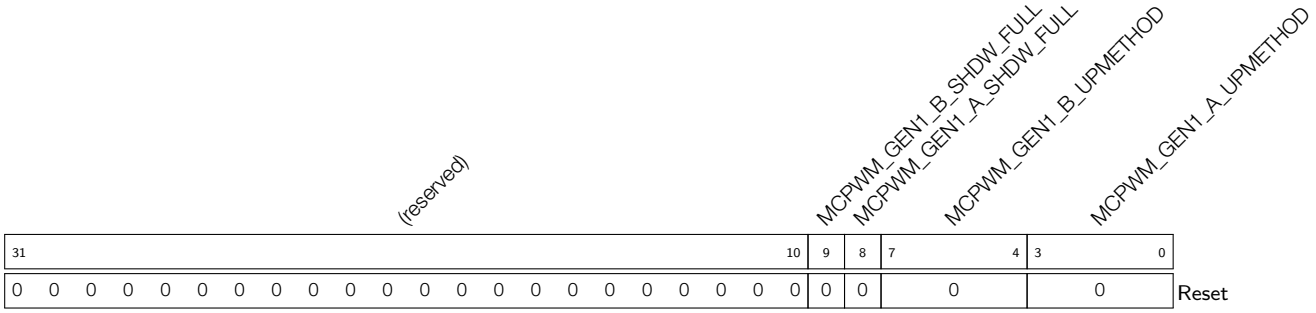
MCPWM_FH0_OST_ON 表示是否正在进行一次性模式的操作。此字段由硬件置 1 和清零。

0：不存在正在进行的一次性模式操作

1：一次性模式的操作正在进行

(RO)

Register 33.30. MCPWM_GEN1_STMP_CFG_REG (0x0074)



MCPWM_GEN1_A_UPMETHOD PWM 生成器 1 时间戳寄存器 A 有效寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN1_B_UPMETHOD PWM 生成器 1 时间戳寄存器 B 有效寄存器的更新方式。具体配置值请参考 MCPWM_GEN1_A_UPMETHOD。(R/W)

MCPWM_GEN1_A_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。此字段由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 A 中

1：将值写入 PWM 生成器 1 时间戳寄存器 A 的影子寄存器中，等待传输给有效寄存器 A

(R/SC/WTC)

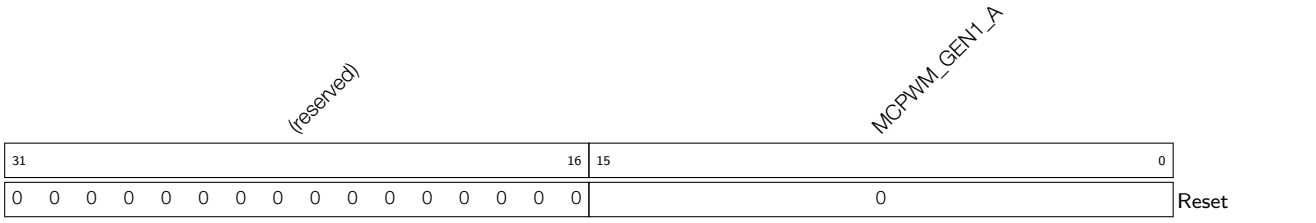
MCPWM_GEN1_B_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。此字段由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 B 中

1：将值写入 PWM 生成器 1 时间戳寄存器 B 的影子寄存器中，等待传输给有效寄存器 A

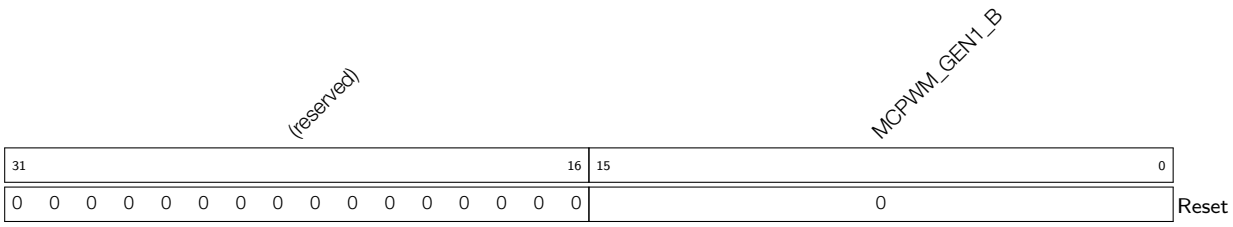
(R/SC/WTC)

Register 33.31. MCPWM_GEN1_TSTMP_A_REG (0x0078)



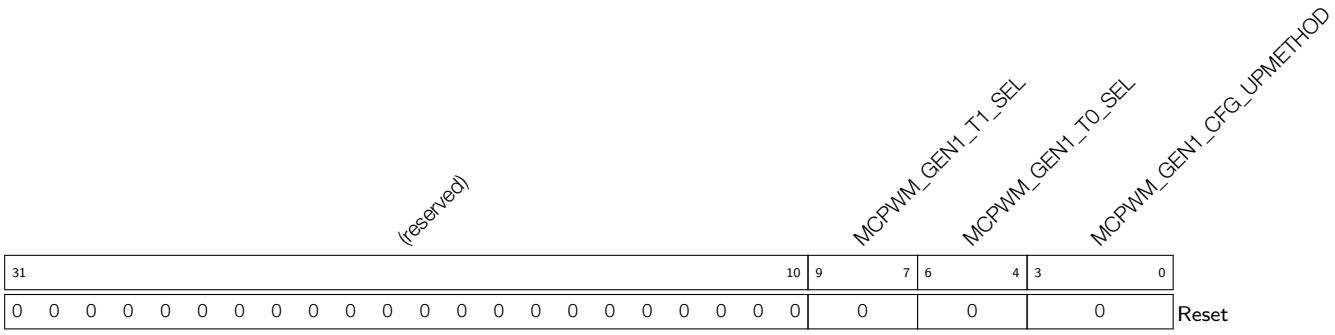
MCPWM_GEN1_A PWM 生成器 1 时间戳寄存器 A 的影子寄存器。(R/W)

Register 33.32. MCPWM_GEN1_TSTMP_B_REG (0x007C)



MCPWM_GEN1_B PWM 生成器 1 时间戳寄存器 B 的影子寄存器。(R/W)

Register 33.33. MCPWM_GEN1_CFG0_REG (0x0080)



MCPWM_GEN1_CFG_UPMETHOD 配置 PWM 生成器 1 有效寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN1_T0_SEL 配置 PWM 生成器 1 event_t0 的信号源，立即生效

0：fault_event0

1：fault_event1

2：fault_event2

3：sync_taken

4：无

(R/W)

MCPWM_GEN1_T1_SEL 配置 PWM 生成器 1 event_t1 的信号源，立即生效。

0：fault_event0

1：fault_event1

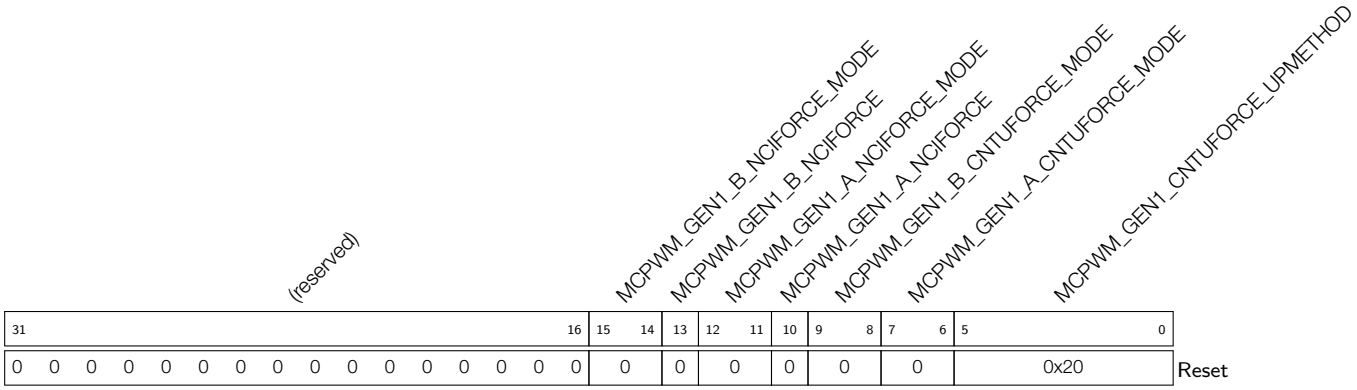
2：fault_event2

3：sync_taken

4：无

(R/W)

Register 33.34. MCPWM_GEN1_FORCE_REG (0x0084)



MCPWM_GEN1_CNTUFORCE_UPMETHOD PWM 生成器 1 持续软件强制的更新方式。

0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生 TEA 事件时更新

bit3 为 1：发生 TEB 事件时更新

bit4 为 1：发生同步时间时更新

bit5 为 1：关闭更新

本文档中的 TEA/B 表示定时器值等于寄存器 A/B 生成的事件。

(R/W)

MCPWM_GEN1_A_CNTUFORCE_MODE 用于 PWM1A 的连续软件强制事件。

0：关闭

1：拉低

2：拉高

3：关闭

(R/W)

MCPWM_GEN1_B_CNTUFORCE_MODE 用于 PWM1B 的连续软件强制事件。

0：关闭

1：拉低

2：拉高

3：关闭

(R/W)

MCPWM_GEN1_A_NCIFORCE 该位的值取反时将触发 PWM1A 上的非连续即时软件强制事件。(R/W)

MCPWM_GEN1_A_NCIFORCE_MODE 用于 PWM1A 的非持续性即时软件强制事件。

0：关闭

1：拉低

2：拉高

3：关闭

(R/W)

见下页...

Register 33.34. MCPWM_GEN1_FORCE_REG (0x0034)

... 接上页

MCPWM_GEN1_B_NCIFORCE 该位的值取反时将触发 PWM1B 上的非连续即时软件强制事件。(R/W)

MCPWM_GEN1_B_NCIFORCE_MODE 用于 PWM1B 的非持续性即时软件强制事件。

0：关闭

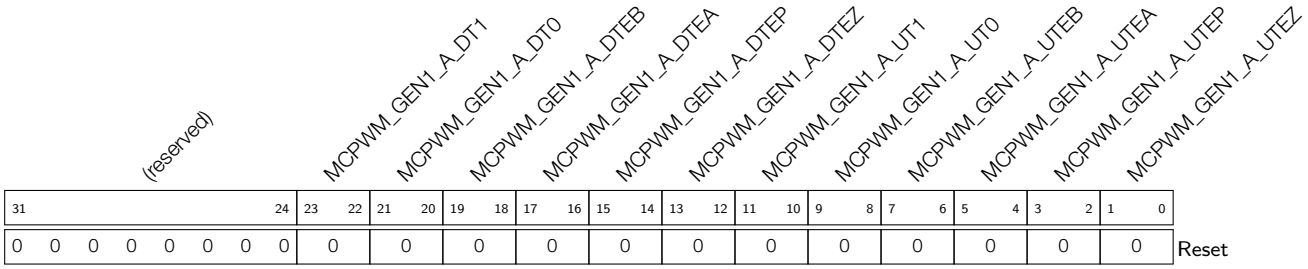
1：拉低

2：拉高

3：关闭

(R/W)

Register 33.35. MCPWM_GEN1_A_REG (0x0088)



MCPWM_GEN1_A_UTEZ 定时器递增计数时，TEZ 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_UTEP 定时器递增计数时，TEP 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_UTEA 定时器递增计数时，TEA 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_UTEB 定时器递增计数时，TEB 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_UT0 定时器递增计数时，event_t0 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 33.35. MCPWM_GEN1_A_REG (0x0034)

... 接上页

MCPWM_GEN1_A_UT1 定时器递增计数时，event_t1 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_DTEZ 定时器递减计数时，TEZ 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_DTEP 定时器递减计数时，TEP 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_DTEA 定时器递减计数时，TEA 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_DTEB 定时器递减计数时，TEB 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

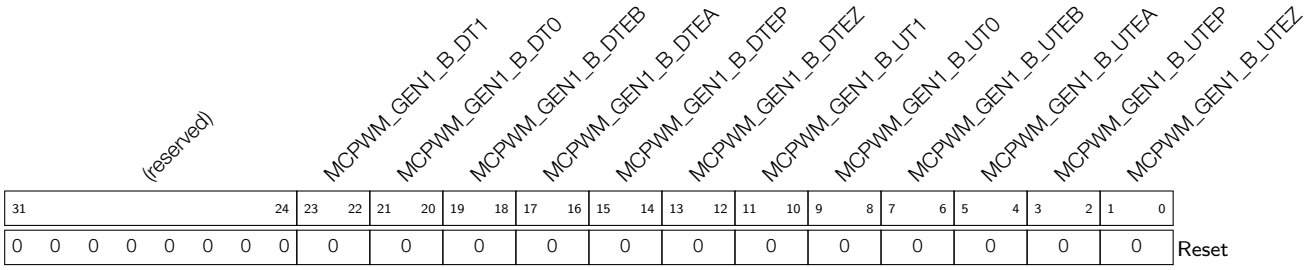
MCPWM_GEN1_A_DT0 定时器递减计数时，event_t0 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_A_DT1 定时器递减计数时，event_t1 在 PWM1A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

Register 33.36. MCPWM_GEN1_B_REG (0x008C)



MCPWM_GEN1_B_UTEZ 定时器递增计数时，TEZ 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_UTEP 定时器递增计数时，TEP 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_UTEA 定时器递增计数时，TEA 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_UTEB 定时器递增计数时，TEB 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_UT0 定时器递增计数时，event_t0 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 33.36. MCPWM_GEN1_B_REG (0x0034)

... 接上页

MCPWM_GEN1_B_UT1 定时器递增计数时，event_t1 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_DTEZ 定时器递减计数时，TEZ 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_DTEP 定时器递减计数时，TEP 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_DTEA 定时器递减计数时，TEA 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_DTEB 定时器递减计数时，TEB 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

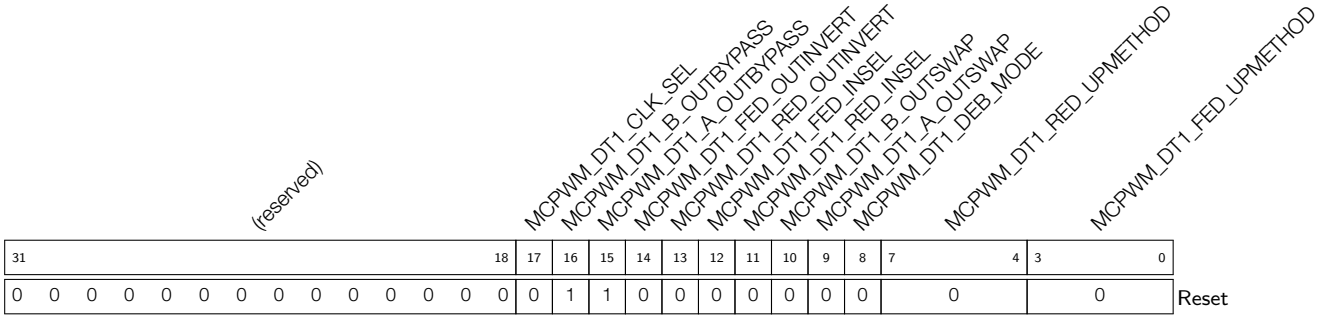
MCPWM_GEN1_B_DT0 定时器递减计数时，event_t0 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN1_B_DT1 定时器递减计数时，event_t1 在 PWM1B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

Register 33.37. MCPWM_DT1_CFG_REG (0x0090)



MCPWM_DT1_FED_UPMETHOD FED(下降沿延迟) 有效寄存器的更新方式。

0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步事件时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DT1_RED_UPMETHOD RED(上升沿延迟) 有效寄存器的更新方式。

0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步事件时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DT1_DEB_MODE 配置表 33-5中的 S8 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_A_OUTSWAP 配置表 33-5 中的 S6 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_B_OUTSWAP 配置表 33-5 中的 S7 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_RED_INSEL 配置表 33-5 中的 S4 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_FED_INSEL 配置表 33-5 中的 S5 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_RED_OUTINVERT 配置表 33-5 中的 S2 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_FED_OUTINVERT 配置表 33-5 中的 S3 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_A_OUTBYPASS 配置表 33-5 中的 S1 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT1_B_OUTBYPASS 配置表 33-5 中的 S0 开关，详细配置信息请参考表 33-6。(R/W)

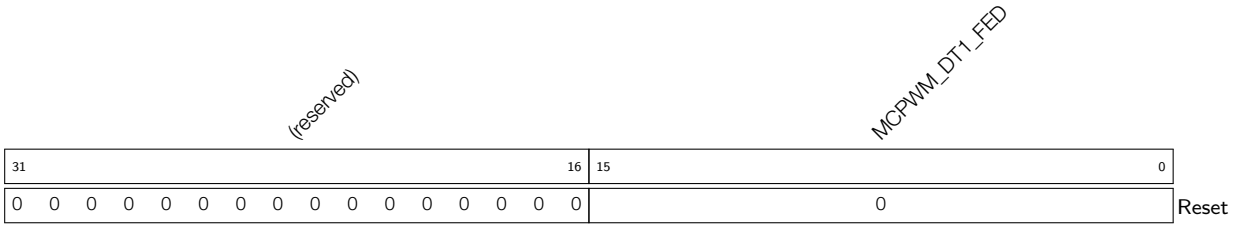
MCPWM_DT1_CLK_SEL 设置死区时间生成器时钟。

0：PWM_CLK

1：PT_CLK

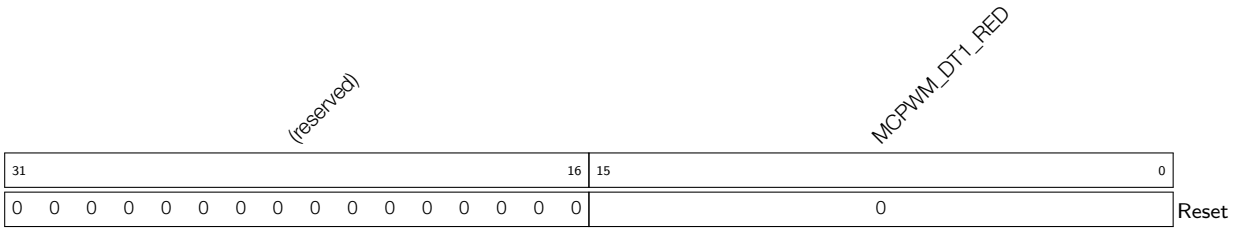
(R/W)

Register 33.38. MCPWM_DT1_FED_CFG_REG (0x0094)



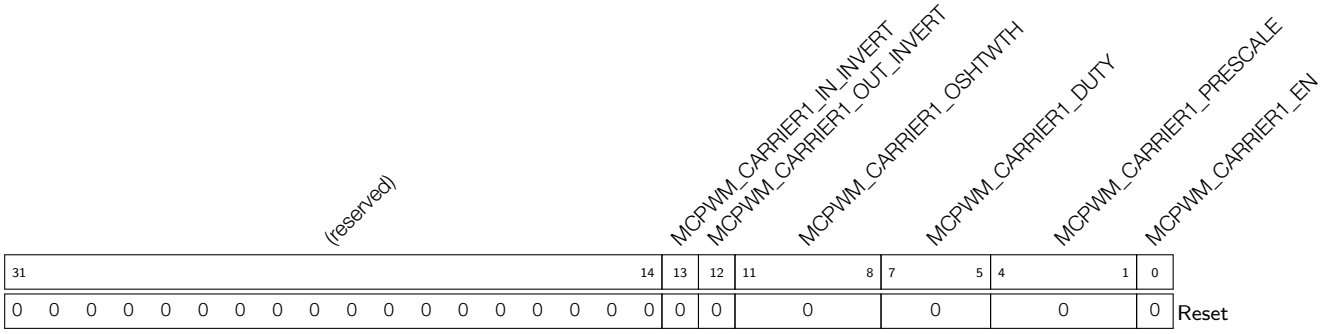
MCPWM_DT1_FED FED 影子寄存器。(R/W)

Register 33.39. MCPWM_DT1_RED_CFG_REG (0x0098)



MCPWM_DT1_RED RED 影子寄存器。(R/W)

Register 33.40. MCPWM_CARRIER1_CFG_REG (0x009C)



MCPWM_CARRIER1_EN 置 1 时，使能载波 1 功能。此位清零时，绕过载波 1。(R/W)

MCPWM_CARRIER1_PRESCALE 配置 PWM 载波 1 时钟 (PC_CLK) 预分频值。PC_CLK 周期 = PWM_CLK 周期 * (PWM_CARRIER0_PRESCALE + 1)。(R/W)

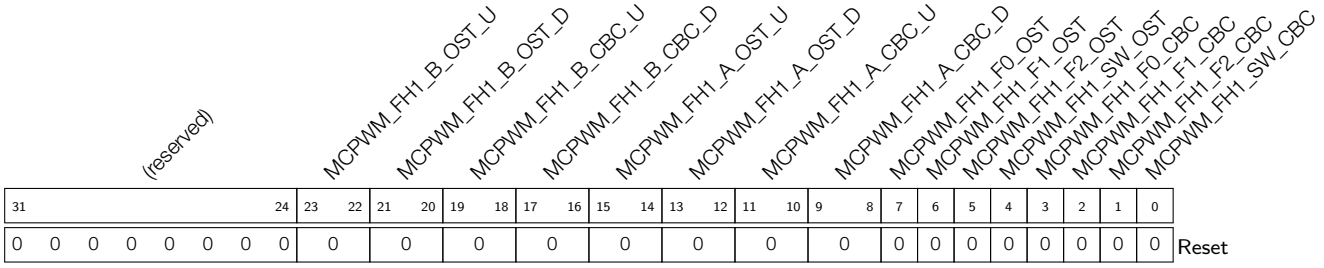
MCPWM_CARRIER1_DUTY 设置载波占空比。占空比 = PWM_CARRIER0_DUTY / 8。(R/W)

MCPWM_CARRIER1_OSHTWTH 配置载波第一个脉冲的宽度，单位为载波周期。(R/W)

MCPWM_CARRIER1_OUT_INVERT 置 1 时，将此模块的 PWM1A 和 PWM1B 输出反相。(R/W)

MCPWM_CARRIER1_IN_INVERT 置 1 时，将此模块的 PWM1A 和 PWM1B 输入反相。(R/W)

Register 33.41. MCPWM_FH1_CFG0_REG (0x00A0)



MCPWM_FH1_SW_CBC 软件强制逐周期模式操作的使能寄存器。

0：关闭

1：使能

(R/W)

MCPWM_FH1_F2_CBC 设置 fault_event2 触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH1_F1_CBC 设置 fault_event1 触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH1_F0_CBC 设置 fault_event0 触发逐周期模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH1_SW_OST 软件强制一次性模式操作的使能寄存器。

0：关闭

1：使能

(R/W)

MCPWM_FH1_F2_OST 设置 fault_event2 触发一次性模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH1_F1_OST 设置 fault_event1 触发一次性模式操作。

0：关闭

1：使能

(R/W)

MCPWM_FH1_F0_OST 设置 fault_event0 触发一次性模式操作。

0：关闭

1：使能

(R/W)

见下页...

Register 33.41. MCPWM_FH1_CFG0_REG (0x0034)

... 接上页

MCPWM_FH1_A_CBC_D 定时器递减计数并且发生故障事件时，PWM1A 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_FH1_A_CBC_U 定时器递增计数并且发生故障事件时，PWM1A 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_FH1_A_OST_D 定时器递减计数并且发生故障事件时，PWM1A 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_FH1_A_OST_U 定时器递增计数并且发生故障事件时，PWM1A 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_FH1_B_CBC_D 定时器递减计数并且发生故障事件时，PWM1B 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_FH1_B_CBC_U 定时器递增计数并且发生故障事件时，PWM1B 上的逐周期模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

见下页...

Register 33.41. MCPWM_FH1_CFG0_REG (0x0034)

... 接上页

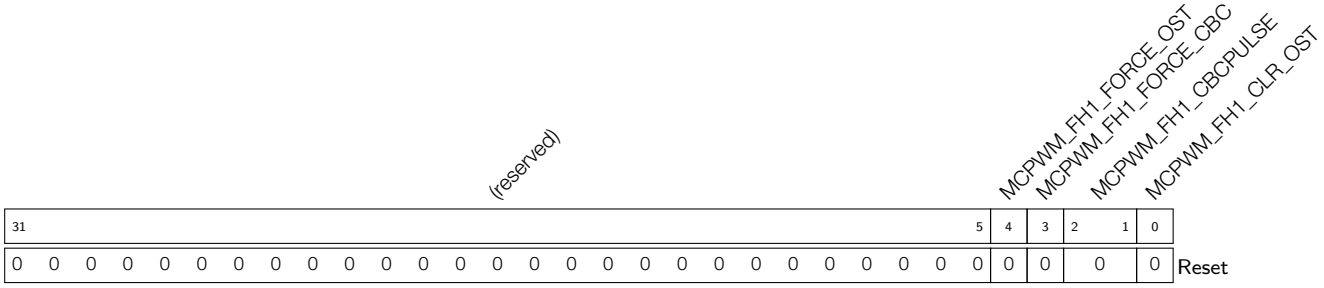
MCPWM_FH1_B_OST_D 定时器递减计数并且发生故障事件时，PWM1B 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

MCPWM_FH1_B_OST_U 定时器递增计数并且发生故障事件时，PWM1B 上的一次性模式操作。

0：无
1：强制拉低
2：强制拉高
3：取反
(R/W)

Register 33.42. MCPWM_FH1_CFG1_REG (0x00A4)



MCPWM_FH1_CLR_OST 置 1 清除正在进行的一次性模式操作。(R/W)

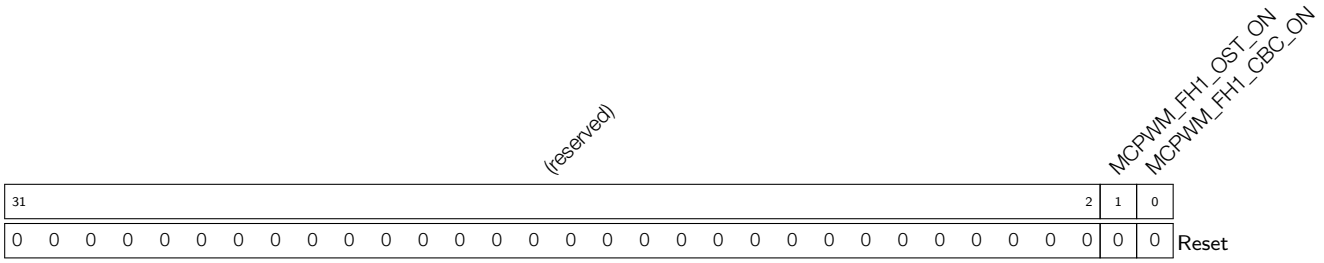
MCPWM_FH1_CBCPULSE 设置逐周期模式操作的更新方式。

bit0 为 1：发生 TEZ 事件时更新
bit1 为 1：发生 TEP 事件时更新
(R/W)

MCPWM_FH1_FORCE_CBC 通过软件将该位取反时，触发逐周期模式操作。(R/W)

MCPWM_FH1_FORCE_OST 通过软件将该位取反时，触发一次性模式操作。(R/W)

Register 33.43. MCPWM_FH1_STATUS_REG (0x00A8)



MCPWM_FH1_CBC_ON 表示是否正在进行逐周期模式的操作。此字段由硬件置 1 和清零。

0：不存在正在进行的逐周期模式操作

1：逐周期模式的操作正在进行

(RO)

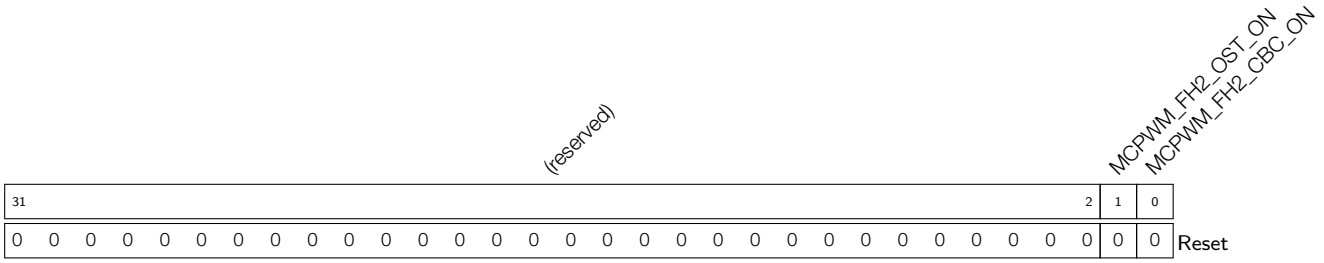
MCPWM_FH1_OST_ON 表示是否正在进行一次性模式的操作。此字段由硬件置 1 和清零。

0：不存在正在进行的一次性模式操作

1：一次性模式的操作正在进行

(RO)

Register 33.44. MCPWM_FH2_STATUS_REG (0x00E0)



MCPWM_FH2_CBC_ON 表示是否正在进行逐周期模式的操作。此字段由硬件置 1 和清零。

0：不存在正在进行的逐周期模式操作

1：逐周期模式的操作正在进行

(RO)

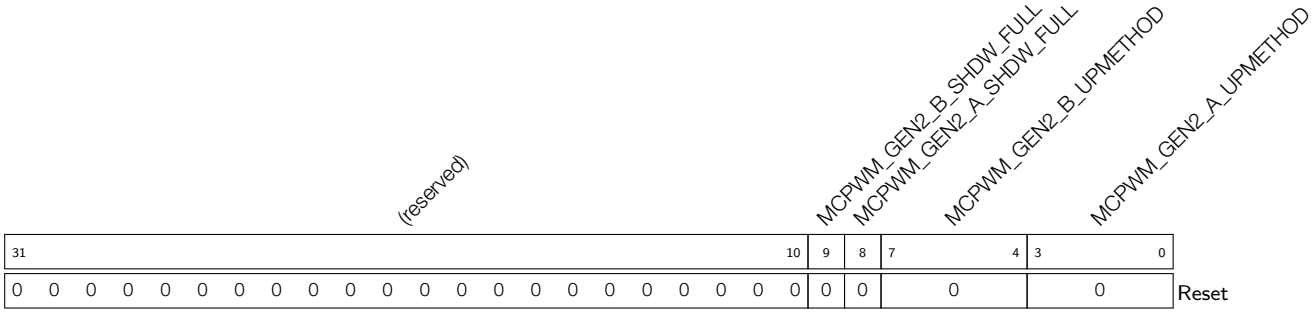
MCPWM_FH2_OST_ON 表示是否正在进行一次性模式的操作。此字段由硬件置 1 和清零。

0：不存在正在进行的一次性模式操作

1：一次性模式的操作正在进行

(RO)

Register 33.45. MCPWM_GEN2_STMP_CFG_REG (0x00AC)



MCPWM_GEN2_A_UPMETHOD 生成器 2 时间戳寄存器 A 有效寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN2_B_UPMETHOD 生成器 2 时间戳寄存器 B 有效寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN2_A_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。此字段由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 A 中

1：将值写入 PWM 生成器 2 时间戳寄存器 A 的影子寄存器中，等待传输给有效寄存器 A。

(R/SC/WTC)

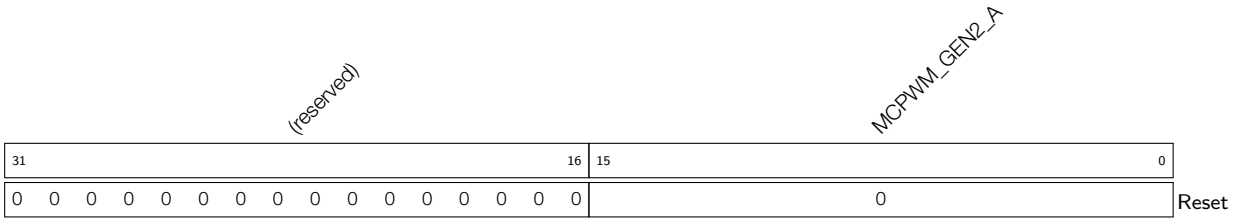
MCPWM_GEN2_B_SHDW_FULL 配置是否在某个特定时间将影子寄存器中的值写入对应的有效寄存器中。此字段由硬件置 1 或复位。

0：将影子寄存器中最新的值写入有效寄存器 B 中

1：将值写入 PWM 生成器 2 时间戳寄存器 B 的影子寄存器中，等待传输给有效寄存器 B。

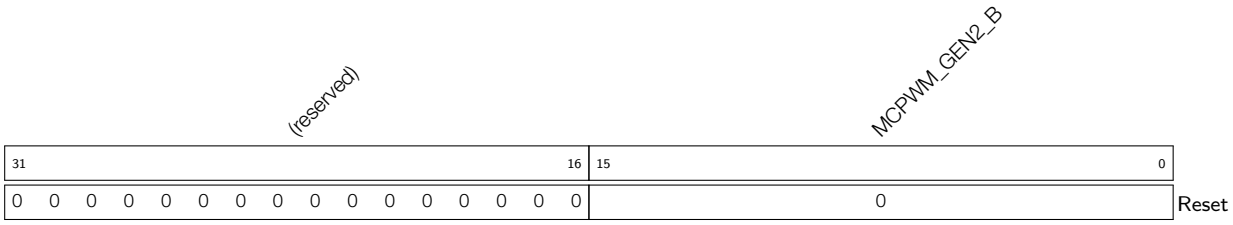
(R/SC/WTC)

Register 33.46. MCPWM_GEN2_STMP_A_REG (0x00B0)



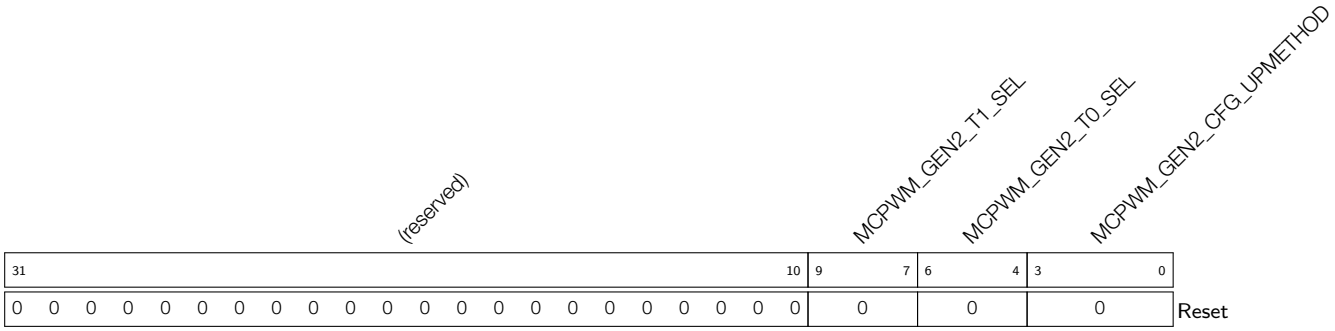
MCPWM_GEN2_A PWM 生成器 2 时间戳 A 的影子寄存器。(R/W)

Register 33.47. MCPWM_GEN2_STMP_B_REG (0x00B4)



MCPWM_GEN2_B PWM 生成器 2 时间戳 B 的影子寄存器。(R/W)

Register 33.48. MCPWM_GEN2_CFG0_REG (0x00B8)



MCPWM_GEN2_CFG_UPMETHOD PWM 生成器 2 有效配置寄存器的更新方式。

所有 bit 值为 0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步时间时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_GEN2_T0_SEL 设置 PWM 操作器 2 event_t0 信号源，立即生效。

0：fault_event0

1：fault_event1

2：fault_event2

3：sync_taken

4：无

(R/W)

MCPWM_GEN2_T1_SEL 设置 PWM 操作器 2 event_t1 信号源，立即生效。

0：fault_event0

1：fault_event1

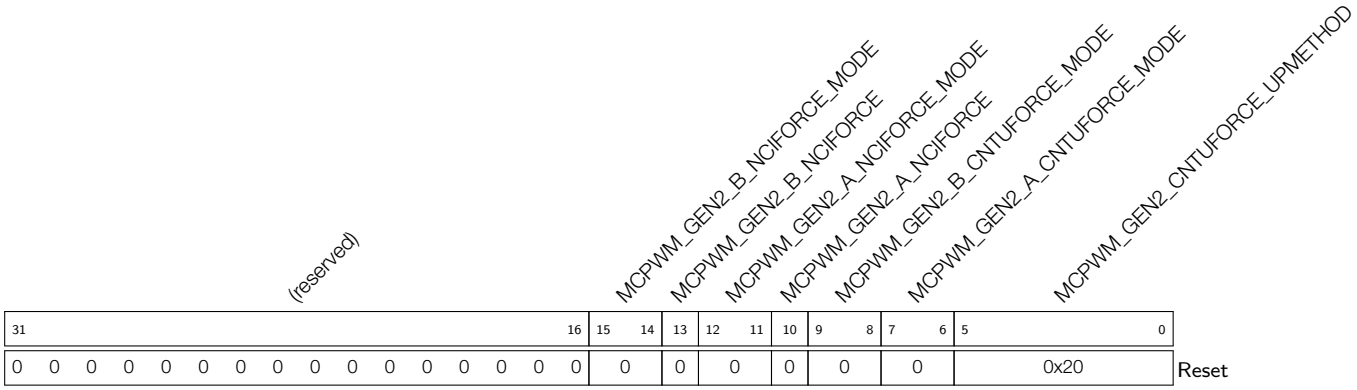
2：fault_event2

3：sync_taken

4：无

(R/W)

Register 33.49. MCPWM_GEN2_FORCE_REG (0x00BC)



MCPWM_GEN2_CNTUFORCE_UPMETHOD 配置 PWM 生成器 2 的持续性软件强制事件的更新方式。

0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生 TEA 事件时更新

bit3 为 1：发生 TEB 事件时更新

bit4 为 1：发生同步时间时更新

bit5 为 1：关闭更新

本文档中的 TEA/B 表示定时器值等于寄存器 A/B 生成的事件

(R/W)

MCPWM_GEN2_A_CNTUFORCE_MODE 配置 PWM2A 的持续性即时软件强制事件。

0：关闭

1：拉低

2：拉高

3：关闭

(R/W)

MCPWM_GEN2_B_CNTUFORCE_MODE 配置 PWM2B 的持续性即时软件强制事件。

0：关闭

1：拉低

2：拉高

3：关闭

(R/W)

MCPWM_GEN2_A_NCIFORCE 配置是否触发 PWM2A 上的非连续即时软件强制事件。

0：无效

1：触发 PWM2A 上的非连续即时软件强制事件

(R/W)

见下页...

