ESP8684

技术参考手册

版本 1.2

乐鑫信息科技

关于本文档

ESP8684 技术参考手册面向使用 ESP8684 系列产品进行底层软件开发的人员，介绍了 ESP8684 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

No.	ESP8684 章节	最新进度
1	ESP-RISC-V CPU	已发布
2	通用 DMA 控制器 (GDMA)	已发布
3	系统和存储器	已发布
4	eFuse 控制器 (eFuse)	已发布
5	IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)	已发布
6	复位和时钟	已发布
7	芯片 Boot 控制	已发布
8	中断矩阵 (INTMTRX)	已发布
9	低功耗管理 (RTC_CNTL)	已发布
10	系统定时器 (SYSTIMER)	已发布
11	定时器组 (TIMG)	已发布
12	看门狗定时器 (WDT)	已发布
13	系统寄存器 (SYSTEM)	已发布
14	辅助调试 (ASSIST_DEBUG)	已发布
15	ECC 硬件加速器 (ECC)	已发布
16	SHA 加速器 (SHA)	已发布
17	片外存储器加密与解密 (XTS_AES)	已发布
18	随机数发生器 (RNG)	已发布
19	UART 控制器 (UART)	已发布
20	SPI 控制器 (SPI)	已发布
21	I2C 主机控制器 (I2C)	已发布
22	LED PWM 控制器 (LEDC)	已发布
23	片上传感器与模拟信号处理	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/esp8684_technical_reference_manual_cn.pdf

目录

表格

插图

1 ESP-RISC-V CPU

1.1 概述

ESP-RISC-V CPU 是基于 RISC-V ISA 的 32 位内核，包括基本整数 (I)，乘法/除法 (M) 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 内核架构包含中断控制器 (INTC)、调试模块 (DM)，以及用于访问存储器和外设的系统总线 (SYS BUS) 接口。

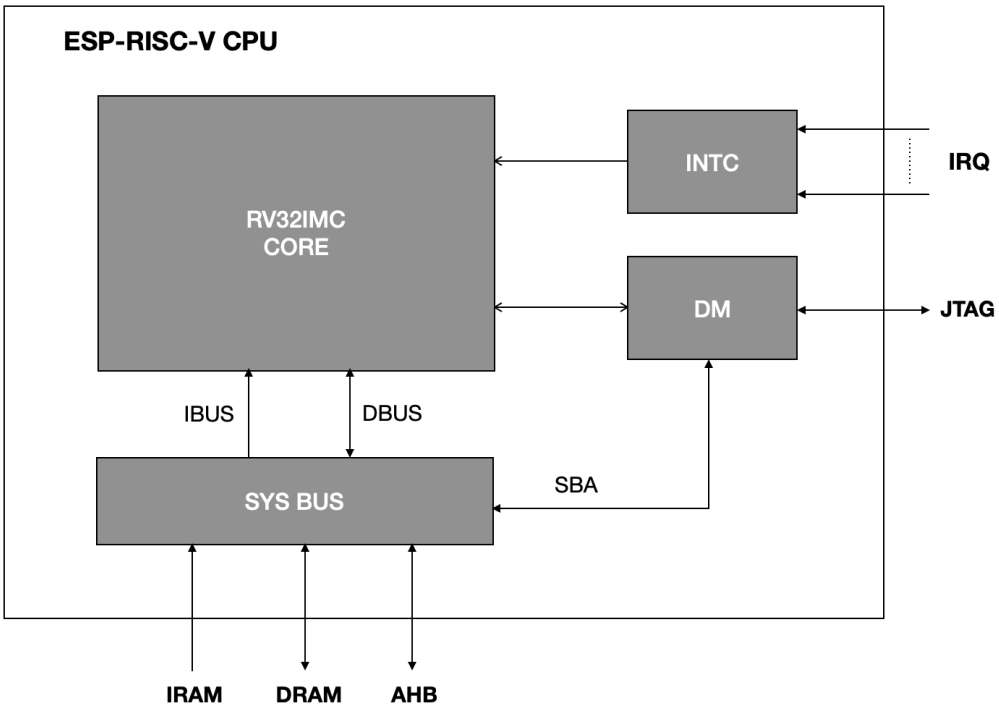


图 1-1. CPU 框图

1.2 特性

ESP-RISC-V CPU 具有如下特性：

- 时钟工作频率高达 120 MHz
- 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
- 中断控制器 (INTC) 具有多达 31 个向量中断，可配置优先级和阈值级别
- 调试模块 (DM) 符合 RISC-V 调试规范 v0.13，支持通过行业标准的 JTAG/USB 端口连接外部调试器
- 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 2 个断点/观察点
- 物理存储器保护 (PMP)，支持 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 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 1-1. CPU 地址分布

名称	描述	起始地址	结束地址	访问
IRAM	指令地址空间	0x4000_0000	0x47FF_FFFF	读/写
DRAM	数据地址空间	0x3800_0000	0x3FFF_FFFF	读/写
DM	调试地址空间	0x2000_0000	0x27FF_FFFF	读/写
AHB	AHB 地址空间	* 默认	* 默认	读/写

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

1.4 配置与状态寄存器 (CSR)

1.4.1 寄存器列表

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

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	只读
marchid	机器模式架构编号寄存器	0xF12	只读
mimpid	机器模式硬件实现编号寄存器	0xF13	只读
mhartid	机器模式硬件线程编号寄存器	0xF14	只读
机器模式异常设置 CSR			
mstatus	机器模式状态寄存器	0x300	读/写
misa [1]	机器模式 ISA 寄存器	0x301	读/写
mtvec [2]	机器模式异常向量寄存器	0x305	读/写
机器模式异常处理 CSR			
mscratch	机器模式暂存寄存器	0x340	读/写
mepc	机器模式异常程序计数器	0x341	读/写
mcause [3]	机器模式异常原因寄存器	0x342	读/写
mtval	机器模式异常值寄存器	0x343	读/写
物理存储器保护 (PMP) CSR			
pmpcfg0	物理存储器保护配置寄存器	0x3A0	读/写
pmpcfg1	物理存储器保护配置寄存器	0x3A1	读/写
pmpcfg2	物理存储器保护配置寄存器	0x3A2	读/写
pmpcfg3	物理存储器保护配置寄存器	0x3A3	读/写

[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。

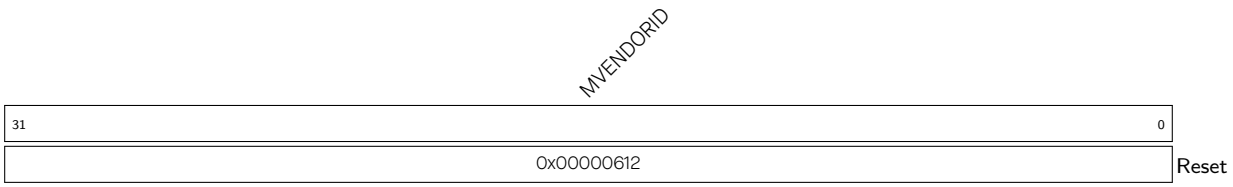
名称	描述	地址	访问
pmpaddr0	物理存储器保护地址	0x3B0	读/写
pmpaddr1	物理存储器保护地址	0x3B1	读/写
pmpaddr2	物理存储器保护地址	0x3B2	读/写
pmpaddr3	物理存储器保护地址	0x3B3	只读
pmpaddr4	物理存储器保护地址	0x3B4	只读
pmpaddr5	物理存储器保护地址	0x3B5	只读
pmpaddr6	物理存储器保护地址	0x3B6	只读
pmpaddr7	物理存储器保护地址	0x3B7	只读
pmpaddr8	物理存储器保护地址	0x3B8	只读
pmpaddr9	物理存储器保护地址	0x3B9	只读
pmpaddr10	物理存储器保护地址	0x3BA	只读
pmpaddr11	物理存储器保护地址	0x3BB	只读
pmpaddr12	物理存储器保护地址	0x3BC	只读
pmpaddr13	物理存储器保护地址	0x3BD	只读
pmpaddr14	物理存储器保护地址	0x3BE	只读
pmpaddr15	物理存储器保护地址	0x3BF	只读
触发器模块 CSR（与调试模式共用）			
tselect	触发器选择寄存器	0x7A0	读/写
tdata1	触发器抽象数据寄存器 1	0x7A1	读/写
tdata2	触发器抽象数据寄存器 2	0x7A2	读/写
tcontrol	全局触发器控制寄存器	0x7A5	读/写
调试模式 CSR			
dcsr	调试模式控制与状态寄存器	0x7B0	读/写
dpc	调试模式 PC 寄存器	0x7B1	读/写
dscratch0	调试模式暂存寄存器 0	0x7B2	读/写
dscratch1	调试模式暂存寄存器 1	0x7B3	读/写
性能计数器 CSR（自定义）[4]			
mpcer	性能计数器事件寄存器	0x7E0	读/写
mpcmr	性能计数器模式寄存器	0x7E1	读/写
mpccr	性能计数器计数寄存器	0x7E2	读/写
GPIO 访问 CSR（自定义）			
cpu_gpio_oen	GPIO 输出使能寄存器	0x803	读/写
cpu_gpio_in	GPIO 读输入值寄存器	0x804	只读
cpu_gpio_out	GPIO 写输出值寄存器	0x805	读/写

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。

[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。

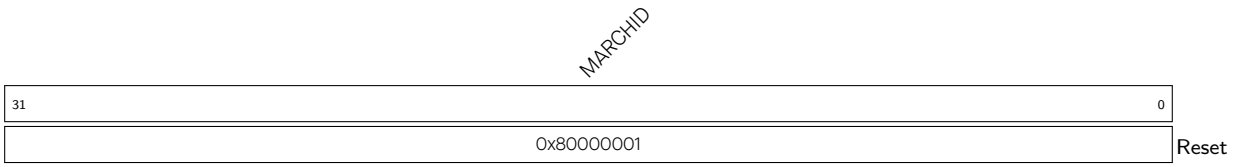
1.4.2 寄存器

Register 1.1. mvendorid (0xF11)



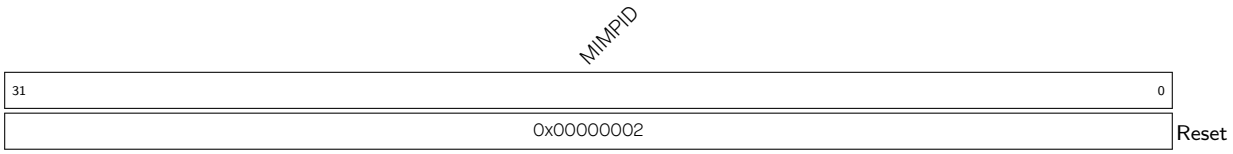
MVENDORID 供应商编号。(只读)

Register 1.2. marchid (0xF12)



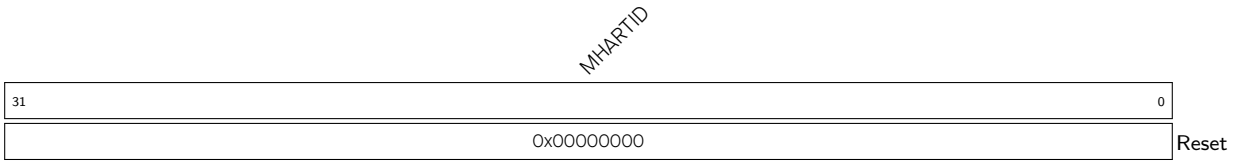
MARCHID 架构编号。(只读)

Register 1.3. mimpid (0xF13)



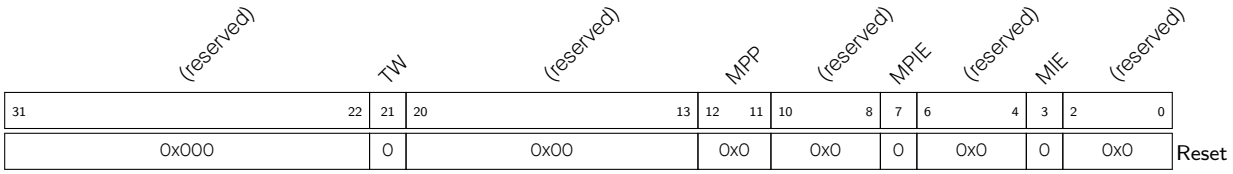
MIMPID 实现编号。(只读)

Register 1.4. mhartid (0xF14)



MHARTID 硬件线程编号。(只读)

Register 1.5. mstatus (0x300)



MIE 全局机器模式中断使能。(读/写)

MPIE 之前的 MIE。(读/写)

MPP 机器之前的特权模式。(读/写)

可能的值:

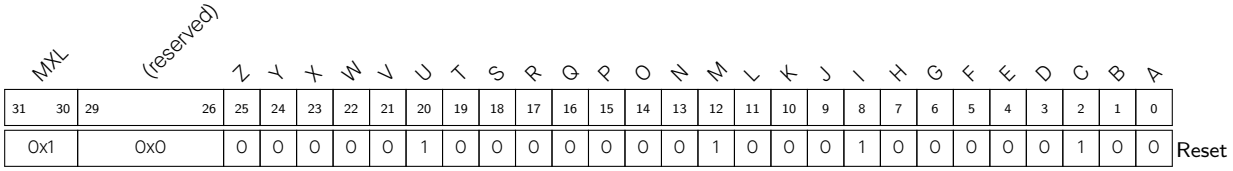
- 0x0：用户模式
- 0x3：机器模式

说明：仅低位可写。由于高位直接绑定低位，写入高位将被忽略。

TW 超时等待。(读/写)

如果该位置 1，用户模式下的 WFI（等待中断）指令将导致非法指令异常。

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	0	Reset

MXL 机器 XLEN = 1（32 位）。（只读）

Z 保留 = 0。（只读）

Y 保留 = 0。（只读）

X 非标准扩展 = 0。（只读）

W 保留 = 0。（只读）

V 保留 = 0。（只读）

U 实现用户模式 = 1。（只读）

T 保留 = 0。（只读）

S 实现监督模式 = 0。（只读）

R 保留 = 0。（只读）

Q 四精度浮点扩展 = 0。（只读）

P 保留 = 0。（只读）

O 保留 = 0。（只读）

N 支持用户级别中断 = 0。（只读）

M 整数乘除法标准扩展 = 1。（只读）

L 保留 = 0。（只读）

K 保留 = 0。（只读）

J 保留 = 0。（只读）

I RV32I 基本 ISA = 1。（只读）

H 虚拟机管理程序扩展 = 0。（只读）

G 其他标准扩展 = 0。（只读）

F 单精度浮点扩展 = 0。（只读）

E RV32E 基本 ISA = 0。（只读）

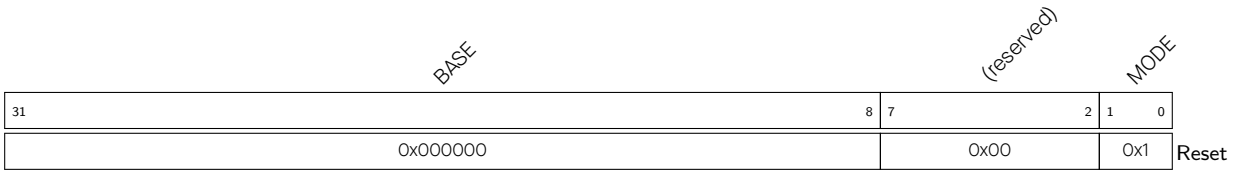
D 双精度浮点扩展 = 0。（只读）

C 压缩标准扩展 = 1。（只读）

B 保留 = 0。（只读）

A 原子标准扩展 = 0。（只读）

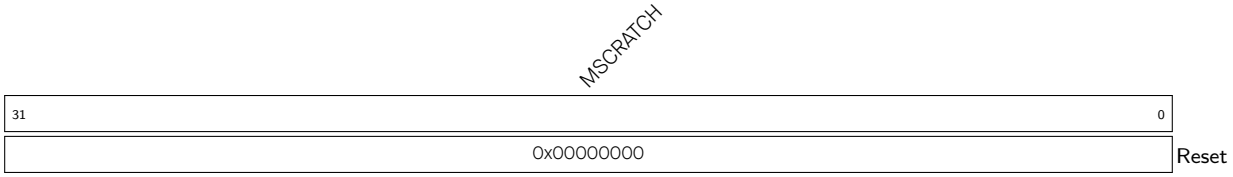
Register 1.7. mtvec (0x305)



MODE 仅支持向量模式 0x1。(只读)

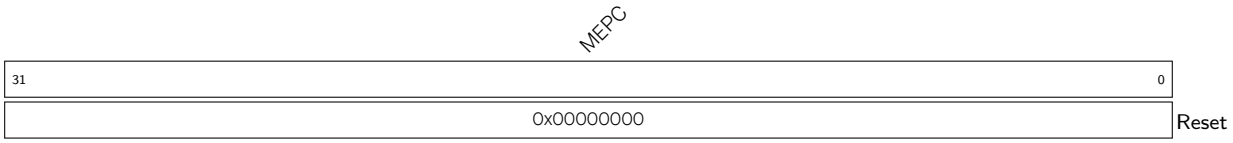
BASE 异常向量基址的高 24 位为 256 字节对齐。(读/写)

Register 1.8. mscratch (0x340)



MSCRATCH 用户自定义的机器暂存寄存器。(读/写)

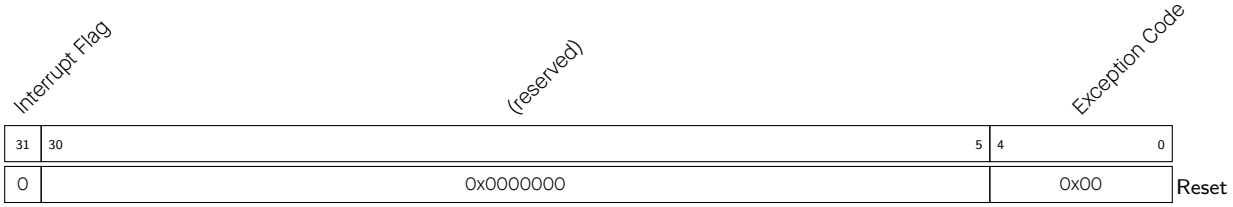
Register 1.9. mepc (0x341)



MEPC 机器陷阱/异常程序计数器。(读/写)

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。

Register 1.10. 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。(读/写)

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
- 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

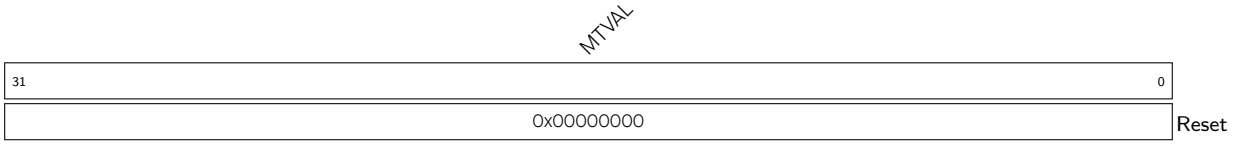
说明：异常 *ID 0x0*（指令地址非对齐）不存在，因为 *CPU* 在取指时始终屏蔽地址的最低位。

Interrupt Flag CPU 进入异常时，此标志位将自动更新。(读/写)

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。

说明：中断控制器将中断编号 *1-31* 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*。

Register 1.11. mtval (0x343)



MTVAL 机器模式异常值。(读/写)

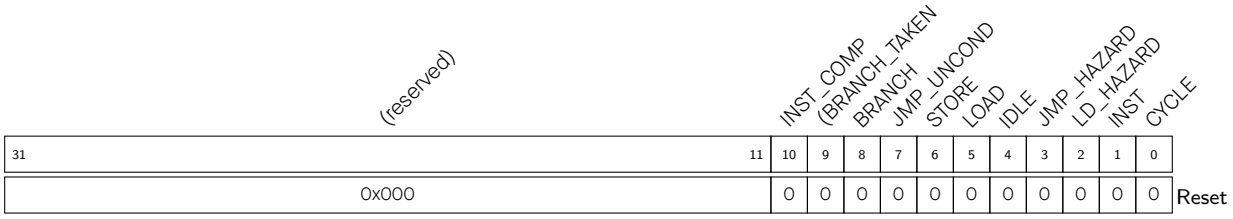
将自动更新为与异常有关的数据，该数据可能有助于处理该异常。

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

说明：该寄存器不支持其他异常 *ID* 和中断。

Register 1.12. mpcer (0x7E0)



INST_COMP 计数压缩指令。(读/写)

BRANCH_TAKEN 计数跳转的分支。(读/写)

BRANCH 计数分支。(读/写)

JMP_UNCOND 计数无条件跳转。(读/写)

STORE 计数存储器写操作。(读/写)

LOAD 计数存储器读操作。(读/写)

IDLE 计数 IDLE 周期。(读/写)

JMP_HAZARD 计数跳转冲突。(读/写)

LD_HAZARD 计数存储器读操作冲突。(读/写)

INST 计数指令。(读/写)

CYCLE 计数时钟周期，WFI 模式下周期计数不增加。(读/写)

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*。

Register 1.13. mpcmr (0x7E1)

COUNT_SAT 计数器饱和控制。(读/写)

可能的值：

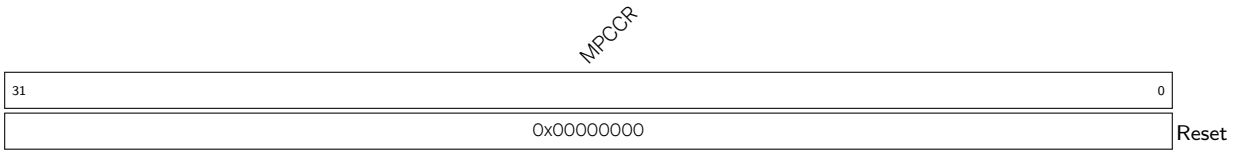
- 0：超出最大值上溢
- 1：超出最大值暂停

COUNT_EN 计数器使能控制。(读/写)

可能的值：

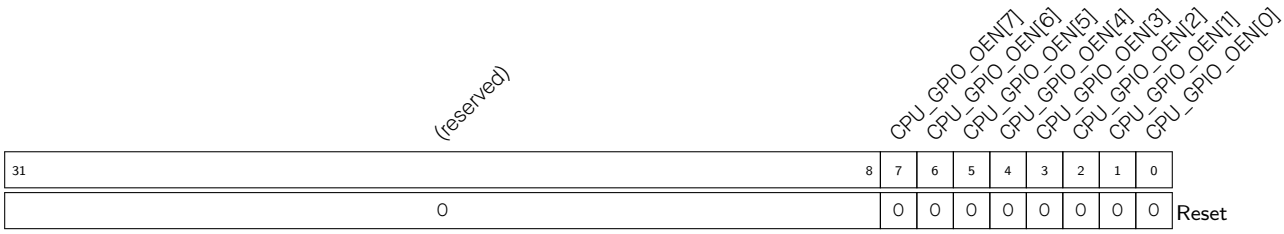
- 0：禁能
- 1：使能

Register 1.14. mpccr (0x7E2)



MPCCR 性能计数器计数的值。(读/写)

Register 1.15. cpu_gpio_oen (0x803)

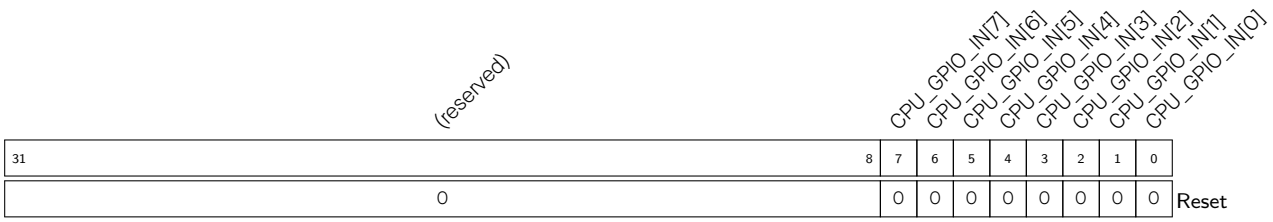


CPU_GPIO_OEN GPIOn (n=0 ~ 21) 输出使能。CPU_GPIO_OEN[7:0] 分别对应章节 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 中表 5-2 里的 cpu_gpio_out_oen[7:0] 输出使能信号。CPU_GPIO_OEN 的值与 cpu_gpio_out_oen 的值对应。

此寄存器是 CPU_GPIO_OUT 的使能寄存器。(读/写)

- 0：GPIO 输出关闭
- 1：GPIO 输出使能

Register 1.16. cpu_gpio_in (0x804)

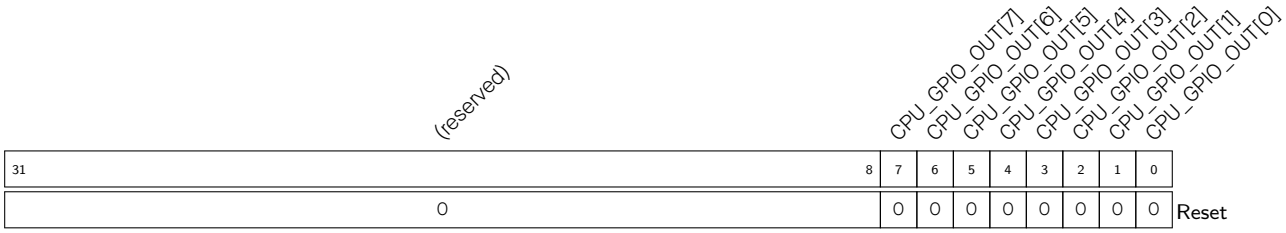


CPU_GPIO_IN 读取 SoC GPIOn (n=0 ~ 21) 的输入值 (1 为高电平，0 为低电平)。

CPU_GPIO_IN[7:0] 分别对应章节 *IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 中表 5-2 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 2。(只读)

Register 1.17. 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 5-2 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 2。（读/写）

1.5 中断控制器

1.5.1 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 31 个具有唯一 ID (1-31) 的异步中断
- 支持通过读写存储器匹配寄存器进行配置
- 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
- 与异常向量地址偏移量匹配的中断 ID

1.5.2 功能描述

每个中断 ID 都有 5 个属性：

1. 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 INTERRUPT_COREO_CPU_INT_ENABLE_REG 相应的域进行配置。
2. 类型 (0-1)：
   - 在中断信号的上升沿使能闩锁状态。
   - 通过写入 INTERRUPT_COREO_CPU_INT_TYPE_REG 相应的域进行配置。
   - 类型保持为 0 的中断称为“电平”类型中断。
   - 类型保持为 1 的中断称为“边沿”类型中断。
3. 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。
   - 通过写入中断 ID *n* (1-31) 的 INTERRUPT_COREO_CPU_INT_PRI_*n*_REG 进行配置。
   - 优先级为零或小于 INTERRUPT_COREO_CPU_INT_THRESH_REG 指定阈值的中断将被屏蔽。
   - 优先级最低为 1，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。
4. 等待状态 (0-1)：
   - 反映已使能且未被屏蔽的中断信号被捕获时的状态。
   - 通过读取 INTERRUPT_COREO_CPU_INT_EIP_STATUS_REG 中的相应位获得每个中断 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 进入异常。
   - 如果在等待的中断抢占 CPU 并导致其跳转到相应的异常向量地址，则称该中断为“已声明”。
   - 所有在等待的中断都为“未声明”。

5. 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 INTERRUPT_COREO_CPU_INT_CLEAR_REG 中的相应位进行切换。
   - 电平类型的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 INTERRUPT_COREO_CPU_INT_ENABLE_REG 中的相应位再置位 INTERRUPT_COREO_CPU_INT_CLEAR_REG 中的相同位。

当 CPU 处理在等待的中断时，会进行以下操作：

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

表 1-3 列出了每个中断 ID 及其对应的异常向量地址。简而言之，中断 $ID = i$ 的字对齐的异常地址 = $(mtvec + 4i)$.