Register 33.49. MCPWM_GEN2_FORCE_REG (0x0034)

... 接上页

MCPWM_GEN2_A_NCIFORCE_MODE 配置 PWM2A 的非持续性即时软件强制事件。

0：关闭
1：拉低
2：拉高
3：关闭
(R/W)

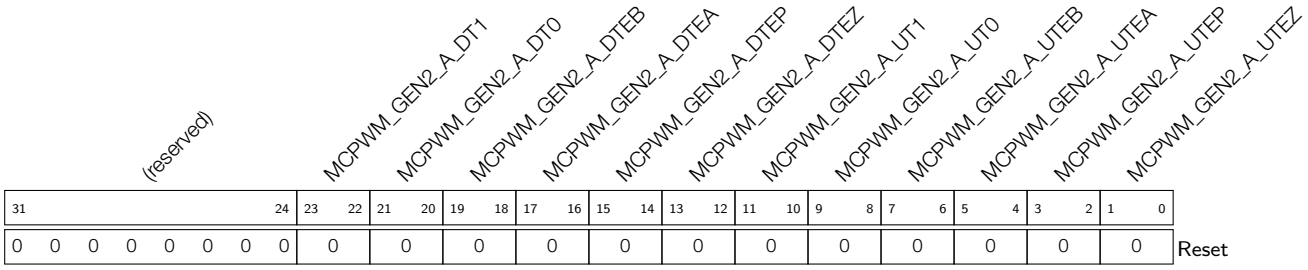
MCPWM_GEN2_B_NCIFORCE 配置是否触发 PWM2B 上的非连续即时软件强制事件。

0：无效
1：触发 PWM2B 上的非连续即时软件强制事件
(R/W)

MCPWM_GEN2_B_NCIFORCE_MODE 配置 PWM2B 的非持续性即时软件强制模式。

0：关闭
1：拉低
2：拉高
3：关闭
(R/W)

Register 33.50. MCPWM_GEN2_A_REG (0x00C0)



MCPWM_GEN2_A_UTEZ 配置定时器递增时，TEZ 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_UTEP 配置定时器递增时，TEP 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_UTEA 配置定时器递增时，TEA 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_UTEB 配置定时器递增时，TEB 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_UT0 配置定时器递增时，event_t0 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 33.50. MCPWM_GEN2_A_REG (0x0034)

... 接上页

MCPWM_GEN2_A_UT1 配置定时器递增时，event_t1 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_DTEZ 配置定时器递减时，TEZ 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_DTEP 配置定时器递减时，TEP 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_DTEA 配置定时器递减时，TEA 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_DTEB 配置定时器递减时，TEB 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

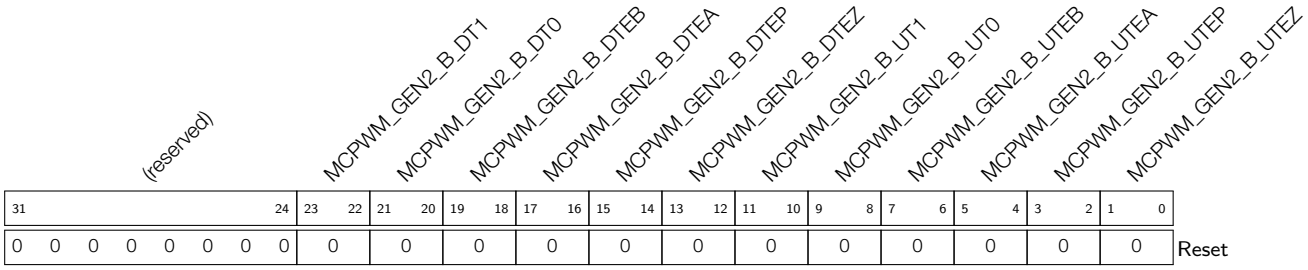
MCPWM_GEN2_A_DT0 配置定时器递减时，event_t0 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_A_DT1 配置定时器递减时，event_t1 在 PWM2A 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

Register 33.51. MCPWM_GEN2_B_REG (0x00C4)



MCPWM_GEN2_B_UTEZ 配置定时器递增时，TEZ 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_UTEP 配置定时器递增时，TEP 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_UTEA 配置定时器递增时，TEA 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_UTEB 配置定时器递增时，TEB 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_UT0 配置定时器递增时，event_t0 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_UT1 配置定时器递增时，event_t1 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

见下页...

Register 33.51. MCPWM_GEN2_B_REG (0x0034)

... 接上页

MCPWM_GEN2_B_DTEZ 配置定时器递减时，TEZ 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_DTEP 配置定时器递减时，TEP 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_DTEA 配置定时器递减时，TEA 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_DTEB 配置定时器递减时，TEB 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

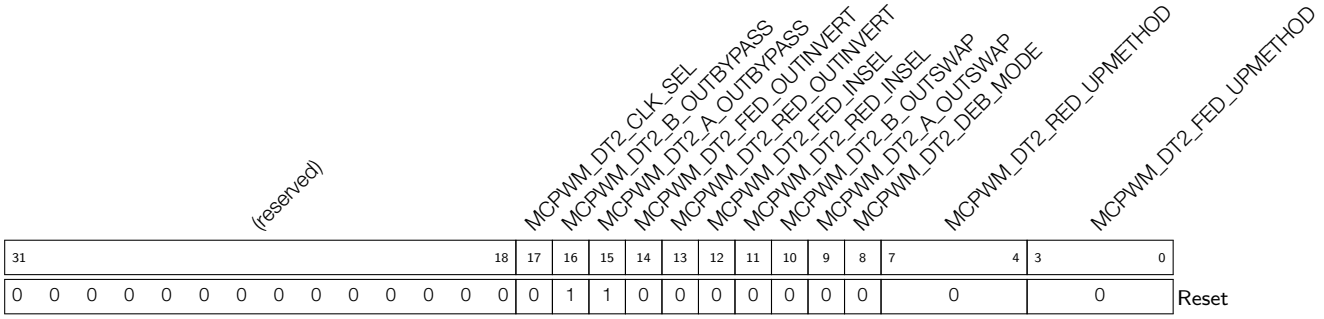
MCPWM_GEN2_B_DT0 配置定时器递减时，event_t0 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

MCPWM_GEN2_B_DT1 配置定时器递减时，event_t1 在 PWM2B 上触发的操作。

0：无
1：拉低
2：拉高
3：取反
(R/W)

Register 33.52. MCPWM_DT2_CFG_REG (0x00C8)



MCPWM_DT2_FED_UPMETHOD FED(下降沿延迟) 有效寄存器的更新方式。

0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步事件时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DT2_RED_UPMETHOD RED(上升沿延迟) 有效寄存器的更新方式。

0：立即更新

bit0 为 1：发生 TEZ 事件时更新

bit1 为 1：发生 TEP 事件时更新

bit2 为 1：发生同步事件时更新

bit3 为 1：关闭更新

(R/W)

MCPWM_DT2_DEB_MODE 配置表 33-5中的 S8 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_A_OUTSWAP 配置表 33-5 中的 S6 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_B_OUTSWAP 配置表 33-5 中的 S7 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_RED_INSEL 配置表 33-5 中的 S4 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_FED_INSEL 配置表 33-5 中的 S5 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_RED_OUTINVERT 配置表 33-5 中的 S2 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_FED_OUTINVERT 配置表 33-5 中的 S3 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_A_OUTBYPASS 配置表 33-5 中的 S1 开关，详细配置信息请参考表 33-6。(R/W)

MCPWM_DT2_B_OUTBYPASS 配置表 33-5 中的 S0 开关，详细配置信息请参考表 33-6。(R/W)

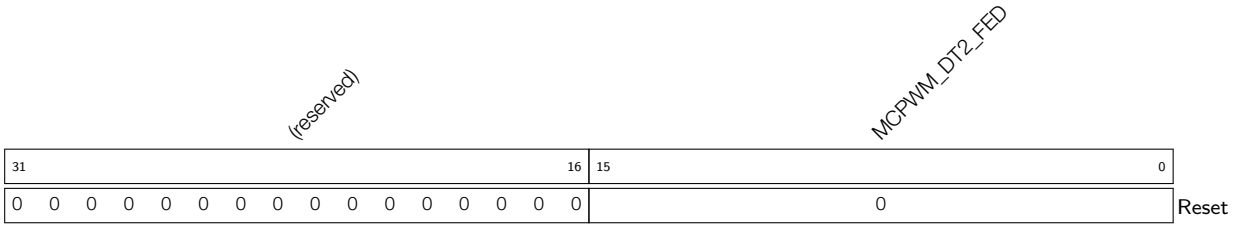
MCPWM_DT2_CLK_SEL 配置死区时间生成器时钟。

0：PWM_CLK

1：PT_CLK

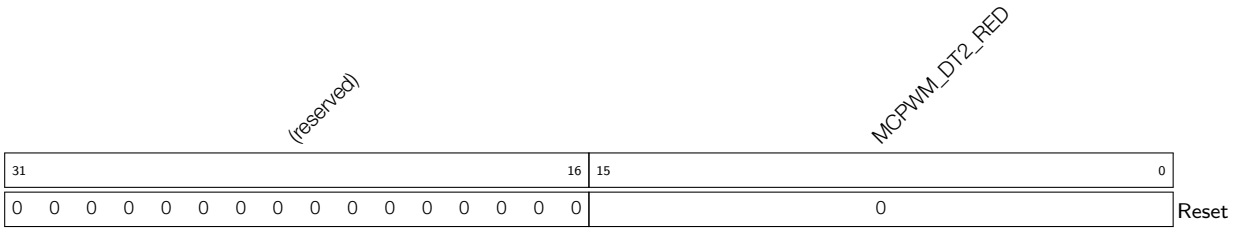
(R/W)

Register 33.53. MCPWM_DT2_FED_CFG_REG (0x00CC)



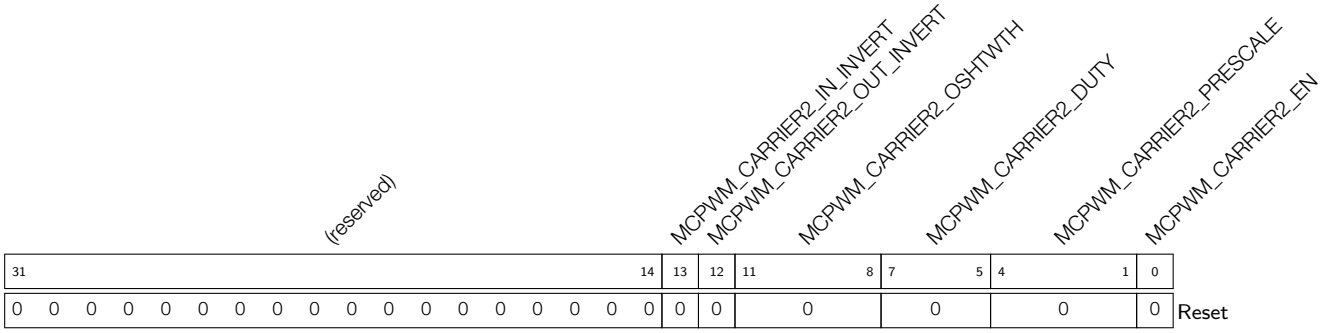
MCPWM_DT2_FED FED 影子寄存器。(R/W)

Register 33.54. MCPWM_DT2_RED_CFG_REG (0x00D0)



MCPWM_DT2_RED RED 影子寄存器。(R/W)

Register 33.55. MCPWM_CARRIER2_CFG_REG (0x00D4)



MCPWM_CARRIER2_EN 配置是否使能载波 2 功能。

0：绕过载波 2

1：使能载波 2 功能

(R/W)

MCPWM_CARRIER2_PRESCALE 配置 PWM 载波 2 时钟 (PC_CLK) 的预分频值。PC_CLK 周期 = PWM_CLK 周期 * (PWM_CARRIER0_PRESCALE + 1)。(R/W)

MCPWM_CARRIER2_DUTY 配置载波占空比。占空比 = PWM_CARRIER0_DUTY / 8。(R/W)

MCPWM_CARRIER2_OSHTWTH 配置载波第一个脉冲的宽度，单位为载波周期。(R/W)

MCPWM_CARRIER2_OUT_INVERT 配置是否将此模块的 PWM2A 和 PWM2B 输出反相。

0：无效

1：反相

(R/W)

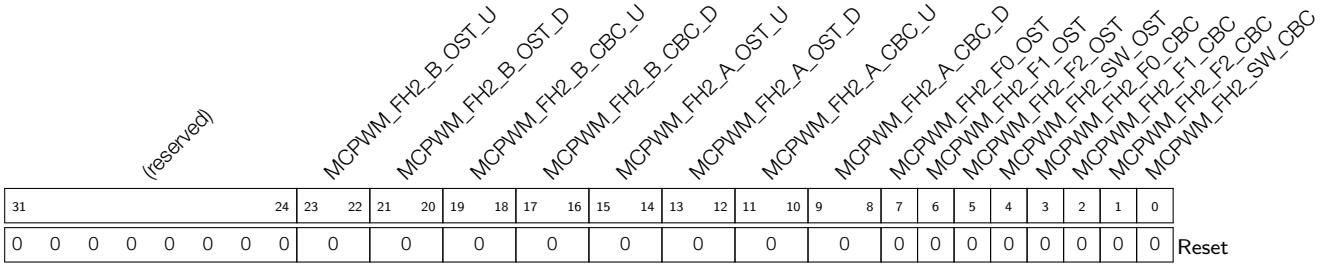
MCPWM_CARRIER2_IN_INVERT 配置是否将此模块的 PWM2A 和 PWM2B 输入反相。

0：无效

1：反相

(R/W)

Register 33.56. MCPWM_FH2_CFG0_REG (0x00D8)



MCPWM_FH2_SW_CBC 软件强制逐周期模式操作的使能寄存器。0：关闭。1：使能。(R/W)

MCPWM_FH2_F2_CBC 设置 fault_event2 触发逐周期模式操作。0：关闭。1：使能。(R/W)

MCPWM_FH2_F1_CBC 设置 fault_event1 触发逐周期模式操作。0：关闭。1：使能。(R/W)

MCPWM_FH2_F0_CBC 设置 fault_event0 触发逐周期模式操作。0：关闭。1：使能。(R/W)

MCPWM_FH2_SW_OST 软件强制一次性模式操作的使能寄存器。0：关闭。1：使能。(R/W)

MCPWM_FH2_F2_OST 设置 fault_event2 触发一次性模式操作。0：关闭。1：使能。(R/W)

MCPWM_FH2_F1_OST 设置 fault_event1 触发一次性模式操作。0：关闭。1：使能。(R/W)

MCPWM_FH2_F0_OST 设置 fault_event0 触发一次性模式操作。0：关闭。1：使能。(R/W)

MCPWM_FH2_A_CBC_D 发生故障事件并且定时器递减时，PWM2A 上的逐周期模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

MCPWM_FH2_A_CBC_U 发生故障事件并且定时器递增时，PWM2A 上的逐周期模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

MCPWM_FH2_A_OST_D 发生故障事件并且定时器递减时，PWM2A 上的一次性模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

MCPWM_FH2_A_OST_U 发生故障事件并且定时器递增时，PWM2A 上的一次性模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

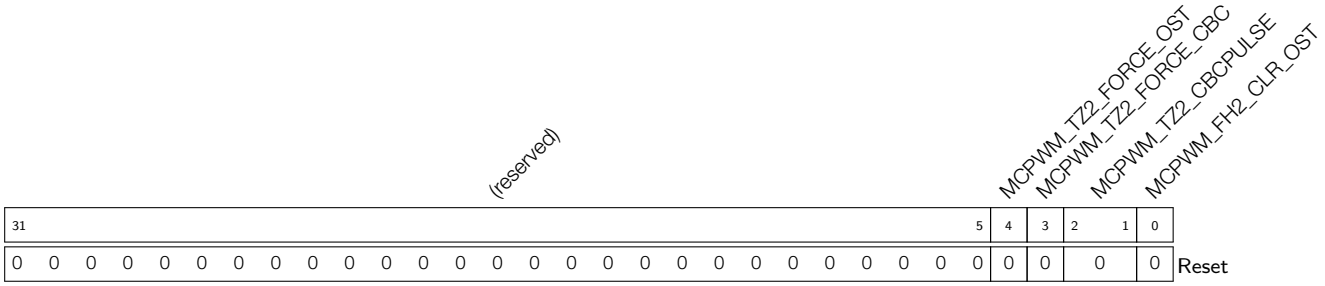
MCPWM_FH2_B_CBC_D 发生故障事件并且定时器递减时，PWM2B 上的逐周期模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

MCPWM_FH2_B_CBC_U 发生故障事件并且定时器递增时，PWM2B 上的逐周期模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

MCPWM_FH2_B_OST_D 发生故障事件并且定时器递减时，PWM2B 上的一次性模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

MCPWM_FH2_B_OST_U 发生故障事件并且定时器递增时，PWM2B 上的一次性模式操作。0：无。1：强制拉低。2：强制拉高。3：取反。(R/W)

Register 33.57. MCPWM_FH2_CFG1_REG (0x00DC)



MCPWM_FH2_CLR_OST 配置是否使上升沿清除正在进行的一次性模式的操作。

0: 不清除

1: 清除

(R/W)

MCPWM_TZ2_CBCPULSE 配置逐周期模式的更新方式。

bit0 为 1：发生 TEZ 事件时

bit1 为 1：发生 TEP 事件时

(R/W)

MCPWM_TZ2_FORCE_CBC 配置是否触发逐周期模式操作。

0：无效

1：触发逐周期模式操作

(R/W)

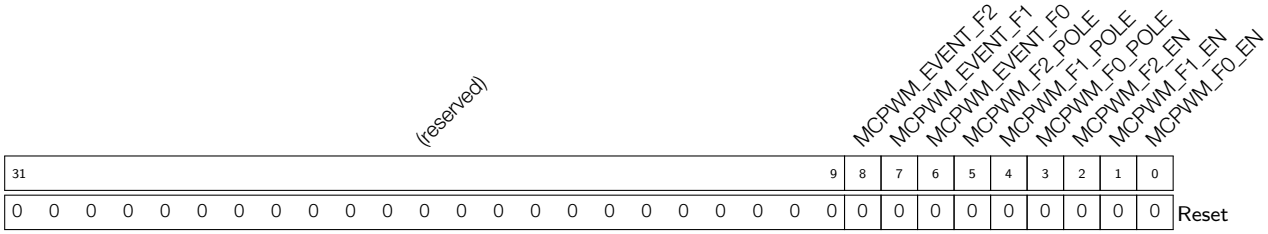
MCPWM_TZ2_FORCE_OST 配置是否触发一次性模式操作。

0：无效

1：触发一次性模式操作

(R/W)

Register 33.58. MCPWM_FAULT_DETECT_REG (0x00E4)



MCPWM_F0_EN 配置是否使能生成 fault_event0。

0：无效

1：使能

(R/W)

MCPWM_F1_EN 配置是否使能生成 fault_event1。

0：无效

1：使能

(R/W)

MCPWM_F2_EN 配置是否使能生成 fault_event2。

0：无效

1：使能

(R/W)

MCPWM_F0_POLE 配置来自 GPIO 矩阵的 FAULT0 信号源触发 fault_event0 时极性。

0：低电平触发

1：高电平触发

(R/W)

MCPWM_F1_POLE 配置来自 GPIO 矩阵的 FAULT1 信号源触发 fault_event1 时极性。

0：低电平触发

1：高电平触发

(R/W)

MCPWM_F2_POLE 配置来自 GPIO 矩阵的 FAULT2 信号源触发 fault_event2 时极性。

0：低电平触发

1：高电平触发

(R/W)

MCPWM_EVENT_F0 表示 fault_event0 事件的状态。此字段由硬件置 1 和清零。

0：fault_event0 事件停止

1：fault_event0 事件持续

(RO)

MCPWM_EVENT_F1 表示 fault_event1 事件的状态。此字段由硬件置 1 和清零。

0：fault_event1 事件停止

1：fault_event1 事件持续

(RO)

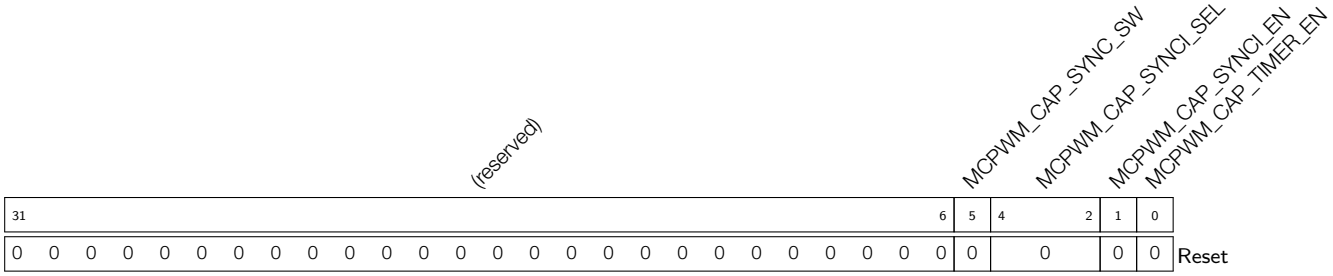
MCPWM_EVENT_F2 表示 fault_event2 事件的状态。此字段由硬件置 1 和清零。

0：fault_event2 事件停止

1：fault_event2 事件持续

(RO)

Register 33.59. MCPWM_CAP_TIMER_CFG_REG (0x00E8)



MCPWM_CAP_TIMER_EN 配置是否使能捕获定时器在 APB_CLK 下的递增。

0：无效

1：使能捕获定时器在 APB_CLK 下的递增

(R/W)

MCPWM_CAP_SYNCI_EN 配置是否使能捕获定时器同步。

0：无效

1：使能捕获定时器同步

(R/W)

MCPWM_CAP_SYNCI_SEL 配置捕获模块的同步输入。

0：无

1：定时器 0 的同步输出

2：定时器 1 的同步输出

3：定时器 2 的同步输出

4：来自 GPIO 矩阵的 SYNC0

5：来自 GPIO 矩阵的 SYNC1

6：来自 GPIO 矩阵的 SYNC2

(R/W)

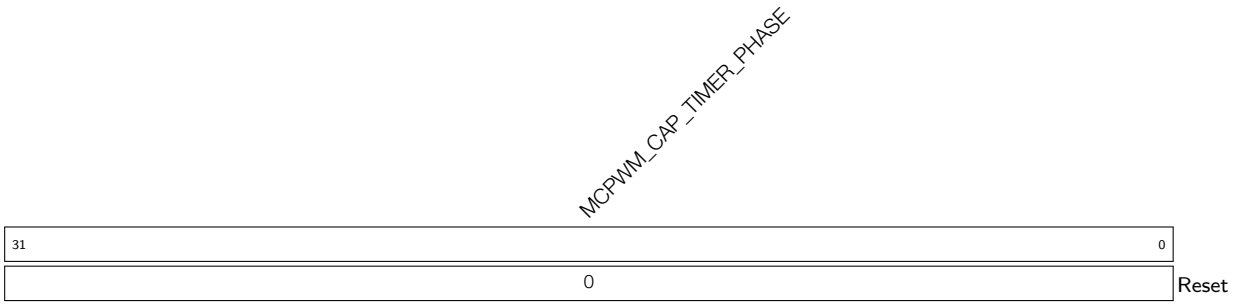
MCPWM_CAP_SYNC_SW 当 MCPWM_CAP_SYNCI_EN 置位时，配置是否同步捕获定时器，在捕获定时器中写入相位寄存器的值。

0：无效

1：同步捕获定时器

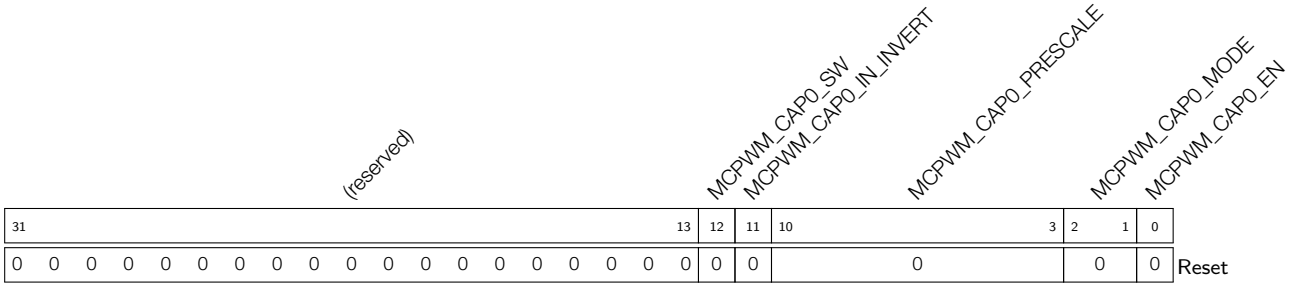
(WT)

Register 33.60. MCPWM_CAP_TIMER_PHASE_REG (0x00EC)



MCPWM_CAP_TIMER_PHASE 捕获定时器同步操作的相位值。(R/W)

Register 33.61. MCPWM_CAP_CH0_CFG_REG (0x00F0)



MCPWM_CAP0_EN 配置是否使能信道 0 上的捕获事件。

0：不使能

1：使能

(R/W)

MCPWM_CAP0_MODE 配置预分频后信道 0 上的捕获沿。

bit0 为 1：使能下降沿捕获。

bit1 为 1：使能上升沿捕获。

(R/W)

MCPWM_CAP0_PRESCALE 配置 CAP0 上升沿的预分频值。预分频值 = PWM_CAP0_PRESCALE + 1。(R/W)

MCPWM_CAP0_IN_INVERT 配置是否在预分频前反相来自 GPIO 矩阵的 CAP0。

0：无效

1：在预分频前反相来自 GPIO 矩阵的 CAP0

(R/W)

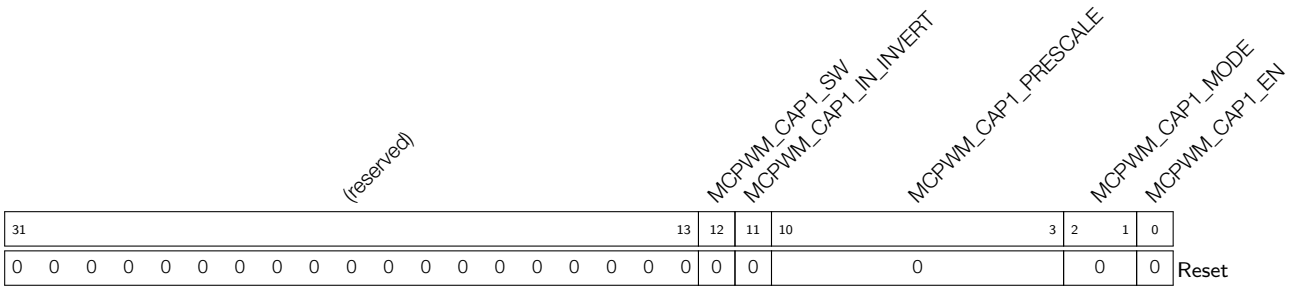
MCPWM_CAP0_SW 配置是否触发信道 0 上的软件强制捕获事件。

0：不触发

1：触发

(WT)

Register 33.62. MCPWM_CAP_CH1_CFG_REG (0x00F4)



MCPWM_CAP1_EN 配置是否使能信道 1 上的捕获事件。

0：不使能

1：使能

(R/W)

MCPWM_CAP1_MODE 配置预分频后信道 1 上的捕获沿。

bit0 为 1：使能下降沿捕获。

bit1 为 1：使能上升沿捕获。

(R/W)

MCPWM_CAP1_PRESCALE 配置 CAP1 上升沿的预分频值。预分频值 = PWM_CAP1_PRESCALE + 1。(R/W)

MCPWM_CAP1_IN_INVERT 配置是否在预分频前反相来自 GPIO 矩阵的 CAP1。

0：无效

1：在预分频前反相来自 GPIO 矩阵的 CAP1

(R/W)

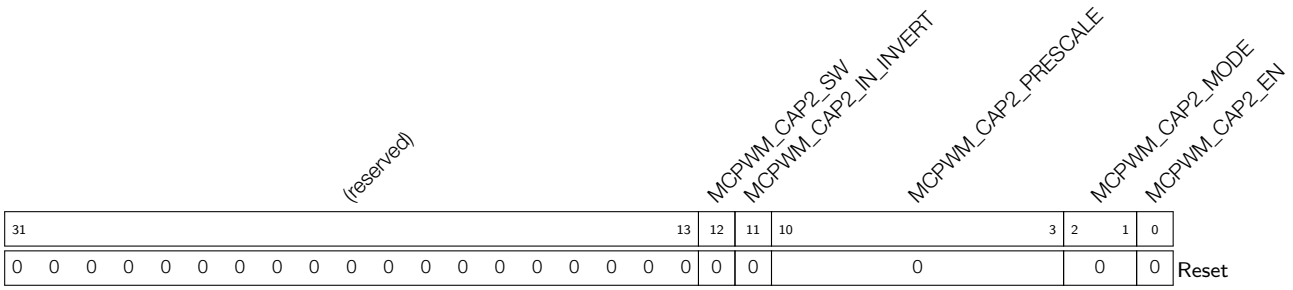
MCPWM_CAP1_SW 配置是否触发信道 1 上的软件强制捕获事件。

0：不触发

1：触发

(WT)

Register 33.63. MCPWM_CAP_CH2_CFG_REG (0x00F8)



MCPWM_CAP2_EN 配置是否使能信道 2 上的捕获事件。

0：不使能

1：使能

(R/W)

MCPWM_CAP2_MODE 配置预分频后信道 2 上的捕获沿。

bit0 为 1：使能下降沿捕获。

bit1 为 1：使能上升沿捕获。

(R/W)

MCPWM_CAP2_PRESCALE 配置 CAP2 上升沿的预分频值。该预分频值 = PWM_CAP2_PRESCALE + 1. (R/W)

MCPWM_CAP2_IN_INVERT 配置是否在预分频前反相来自 GPIO 矩阵的 CAP2。

0：无效

1：在预分频前反相来自 GPIO 矩阵的 CAP2

(R/W)

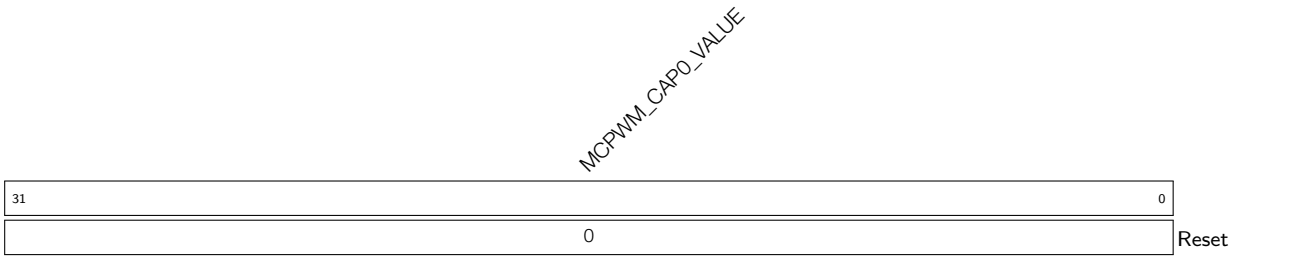
MCPWM_CAP2_SW 配置是否触发信道 2 上的软件强制捕获事件。

0：不触发

1：触发

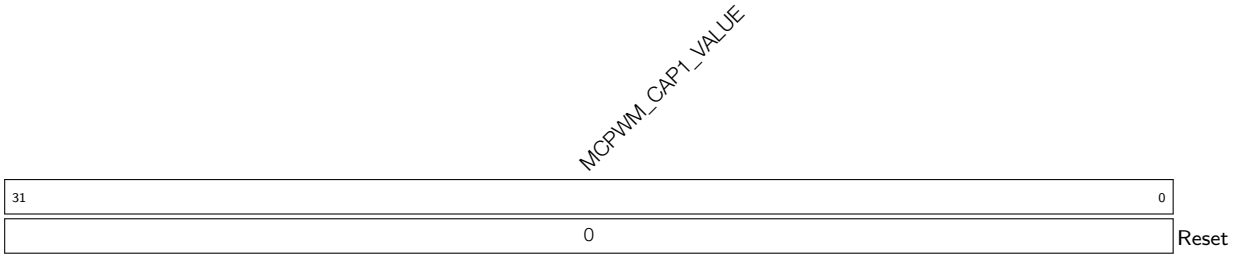
(WT)

Register 33.64. MCPWM_CAP_CH0_REG (0x00FC)



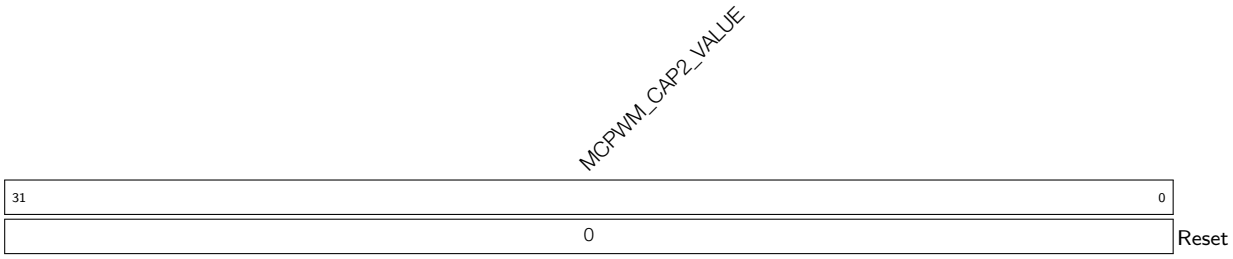
MCPWM_CAP0_VALUE 表示信道 0 上一次捕获的值。(RO)

Register 33.65. MCPWM_CAP_CH1_REG (0x0100)



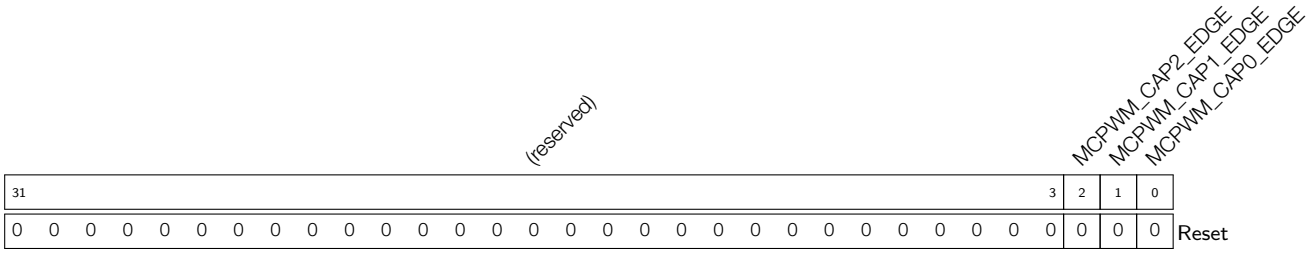
MCPWM_CAP1_VALUE 表示信道 1 上一次捕获的值。(RO)

Register 33.66. MCPWM_CAP_CH2_REG (0x0104)



MCPWM_CAP2_VALUE 表示信道 2 上一次捕获的值。(RO)

Register 33.67. MCPWM_CAP_STATUS_REG (0x0108)



MCPWM_CAP0_EDGE 信道 0 上一次捕获触发事件的边沿。

0：上升沿。

1：下降沿。

(RO)

MCPWM_CAP1_EDGE 信道 1 上一次捕获触发事件的边沿。

0：上升沿。

1：下降沿。

(RO)

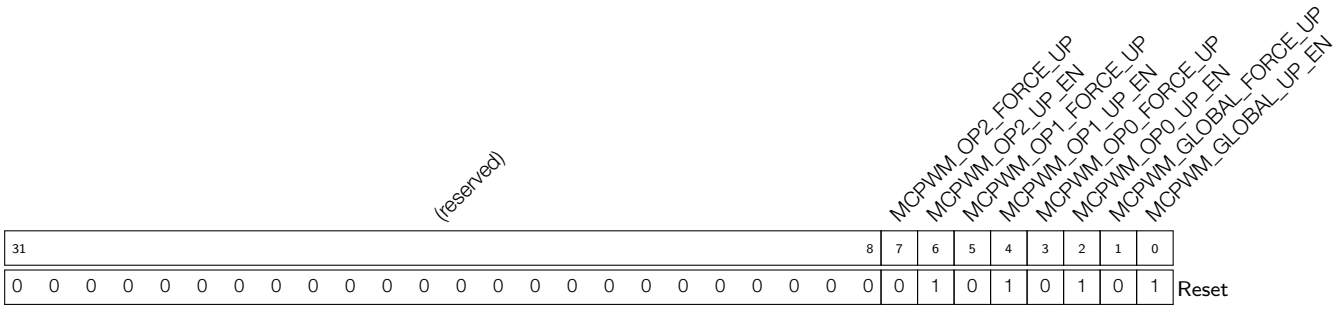
MCPWM_CAP2_EDGE 信道 2 上一次捕获触发事件的边沿。

0：上升沿。

1：下降沿。

(RO)

Register 33.68. MCPWM_UPDATE_CFG_REG (0x010C)



MCPWM_GLOBAL_UP_EN 配置是否更新 MCPWM 模块的所有有效寄存器。

0：无效 1：更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_GLOBAL_FORCE_UP 配置是否强制更新 MCPWM 模块的所有有效寄存器。

0：无效 1：强制更新 MCPWM 模块的所有有效寄存器

(R/W)

MCPWM_OP0_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 0 中的有效寄存器。

0：无效 1：更新 PWM 操作器 0 中的有效寄存器

(R/W)

MCPWM_OP0_FORCE_UP 配置是否强制更新 PWM 操作器 0 中的有效寄存器。

0：无效 1：强制更新

(R/W)

MCPWM_OP1_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 1 中的有效寄存器。

0：无效 1：更新 PWM 操作器 1 中的有效寄存器

(R/W)

MCPWM_OP1_FORCE_UP 配置是否强制更新 PWM 操作器 1 中的有效寄存器。

0：无效 1：强制更新

(R/W)

MCPWM_OP2_UP_EN 当 MCPWM_GLOBAL_UP_EN 置 1 时，配置是否更新 PWM 操作器 2 中的有效寄存器。

0：无效 1：更新 PWM 操作器 2 中的有效寄存器

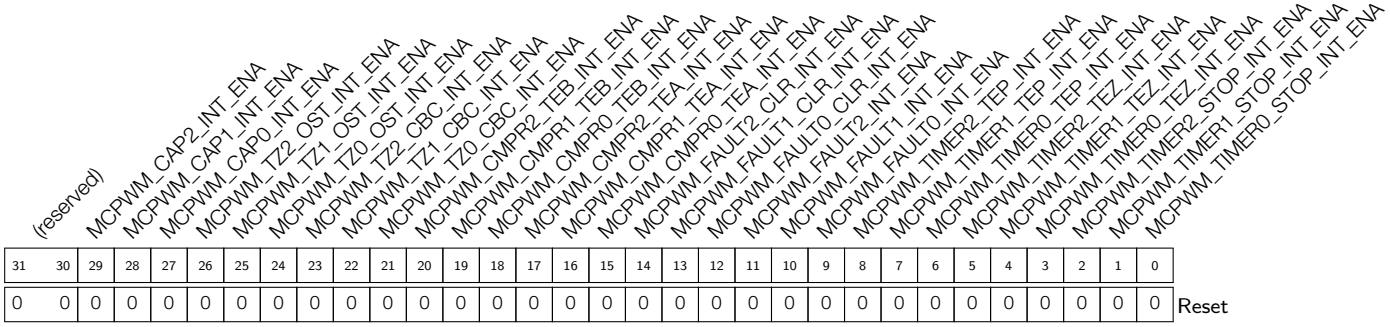
(R/W)

MCPWM_OP2_FORCE_UP 配置是否强制更新 PWM 操作器 2 中的有效寄存器。

0：无效 1：强制更新

(R/W)

Register 33.69. MCPWM_INT_ENA_REG (0x0110)



MCPWM_TIMER0_STOP_INT_ENA 写 1 使能 MCPWM_TIMER0_STOP_INT。(R/W)

MCPWM_TIMER1_STOP_INT_ENA 写 1 使能 MCPWM_TIMER1_STOP_INT。(R/W)

MCPWM_TIMER2_STOP_INT_ENA 写 1 使能 MCPWM_TIMER2_STOP_INT。(R/W)

MCPWM_TIMER0_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER0_TEZ_INT。(R/W)

MCPWM_TIMER1_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER1_TEZ_INT。(R/W)

MCPWM_TIMER2_TEZ_INT_ENA 写 1 使能 MCPWM_TIMER2_TEZ_INT。(R/W)

MCPWM_TIMER0_TEP_INT_ENA 写 1 使能 MCPWM_TIMER0_TEP_INT。(R/W)

MCPWM_TIMER1_TEP_INT_ENA 写 1 使能 MCPWM_TIMER1_TEP_INT。(R/W)

MCPWM_TIMER2_TEP_INT_ENA 写 1 使能 MCPWM_TIMER2_TEP_INT。(R/W)

MCPWM_FAULT0_INT_ENA 写 1 使能 MCPWM_FAULT0_INT。(R/W)

MCPWM_FAULT1_INT_ENA 写 1 使能 MCPWM_FAULT1_INT。(R/W)

MCPWM_FAULT2_INT_ENA 写 1 使能 MCPWM_FAULT2_INT。(R/W)

MCPWM_FAULT0_CLR_INT_ENA 写 1 使能 MCPWM_FAULT0_CLR_INT。(R/W)

MCPWM_FAULT1_CLR_INT_ENA 写 1 使能 MCPWM_FAULT1_CLR_INT。(R/W)

MCPWM_FAULT2_CLR_INT_ENA 写 1 使能 MCPWM_FAULT2_CLR_INT。(R/W)

MCPWM_CMPR0_TEA_INT_ENA 写 1 使能 MCPWM_CMPR0_TEA_INT。(R/W)

MCPWM_CMPR1_TEA_INT_ENA 写 1 使能 MCPWM_CMPR1_TEA_INT。(R/W)

MCPWM_CMPR2_TEA_INT_ENA 写 1 使能 MCPWM_CMPR2_TEA_INT。(R/W)

见下页...

Register 33.69. MCPWM_INT_ENA_REG (0x0110)

... 接上页

MCPWM_CMPR0_TEB_INT_ENA 写 1 使能 MCPWM_CMPR0_TEB_INT。(R/W)

MCPWM_CMPR1_TEB_INT_ENA 写 1 使能 MCPWM_CMPR1_TEB_INT。(R/W)

MCPWM_CMPR2_TEB_INT_ENA 写 1 使能 MCPWM_CMPR2_TEB_INT。(R/W)

MCPWM_TZ0_CBC_INT_ENA 写 1 使能 MCPWM_TZ0_CBC_INT。(R/W)

MCPWM_TZ1_CBC_INT_ENA 写 1 使能 MCPWM_TZ1_CBC_INT。(R/W)

MCPWM_TZ2_CBC_INT_ENA 写 1 使能 MCPWM_TZ2_CBC_INT。(R/W)

MCPWM_TZ0_OST_INT_ENA 写 1 使能 MCPWM_TZ0_OST_INT。(R/W)

MCPWM_TZ1_OST_INT_ENA 写 1 使能 MCPWM_TZ1_OST_INT。(R/W)

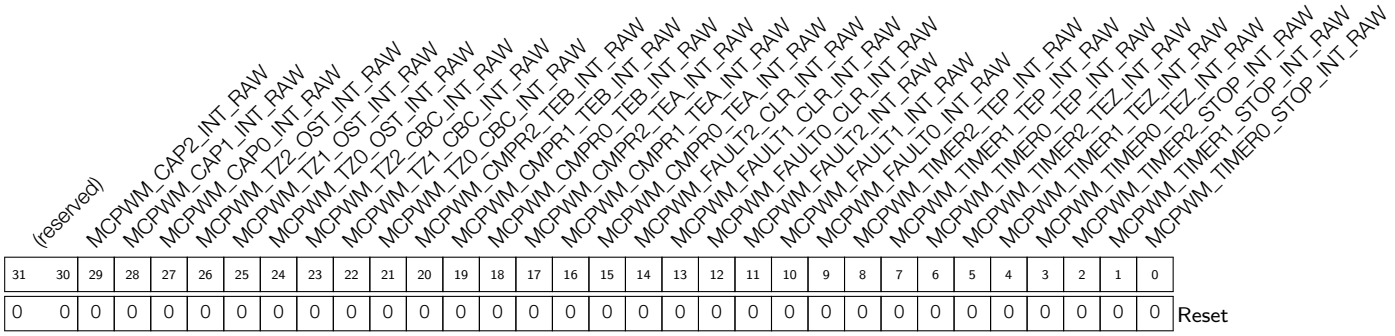
MCPWM_TZ2_OST_INT_ENA 写 1 使能 MCPWM_TZ2_OST_INT。(R/W)

MCPWM_CAP0_INT_ENA 写 1 使能 MCPWM_CAP0_INT。(R/W)

MCPWM_CAP1_INT_ENA 写 1 使能 MCPWM_CAP1_INT。(R/W)

MCPWM_CAP2_INT_ENA 写 1 使能 MCPWM_CAP2_INT。(R/W)

Register 33.70. MCPWM_INT_RAW_REG (0x0114)



MCPWM_TIMER0_STOP_INT_RAW MCPWM_TIMER0_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_STOP_INT_RAW MCPWM_TIMER1_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_STOP_INT_RAW MCPWM_TIMER2_STOP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEZ_INT_RAW MCPWM_TIMER0_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEZ_INT_RAW MCPWM_TIMER1_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEZ_INT_RAW MCPWM_TIMER2_TEZ_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER0_TEP_INT_RAW MCPWM_TIMER0_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER1_TEP_INT_RAW MCPWM_TIMER1_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_TIMER2_TEP_INT_RAW MCPWM_TIMER2_TEP_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_INT_RAW MCPWM_FAULT0_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_INT_RAW MCPWM_FAULT1_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_INT_RAW MCPWM_FAULT2_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT0_CLR_INT_RAW MCPWM_FAULT0_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT1_CLR_INT_RAW MCPWM_FAULT1_CLR_INT 的原始状态位。(R/WTC/SS)

MCPWM_FAULT2_CLR_INT_RAW MCPWM_FAULT2_CLR_INT 的原始状态位。(R/WTC/SS)

见下页...

Register 33.70. MCPWM_INT_RAW_REG (0x0114)

... 接上页

MCPWM_CMPR0_TEA_INT_RAW MCPWM_CMPR0_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEA_INT_RAW MCPWM_CMPR1_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEA_INT_RAW MCPWM_CMPR2_TEA_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR0_TEB_INT_RAW MCPWM_CMPR0_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR1_TEB_INT_RAW MCPWM_CMPR1_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_CMPR2_TEB_INT_RAW MCPWM_CMPR2_TEB_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ0_CBC_INT_RAW MCPWM_TZ0_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_CBC_INT_RAW MCPWM_TZ1_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ2_CBC_INT_RAW MCPWM_TZ2_CBC_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ0_OST_INT_RAW MCPWM_TZ0_OST_INT 的原始状态位。(R/WTC/SS)

MCPWM_TZ1_OST_INT_RAW MCPWM_TZ1_OST_INT 的原始状态位。(R/WTC/SS)

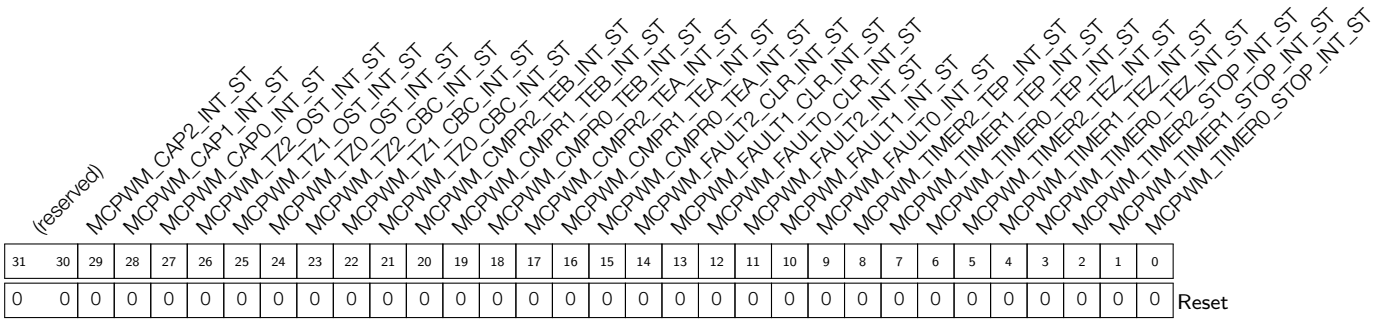
MCPWM_TZ2_OST_INT_RAW MCPWM_TZ2_OST_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP0_INT_RAW MCPWM_CAP0_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP1_INT_RAW MCPWM_CAP1_INT 的原始状态位。(R/WTC/SS)

MCPWM_CAP2_INT_RAW MCPWM_CAP2_INT 的原始状态位。(R/WTC/SS)

Register 33.71. MCPWM_INT_ST_REG (0x0118)



MCPWM_TIMER0_STOP_INT_ST MCPWM_TIMER0_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_STOP_INT_ST MCPWM_TIMER1_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_STOP_INT_ST MCPWM_TIMER2_STOP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEZ_INT_ST MCPWM_TIMER0_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEZ_INT_ST MCPWM_TIMER1_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEZ_INT_ST MCPWM_TIMER2_TEZ_INT 的屏蔽状态位。(RO)

MCPWM_TIMER0_TEP_INT_ST MCPWM_TIMER0_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER1_TEP_INT_ST MCPWM_TIMER1_TEP_INT 的屏蔽状态位。(RO)

MCPWM_TIMER2_TEP_INT_ST MCPWM_TIMER2_TEP_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_INT_ST MCPWM_FAULT0_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_INT_ST MCPWM_FAULT1_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_INT_ST MCPWM_FAULT2_INT_INT 的屏蔽状态位。(RO)

MCPWM_FAULT0_CLR_INT_ST MCPWM_FAULT0_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT1_CLR_INT_ST MCPWM_FAULT1_CLR_INT 的屏蔽状态位。(RO)

MCPWM_FAULT2_CLR_INT_ST MCPWM_FAULT2_CLR_INT 的屏蔽状态位。(RO)

MCPWM_CMPR0_TEA_INT_ST MCPWM_CMPR0_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEA_INT_ST MCPWM_CMPR1_TEA_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEA_INT_ST MCPWM_CMPR2_TEA_INT 的屏蔽状态位。(RO)

见下页...

Register 33.71. MCPWM_INT_ST_REG (0x0118)

... 接上页

MCPWM_CMPR0_TEB_INT_ST MCPWM_CMPR0_TEB_INT 的屏蔽状态位。(RO)

MCPWM_CMPR1_TEB_INT_ST MCPWM_CMPR1_TEB_INT 的屏蔽状态位。(RO)

MCPWM_CMPR2_TEB_INT_ST MCPWM_CMPR2_TEB_INT 的屏蔽状态位。(RO)

MCPWM_TZ0_CBC_INT_ST MCPWM_TZ0_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ1_CBC_INT_ST MCPWM_TZ1_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ2_CBC_INT_ST MCPWM_TZ2_CBC_INT 的屏蔽状态位。(RO)

MCPWM_TZ0_OST_INT_ST MCPWM_TZ0_OST_INT 的屏蔽状态位。(RO)

MCPWM_TZ1_OST_INT_ST MCPWM_TZ1_OST_INT 的屏蔽状态位。(RO)

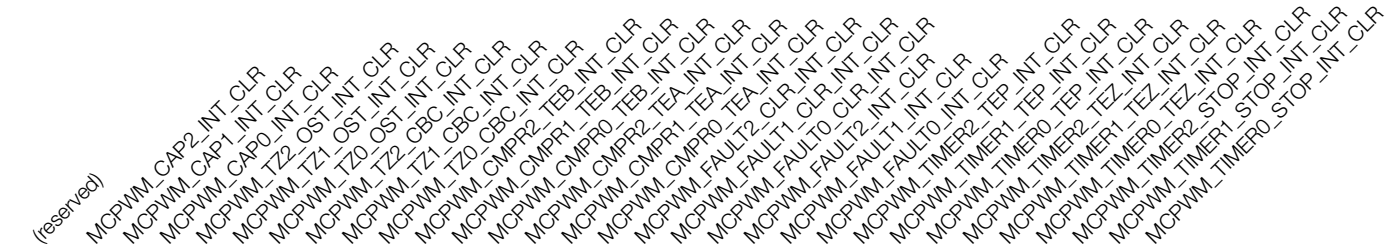
MCPWM_TZ2_OST_INT_ST MCPWM_TZ2_OST_INT 的屏蔽状态位。(RO)

MCPWM_CAP0_INT_ST MCPWM_CAP0_INT 的屏蔽状态位。(RO)

MCPWM_CAP1_INT_ST MCPWM_CAP1_INT 的屏蔽状态位。(RO)

MCPWM_CAP2_INT_ST MCPWM_CAP2_INT 的屏蔽状态位。(RO)

Register 33.72. MCPWM_INT_CLR_REG (0x011C)



MCPWM_TIMER0_STOP_INT_CLR 写 1 清除 MCPWM_TIMER0_STOP_INT。(WT)

MCPWM_TIMER1_STOP_INT_CLR 写 1 清除 MCPWM_TIMER1_STOP_INT。(WT)

MCPWM_TIMER2_STOP_INT_CLR 写 1 清除 MCPWM_TIMER2_STOP_INT。(WT)

MCPWM_TIMER0_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER0_TEZ_INT。(WT)

MCPWM_TIMER1_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER1_TEZ_INT。(WT)

MCPWM_TIMER2_TEZ_INT_CLR 写 1 清除 MCPWM_TIMER2_TEZ_INT。(WT)

MCPWM_TIMER0_TEP_INT_CLR 写 1 清除 MCPWM_TIMER0_TEP_INT。(WT)

MCPWM_TIMER1_TEP_INT_CLR 写 1 清除 MCPWM_TIMER1_TEP_INT。(WT)

MCPWM_TIMER2_TEP_INT_CLR 写 1 清除 MCPWM_TIMER2_TEP_INT。(WT)

MCPWM_FAULT0_INT_CLR 写 1 清除 MCPWM_FAULT0_INT。(WT)

MCPWM_FAULT1_INT_CLR 写 1 清除 MCPWM_FAULT1_INT。(WT)

MCPWM_FAULT2_INT_CLR 写 1 清除 MCPWM_FAULT2_INT。(WT)

MCPWM_FAULT0_CLR_INT_CLR 写 1 清除 MCPWM_FAULT0_CLR_INT。(WT)

MCPWM_FAULT1_CLR_INT_CLR 写 1 清除 MCPWM_FAULT1_CLR_INT。(WT)

MCPWM_FAULT2_CLR_INT_CLR 写 1 清除 MCPWM_FAULT2_CLR_INT。(WT)

见下页...

Register 33.72. MCPWM_INT_CLR_REG (0x011C)

... 接上页

MCPWM_CMPR0_TEA_INT_CLR 写 1 清除 MCPWM_CMPR0_TEA_INT。(WT)

MCPWM_CMPR1_TEA_INT_CLR 写 1 清除 MCPWM_CMPR1_TEA_INT。(WT)

MCPWM_CMPR2_TEA_INT_CLR 写 1 清除 MCPWM_CMPR2_TEA_INT。(WT)

MCPWM_CMPR0_TEB_INT_CLR 写 1 清除 MCPWM_CMPR0_TEB_INT。(WT)

MCPWM_CMPR1_TEB_INT_CLR 写 1 清除 MCPWM_CMPR1_TEB_INT。(WT)

MCPWM_CMPR2_TEB_INT_CLR 写 1 清除 MCPWM_CMPR2_TEB_INT。(WT)

MCPWM_TZ0_CBC_INT_CLR 写 1 清除 MCPWM_TZ0_CBC_INT。(WT)

MCPWM_TZ1_CBC_INT_CLR 写 1 清除 MCPWM_TZ1_CBC_INT。(WT)

MCPWM_TZ2_CBC_INT_CLR 写 1 清除 MCPWM_TZ2_CBC_INT。(WT)

MCPWM_TZ0_OST_INT_CLR 写 1 清除 MCPWM_TZ0_OST_INT。(WT)

MCPWM_TZ1_OST_INT_CLR 写 1 清除 MCPWM_TZ1_OST_INT。(WT)

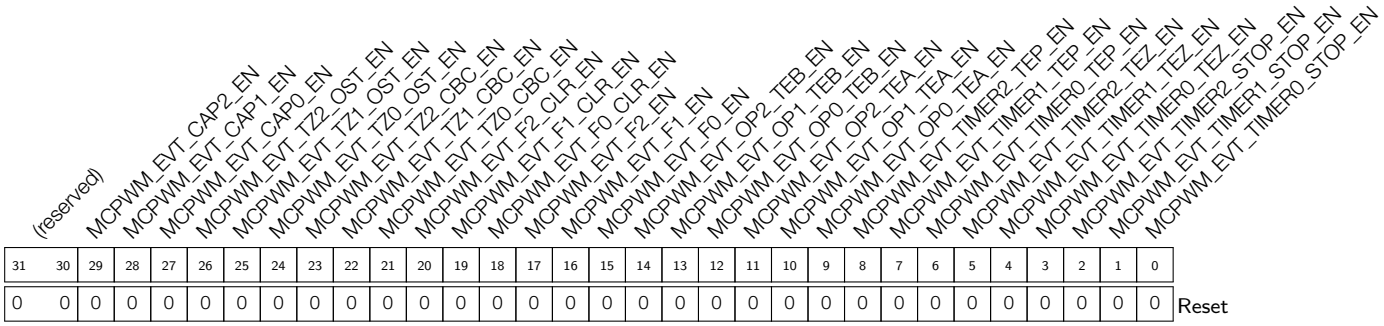
MCPWM_TZ2_OST_INT_CLR 写 1 清除 MCPWM_TZ2_OST_INT。(WT)

MCPWM_CAP0_INT_CLR 写 1 清除 MCPWM_CAP0_INT。(WT)

MCPWM_CAP1_INT_CLR 写 1 清除 MCPWM_CAP1_INT。(WT)

MCPWM_CAP2_INT_CLR 写 1 清除 MCPWM_CAP2_INT。(WT)

Register 33.73. MCPWM_EVT_EN_REG (0x0120)



MCPWM_EVT_TIMER0_STOP_EN 配置是否使能当定时器 0 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_STOP_EN 配置是否使能当定时器 1 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER2_STOP_EN 配置是否使能当定时器 2 停止时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER0_TEZ_EN 配置是否使能当定时器 0 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_TEZ_EN 配置是否使能当定时器 1 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER2_TEZ_EN 配置是否使能当定时器 2 等于零时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER0_TEP_EN 配置是否使能当定时器 0 等于周期时的事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_TIMER1_TEP_EN 配置是否使能当定时器 1 等于周期时的事件生成。

0：关闭

1：使能

(R/W)

见下页...

Register 33.73. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TIMER2_TEP_EN 配置是否使能当定时器 2 等于周期时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP0_TEA_EN 配置是否使能当生成器 0 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP1_TEA_EN 配置是否使能当生成器 1 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP2_TEA_EN 配置是否使能当生成器 2 等于时间戳 A 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP0_TEB_EN 配置是否使能当生成器 0 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP1_TEB_EN 配置是否使能当生成器 1 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_OP2_TEB_EN 配置是否使能当生成器 2 等于时间戳 B 时的事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F0_EN 配置是否使能 FAULT0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F1_EN 配置是否使能 FAULT1 事件生成。

0：关闭
1：使能
(R/W)

见下页...

Register 33.73. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_F2_EN 配置是否使能 FAULT2 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F0_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F1_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_F2_CLR_EN 配置是否使能 FAULT0 清除事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ0_CBC_EN 配置是否使能 CBC0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ1_CBC_EN 配置是否使能 CBC1 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ2_CBC_EN 配置是否使能 CBC2 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ0_OST_EN 配置是否使能 OST0 事件生成。

0：关闭
1：使能
(R/W)

MCPWM_EVT_TZ1_OST_EN 配置是否使能 OST1 事件生成。

0：关闭
1：使能
(R/W)

见下页...

Register 33.73. MCPWM_EVT_EN_REG (0x0120)

... 接上页

MCPWM_EVT_TZ2_OST_EN 配置是否使能 OST2 事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_CAP0_EN 配置是否使能 capture0 事件生成。

0：关闭

1：使能

(R/W)

MCPWM_EVT_CAP1_EN 配置是否使能 capture1 事件生成。

0：关闭

1：使能

(R/W)

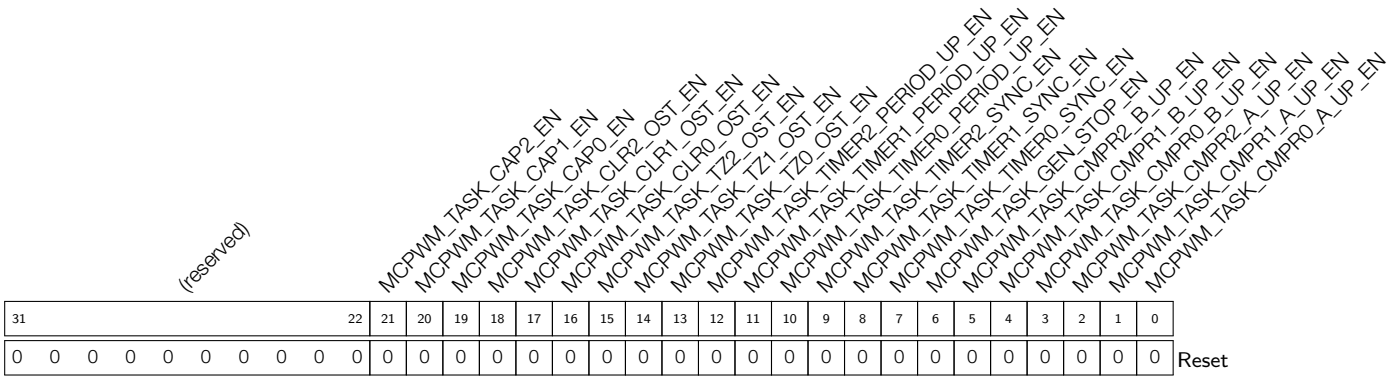
MCPWM_EVT_CAP2_EN 配置是否使能 capture2 事件生成。

0：关闭

1：使能

(R/W)

Register 33.74. MCPWM_TASK_EN_REG (0x0124)



MCPWM_TASK_CMPR0_A_UP_EN 配置是否接收 PWM 生成器 0 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR1_A_UP_EN 配置是否接收 PWM 生成器 1 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR2_A_UP_EN 配置是否接收 PWM 生成器 2 时间戳 A 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR0_B_UP_EN 配置是否接收 PWM 生成器 0 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR1_B_UP_EN 配置是否接收 PWM 生成器 1 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CMPR2_B_UP_EN 配置是否接收 PWM 生成器 2 时间戳 B 的影子寄存器的更新任务。

0：无效

1：接收

(R/W)

见下页...

Register 33.74. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_GEN_STOP_EN 配置是否接收所有 PWM 生成器的停止任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER0_SYNC_EN 配置是否接收定时器 0 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER1_SYNC_EN 配置是否接收定时器 1 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER2_SYNC_EN 配置是否接收定时器 2 的同步任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER0_PERIOD_UP_EN 配置是否接收定时器 0 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER1_PERIOD_UP_EN 配置是否接收定时器 1 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TIMER2_PERIOD_UP_EN 配置是否接收定时器 2 的周期更新任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TZ0_OST_EN 配置是否接收 OST0 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_TZ1_OST_EN 配置是否接收 OST1 任务。

0：无效

1：接收

(R/W)

见下页...

Register 33.74. MCPWM_TASK_EN_REG (0x0124)

... 接上页

MCPWM_TASK_TZ2_OST_EN 配置是否接收 OST2 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR0_OST_EN 配置是否接收 OST0 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR1_OST_EN 配置是否接收 OST1 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CLR2_OST_EN 配置是否接收 OST2 清除任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CAP0_EN 配置是否接收 capture0 任务。

0：无效

1：接收

(R/W)

MCPWM_TASK_CAP1_EN 配置是否接收 capture1 任务。

0：无效

1：接收

(R/W)

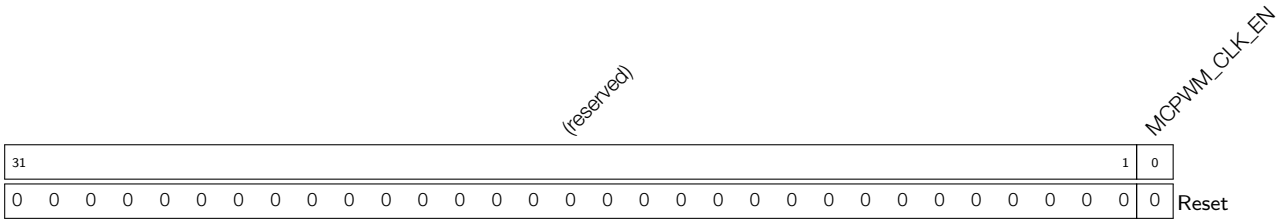
MCPWM_TASK_CAP2_EN 配置是否接收 capture2 任务。

0：无效

1：接收

(R/W)

Register 33.75. MCPWM_CLK_REG (0x0128)



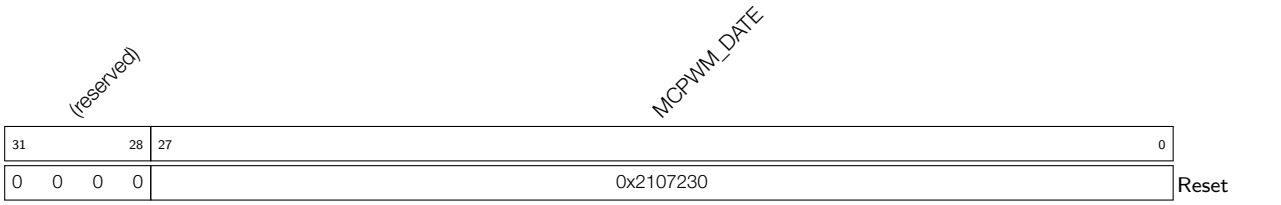
MCPWM_CLK_EN 配置是否强制使能时钟。

0：无效

1：强制使能时钟

(R/W)

Register 33.76. MCPWM_VERSION_REG (0x012C)



MCPWM_DATE 版本控制寄存器。(R/W)

34 红外遥控 (RMT)

34.1 概述

RMT（红外遥控）是一个红外发送和接收控制器，可通过软件编解码多种红外协议。RMT 模块可以实现将模块内置 RAM 中的脉冲编码转换为信号输出，或将模块的输入信号转换为脉冲编码存入 RAM 中。此外，RMT 模块也支持对输出信号进行载波调制或对输入信号进行解调和滤波处理。

RMT 共有四个通道，编码为 0 ~ 3，各通道可独立用于发送或接收信号：

- 0 ~ 1 通道专门用于信号发送；
- 2 ~ 3 通道专门用于信号接收。

每个发送通道和接收通道分别有一组功能相同的寄存器。为了方便叙述，以 *n* 表示各个发送通道，以 *m* 表示各个接收通道。

34.2 主要特性

RMT 控制器具有如下特性：

- 共配置四个通道：
  - 两个通道支持发送
  - 两个通道支持接收
  - 四个通道共享 192 x 32 位的 RAM
- 发射器支持以下模式：
  - 普通发送模式
  - 乒乓发送模式
  - 持续发送模式
  - 载波调制
  - 多通道同时发送
- 接收器支持以下模式：
  - 普通接收模式
  - 乒乓接收模式
  - 接收滤波
  - 载波解调

34.3 功能描述

34.3.1 RMT 架构

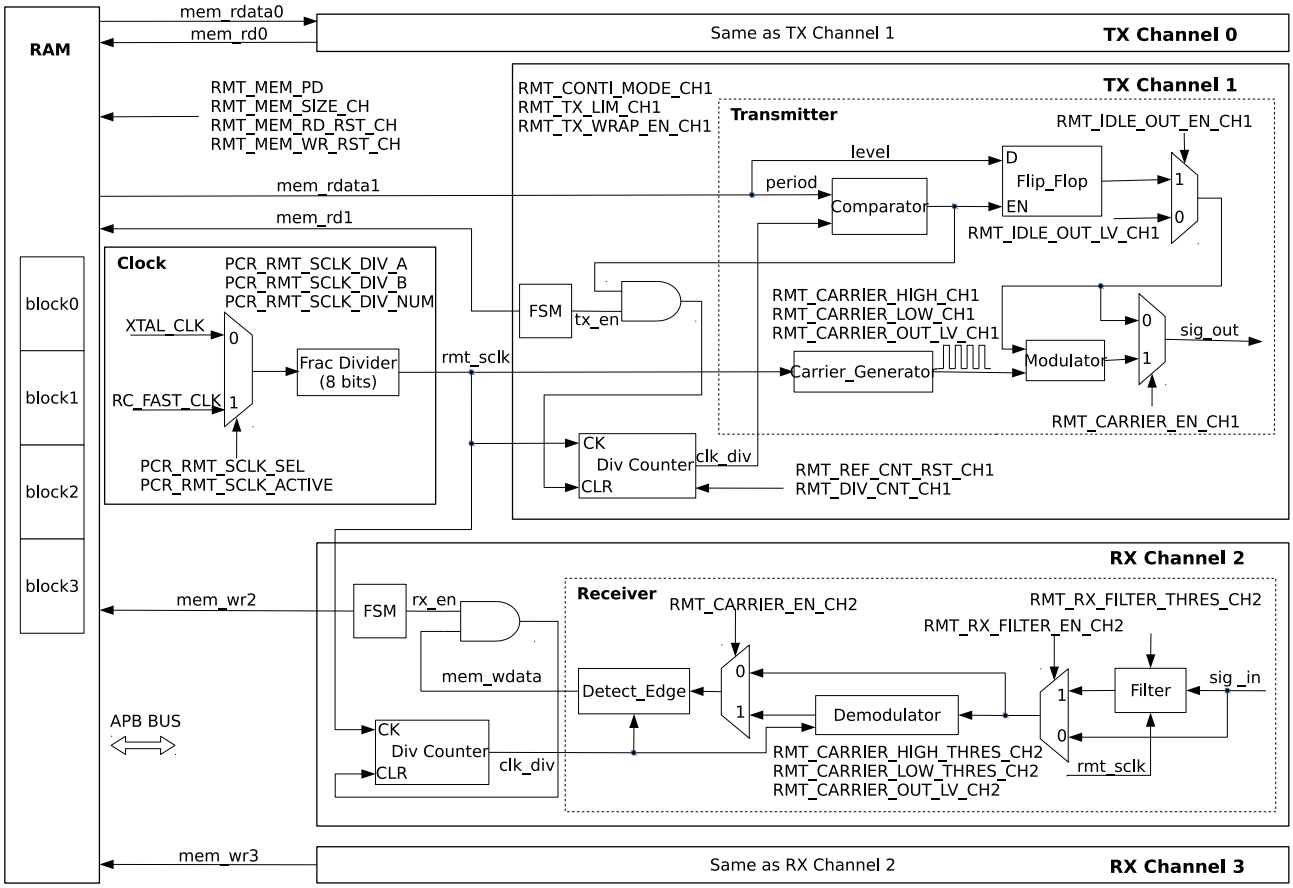


图 34-1. RMT 结构框图

如图 34-1 所示，每个发送通道内部各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个发射器 (Transmitter)

每个接收通道内部也各有：

- 一个时钟分频计数器 (Div Counter)
- 一个状态机 (FSM)
- 一个接收器 (Receiver)

四个通道共享一块 192 x 32 位的 RAM。

34.3.2 RMT RAM

34.3.2.1 RAM 结构

RAM 中脉冲编码结构如图 34-2 所示。每个脉冲编码为 16 位，由 level 与 period 两部分组成。其中 level 表示输入或输出信号的逻辑电平值（0 或 1），period 表示该电平信号持续的时钟（见图 34-1 中的 clk_div）周期数。

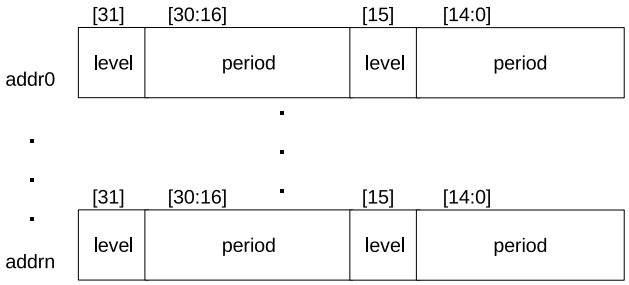


图 34-2. RAM 中脉冲编码结构

period 的最小值为 0，period 为 0 是一次传输的结束标志。对于非 0 的 period，即非结束标志的 period，其值需要满足与 APB 时钟和 RMT 时钟相关的限制条件，具体请参考以下公式：

$$3 \times T_{apb\_clk} + 5 \times T_{rmt\_sclk} < period \times T_{clk\_div} \quad (1)$$

34.3.2.2 RAM 使用说明

RAM 按照 48 x 32 位分成四个 block。默认情况下，每个通道只能使用一个 block（固定为通道 0 使用 block 0，通道 1 使用 block 1，以此类推）。

当发送通道 *n* 或接收通道 *m* 单次传输的脉冲编码数大于一个 block 时，可以：

- 置位 RMT_MEM_TX/RX_WRAP_EN_CH*n*/*m* 使能乒乓操作；
- 或通过配置 RMT_MEM_SIZE_CH*n*/*m* 寄存器，允许该通道占用多个 block。

当设置 RMT_MEM_SIZE_CH*n*/*m* > 1 时，通道 *n*/*m* 将占用 block (*n*/*m*) ~ block (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* −1) 的存储空间，此时通道 (*n*/*m* + 1) ~ (*n*/*m* + RMT_MEM_SIZE_CH*n*/*m* − 1) 因为对应的 RAM block 被占用而无法使用。例如，如果通道 0 配置使用了 block 0 和 block 1，则通道 1 无法使用，通道 2 和通道 3 不受影响。

注意，每个通道使用 RAM 的空间是根据地址从低到高进行映射的，因此通道 0 可以通过配置 RMT_MEM_SIZE_CH0 寄存器来使用通道 1、2、3 的 RAM 空间，但是通道 3 不能使用通道 0、1 或 2 的 RAM 空间。因此 RMT_MEM_SIZE_CH*n* 的最大值不应超过 (4 − *n*)，RMT_MEM_SIZE_CH*m* 的最大值不应超过 (2 − *m*)。

RAM 可被 APB 总线及通道的发射器或接收器访问。为了防止接收通道访问 RAM 和 APB 访问时发生冲突（例如 APB 读 RAM 时，通道向 RAM 发起写操作），可以通过配置 RMT_MEM_OWNER_CH*m* 来决定当前 RAM 的使用权。当接收通道发生越权访问时会产生 RMT_MEM_OWNER_ERR_CH*m* 标志信号。

当 RMT 模块不工作时，可以通过配置 RMT_MEM_FORCE_PD 寄存器使 RAM 工作于低功耗模式。

34.3.2.3 RAM 访问方式

APB 总线访问 RAM 有 FIFO 和 NONFIFO（直接地址）两种模式：

- RMT_APB_FIFO_MASK 置 0 时，选择 FIFO 模式；
- RMT_APB_FIFO_MASK 置 1 时，选择 NONFIFO 模式。

FIFO 模式

在 FIFO 模式下，APB 通过固定地址 RMT_CH*n*/*m*DATA_REG 向 RAM 写数据或从 RAM 读数据。

NONFIFO 模式

在 NONFIFO 模式下，APB 向连续地址段写入数据，或从连续地址段读出数据：

- 发送通道 *n* 对应的写地址段的首地址是：RMT 基地址 + 0x400 + (n − 1) x 48，第 2 个数据的访问地址是 RMT 基地址 + 0x400 + (n − 1) x 48 + 0x4，以此类推，后面的地址依次加上 0x4。
- 接收通道 *m* 对应的读地址段的首地址是：RMT 基地址 + 0x460 + (m − 1) x 48，第 2 个数据的访问地址是 RMT 基地址 + 0x460 + (m − 1) x 48 + 0x4，以此类推，后面的地址依次加上 0x4。

34.3.3 时钟

用户可以通过配置 PCR_RMT_SCLK_SEL 选择 RMT 的时钟源：XTAL_CLK 或 RC_FAST_CLK，置位 PCR_RMT_SCLK_EN 来打开 RMT 的时钟。选择后的时钟经过小数分频得到 RMT 的工作时钟（见图 34-1 中的 rmt_sclk），分频系数为：

$$PCR\_RMT\_SCLK\_DIV\_NUM + 1 + PCR\_RMT\_SCLK\_DIV\_A/PCR\_RMT\_SCLK\_DIV\_B$$

更多信息，请参考章节 7 复位和时钟。RMT_DIV_CNT_CH*n*/*m* 用于配置 RMT 通道内部的时钟分频器的分频系数，除 0 表示 256 分频外，其他分频数等同于寄存器 RMT_DIV_CNT_CH*n*/*m* 的值。时钟分频器可以通过配置 RMT_REF_CNT_RST_CH*n*/*m* 进行复位。时钟分频器的分频时钟可供计数器使用，见图 34-1。

34.3.4 发射器

> **说明：**
> 本小节以及后续小节所述的配置，均需要通过置位 RMT_CONF_UPDATE_CH*n*/*m* 的方式来进行更新，详情见第 34.3.6 小节。

34.3.4.1 普通发送模式

当 RMT_TX_START_CH*n* 置为 1 时，通道 *n* 的发射器开始从通道对应 RAM block 的起始地址，按照地址从低到高依次读取脉冲编码进行发送。当遇到结束标志（period 等于 0）时，发射器将结束发送返回空闲状态，并产生 RMT_CH*n*_TX_END_INT 中断。配置 RMT_TX_STOP_CH*n* 可以使发射器立刻停止发送并进入空闲状态。发射器空闲状态发送的电平由结束标志中的 level 段或者是 RMT_IDLE_OUT_LV_CH*n* 决定。用户可以配置 RMT_IDLE_OUT_EN_CH*n* 来选择这两种方式：

- RMT_IDLE_OUT_EN_CH*n* 置 0 时，发射器空闲状态发送的电平由结束标志中的 level 段决定；
- RMT_IDLE_OUT_EN_CH*n* 置 1 时，发射器空闲状态发送的电平由 RMT_IDLE_OUT_LV_CH*n* 决定。

34.3.4.2 乒乓发送模式

当发送的脉冲编码较多时，可通过置位 RMT_MEM_TX_WRAP_EN_CH*n* 使能通道 *n* 的乒乓模式。在乒乓操作模式下，发射器会循环从通道对应的 RAM 区域取出脉冲编码进行发送，直至遇到结束标识为止。例如，当

RMT_MEM_SIZE_CH*n* = 1 时，发射器将从 48 x *n* 地址开始发送，然后对应 RAM 的地址递增。发完 (48 x (*n* + 1) − 1) 地址的数据后，下次继续从 48 x *n* 地址开始递增发送数据，依此类推，遇到结束标识时停止发送。RMT_MEM_SIZE_CH*n* > 1 的情形下，乒乓操作同样适用。

每当发射器发送的脉冲编码数大于等于 RMT_TX_LIM_CH*n* 时，会产生 RMT_CH*n*_TX_THR_EVENT_INT 中断。在乒乓模式下，可以设置 RMT_TX_LIM_CH*n* 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 RMT_CH*n*_TX_THR_EVENT_INT 中断之后，可以回收已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

34.3.4.3 发送加载波

此外，发射器还可以对输出信号进行载波调制，置位 RMT_CARRIER_EN_CH*n* 可以使能该功能。载波的波形可配置。一个载波周期中高电平持续时间为 (RMT_CARRIER_HIGH_CH*n* + 1) 个 rmt_sclk 时钟周期，低电平持续的时间为 (RMT_CARRIER_LOW_CH*n* + 1) 个 rmt_sclk 时钟周期。置位 RMT_CARRIER_OUT_LV_CH*n* 时在输出信号高电平上加载波信号，清零 RMT_CARRIER_OUT_LV_CH*n* 时在输出信号低电平上加载波信号。同时，在进行载波调制时，载波可以一直加载在输出信号上，也可以仅加载在有效的脉冲编码（RAM 中的数据）上。通过配置 RMT_CARRIER_EFF_EN_CH*n* 寄存器，可以选择这两种模式：

- RMT_CARRIER_EFF_EN_CH*n* 置 0 时，在所有信号上加载波；
- RMT_CARRIER_EFF_EN_CH*n* 置 1 时，在有效信号上加载波。

34.3.4.4 持续发送模式

置位 RMT_TX_CONTI_MODE_CH*n* 可以使能发射器的持续发送功能。置位后，发射器会循环发送 RAM 中的脉冲编码。持续发送模式下：

- 如果遇到结束标志，则重新从该通道 RAM 中的第一个数据开始发送；
- 如果发送到 RAM 中的最后一个数都没有结束标志，存在两种情况。普通发送模式下（RMT_MEM_TX_WRAP_EN_CH*n* = 0），会因为 RAM 被读空，没有可以继续发送的数，而产生错误中断；乒乓发送模式下（RMT_MEM_TX_WRAP_EN_CH*n* = 1），会在发送到最后一个数据处回卷，重新开始发送第一个数据。

配置 RMT_TX_LOOP_CNT_EN_CH*n* 后，发射器每遇到一次结束标志，循环发送的次数会加 1。当该次数达到 RMT_TX_LOOP_NUM_CH*n* 设定的值时，会产生 RMT_CH*n*_TX_LOOP_INT 中断。如果置位 RMT_LOOP_STOP_EN_CH*n*，则发送会在产生 RMT_CH*n*_TX_LOOP_INT 中断后立即停止，否则会继续发送。持续发送模式下，如果遇到的结束标志类型是 period[14:0] 为 0，那么这个结束标志前一个数据的 period 需要满足：

$$6 \times T_{apb\_clk} + 12 \times T_{rmt\_sclk} < period \times T_{clk\_div} \quad (2)$$

其他数据的 period，满足 关系式 (1) 即可。

34.3.4.5 多通道同时发送

RMT 模块支持多通道同时发送，具体配置步骤如下所示：

1. 首先配置 RMT_TX_SIM_CH*n*，选择同步发送的通道；
2. 然后置位 RMT_TX_SIM_EN，使能发射器多个通道同步发送的功能；
3. 最后将所选同步发送通道的 RMT_TX_START_CH*n* 置为 1。