说明：$ID = 0$ 不可用，不能用于捕获中断，这是因为相应的异常向量地址 *(mtvec + 0x00)* 已经预留给异常。

表 1-3. 中断 ID 与异常向量地址

ID	地址	ID	地址	ID	地址	ID	地址
0	NA	8	mtvec + 0x20	16	mtvec + 0x40	24	mtvec + 0x60
1	mtvec + 0x04	9	mtvec + 0x24	17	mtvec + 0x44	25	mtvec + 0x64
2	mtvec + 0x08	10	mtvec + 0x28	18	mtvec + 0x48	26	mtvec + 0x68
3	mtvec + 0x0c	11	mtvec + 0x2c	19	mtvec + 0x4c	27	mtvec + 0x6c
4	mtvec + 0x10	12	mtvec + 0x30	20	mtvec + 0x50	28	mtvec + 0x70
5	mtvec + 0x14	13	mtvec + 0x34	21	mtvec + 0x54	29	mtvec + 0x74
6	mtvec + 0x18	14	mtvec + 0x38	22	mtvec + 0x58	30	mtvec + 0x78
7	mtvec + 0x1c	15	mtvec + 0x3c	23	mtvec + 0x5c	31	mtvec + 0x7c

在跳转到异常向量之后，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 指令后恢复正常程序流。

执行 MRET 指令后，CPU 将进行以下操作：

- 将 MPIE 的状态复制回 MIE，然后清零 MPIE。这意味着，如果之前置位了 MPIE，则执行 MRET 后 MIE 将被置位，进而全局使能中断。
- 跳转到 mepc 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 1.5.3。

中断控制器具有以下行为特点：

- 仅当中断具有非零优先级、大于或等于阈值寄存器中的值时，它才会反映在 INTERRUPT_COREO_CPU_INT_EIP_STATUS_REG 中。

- 如果一个中断反映在 INTERRUPT_COREO_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 INTERRUPT_COREO_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）。

1.5.3 建议操作

1.5.3.1 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断言到被 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。

1.5.3.2 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。

在正常情况下，如果要使能某个中断 *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），置位或取消置位 INTERRUPT_COREO_CPU_INT_TYPE_REG 中的第 *n* 个位
3. 通过写入 INTERRUPT_COREO_CPU_INT_PRI_*n*_REG 指定优先级（最低为 1，最高为 15）
4. 置位 INTERRUPT_COREO_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 相对应的异常向量地址。软件可以通过读取 mcause 来推断异常类型（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)。如果异常向量中的每个表项都是指向不同异常处理程序的跳转指令，则软件无需做此推断。最后，异常处理程序会将程序指引到该中断相应的 ISR。

进入 ISR 后，如果中断为边沿类型，则软件必须切换 INTERRUPT_COREO_CPU_INT_CLEAR_REG 中的第 *n* 个位，如果是电平类型中断，则必须清除相应的中断源。

软件还可以更新 INTERRUPT_COREO_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 返回到异常处理程序之后，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 INTERRUPT_COREO_CPU_INT_EIP_STATUS_REG 检查中断是否在等待
3. 置位/取消置位 INTERRUPT_COREO_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 INTERRUPT_COREO_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。

1.5.4 寄存器列表

本小节的所有地址均为相对于中断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 3 *系统和存储器* 中的表 3-3。

中断寄存器的完整列表及详细描述请见章节 8 *中断矩阵 (INTMTRX)*，8.4 小节中的 “CPU 中断寄存器”。

1.5.5 寄存器

本小节的所有地址均为相对于中断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 3 *系统和存储器* 中的表 3-3。

中断寄存器的完整列表及详细描述请见章节 8 *中断矩阵 (INTMTRX)*，8.4 小节中的 “CPU 中断寄存器”

1.6 调试

1.6.1 概述

本节介绍如何调试和测试在 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 外部调试支持规范版本 0.13 (RISC-V External Debug Support Specification version 0.13)。

图 1-2 为外部调试系统架构图。

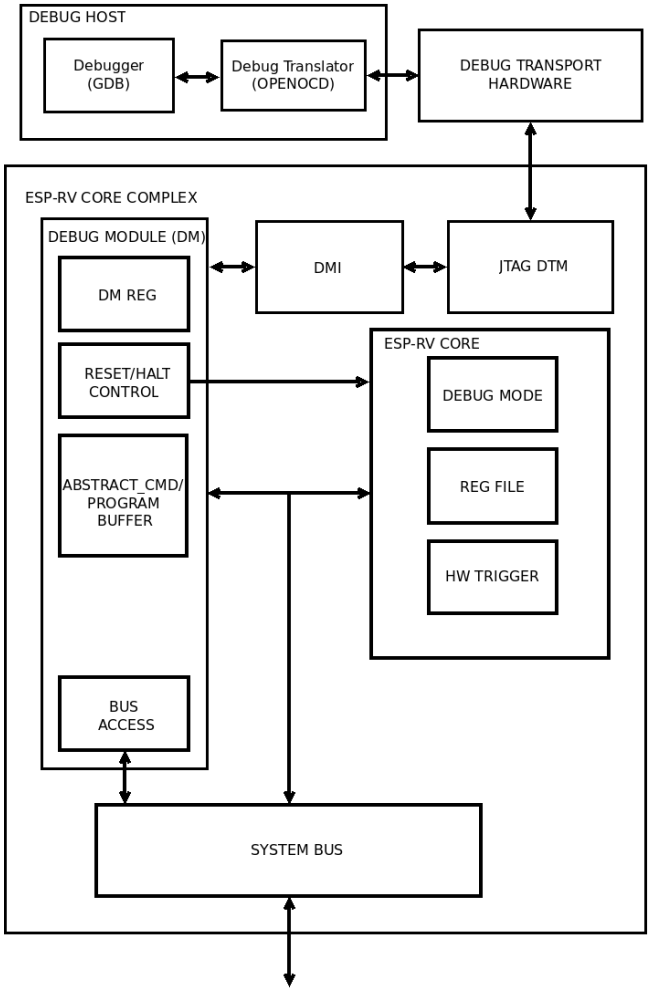


图 1-2. 调试系统架构

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 Olimex USB-JTAG 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-RV 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-RV 内核带有一个支持 2 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。

1.6.2 特性

基础调试功能具有以下特性：

- 支持暂停和恢复 CPU 内核
- 可访存 CSR 和 GPR 寄存器
- 可从复位后执行的第一条指令开始调试
- 可复位 CPU 内核
- 支持软件断点
- 硬件单步调试
- 16 字的程序缓冲区
- 支持系统总线访问
- 支持 2 个硬件触发器

1.6.3 功能描述

调试机制遵守 RISC-V 外部调试支持规范版本 0.13。有关调试功能的详细介绍，请参考 RISC-V 外部调试支持规范。

1.6.4 寄存器列表

下表列出了 ESP-RV 内核支持的调试 CSR。

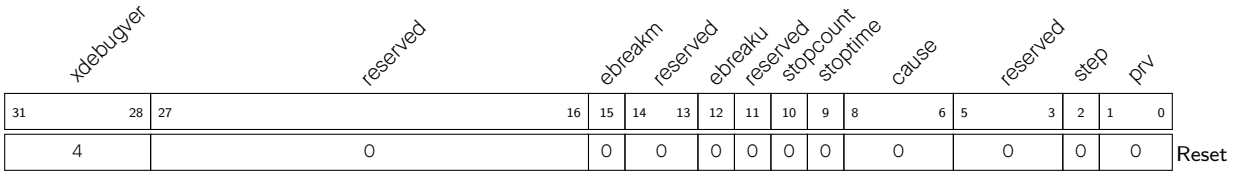
名称	描述	地址	访问
dcsr	调试控制和状态寄存器	0x7B0	读/写
dpc	调试 PC 寄存器	0x7B1	读/写
dscratch0	调试暂存寄存器 0	0x7B2	读/写
dscratch1	调试暂存寄存器 1	0x7B3	读/写

所有调试模块寄存器的实现均符合 RISC-V 外部调试支持规范版本 0.13。请参考 RISC-V 外部调试支持规范获取详细信息。

1.6.5 寄存器

以下是 ESP-RV 内核支持的调试 CSR 的详细描述。

Register 1.18. dcsr (0x7B0)



xdebugver 调试版本。(只读)

- 4：存在外部调试支持

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

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

stopcount 此域没有实现。调试器会始终读出 0。(只读)

stoptime 此性能没有实现。调试器会始终读出 0。(只读)

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。(只读)

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）

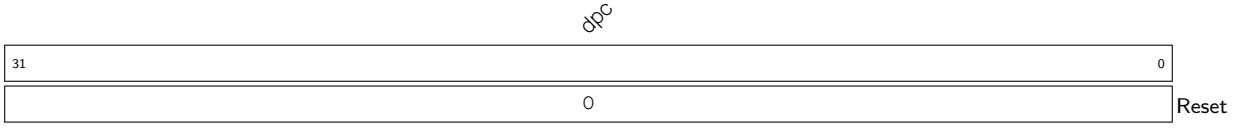
其他值保留供以后使用。

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。(读/写)

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）。

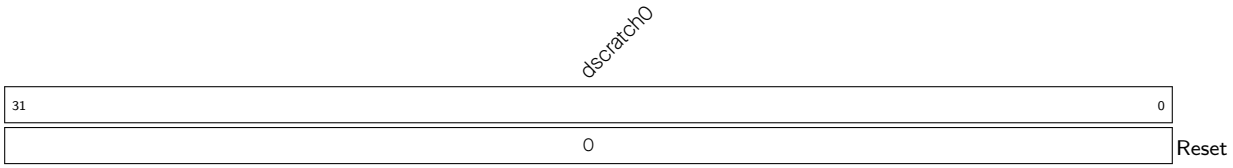
* **注意**：与 RISC-V 调试规格版本 0.13 不同。

Register 1.19. dpc (0x7B1)



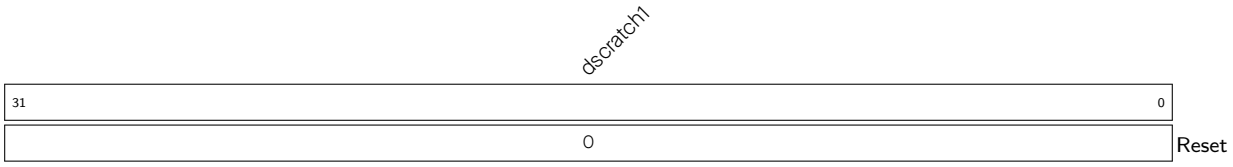
dpc 进入调试模式后，dpc 将写入遇到异常的指令的虚拟地址。恢复执行时，CPU 内核的 PC 将更新为 dpc 保存的虚拟地址。调试器可以写入 dpc 配置 CPU 恢复执行的位置。(读/写)

Register 1.20. dscratch0 (0x7B2)



dscratch0 供调试模块内部使用。(读/写)

Register 1.21. dscratch1 (0x7B3)



dscratch1 供调试模块内部使用。(读/写)

1.7 硬件触发器

1.7.1 特性

硬件触发器模块提供了断点和观察点功能，供调试使用。硬件触发器具有以下特性：

- 2 个独立触发单元
- 匹配程序计数器的地址或存储器访问地址
- 可通过引起断点异常来抢占执行
- 可暂停执行并将控制权转交给调试器
- 支持 NAPOT（2 的幂次方对齐）地址编码

1.7.2 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 (0-7) 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，代表匹配类型触发器，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，虽然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 1-5）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 1-5. 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，当程序在异常处理程序中执行时，该寄存器可用于阻止触发器重复引起异常。默认情况下 ISR 内部的断点异常也被禁用，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。

1.7.3 触发执行流程

当触发器触发引起硬件线程暂停并进入调试模式时 (action = 1)：

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

当触发器触发引起硬件线程进入异常时 (action = 0)：

- mepc 被设置为当前 PC（在解码阶段）
- mcause 被设置为 3，即断点异常
- mpte 被设置为异常发生之前的 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，则硬件线程会暂停并进入调试模式。

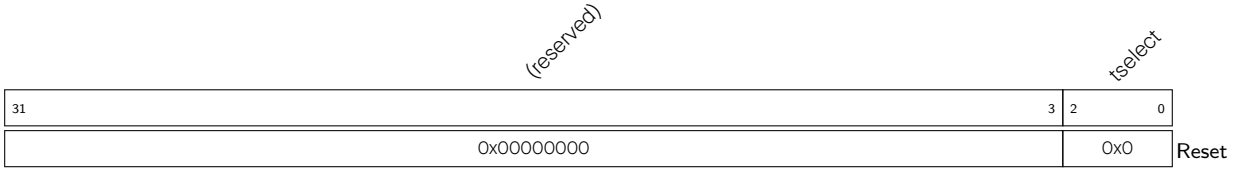
1.7.4 寄存器列表

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	读/写
tdata1	触发器抽象数据寄存器 1	0x7A1	读/写
tdata2	触发器抽象数据寄存器 2	0x7A2	读/写
tcontrol	全局触发器控制寄存器	0x7A5	读/写

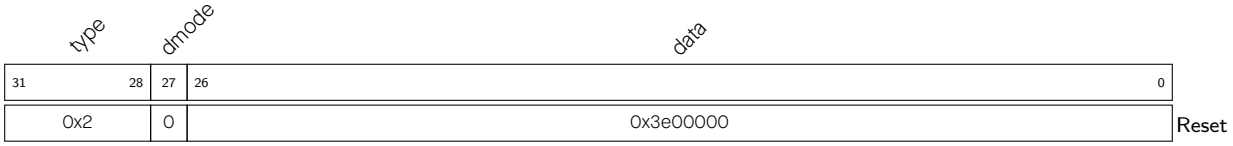
1.7.5 寄存器

Register 1.22. tselect (0x7A0)



tselect 触发器单元编号 (0-7)。(读/写)

Register 1.23. tdata1 (0x7A1)



type 触发器类型。(只读)

仅支持匹配类型 (0x2)，此域保留。

dmode 如果某触发器正在被调试器使用，则此域置为 1.(读/写 *)

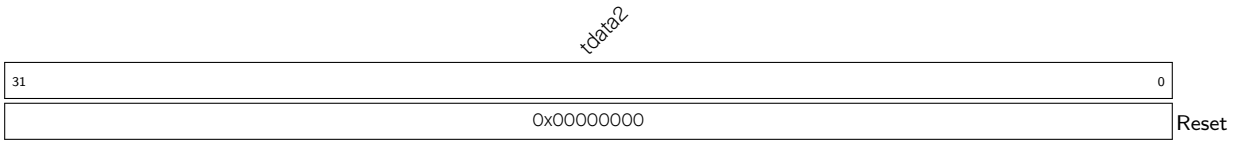
- 0：在调试模式和机器模式下都能写入 tdata1 和 tdata2
- 1：只有在调试模式下才能写入 tdata1 和 tdata2。其他模式下的写操作将被忽略。

* 说明：仅支持调试模式下的写操作。

data 保存抽象 tdata1 的内容。(读/写)

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。

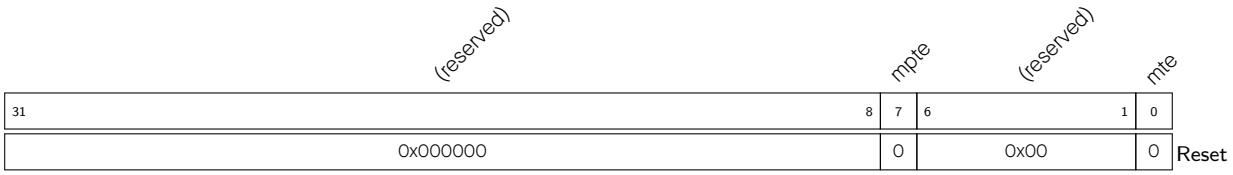
Register 1.24. tdata2 (0x7A2)



tdata2 保存抽象 tdata2 的内容。(读/写)

由于仅支持匹配类型 (0x2) 触发器，此域将始终被解读为 maddress。

Register 1.25. tcontrol (0x7A5)



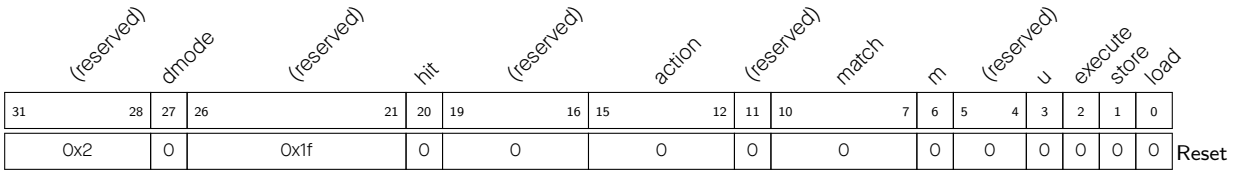
mpte 机器模式下前一个触发器使能域。(读/写)

- 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。

mte 机器模式下触发器使能域。(读/写)

- 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。

Register 1.26. mcontrol (0x7A1)



dmode 与 tdata1 的 dmode 一致。

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

action 配置选定的触发器在触发时进行以下操作。(读/写)
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）

说明：写入无效数值会导致此域变为默认值 *0x0*。

match 配置触发器进行数据/指令地址的下匹配操作。(读/写)
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。

说明：写入超过最大值的数值会被裁剪为最大值 *0x1*。

m 置位使选定的触发器在机器模式下操作。(读/写)

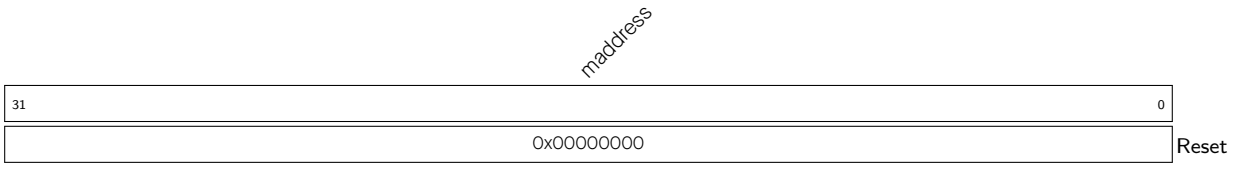
u 置位使选定的触发器在用户模式下操作。(读/写)

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。(读/写)

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。(读/写)

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。(读/写)

Register 1.27. maddress (0x7A2)



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

1.8 存储器保护

1.8.1 概述

CPU 内核包含一个物理存储器保护 (PMP) 单元，可以供软件设置存储器访问特权（读、写、执行权限）。它支持 16 个区域，其中一些区域已经根据 ESP8684 的存储器映射结构进行了硬编码，其余区域可配，可以根据软件代码大小将 SRAM 分成单独的 IRAM/DRAM 区域。

该物理存储器保护机制完全遵循 RISC-V 指令集手册 V1.10 第二卷 “特权架构” 中的规范。不过，为了节省空间，代码中已经对 13 个 pmpaddrX 寄存器进行了硬编码（见寄存器列表），下文会具体说明。

如需了解 RISC-V PMP 的更多信息，请参考 RISC-V 指令集手册 V1.10 第二卷 “特权架构”。

1.8.2 特性

PMP 单元具有以下特性：

- 支持 16 个 PMP 区域
- pmpaddr0-2 寄存器 pmpaddr0-2 可配
- pmpaddr3-15 的值已经根据 ESP8684 的存储器映射结构进行了硬编码

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。PMP CSR 只能在机器模式下进行配置。写入、读取和执行权限检测一旦被使能，则将根据 pmpcfgX 和 pmpaddrX 寄存器中的配置值作用于用户模式下的所有存储器访问。

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。这意味着软件必须通过 pmpcfgX 和 pmpaddr 寄存器设置用户模式下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 检测，则软件可以将所需 PMP 表项的锁定位置位来使能权限检测。锁定位一旦置位，就只能通过 CPU 复位被清零。

如果在没有执行权限的情况下从存储器区域取指，则会在处理器级别生成异常，并且 mcause CSR 中会写入异常原因为指令访问错误。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 中会写入异常原因为读访问错误或写访问错误。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。如前文所述，软件可根据需要配置 pmpaddrX0-2 从而对 SRAM 进行分区。pmpaddrX3-15 已经写入 “CSR 复位值”，这些值是根据芯片存储器结构（“PMP 区域” 一栏）设定的。要启用任一 PMP 区域，相应 pmpcfgX 寄存器的 A 字段都应该配置为 “地址匹配模式” 栏中标明的值。

名称	描述	CSR 地址	CSR 复位值	CSR 访问	地址匹配模式	PMP 区域
pmpcfg0	PMP 配置寄存器	0x3A0	0x0	读/写	-	-
pmpcfg1	PMP 配置寄存器	0x3A1	0x0	读/写	-	-
pmpcfg2	PMP 配置寄存器	0x3A2	0x0	读/写	-	-
pmpcfg3	PMP 配置寄存器	0x3A3	0x0	读/写	-	-
pmpaddr0	PMP 地址寄存器	0x3B0	0x0	读/写	OFF	IRAM 基地址