最后一个通道完成配置时，多个通道会同时启动发送。由于硬件条件的限制，不能保证两条通道完全同时开始发送，其时间间隔在 3 x T_{clk_div} 以内。

34.3.5 接收器

34.3.5.1 普通接收模式

通过配置 RMT_RX_EN_CH*m* 寄存器，可以选择是否启用接收模式：

- RMT_RX_EN_CH*m* 置 0 时，停止接收；
- RMT_RX_EN_CH*m* 置 1 时，接收器开始工作。

接收器会从信号的第一个跳变沿开始计数，并检测信号电平及其持续的时钟周期数，将其按照脉冲编码的格式存入 RAM 中。当信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时，接收器结束接收过程，返回空闲状态，并产生 RMT_CH*m*_RX_END_INT 中断。RMT_IDLE_THRES_CH*m* 的值需要大于输入信号的高电平或低电平的最大时钟周期数，否则接收器会将该信号误判为空闲信号而进入空闲状态。当接收数据存满了接收通道设置的 RAM 空间时，会停止接收，在非乒乓模式下会产生 RMT_CH*n*_ERR_INT 中断（由 RAM 满事件触发）。

34.3.5.2 乒乓接收模式

当接收的脉冲编码较多时，可通过置位 RMT_MEM_RX_WRAP_EN_CH*m* 使能通道 *m* 的乒乓模式。在乒乓操作模式下，接收器会将接收到的脉冲编码循环存入通道对应的 RAM 区域，当信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时，接收器结束接收过程，返回空闲状态，并产生 RMT_CH*m*_RX_END_INT 中断。例如，当 RMT_MEM_SIZE_CH*m* = 1 时，接收器将从 48 x *m* 地址开始接收，然后对应 RAM 的地址递增。收完 (48 x (*m* +1) − 1) 地址的数据后，下次继续从 48 x *m* 地址开始递增接收数据，以此类推，遇到信号在一个电平下持续的时钟周期数超过 RMT_IDLE_THRES_CH*m* 时停止接收。RMT_MEM_SIZE_CH*m* > 1 的情形下，乒乓操作同样适用。

每当接收器接收的脉冲编码数大于或等于 RMT_CH*m*_RX_LIM_REG 时，会产生 RMT_CH*m*_RX_THR_EVENT_INT 中断。在乒乓模式下，可以设置 RMT_CH*m*_RX_LIM_REG 为每个通道对应 RAM 空间的一半或几分之一。软件在检测到 RMT_CH*m*_RX_THR_EVENT_INT 中断之后，可以回收已使用过的 RAM 区域的脉冲编码，从而实现乒乓操作。

34.3.5.3 接收滤波

置位 RMT_RX_FILTER_EN_CH*m* 可使能通道 *m* 的接收器对输入信号进行滤波。滤波器的功能为连续采样输入信号，如果输入信号在连续 RMT_RX_FILTER _THRES_CH*m* 个 rmt_sclk 时钟周期内保持不变，则输入信号有效，否则输入信号无效。只有有效的输入信号才能通过滤波器。因此，滤波器会滤除脉冲宽度小于 RMT_RX_FILTER_THRES_CH*m* 个 rmt_sclk 时钟周期的线路毛刺。

34.3.5.4 接收去载波

此外，接收器还可以对输入信号或滤波后的信号进行去载波调制，置位 RMT_CARRIER_EN_CH*m* 可以使能该功能。去载波可以对高电平调制或低电平调制的载波进行滤除：

- RMT_CARRIER_OUT_LV_CH*m* 置 0 时，配置滤除低电平载波；
- RMT_CARRIER_OUT_LV_CH*m* 置 1 时，配置滤除高电平载波。

配置 RMT_CARRIER_HIGH_THRES_CH*m* 和 RMT_CARRIER_LOW_THRES_CH*m* 可设置去载波的高电平阈值和低电平阈值。当信号的高电平持续时间小于 RMT_CARRIER_HIGH_THRES_CH*m* 个 clk_div 分频时钟周期，或者信号的低电平持续时间小于 RMT_CARRIER _LOW_THRES_CH*m* 个 clk_div 分频时钟周期，则信号会被认为是载波而被滤除。

34.3.6 配置参数更新

RMT 的配置寄存器均需要配置各自通道的 RMT_CONF_UPDATE_CH*n*/*m* 位来更新进入各自通道，更新方法是向 RMT_CONF_UPDATE_CH*n*/*m* 写入高电平。发送通道和接收通道需要通过这种方法更新的配置参数见表 34-1。

表 34-1. 更新配置参数

配置寄存器	配置参数
发送通道	
RMT_CH*n*CONF0_REG	RMT_CARRIER_OUT_LV_CH*n*
	RMT_CARRIER_EN_CH*n*
	RMT_CARRIER_EFF_EN_CH*n*
	RMT_DIV_CNT_CH*n*
	RMT_IDLE_OUT_EN_CH*n*
	RMT_IDLE_OUT_LV_CH*n*
	RMT_TX_CONTI_MODE_CH*n*
RMT_CH*n*CARRIER_DUTY_REG	RMT_CARRIER_HIGH_CH*n*
	RMT_CARRIER_LOW_CH*n*
RMT_CH*n*_TX_LIM_REG	RMT_TX_LOOP_CNT_EN_CH*n*
	RMT_TX_LOOP_NUM_CH*n*
	RMT_TX_LIM_CH*n*
RMT_TX_SIM_REG	RMT_TX_SIM_EN
接收通道	
RMT_CH*m*CONF0_REG	RMT_CARRIER_OUT_LV_CH*m*
	RMT_CARRIER_EN_CH*m*
	RMT_IDLE_THRES_CH*m*
	RMT_DIV_CNT_CH*m*
RMT_CH*m*CONF1_REG	RMT_RX_FILTER_THRES_CH*m*
	RMT_RX_EN_CH*m*
RMT_CH*m*_RX_CARRIER_RM_REG	RMT_CARRIER_HIGH_THRES_CH*m*
	RMT_CARRIER_LOW_THRES_CH*m*
RMT_CH*m*_RX_LIM_REG	RMT_RX_LIM_CH*m*
RMT_REF_CNT_RST_REG	RMT_REF_CNT_RST_CH*m*

34.3.7 中断

- RMT_CH*n*/*m*_ERR_INT：当通道 *n*/*m* 发生读写数据不正确，或内存空满错误时，即触发此中断。
- RMT_CH*n*_TX_THR_EVENT_INT：发射器每发送 RMT_CH*n*_TX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*m*_RX_THR_EVENT_INT：接收器每接收 RMT_CH*m*_RX_LIM_REG 的数据，即触发一次此中断。
- RMT_CH*n*_TX_END_INT：当发射器停止发送信号时，即触发此中断。

- RMT_CH*m*_RX_END_INT：当接收器停止接收信号时，即触发此中断。
- RMT_CH*n*_TX_LOOP_INT：发射器处于循环发送模式时，当循环次数达到 RMT_TX_LOOP_NUM_CH*n* 的值后，会产生此中断。

34.4 寄存器列表

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

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

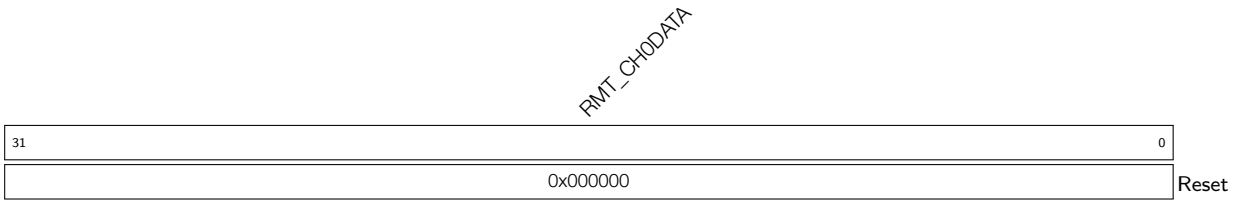
名称	描述	地址	访问
FIFO 读/写寄存器			
RMT_CH0DATA_REG	通道 0 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0000	HRO
RMT_CH1DATA_REG	通道 1 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0004	HRO
RMT_CH2DATA_REG	通道 2 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x0008	HRO
RMT_CH3DATA_REG	通道 3 通过 APB FIFO 进行读写操作时用到的数据寄存器	0x000C	HRO
配置寄存器			
RMT_CH0CONF0_REG	通道 0 的配置寄存器 0	0x0010	varies
RMT_CH1CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
RMT_CH2CONF0_REG	通道 2 的配置寄存器 0	0x0018	R/W
RMT_CH2CONF1_REG	通道 2 的配置寄存器 1	0x001C	varies
RMT_CH3CONF0_REG	通道 3 的配置寄存器 0	0x0020	R/W
RMT_CH3CONF1_REG	通道 3 的配置寄存器 1	0x0024	varies
RMT_SYS_CONF_REG	RMT APB 配置寄存器	0x0068	R/W
RMT_REF_CNT_RST_REG	RMT 时钟分频器复位寄存器	0x0070	WT
状态寄存器			
RMT_CH0STATUS_REG	通道 0 的状态寄存器	0x0028	RO
RMT_CH1STATUS_REG	通道 1 的状态寄存器	0x002C	RO
RMT_CH2STATUS_REG	通道 2 的状态寄存器	0x0030	RO
RMT_CH3STATUS_REG	通道 3 的状态寄存器	0x0034	RO
中断寄存器			
RMT_INT_RAW_REG	原始中断状态寄存器	0x0038	R/ WTC/ SS
RMT_INT_ST_REG	屏蔽中断状态寄存器	0x003C	RO
RMT_INT_ENA_REG	中断使能寄存器	0x0040	R/W
RMT_INT_CLR_REG	中断清零寄存器	0x0044	WT
载波占空比寄存器			
RMT_CH0CARRIER_DUTY_REG	通道 0 的占空比配置寄存器	0x0048	R/W
RMT_CH1CARRIER_DUTY_REG	通道 1 的占空比配置寄存器	0x004C	R/W
RMT_CH2_RX_CARRIER_RM_REG	通道 2 的去载波寄存器	0x0050	R/W
RMT_CH3_RX_CARRIER_RM_REG	通道 3 的去载波寄存器	0x0054	R/W
TX 事件配置寄存器			
RMT_CH0_TX_LIM_REG	通道 0 的 TX 事件配置寄存器	0x0058	varies
RMT_CH1_TX_LIM_REG	通道 1 的 TX 事件配置寄存器	0x005C	varies
RMT_TX_SIM_REG	RMT TX 同步发送寄存器	0x006C	R/W

名称	描述	地址	访问
RX 事件配置寄存器			
RMT_CH2_RX_LIM_REG	通道 2 RX 事件配置寄存器	0x0060	R/W
RMT_CH3_RX_LIM_REG	通道 3 RX 事件配置寄存器	0x0064	R/W
版本寄存器			
RMT_DATE_REG	版本控制寄存器	0x00CC	R/W

34.5 寄存器

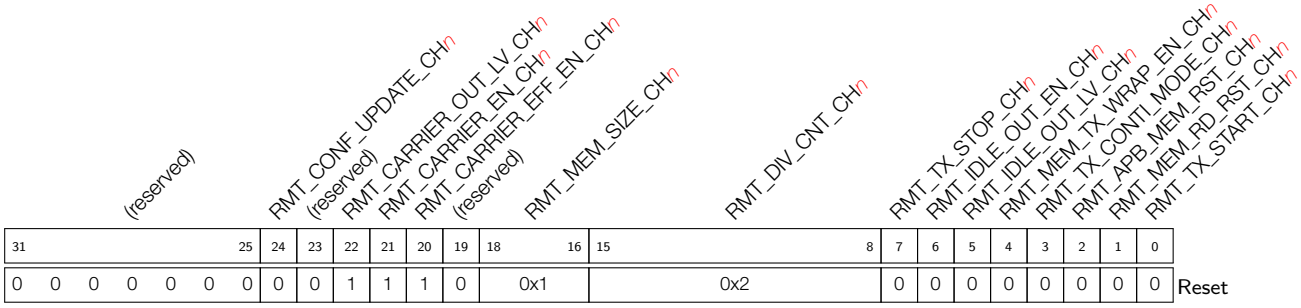
本小节的所有地址均为相对于红外遥控基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 34.1. RMT_CH*n*DATA_REG (*n*: 0-3) (0x0000+0x4n*)**



RMT_CH*n*DATA 通道 *n* 通过 APB FIFO 进行读写操作时用到的数据寄存器。(HRO)

Register 34.2. RMT_CH*n*CONF0_REG (*n*: 0-1) (0x0010+0x4n*)**



RMT_TX_START_CH*n* 配置是否使能通道 *n* 开始发送数据。

0：无效

1：使能

(WT)

RMT_MEM_RD_RST_CH*n* 配置是否复位通道 *n* 通过发射器访问的 RAM 读地址。

0：无效

1：复位

(WT)

RMT_APB_MEM_RST_CH*n* 配置是否复位通道 *n* 通过 APB FIFO 访问的读/写 RAM 地址。

0：无效

1：复位

(WT)

RMT_TX_CONTI_MODE_CH*n* 配置是否使能通道 *n* 的持续发送模式。

0：无效

1：使能

在这种模式下，发射器从第一个数据开始发送，如果遇到结束标志，则再次发送第一个数据；如果没有遇到结束标志，发送完最后一个数据后，回卷到第一个数据再次继续发送。

(R/W)

RMT_MEM_TX_WRAP_EN_CH*n* 配置是否使能通道 *n* 的乒乓发送模式。

0：无效

1：使能

在这种模式下，如果待发送的数据长度大于该通道的 RAM Block 长度，则发射器将继续从第一个数据开始循环发送。

(R/W)

RMT_IDLE_OUT_LV_CH*n* 配置通道 *n* 在空闲模式下的输出信号电平。(R/W)

RMT_IDLE_OUT_EN_CH*n* 配置是否使能通道 *n* 在空闲状态下的输出模式。

0：无效

1：使能

(R/W)

见下页...

Register 34.2. RMT_CH*n*CONF0_REG (*n*: 0-1) (0x0010+0x4n*)**

接上页...

RMT_TX_STOP_CH*n* 配置通道 *n* 的发射器是否停止发送数据。

0：无效

1：停止

(R/W/SC)

RMT_DIV_CNT_CH*n* 配置通道 *n* 的时钟分频系数。

测量单位：rmt_sclk

(R/W)

RMT_MEM_SIZE_CH*n* 配置通道 *n* 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EFF_EN_CH*n* 配置是否仅在发送数据状态下对输出信号载波调制。

0：配置通道 *n* 对发送数据状态和空闲状态均加载载波

1：配置通道 *n* 仅在发送数据状态下对输出信号载波调制

仅在 RMT_CARRIER_EN_CH*n* 为 1 时有效。

(R/W)

RMT_CARRIER_EN_CH*n* 配置是否使能通道 *n* 对输出信号进行载波调制。

0：不使能

1：使能

(R/W)

RMT_CARRIER_OUT_LV_CH*n* 配置通道 *n* 的载波调制方式。

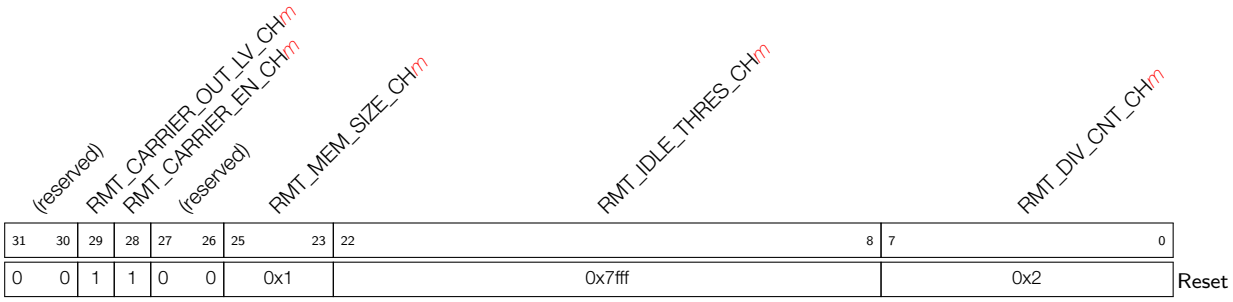
0：载波加载在低电平上

1：载波加载在高电平上

(R/W)

RMT_CONF_UPDATE_CH*n* 通道 *n* 的同步位。(WT)

Register 34.3. RMT_CH*m*CONF0_REG (*m*: 2-3) (0x0008+0x8m*)**



RMT_DIV_CNT_CH*m* 配置通道 *m* 的时钟分频时钟。

测量单位：rmt_sclk

(R/W)

RMT_IDLE_THRES_CH*m* 配置接收阈值。

接收器长时间检测不到信号变化，且持续的时间大于 RMT_IDLE_THRES_CH*m* 的值，则接收器停止接收过程。

测量单位：clk_div

(R/W)

RMT_MEM_SIZE_CH*m* 配置通道 *m* 可用的最大 RAM Block 数量。(R/W)

RMT_CARRIER_EN_CH*m* 配置是否使能通道 *m* 对输出信号进行载波调制和解调。

0：不使能

1：使能

(R/W)

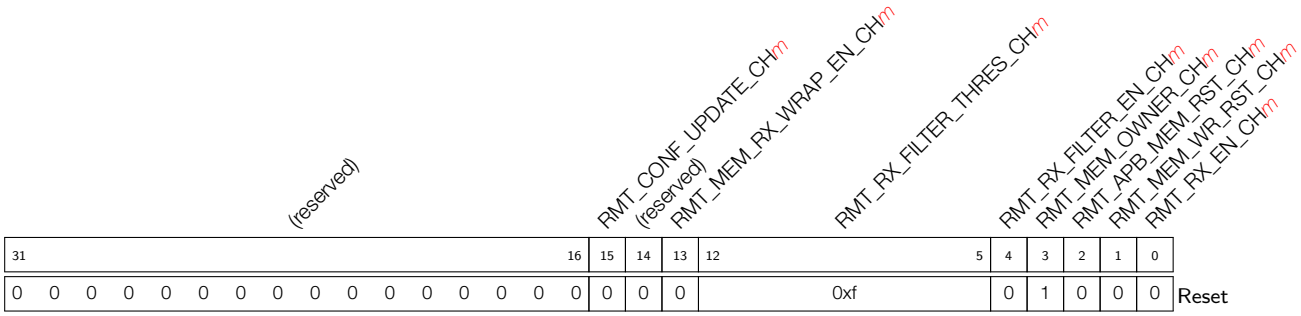
RMT_CARRIER_OUT_LV_CH*m* 配置通道 *m* 的载波调制或解调方式。

0：在低电平上进行载波调制（发射器）或解调（接收器）

1：在高电平上进行载波调制（发射器）或解调（接收器）

(R/W)

Register 34.4. RMT_CH*m*CONF1_REG (*m*: 2-3) (0x000C+0x8m*)**



RMT_RX_EN_CH*m* 配置是否使能通道 *m* 的接收器，使接收器开始接收数据。

0：不使能

1：使能

(R/W)

RMT_MEM_WR_RST_CH*m* 配置是否复位通道 *m* 通过接收器访问的 RAM 写地址。

0：无效

1：复位

(WT)

RMT_APB_MEM_RST_CH*m* 配置是否复位通道 *m* 通过 APB FIFO 访问的读/写 RAM 地址。

0：无效

1：复位

(WT)

RMT_MEM_OWNER_CH*m* 配置通道 *m* 的 RAM 使用权。

0：APB 总线有权使用该通道的 RAM

1：接收器有权使用该通道的 RAM

(R/W/SC)

RMT_RX_FILTER_EN_CH*m* 配置是否使能通道 *m* 的接收器滤波功能。

0：不使能

1：使能

(R/W)

RMT_RX_FILTER_THRES_CH*m* 配置是否在接收模式下，忽略宽度小于 RMT_RX_FILTER_THRES_CH*m* 个 rmt_sclk 周期的输入脉冲。

0：无效

1：忽略

(R/W)

RMT_MEM_RX_WRAP_EN_CH*m* 配置是否使能通道 *m* 的乒乓接收模式。

0：不使能

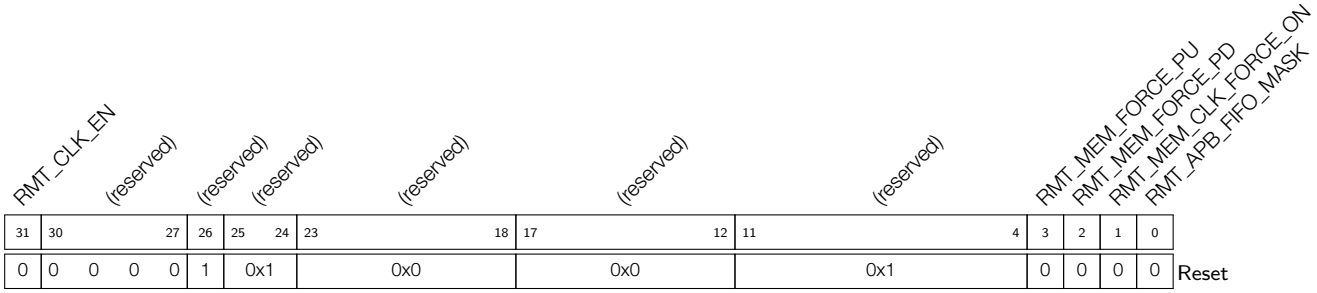
1：使能

在这种模式下，如果待接收的数据长度大于该通道的 RAM Block 长度，则接收器将继续把接收数据存入 RAM 的第一个地址，依次循环。

(R/W)

RMT_CONF_UPDATE_CH*m* 通道 *m* 的同步位。(WT)

Register 34.5. RMT_SYS_CONF_REG (0x0068)



RMT_APB_FIFO_MASK 配置 RAM 的访问模式。

0：通过 FIFO 访问 RAM

1：直接访问 RAM

(R/W)

RMT_MEM_CLK_FORCE_ON 配置是否使能 RMT 的 RAM 时钟。

0：不使能

1：使能

(R/W)

RMT_MEM_FORCE_PD 配置是否关闭 RMT RAM。

0：无效

1：关闭

(R/W)

RMT_MEM_FORCE_PU 配置是否禁用 RMT RAM 的 Light-sleep 低功耗模式。

0：RMT 处于 Light-sleep 模式时，关闭 RMT RAM

1：禁用 RMT RAM 的 Light-sleep 低功耗模式

(R/W)

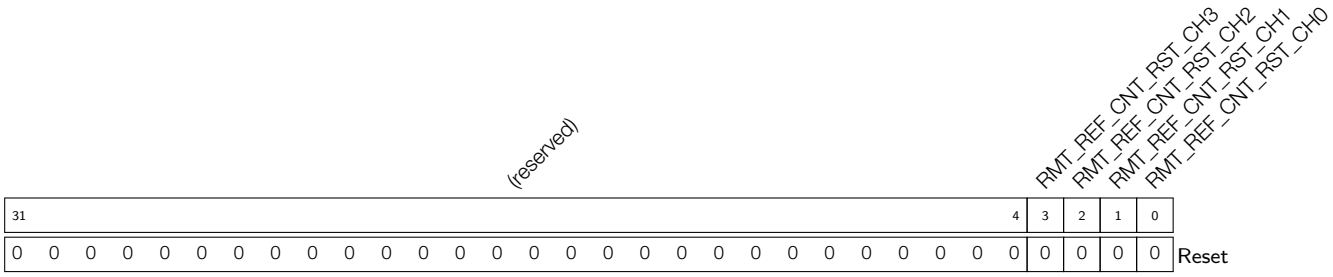
RMT_CLK_EN 配置是否使能 RMT 寄存器的时钟门控。

0：关闭寄存器的驱动时钟

1：打开寄存器的驱动时钟

(R/W)

Register 34.6. RMT_REF_CNT_RST_REG (0x0070)



RMT_REF_CNT_RST_CH*n* 配置是否复位通道 *n* 的时钟分频器。

0：无效

1：复位

(WT)

RMT_REF_CNT_RST_CH*m* 配置是否复位通道 *m* 的时钟分频器。

0：无效

1：复位

(WT)

Register 34.7. RMT_CH*n*STATUS_REG (*n*: 0-1) (0x0028+0x4n*)**

RMT_MEM_RADDR_EX_CH*n* 表示通道 *n* 发射器使用 RAM 时的地址偏移量。(RO)

RMT_STATE_CH*n* 表示通道 *n* 的 FSM 状态。(RO)

RMT_APB_MEM_WADDR_CH*n* 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

RMT_APB_MEM_RD_ERR_CH*n* 表示 RMT 使用 APB 总线进行读 RAM 操作时，偏移地址是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

RMT_MEM_EMPTY_CH*n* 表示是否发送的数据长度是否溢出 RAM Block，且乒乓模式未启用。

0：未溢出

1：溢出

(RO)

RMT_APB_MEM_WR_ERR_CH*n* 表示 RMT 使用 APB 总线进行写 RAM 操作时，偏移地址是否溢出 RAM Block。

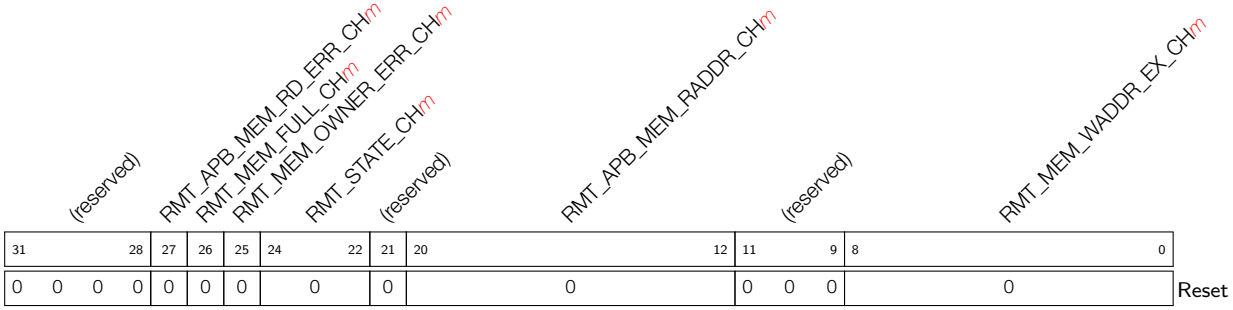
0：未溢出

1：溢出

(RO)

RMT_APB_MEM_RADDR_CH*n* 表示 RMT 使用 APB 总线读 RAM 时的地址偏移量。(RO)

Register 34.8. RMT_CH*m*STATUS_REG (*m*: 2-3) (0x0028+0x4m*)**



RMT_MEM_WADDR_EX_CH*m* 表示通道 *m* 接收器使用 RAM 时的地址偏移量。(RO)

RMT_APB_MEM_RADDR_CH*m* 表示 RMT 使用 APB 总线访问 RAM 时的地址偏移量。(RO)

RMT_STATE_CH*m* 表示通道 *m* 的 FSM 状态。(RO)

RMT_MEM_OWNER_ERR_CH*m* 表示 RAM Block 使用权是否发生错误。

0：无错误

1：发生错误

(RO)

RMT_MEM_FULL_CH*m* 表示接收器接收的数据长度是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

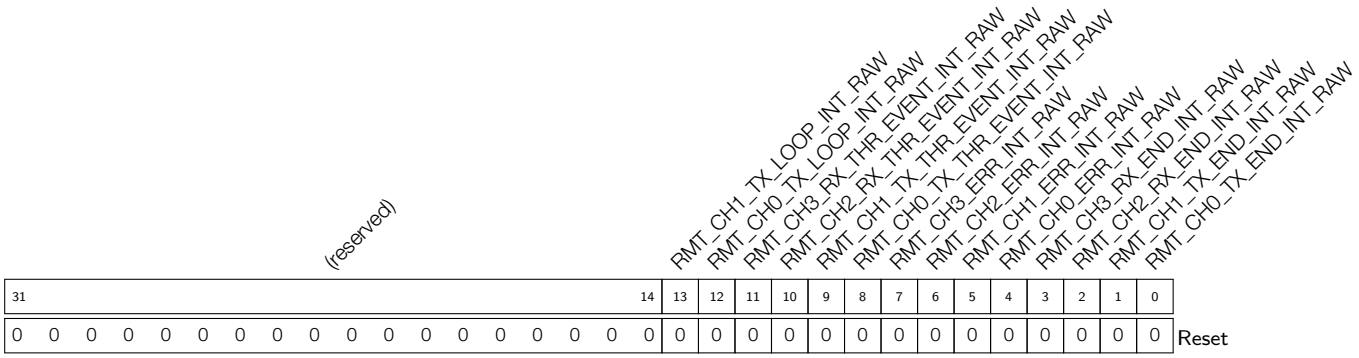
RMT_APB_MEM_RD_ERR_CH*m* 表示 RMT 使用 APB 总线执行 RAM 读操作时，偏移地址是否溢出 RAM Block。

0：未溢出

1：溢出

(RO)

Register 34.9. RMT_INT_RAW_REG (0x0038)



RMT_CH*n*_TX_END_INT_RAW RMT_CH*n*_TX_END_INT 的原始中断状态。发送完成时触发。(R/WTC/SS)

RMT_CH*m*_RX_END_INT_RAW RMT_CH*n*_RX_END_INT 的原始中断状态。接收完成时触发。(R/WTC/SS)

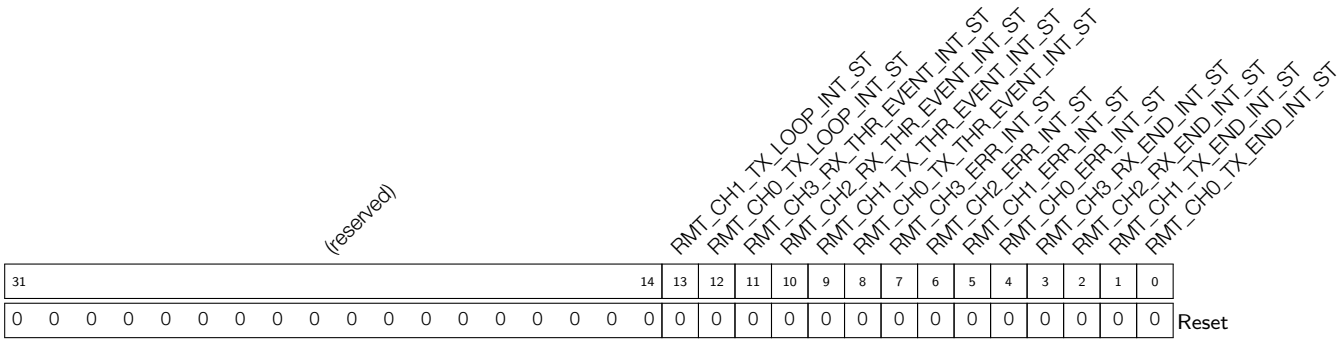
RMT_CH*n*/*m*_ERR_INT_RAW RMT_CH*n*/*m*_ERR_INT 的原始中断状态。发生错误时触发。(R/WTC/SS)

RMT_CH*n*_TX_THR_EVENT_INT_RAW RMT_CH*n*_TX_THR_EVENT_INT 的原始中断状态。发送器发送多于配置值时触发。(R/WTC/SS)

RMT_CH*m*_RX_THR_EVENT_INT_RAW RMT_CH*m*_RX_THR_EVENT_INT 的原始中断状态。接收器接收多于配置值时触发。(R/WTC/SS)

RMT_CH*n*_TX_LOOP_INT_RAW RMT_CH*n*_TX_LOOP_INT 的原始中断状态。循环计数次数到达配置阈值时触发。(R/WTC/SS)

Register 34.10. RMT_INT_ST_REG (0x003C)



RMT_CH*n*_TX_END_INT_ST RMT_CH*n*_TX_END_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_RX_END_INT_ST RMT_CH*n*_RX_END_INT 的屏蔽中断状态。(RO)

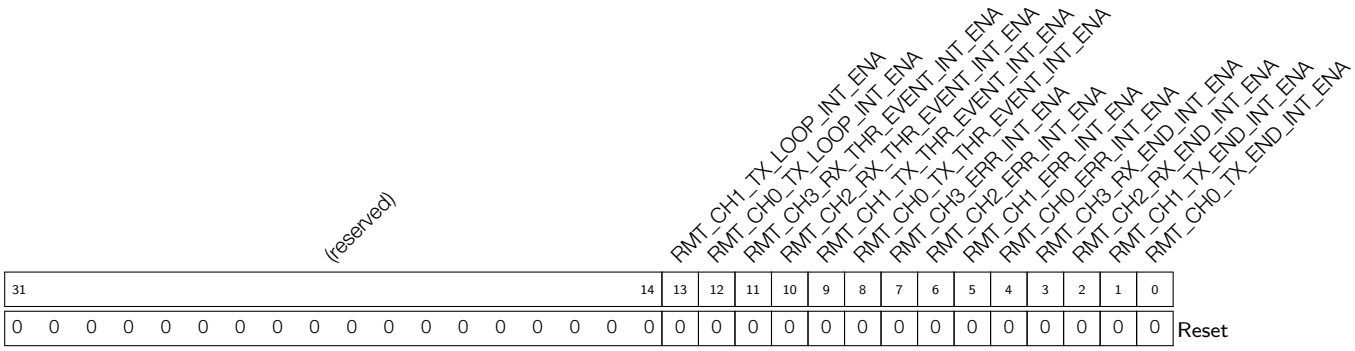
RMT_CH*n*/*m*_ERR_INT_ST RMT_CH*n*/*m*_ERR_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_TX_THR_EVENT_INT_ST RMT_CH*n*_TX_THR_EVENT_INT 的屏蔽中断状态。(RO)

RMT_CH*m*_RX_THR_EVENT_INT_ST RMT_CH*m*_RX_THR_EVENT_INT 的屏蔽中断状态。(RO)

RMT_CH*n*_TX_LOOP_INT_ST RMT_CH*n*_TX_LOOP_INT 的屏蔽中断状态。(RO)

Register 34.11. RMT_INT_ENA_REG (0x0040)



RMT_CH*n*_TX_END_INT_ENA 写 1 使能 RMT_CH*n*_TX_END_INT 。(R/W)

RMT_CH*m*_RX_END_INT_ENA 写 1 使能 RMT_CH*n*_RX_END_INT。(R/W)

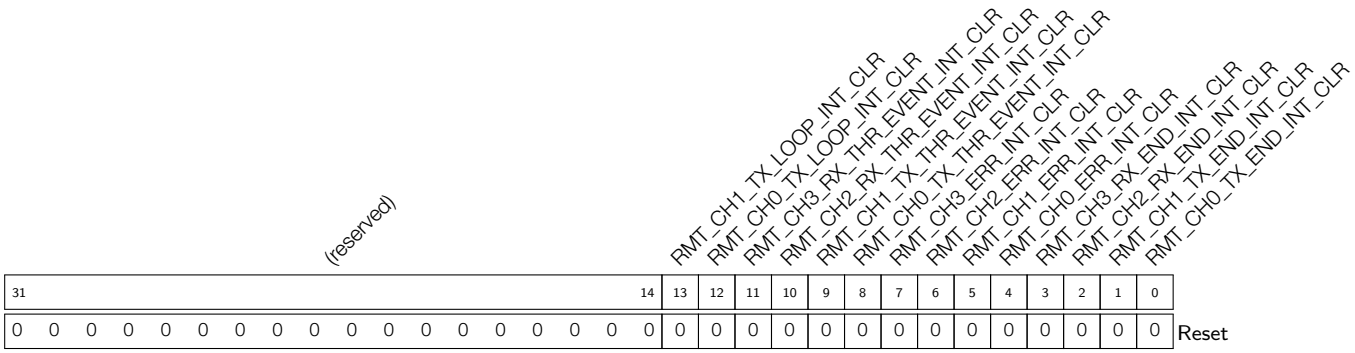
RMT_CH*n*/*m*_ERR_INT_ENA 写 1 使能 RMT_CH*n*/*m*_ERR_INT。(R/W)

RMT_CH*n*_TX_THR_EVENT_INT_ENA 写 1 使能 RMT_CH*n*_TX_THR_EVENT_INT。(R/W)

RMT_CH*m*_RX_THR_EVENT_INT_ENA 写 1 使能 RMT_CH*m*_RX_THR_EVENT_INT。(R/W)

RMT_CH*n*_TX_LOOP_INT_ENA 写 1 使能 RMT_CH*n*_TX_LOOP_INT。(R/W)

Register 34.12. RMT_INT_CLR_REG (0x0044)



RMT_CH*n*_TX_END_INT_CLR 写 1 清除 RMT_CH*n*_TX_END_INT 。(WT)

RMT_CH*m*_RX_END_INT_CLR 写 1 清除 RMT_CH*n*_RX_END_INT。(WT)

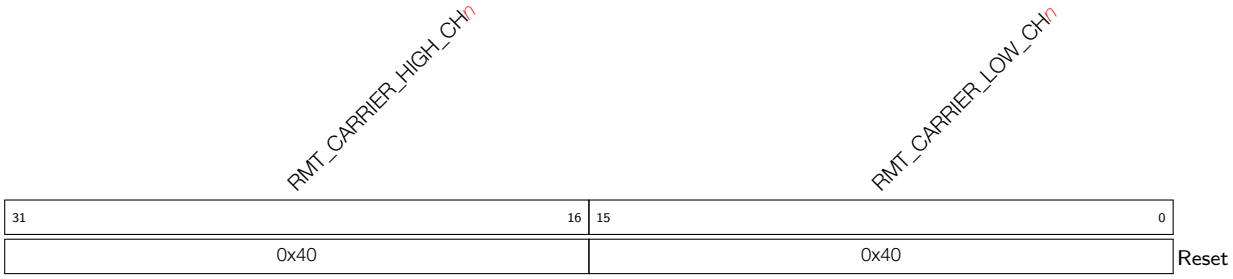
RMT_CH*n*/*m*_ERR_INT_CLR 写 1 清除 RMT_CH*n*/*m*_ERR_INT。(WT)

RMT_CH*n*_TX_THR_EVENT_INT_CLR 写 1 清除 RMT_CH*n*_TX_THR_EVENT_INT。(WT)

RMT_CH*m*_RX_THR_EVENT_INT_CLR 写 1 清除 RMT_CH*m*_RX_THR_EVENT_INT。(WT)

RMT_CH*n*_TX_LOOP_INT_CLR 写 1 清除 RMT_CH*n*_TX_LOOP_INT。(WT)

Register 34.13. RMT_CH*n*CARRIER_DUTY_REG (*n*: 0-1) (0x0048+0x4n*)**



RMT_CARRIER_LOW_CH*n* 配置通道 *n* 载波的低电平时钟周期。

测量单位：rmt_sclk

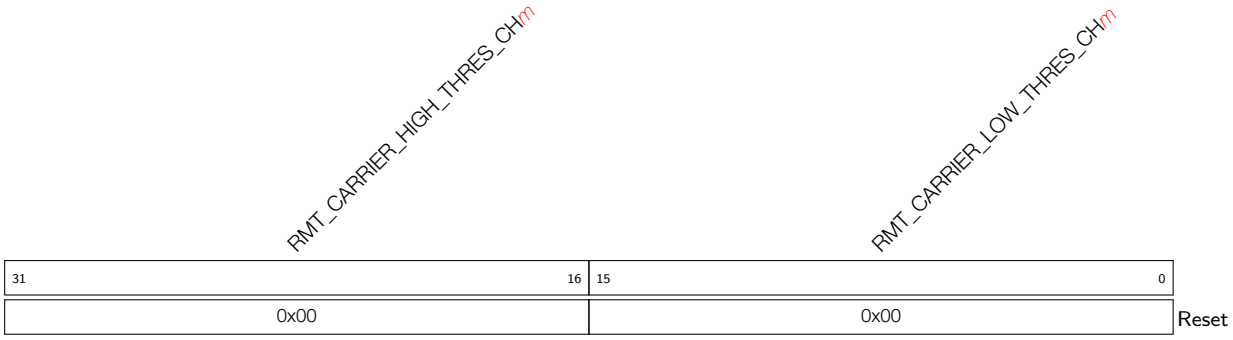
(R/W)

RMT_CARRIER_HIGH_CH*n* 配置通道 *n* 载波的高电平时钟周期。

测量单位：rmt_sclk

(R/W)

Register 34.14. RMT_CH*m*_RX_CARRIER_RM_REG (*m*: 2-3) (0x0048+0x4m*)**



RMT_CARRIER_LOW_THRES_CH*m* 配置载波调制模式下通道 *m* 的低电平周期。

载波调制模式下，通道 *m* 低电平周期为 RMT_CARRIER_LOW_THRES_CH*m* + 1。

测量单位：clk_div

(R/W)

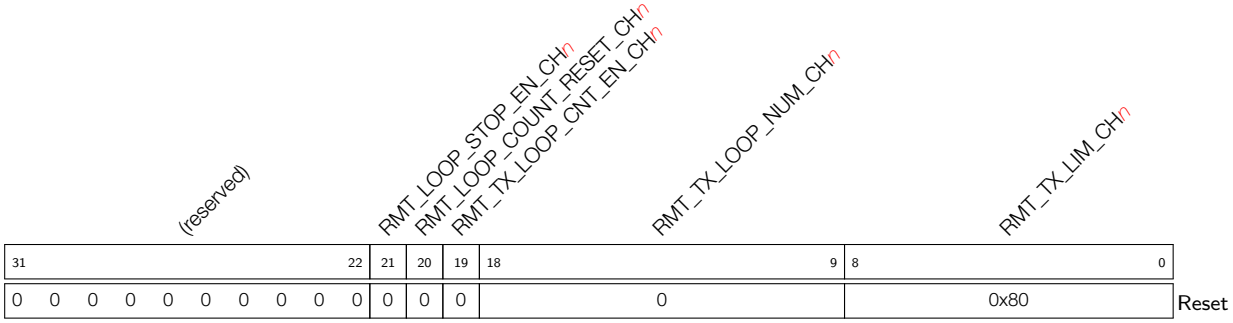
RMT_CARRIER_HIGH_THRES_CH*m* 配置载波调制模式下通道 *m* 的高电平周期。

载波调制模式下，通道 *m* 高电平周期为 RMT_CARRIER_HIGH_THRES_CH*m* + 1。

测量单位：clk_div

(R/W)

Register 34.15. RMT_CH*n*_TX_LIM_REG (*n*: 0-1) (0x0058+0x4n*)**



RMT_TX_LIM_CH*n* 配置通道 *n* 发送脉冲编码数量的上限值。(R/W)

RMT_TX_LOOP_NUM_CH*n* 配置持续发送模式下最大循环发送次数。(R/W)

RMT_TX_LOOP_CNT_EN_CH*n* 配置是否使能循环次数计数。

0：无效

1：使能

(R/W)

RMT_LOOP_COUNT_RESET_CH*n* 配置是否重置持续发送模式下的循环计数器。

0：无效

1：重置

(WT)

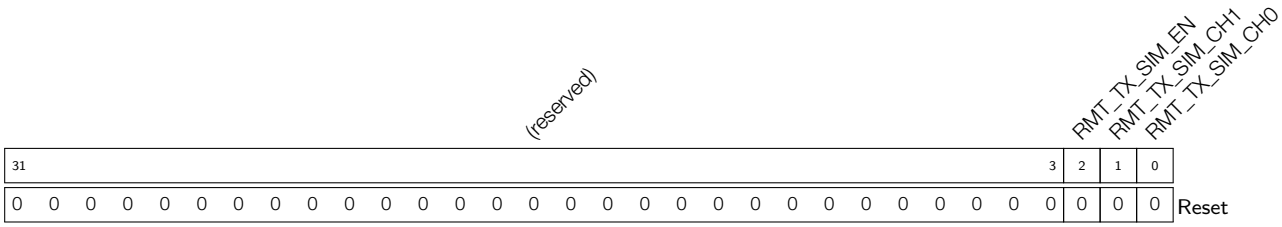
RMT_LOOP_STOP_EN_CH*n* 配置是否在通道 *n* 的循环计数器记到循环数后使能循环发送停止功能。

0：无效

1：使能

(R/W)

Register 34.16. RMT_TX_SIM_REG (0x006C)



RMT_TX_SIM_CH*n* 配置是否使能通道 *n* 与其它启用的通道同步开始发送数据。

0: 不使能

1: 使能

(R/W)

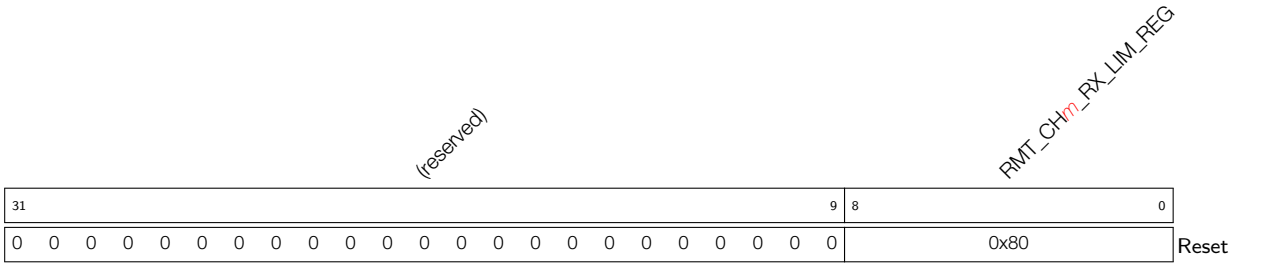
RMT_TX_SIM_EN 配置是否使能多个通道开始同步发送数据。

0: 不使能

1: 使能

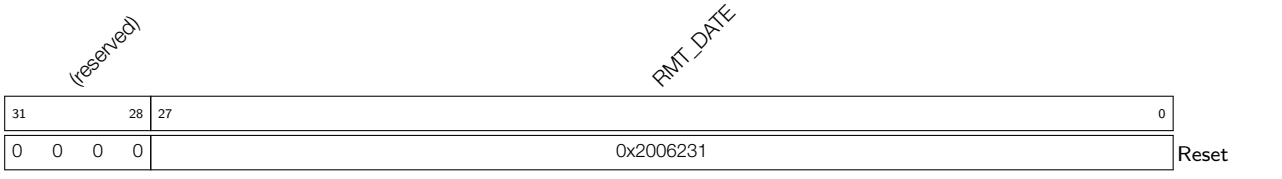
(R/W)

Register 34.17. RMT_CH*m*_RX_LIM_REG (*m*: 2-3) (0x0058+0x4**m*)



RMT_CH*m*_RX_LIM_REG 配置通道 *m* 最大可接收的脉冲编码数量。(R/W)

Register 34.18. RMT_DATE_REG (0x00CC)



RMT_DATE 版本控制寄存器。(R/W)

35 并行 IO 控制器 (PARL_IO)

35.1 概况

ESP32-H2 包含一个并行 IO 控制器 (PARLIO)，支持通过 GDMA 在并行总线上实现外部设备和内部存储器之间的数据通信。它由 TX 和 RX 两个子模块组成，分别作为发送器和接收器。PARLIO 支持全双工通信。

鉴于 IO 数据的灵活性，PARLIO 可用作通用接口，连接各种外围设备。例如可以把 SPI 作为主机，PARLIO 作为从机，实现点对点传输。详细的应用实例，请参考小节 35.7。

35.2 术语

为了更好地说明 PARLIO 的功能，本章使用了以下术语。

RX 模块	PARLIO 子模块，从外部并行总线接收数据并将其存储在内部存储器中。
TX 模块	PARLIO 子模块，将数据从内部存储器传输到外部并行总线上。
RXD	从 RX 模块的 IO 接口接收的并行数据。
TXD	从 TX 模块的 IO 接口发送的并行数据。
帧	传输数据的单位，从设置 START 信号时起，到收到 EOF 信号止。
自由运行时钟	持续翻转的时钟。若非自由运行时钟，则时钟只在收到有效数据的期间翻转，其余时间停止翻转。
GDMA SUC EOF	GDMA 帧传输成功结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧正确，且数据接收结束。
GDMA ERR EOF	GDMA 帧传输错误结束的信号。GDMA 接收到此信号时，触发一个 GDMA 中断，表明当前帧错误，且数据接收结束。
CDC	跨时钟域处理 (Clock Domain Crossing)。

35.3 特性

PARLIO 模块具备以下特性：

- 支持多种时钟源可选：
  - 包括外部 IO 时钟 PAD_CLK_TX/RX、内部系统时钟 XTAL_CLK、PLL_F96M_CLK 和 RC_FAST_CLK
  - 支持最大时钟频率 40 MHz
  - 支持对时钟频率进行整数分频
- 支持将传输数据总线位宽配置为 1/2/4/8 位
- 支持 8 位全双工传输
- 总线位宽为 1/2/4 位时，支持在一个字节范围内对比特数据顺序进行翻转
- 包含用于接收 IO 并行数据的 RX 子模块：
  - 支持对输出时钟进行门控
  - 支持 RX 子模块输入时钟和输出时钟分别取反
  - 支持多种接收模式

- 支持配置 GDMA SUC EOF 信号生成模式
- 支持配置外部使能信号的 IO 管脚

- 包含用于发送 IO 并行数据的 TX 子模块：
  - 支持对输出时钟进行门控
  - 支持 TX 子模块输入时钟和输出时钟分别取反
  - 支持有效信号输出
  - 支持配置总线空闲时数值

35.4 系统架构

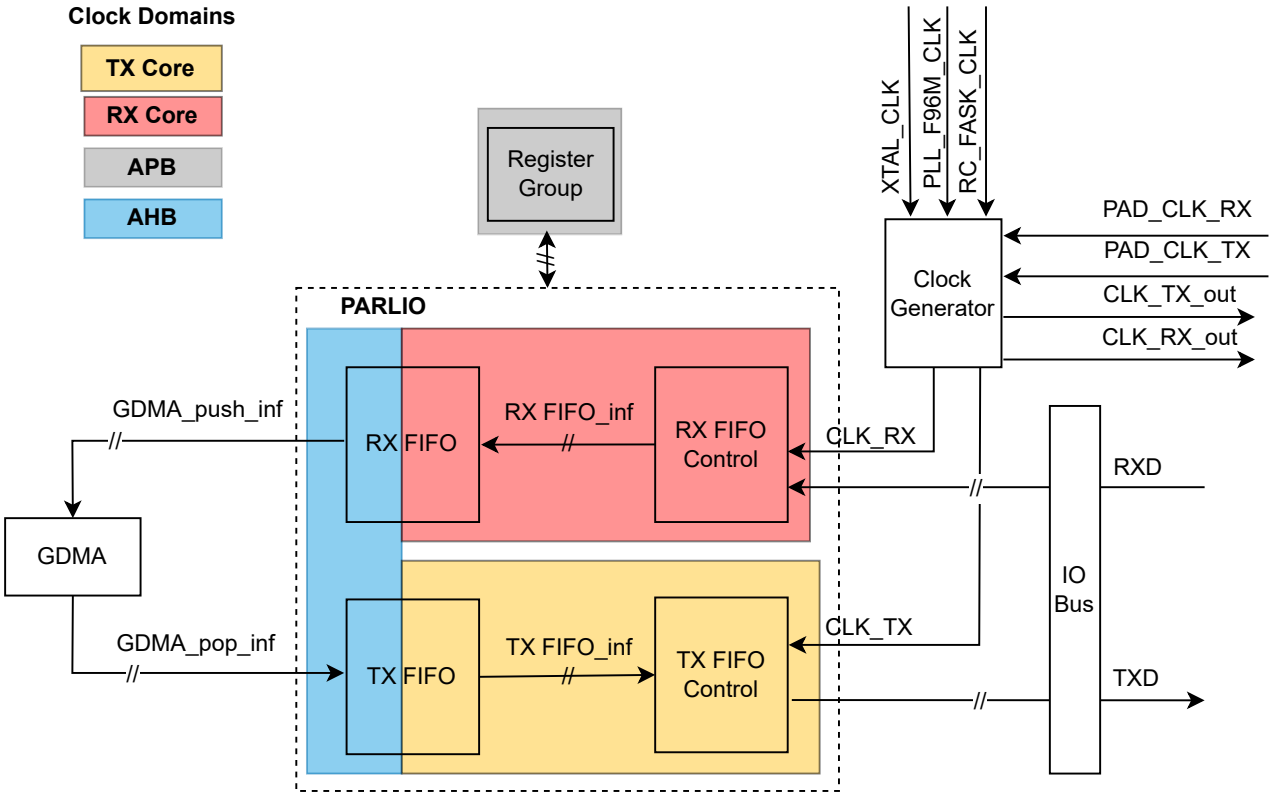


图 35-1. PARLIO 系统框图

图 35-1 展示了 PARLIO 的系统内部架构。除了 RX 模块和 TX 模块外，PARLIO 还包括一组状态配置寄存器。

RX 模块将 RXD 转换为异步 FIFO 接口，异步 FIFO 将 RXD 同步到 AHB 时钟域，最后 RXD 被转换为标准的 GDMA 接口，并被传输到内部存储器。

TX 模块通过 GDMA 从内部存储器获取数据，并将 GDMA 接口转换为异步 FIFO 接口。异步 FIFO 将数据同步到 TX Core 时钟域，最后将其转换为 TXD，用于并行 IO 总线输出。

35.5 功能描述

35.5.1 时钟生成器

PARLIO 模块有四个输入时钟域，即 RX Core、TX Core、AHB 和 APB，如图 35-1 所示。

状态配置寄存器组位于 APB 时钟域内。

GDMA 接口逻辑位于 AHB 时钟域内。

RX Core 和 TX Core 时钟域各有四个时钟源可供选择，分别为内部系统时钟源 XTAL_CLK、RC_FAST_CLK、PLL_F96M_CLK 和外部时钟源 (PAD_CLK_TX/RX)，如图 35-2 所示。配置 PCR_PARL_CLK_RX_SEL 和 PCR_PARL_CLK_TX_SEL 选择时钟源，配置 PCR_PARL_CLK_RX_DIV_NUM 和 PCR_PARL_CLK_TX_DIV_NUM 对时钟进行分频，分频系数最大可配置为 $(2^{16} - 1)$。

RX 模块和 TX 模块支持对输入时钟进行取反，同时其工作时钟可以输出到 IO，并在输出前支持对输出时钟进行取反。此外 RX 模块和 TX 模块还支持对输出时钟进行门控。

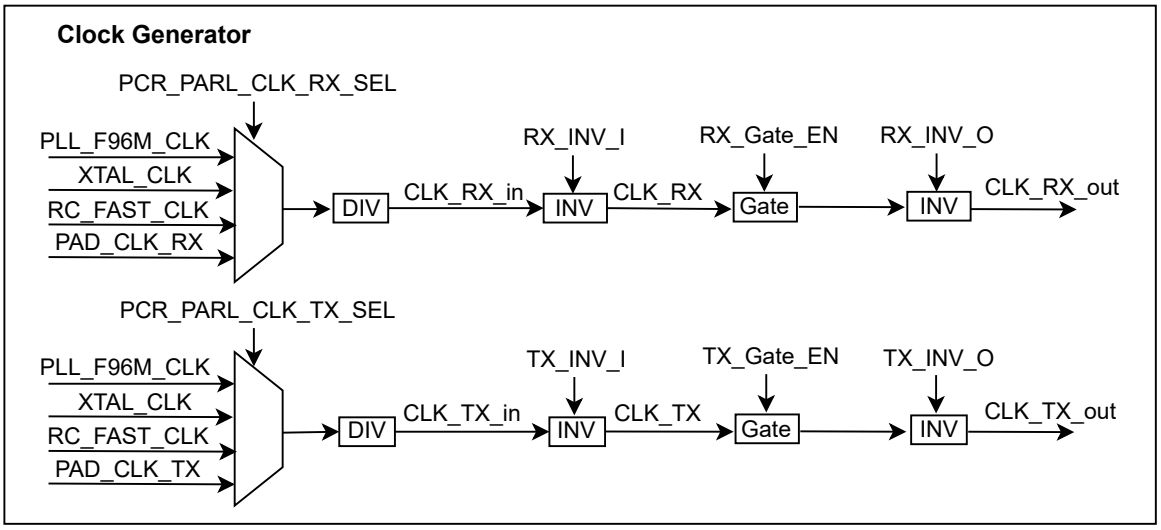


图 35-2. PARLIO 时钟生成

35.5.2 时钟复位使用限制

由于 PARLIO 模块的通用性，其 PAD 时钟 (PAD_CLK_TX/RX) 可能来自不同的主机（外部设备或内部时钟源），这些时钟可能是自由运行时钟，也可能不是。如果外部运行时钟不是自由运行时钟，PARLIO 内部的一些控制信号将无法进行 CDC，因此对操作过程会有一定限制。

1. 异步 FIFO 复位时，需要两个时钟周期才能从 AHB 时钟域同步到 Core 时钟域。因此如果需要在使用非自由运行时钟时复位 AHB 时钟域，需保证提前两个时钟周期进行复位同步。具体的操作步骤见下表。

表 35-2. 时钟限制下复位 AHB 时钟域操作步骤

时钟限制		具体操作
当前帧传输时钟	下一帧传输时钟	
自由运行时钟	非自由运行时钟	在切换到非自由运行时钟之前复位下一帧传输，并在复位完成后配置时钟切换。
非自由运行时钟	自由运行时钟	只需确保在复位和正式开始传输期间有两个时钟周期的间隔。
非自由运行时钟	非自由运行时钟	如需复位下一帧传输，需要先切换到内部的自由运行时钟，复位完成后再切换回实际时钟。

2. 由于非自由运行时钟造成的限制，PARL_IO_RX_START 和 PARL_IO_TX_START 无法进行 CDC，所以在操作时必须等到 PARL_IO_RX_START 和 PARL_IO_TX_START 稳定后再开始数据传输，否则可能会发生亚稳态的问题。

   RX 模块中的具体操作步骤为：

   - 清除 PCR_PARL_CLK_RX_EN 关闭 RX Core 时钟域；
   - 向 PARL_IO_RX_START 写 1；
   - 置位 PCR_PARL_CLK_RX_EN 打开 RX Core 时钟域；
   - 操作外部设备，使其开始发送数据；
   - 清除 PCR_PARL_CLK_RX_EN 关闭 RX Core 时钟域；
   - 向 PARL_IO_TX_START 写 0。

   TX 模块中的具体操作步骤为：

   - 清除 PCR_PARL_CLK_TX_EN 关闭 TX Core 时钟域；
   - 向 PARL_IO_TX_START 写 1；
   - 置位 PCR_PARL_CLK_TX_EN 打开 TX Core 时钟域；
   - 操作外部设备，使其开始接收数据；
   - 清除 PCR_PARL_CLK_TX_EN 关闭 TX Core 时钟域；
   - 向 PARL_IO_TX_START 写 0。

3. 复位需遵循以下要求：

   - 芯片启动时，时钟复位应遵循以下顺序：

     (a) 复位 APB 时钟域；

     (b) 复位 AHB 时钟域；

     (c) 复位 Core 时钟域。

   - 帧间传输需要复位 Core 时钟域和异步 FIFO。

35.5.3 主从机模式

RX 模块和 TX 模块均支持主从机模式。

当 TX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。TX 模块在时钟的上升沿驱动 TXD。

当 TX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 35-3. 时钟限制下 TX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求。
非自由运行时钟	正向波形（图 35-3）	主机需要在下降沿时采集。
非自由运行时钟	负向波形（图 35-4）	主机在输出前需要对原时钟进行取反，将其转换成正向波形。

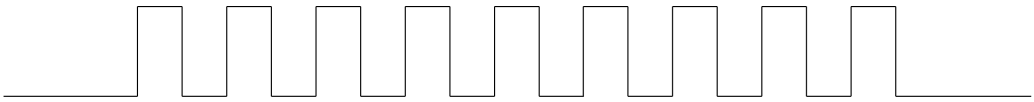


图 35-3. 主机时钟正向波形

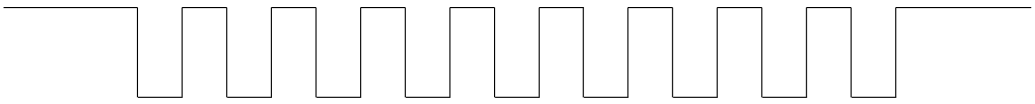


图 35-4. 主机时钟负向波形

当 RX 模块作为主机时，必须将时钟源设置为内部自由运行时钟。RX 模块在时钟的上升沿驱动 RXD。

当 RX 模块作为从机时，可能出现的三种情况及其具体要求如下表所示：

表 35-4. 时钟限制下 RX 模块从机模式操作要求

时钟条件		具体要求
主机发送时钟	时钟波形	
自由运行时钟	任意波形	对主机的驱动时钟采集边沿没有要求，采集的有效数据以外部使能信号为准。
非自由运行时钟	正向波形（图 35-3）	主机需在上升沿驱动数据，RX 模块需在下降沿采集数据（即对主时钟进行翻转）。
非自由运行时钟	负向波形（图 35-4）	主机需在下降沿驱动数据，RX 模块需在上升沿采集数据（即使用原始主时钟）

35.5.4 RX 模块接收模式

PARLIO 的 RX 模块有八种接收模式，可根据使能信号规格分为三大类：

- 电平使能模式：接收的数据由外部信号电平使能；
- 脉冲使能模式：接收的数据由外部信号脉冲使能；
- 软件使能模式：数据接收的使能信号可由用户直接配置。

RX 模块还支持对外部使能信号进行翻转。如果外部使能信号为低电平或负脉冲，可通过置位 PARL_IO_RX_EXT_EN_INV 使能该功能，转换为以下相应的接收模式。

35.5.4.1 电平使能模式

电平使能模式如图 35-5 所示。在这种模式下，外部使能信号的有效电平必须与有效数据对齐。由于外部电平使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 7 个。

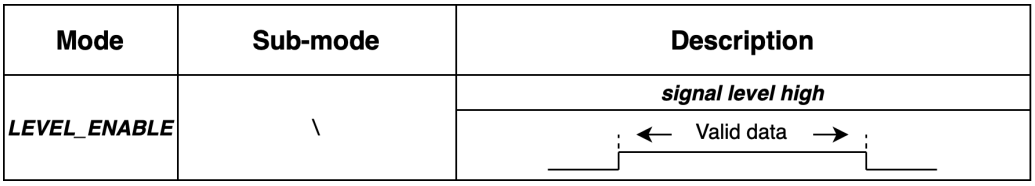


图 35-5. RX 模块电平使能子模式

35.5.4.2 脉冲使能模式

根据脉冲有效电平及其与有效数据的对应性，脉冲使能模式可分为六种子模式，详细分类可参见图 35-6。

子模式 1 ~ 4 都包含起始脉冲和结束脉冲，不同之处在于起始脉冲和结束脉冲是否与有效数据对齐。

子模式 5 ~ 6 只包含起始脉冲，通过配置 PARL_IO_RX_BITLEN 可表示有效数据传输结束。

由于外部脉冲使能信号需要占用一个 IO 管脚，RXD 可用的 IO 管脚最多为 7 个。但在子模式 4 和子模式 6 下，数据在脉冲的第一个边沿后和最后一个边沿前被认为有效，可以实现使能信号的 IO 管脚与数据 IO 管脚的复用，此时 RXD 可使用 8 个 IO 管脚。

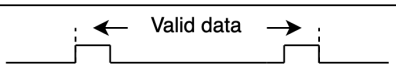
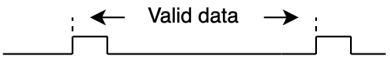
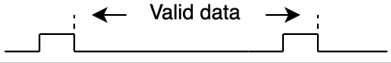
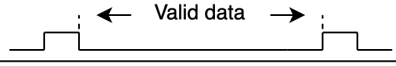
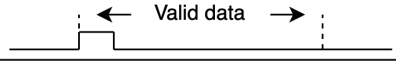
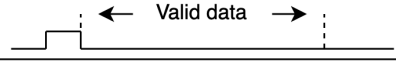
Mode	Sub-mode	Description
PULSE_ENABLE	sub-mode1	***pulse start(data bit include) && pulse end(data bit include)*** 
	sub-mode2	***pulse start(data bit include) && pulse end(data bit exclude)*** 
	sub-mode3	***pulse start(data bit exclude) && pulse end(data bit include)*** 
	sub-mode4	***pulse start(data bit exclude) && pulse end(data bit exclude)*** 
	sub-mode5	***pulse start(data bit include) && length end*** 
	sub-mode6	***pulse start(data bit exclude) && length end*** 

图 35-6. RX 模块脉冲使能子模式

35.5.4.3 软件使能模式

软件使能模式下的使能信号由内部配置寄存器决定。用户切换到此模式时，只有同时写 1 到 PARL_IO_RX_SW_EN 和 PARL_IO_RX_START，才能开始接收数据。

由于使能信号不占用接口 IO 管脚，RXD 在接口上的最大可用 IO 管脚数量为 8。由于时钟域不同，使能信号无法与有效数据对齐，因此只能通过有效时钟沿来判定有效数据。此时，RX Core 时钟需要与有效数据一一对应。

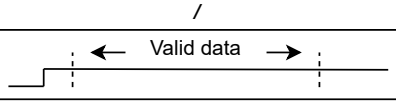
Mode	Sub-mode	Description
SW_ENABLE	/	/ 

图 35-7. RX 模块软件使能子模式

35.5.5 RX 模块 GDMA SUC EOF 信号生成

RX 模块产生一个 GDMA SUC EOF 信号，表示当前帧传输结束，并将其发送到 GDMA 接口。GDMA SUC EOF 信号可由外部使能信号生成，也可基于内部配置的比特长度触发。

- 当 GDMA SUC EOF 信号由内部配置的比特长度生成时，接收模式的选择不受限制。但在这种情况下必须配置 PARL_IO_RX_BITLEN。如果 PARL_IO_RX_BITLEN 的配置值小于实际接收的数据，将提前触发 GDMA SUC EOF，此时 RX 模块停止从 FIFO 读取数据，但 FIFO 会继续接收外部数据，直到 RX_FIFO_WOVF_INT 中断被触发。
- 当 GDMA SUC EOF 信号由外部使能信号产生时，只可选脉冲使能模式的子模式 1、3。在这种模式下，传输不受 PARL_IO_RX_BITLEN 值的影响，并且传输帧的数据长度不受限制。

35.5.6 RX 模块超时

RX 模块支持接收超时功能。当 RX 模块 FIFO 长时间未接收到有效数据，超时将被触发，产生一个 GDMA ERR EOF 信号，并发送至 GDMA 接口以表示结束数据接收。配置 PARL_IO_RX_TIMEOUT_THRES 可设置超时阈值。

默认情况下启用超时功能，用户也可以选择禁用该功能。超时阈值的配置上限为 AHB 时钟域的 $(2^{16} - 1)$ 周期，下限取决于 AHB 时钟域和 RX Core 时钟域之间的相对频率关系。建议将 PARL_IO_RX_TIMEOUT_THRES 设置为较大的数值，以避免生成非期望的 GDMA ERR EOF 信号。

35.5.7 TX 模块输出时钟门控

TX 模块支持对输出时钟进行门控。时钟门控功能默认关闭，可通过软件置位 PARL_IO_TX_GATING_EN 使能该功能。门控信号固定为 TXD 最高位，该位为高电平时，时钟信号可以翻转。目前 TXD 最高位有两个控制源可配置，分别为 DMA 输出数据和有效信号。将 PARL_IO_TX_VALID_OUTPUT_EN 配置为 0 即选择 DMA 输出数据为控制源，此时必须将 TX 模块配置为最大位宽；将 PARL_IO_TX_VALID_OUTPUT_EN 配置为 1 即选择有效信号为控制源，此时对位宽无限制。

若启用时钟门控功能，门控信号将占用一个 IO，此时 TXD 可以使用的 IO 管脚最多为 7 个。

35.5.8 TX 模块有效信号输出

TX 模块可生成与 TXD 对齐的有效信号，可通过配置 PARL_IO_TX_VALID_OUTPUT_EN 选择是否将其输出到 TXD。有效信号的极性固定为高电平有效。

默认情况下禁用有效输出功能。若启用该功能，输出的有效信号将占据 TXD 的最高有效位，即无论原始值为多少，TXD 的第 7 位将维持在高电平，并作为有效信号输出。然而，有效信号管脚并不影响总线宽度的配置。例如，如果用户将数据总线宽度配置为 1 位，即数据管脚为 TXD[0]，此时用户仍然可以启用有效输出功能，将 TXD[7] 固定为有效信号输出管脚。

> **说明：**
> 同时启用有效信号输出功能和门控功能时，TXD 最高位（即门控信号）为连续的高电平信号。若软件对门控信号有非连续高电平的特殊要求，则不可以同时启用两个功能。

35.5.9 TX 模块总线空闲值

TX 模块不传输数据时被认为保持在空闲状态。TX 模块支持总线空闲值可配置。

总线空闲值默认为 0x0，最大可配置值为 0xFF。注意，应避免配置的空闲值与其他已开启的功能产生冲突，例如当使用 TXD 的最高有效位作为有效信号时，应避免将空闲值的最高有效位配置为 1。

35.5.10 单帧数据传输

RX/TX 模块以比特为单位进行传输，即单帧至少传输 1 个比特的数据。

当 RX 模块通过比特长度生成 GDMA EOF 信号时，单帧传输的最大长度是 $(2^{19} - 1)$ 比特。当 RX 模块通过外部设备的使能信号产生 GDMA EOF 信号时，单帧传输的比特数没有限制。TX 模块仅通过比特长度生成 EOF 信号，因此单帧传输的最大长度为 $(2^{19} - 1)$ 比特。

由于 PARLIO 在 GDMA 一侧以字节为单位进行传输，因此当 IO 一侧的数据不与字节对齐时，PARLIO 会对数据进行特殊处理。

- 在接收状态下，通过 GDMA 存入内存的数据将自动在高位补 0，从而与字节对齐。
- 在发送状态下，从内存中取出的数据将根据配置的 PARL_IO_RX_BITLEN 值被截断，超过该配置值的数据不会被发送。

当配置的数据总线宽度为 2/4/8 位时，比特长度必须配置为对应总线宽度的倍数。

35.5.11 字节范围内数据顺序翻转

总线位宽为 1/2/4 位时，可以对一个字节范围内的数据顺序进行翻转。以 RX 模块为例，当配置总线位宽为 2 比特时，数据在被写入 RX FIFO 之前需要被打包成一个字节。

假设原始的比特数据排列顺序为：

{ { b_0, b_1 }, { b_2, b_3 }, { b_4, b_5 }, { b_6, b_7 } }

则启用该功能后，数据顺序将更改为：

{ { b_6, b_7 }, { b_4, b_5 }, { b_2, b_3 }, { b_0, b_1 } }

35.6 配置流程

35.6.1 数据接收操作流程

本节介绍 RX 模块接收数据的配置流程。参照以下流程，从连接到外部设备的 IO 管脚接收并行数据，并存储在内部存储器中。关于 RX 模块时钟和复位的详细操作限制，请参考小节 35.5.2。

1. 复位 RX 模块。具体复位方法与顺序，请参考小节 35.5.2。
2. 置位 PARL_IO_RX_FIFO_WOVF_INT_CLR 和 PARL_IO_RX_FIFO_WOVF_INT_ENA。
3. 选择 RXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置 PCR 寄存器，选择时钟源并对时钟进行分频。
5. 关闭 RX 模块 Core 时钟域的时钟。
6. 按照小节 35.3 和 35.5 中的描述，选择所需接收模式并启用相关功能。
7. 配置 GDMA 接收链表。
8. 置位 PARL_IO_RX_REG_UPDATE，同步寄存器信号。
9. 置位 PARL_IO_RX_START。

10. 打开 RX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其发送数据。
12. 轮询 GDMA SUC EOF 中断信号。
13. 清除 GDMA SUC EOF 中断信号。
14. 关闭 RX 模块 Core 时钟域的时钟。
15. 清除 PARL_IO_RX_START。

35.6.2 数据发送操作流程

本节介绍 TX 模块发送数据的编程流程。参照以下流程，将内部存储器中的并行数据传输到与外部设备相连的 IO 管脚。关于 TX 模块时钟和复位的详细操作限制，请参考小节 35.5.2。

1. 复位 TX 模块。具体复位方法与顺序，请参考小节 35.5.2。
2. 依次置位 PARL_IO_TX_FIFO_REMPTY_INT_CLR、PARL_IO_TX_EOF_INT_CLR、PARL_IO_TX_FIFO_REMPTY_INT_ENA 和 PARL_IO_TX_EOF_INT_ENA。
3. 选择 TXD IO 管脚。如果用户使用 PAD 时钟，则也需配置时钟 IO 管脚。
4. 配置 PCR 寄存器，选择时钟源并对时钟进行分频。
5. 关闭 TX 模块 Core 时钟域的时钟。
6. 按照小节 35.5 中的描述，选择开启所需功能。
7. 配置 GDMA 发送链表。
8. 轮询 PARL_IO_TX_READY。
9. 置位 PARL_IO_TX_START。
10. 打开 TX 模块 Core 时钟域的时钟。
11. 操作外部设备，使其接收数据。
12. 轮询 PARL_IO_TX_EOF_INT_ST 中断信号。
13. 置位 PARL_IO_TX_EOF_INT_CLR 中断信号。
14. 关闭 TX 模块 Core 时钟域的时钟。
15. 清除 PARL_IO_TX_START。

35.7 应用示例

本节介绍 PARLIO 模块的应用示例及其具体操作过程。示例中使用的外围设备都来自 ESP 系列芯片，这些外设均可以和 PARLIO 模块组成完整的数据通路。

> **说明：**
> 示例中所构建的数据通路可能不是最佳方案，例如，在实际应用中，用户可使用两颗相同 ESP 芯片上的 SPI 外设来完成点对点传输，而无需使用 PARLIO。但本小节中的示例在一定程度上证明了 PARLIO 接口的灵活性。

35.7.1 与 SPI 进行数据传输

在本示例中，外部 SPI 作为主机发送数据，PARLIO RX 模块作为从机接收数据，与此同时，PARLIO TX 模块作为主机发送数据，外部 SPI 作为从机接收数据，从而实现点对点串行数据传输。

- 实现 SPI 发送、PARLIO 接收，请参考以下操作过程：
  - 配置 SPI 时钟。
  - 将 SPI 配置为主机。
  - 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_RX，FSPICS0 管脚连接至 RXD[7]，FSPID 管脚连接至 RXD[0]。
  - 将发送的数据写入 SPI 缓冲器中，并配置发送数据的位长。
  - 置位 SPI_UPDATE，更新配置的寄存器值。
  - 复位 PARLIO RX 模块。
  - 配置 PARLIO RX 模块时钟。
  - 关闭 PARLIO RX 模块 Core 时钟域。
  - 将 PARLIO 接收模式配置为电平使能模式的子模式 1，配置 RX 模块数据总线宽度为 1 位，然后根据 SPI 的发送数据长度配置 PARL_IO_RX_BITLEN，最后置位 PARL_IO_RX_REG_UPDATE。
  - 配置 PARLIO GDMA 接收链表。
  - 置位 PARL_IO_RX_START。
  - 打开 PARLIO RX 模块 Core 时钟域。
  - 置位 SPI_USR，开始传输 SPI 的数据。
  - 轮询 GDMA SUC EOF 中断信号。
  - 清除 PARL_IO_RX_START。
- 实现 PARLIO 发送、SPI 接收，请参考以下操作过程：
  - 配置 SPI 时钟。
  - 将 SPI 配置为从机。
  - 配置信号管脚，将 FSPICLK 管脚连接至 PAD_CLK_TX，FSPICS0 管脚连接至 TXD[7]，FSPID 管脚连接至 TXD[0]。
  - 置位 SPI_RD_BIT_ORDER，翻转比特数据顺序。
  - 置位 SPI_UPDATE，更新配置的寄存器值。
  - 复位 PARLIO TX 模块。
  - 置位 PARL_IO_TX_EOF_INT_CLR 和 PARL_IO_TX_EOF_INT_ENA。
  - 配置 PARLIO TX 模块时钟。
  - 关闭 PARLIO TX 模块 Core 时钟域。
  - 配置数据总线宽度为 1 位，写 1 至 PARL_IO_TX_VALID_OUTPUT_EN，然后配置 PARL_IO_TX_BITLEN。

- 配置 PARLIO GDMA 发送链表。
- 轮询 PARL_IO_TX_READY。
- 写 1 至 PARL_IO_TX_START。
- 打开 PARLIO TX 模块 Core 时钟域。
- 开始数据传输。
- 轮询 PARL_IO_TX_EOF_INT_ST。
- 置位 PARL_IO_TX_EOF_INT_CLR。
- 关闭 PARLIO TX 模块 Core 时钟域。
- 清除 PARL_IO_TX_START。

35.7.2 与 I2S 进行数据传输

在本示例中，外部 I2S 作为主机发送数据，PARLIO RX 模块作为从机接收数据。PARLIO 支持 I2S TDM MSB 对齐标准模式和 TDM PCM 标准模式下的传输。当 I2S 传输协议为 TDM MSB 对齐标准模式时，需将 PARLIO 的接收模式配置为电平使能模式。当 I2S 传输协议为 TDM PCM 标准模式时，需将 PARLIO 的接收模式配置为脉冲使能模式的子模式 4，并置位 PARL_IO_RX_EXT_EN_INV。

本节以 TDM PCM 标准模式为例，具体操作过程如下：

1. 配置 I2S 时钟。
2. 配置信号管脚，将 I2SO_BCK_out 连接至 PAD_CLK_RX，I2SO_WS_out 连接至 RXD[7]，I2SO_Data_out 管脚连接至 RXD[0]。
3. 将 I2S 配置为主机。
4. 配置所需 I2S TX 数据和通道模式，并置位 I2S_TX_UPDATE。
5. 复位 I2S TX 模块和 TX FIFO。
6. 使能 I2S_TX_DONE_INT。
7. 配置 I2S GDMA 发送链表。
8. 置位 I2S_TX_STOP_EN。
9. 复位 PARLIO RX 模块。
10. 配置 PARLIO RX 模块时钟。
11. 关闭 PARLIO RX 模块 Core 时钟域。
12. 将 PARLIO 接收模式配置为脉冲使能模式的子模式 10，配置 RX 模块数据总线宽度为 1 位，然后根据 I2S 的发送数据长度配置 PARL_IO_RX_BITLEN，最后置位 PARL_IO_RX_REG_UPDATE。
13. 配置 PARLIO GDMA 接收链表。
14. 置位 PARL_IO_RX_START。
15. 打开 PARLIO RX 模块 Core 时钟域。
16. 置位 I2S_TX_START，开始发送数据。
17. 轮询 I2S_TX_DONE_INT。

18. 轮询 GDMA SUC EOF 中断信号。
19. 清除 I2S_TX_START。
20. 清除 PARL_IO_RX_START。

35.8 中断

- TX_FIFO_REMPTY_INT：TX 模块 FIFO 空时触发该中断，此时 TX 发送的数据可能出现错误。
- RX_FIFO_WOVF_INT：RX 模块 FIFO 满时触发该中断，此时 RX 接收的数据可能出现错误。
- TX_EOF_INT：TX 模块发送完成完整的一帧数据时触发该中断。

35.9 寄存器列表

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

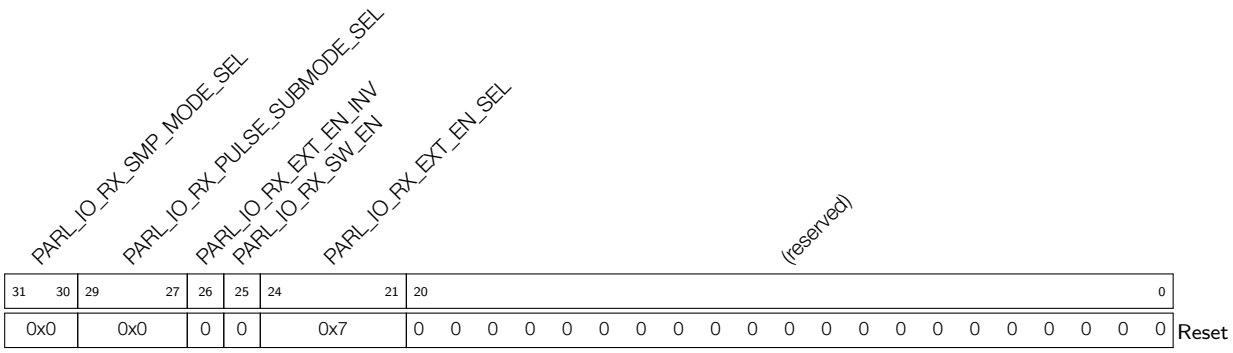
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
PARLIO RX 配置寄存器			
PARL_IO_RX_MODE_CFG_REG	PARLIO RX 采样模式配置寄存器	0x0000	R/W
PARL_IO_RX_DATA_CFG_REG	PARLIO RX 数据配置寄存器	0x0004	R/W
PARL_IO_RX_GENRL_CFG_REG	PARLIO RX 通用配置寄存器	0x0008	R/W
PARL_IO_RX_START_CFG_REG	PARLIO RX 开始配置寄存器	0x000C	R/W
PARLIO TX 配置寄存器			
PARL_IO_TX_DATA_CFG_REG	PARLIO TX 数据配置寄存器	0x0010	R/W
PARL_IO_TX_START_CFG_REG	PARLIO TX 开始配置寄存器	0x0014	R/W
PARL_IO_TX_GENRL_CFG_REG	PARLIO TX 通用配置寄存器	0x0018	R/W
PARLIO 配置与状态寄存器			
PARL_IO_FIFO_CFG_REG	PARLIO FIFO 配置寄存器	0x001C	R/W
PARL_IO_REG_UPDATE_REG	PARLIO 寄存器更新配置寄存器	0x0020	WT
PARL_IO_ST_REG	PARLIO 状态寄存器	0x0024	RO
PARLIO 中断配置与状态寄存器			
PARL_IO_INT_ENA_REG	PARLIO 中断使能寄存器	0x0028	R/W
PARL_IO_INT_RAW_REG	PARLIO 原始中断寄存器	0x002C	R/SS/WTC
PARL_IO_INT_ST_REG	PARLIO 中断状态寄存器	0x0030	RO
PARL_IO_INT_CLR_REG	PARLIO 中断清除寄存器	0x0034	WT
PARLIO RX/TX 状态寄存器			
PARL_IO_RX_ST0_REG	PARLIO RX 状态寄存器 0	0x0038	RO
PARL_IO_RX_ST1_REG	PARLIO RX 状态寄存器 1	0x003C	RO
PARL_IO_TX_ST0_REG	PARLIO TX 状态寄存器 0	0x0040	RO
PARLIO 时钟配置寄存器			
PARL_IO_RX_CLK_CFG_REG	PARLIO RX 时钟配置寄存器	0x0044	R/W
PARL_IO_TX_CLK_CFG_REG	PARLIO TX 时钟配置寄存器	0x0048	R/W
PARL_IO_CLK_REG	PARLIO 时钟配置寄存器	0x0120	R/W
PARLIO 版本寄存器			
PARL_IO_VERSION_REG	版本控制寄存器	0x03FC	R/W