名称	描述	CSR 地址	CSR 复位值	CSR 访问	地址匹配模式	PMP 区域
pmpaddr1	PMP 地址寄存器	0x3B1	0x0	读/写	TOR	IRAM 结束地址
pmpaddr2	PMP 地址寄存器	0x3B2	0x0	读/写	OFF	DRAM 基地址
pmpaddr3	PMP 地址寄存器	0x3B3	0x0FF38000	只读	TOR	DRAM 结束地址 0x3FCDFFFF
pmpaddr4	PMP 地址寄存器	0x3B4	0x08FFFFFF	只读	NAPOT	0x20000000 - 0x27FFFFFF (128 MB)
pmpaddr5	PMP 地址寄存器	0x3B5	0x0F07FFFF	只读	NAPOT	0x3C000000 - 0x3C3FFFFF (4 MB)
pmpaddr6	PMP 地址寄存器	0x3B6	0x0FFC0000	只读	OFF	0x3FF00000
pmpaddr7	PMP 地址寄存器	0x3B7	0x0FFD4000	只读	TOR	0x3FF00000 - 0x3FF4FFFF (320 KB)
pmpaddr8	PMP 地址寄存器	0x3B8	0x10000000	只读	OFF	0x40000000
pmpaddr9	PMP 地址寄存器	0x3B9	0x10024000	只读	TOR	0x40000000 - 0x4008FFFF (576 KB)
pmpaddr10	PMP 地址寄存器	0x3BA	0x1087FFFF	只读	NAPOT	0x42000000 - 0x423FFFFF (4 MB)
pmpaddr11	PMP 地址寄存器	0x3BB	0x1801FFFF	只读	NAPOT	0x60000000 - 0x600FFFFF (1 MB)
pmpaddr12	PMP 地址寄存器	0x3BC	0x100DF7FF	只读	NAPOT	0x4037C000 - 0x4037FFFF (16 KB)
pmpaddr13	PMP 地址寄存器	0x3BD	0x3FFFFFFF	只读	NA4	0xFFFFFFFF (4 Byte)
pmpaddr14	PMP 地址寄存器	0x3BE	0x0	只读	OFF	0x0
pmpaddr15	PMP 地址寄存器	0x3BF	0x3FFFFFFF	只读	TOR	0xFFFFFFFE (4 GB)

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”中定义的全部 `pmpcfg0-3` 和 `pmpaddr0-15` CSR。

2 通用 DMA 控制器 (GDMA)

2.1 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。

ESP8684 GDMA 共有 2 个独立的通道，其中包括 1 个发送通道（即发送通道 0）和 1 个接收通道（即接收通道 0）。这 2 个通道被支持 GDMA 功能的外设所共享，用户可以将通道分配给任何支持 DMA 功能的外设。这些外设包括：SPI2 和 SHA。

GDMA 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。

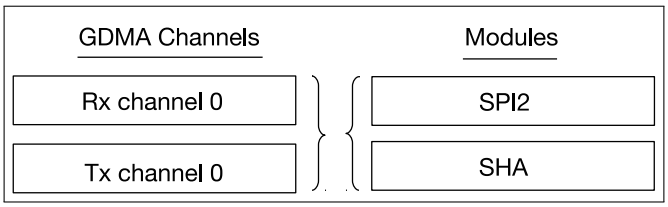


图 2-1. 具有 GDMA 功能的模块和 GDMA 通道

2.2 特性

GDMA 控制器具有以下几个特点：

- 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 256 KB
- 包含 1 个 TX、1 个 RX 通道
- 任一通道支持可配置的外设选择
- 通道间固定优先级及轮询仲裁
- AHB 总线架构

2.3 架构

ESP8684 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 2-2 为 GDMA 引擎基本架构图。

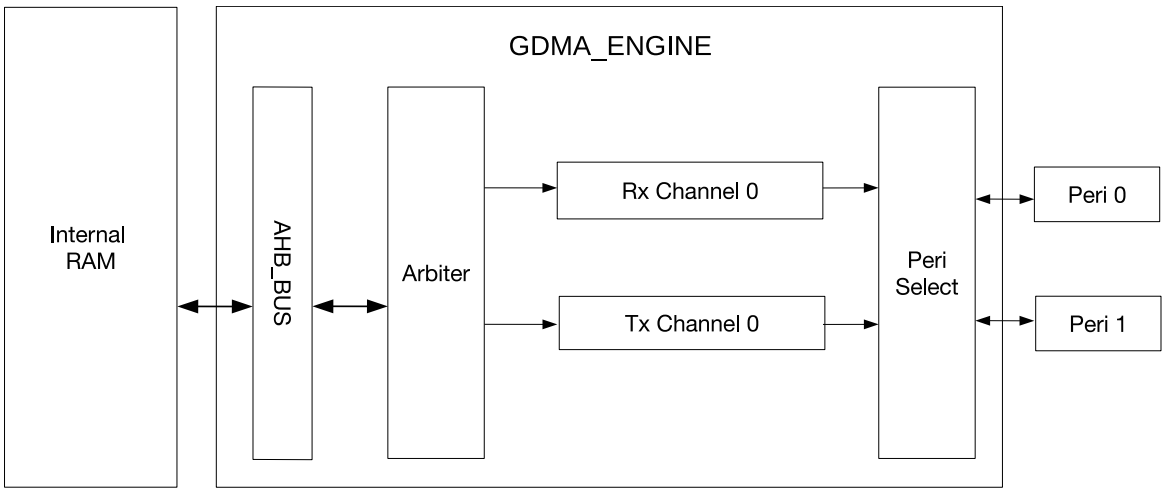


图 2-2. GDMA 引擎的架构

GDMA 引擎共有 2 个独立的通道，其中包括 1 个发送通道和 1 个接收通道。每个通道可选择与不同的外设相连，从而实现通道资源被外设共享。

GDMA 引擎通过 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 3 系统和存储器。

软件可以通过挂载链表的方式来使用 GDMA 引擎。链表本身须存储在片内 RAM 中，包括 outlink（发送链表）与 inlink（接收链表）。GDMA 从片内 RAM 中取得链表，然后根据 outlink 中的内容将相应 RAM 中的数据发送出去，或者根据 inlink 中的内容将接收的数据存入指定 RAM 地址空间。

2.4 功能描述

2.4.1 外设和存储间的数据传输

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 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 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2-1 所示为配置寄存器与其对应外设的关系。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 2-1. 配置寄存器与外设选择关系表

GDMA_PERI_IN_SEL_CHO GDMA_PERI_OUT_SEL_CHO	外设
0	SPI2
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	SHA

8	Reserved

2.4.2 存储到存储的数据传输

GDMA 支持存储到存储的数据传输。置位 GDMA_MEM_TRANS_EN_CHO，发送通道 0 的输出将与接收通道 0 的输入相连，从而使能存储到存储的数据传输功能。

2.4.3 链表

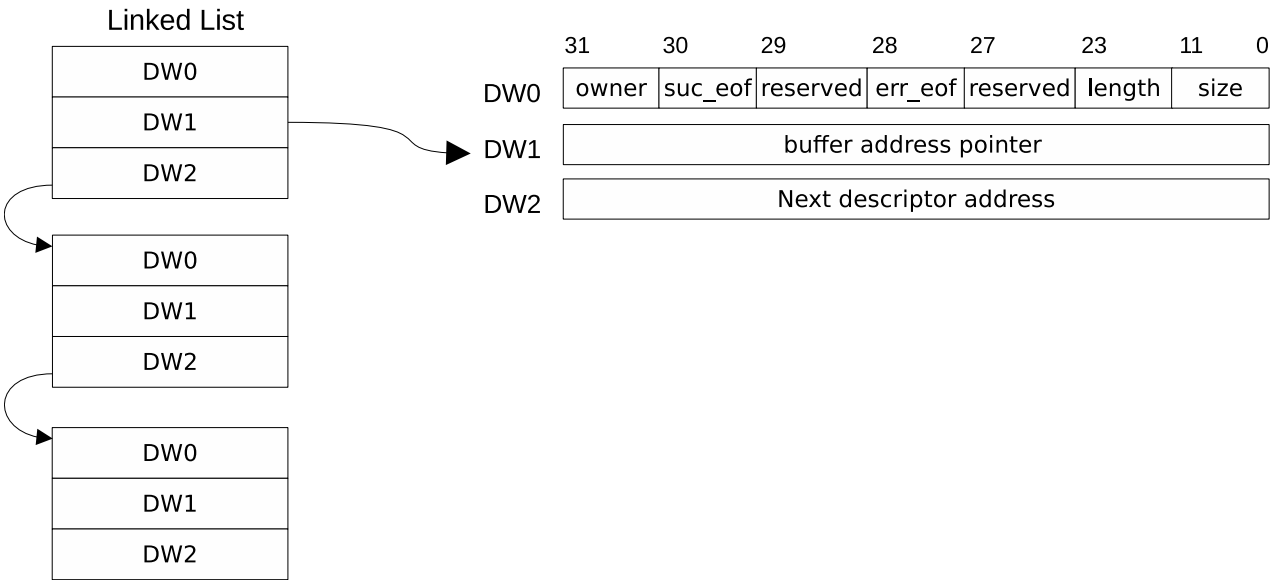


图 2-3. 链表结构图

图 2-3 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 引擎使用。描述符每一字段的意义如下：

- owner (DW0) [31]：表示当前描述符对应的 buffer 允许的操作者。
  1'b0：允许的操作者为 CPU；
  1'b1：允许的操作者为 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_CHO 置 1，硬件才会自动将该位清零。软件在挂载链表前需要将该位置 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]：表示一个描述符对应的数据成功传输后是否触发 GDMA_IN_SUC_EOF_CH*n*_INT 或 GDMA_OUT_EOF_CH*n*_INT 中断。
  1'b0：当前描述符成功传输后不会触发中断；
  1'b1：当前描述符成功传输后触发中断。
  对于接收描述符，需要软件将该位写 0，硬件会在接收到包含 EOF 标志的数据后将该位置 1。
  对于发送描述符，需要软件按需要将描述符中的该位置 1。如果软件将在某个描述符中将该位配置为 1，则 GDMA 在处理完该描述符时，会在发送给外设的数据中加入 EOF 标志，告知外设该段数据是一个阶段性结束。

- Reserved (DW0) [29]：保留。此位为无关项。
- err_eof (DW0) [28]：表示接收结束错误标志。
  对于接收描述符，外设在收完描述符对应的数据段并检测到接收数据错误会将该位置 1。
- Reserved (DW0) [27:24]：保留。
- length (DW0) [23:12]：表示当前描述符对应的 buffer 中的有效字节数。对于发送描述符，该段由软件填写，表示从 buffer 中读取数据时需要读取的字节数；对于接收描述符，该段由硬件使用完该 buffer 后或者接收到最后一个数据时自动填写，表示 buffer 中存储的有效字节数。
- size (DW0) [11:0]：表示当前描述符对应的 buffer 容量的字节数，size 需要大于或者等于 length。
- buffer address pointer (DW1)：buffer 的地址。
- next descriptor address (DW2)：下一个描述符的地址。该地址必须指向片内 RAM 的地址空间。如果当前描述符为链表中最后一个描述符时，该值填 0。

用 GDMA 接收数据时，如果数据帧或者包结束时，当前描述符的 suc_eof 位将被置 1，GDMA 停止继续向该描述符指定的 buffer 接收数据。即使此时已接收数据的长度小于当前描述符指定的 buffer 长度，后续接收的数据也不会继续占用该 buffer 的剩余空间，而会进入下一个描述符指定的 buffer 。

2.4.4 启动 DMA

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_CH0 字段指向第一个接收链表描述符，然后置位 GDMA_INLINK_START_CH0 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_CH0 字段指向第一个发送链表描述符，然后置位 GDMA_OUTLINK_START_CH0 位启动 GDMA。

GDMA_INLINK_START_CH0 与 GDMA_OUTLINK_START_CH0 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，原本看似只需将链表最末尾描述符的 next descriptor address (DW2) 字段配置为新链表第一个描述符对应的地址。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 引擎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 引擎会确保顾及到新追加的描述符；如果数据传输已经结束，GDMA 引擎会重启数据传输，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_CH0 或者 GDMA_OUTLINK_RESTART_CH0（这两个位由硬件自动清零），如图 2-4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。

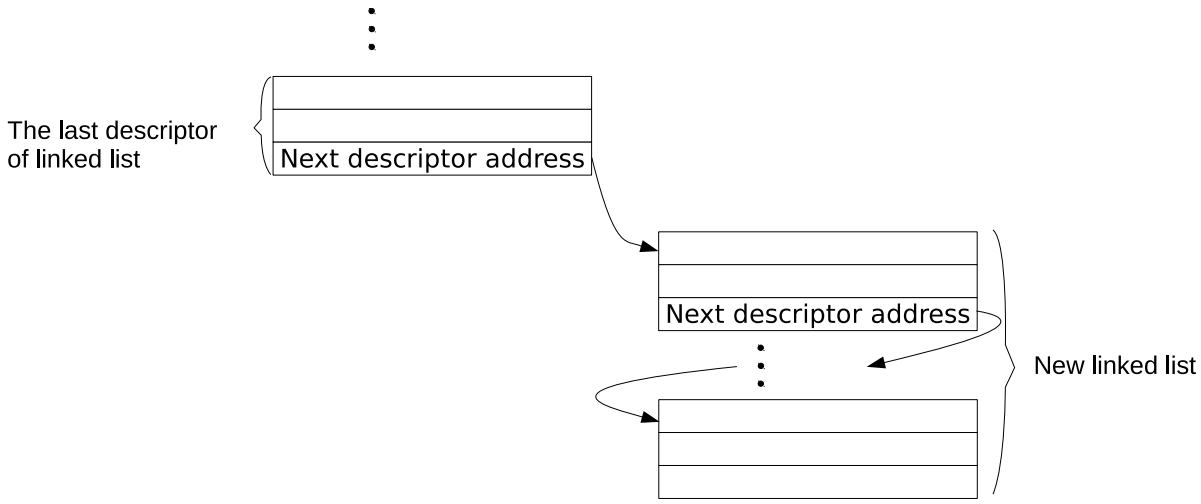


图 2-4. 链表关系图

2.4.5 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_CH0_INT 或者 GDMA_OUT_DSCR_ERR_CH0_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_CH0 或者 GDMA_OUT_CHECK_OWNER_CH0 置 1 时，检查描述符的 owner 位。如果该位为 0，表示当前操作者应为 CPU，检查失败。将GDMA_IN_CHECK_OWNER_CH0 或者 GDMA_OUT_CHECK_OWNER_CH0 置 0 可以跳过检查；
- 检查描述符中第二个字指示的地址是否在 0x3FCA0000 ~ 0x3FCDFFFF 范围内（请参见本节 2.4.7）。如果不在该范围内，则检查失败。

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，重新配置该 DMA 通道并启动。具体流程详见章节 2.6.2、章节 2.6.3 和章节 2.6.4。

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。

2.4.6 数据传输结束标志

GDMA 通过 EOF 来指示对应描述符所需传输的数据段数据传输结束。

发送数据时，置位 GDMA_OUT_EOF_CH0_INT_ENA 位使能 GDMA_OUT_EOF_CH0_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_CH0_INT_ENA 位使能 GDMA_IN_SUC_EOF_CH0_INT 中断，表示带有 EOF 标志的数据段数据接收完成。GDMA 还支持 GDMA_IN_ERR_CH0_EOF_INT 中断，置位 GDMA_IN_ERR_EOF_CH0_INT_ENA 使能该中断，表示对应描述符所需传输的数据段数据接收完成，但该帧或包接收数据有错误。

软件在检测到 GDMA_OUT_TOTAL_EOF_CH0_INT 或 GDMA_IN_SUC_EOF_CH0_INT 中断时，可以记录 GDMA_OUT_EOF_DES_ADDR_CH0 或 GDMA_IN_SUC_EOF_DES_ADDR_CH0 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。

2.4.7 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0x3FCA0000 ~ 0x3FCDFFFF。为加速数据传输速率，支持突发传输模式。置位 GDMA_IN_DATA_BURST_EN_CH0 使能 RX 通道突发传输模式；置位 GDMA_OUT_DATA_BURST_EN_CH0 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 2-2. 链表描述符参数对齐要求

链表	突发传输	size	length	buffer address pointer
接收链表	0	—[1]	—	—
	1	字对齐	—	字对齐
发送链表	0	—	—	—
	1	—	—	—

[1] "—" 表示无对齐要求。

如表2-2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。

2.4.8 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 10（0 ~ 9）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。

需要注意的是，所有外设总的吞吐率之和不能超过 GDMA 能支持的最大有效带宽，否则低优先级的外设请求可能无法获得及时响应。

2.5 GDMA 中断

- GDMA_IN_DSCR_EMPTY_CH0_INT：对于接收通道 0，当接收链表描述符指向的 buffer 大小小于待接收数据长度时触发此中断。
- GDMA_IN_DSCR_ERR_CH0_INT：对于接收通道 0，当接收链表描述符里有错误时触发此中断。
- GDMA_IN_ERR_EOF_CH0_INT：对于接收通道 0，当接收的描述符对应数据段中有错误发生时触发此中断。
- GDMA_IN_SUC_EOF_CH0_INT：对于接收通道 0，当一个接收链表描述符对应的数据接收完成，并且描述符的 suc_eof 为 1 时触发此中断。
- GDMA_IN_DONE_CH0_INT：对于接收通道 0，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。
- GDMA_OUT_TOTAL_EOF_CH0_INT：对于发送通道 0，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_CH0_INT：对于发送通道 0，当发送链表描述符里有错误时触发此中断。
- GDMA_OUT_EOF_CH0_INT：对于发送通道 0，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_CH0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_CH0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_CH0_INT：对于发送通道 0，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。

2.6 编程流程

2.6.1 GDMA 时钟与复位配置流程

GDMA 的时钟与复位配置流程如下：

1. 置位系统寄存器 SYSTEM_DMA_CLK_EN，使能 GDMA 模块时钟；
2. 对系统寄存器 SYSTEM_DMA_RST 置 0，释放 GDMA 的复位信号。

2.6.2 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_CH0 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_CH0 指向第一个发送链表描述符；
3. 配置 GDMA_PERI_OUT_SEL_CH0 为对应的外设号，见表2-1；
4. 置位 GDMA_OUTLINK_START_CH0 启动 GDMA TX 通道发送数据；
5. 配置对应的外设（SPI2 或 SHA），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_TOTAL_EOF_CH0_INT 中断，即数据传输完成。

2.6.3 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_CH0 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_CH0 指向第一个接收链表描述符；
3. 配置 GDMA_PERI_IN_SEL_CH0 为对应的外设号，见表2-1；
4. 置位 GDMA_INLINK_START_CH0 启动 GDMA RX 通道发送数据；
5. 配置对应的外设（SPI2），并启动该外设，具体配置请参考对应的外设章节；

2.6.4 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_CH0 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_CH0 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_CH0 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_CH0 指向第一个接收链表描述符；
5. 置位 GDMA_MEM_TRANS_EN_CH0 使能 memory-to-memory 传输功能；
6. 置位 GDMA_OUTLINK_START_CH0 启动 GDMA TX 通道发送数据；
7. 置位 GDMA_INLINK_START_CH0 启动 GDMA RX 通道发送数据；

8. 如果某个发送链表描述符的 suc_eof 位配置为 1，则该描述符对应的数据段传输结束后会触发 GDMA_IN_SUC_EOF_CHO_INT 中断。

2.7 寄存器列表

本小节的所有地址均为相对于 GDMA 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 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

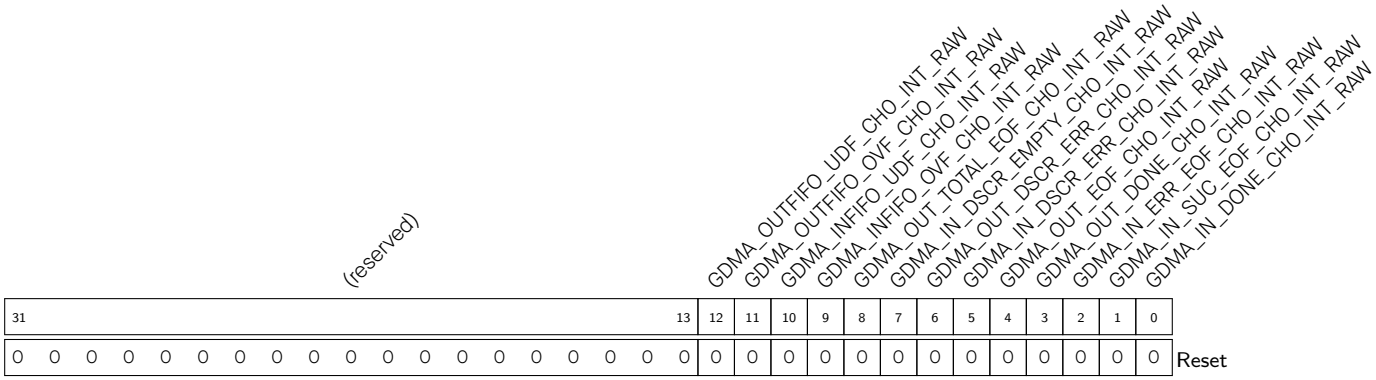
名称	描述	地址	访问
中段寄存器			
GDMA_INT_RAW_CH0_REG	接收通道 0 的原始中断状态	0x0000	R/WTC/SS
GDMA_INT_ST_CH0_REG	接收通道 0 的屏蔽中断	0x0004	RO
GDMA_INT_ENA_CH0_REG	接收通道 0 的中断使能位	0x0008	R/W
GDMA_INT_CLR_CH0_REG	接收通道 0 的中断清除位	0x000C	WT
配置寄存器			
GDMA_MISC_CONF_REG	杂项控制寄存器	0x0044	R/W
GDMA_IN_CONF0_CH0_REG	接收通道 0 的配置寄存器 0	0x0070	R/W
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 的数据弹出控制寄存器	0x007C	varies
GDMA_IN_LINK_CH0_REG	接收通道 0 的链表配置和控制寄存器	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	发送通道 0 的数据推送控制寄存器	0x00DC	varies
GDMA_OUT_LINK_CH0_REG	发送通道 0 的链表配置和控制寄存器	0x00E0	varies
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 传输完成时的接收链表描述符地址	0x0088	RO
GDMA_IN_ERR_EOF_DES_ADDR_CH0_REG	接收通道 0 发生错误时的接收链表描述符地址	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
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 传输完成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 传输完成时的最后一个发送链表描述符地址	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

名称	描述	地址	访问
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
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
外设选择寄存器			
GDMA_IN_PERI_SEL_CH0_REG	接收通道 0 的外设选择	0x00A0	R/W
GDMA_OUT_PERI_SEL_CH0_REG	发送通道 0 的外设选择	0x0100	R/W
版本寄存器			
GDMA_DATE_REG	版本控制寄存器	0x0048	R/W

2.8 寄存器

本小节的所有地址均为相对于 GDMA 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 3 系统和存储器 中的表 3-3。

Register 2.1. GDMA_INT_RAW_CH0_REG (0x0000)



GDMA_IN_DONE_CH0_INT_RAW 接收通道 0 接收到接收链表描述符指向的最后一个字节数据时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_IN_SUC_EOF_CH0_INT_RAW 接收通道 0 接收到接收链表描述符指向的最后一个字节数据且描述符的 suc_eof 位为 1 时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_IN_ERR_EOF_CH0_INT_RAW 保留。(R/WTC/SS)

GDMA_OUT_DONE_CH0_INT_RAW 发送通道 0 将发送链表描述符指向的最后一个字节数据发送给外设时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_OUT_EOF_CH0_INT_RAW 发送通道 0 从存储器读取了发送链表描述符指向的最后一个字节数据时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_IN_DSCR_ERR_CH0_INT_RAW 接收通道 0 检测到接收链表描述符错误时，包括 owner 位错误、接收链表描述符的第二个字和第三个字错误，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_OUT_DSCR_ERR_CH0_INT_RAW 发送通道 0 检测到发送链表描述符错误时，包括 owner 位错误、发送链表描述符的第二个字和第三个字错误，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_IN_DSCR_EMPTY_CH0_INT_RAW 接收通道 0 接收链表指向的 RX FIFO 已满，数据接收未完成，但没有更多接收链表时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_OUT_TOTAL_EOF_CH0_INT_RAW 发送通道 0 发送了发送链表（包括一个或几个发送链表描述符）对应的数据时，该原始中断位翻转至高电平。(R/WTC/SS)

见下页...

Register 2.1. GDMA_INT_RAW_CH0_REG (0x0000)

接上页...

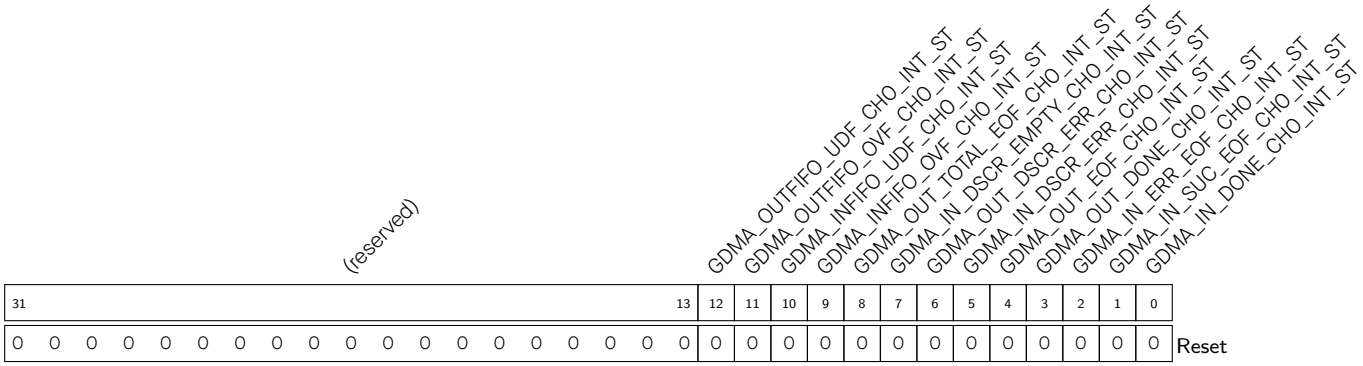
GDMA_INFIFO_OVF_CH0_INT_RAW 接收通道 0 的 L1 FIFO 上溢时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_INFIFO_UDF_CH0_INT_RAW 接收通道 0 的 L1 FIFO 下溢时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_OUTFIFO_OVF_CH0_INT_RAW 发送通道 0 的 L1 FIFO 上溢时，该原始中断位翻转至高电平。(R/WTC/SS)

GDMA_OUTFIFO_UDF_CH0_INT_RAW 发送通道 0 的 L1 FIFO 下溢时，该原始中断位翻转至高电平。(R/WTC/SS)

Register 2.2. GDMA_INT_ST_CH0_REG (0x0004)



GDMA_IN_DONE_CH0_INT_ST GDMA_IN_DONE_CH_INT 中断的原始状态位。(RO)

GDMA_IN_SUC_EOF_CH0_INT_ST GDMA_IN_SUC_EOF_CH_INT 中断的原始状态位。(RO)

GDMA_IN_ERR_EOF_CH0_INT_ST GDMA_IN_ERR_EOF_CH_INT 中断的原始状态位。(RO)

GDMA_OUT_DONE_CH0_INT_ST GDMA_OUT_DONE_CH_INT 中断的原始状态位。(RO)

GDMA_OUT_EOF_CH0_INT_ST GDMA_OUT_EOF_CH_INT 中断的原始状态位。(RO)

GDMA_IN_DSCR_ERR_CH0_INT_ST GDMA_IN_DSCR_ERR_CH_INT 中断的原始状态位。(RO)

GDMA_OUT_DSCR_ERR_CH0_INT_ST GDMA_OUT_DSCR_ERR_CH_INT 中断的原始状态位。(RO)

GDMA_IN_DSCR_EMPTY_CH0_INT_ST GDMA_IN_DSCR_EMPTY_CH_INT 中断的原始状态位。(RO)

GDMA_OUT_TOTAL_EOF_CH0_INT_ST GDMA_OUT_TOTAL_EOF_CH_INT 中断的原始状态位。(RO)

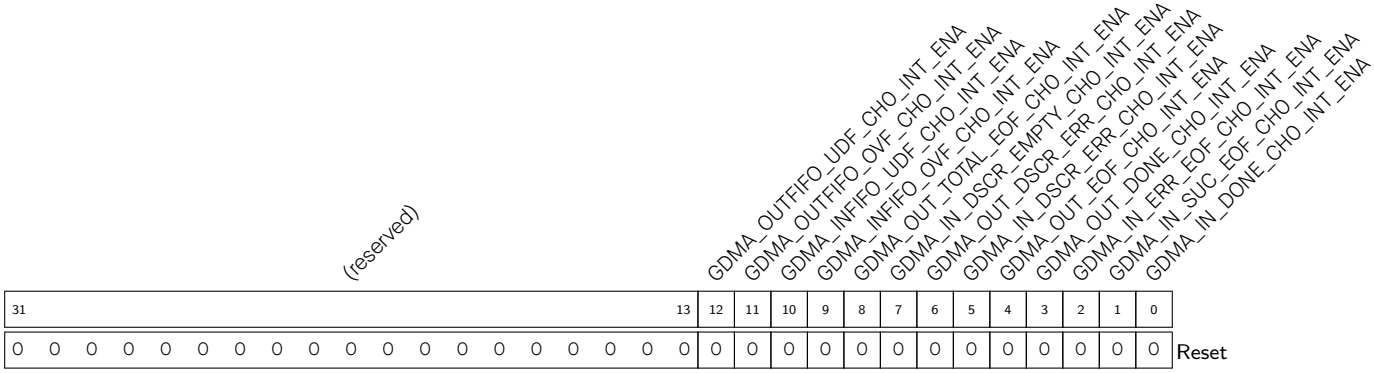
GDMA_INFIFO_OVF_CH0_INT_ST GDMA_ INFIFO_OVF_L1_CH_INT 中断的原始状态位。(RO)

GDMA_INFIFO_UDF_CH0_INT_ST GDMA_ INFIFO_UDF_L1_CH_INT 中断的原始状态位。(RO)

GDMA_OUTFIFO_OVF_CH0_INT_ST GDMA_OUTFIFO_OVF_L1_CH_INT 中断的原始状态位。(RO)

GDMA_OUTFIFO_UDF_CH0_INT_ST GDMA_OUTFIFO_UDF_L1_CH_INT 中断的原始状态位。(RO)

Register 2.3. GDMA_INT_ENA_CH0_REG (0x0008)



GDMA_IN_DONE_CH0_INT_ENA GDMA_IN_DONE_CH_INT 中断的使能位。(R/W)

GDMA_IN_SUC_EOF_CH0_INT_ENA GDMA_IN_SUC_EOF_CH_INT 中断的使能位。(R/W)

GDMA_IN_ERR_EOF_CH0_INT_ENA GDMA_IN_ERR_EOF_CH_INT 中断的使能位。(R/W)

GDMA_OUT_DONE_CH0_INT_ENA GDMA_OUT_DONE_CH_INT 中断的使能位。(R/W)

GDMA_OUT_EOF_CH0_INT_ENA GDMA_OUT_DONE_CH_INT 中断的使能位。(R/W)

GDMA_IN_DSCR_ERR_CH0_INT_ENA GDMA_IN_DSCR_ERR_CH_INT 中断的使能位。(R/W)

GDMA_OUT_DSCR_ERR_CH0_INT_ENA GDMA_OUT_DSCR_ERR_CH_INT 中断的使能位。(R/W)

GDMA_IN_DSCR_EMPTY_CH0_INT_ENA GDMA_IN_DSCR_EMPTY_CH_INT 中断的使能位。(R/W)

GDMA_OUT_TOTAL_EOF_CH0_INT_ENA GDMA_OUT_TOTAL_EOF_CH_INT 中断的使能位。(R/W)

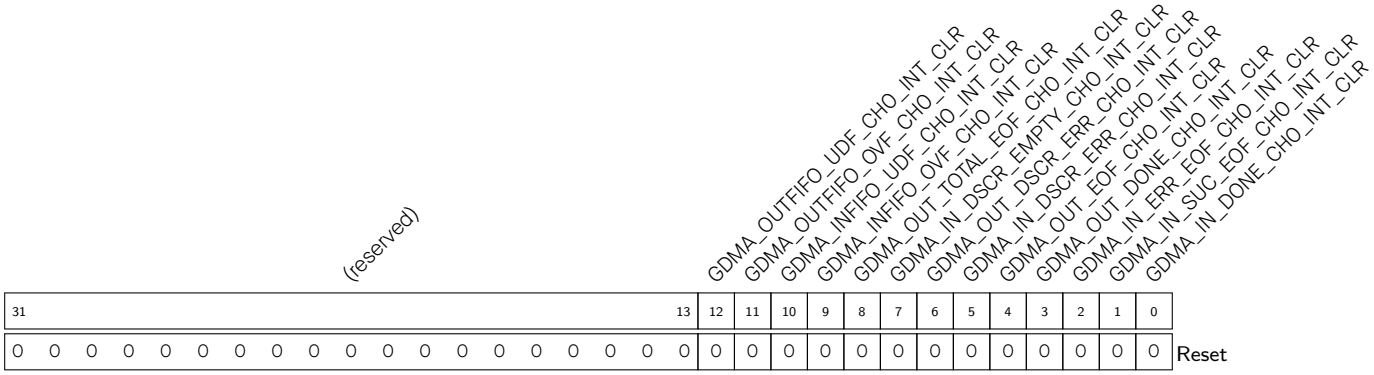
GDMA_INFIFO_OVF_CH0_INT_ENA GDMA_ INFIFO_OVF_L1_CH_INT 中断的使能位。(R/W)

GDMA_INFIFO_UDF_CH0_INT_ENA GDMA_ INFIFO_UDF_L1_CH_INT 中断的使能位。(R/W)

GDMA_OUTFIFO_OVF_CH0_INT_ENA GDMA_OUTFIFO_OVF_L1_CH_INT 中断的使能位。(R/W)

GDMA_OUTFIFO_UDF_CH0_INT_ENA GDMA_OUTFIFO_UDF_L1_CH_INT 中断的使能位。(R/W)

Register 2.4. GDMA_INT_CLR_CH0_REG (0x000C)



GDMA_IN_DONE_CH0_INT_CLR 置位此位，清除 GDMA_IN_DONE_CH_INT 中断。(WT)

GDMA_IN_SUC_EOF_CH0_INT_CLR 置位此位，清除 GDMA_IN_SUC_EOF_CH_INT 中断。(WT)

GDMA_IN_ERR_EOF_CH0_INT_CLR 置位此位，清除 GDMA_IN_ERR_EOF_CH_INT 中断。(WT)

GDMA_OUT_DONE_CH0_INT_CLR 置位此位，清除 GDMA_OUT_DONE_CH_INT 中断。(WT)

GDMA_OUT_EOF_CH0_INT_CLR 置位此位，清除 GDMA_OUT_EOF_CH_INT 中断。(WT)

GDMA_IN_DSCR_ERR_CH0_INT_CLR 置位此位，清除 GDMA_IN_DSCR_ERR_CH_INT 中断。(WT)

GDMA_OUT_DSCR_ERR_CH0_INT_CLR 置位此位，清除 GDMA_OUT_DSCR_ERR_CH_INT 中断。(WT)

GDMA_IN_DSCR_EMPTY_CH0_INT_CLR 置位此位，清除 GDMA_IN_DSCR_EMPTY_CH_INT 中断。(WT)

GDMA_OUT_TOTAL_EOF_CH0_INT_CLR 置位此位，清除 GDMA_OUT_TOTAL_EOF_CH_INT 中断。(WT)

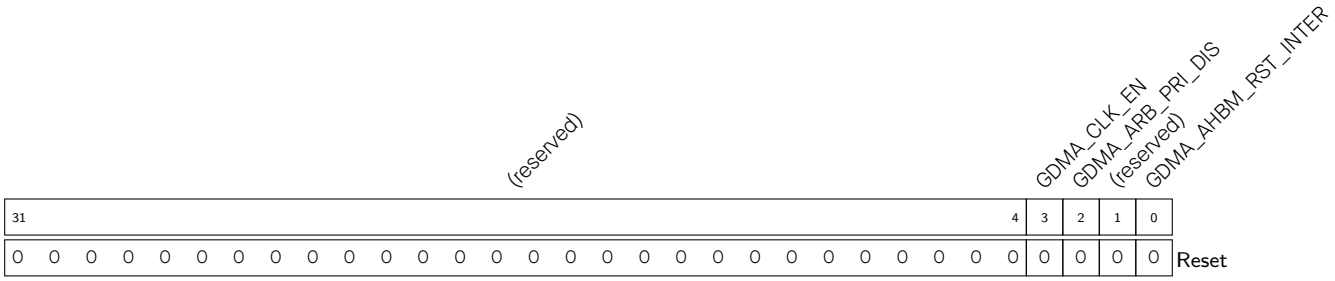
GDMA_INFIFO_OVF_CH0_INT_CLR 置位此位，清除 GDMA_ INFIFO_OVF_L1_CH_INT 中断。(WT)

GDMA_INFIFO_UDF_CH0_INT_CLR 置位此位，清除 GDMA_ INFIFO_UDF_L1_CH_INT 中断。(WT)

GDMA_OUTFIFO_OVF_CH0_INT_CLR 置位此位，清除 GDMA_OUTFIFO_OVF_L1_CH_INT 中断。(WT)

GDMA_OUTFIFO_UDF_CH0_INT_CLR 置位此位，清除 GDMA_OUTFIFO_UDF_L1_CH_INT 中断。(WT)

Register 2.5. GDMA_MISC_CONF_REG (0x0044)

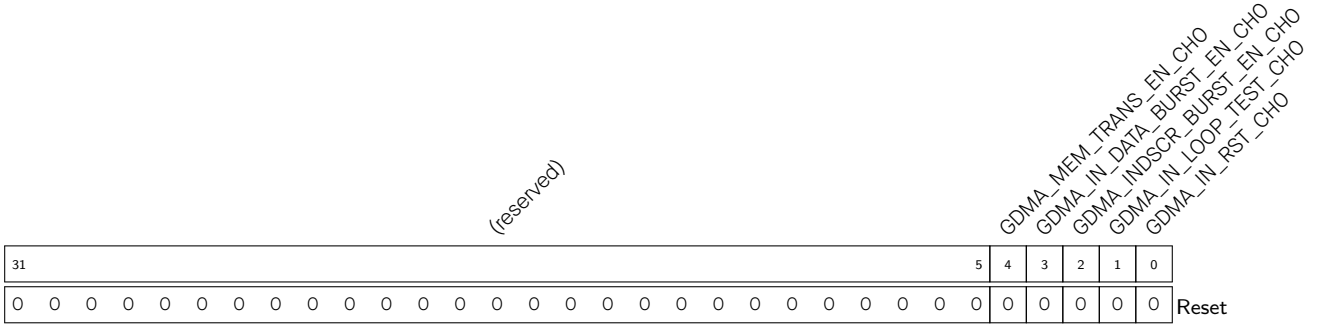


GDMA_AHBM_RST_INTER 置位此位，然后清零此位，重置内部 AHB 状态机。(R/W)

GDMA_ARB_PRI_DIS 置位此位，关闭优先级仲裁功能。(R/W)

GDMA_CLK_EN 0：仅在软件写寄存器时开启时钟。1：强制开启寄存器时钟。(R/W)

Register 2.6. GDMA_IN_CONF0_CH0_REG (0x0070)



GDMA_IN_RST_CH0 用于复位 GDMA 通道 0 的 RX 状态机和 RX FIFO 指针。(R/W)

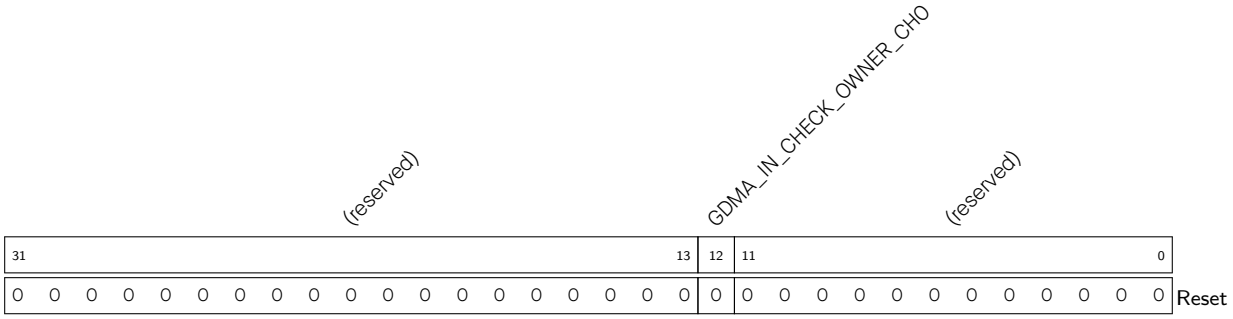
GDMA_IN_LOOP_TEST_CH0 保留。(R/W)

GDMA_INDSCR_BURST_EN_CH0 将此位置 1，在接收通道 0 访问内部 RAM 读取接收链表描述符时使能 INCR 突发传输。(R/W)

GDMA_IN_DATA_BURST_EN_CH0 将此位置 1，在接收通道 0 访问内部 RAM 接收数据时使能 INCR 突发传输。(R/W)

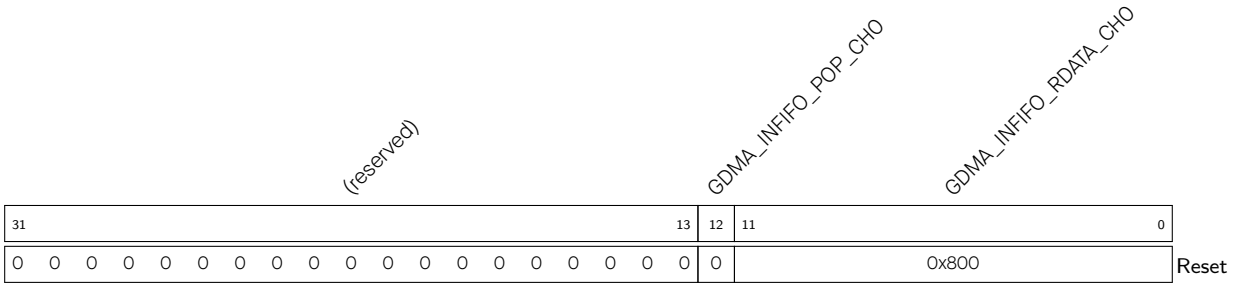
GDMA_MEM_TRANS_EN_CH0 将此位置 1，使能 GDMA 存储器到存储器自动传输。(R/W)

Register 2.7. GDMA_IN_CONF1_CH0_REG (0x0074)



GDMA_IN_CHECK_OWNER_CH0 置位此位，使能链表描述符 owner 位检查。(R/W)

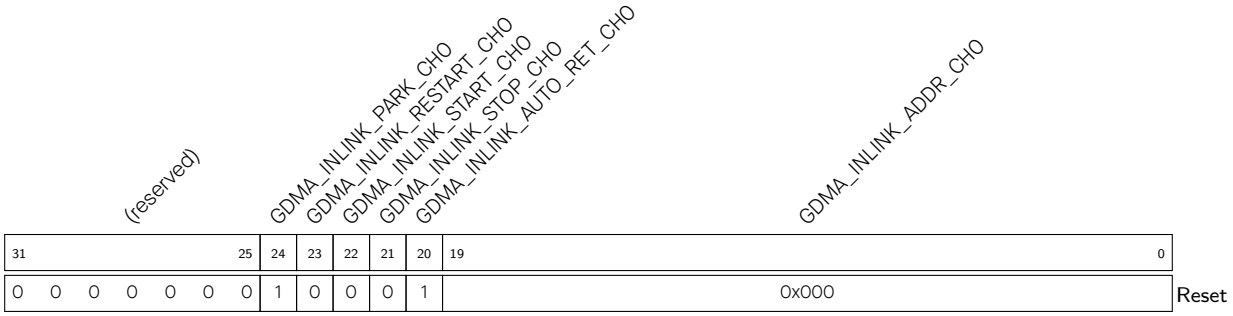
Register 2.8. GDMA_IN_POP_CH0_REG (0x007C)



GDMA_INFIFO_RDATA_CH0 存储从 GDMA FIFO 中弹出的数据（用于排错）。(RO)

GDMA_INFIFO_POP_CH0 置位此位，从 GDMA FIFO 中弹出数据（用于排错）。(R/W/SC)

Register 2.9. GDMA_IN_LINK_CH0_REG (0x0080)



GDMA_INLINK_ADDR_CH0 存储第一个接收链表描述符地址的低 20 位。(R/W)

GDMA_INLINK_AUTO_RET_CH0 当前接收数据有错误时，置位此位返回当前接收链表描述符的地址。(R/W)

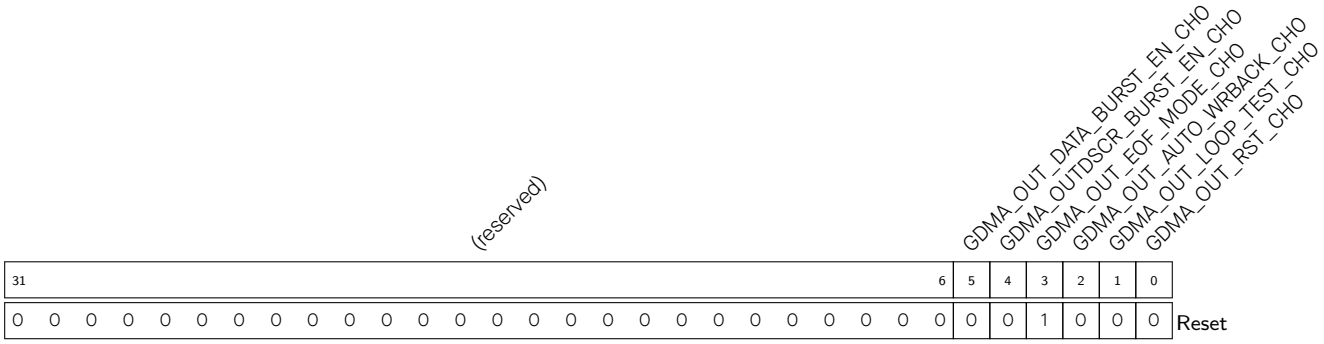
GDMA_INLINK_STOP_CH0 置位此位，停止处理接收链表描述符。(R/W/SC)

GDMA_INLINK_START_CH0 置位此位，开始处理接收链表描述符。(R/W/SC)

GDMA_INLINK_RESTART_CH0 置位此位，挂载新的接收链表描述符。(R/W/SC)

GDMA_INLINK_PARK_CH0 1：接收链表描述符的状态机空闲；0：接收链表描述符的状态机工作中。(RO)

Register 2.10. GDMA_OUT_CONF0_CH0_REG (0x00D0)



GDMA_OUT_RST_CH0 用于复位 GDMA 通道 0 的 TX 状态机和 TX FIFO 指针。(R/W)

GDMA_OUT_LOOP_TEST_CH0 保留。(R/W)

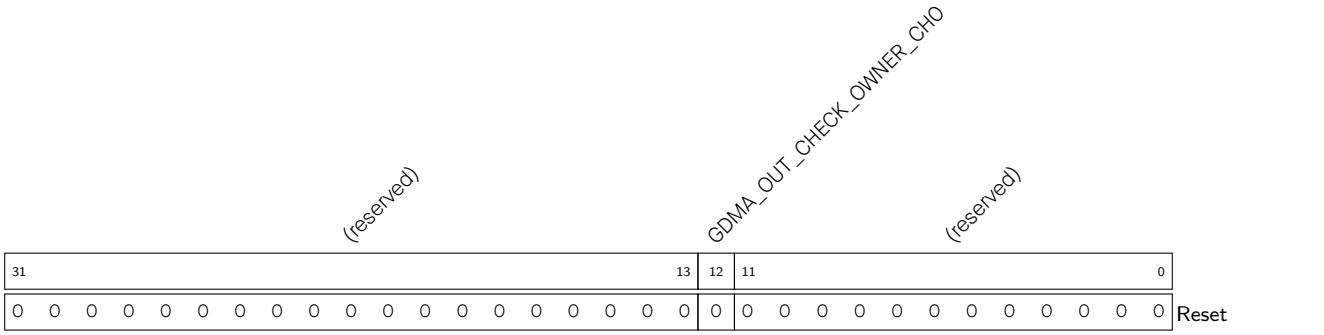
GDMA_OUT_AUTO_WRBACK_CH0 置位此位，在 TX FIFO 中所有数据发送出去后自动回写发送链表。(R/W)