35.10 寄存器

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

Register 35.1. PARL_IO_RX_MODE_CFG_REG (0x0000)



PARL_IO_RX_EXT_EN_SEL 配置 RX 外部使能信号的 IO 管脚。(R/W)

PARL_IO_RX_SW_EN 配置是否使能软件数据采集。

0：不使能

1：使能

(R/W)

PARL_IO_RX_EXT_EN_INV 配置是否翻转外部使能信号。

0：无效

1：翻转

(R/W)

PARL_IO_RX_PULSE_SUBMODE_SEL 配置 RXD 脉冲使能子模式。

0：正脉冲起始（包括数据位）& 正脉冲结束（包括数据位）

1：正脉冲起始（包括数据位）& 正脉冲结束（不包括数据位）

2：正脉冲起始（不包括数据位）& 正脉冲结束（包括数据位）

3：正脉冲起始（不包括数据位）& 正脉冲结束（不包括数据位）

4：正脉冲起始（包括数据位）& 长度结束

5：正脉冲起始（不包括数据位）& 长度结束

(R/W)

PARL_IO_RX_SMP_MODE_SEL 配置 RXD 采集模式。

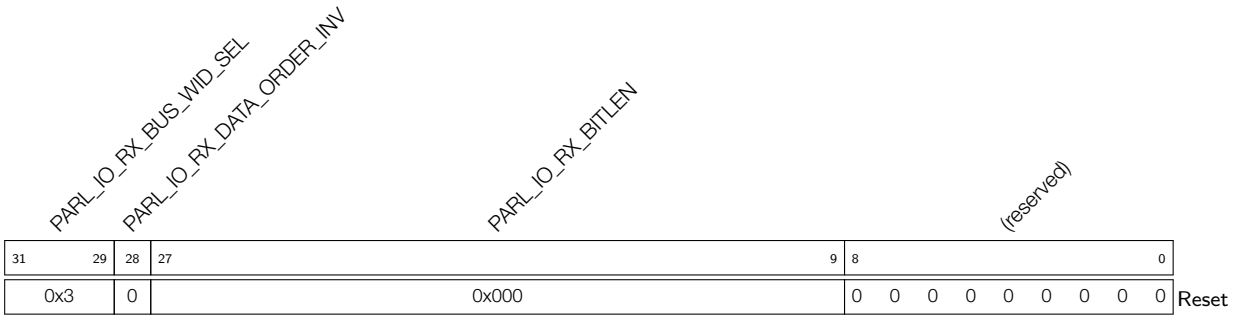
0：外部电平使能模式

1：外部脉冲使能模式

2：内部软件使能模式

(R/W)

Register 35.2. PARL_IO_RX_DATA_CFG_REG (0x0004)



PARL_IO_RX_BITLEN 配置 RXD 的比特长度。(R/W)

PARL_IO_RX_DATA_ORDER_INV 配置是否翻转从 RX_FIFO 发送到 GDMA 的数据的字节范围内比特顺序。

0：无效

1：翻转数据顺序

(R/W)

PARL_IO_RX_BUS_WID_SEL 配置 RXD 总线位宽。

0：1 位

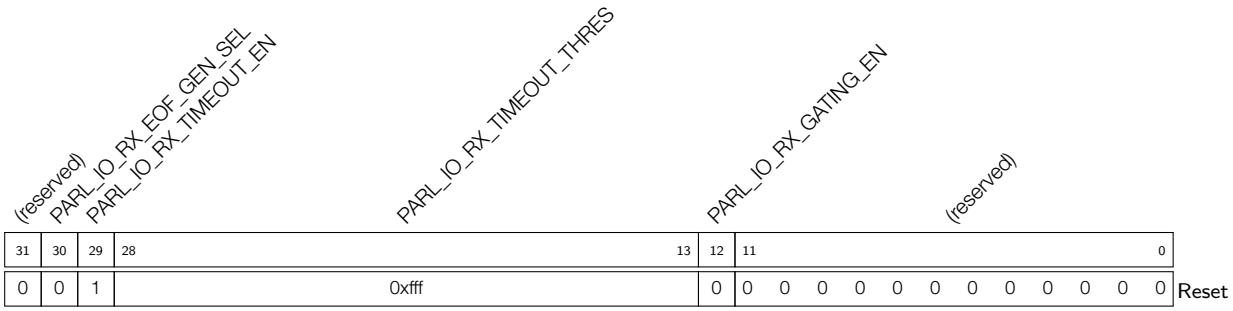
1：2 位

2：4 位

3：8 位

(R/W)

Register 35.3. PARL_IO_RX_GENRL_CFG_REG (0x0008)



PARL_IO_RX_GATING_EN 配置是否使能 RX 输出时钟的时钟门控。

0：不使能

1：使能

(R/W)

PARL_IO_RX_TIMEOUT_THRES 配置 RX 超时计数器的阈值。(R/W)

PARL_IO_RX_TIMEOUT_EN 配置是否使能超时功能来生成 GDMA ERR EOF。

0：不使能

1：使能

(R/W)

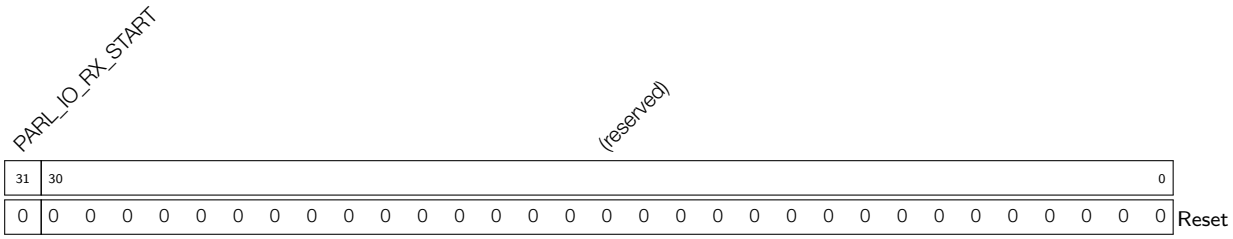
PARL_IO_RX_EOF_GEN_SEL 配置 GDMA SUC EOF 的生成模式。

0：通过配置的数据比特长度生成 GDMA SUC EOF

1：通过外部使能信号生成 GDMA SUC EOF

(R/W)

Register 35.4. PARL_IO_RX_START_CFG_REG (0x000C)



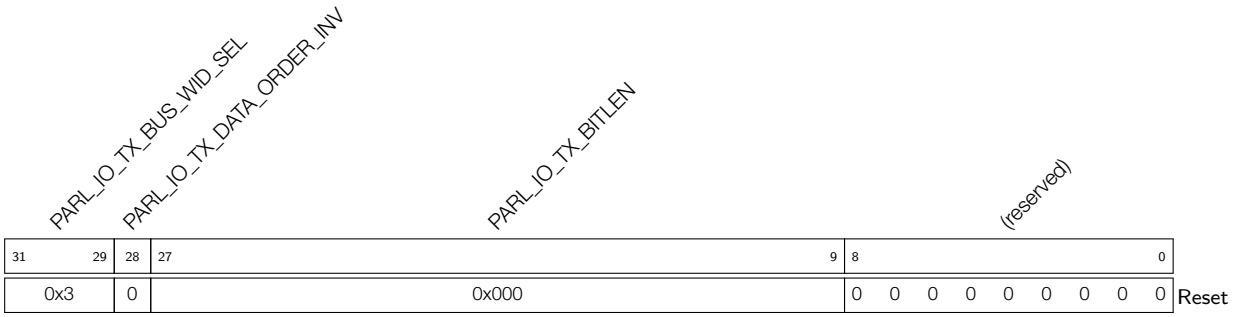
PARL_IO_RX_START 配置是否开始 RX 数据采集。

0：无效

1：开始

(R/W)

Register 35.5. PARL_IO_TX_DATA_CFG_REG (0x0010)



PARL_IO_TX_BITLEN 配置 TXD 的比特长度。(R/W)

PARL_IO_TX_DATA_ORDER_INV 配置是否翻转从 TX_FIFO 发送到 IO 的数据的字节范围内比特顺序。

0：无效

1：翻转数据顺序

(R/W)

PARL_IO_TX_BUS_WID_SEL 配置 TXD 总线位宽。

0：1 位

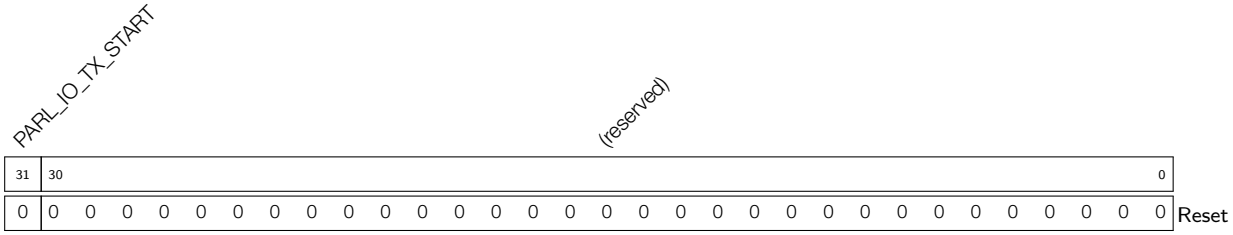
1：2 位

2：4 位

3：8 位

(R/W)

Register 35.6. PARL_IO_TX_START_CFG_REG (0x0014)



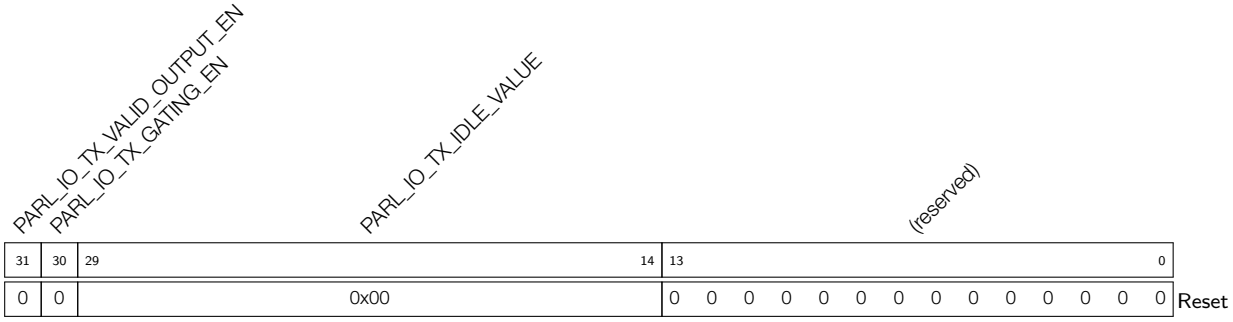
PARL_IO_TX_START 配置是否开始 TX 数据输出。

0：无效

1：开始

(R/W)

Register 35.7. PARL_IO_TX_GENRL_CFG_REG (0x0018)



PARL_IO_TX_IDLE_VALUE 配置 TX 总线上在未发送状态下的数据值。(R/W)

PARL_IO_TX_GATING_EN 配置是否使能 TX 输出时钟的时钟门控。

0：不使能

1：使能

(R/W)

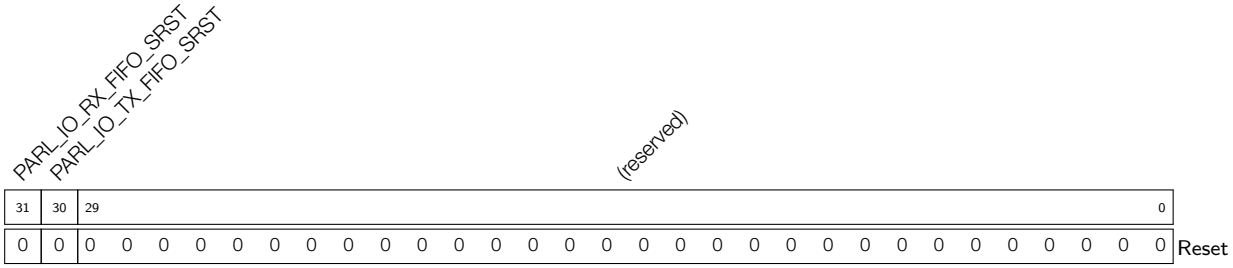
PARL_IO_TX_VALID_OUTPUT_EN 配置是否使能 TX 数据有效信号输出。

0：不使能

1：使能

(R/W)

Register 35.8. PARL_IO_FIFO_CFG_REG (0x001C)



PARL_IO_TX_FIFO_SRST 配置是否复位 TX 异步 FIFO。

0：无效

1：复位

(R/W)

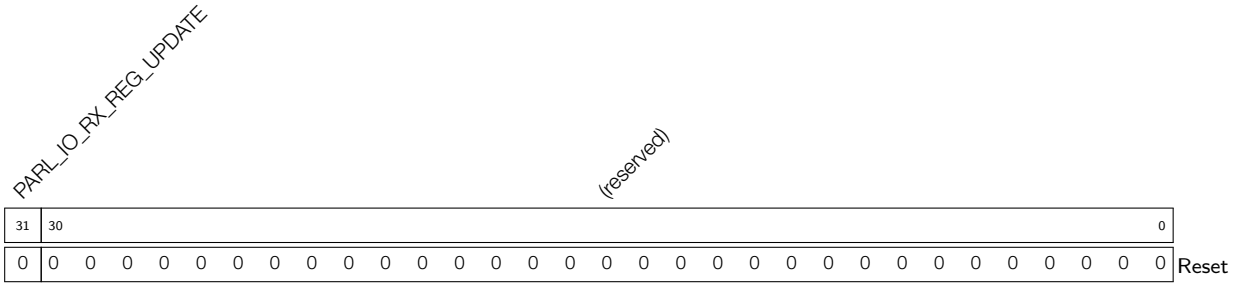
PARL_IO_RX_FIFO_SRST 配置是否复位 RX 异步 FIFO。

0：无效

1：复位

(R/W)

Register 35.9. PARL_IO_REG_UPDATE_REG (0x0020)



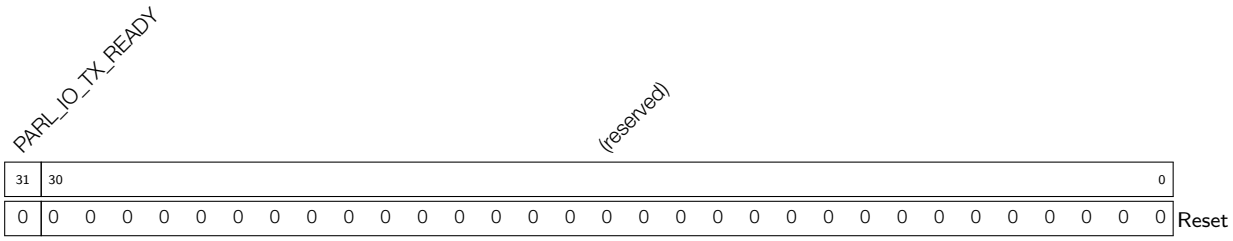
PARL_IO_RX_REG_UPDATE 配置是否更新 RX 寄存器配置。

0：无效

1：更新

(WT)

Register 35.10. PARL_IO_ST_REG (0x0024)



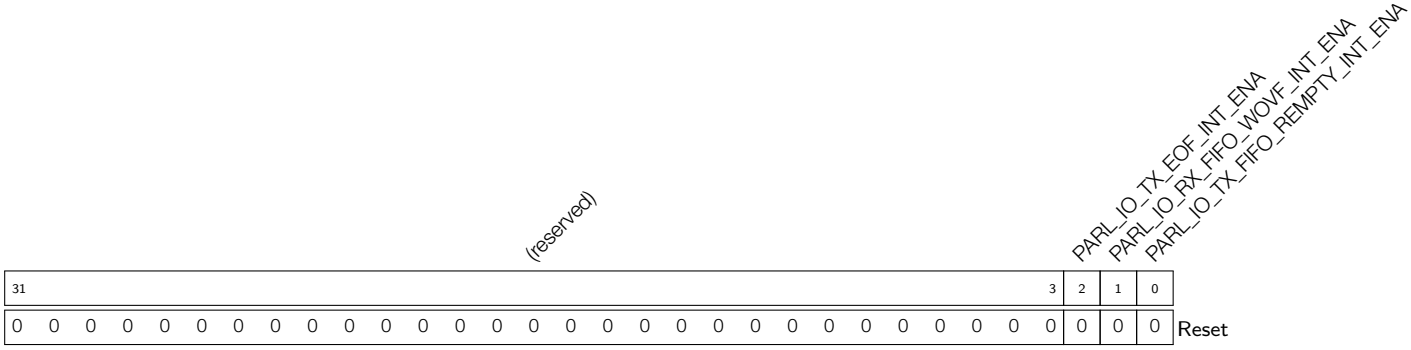
PARL_IO_TX_READY 表示 TX 模块是否已准备好传输数据。

0：未准备好

1：已准备好

(RO)

Register 35.11. PARL_IO_INT_ENA_REG (0x0028)

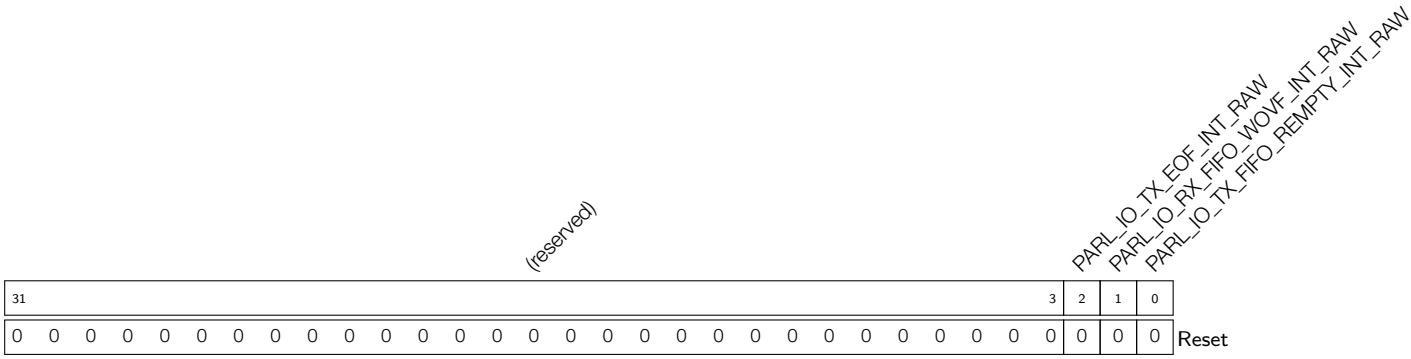


PARL_IO_TX_FIFO_REMPTY_INT_ENA 写 1 使能 TX_FIFO_REMPTY_INT 的屏蔽中断状态。(R/W)

PARL_IO_RX_FIFO_WOVF_INT_ENA 写 1 使能 RX_FIFO_WOVF_INT 的屏蔽中断状态。(R/W)

PARL_IO_TX_EOF_INT_ENA 写 1 使能 TX_EOF_INT 的屏蔽中断状态。(R/W)

Register 35.12. PARL_IO_INT_RAW_REG (0x002C)

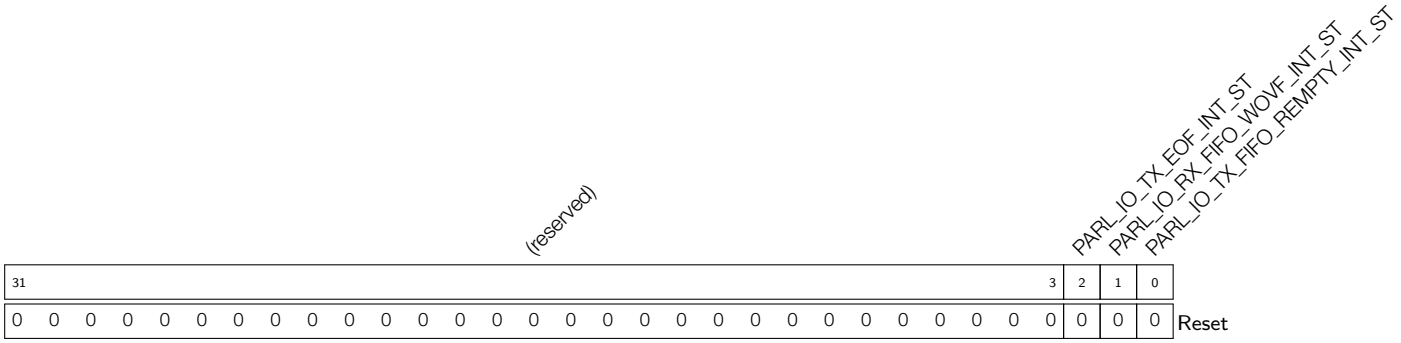


PARL_IO_TX_FIFO_REMPTY_INT_RAW TX_FIFO_REMPTY_INT 的原始中断状态。(R/WTC/SS)

PARL_IO_RX_FIFO_WOVF_INT_RAW RX_FIFO_WOVF_INT 的原始中断状态。(R/WTC/SS)

PARL_IO_TX_EOF_INT_RAW TX_EOF_INT 的原始中断状态。(R/WTC/SS)

Register 35.13. PARL_IO_INT_ST_REG (0x0030)

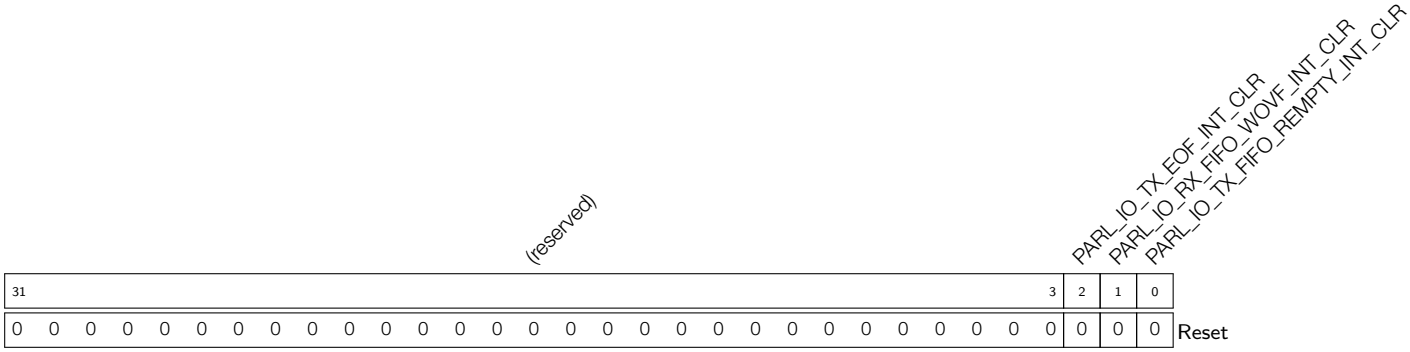


PARL_IO_TX_FIFO_REMPTY_INT_ST TX_FIFO_REMPTY_INT 的屏蔽中断状态。(RO)

PARL_IO_RX_FIFO_WOVF_INT_ST RX_FIFO_WOVF_INT 的屏蔽中断状态。(RO)

PARL_IO_TX_EOF_INT_ST TX_EOF_INT 的屏蔽中断状态。(RO)

Register 35.14. PARL_IO_INT_CLR_REG (0x0034)

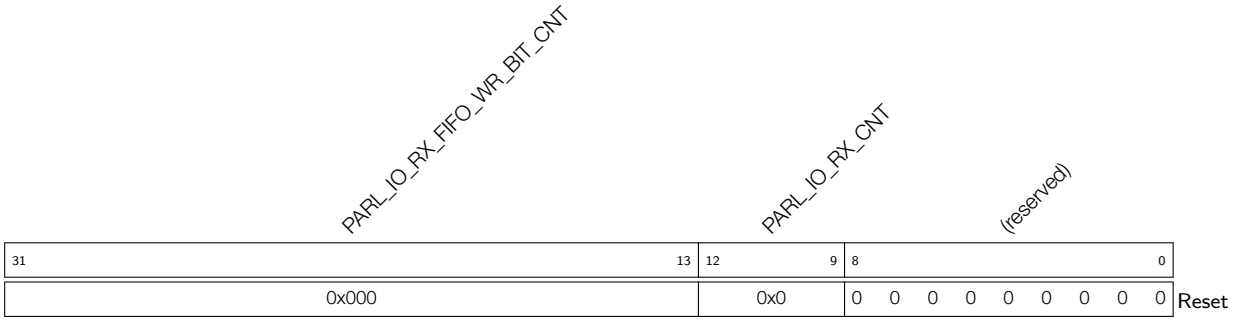


PARL_IO_TX_FIFO_REMPTY_INT_CLR 写 1 清除 TX_FIFO_REMPTY_INT。(WT)

PARL_IO_RX_FIFO_WOVF_INT_CLR 写 1 清除 RX_FIFO_WOVF_INT。(WT)

PARL_IO_TX_EOF_INT_CLR 写 1 清除 TX_EOF_INT。(WT)

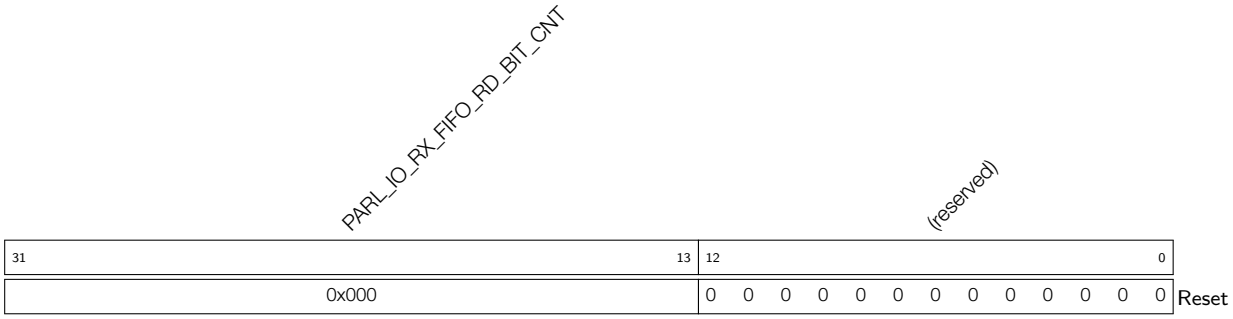
Register 35.15. PARL_IO_RX_ST0_REG (0x0038)



PARL_IO_RX_CNT 表示读取 RX FIFO 的时钟周期数。(RO)

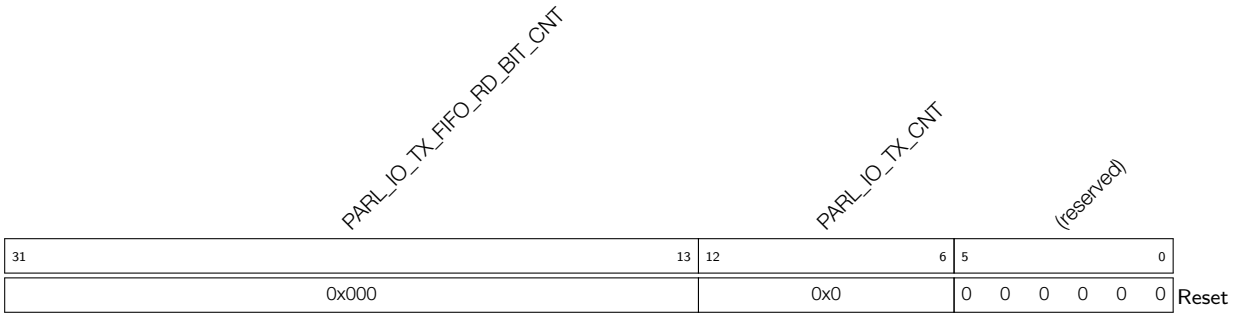
PARL_IO_RX_FIFO_WR_BIT_CNT 表示当前写入 RX FIFO 的位数。(RO)

Register 35.16. PARL_IO_RX_ST1_REG (0x003C)



PARL_IO_RX_FIFO_RD_BIT_CNT 表示当前从 RX FIFO 中读取的位数。(RO)

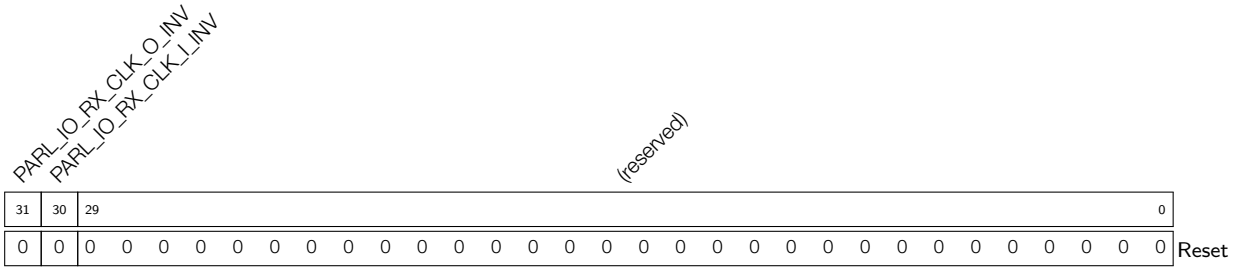
Register 35.17. PARL_IO_TX_ST0_REG (0x0040)



PARL_IO_TX_CNT 表示读取 TX FIFO 的周期数。(RO)

PARL_IO_TX_FIFO_RD_BIT_CNT 表示当前从 TX FIFO 中读取的位数。(RO)

Register 35.18. PARL_IO_RX_CLK_CFG_REG (0x0044)



PARL_IO_RX_CLK_I_INV 配置是否翻转 RX 的输入 Core 时钟。

0：无效

1：翻转

(R/W)

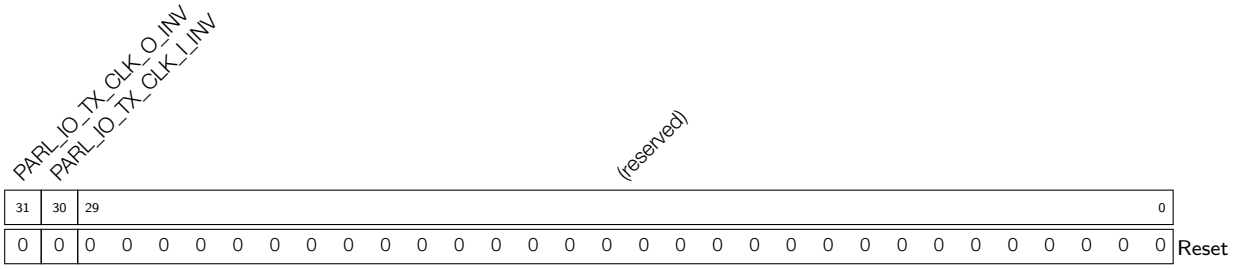
PARL_IO_RX_CLK_O_INV 配置是否翻转 RX 的输出 Core 时钟。

0：无效

1：翻转

(R/W)

Register 35.19. PARL_IO_TX_CLK_CFG_REG (0x0048)



PARL_IO_TX_CLK_I_INV 配置是否翻转 TX 的输入 Core 时钟。

0：无效

1：翻转

(R/W)

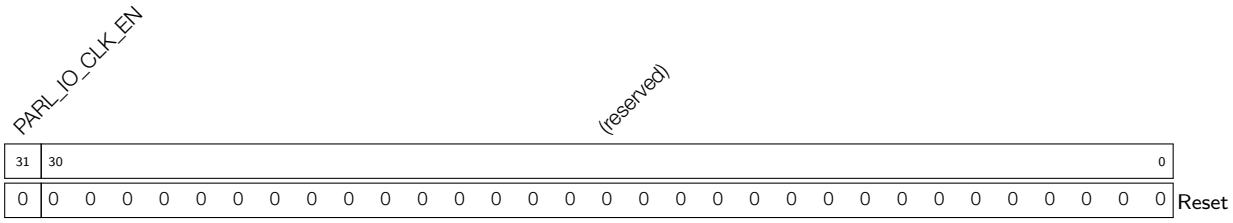
PARL_IO_TX_CLK_O_INV 配置是否翻转 TX 的输出 Core 时钟。

0：无效

1：翻转

(R/W)

Register 35.20. PARL_IO_CLK_REG (0x0120)



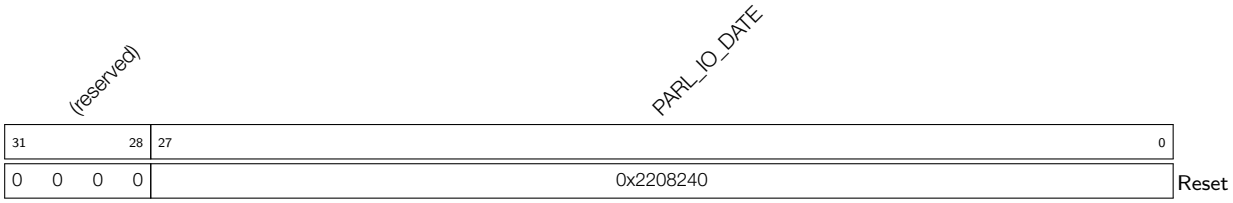
PARL_IO_CLK_EN 配置是否为此寄存器文件强制开启时钟。

0：无效

1：强制开启

(R/W)

Register 35.21. PARL_IO_VERSION_REG (0x03FC)



PARL_IO_DATE 版本控制寄存器。(R/W)

36 SAR ADC 转换器与温度传感器

36.1 概述

ESP32-H2 搭载了以下模拟外设：

- 一个 12 位逐次逼近型模拟数字转换器 (SAR ADC)，用于测量最多来自 5 个管脚上的模拟信号。
- 一个温度传感器，用于测量及监测芯片内部温度。

36.2 SAR ADC

36.2.1 介绍

ESP32-H2 内置的 12 位 SAR ADC 是一种逐次逼近型模拟数字转换器。它有 5 个通道，可测量来自 5 个管脚的模拟信号。SAR ADC 由 DIG ADC 控制器控制，该控制器驱动电压采样，支持单次采样和多通道采样。

36.2.2 特性

SAR ADC 具有以下特性：

- 12 位分辨率
- 支持采集最多 5 个管脚上的模拟信号
- 支持单次采样模式和多通道采样模式
- 在多通道采样模式下，支持：
  - 自定义采样通道顺序
  - 两个滤波器，滤波系数可配
  - 阈值监控，滤波后数据大于设置的高阈值或小于设置的低阈值将产生中断
  - GDMA 连续数据搬运
- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

36.2.3 结构概览

SAR ADC 的主要子模块与连接情况见图 36-1。

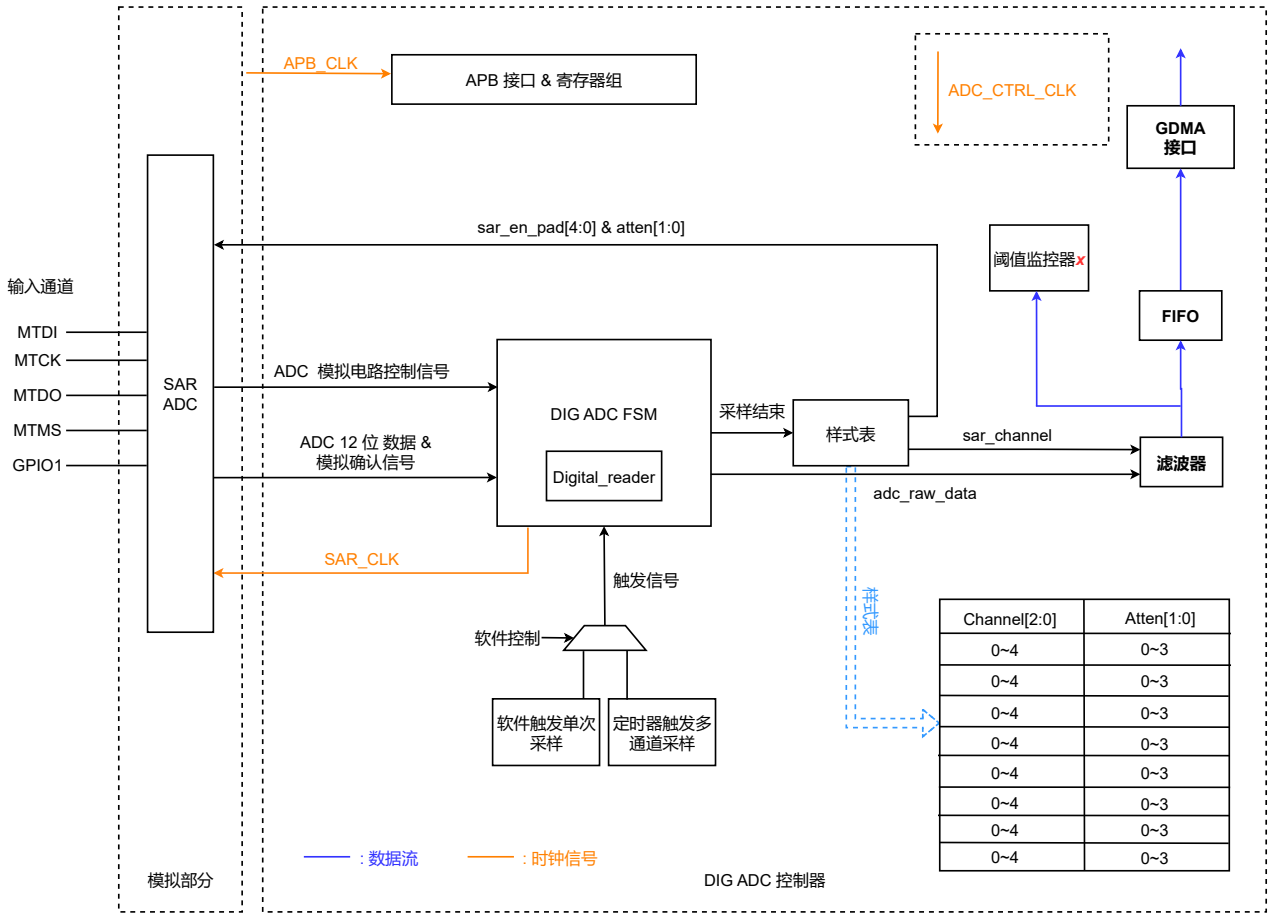


图 36-1. SAR ADC 结构图

根据图 36-1，按照从左至右、从上到下的顺序，SAR ADC 模块主要包括以下功能模块：

- 5 个通道，连接 5 个芯片管脚
- SAR ADC：SAR ADC 的模拟部分
- DIG ADC 控制器：SAR ADC 的数字部分，主要包括：
  - 时钟管理模块：对时钟源进行选择和分频
  - DIG ADC FSM（有限状态机）：生成整个 ADC 采样过程中所需的各种信号
  - Digital_reader：读取 SAR ADC 的数值，由 DIG ADC FSM 驱动
  - 滤波器：用于在多通道采样模式下对目标通道的转换数据进行滤波
  - Threshold Monitor*x*：阈值监控器 1 和阈值监控器 2。可在采样值大于设定的高阈值，或小于设定的低阈值时触发中断。