GDMA_OUT_EOF_MODE_CH0 发送数据时生成 EOF 标志位。1：需要发送的数据已从 GDMA FIFO 中弹出时，发送通道 0 的 EOF 标志生成。(R/W)

GDMA_OUTDSCR_BURST_EN_CH0 将此位置 1，在发送通道 0 访问内部 RAM 读取发送链表描述符时使能 INCR 突发传输。(R/W)

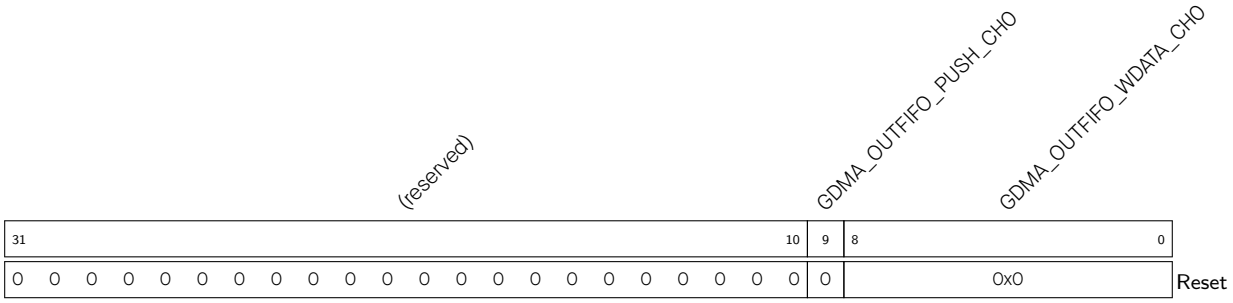
GDMA_OUT_DATA_BURST_EN_CH0 将此位置 1，在发送通道 0 访问内部 RAM 发送数据时使能 INCR 突发传输。(R/W)

Register 2.11. GDMA_OUT_CONF1_CH0_REG (0x00D4)



GDMA_OUT_CHECK_OWNER_CH0 置位此位，使能链表描述符的 owner 位检查。(R/W)

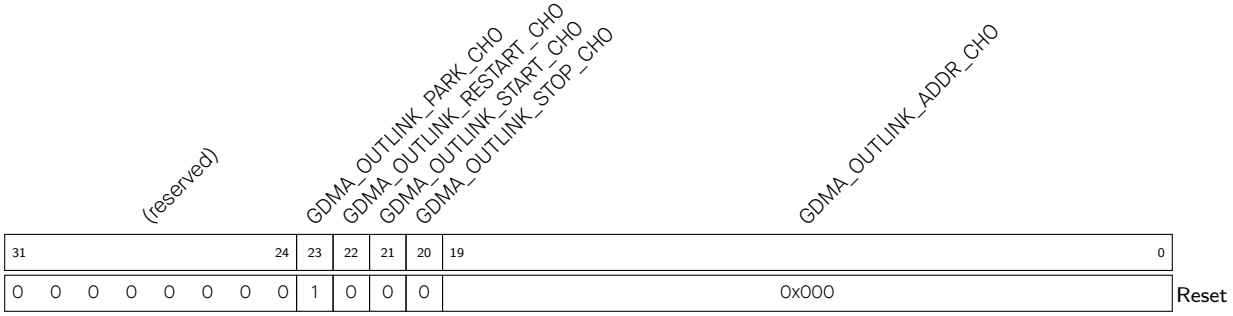
Register 2.12. GDMA_OUT_PUSH_CH0_REG (0x00DC)



GDMA_OUTFIFO_WDATA_CH0 存储需推送至 GDMA FIFO 的数据。(R/W)

GDMA_OUTFIFO_PUSH_CH0 置位此位，将数据推送至 GDMA FIFO 中。(R/W/SC)

Register 2.13. GDMA_OUT_LINK_CH0_REG (0x00E0)



GDMA_OUTLINK_ADDR_CH0 存储第一个发送链表描述符地址的低 20 位。(R/W)

GDMA_OUTLINK_STOP_CH0 置位此位，停止处理发送链表描述符。(R/W/SC)

GDMA_OUTLINK_START_CH0 置位此位，开始处理发送链表描述符。(R/W/SC)

GDMA_OUTLINK_RESTART_CH0 置位此位，在最后一个地址挂载新的发送链表。(R/W/SC)

GDMA_OUTLINK_PARK_CH0 1：发送链表描述符的状态机空闲；0：发送链表描述符的状态机工作中。(RO)

Register 2.14. GDMA_INFIFO_STATUS_CH0_REG (0x0078)

(reserved)	GDMA_IN_BUF_HUNGRY_CH0	GDMA_IN_REMAIN_UNDER_4B_CH0	GDMA_IN_REMAIN_UNDER_3B_CH0	GDMA_IN_REMAIN_UNDER_2B_CH0	GDMA_IN_REMAIN_UNDER_1B_CH0	(reserved)	GDMA_INFIFO_CNT_CH0	GDMA_INFIFO_EMPTY_CH0	GDMA_INFIFO_FULL_CH0	
31 28	27	26	25	24	23	22 8	7 2	1	0	
0 0 0 0	0	1	1	1	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	1	1	Reset

GDMA_INFIFO_FULL_CH0 接收通道 0 的 L1 RX FIFO 已满。(RO)

GDMA_INFIFO_EMPTY_CH0 接收通道 0 的 L1 RX FIFO 为空。(RO)

GDMA_INFIFO_CNT_CH0 接收通道 0 的 L1 RX FIFO 存储的字节数。(RO)

GDMA_IN_REMAIN_UNDER_1B_CH0 保留。(RO)

GDMA_IN_REMAIN_UNDER_2B_CH0 保留。(RO)

GDMA_IN_REMAIN_UNDER_3B_CH0 保留。(RO)

GDMA_IN_REMAIN_UNDER_4B_CH0 保留。(RO)

GDMA_IN_BUF_HUNGRY_CH0 保留。(RO)

Register 2.15. GDMA_IN_STATE_CH0_REG (0x0084)

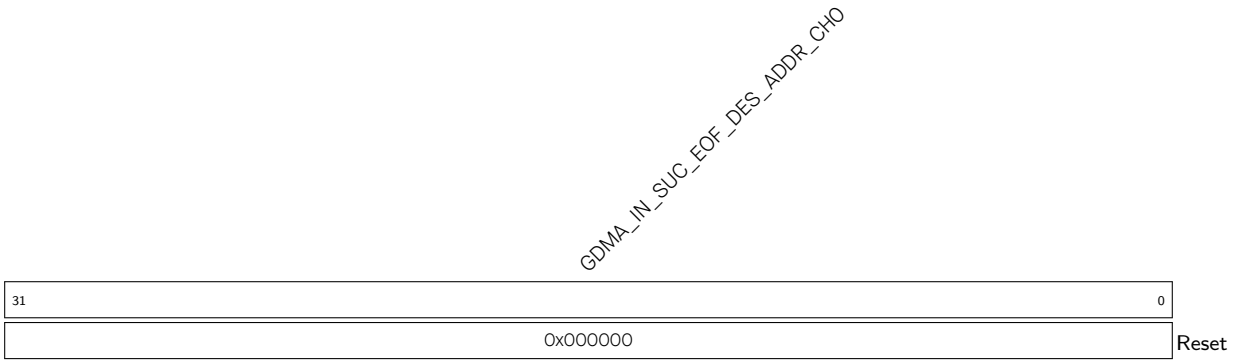
(reserved)	GDMA_IN_STATE_CH0	GDMA_IN_DSCR_STATE_CH0	GDMA_INLINK_DSCR_ADDR_CH0	
31 23	22 20	19 18	17 0	
0 0 0 0 0 0 0 0 0	0	0	0	Reset

GDMA_INLINK_DSCR_ADDR_CH0 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_CH0 保留。(RO)

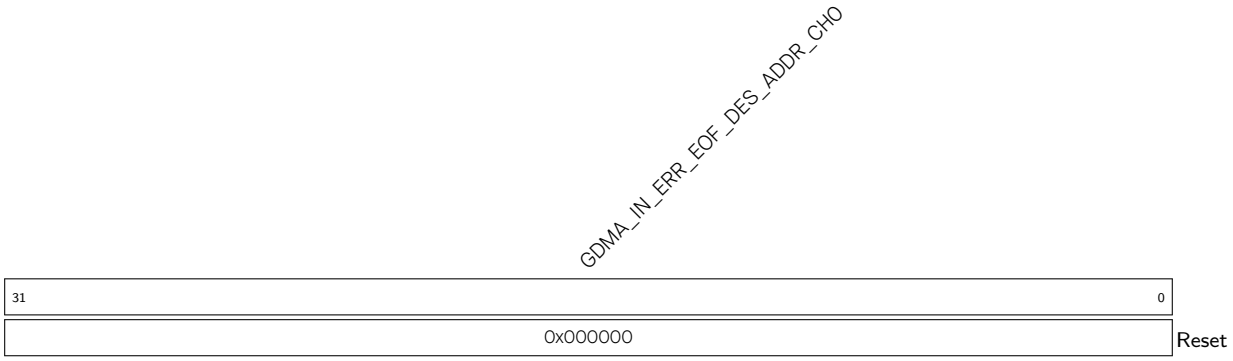
GDMA_IN_STATE_CH0 保留。(RO)

Register 2.16. GDMA_IN_SUC_EOF_DES_ADDR_CH0_REG (0x0088)



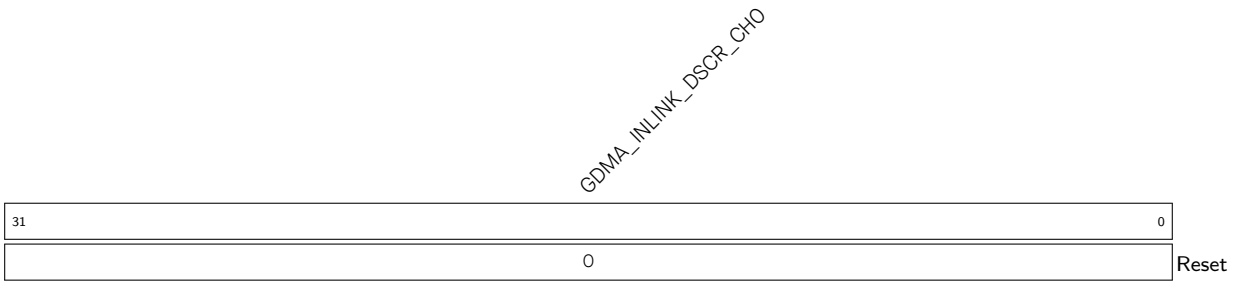
GDMA_IN_SUC_EOF_DES_ADDR_CH0 接收链表描述符的 EOF 为 1 时，该描述符的地址。(RO)

Register 2.17. GDMA_IN_ERR_EOF_DES_ADDR_CH0_REG (0x008C)



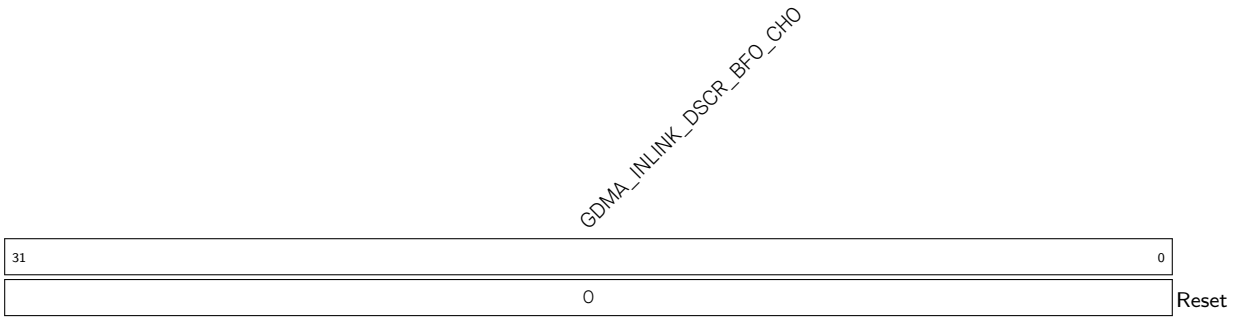
GDMA_IN_ERR_EOF_DES_ADDR_CH0 保留。(RO)

Register 2.18. GDMA_IN_DSCR_CH0_REG (0x0090)



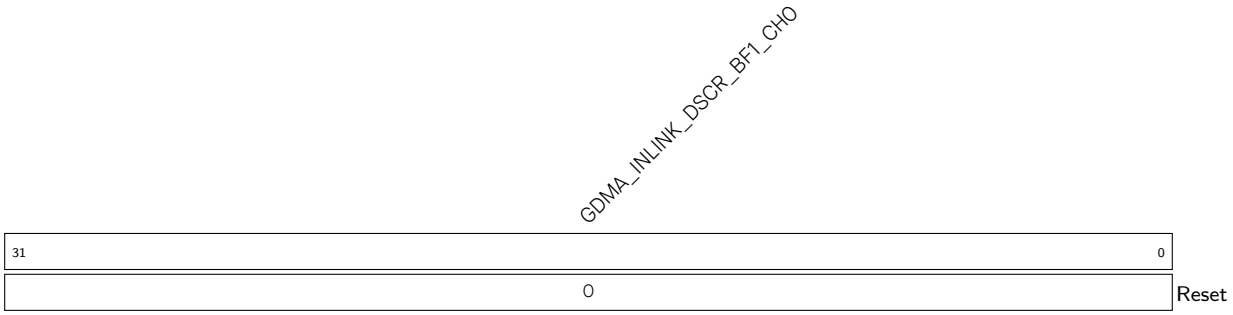
GDMA_INLINK_DSCR_CH0 表示当前已预读取的接收链表描述符指向的下一个接收链表描述符地址 x+1。(RO)

Register 2.19. GDMA_IN_DSCR_BF0_CH0_REG (0x0094)



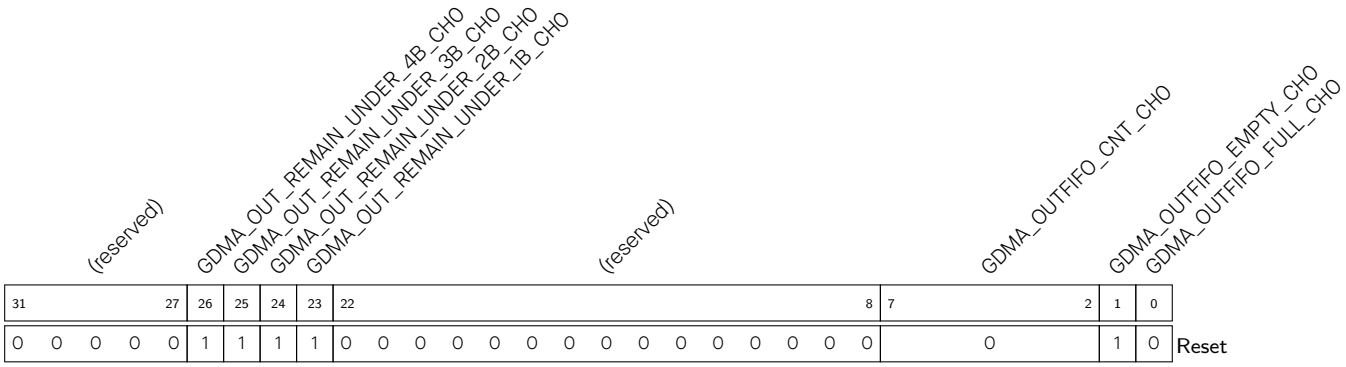
GDMA_INLINK_DSCR_BF0_CH0 表示当前已预读取的接收链表描述符所在地址 x。(RO)

Register 2.20. GDMA_IN_DSCR_BF1_CH0_REG (0x0098)



GDMA_INLINK_DSCR_BF1_CH0 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 2.21. GDMA_OUTFIFO_STATUS_CH0_REG (0x00D8)



GDMA_OUTFIFO_FULL_CH0 发送通道 0 的 L1 TX FIFO 已满。(RO)

GDMA_OUTFIFO_EMPTY_CH0 发送通道 0 的 L1 TX FIFO 为空。(RO)

GDMA_OUTFIFO_CNT_CH0 发送通道 0 的 L1 TX FIFO 存储的字节数。(RO)

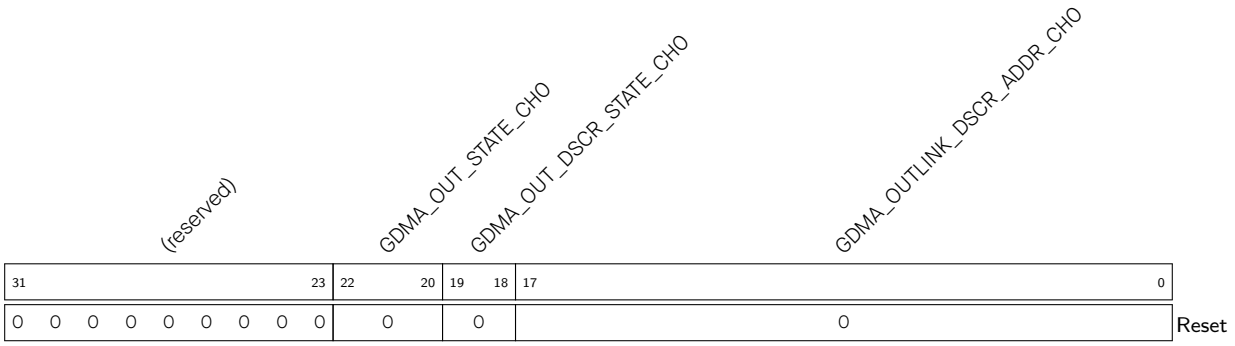
GDMA_OUT_REMAIN_UNDER_1B_CH0 保留。(RO)

GDMA_OUT_REMAIN_UNDER_2B_CH0 保留。(RO)

GDMA_OUT_REMAIN_UNDER_3B_CH0 保留。(RO)

GDMA_OUT_REMAIN_UNDER_4B_CH0 保留。(RO)

Register 2.22. GDMA_OUT_STATE_CH0_REG (0x00E4)

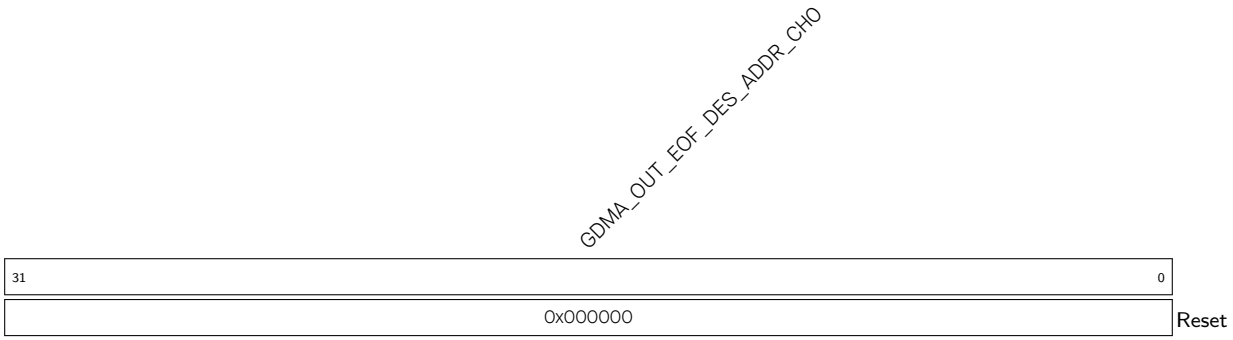


GDMA_OUTLINK_DSCR_ADDR_CH0 表示下一个预读取（但未处理）的发送接收链表描述符所在地址的低 18 位。如果当前处理的发送链表描述符是最后一个描述符，则该字段表示当前处理的发送链表描述符地址。(RO)

GDMA_OUT_DSCR_STATE_CH0 保留。(RO)

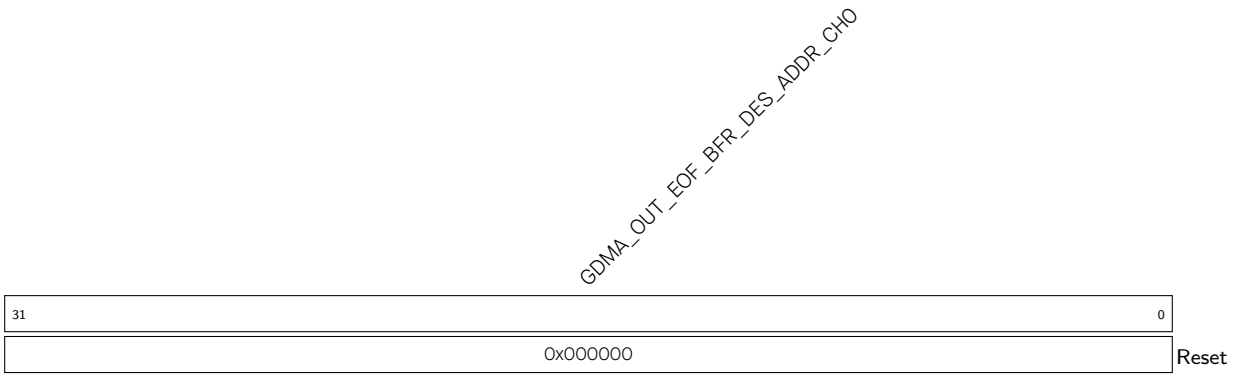
GDMA_OUT_STATE_CH0 保留。(RO)

Register 2.23. GDMA_OUT_EOF_DES_ADDR_CH0_REG (0x00E8)



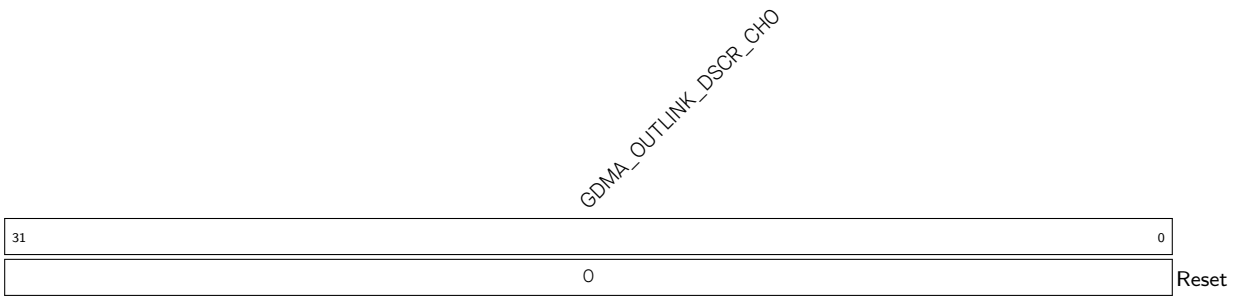
GDMA_OUT_EOF_DES_ADDR_CH0 发送链表描述符的 EOF 为 1 时，该描述符的地址。(RO)

Register 2.24. GDMA_OUT_EOF_BFR_DES_ADDR_CH0_REG (0x00EC)



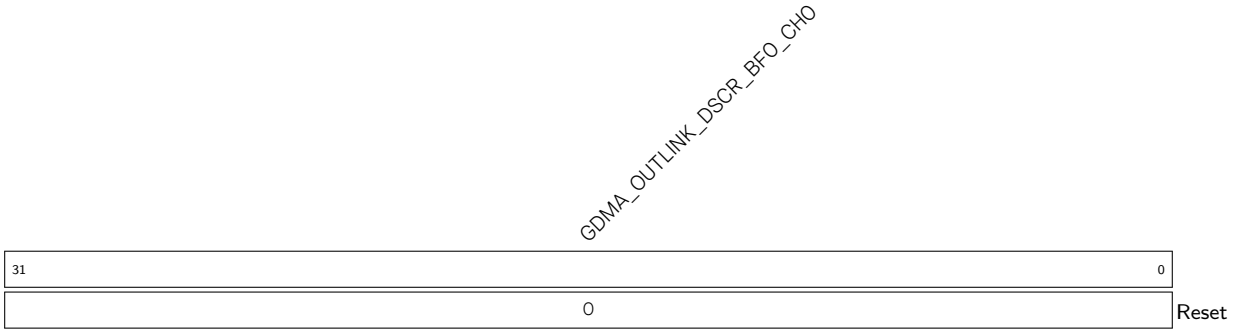
GDMA_OUT_EOF_BFR_DES_ADDR_CH0 倒数第二个发送链表描述符的地址。(RO)

Register 2.25. GDMA_OUT_DSCR_CH0_REG (0x00F0)



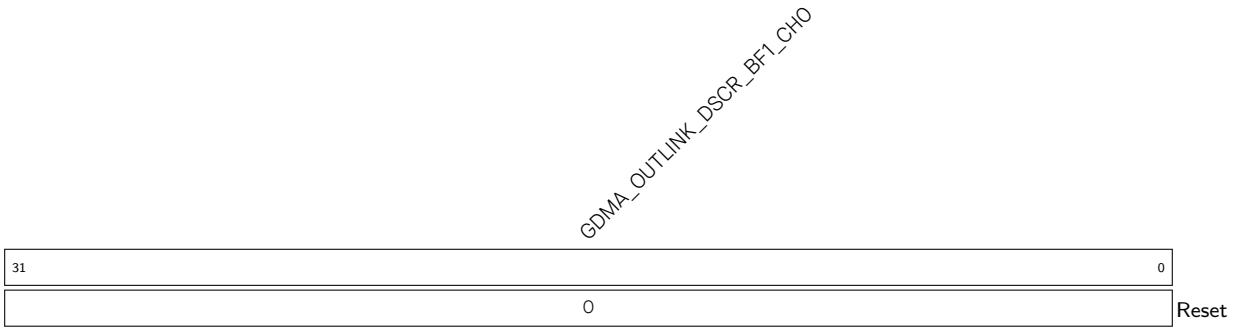
GDMA_OUTLINK_DSCR_CH0 表示当前已预读取的发送链表描述符指向的下一个发送链表描述符地址 y+1。(RO)

Register 2.26. GDMA_OUT_DSCR_BF0_CH0_REG (0x00F4)



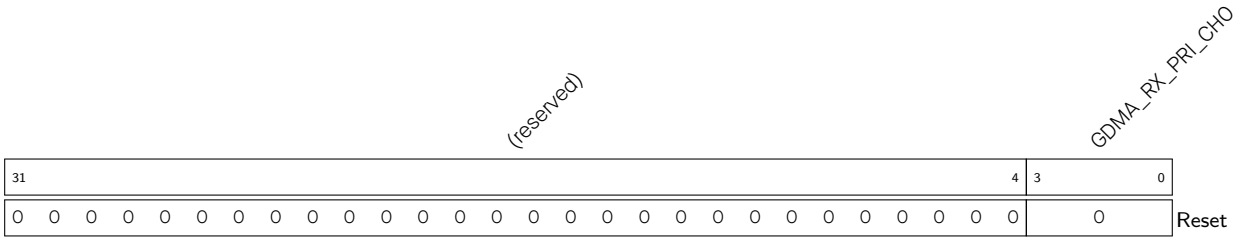
GDMA_OUTLINK_DSCR_BF0_CH0 表示当前已预读取的发送链表描述符所在地址 y。(RO)

Register 2.27. GDMA_OUT_DSCR_BF1_CH0_REG (0x00F8)



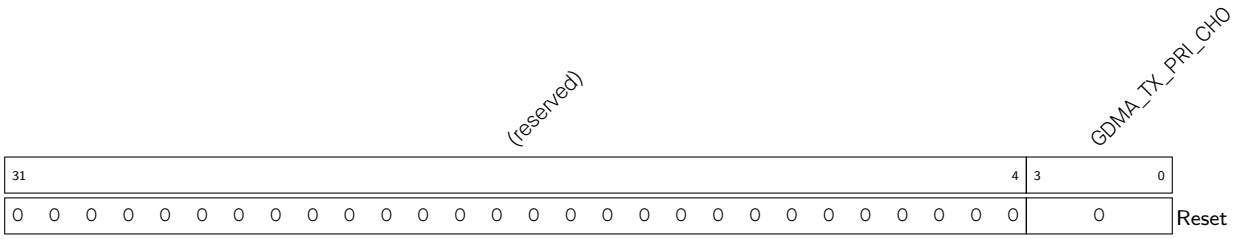
GDMA_OUTLINK_DSCR_BF1_CH0 表示前一个已预读取的发送链表描述符所在地址 y-1。(RO)

Register 2.28. GDMA_IN_PRI_CH0_REG (0x009C)



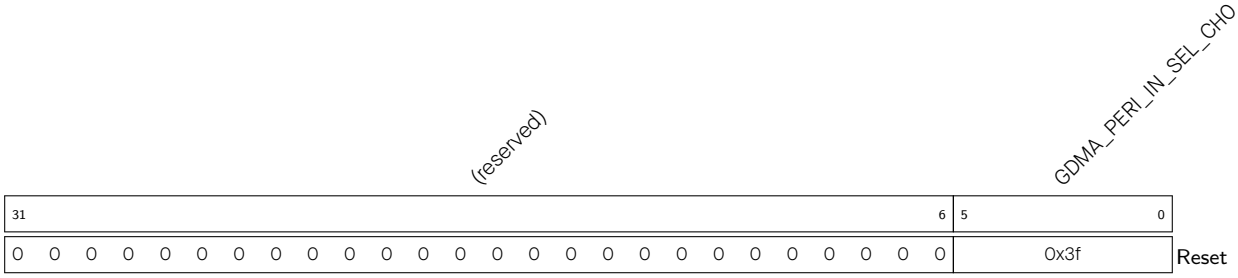
GDMA_RX_PRI_CH0 接收通道 0 的优先级。该值越大，优先级越高。(R/W)

Register 2.29. GDMA_OUT_PRI_CH0_REG (0x00FC)



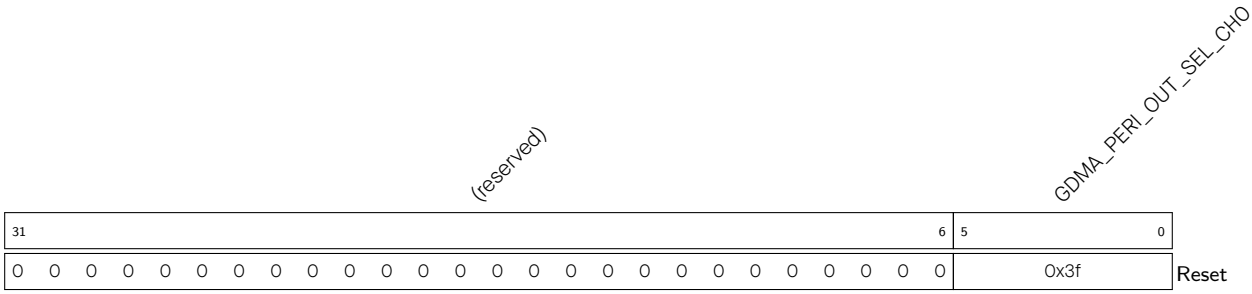
GDMA_TX_PRI_CH0 发送通道 0 的优先级。该值越大，优先级越高。(R/W)

Register 2.30. GDMA_IN_PERI_SEL_CH0_REG (0x00A0)



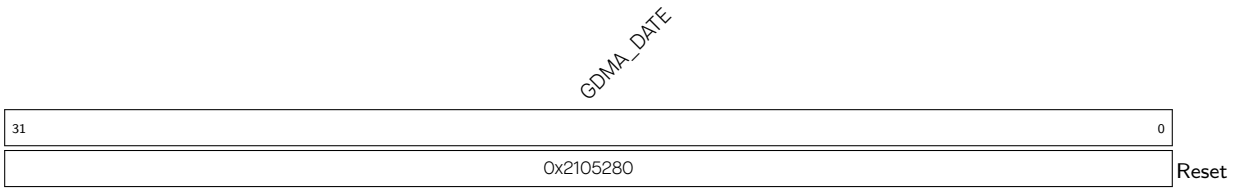
GDMA_PERI_IN_SEL_CH0 用于选择接收通道 0 连接的外设。0：SPI2；1：保留；2：保留；3：保留；4：保留；5：保留；6：保留；7：保留；8：保留；9 ~ 63：无效值。(R/W)

Register 2.31. GDMA_OUT_PERI_SEL_CH0_REG (0x0100)



GDMA_PERI_OUT_SEL_CH0 用于选择发送通道 0 连接的外设。0：SPI2；1：保留；2：保留；3：保留；4：保留；5：保留；6：保留；7：SHA；8：保留；9 ~ 63：无效值。(R/W)

Register 2.32. GDMA_DATE_REG (0x0048)



GDMA_DATE 版本控制寄存器。(R/W)

3 系统和存储器

3.1 概述

ESP8684 是一个超低功耗和高度集成的片上系统，它集成了一颗 RISC-V 32 位单核处理器，四级流水线架构，主频高达 120 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。

3.2 主要特性

ESP8684 的系统与存储器具有如下特性：

- **地址空间**
  - 848 KB 内部存储器指令地址空间
  - 576 KB 内部存储器数据地址空间
  - 828 KB 外设地址空间
  - 4 MB 外部存储器指令虚地址空间
  - 4 MB 外部存储器数据虚地址空间
  - 576 KB 内部 DMA 地址空间
- **内部存储器**
  - 576 KB 内部 ROM
  - 272 KB 内部 SRAM
- **外部存储器**
  - 最大支持 16 MB 片外 flash
- **外设空间**
  - 总计 22 个模块/外设
- **GDMA**
  - 2 个具有 GDMA 功能的模块/外设

图 3-1 描述了系统结构与地址映射结构。

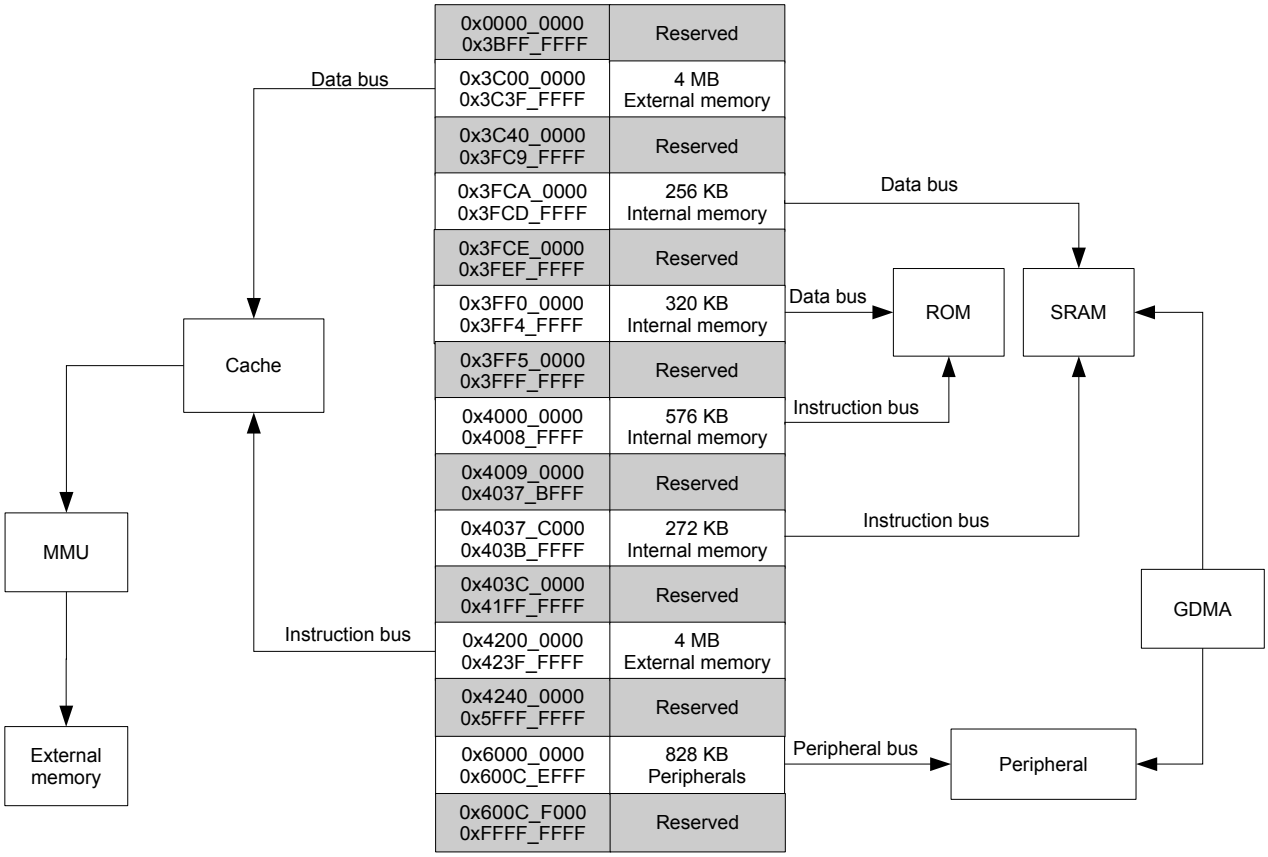


图 3-1. 系统结构与地址映射结构

说明：

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

3.3 功能描述

3.3.1 地址映射

地址 0x4000_0000 以下的部分属于数据总线的地址范围，地址 0x4000_0000 ~ 0x4FFF_FFFF 部分为指令总线的地址范围，地址 0x5000_0000 及以上的部分是外设总线的地址范围。

CPU 的数据总线与指令总线都为小端序。CPU 可以通过数据总线进行单字节、双字节、4 字节的数据访问。CPU 也可以通过指令总线进行数据访问，但只能是 4 字节对齐的访问。

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 访问映射到虚地址空间的外部存储器；
- 通过外设总线直接访问模块/外设。

图 3-1 描述了数据总线、指令总线与外设总线中的各段地址所能访问的目标。

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

3.3.2 内部存储器

ESP8684 的内部存储器包含如下两种类型：

- 内部 ROM (576 KB)：内部 ROM 是只读存储器，不可编程。其中存放有一些系统底层软件的 ROM 代码（软件指令和一些只读数据）。
- 内部 SRAM (272 KB)：内部静态存储器（SRAM）是易失性存储器，可以快速响应 CPU 的访问请求（通常一个 CPU 时钟周期）。
  - SRAM 中的一部分可以被配置成外部存储器访问的缓存。
  - SRAM 中的某些部分只可以被 CPU 的指令总线访问。
  - SRAM 中的某些部分既可以被 CPU 的指令总线访问，又可以被 CPU 的数据总线访问。

基于上述对两种类型的内部存储器的描述，ESP8684 的内部存储器可以被分为两个部分：内部 ROM (576 KB)、内部 SRAM (272 KB)。

CPU 通过不同的总线访问这几部分内部存储器时会有些许限制（如某些部分只允许 CPU 通过数据总线访问），据此内部存储器可以被区分的更加细致。表 3-1 列出了所有内部存储器以及可以访问内部存储器的数据总线与指令总线地址段。

表 3-1. 内部存储器地址映射

总线类型	边界地址		容量 (KB)	目标
	低位地址	高位地址		
数据	0x3FF0_0000	0x3FF4_FFFF	320	内部 ROM 1
	0x3FCA_0000	0x3FCD_FFFF	256	内部 SRAM 1
指令	0x4000_0000	0x4003_FFFF	256	内部 ROM 0
	0x4004_0000	0x4008_FFFF	320	内部 ROM 1
	0x4037_C000	0x4037_FFFF	16	内部 SRAM 0
	0x4038_0000	0x403B_FFFF	256	内部 SRAM 1

1. 内部 ROM 0

内部 ROM 0 的容量为 256 KB，只读。如表 3-1 所示，CPU 只可以通过指令总线地址段 0x4000_0000 ~ 0x4003_FFFF 访问这部分存储器。

2. 内部 ROM 1

内部 ROM 1 的容量为 320 KB，只读。如表 3-1 所示，CPU 可以通过指令总线地址段 0x4004_0000 ~ 0x4008_FFFF 或数据总线地址段 0x3FF0_0000 ~ 0x3FF4_FFFF 同序访问这部分存储器。

这两段地址同序访问内部 ROM 1 是指：地址 04004_0000 与 0x3FF0_0000 访问到相同的字，0x4004_0004 与 0x3FF0_0004 访问到相同的字，0x4004_0008 与 0x3FF0_0008 访问到相同的字，以此类推（下文的“同序访问”也参照此描述）。

3. 内部 SRAM 0

内部 SRAM 0 的容量为 16 KB，可读可写。如表 3-1 所示，CPU 只可以通过指令总线访问这部分存储器。

这部分存储器可以被配置为指令缓存，用来缓存外部存储器的指令或只读数据。此时，已被配置为指令缓存的部分不可以被 CPU 访问。

4. 内部 SRAM 1

内部 SRAM 1 容量为 256 KB，可读可写。如表 3-1 所示，CPU 可以通过数据或指令总线同序访问。

3.3.3 外部存储器

ESP8684 支持以 SPI、Dual SPI、Quad SPI、QPI 等接口形式连接片外 flash。ESP8684 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。

3.3.3.1 外部存储器地址映射

CPU 借助缓存（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 的地址映射为访问片外存储的实地址。经过地址映射，ESP8684 最大支持 16 MB 的片外 flash。

通过高速缓存，ESP8684 可支持以下地址空间映射。请注意，指令总线地址空间（4 MB）和数据总线地址空间 (4 MB) 是共用的。

- 4 MB 的指令总线地址空间以 64 KB 为单位映射到片外 flash。
- 4 MB 的数据总线（只读）地址空间以 64 KB 为单位映射到片外 flash。

表 3-2 列出了在访问外部存储器时 CPU 的数据总线与指令总线与 Cache 的对应关系。

表 3-2. 外部存储器地址映射

总线类型	边界地址		容量 (MB)	目标
	低位地址	高位地址		
数据（只读）	0x3C00_0000	0x3C3F_FFFF	4	Uniform Cache
指令	0x4200_0000	0x423F_FFFF	4	Uniform Cache

3.3.3.2 高速缓存

如图 3-2 所示，ESP8684 采用一个只读的统一 cache，为四路组相联，容量为 16 KB，块大小为 32 字节。当 cache 处于工作状态时，将占用部分内部存储空间（参见第 3.3.2 节关于内部 SRAM 0 的描述）。

指令总线和数据总线可以同时访问该 cache，但此时 cache 只能对其中一个作出相应。当 cache 缺失时，cache 控制器会向外部存储器发起请求。

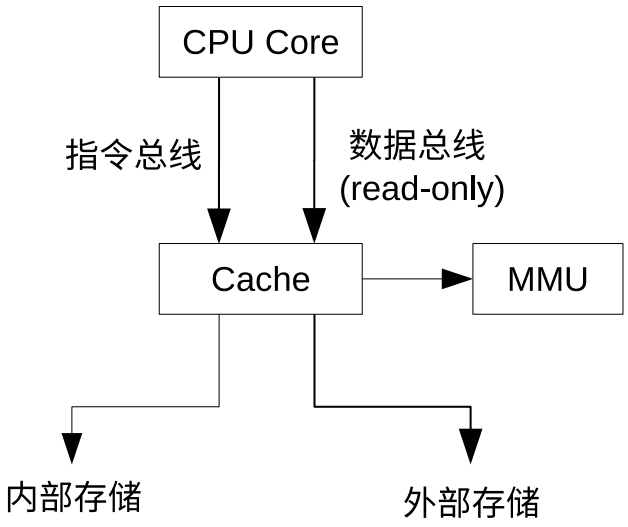


图 3-2. Cache 系统结构

3.3.3.3 Cache 操作

ESP8684 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：自动失效 (Auto-Invalidate) 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，而自动失效会对 cache 中的所有数据做失效处理。
2. **预取 (Preload)**：功能用于将指令和数据提前加载到 cache 中。预取操作的最小单位为 1 个块 (32 字节)。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，cache 在填充缺失数据到 cache 时，如果该数据落在指定区域，则将该数据锁定，未落入指定区域的数据不会被锁定。手动锁定开启时，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。

3.3.4 GDMA 地址空间

ESP8684 中的 GDMA (General Direct Memory Access) 外设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内部 SRAM 1，即 GDMA 通过地址 0x3FCA_0000 ~ 0x3FCD_FFFF 访问内部 SRAM 1。请注意，GDMA 无法访问被 cache 占用的内部存储器。

ESP8684 中共有 2 个外设/模块可以和 GDMA 联合工作，分别是 SPI2 和 SHA Accelerator，两个外设共用 GDMA 的一个通道，不可以同时开启 GDMA 功能。

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 *2 通用 DMA 控制器 (GDMA)*。

3.3.5 模块/外设

CPU 可通过外设总线的地址段 0x6000_0000 ~ 0x600C_EFFF 访问模块/外设。

3.3.5.1 模块/外设地址空间映射

表 3-3 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 3-3. 模块/外设地址空间映射表

目标	边界地址		容量 (KB)	说明
	低位地址	高位地址		
UART 控制器 0	0x6000_0000	0x6000_0FFF	4	
保留	0x6000_1000	0x6000_1FFF		
SPI 控制器 1	0x6000_2000	0x6000_2FFF	4	
SPI 控制器 0	0x6000_3000	0x6000_3FFF	4	
GPIO	0x6000_4000	0x6000_4FFF	4	
保留	0x6000_5000	0x6000_7FFF		
低功耗管理	0x6000_8000	0x6000_8FFF	4	
IO MUX	0x6000_9000	0x6000_9FFF	4	
保留	0x6000_A000	0x6000_CFFF		
MISC	0x6000_D000	0x6000_DFFF	4	
保留	0x6000_E000	0x6000_FFFF		
UART 控制器 1	0x6001_0000	0x6001_0FFF	4	
保留	0x6001_1000	0x6001_2FFF		
I2C 控制器	0x6001_3000	0x6001_3FFF	4	
保留	0x6001_4000	0x6001_8FFF		
LED PWM 控制器	0x6001_9000	0x6001_9FFF	4	
保留	0x6001_A000	0x6001_EFFF		
定时器组 0	0x6001_F000	0x6001_FFFF	4	
保留	0x6002_0000	0x6002_2FFF		
系统定时器	0x6002_3000	0x6002_3FFF	4	
SPI 控制器 2	0x6002_4000	0x6002_4FFF	4	
保留	0x6002_5000	0x6002_5FFF		
SYSCON	0x6002_6000	0x6002_6FFF	4	
保留	0x6002_7000	0x6003_AFFF		
SHA 加速器	0x6003_B000	0x6003_BFFF	4	
ECC 加速器	0x6003_E000	0x6003_EFFF	4	
保留	0x6002_C000	0x6003_EFFF		
通用 DMA 控制器	0x6003_F000	0x6003_FFFF	4	

见下页

表 3-3 – 接上页

目标	边界地址		容量 (KB)	说明
	低位地址	高位地址		
ADC 控制器	0x6004_0000	0x6004_0FFF	4	
保留	0x6004_1000	0x600B_FFFF		
系统寄存器	0x600C_0000	0x600C_0FFF	4	
Sensitive Register	0x600C_1000	0x600C_1FFF	4	
中断矩阵	0x600C_2000	0x600C_2FFF	4	
保留	0x600C_3000	0x600C_3FFF		
Configure Cache	0x600C_4000	0x600C_DFFF	40	
保留	0x600C_E000	0x600C_DFFF		
辅助调试	0x600C_E000	0x600C_EFFF	4	

4 eFuse 控制器 (eFuse)

4.1 概述

ESP8684 系统中有一块 1024 位的 eFuse 存储器用于存储参数内容和用户数据，参数内容包括一些硬件模块的控制参数、系统数据参数以及加解密模块使用的密钥等。eFuse 存储器的各个位一旦被烧写为 1，则不能再恢复为 0。eFuse 控制器按照用户配置完成对 eFuse 存储器中各参数中的各个位的烧写。从芯片外部，eFuse 数据只能通过 eFuse 控制器读取。对于某些数据，如果未启用读保护，则可以从芯片外部读取该数据；如果启用了读保护，则无法从芯片外部读取该数据。不过，存储在 eFuse 中的某些密钥始终可以供硬件加密模块 (例如数字签名、HMAC 等) 在内部使用，芯片外部无法获得这些数据。