36.2.4 功能描述

36.2.4.1 ADC 上电

置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 寄存器给 ADC 上电。上电后即可进行采样，硬件设计上已考虑等待时间，因此用户无需特意等待。

36.2.4.2 ADC 通道

SAR ADC 具有 5 个通道，连接芯片的 5 个管脚。ADC 需首先通过内部多路器选择待测量的模拟管脚，然后才能采样模拟信号。

表 36-1 列出了用作 ADC 通道的芯片管脚和相应的 GPIO 编号。

表 36-1. SAR ADC 通道

芯片管脚名称	GPIO 编号	ADC 通道编号
GPIO1	GPIO1	0
MTMS	GPIO2	1
MTDO	GPIO3	2
MTCK	GPIO4	3
MTDI	GPIO5	4

36.2.4.3 ADC 时钟

图 36-2 为 SAR ADC 的时钟结构图。

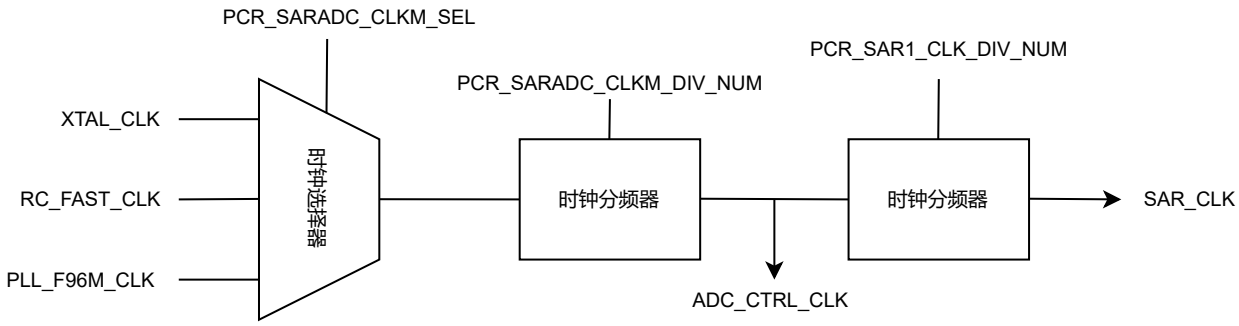


图 36-2. SAR ADC 时钟结构图

SAR ADC 的系统时钟为 ADC_CTRL_CLK，它是 DIG ADC FSM 和其他逻辑电路的工作时钟（APB 接口和 Digital_reader 除外）。

ADC_CTRL_CLK 可选择三个时钟源：XTAL_CLK，RC_FAST_CLK，或 PLL_F96M_CLK。通过寄存器 PCR_SARADC_CLKM_SEL 选择时钟源。

SAR ADC 的分频时钟为 SAR_CLK，由 ADC_CTRL_CLK 分频而得。它是 SAR ADC 和 Digital_reader 的工作时钟，频率不得超过 5 MHz。

更多有关时钟的信息请参考章节 7 复位和时钟。

36.2.4.4 单次采样模式

单次采样模式下，ADC 只在一个通道进行一次采样。该模式由软件通过寄存器 APB_SARADC_ONETIME_SAMPLE 触发。一旦采样完成，转换数据将被存储在 APB_SARADC_DATA 中。如需转换采样通道，可继续配置 APB_SARADC_ONETIME_SAMPLE 寄存器。

36.2.4.5 多通道采样模式

多通道采样模式由专用定时器触发。在该模式下，ADC 会按照指定的顺序采样一组通道。通道顺序在样式表中进行定义。多通道采样模式也可以配置成连续采样一个通道。

用户通过配置寄存器 APB_SARADC_TIMER_EN 使能定时器，配置 APB_SARADC_TIMER_TARGET 设置定时器的触发周期。当定时器计数到配置周期数的 2 倍时，触发采样。定时器的工作时钟是 ADC_CTRL_CLK。

采样完成后，定时器复位为 0，重新开始计数。数据通过 GDMA 连续搬运至内存空间。

> **说明：**
> 单次采样和多通道采样不能同时使用。

36.2.4.6 ADC 转换和衰减

SAR ADC 转换模拟信号时，转换电压范围为 0 mV ~ V_{ref}。V_{ref} 为 ADC 内部参考电压，出厂设定为 1100 mV。ADC 的转换结果是个 12 位的数字值，是原始值。用户可使用以下公式将原始值 $data$ 转换成模拟电压 V_{data}：

$$V_{data} = \frac{V_{ref}}{k} \times \frac{data}{4095}$$

其中 k 为衰减对应的系数。

如需转换大于 V_{ref} 的电压，信号输入 SAR ADC 前可进行衰减。使用寄存器 APB_SARADC_ONETIME_ATTEN 可配置衰减为 0 dB、2.5 dB、6 dB 或 12 dB。

36.2.4.7 DIG ADC FSM

DIG ADC FSM（下文简称 FSM）在多通道采样模式下工作，用于生成整个 ADC 采样过程中所需的各种信号。图 36-3 展示了 FSM 的工作原理。

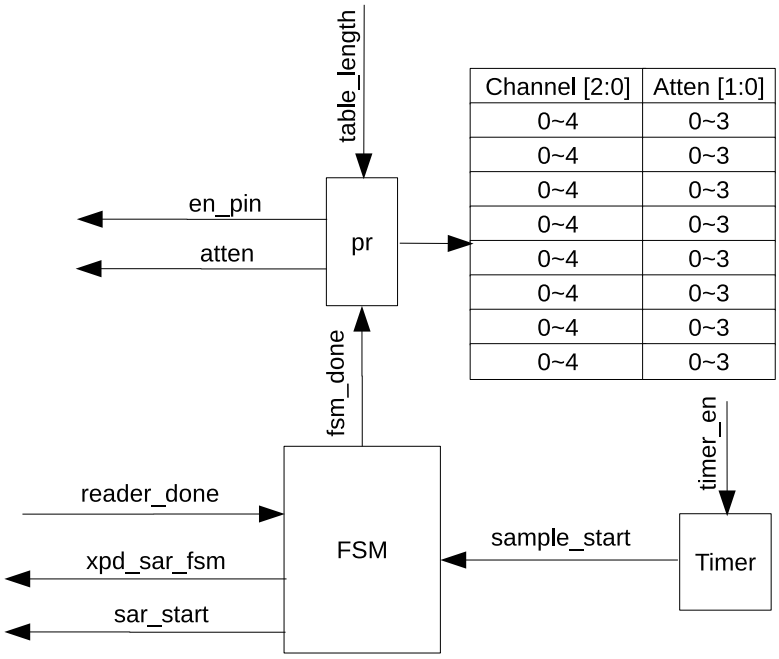


图 36-3. DIG ADC FSM 概况

其中，

- Timer（定时器）：表示 DIG ADC 控制器的专用定时器，用于触发多通道采样模式。定时器可产生 sample_start 信号，表示开始采样。
- pr：样式表指针，FSM 将根据该指针指向的样式配置，发送相应信号。

用户通过置位 APB_SARADC_TIMER_EN 使能定时器。定时器超时将触发 sample_start 信号驱动 FSM 模块开始采样。FSM 模块收到 sample_start 信号后，执行以下操作：

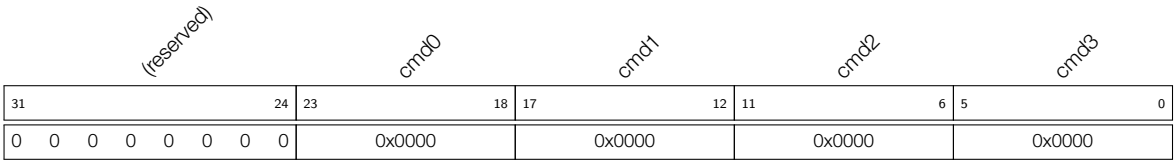
- 开启 SAR ADC 电源；
- 根据当前 pr 指向的样式，决定该次采样的 ADC 通道以及衰减；
- 根据配置信息，输出相应的 en_pin（使能管脚）以及 atten（衰减）信号到模拟端；
- 发起 sar_start 信号，开启采样。

FSM 收到 Digital_reader 返回的 reader_done 信号后，执行以下操作：

- 结束采样；
- 将数据传输给滤波器，然后阈值监控器通过 GDMA 将数据传输给内存（见图 36-1）；
- 更新样式表指针 pr，等待下一次采样。pr 在 0 - APB_SARADC_SAR_PATT_LEN (table_length) 之间循环计数。

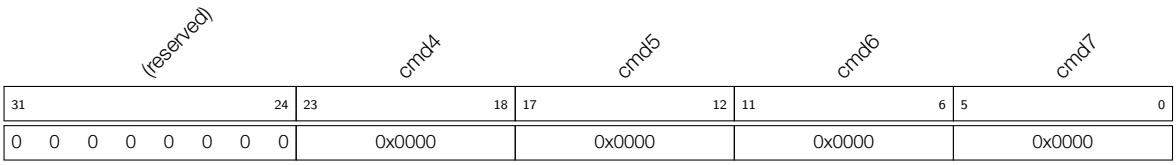
36.2.4.8 样式表

FSM 包含一个样式表，由 APB_SARADC_SAR_PATT_TAB1_REG 和 APB_SARADC_SAR_PATT_TAB2_REG 两个寄存器组成，每个寄存器包含四个样式，每个样式长度为六位，如图 36-4 和图 36-5 所示。



cmd$x$ 表示样式表中的样式，即样式 0 ~ 样式 3。

图 36-4. APB_SARADC_SAR_PATT_TAB1_REG 包含样式 0 - 3



cmd$x$ 表示样式表中的样式，即样式 4 ~ 样式 7。

图 36-5. APB_SARADC_SAR_PATT_TAB2_REG 包含样式 4 - 7

每个样式长度为六位，共包含三个字段，每个字段存储的是采样规则。图 36-6 所示为样式结构，字段描述参见后文。

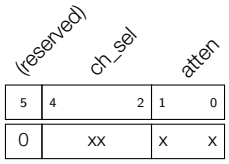


图 36-6. 样式结构

atten 配置衰减：

0：0 dB
1：2.5 dB
2：6 dB
3：12 dB

ch_sel 配置通道：

0：通道 0
1：通道 1
2：通道 2
3：通道 3
4：通道 4

(reserved) 保留字段

样式配置示例

例如，希望实现如下所示的多通道采样方式：

- 采样 SAR ADC 的通道 0，且衰减配置为 2.5 dB；
- 采样 SAR ADC 的通道 2，且衰减配置为 12 dB。

则具体的配置如下：

- 配置第一个样式 cmd0，如下图所示：

图 36-7. cmd0 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 0，即选择通道 0。

- 配置第二个样式 cmd1，如下图所示：

图 36-8. cmd1 配置示例

atten 配置该字段的值为 3，即衰减配置为 12 dB。

ch_sel 配置该字段的值为 2，即选择通道 2。

- 配置 APB_SARADC_SAR_PATT_LEN 为 1，即选择使用上述配置好的样式 0 和样式 1；
- 使能定时器，则 DIG ADC 控制器将根据上述样式配置周期性采样两个通道。

36.2.4.9 ADC 滤波器

DIG ADC 控制器提供两个滤波器，用于在多通道采样模式下对目标通道的转换数据进行滤波。两个滤波器均可配置给 SAR ADC 的任一通道，但不能配置给同一通道。如果配置给同一通道，则第一次配置的滤波器生效。

滤波公式如下所示：

$$data_{cur} = \frac{(k-1)data_{prev}}{k} + \frac{data_{in}}{k} - 0.5$$

- $data_{cur}$：滤波后数据
- $data_{in}$：ADC 转换数据
- $data_{prev}$：上次滤波数据
- k：滤波系数

配置滤波器如下：

- 配置 APB_SARADC_FILTER_CHANNEL*x* 设置滤波器 *x* (*x*=0, 1) 作用的 ADC 通道；
- 配置 APB_SARADC_FILTER_FACTOR*x* 设置滤波器 *x* 的滤波系数 k。

36.2.4.10 阈值监控器

DIG ADC 控制器包含两个阈值监控器，用于在多通道采样模式下对滤波之后的数据进行监控。当数据大于设定的高阈值，则触发高阈值中断；当数据小于设定的低阈值，则触发低阈值中断。两个监控器可配置给 SAR ADC 的任一通道，但不能配置给同一通道。

阈值监控配置如下：

- 置位 APB_SARADC_THRES*x*_EN 使能阈值监控器 *x* (*x*=0, 1)；
- 配置 APB_SARADC_THRES*x*_LOW 设置低阈值；
- 配置 APB_SARADC_THRES*x*_HIGH 设置高阈值；
- 配置 APB_SARADC_THRES*x*_CHANNEL 设置监控的通道；
- 配置 APB_SARADC_THRES_ALL_EN 使能阈值监控。

36.2.4.11 GDMA 支持

由于 ADC 仅有一个数据寄存器 APB_SARADC_DATA 用于存储单次采样的转换数据，因此当转换多个通道时需要使用 GDMA，通过 GDMA 接口将转换数据连续存储至内存。

由 DIG ADC 控制器的专用定时器产生 GDMA 功能触发信号。用户可通过软件配置 APB_SARADC_APB_ADC_TRANS 将 GDMA 的数据通路切换到 DIG ADC 控制器。关于 GDMA 的具体配置，请参考章节 3 *通用 DMA 控制器 (GDMA)*。

GDMA 数据格式

ADC 最终向 GDMA 传递 32 位数据，如下图所示：

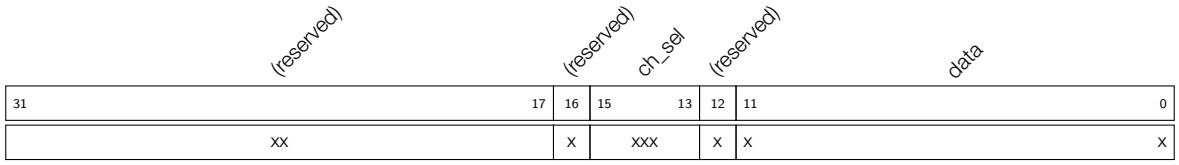


图 36-9. GDMA 数据格式

data ADC 转换结果，12 位

ch_sel 通道信息，3 位

36.2.5 配置流程

36.2.5.1 单次采样配置

软件驱动的单次采样的具体配置如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 给 ADC 上电；
2. 配置 PCR_SARADC_CLKM_SEL 选择时钟源；
3. 配置时钟分频寄存器 PCR_SARADC_CLKM_DIV_NUM 和 PCR_SAR1_CLK_DIV_NUM；
4. 置位 PCR_SARADC_CLKM_EN 使能 SAR ADC 时钟；
5. 置位 APB_SARADC_ONETIME_SAMPLE 使能单次采样；

6. 配置 APB_SARADC_ONETIME_CHANNEL 选择采样通道；
7. 根据需要，配置 APB_SARADC_ONETIME_ATTEN 选择衰减；
8. 置位 APB_SARADC_ONETIME_START 启动单次采样。

采样结束即触发 APB_SARADC_ADC_DONE_INT_RAW 中断。软件检测到该中断后，可启用从 APB_SARADC_ADC_DATA 中读取样本值。如需切换采样通道无需重复 1 至 5 步骤。

36.2.5.2 多通道采样配置

专用定时器驱动的多通道采样的具体配置如下：

1. 置位 PMU_XPD_PERIF_I2C 和 PMU_PERIF_I2C_RST 给 ADC 上电；
2. 配置 PCR_SARADC_CLKM_SEL 选择时钟源；
3. 配置时钟分频寄存器 PCR_SARADC_CLKM_DIV_NUM 和 PCR_SAR1_CLK_DIV_NUM；
4. 置位 PCR_SARADC_CLKM_EN 使能 SAR ADC 时钟；
5. 按照章节 36.2.4.8 *样式表* 所述步骤配置样式表；
6. 按照章节 36.2.4.9 *ADC 滤波器* 所述步骤配置滤波器作用的通道和滤波系数；
7. 根据需要，按照章节 36.2.4.10 *阈值监控器* 所述步骤设置阈值监控；
8. 置位 APB_SARADC_APB_ADC_TRANS 选择使用 GDMA；
9. 配置 APB_SARADC_TIMER_TARGET 设置 DIG ADC 定时器的触发周期；
10. 置位 APB_SARADC_TIMER_EN 使能定时器。

定时器超时则将驱动 DIG ADC FSM 根据样式表进行采样。数据通过 GDMA 自动存储到内存空间中。当采集到 APB_SARADC_APB_ADC_EOF_NUM 寄存器设置的次数时，采样停止。

采样完成将产生 APB_SARADC_ADC_DONE_INT_RAW 中断。

36.2.6 中断

- APB_SARADC_ADC_DONE_INT：SAR ADC 完成一次转换，即触发此中断。
- APB_SARADC_THRES*x*_HIGH_INT：过滤值超过阈值监控器 *x* 的高阈值，即触发此中断。
- APB_SARADC_THRES*x*_LOW_INT：过滤值低于阈值监控器 *x* 的低阈值，即触发此中断。

36.3 温度传感器

36.3.1 介绍

ESP32-H2 搭载了一个温度传感器，用于实时测量芯片内部温度。温度传感器可将输出的电压转换成数字值，并且带有补偿温度偏移的功能。

36.3.2 特性

温度传感器的主要特性包括：

- 支持软件触发测量温度，且一旦触发后，传感器可持续测量温度，软件可实时读取数据

- 支持硬件触发自动监测温度
- 支持两种自动监测唤醒模式
- 支持根据使用环境配置温度偏移，提高测试精度
- 温度测量范围可配置
- 支持多个事件任务矩阵 (ETM) 相关的事件和任务

36.3.3 结构概览

温度传感器的内部结构如图 36-10 所示。

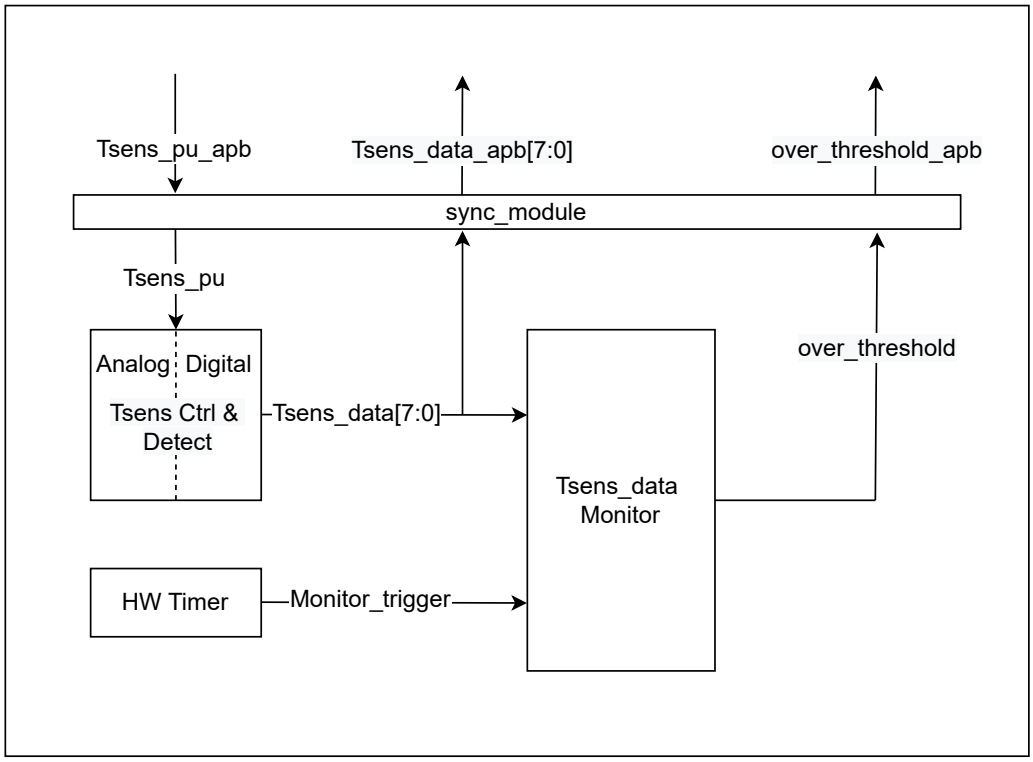


图 36-10. 温度传感器结构框图

根据图 36-10，温度传感器模块主要包括以下功能模块：

- Tsens Ctrl & Detect：温度传感器
- HW Timer（硬件定时器）：用于触发温度监测
- Tsens_data 监测器：用于监测温度是否超出阈值
- sync_module：APB 时钟域与温度传感器时钟域间的同步模块

36.3.4 功能描述

36.3.4.1 温度传感器上电

通过设置 APB_SARADC_TSENS_PU 给温度传感器上电。

36.3.4.2 时钟管理

温度传感器有两个时钟源：RC_FAST_CLK 和 XTAL_CLK。通过寄存器 PCR_TSENS_CLK_SEL 选择时钟源。时钟分频由寄存器 APB_SARADC_TSENS_CLK_DIV 配置。

36.3.4.3 自动监测唤醒模式

温度传感器自动监测唤醒模式有两种：绝对值模式和变化量模式，通过寄存器 APB_SARADC_WAKEUP_MODE 进行选择。

- 绝对值模式：
  - 用于监测当前温度的绝对值。配置 APB_SARADC_WAKEUP_TH_LOW 和 APB_SARADC_WAKEUP_TH_HIGH 来决定温度阈值，温度超过高阈值或小于低阈值时将触发唤醒。
- 变化量模式：
  - 用于监测温度的变化量。若连续两次采样的温度值增量超过 APB_SARADC_WAKEUP_TH_HIGH 的配置值或减少量超过 APB_SARADC_WAKEUP_TH_LOW 的配置值，将触发唤醒。例如当 APB_SARADC_WAKEUP_TH_LOW 的值为 8 时，若连续两次温度采样分别为 28 和 19，即温度减少量为 9，将触发唤醒。

36.3.4.4 温度测量范围与温度偏移

为了提高测试精度，温度传感器的测量范围被设计为 5 个范围，如表 36-2 所示。每个测量范围都有一定的测温误差。用户无需关注测温误差，只需选择相应的温度偏移，即可获得校准后的数据。

表 36-2. 温度测量范围与温度偏移

测量范围 (°C)	温度偏移
50 ~ 125	–2
20 ~ 100	–1
–10 ~ 80	0
–30 ~ 50	1
–40 ~ 20	2

36.3.4.5 数据转换

温度传感器的输出值存储在寄存器 APB_SARADC_TSENS_OUT 中。用户可使用以下公式将输出值 $VALUE$ 转换成实际的温度值 T (°C)。转换公式如下：

$$T = 0.4386 \times VALUE - 27.88 \times offset - 20.52$$

其中，$offset$ 即表 36-2 所示温度偏移。

36.3.5 配置流程

温度传感器可由软件启动，具体配置如下：

1. 置位 APB_SARADC_TSENS_PU 使温度传感器上电；

2. 置位 PCR_TSENS_CLK_EN 使能温度传感器时钟；
3. 配置 APB_SARADC_TSENS_CLK_SEL 选择传感器时钟；
4. 等待 APB_SARADC_TSENS_XPD_WAIT 个时钟周期后，温度传感器的复位释放，开始测量环境温度；
5. 等待 100 μs 后（输出值会随着测量时间的增加而逐渐逼近真实的温度值），直接从 APB_SARADC_TSENS_OUT 中读取温度值。

如要使用硬件自动温度监测功能，则可在给温度传感器上电之前增加如下步骤：

1. 配置 APB_SARADC_TSENS_SAMPLE_RATE 设置采样率；
2. 配置 APB_SARADC_WAKEUP_MODE 选择温度监测唤醒模式；
3. 配置 APB_SARADC_WAKEUP_TH_HIGH/LOW 设置温度高低阈值；
4. 置位 APB_SARADC_WAKEUP_EN 启用温度监测；
5. 置位 APB_SARADC_TSENS_SAMPLE_EN 自动触发持续温度监测。

自动温度监测工作模式下，温度传感器的输出值不会存储，但用户可以随时读取寄存器 APB_SARADC_TSENS_OUT 获取当前输出值。

36.3.6 中断

- APB_SARADC_TSENS_INT：温度采样值超过阈值，即触发此中断。

36.4 事件任务矩阵功能

在 ESP32-H2 中，SAR ADC 及温度传感器支持 ETM 功能，即可以通过任意外设的 ETM 事件触发 SAR ADC/温度传感器的 ETM 任务，或者通过 SAR ADC/温度传感器的 ETM 事件触发任意外设的 ETM 任务。关于 ETM 更多详细信息，请参考章节 10 事件任务矩阵 (SOC_ETM)。这里仅介绍与 SAR ADC/温度传感器相关的 ETM 任务和 ETM 事件。

36.4.1 SAR ADC 的 ETM 功能

SAR ADC 可接收的 ETM 任务有：

- ADC_TASK_SAMPLE：该任务被触发时，ADC 开始进行单次采样。
- ADC_TASK_START：该任务被触发时，ADC 开始进行多通道采样。
- ADC_TASK_STOP：该任务被触发时，ADC 停止采样。

SAR ADC 可产生的 ETM 事件有：

- ADC_EVT_CONV_CMPLT：在单次采样模式和多通道采样模式下，ADC 每完成一次采样时生成。
- ADC_EVT_EQ_ABOVE_THRESH*x*：ADC 过滤后数据高于阈值时生成。*x*=0, 1，代表阈值监控器 0, 1。
- ADC_EVT_EQ_BELOW_THRESH*x*: ADC 过滤后数据低于阈值时生成。*x*=0, 1，代表阈值监控器 0, 1。
- ADC_EVT_STARTED：ADC 开始采样时生成，单次采样不会触发该事件。
- ADC_EVT_STOPPED：ADC 停止采样时生成，单次采样不会触发该事件。

在具体应用中，SAR ADC 的 ETM 事件可以用来触发 SAR ADC 的 ETM 任务，例如，ADC_EVT_EQ_ABOVE_THRESH*x* 事件可以触发 ADC_TASK_STOP 任务。

36.4.2 温度传感器的 ETM 功能

温度传感器可接收的 ETM 任务有：

- TMPSNSR_TASK_START：该任务被触发时，温度传感器开始采样。
- TMPSNSR_TASK_STOP：该任务被触发时，温度传感器停止采样。

温度传感器可产生的 ETM 事件有：

- TMPSNSR_EVT_OVER_LIMIT：温度传感器发现温度超过阈值时生成。

在具体应用中，温度传感器的 ETM 事件可以用来触发温度传感器的 ETM 任务，例如，TMPSNSR_EVT_OVER_LIMIT 事件可以触发 TMPSNSR_TASK_STOP 任务。

36.5 寄存器列表

本小节的所有地址均为相对于片上传感器与模拟信号处理基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

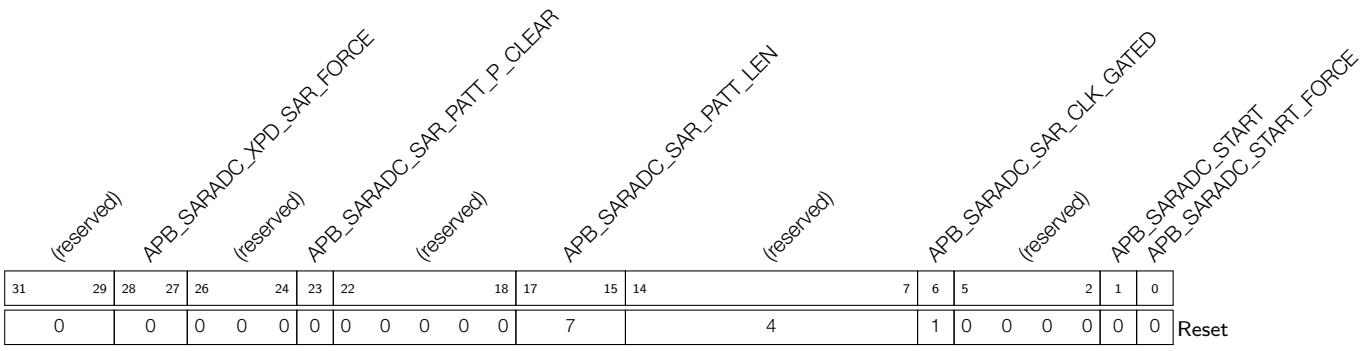
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
APB_SARADC_CTRL_REG	SAR ADC 控制寄存器 1	0x0000	R/W
APB_SARADC_CTRL2_REG	SAR ADC 控制寄存器 2	0x0004	R/W
APB_SARADC_FILTER_CTRL1_REG	滤波控制寄存器 1	0x0008	R/W
APB_SARADC_SAR_PATT_TAB1_REG	样式表寄存器 1	0x0018	R/W
APB_SARADC_SAR_PATT_TAB2_REG	样式表寄存器 2	0x001C	R/W
APB_SARADC_ONETIME_SAMPLE_REG	单次采样配置寄存器	0x0020	R/W
APB_SARADC_FILTER_CTRL0_REG	滤波控制寄存器 0	0x0028	R/W
APB_SARADC_SAR1DATA_STATUS_REG	SAR ADC 转换数据存储寄存器	0x002C	RO
APB_SARADC_THRES0_CTRL_REG	采样阈值控制寄存器 0	0x0034	R/W
APB_SARADC_THRES1_CTRL_REG	采样阈值控制寄存器 1	0x0038	R/W
APB_SARADC_THRES_CTRL_REG	采样阈值控制寄存器	0x003C	R/W
APB_SARADC_INT_ENA_REG	SAR ADC 中断使能寄存器	0x0040	R/W
APB_SARADC_INT_RAW_REG	SAR ADC 原始中断寄存器	0x0044	RO
APB_SARADC_INT_ST_REG	SAR ADC 中断状态寄存器	0x0048	RO
APB_SARADC_INT_CLR_REG	SAR ADC 中断清除寄存器	0x004C	WO
APB_SARADC_DMA_CONF_REG	SAR ADC DMA 配置寄存器	0x0050	R/W
APB_SARADC_APB_TSENS_CTRL_REG	温度传感器控制寄存器 1	0x0058	varies
APB_SARADC_APB_TSENS_CTRL2_REG	温度传感器控制寄存器 2	0x005C	R/W
APB_TSENS_WAKE_REG	温度传感器唤醒功能配置寄存器	0x0064	varies
APB_TSENS_SAMPLE_REG	温度传感器采样配置寄存器	0x0068	R/W
版本控制寄存器			
APB_SARADC_APB_CTRL_DATE_REG	版本控制寄存器	0x03FC	R/W

36.6 寄存器

本小节的所有地址均为相对于片上传感器与模拟信号处理基地址的地址偏移量（相对地址），具体基地址请见章节 4 系统和存储器 中的表 4-2。

Register 36.1. APB_SARADC_CTRL_REG (0x0000)



APB_SARADC_START_FORCE 配置是否使用软件启动 SAR ADC 采样。

0：选择使用 FSM 启动 SAR ADC 采样

1：选择使用软件启动 SAR ADC 采样

(R/W)

APB_SARADC_START 配置是否使用软件启动 SAR ADC。

0：无效

1：使用软件启动 SAR ADC

仅当 APB_SARADC_START_FORCE = 1 时有效。

(R/W)

APB_SARADC_SAR_CLK_GATED 配置是否使能 SAR ADC 时钟门控。

0：不使能

1：使能

(R/W)

APB_SARADC_SAR_PATT_LEN 配置需要使用的样式数量。

0：仅使用 cmd0

1：使用 cmd0 和 cmd1

n：依次使用 cmd0 至 cmdn, n 最大取 7

(R/W)

APB_SARADC_SAR_PATT_P_CLEAR 配置是否清除 DIG ADC 控制器样式表指针。

0：无效

1：清除

(R/W)

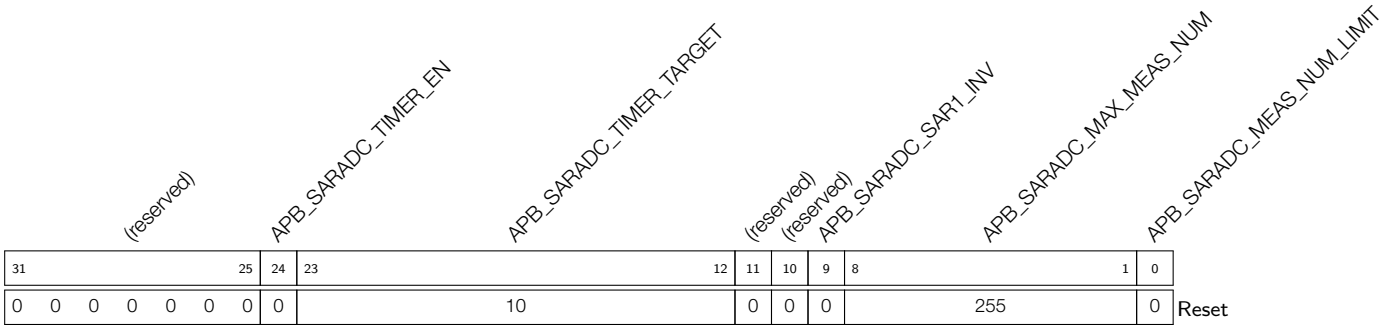
APB_SARADC_XPD_SAR_FORCE 配置是否强制上电 SAR ADC。

0：无效

1：强制上电

(R/W)

Register 36.2. APB_SARADC_CTRL2_REG (0x0004)



APB_SARADC_MEAS_NUM_LIMIT 配置是否使能 SAR ADC 最大转换次数限制。

0：不使能

1：使能

(R/W)

APB_SARADC_MAX_MEAS_NUM 配置 SAR ADC 的最大转换次数。(R/W)

APB_SARADC_SAR1_INV 配置是否反转 SAR ADC 数据。

0：无效

1：反转 SAR ADC 数据

(R/W)

APB_SARADC_TIMER_TARGET 设置 SAR ADC 定时器目标，即定时器的触发周期。(R/W)

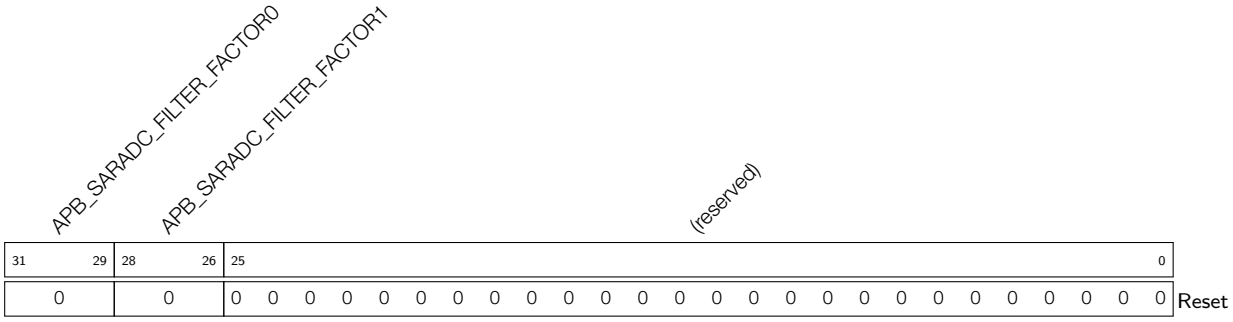
APB_SARADC_TIMER_EN 配置是否使能 SAR ADC 定时器触发。

0：不使能

1：使能

(R/W)

Register 36.3. APB_SARADC_FILTER_CTRL1_REG (0x0008)



APB_SARADC_FILTER_FACTOR1 设置 SAR ADC 滤波器 1 的滤波系数 k。

0: k=0（即滤波器关闭）

1: k=2

2: k=4

3: k=8

4: k=16

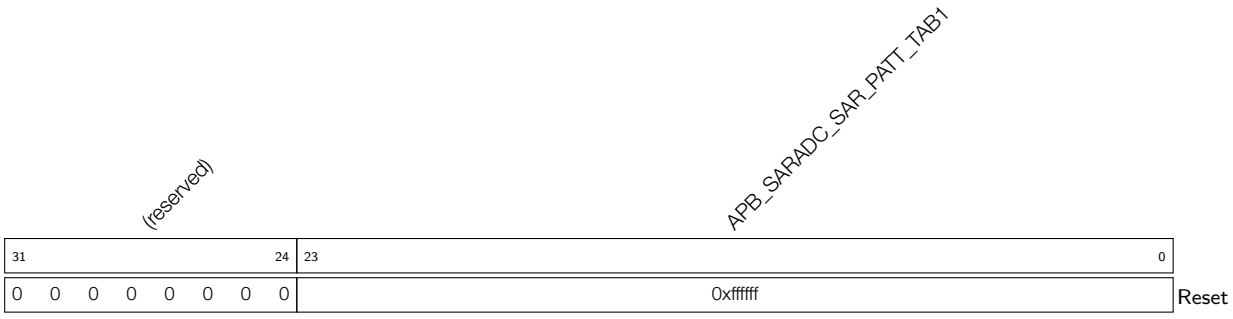
5: k=32

6: k=64

(R/W)

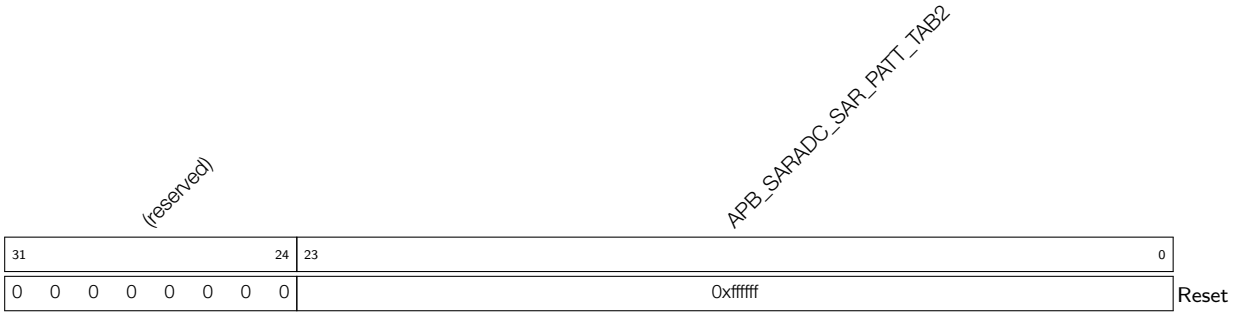
APB_SARADC_FILTER_FACTOR0 设置 SAR ADC 滤波器 0 的滤波系数 k（配置同上）。(R/W)

Register 36.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)



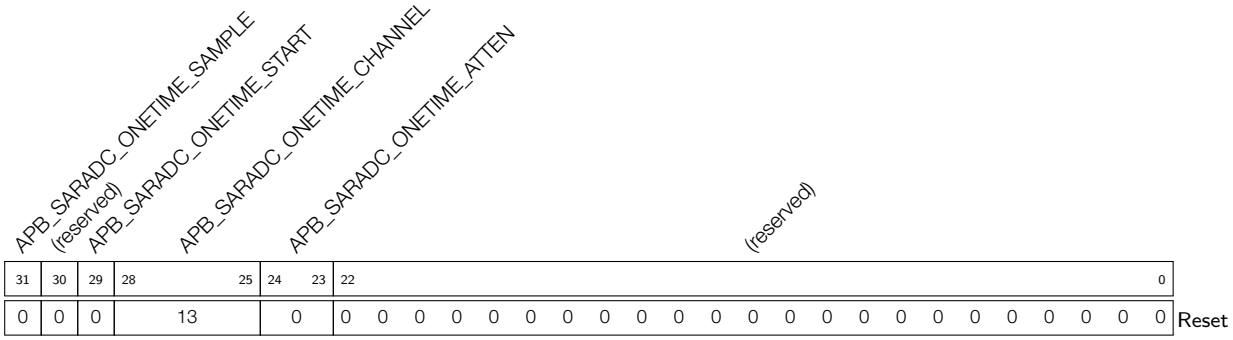
APB_SARADC_SAR_PATT_TAB1 配置样式表的第 0 ~ 3 个样式。每个样式占六位。具体见小节 36.2.4.8 样式表。(R/W)

Register 36.5. APB_SARADC_SAR_PATT_TAB2_REG (0x001C)



APB_SARADC_SAR_PATT_TAB2 配置样式表的第 4 ~ 7 个样式，每个样式占六位。具体见小节 36.2.4.8 样式表。(R/W)

Register 36.6. APB_SARADC_ONETIME_SAMPLE_REG (0x0020)



APB_SARADC_ONETIME_ATTEN 配置单次采样的衰减。

0：0 dB

1：2.5 dB

2：6 dB

3：12 dB

(R/W)

APB_SARADC_ONETIME_CHANNEL 配置单次采样的通道。

0：通道 0

1：通道 1

2：通道 2

3：通道 3

4：通道 4

(R/W)

APB_SARADC_ONETIME_START 配置是否启动 SAR ADC 单次采样。

0：无效

1：启动

(R/W)

APB_SARADC_ONETIME_SAMPLE 配置是否使能 SAR ADC 单次采样。

0：不使能

1：使能

(R/W)

Register 36.7. APB_SARADC_FILTER_CTRL0_REG (0x0028)

APB_SARADC_FILTER_CHANNEL1 配置 SAR ADC 滤波通道 1。(R/W)

APB_SARADC_FILTER_CHANNEL0 配置 SAR ADC 滤波通道 0。(R/W)

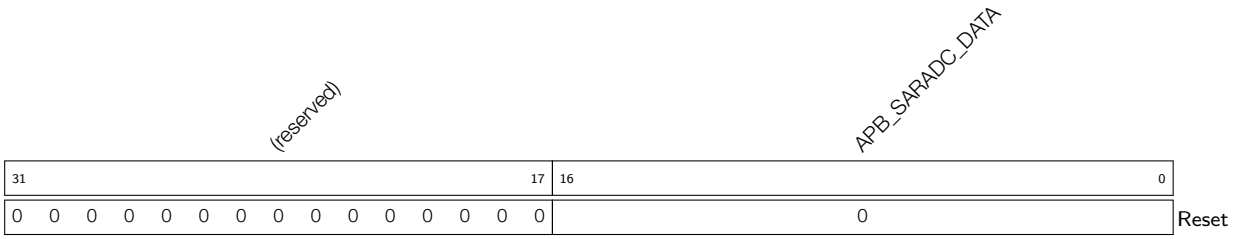
APB_SARADC_FILTER_RESET 配置是否复位 SAR ADC 滤波器。

0：无效

1：复位

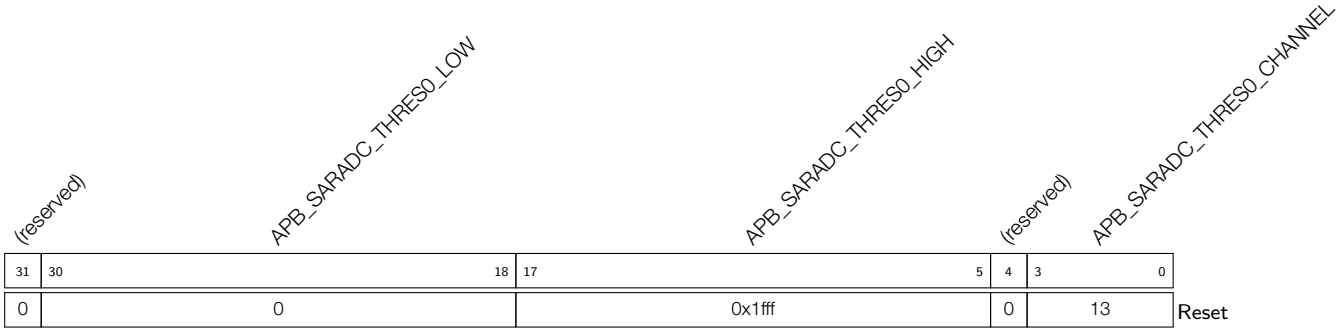
(R/W)

Register 36.8. APB_SARADC_1_DATA_STATUS_REG (0x002C)



APB_SARADC_DATA 存储 SAR ADC 单次采样的转换数据。(RO)

Register 36.9. APB_SARADC_THRES0_CTRL_REG (0x0034)

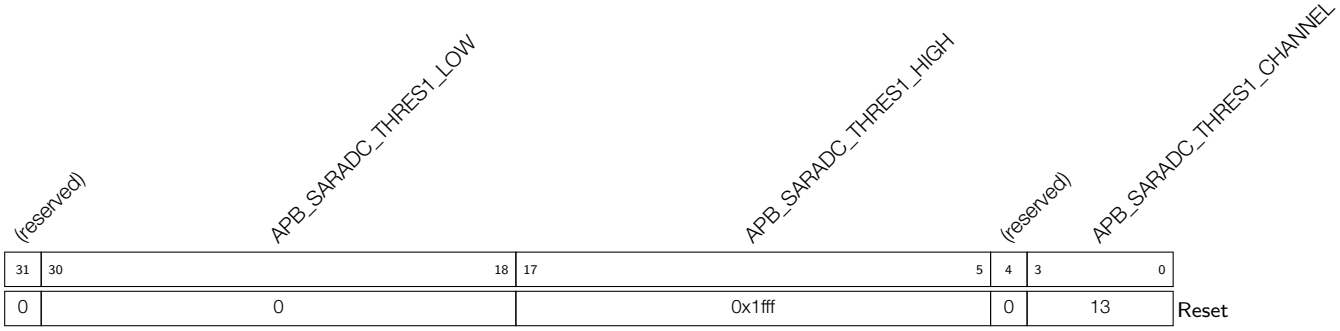


APB_SARADC_THRES0_CHANNEL 配置 SAR ADC 阈值监控器 0 需要监控的通道。(R/W)

APB_SARADC_THRES0_HIGH 配置 SAR ADC 阈值监控器 0 的高阈值。(R/W)

APB_SARADC_THRES0_LOW 配置 SAR ADC 阈值监控器 0 的低阈值。(R/W)

Register 36.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

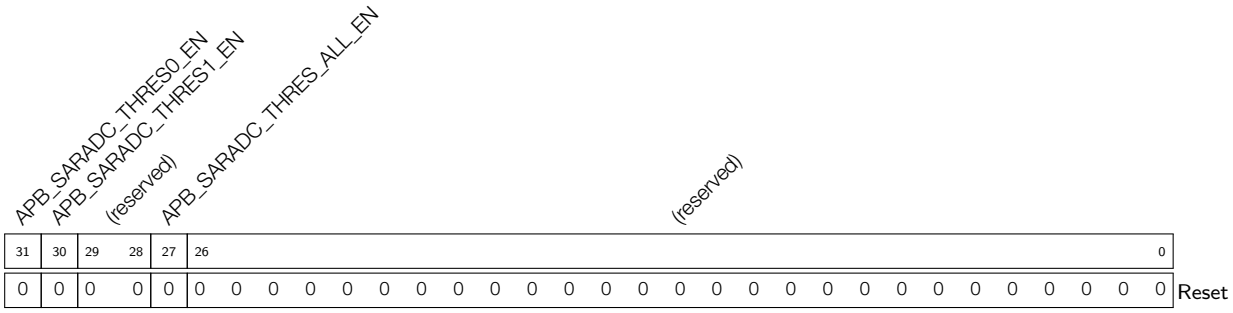


APB_SARADC_THRES1_CHANNEL 配置 SAR ADC 阈值监控器 1 需要监控的通道。(R/W)

APB_SARADC_THRES1_HIGH 配置 SAR ADC 阈值监控器 1 的高阈值。(R/W)

APB_SARADC_THRES1_LOW 配置 SAR ADC 阈值监控器 1 的低阈值。(R/W)

Register 36.11. APB_SARADC_THRES_CTRL_REG (0x003C)



APB_SARADC_THRES_ALL_EN 配置是否使能所有已配置通道的阈值监控。

0：不使能

1：使能

(R/W)

APB_SARADC_THRES1_EN 配置是否使能阈值监控器 1。

0：不使能

1：使能

(R/W)

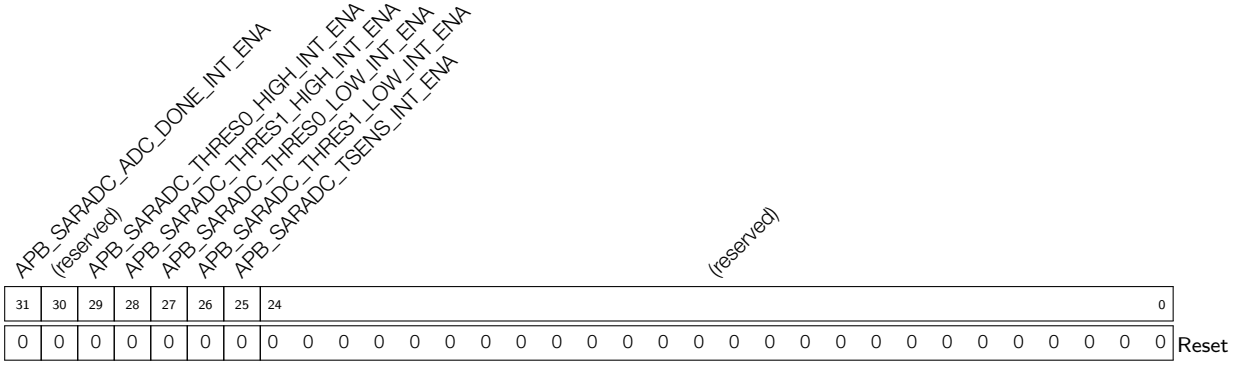
APB_SARADC_THRES0_EN 配置是否使能阈值监控器 0。

0：不使能

1：使能

(R/W)

Register 36.12. APB_SARADC_INT_ENA_REG (0x0040)



APB_SARADC_TSENS_INT_ENA 写 1 使能 APB_SARADC_TSENS_INT 中断。(R/W)

APB_SARADC_THRES1_LOW_INT_ENA 写 1 使能 APB_SARADC_THRES1_LOW_INT 中断。(R/W)

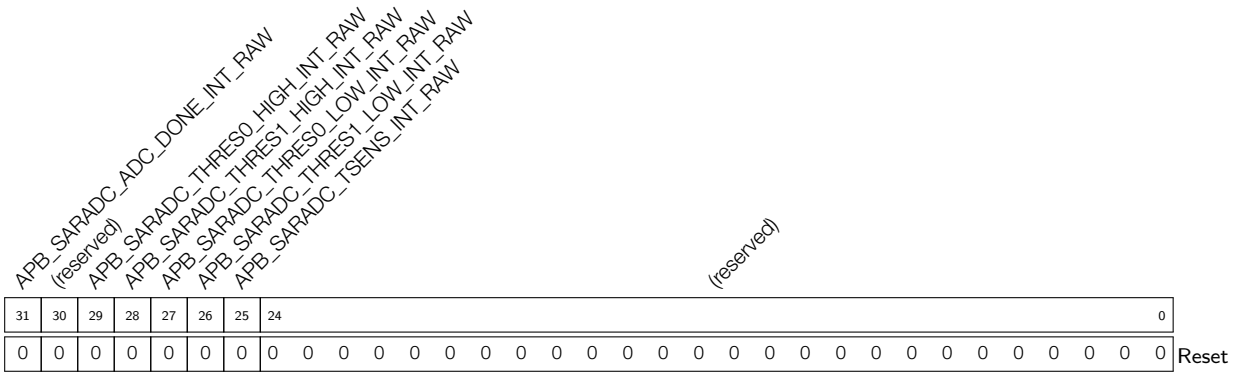
APB_SARADC_THRES0_LOW_INT_ENA 写 1 使能 APB_SARADC_THRES0_LOW_INT 中断。(R/W)

APB_SARADC_THRES1_HIGH_INT_ENA 写 1 使能 APB_SARADC_THRES1_HIGH_INT 中断。(R/W)

APB_SARADC_THRES0_HIGH_INT_ENA 写 1 使能 APB_SARADC_THRES0_HIGH_INT 中断。(R/W)

APB_SARADC_ADC_DONE_INT_ENA 写 1 使能 APB_SARADC_ADC_DONE_INT 中断。(R/W)

Register 36.13. APB_SARADC_INT_RAW_REG (0x0044)



APB_SARADC_TSENS_INT_RAW APB_SARADC_TSENS_INT 的原始中断状态。(R/WTC/SS)

APB_SARADC_THRES1_LOW_INT_RAW APB_SARADC_THRES1_LOW_INT 的原始中断状态。(RO)

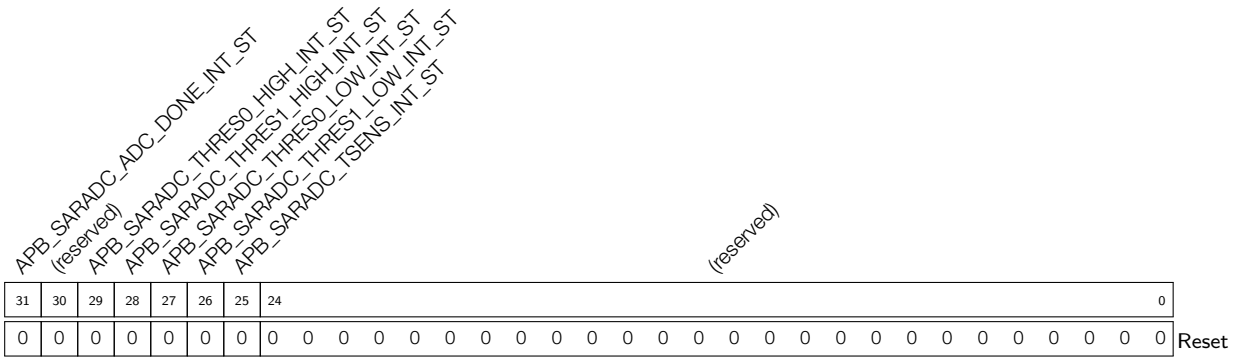
APB_SARADC_THRES0_LOW_INT_RAW APB_SARADC_THRES0_LOW_INT 的原始中断状态。(RO)

APB_SARADC_THRES1_HIGH_INT_RAW APB_SARADC_THRES1_HIGH_INT 的原始中断状态。(RO)

APB_SARADC_THRES0_HIGH_INT_RAW APB_SARADC_THRES0_HIGH_INT 的原始中断状态。(RO)

APB_SARADC_ADC_DONE_INT_RAW APB_SARADC_ADC_DONE_INT 的原始中断状态。(RO)

Register 36.14. APB_SARADC_INT_ST_REG (0x0048)



APB_SARADC_TSENS_INT_ST APB_SARADC_TSENS_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES1_LOW_INT_ST APB_SARADC_THRES1_LOW_INT 的屏蔽中断状态。(RO)

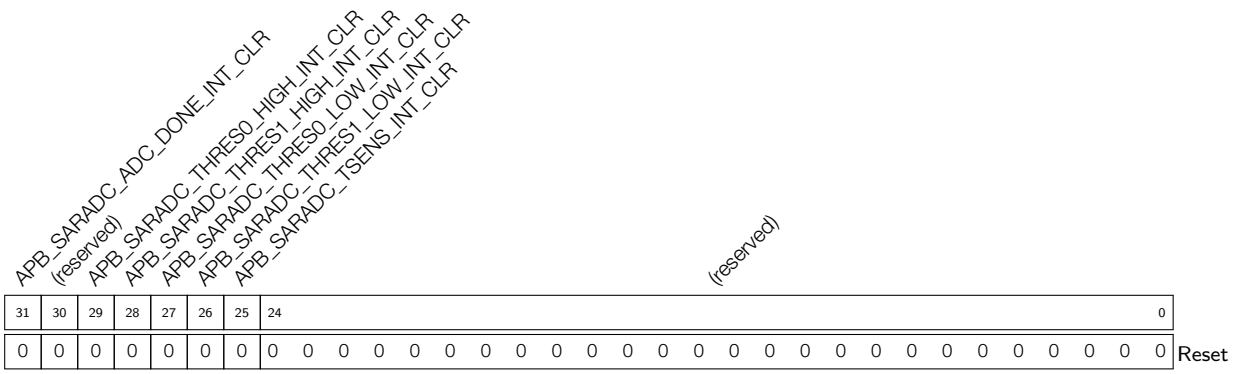
APB_SARADC_THRES0_LOW_INT_ST APB_SARADC_THRES0_LOW_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES1_HIGH_INT_ST APB_SARADC_THRES1_HIGH_INT 的屏蔽中断状态。(RO)

APB_SARADC_THRES0_HIGH_INT_ST APB_SARADC_THRES0_HIGH_INT 的屏蔽中断状态。(RO)

APB_SARADC_ADC_DONE_INT_ST APB_SARADC_ADC_DONE_INT 的屏蔽中断状态。(RO)

Register 36.15. APB_SARADC_INT_CLR_REG (0x004C)



APB_SARADC_TSENS_INT_CLR 写 1 清除 APB_SARADC_TSENS_INT 中断。(WT)

APB_SARADC_THRES1_LOW_INT_CLR 写 1 清除 APB_SARADC_THRES1_LOW_INT 中断。(WO)

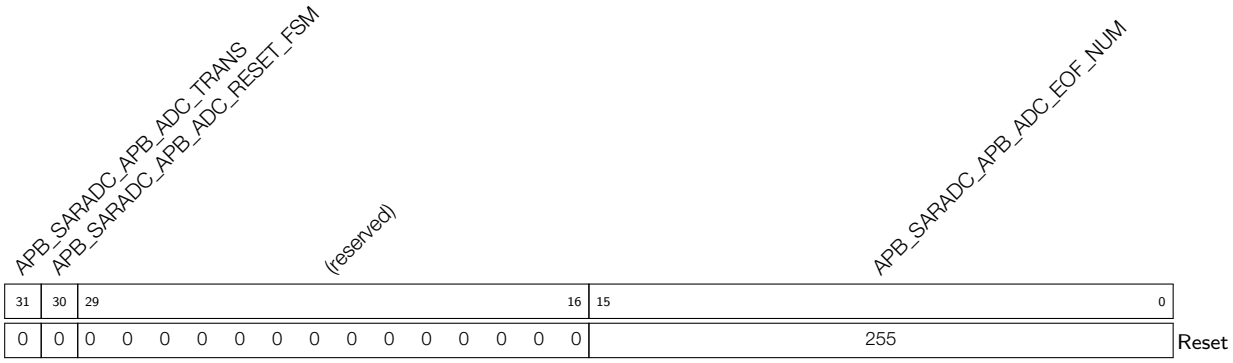
APB_SARADC_THRES0_LOW_INT_CLR 写 1 清除 APB_SARADC_THRES0_LOW_INT 中断。(WO)

APB_SARADC_THRES1_HIGH_INT_CLR 写 1 清除 APB_SARADC_THRES1_HIGH_INT 中断。(WO)

APB_SARADC_THRES0_HIGH_INT_CLR 写 1 清除 APB_SARADC_THRES0_HIGH_INT 中断。(WO)

APB_SARADC_ADC_DONE_INT_CLR 写 1 清除 APB_SARADC_ADC_DONE_INT 中断。(WO)

Register 36.16. APB_SARADC_DMA_CONF_REG (0x0050)



APB_SARADC_APB_ADC_EOF_NUM 当 ADC 采样得到的数据个数等于该寄存器配置值时，发送给 GDMA 的 EOF 标志位将被拉高。

(R/W)

APB_SARADC_APB_ADC_RESET_FSM 配置是否复位 DIG ADC 控制器状态。

0：无效

1：复位

(R/W)

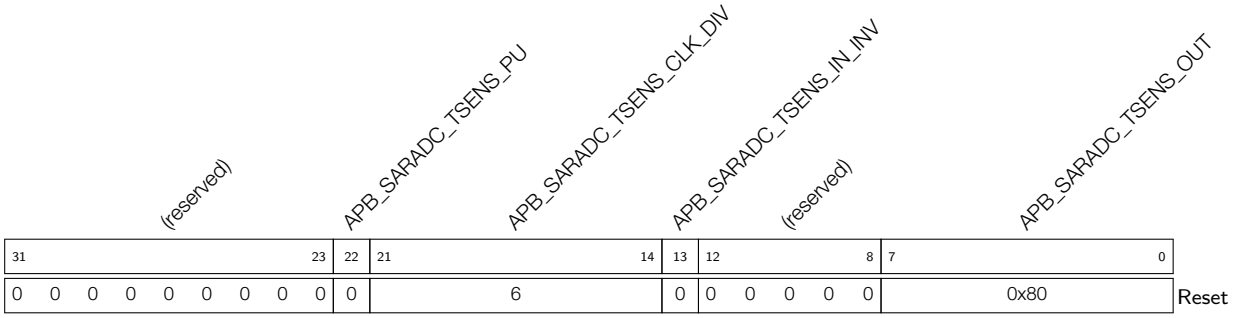
APB_SARADC_APB_ADC_TRANS 配置 DIG ADC 控制器是否使用 GDMA。

0：无效

1：使用 GDMA

(R/W)

Register 36.17. APB_SARADC_APB_TSENS_CTRL_REG (0x0058)



APB_SARADC_TSENS_OUT 表示温度传感器的输出值。(RO)

APB_SARADC_TSENS_IN_INV 配置是否反转温度传感器的输入值。

0：无效

1：反转

(R/W)

APB_SARADC_TSENS_CLK_DIV 表示温度传感器的时钟分频系数。(R/W)

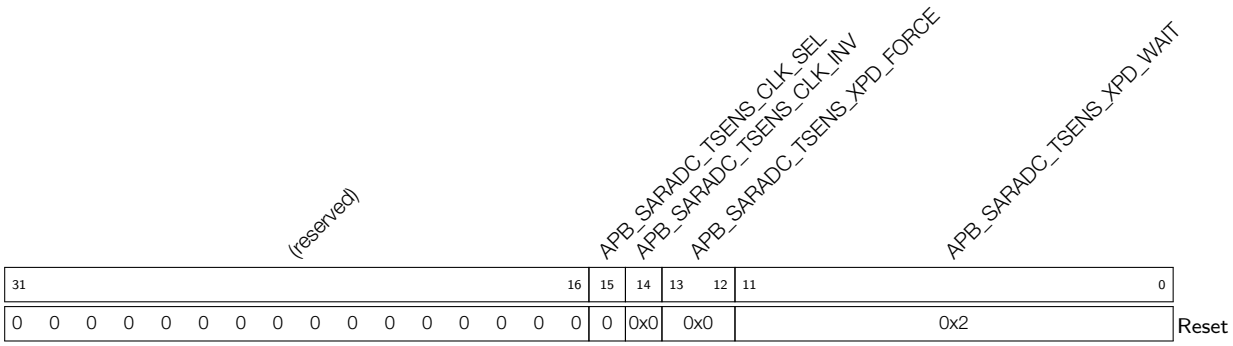
APB_SARADC_TSENS_PU 配置是否使能温度传感器上电。

0：无效

1：使能

(R/W)

Register 36.18. APB_SARADC_APB_TSENS_CTRL2_REG (0x005C)



APB_SARADC_TSENS_CLK_SEL 配置温度传感器时钟。

0：RC_FAST_CLK

1：XTAL_CLK

(R/W)

APB_SARADC_TSENS_CLK_INV 配置传感器采样时钟的相位。(R/W)

APB_SARADC_TSENS_XPD_FORCE 配置是否使能强制上电/断电温度传感器。

0：禁用强制上电功能

1：禁用强制断电功能

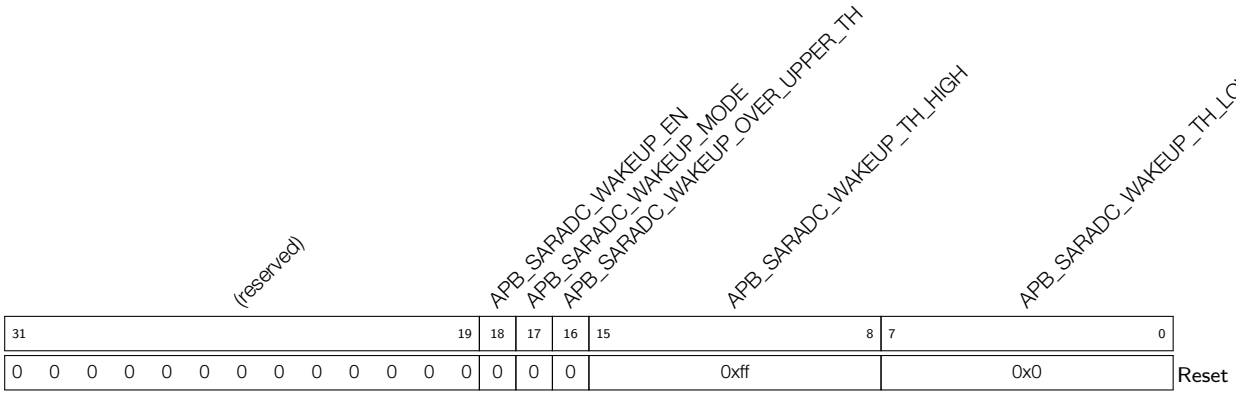
2：强制上电温度传感器

3：强制断电温度传感器

(R/W)

APB_SARADC_TSENS_XPD_WAIT 配置温度传感器的复位释放前需要等待的时钟周期。(R/W)

Register 36.19. APB_TSENS_WAKE_REG (0x0064)



APB_SARADC_WAKEUP_TH_LOW 配置温度传感器唤醒功能的低阈值。(R/W)

APB_SARADC_WAKEUP_TH_HIGH 配置温度传感器唤醒功能的高阈值。(R/W)

APB_SARADC_WAKEUP_OVER_UPPER_TH 表示输出值是否超过阈值。

0：输出值低于低阈值

1：输出值高于高阈值

(RO)

APB_SARADC_WAKEUP_MODE 配置温度监测唤醒模式。仅当 APB_SARADC_WAKEUP_EN=1 时有效。

0：绝对值模式

1：变化量模式

(R/W)

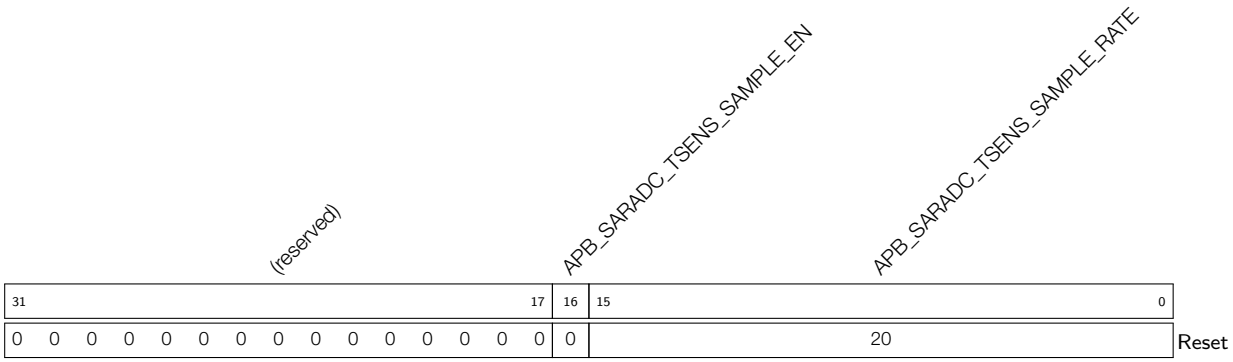
APB_SARADC_WAKEUP_EN 配置是否启用唤醒功能。

0：禁用

1：启用

(R/W)

Register 36.20. APB_TSENS_SAMPLE_REG (0x0068)



APB_SARADC_TSENS_SAMPLE_RATE 配置硬件自动温度监测的采样率。采样周期 = 配置值 × 传感器工作时钟周期。(R/W)

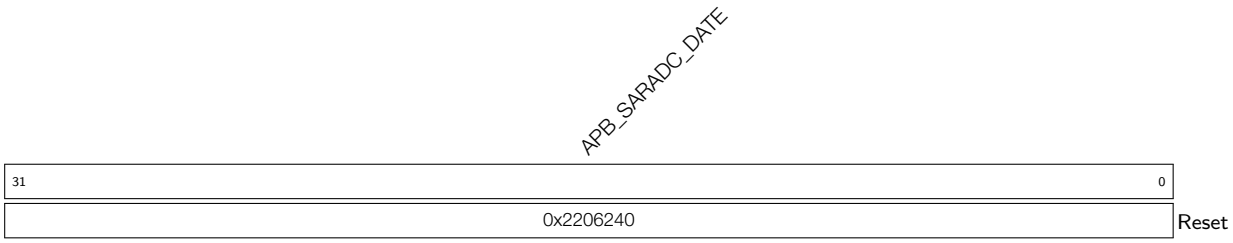
APB_SARADC_TSENS_SAMPLE_EN 配置是否自动触发持续温度监测。

0：不使能

1：使能

(R/W)

Register 36.21. APB_SARADC_CTRL_DATE_REG (0x03FC)



APB_SARADC_DATE SAR ADC 和温度传感器的版本控制寄存器。(R/W)

37 相关文档和资源

相关文档

- 《ESP32-H2 技术规格书》– 提供 ESP32-H2 芯片的硬件技术规格。
- 《ESP32-H2 硬件设计指南》– 提供基于 ESP32-H2 芯片的产品设计规范。
- 证书
  https://espressif.com/zh-hans/support/documents/certificates
- ESP32-H2 产品/工艺变更通知 (PCN)
  https://espressif.com/zh-hans/support/documents/pcns?keys=ESP32-H2
- ESP32-H2 公告 – 提供有关安全、bug、兼容性、器件可靠性的信息
  https://espressif.com/zh-hans/support/documents/advisories?keys=ESP32-H2
- 文档更新和订阅通知
  https://espressif.com/zh-hans/support/download/documents

开发者社区

- 《ESP32-H2 ESP-IDF 编程指南》– ESP-IDF 开发框架的文档中心。
- ESP-IDF 及 GitHub 上的其它开发框架
  https://github.com/espressif
- ESP32 论坛 – 工程师对工程师 (E2E) 的社区，您可以在这里提出问题、解决问题、分享知识、探索观点。
  https://esp32.com/
- *The ESP Journal* – 分享乐鑫工程师的最佳实践、技术文章和工作随笔。
  https://blog.espressif.com/
- SDK 和演示、App、工具、AT 等下载资源
  https://espressif.com/zh-hans/support/download/sdks-demos

产品

- ESP32-H2 系列芯片 – ESP32-H2 全系列芯片。
  https://espressif.com/zh-hans/products/socs?id=ESP32-H2
- ESP32-H2 系列模组 – ESP32-H2 全系列模组。
  https://espressif.com/zh-hans/products/modules?id=ESP32-H2
- ESP32-H2 系列开发板 – ESP32-H2 全系列开发板。
  https://espressif.com/zh-hans/products/devkits?id=ESP32-H2
- ESP Product Selector（乐鑫产品选型工具）– 通过筛选性能参数、进行产品对比快速定位您所需要的产品。
  https://products.espressif.com/#/product-selector?language=zh

联系我们

- 商务问题、技术支持、电路原理图 & PCB 设计审阅、购买样品（线上商店）、成为供应商、意见与建议
  https://espressif.com/zh-hans/contact-us/sales-questions

词汇列表

外设相关词汇

AES	AES 加速器
DSA	数字签名算法
DMA	DMA 控制器
ECC	ECC 加速器
eFuse	eFuse 控制器
ETM	事件任务矩阵
HMAC	HMAC 加速器
I2C	I2C 控制器
I2S	I2S 控制器
LEDC	LED PWM 控制器
MCPWM	电机控制脉宽调制器
PARLIO	并行 IO 控制器
PCNT	脉冲计数器控制器
RMT	红外遥控
RNG	随机数生成器
RSA	RSA 加速器
SDIO	SDIO 2.0 从机控制器
SHA	SHA 加速器
SPI	SPI 控制器
SYSTIMER	系统定时器
TIMG	定时器组
TWAI	双线汽车接口
UART	UART 控制器
WDT	看门狗定时器

寄存器相关缩写

REG	**寄存器**。
SYSREG	**系统寄存器**是一组控制系统复位、存储器、时钟、软件中断、电源管理、时钟门控等的寄存器。
ISO	**隔离**。如果外设或其他芯片组件断电，其输出信号的管脚（若有）将会浮空。ISO 寄存器会隔离上述引脚并令其保持在某个确定值，以使连接到这些引脚的其他非断电外设/设备免受影响。
NMI	**非屏蔽中断**是一种 CPU 指令无法禁用或忽略的硬件中断。出现此类中断说明发生严重错误。
W1TS	添加到寄存器/字段名称中的缩写，表示此类寄存器/字段用于置位名称相似寄存器中的相应字段。例如，寄存器 `GPIO_ENABLE_W1TS_REG` 用于置位寄存器 `GPIO_ENABLE_REG` 中的相应字段。
W1TC	与 *W1TS* 相同，但用于清除相应寄存器中的字段。

寄存器的访问类型

TRM 章节 寄存器列表 和 寄存器 详述了寄存器及其字段的访问类型。

常用访问类型及组合如下：

- RO
- WO
- WT
- R/W
- WL
- R/W/SC
- R/W/SS
- R/W/SS/SC
- R/WC/SS
- R/WC/SC
- R/WC/SS/SC
- R/WS/SC
- R/WS/SS
- R/WS/SS/SC
- R/SS/WTC
- R/SC/WTC
- R/SS/SC/WTC
- RF/WF
- R/SS/RC
- varies

下文提供了所有访问类型的具体描述。

R **软件可读**。用户软件可以读取此寄存器/字段；通常与其他访问类型结合使用。

RO **软件只读**。用户软件只可读取此寄存器/字段。

HRO **硬件只读**。仅硬件可以读取此寄存器/字段；用于存储变量参数的默认设置。

W **软件可写**。用户软件可以写入此寄存器/字段；通常与其他访问类型结合使用。

WO **软件只写**。用户软件只可写入此寄存器/字段。

SS **硬件置位**。在指定事件中，硬件自动将 1 写入此寄存器/字段；与一位字段一同使用。

SC **硬件清零**。在指定事件中，硬件自动将 0 写入此寄存器/字段；与一位和多位字段一同使用。

SM **硬件修改**。在指定事件中，硬件自动将指定值写入此寄存器/字段；与多位字段一同使用。

RS **软件读置位**。如果用户软件读取此寄存器/字段，硬件会自动写 1。

RC **软件读清零**。如果用户软件读取此寄存器/字段，硬件会自动写 0。

RF **软件读 FIFO**。如果用户软件将新数据写入 FIFO，寄存器/字段会自动读取。

WF **软件写 FIFO**。如果用户软件将新数据写入此寄存器/字段，寄存器/字段会自动通过 APB 总线将数据传递到 FIFO。

WS **软件写置位**。如果用户软件写入此寄存器/字段，硬件会自动置位此寄存器/字段。

W1S **软件写 1 置位**。如果用户软件将 1 写入此寄存器/字段，硬件会自动置位此寄存器/字段。

W0S **软件写 0 置位**。如果用户软件将 0 写入此寄存器/字段，硬件会自动置位此寄存器/字段。

WC **软件写清零**。如果用户软件写入此寄存器/字段，硬件会自动清零此寄存器/字段。

W1C **软件写 1 清零**。如果用户软件将 1 写入此寄存器/字段，硬件会自动清零此寄存器/字段。

W0C **软件写 0 清零**。如果用户软件将 0 写入此寄存器/字段，硬件会自动清零此寄存器/字段。

WT **软件写产生边沿触发信号**。如果用户软件将 1 写入此字段，将会产生边沿触发信号（APB 总线中的脉冲）或清除相应的 WTC 字段（详见 WTC）。

WTC **软件写其他寄存器位清零本寄存器位**。如果用户软件将 1 写入相应的 WT 字段，硬件会自动清除此字段（详见 WT）。

W1T **软件写 1 取反**。如果用户软件将 1 写入此字段，硬件会自动取反相应字段，否则不会取反。

W0T **软件写 0 取反**。如果用户软件将 0 写入此字段，硬件会自动取反相应字段，否则不会取反。

WL **软件仅在锁禁用时写**。如果锁被禁用，用户软件可以写入此寄存器/字段。

varies **访问类型不定**。此寄存器中的不同字段访问类型可能不同。

如何配置寄存器的保留域

概述

寄存器的保留域指的是寄存器中不对用户开放、或者配置为非默认值时会导致不可预测的结果的域。

如何配置保留域

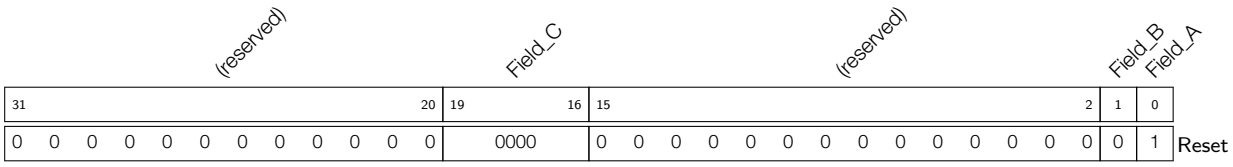
保留域的值不能修改。由于写寄存器时必须整体写，不能只写部分域，因此，在写带有保留域的寄存器时需要特别注意，只能采用以下两种方式：

1. 读取寄存器的值，仅修改需要配置的域，然后将修改后的值以及其他未修改的值一起写回寄存器，这样保留域的值就会保持不变。

   或者

2. 仅修改需要配置的域，然后将保留域的默认值写回寄存器。默认值即寄存器图表中的“Reset”值。例如，寄存器 X 中 Field_A 的默认值为 1。

Register 37.1. 寄存器 X (地址)



假设您要将 寄存器 X 的 Field_A、Field_B 和 Field_C 设置为 0x0、0x1 和 0x2，您可以：

- 使用第 1 种方式，修改这三个域的值，然后把读到的值写回寄存器。假设寄存器读取的值为 0x0000_0003，修改三个域的值后，将值 0x0002_0002 写入寄存器。
- 使用第 2 种方式，修改这三个域的值，然后把保留域的默认值写回寄存器，即把 0x0002_0002 写入寄存器。

中断配置寄存器

大部分外设的内部中断源都有以下配置寄存器：

- **RAW**（原始状态）寄存器：该寄存器指示原始中断状态，每个位对应一个内部中断源。当中断源触发时，其 RAW 位为 1。
- **ENA**（使能）寄存器：该寄存器用于启用或禁用内部中断源，每个位对应一个内部中断源。

  通过操作 ENA 寄存器，可以根据需要屏蔽或取消屏蔽某个内部中断源。当中断源被屏蔽（禁用）时，它不会生成中断信号，但仍可以从 RAW 寄存器中读取其值。
- **ST**（状态）寄存器：该寄存器指示中断源的屏蔽状态，每个位对应一个内部中断源。ST 位为 1 代表 RAW 位和 ENA 位都为 1，即中断源已生成且未被屏蔽。RAW 位和 ENA 位的值为其他组合时，ST 位为 0。

  ENA/RAW/ST 寄存器的配置见表 37-4。
- **CLR**（清除）寄存器：CLR 寄存器负责清除内部中断源。写 1 将清除该位对应的中断源。

表 37-4. ENA/RAW/ST 寄存器的配置

ENA 位的值	RAW 位的值	ST 位的值
0	忽略	0
1	0	0
	1	1

修订历史

日期	版本	发布说明
2023-10-12	v0.4	新增章节 23 *椭圆曲线数字签名算法 (ECDSA)* 更新章节 7 *复位和时钟*：新增 PCR_FOSC_TICK_NUM 字段的描述
2023-08-02	v0.3	新增章节 6 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 更新以下章节： • 章节 8 *芯片 Boot 控制*：更正 8.2.2 *Boot 模式控制* • 章节 9 *中断矩阵 (INTMTX)*：更新章节 9.2 *ESP32-H2 中断术语* • 章节 10 *事件任务矩阵 (SOC_ETM)*：将支持事件任务矩阵的外设由 RTC 看门狗定时器改为 RTC 定时器 新增章节 *中断配置寄存器*
2023-06-29	v0.2	新增以下章节： • 章节 2 *RISC-V 追踪编码器 (TRACE)* • 章节 7 *复位和时钟* • 章节 18 *ECC 加速器 (ECC)* • 章节 28 *I2S 控制器 (I2S)* 更新以下章节： • 章节 1 *ESP-RISC-V CPU*：更正中断 ID • 章节 5 *eFuse 控制器 (EFUSE)*：更新 EFUSE_SEC_DPA_LEVEL 寄存器的描述 • 章节 9 *中断矩阵 (INTMTX)*：删除 MSPI_INTR 中断源及其映射寄存器，原因是 MSPI 模块暂未作说明 • 章节 14 *访问权限管理 (APM)*：在表 14-1 和图 14-1 中增加 EX_MEM，从章节 14.2 *主要特性* 中删除“外部存储器”，因为 APM 没有对其的权限管理 新增章节 *如何配置寄存器的保留域*
2023-05-24	v0.1	首次发布

www.espressif.com

免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，乐鑫不对信息的准确性、真实性做任何保证。

乐鑫不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他乐鑫提案、规格书或样品在他处提到的任何保证。

乐鑫不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。