4.2 主要特性

eFuse 控制器具有以下特性：

- 1024 位一次性可编程存储，有 256 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 eFuse 存储器内的参数内容

4.3 功能描述

4.3.1 结构

eFuse 存储器从结构上分成 4 个块 (BLOCK0 ~ BLOCK3)。

BLOCK0 存储了大部分硬件模块使用的控制参数。

表 4-1 列出了用户可访问（可读并可用）的所有 BLOCK0 中的参数名称、偏移地址、位宽、是否可供硬件使用、烧写保护，以及功能描述。

在这些参数中，**EFUSE_WR_DIS** 用于控制其他参数的烧写，**EFUSE_RD_DIS** 用于控制用户读取 BLOCK3 的参数。更多关于这两个参数的信息请见章节4.3.1.1、4.3.1.2。

表 4-1. BLOCK0 参数

参数	位宽	硬件使用	EFUSE_WR_DIS 烧写保护位	描述
EFUSE_WR_DIS	8	Y	N/A	禁止 eFuse 烧写对应参数
EFUSE_RD_DIS	2	Y	0	禁止用户读取 eFuse BLOCK3 的内容
EFUSE_WDT_DELAY_SEL	2	Y	1	表示 RTC 看门狗超时阈值
EFUSE_DIS_PAD_JTAG	1	Y	1	永久禁用 JTAG
EFUSE_DIS_DOWNLOAD_ICACHE	1	Y	1	在下载模式下关闭 iCache
EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT	1	Y	2	在 download boot 模式下禁用手动 flash 加密功能
EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT	3	Y	2	使能 SPI 启动加解密
EFUSE_XTS_KEY_LENGTH_256	1	Y	2	表示 XTS_AES 密钥长度
EFUSE_UART_PRINT_CONTROL	2	N	3	控制 UART boot 信息输出模式
EFUSE_FORCE_SEND_RESUME	1	N	3	强制 ROM 代码在 SPI 启动过程中发送 SPI flash 恢复指令
EFUSE_DIS_DOWNLOAD_MODE	1	N	3	关闭所有 Download 模式
EFUSE_DIS_DIRECT_BOOT	1	N	3	关闭 Direct_boot 模式
EFUSE_ENABLE_SECURITY_DOWNLOAD	1	N	3	使能 UART 安全下载模式
EFUSE_FALSH_TPUW	4	N	3	SoC 上电后 flash 启动延迟时间
EFUSE_SECURE_BOOT_EN	1	N	2	使能 secure boot
EFUSE_SECURE_VERSION	4	N	4	安全版本
EFUSE_CUSTOM_MAC_USED	1	N	4	使用用户自定义的 MAC

表 4-2 列出了 BLOCK1 ~ BLOCK3 中存储的参数的信息。

表 4-2. BLOCK1-3 参数

块	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_CUSTOMED_MAC	88	N	5	N/A	用户自定义 MAC 地址或用户数据
BLOCK2	EFUSE_SYS_DATA_PART1	48	N	6	N/A	MAC 地址
		208	N	6	N/A	系统数据
BLOCK3	EFUSE_KEY0	128	Y	7	[0]	KEY 或用户数据
		128	Y	7	[1]	KEY 或用户数据

BLOCK1 ~ BLOCK3 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 4.3.1.3 和章节 4.3.2。

4.3.1.1 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 4-1 以及表 4-2 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。

4.3.1.2 EFUSE_RD_DIS

所有参数中，只有 BLCOK3 的参数受用户读取保护状态的约束，即表 4-2 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。

如果对应的 EFUSE_RD_DIS 位为 0，则表示用户可以读取该数据；若对应的 EFUSE_RD_DIS 位为 1，则表示此位管理的参数处于用户读取保护状态。

除 BLOCK3 之外，其他参数不受读取保护状态的约束，均可被用户读取。

4.3.1.3 数据存储方式

eFuse 控制器使用硬件编码机制保护数据，对用户不可见。

BLOCK0 使用 4 备份方式存储参数，即 BLOCK0 中的所有参数（除了 EFUSE_WR_DIS）均在 eFuse 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 8 比特，其余参数为 32 比特，因此 BLOCK0 在 eFuse 存储器中共占据了 8 + 32 * 4 = 136 比特的存储空间。

BLOCK1 ~ BLOCK3 使用 RS (44, 32) 编码方式，最多支持自动校正 6 个字节。本文 RS (44, 32) 使用的本源多项式为 $p(x) = x^8 + x^4 + x^3 + x^2 + 1$。

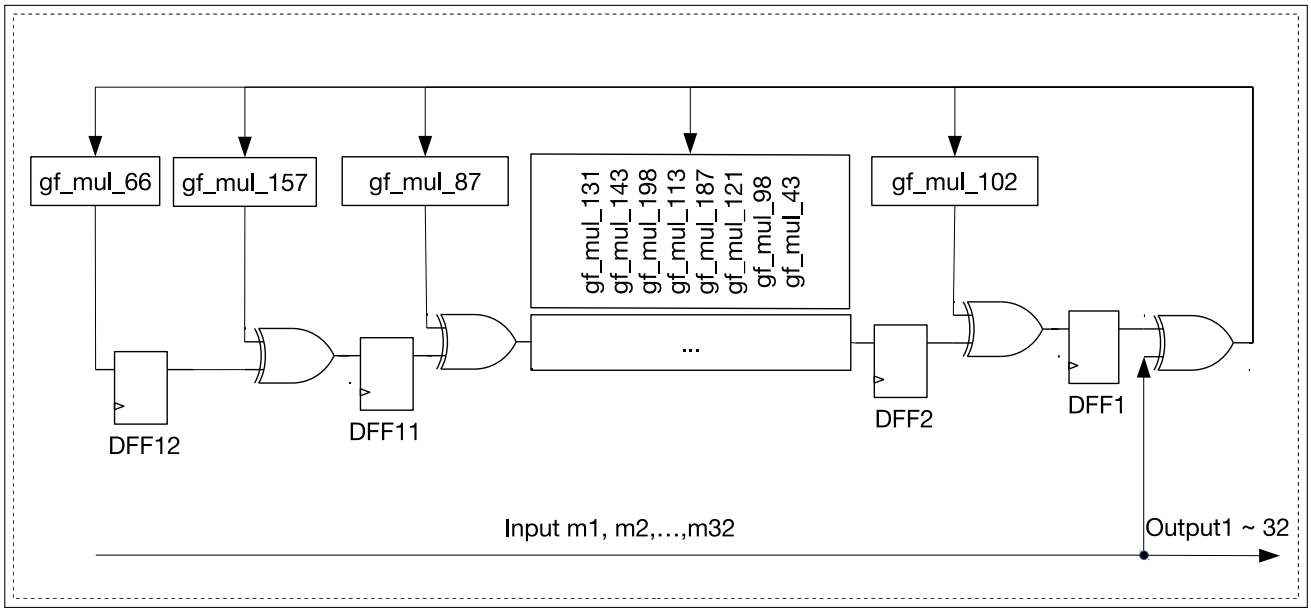


图 4-1. 移位寄存器电路图（前 32 字节）

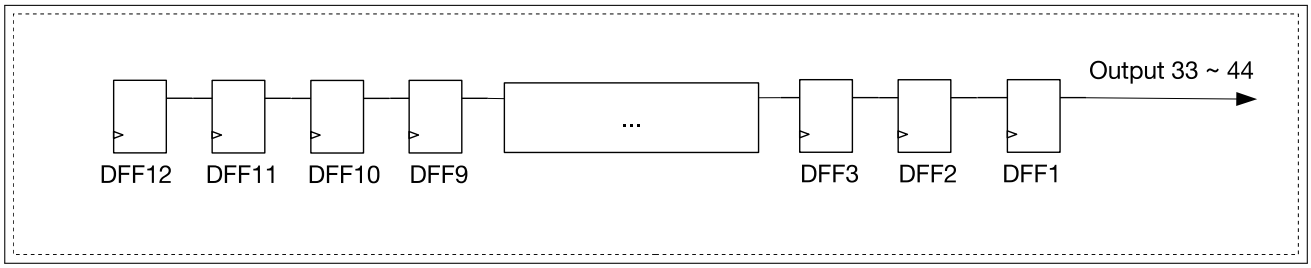


图 4-2. 移位寄存器电路图（后 12 字节）

如图 4-1 和 4-2 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：

- 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 存储器。eFuse 控制器会在读 eFuse 存储器的过程中自动完成解码和自动校正。

由于 RS 校验码是在整个 256 位的 eFuse block 上生成的，因此每个 block 只能写入一次。

BLOCK1 由于数据比特不足 256 比特，因此不足 256 比特的部分在 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 88 比特，RS 校验码为 96 比特，因此 BLOCK1 在 eFuse 存储器中共占据了 88 + 96 = 184 比特的存储空间。

BLOCK2 和 BLOCK3 的数据参数均为 256 比特，RS 校验码为 96 比特，因此在 eFuse 存储器中共占据了（256 + 96 ）* 2 = 704 比特的存储空间。

4.3.2 烧写参数

烧写 eFuse 参数时，需要按块烧写。BLOCK0 ~ BLOCK3 共用同一段地址来存储即将烧写的参数并通过配置 EFUSE_BLK_NUM 参数表明当前需要烧写的是哪一个块。

烧写 BLOCK0

将寄存器域 EFUSE_BLK_NUM 配置为 0。

EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA1_REG 存储着 BLOCK0 即将烧写的参数。

EFUSE_PGM_DATA2_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

将寄存器域 EFUSE_BLK_NUM 配置为 1。EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA2_REG 存储着 BLOCK1 即将烧写的参数，EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中存储着对应的 RS 校验码。EFUSE_PGM_DATA3_REG ~ EFUSE_PGM_DATA7_REG 中的数据不影响 BLOCK1 的烧写。

烧写 BLOCK2 ~ 3

将寄存器域 EFUSE_BLK_NUM 配置为 2 或 3 。EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 存储着即将烧写的参数，EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中存储着对应的 RS 校验码。

烧写流程

烧写参数的流程如下：

1. 配置 EFUSE_BLK_NUM 参数，决定烧写哪一个块。
2. 将需要烧写的参数填写到寄存器 EFUSE_PGM_DATA0_REG ~ EFUSE_PGM_DATA7_REG 和 EFUSE_PGM_CHECK_VALUE0_REG ~ EFUSE_PGM_CHECK_VALUE2_REG 中。
3. 确保 eFuse 烧写电压 VDDQ 的配置正确，具体请参考章节 4.3.4。
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，或者等待烧写完成中断产生。识别烧写/读取完成中断产生的方法详见章节 4.3.3 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 控制器的读寄存器操作使写入的新值生效，具体请参考章节 4.3.3。
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 块，需要检查的错误寄存器如下。
   - BLOCK0: EFUSE_RD_REPEAT_ERR_REG
   - BLOCK1: EFUSE_RD_RS_ERR_REG[3:0]
   - BLOCK2: EFUSE_RD_RS_ERR_REG[7:4]

- BLOCK3: EFUSE_RD_RS_ERR_REG[11:8]

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。但是由于 eFuse 存储器本身的烧写使用寿命限制，我们并不推荐这样做，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。通过该方式可以有效保证烧写保护功能，防止EFUSE_WR_DIS的烧写和其保护的参数烧写出现混乱。另外严禁对已经烧写了的位重复烧写，否则将发生烧写错误。

BLOCK2 中数据信息在出厂时已经烧写完毕，不允许再次烧写。

BLOCK1 和 BLOCK3 中每一个 BLOCK 都只能烧写一次，不允许重复烧写或分多次烧写。

4.3.3 用户读取参数

用户不能直接读取 eFuse 存储器中烧写的信息内容。eFuse 控制器能够将烧写的数据信息读取到对应的地址段的寄存器内，用户再通过读取以 EFUSE_RD_ 开始的寄存器来获取 eFuse 存储器内的信息。下表 4-3 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 4-3. 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_REG	EFUSE_PGM_DATA1_REG
1	EFUSE_RD_BLK1_DATA0 ~ 2_REG	EFUSE_PGM_DATA0 ~ 2_REG
2	EFUSE_RD_BLK2_DATA0 ~ 7_REG	EFUSE_PGM_DATA0 ~ 7_REG
3	EFUSE_RD_BLK3_DATA0 ~ 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进行，也可以根据需要由用户主动发出（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 控制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 interrupt（读取完成中断）产生，识别烧写/读取完成中断产生的方法详见下方说明。
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 控制器读操作。

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

EFUSE_RD_RS_ERR_REG 寄存器记录 eFuse 控制器读 BLOCK1 ~ BLOCK3 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 控制器的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:
  1. 轮询寄存器 EFUSE_INT_RAW_REG 的位 1/0，直到位 1/0 为 1，表示烧写/读取操作完成。
- 方法 2:
  1. 将寄存器 EFUSE_INT_ENA_REG 的位 1/0 置 1，使 eFuse 控制器能够产生烧写/读取完成中断。
  2. 配置中断矩阵使 CPU 能够响应 eFuse 控制器的中断信号，可参见 8 中断矩阵 *(INTMTRX)*。
  3. 等待烧写/读取完成中断产生。
  4. 对寄存器 EFUSE_INT_CLR_REG 的位 1/0 置 1 以清除烧写/读取完成中断。

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。

4.3.4 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 x 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 4-4. 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

4.3.5 硬件模块使用参数

硬件模块使用参数是通过电路连接实现的，用户无法干预这个过程。硬件使用的参数为表 4-1 和 4-2 “硬件使用”一栏中标记为“Y”的参数。

4.3.6 中断

- 烧写完成中断：当 eFuse 控制器烧写完成后，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。

- 读取完成中断：当 eFuse 控制器读取完成后，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。

4.4 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 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

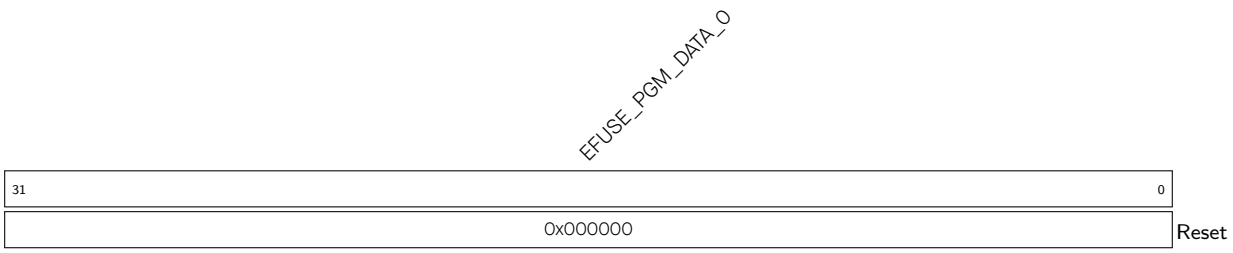
名称	描述	地址	访问
烧写数据寄存器			
EFUSE_PGM_DATA0_REG	待烧写数据配置寄存器 0	0x0000	R/W
EFUSE_PGM_DATA1_REG	待烧写数据配置寄存器 1	0x0004	R/W
EFUSE_PGM_DATA2_REG	待烧写数据配置寄存器 2	0x0008	R/W
EFUSE_PGM_DATA3_REG	待烧写数据配置寄存器 3	0x000C	R/W
EFUSE_PGM_DATA4_REG	待烧写数据配置寄存器 4	0x0010	R/W
EFUSE_PGM_DATA5_REG	待烧写数据配置寄存器 5	0x0014	R/W
EFUSE_PGM_DATA6_REG	待烧写数据配置寄存器 6	0x0018	R/W
EFUSE_PGM_DATA7_REG	待烧写数据配置寄存器 7	0x001C	R/W
EFUSE_PGM_CHECK_VALUE0_REG	待烧写 RS 码配置寄存器 0	0x0020	R/W
EFUSE_PGM_CHECK_VALUE1_REG	待烧写 RS 码配置寄存器 1	0x0024	R/W
EFUSE_PGM_CHECK_VALUE2_REG	待烧写 RS 码配置寄存器 2	0x0028	R/W
读取数据寄存器			
EFUSE_RD_WR_DIS_REG	BLOCK0 wr_dis 数据寄存器 0	0x002C	RO
EFUSE_RD_REPEAT_DATA0_REG	BLOCK0 数据寄存器 1	0x0030	RO
EFUSE_RD_BLK1_DATA0_REG	BLOCK1 数据寄存器 0	0x0034	RO
EFUSE_RD_BLK1_DATA1_REG	BLOCK1 数据寄存器 1	0x0038	RO
EFUSE_RD_BLK1_DATA2_REG	BLOCK1 数据寄存器 2	0x003C	RO
EFUSE_RD_BLK2_DATA0_REG	BLOCK2 数据寄存器 0	0x0040	RO
EFUSE_RD_BLK2_DATA1_REG	BLOCK2 数据寄存器 1	0x0044	RO
EFUSE_RD_BLK2_DATA2_REG	BLOCK2 数据寄存器 2	0x0048	RO
EFUSE_RD_BLK2_DATA3_REG	BLOCK2 数据寄存器 3	0x004C	RO
EFUSE_RD_BLK2_DATA4_REG	BLOCK2 数据寄存器 4	0x0050	RO
EFUSE_RD_BLK2_DATA5_REG	BLOCK2 数据寄存器 5	0x0054	RO
EFUSE_RD_BLK2_DATA6_REG	BLOCK2 数据寄存器 6	0x0058	RO
EFUSE_RD_BLK2_DATA7_REG	BLOCK2 数据寄存器 7	0x005C	RO
EFUSE_RD_BLK3_DATA0_REG	BLOCK3 数据寄存器 0	0x0060	RO
EFUSE_RD_BLK3_DATA1_REG	BLOCK3 数据寄存器 1	0x0064	RO
EFUSE_RD_BLK3_DATA2_REG	BLOCK3 数据寄存器 2	0x0068	RO
EFUSE_RD_BLK3_DATA3_REG	BLOCK3 数据寄存器 3	0x006C	RO
EFUSE_RD_BLK3_DATA4_REG	BLOCK3 数据寄存器 4	0x0070	RO
EFUSE_RD_BLK3_DATA5_REG	BLOCK3 数据寄存器 5	0x0074	RO
EFUSE_RD_BLK3_DATA6_REG	BLOCK3 数据寄存器 6	0x0078	RO
EFUSE_RD_BLK3_DATA7_REG	BLOCK3 数据寄存器 7	0x007C	RO
报告寄存器			
EFUSE_RD_REPEAT_ERR_REG	BLOCK0 烧写错误记录寄存器 0	0x0080	RO
EFUSE_RD_RS_ERR_REG	BLOCK1-3 烧写错误记录寄存器 0	0x0084	RO
配置寄存器			

名称	描述	地址	访问
EFUSE_CLK_REG	eFuse 时钟配置寄存器	0x0088	R/W
EFUSE_CONF_REG	eFuse 运行模式配置寄存器	0x008C	R/W
EFUSE_CMD_REG	eFuse 指令寄存器	0x0094	各位域不同
EFUSE_DAC_CONF_REG	eFuse 烧写电压控制寄存器	0x0108	R/W
EFUSE_RD_TIM_CONF_REG	eFuse 读取时序参数配置寄存器	0x010C	R/W
EFUSE_WR_TIM_CONF1_REG	eFuse 烧写时序参数配置寄存器 1	0x0114	R/W
EFUSE_WR_TIM_CONF2_REG	eFuse 烧写时序参数配置寄存器 2	0x0118	R/W
状态寄存器			
EFUSE_STATUS_REG	eFuse 状态寄存器	0x0090	RO
中断寄存器			
EFUSE_INT_RAW_REG	eFuse 原始中断寄存器	0x0098	R/ WTC/ SS
EFUSE_INT_ST_REG	eFuse 中断状态寄存器	0x009C	RO
EFUSE_INT_ENA_REG	eFuse 中断使能寄存器	0x0100	R/W
EFUSE_INT_CLR_REG	eFuse 中断清除寄存器	0x0104	WT
版本寄存器			
EFUSE_DATE_REG	eFuse 版本控制寄存器	0x01FC	R/W

4.5 寄存器

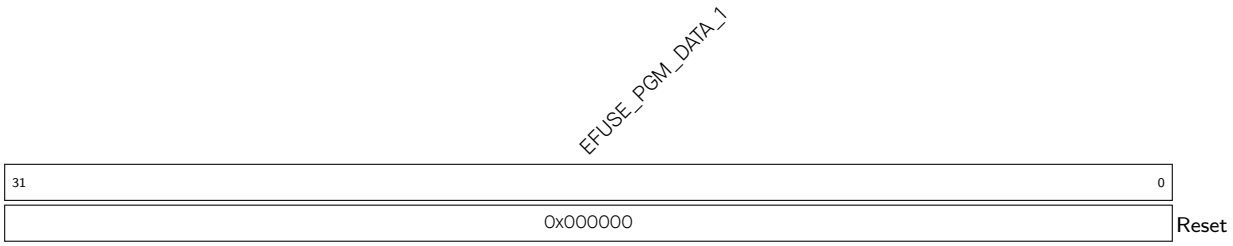
本小节的所有地址均为相对于 eFuse 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 4.1. EFUSE_PGM_DATA0_REG (0x0000)



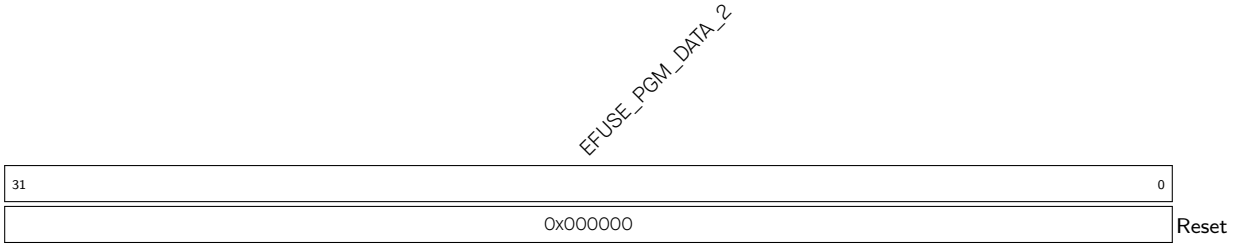
EFUSE_PGM_DATA_0 配置第 0 个待烧写的 32 位数据。(R/W)

Register 4.2. EFUSE_PGM_DATA1_REG (0x0004)



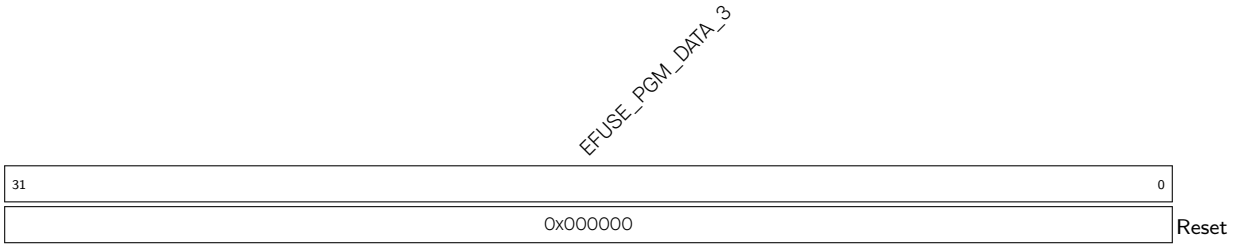
EFUSE_PGM_DATA_1 配置第 1 个待烧写的 32 位数据。(R/W)

Register 4.3. EFUSE_PGM_DATA2_REG (0x0008)



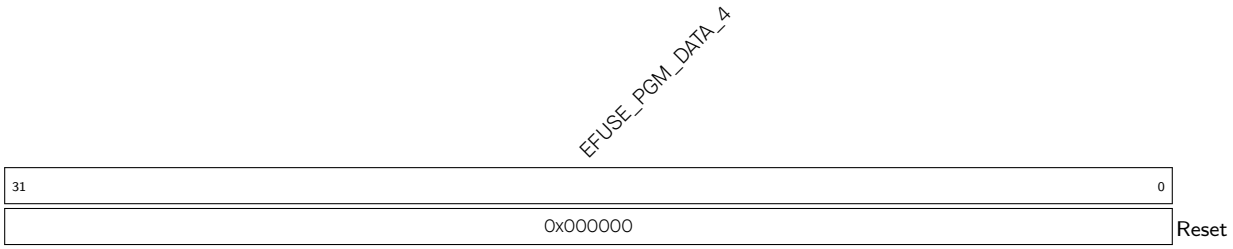
EFUSE_PGM_DATA_2 配置第 2 个待烧写的 32 位数据。(R/W)

Register 4.4. EFUSE_PGM_DATA3_REG (0x000C)



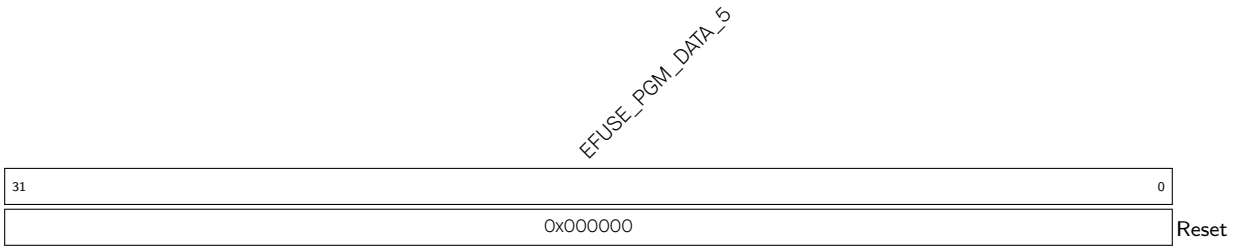
EFUSE_PGM_DATA_3 配置第 3 个待烧写的 32 位数据。(R/W)

Register 4.5. EFUSE_PGM_DATA4_REG (0x0010)



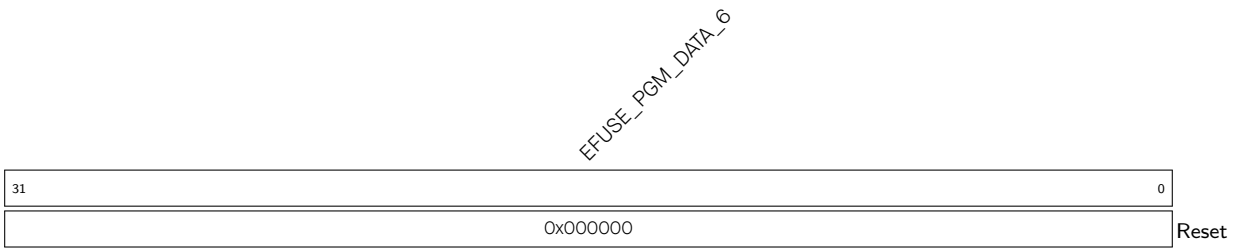
EFUSE_PGM_DATA_4 配置第 4 个待烧写的 32 位数据。(R/W)

Register 4.6. EFUSE_PGM_DATA5_REG (0x0014)



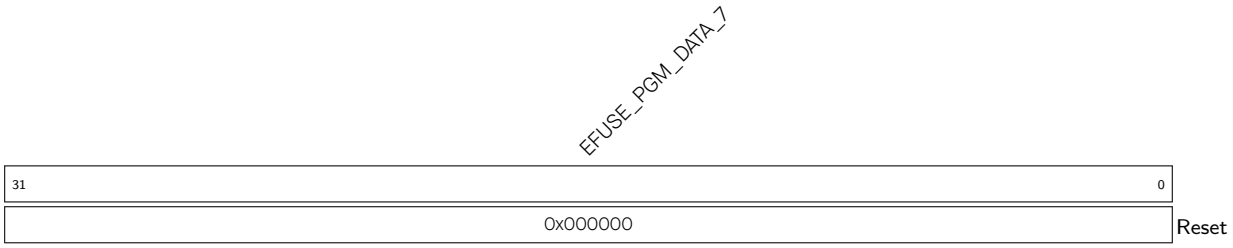
EFUSE_PGM_DATA_5 配置第 5 个待烧写的 32 位数据。(R/W)

Register 4.7. EFUSE_PGM_DATA6_REG (0x0018)



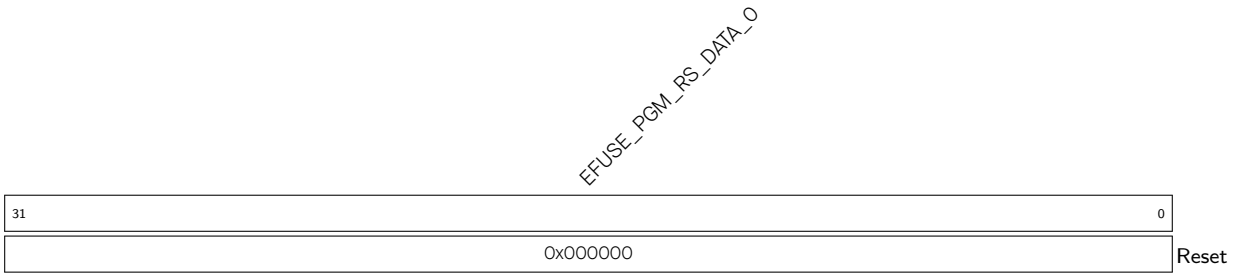
EFUSE_PGM_DATA_6 配置第 6 个待烧写的 32 位数据。(R/W)

Register 4.8. EFUSE_PGM_DATA7_REG (0x001C)



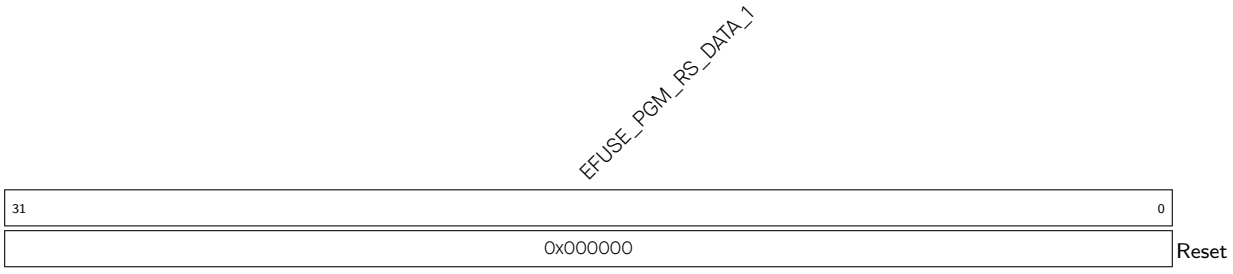
EFUSE_PGM_DATA_7 配置第 7 个待烧写的 32 位数据。(R/W)

Register 4.9. EFUSE_PGM_CHECK_VALUE0_REG (0x0020)



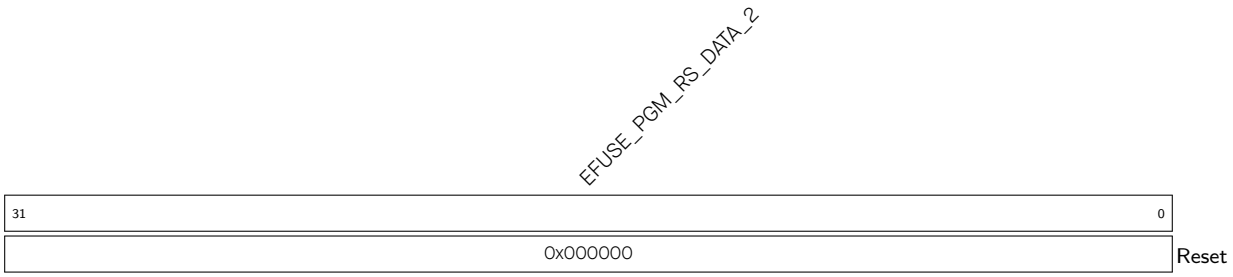
EFUSE_PGM_RS_DATA_0 配置第 0 个待烧写的 32 位 RS 码。(R/W)

Register 4.10. EFUSE_PGM_CHECK_VALUE1_REG (0x0024)



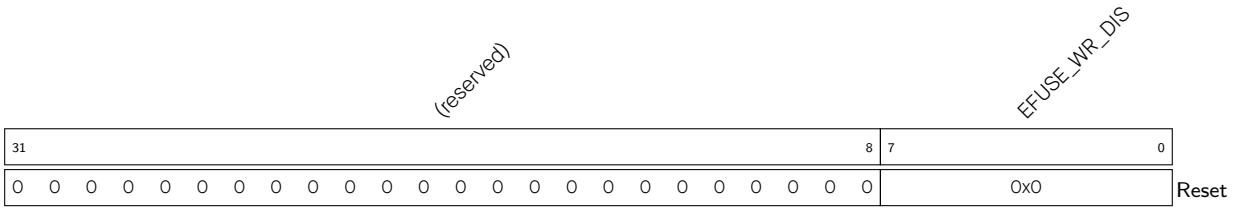
EFUSE_PGM_RS_DATA_1 配置第 1 个待烧写的 32 位 RS 码。(R/W)

Register 4.11. EFUSE_PGM_CHECK_VALUE2_REG (0x0028)



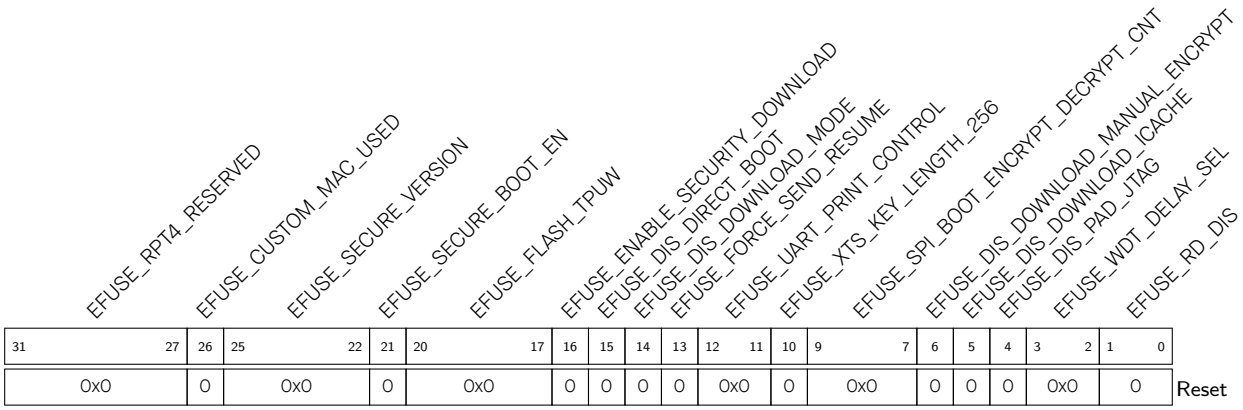
EFUSE_PGM_RS_DATA_2 配置第 2 个待烧写的 32 位 RS 码。(R/W)

Register 4.12. EFUSE_RD_WR_DIS_REG (0x002C)



EFUSE_WR_DIS 表示是否禁用 eFuse 烧写。1: 禁用。0: 启用。(RO)

Register 4.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)



EFUSE_RD_DIS 表示是否禁读取 BLOCK3 高/低 128 位。1: 禁用。0: 启用。(RO)

EFUSE_WDT_DELAY_SEL 表示 RTC 看门狗超时阈值。单位为慢速时钟周期。0：40000；1：80000；2：160000；3：320000。(RO)

EFUSE_DIS_PAD_JTAG 表示是否永久禁用 JTAG 管脚。1: 禁用。0: 启用。(RO)

EFUSE_DIS_DOWNLOAD_ICACHE 表示是否在下载模式下关闭 iCache。1: 禁用。0: 启用。(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 表示是否在 download boot 模式下禁用手动 flash 加密。1: 禁用。0: 启用。(RO)

EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT 表示是否启用 SPI 启动加密/解密。奇数个 1：启用。偶数个 1：禁用。(RO)

EFUSE_XTS_KEY_LENGTH_256 表示 XTS_AES 密钥长度。1：BLOCK3 所有的 256 位。0：BLOCK3 的低 128 位。(RO)

EFUSE_UART_PRINT_CONTROL 表示 UART 启动信息输出模式。2'b00：强制打印；2'b01：由 GPIO8 控制，低电平打印；2'b10：由 GPIO8 控制，高电平打印；2'b11：强制关闭打印。(RO)

EFUSE_FORCE_SEND_RESUME 表示是否强制 ROM 代码在 SPI 启动期间发送 SPI flash 恢复指令。1：发送。0：不发送。(RO)

EFUSE_DIS_DOWNLOAD_MODE 表示是否禁用所有的 Download 模式 (boot_mode[3:0] = 0, 1, 2, 4, 5, 6, 7)。1: 禁用。0: 启用。(RO)

EFUSE_DIS_DIRECT_BOOT 表示是否禁用 Direct_boot 模式。1: 禁用。0: 启用。(RO)

EFUSE_ENABLE_SECURITY_DOWNLOAD 表示是否启用 UART 安全下载模式（仅读写 flash）1：启用。0：禁用。(RO)

见下页

Register 4.13. EFUSE_RD_REPEAT_DATA0_REG (0x0030)

接上页

EFUSE_FLASH_TPUW 表示 SoC 上电后 flash 启动的延迟时间。单位：毫秒。如果该值小于 15，延迟时间为该值。如果该值大于或等于 15，延迟时间为 30 毫秒。(RO)

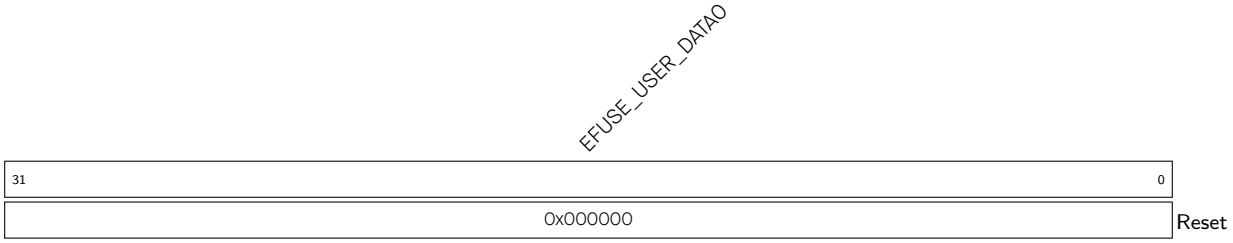
EFUSE_SECURE_BOOT_EN 表示是否启用安全启动。1：启用。0：禁用。(RO)

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

EFUSE_CUSTOM_MAC_USED 表示是否使用用户自定义的 MAC。1：启用。0：禁用。(RO)

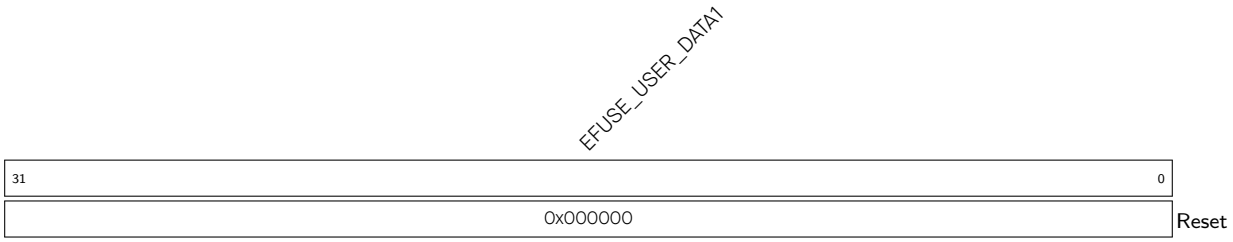
EFUSE_RPT4_RESERVED 保留（采用 4 备份方式）。(RO)

Register 4.14. EFUSE_RD_BLK1_DATA0_REG (0x0034)



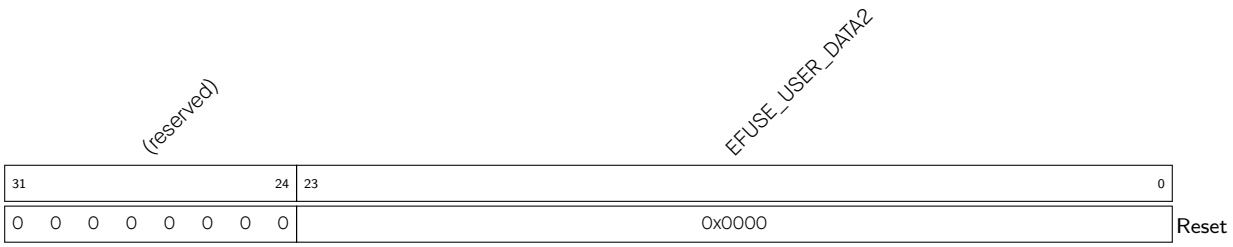
EFUSE_USER_DATA0 存储第 0 个 32 位用户数据。(RO)

Register 4.15. EFUSE_RD_BLK1_DATA1_REG (0x0038)



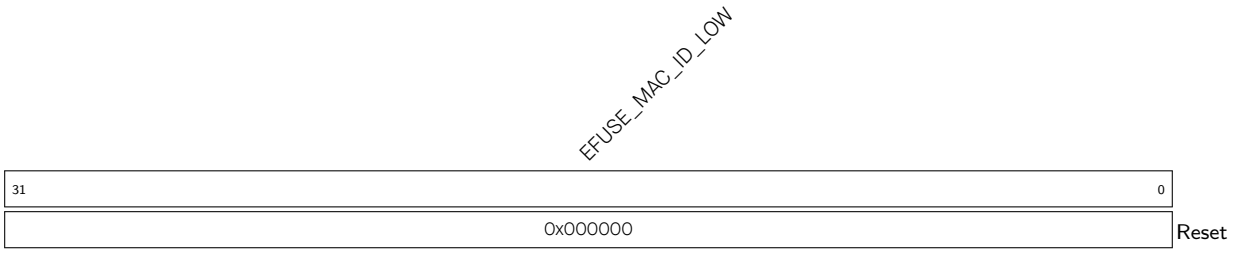
EFUSE_USER_DATA1 存储第 1 个 32 位用户数据。(RO)

Register 4.16. EFUSE_RD_BLK1_DATA2_REG (0x003C)



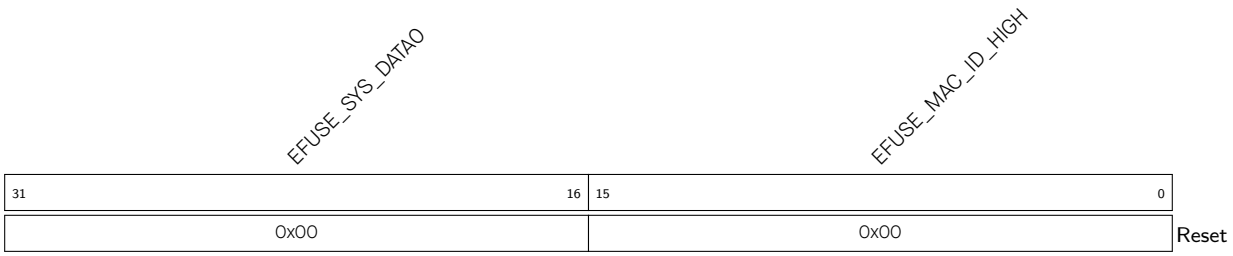
EFUSE_USER_DATA2 存储 [64:87] 位用户数据。(RO)

Register 4.17. EFUSE_RD_BLK2_DATA0_REG (0x0040)



EFUSE_MAC_ID_LOW 存储低 32 位 MAC ID。(RO)

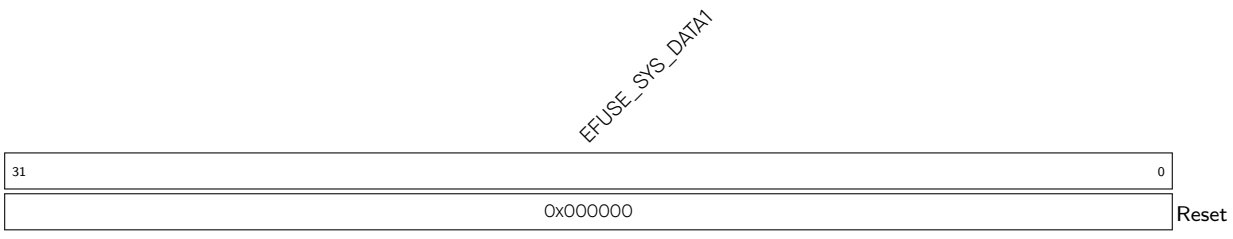
Register 4.18. EFUSE_RD_BLK2_DATA1_REG (0x0044)



EFUSE_MAC_ID_HIGH 存储高 16 位 MAC ID。(RO)

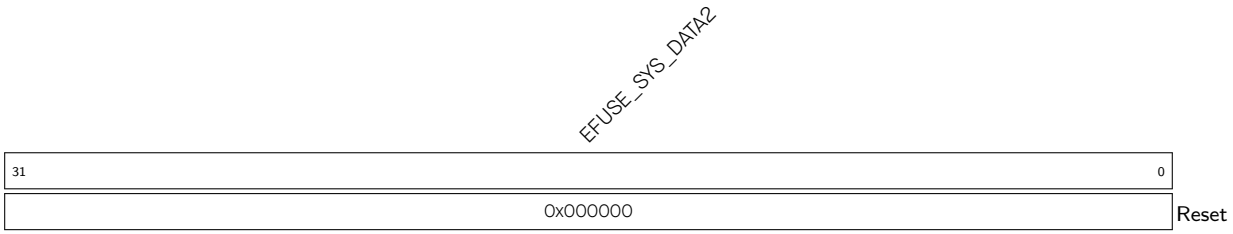
EFUSE_SYS_DATA0 储存第 0 个 16 位系统数据。(RO)

Register 4.19. EFUSE_RD_BLK2_DATA2_REG (0x0048)



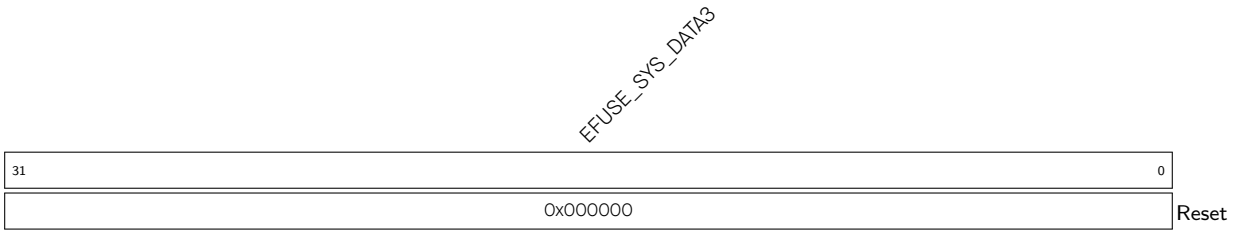
EFUSE_SYS_DATA1 储存第 0 个 32 位系统数据。(RO)

Register 4.20. EFUSE_RD_BLK2_DATA3_REG (0x004C)



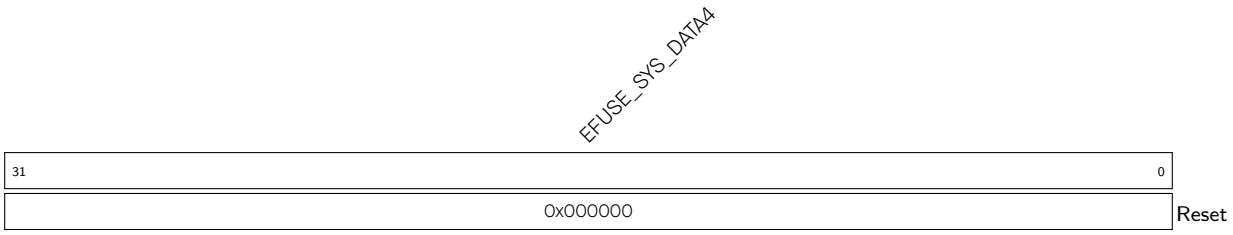
EFUSE_SYS_DATA2 储存第 1 个 32 位系统数据。(RO)

Register 4.21. EFUSE_RD_BLK2_DATA4_REG (0x0050)



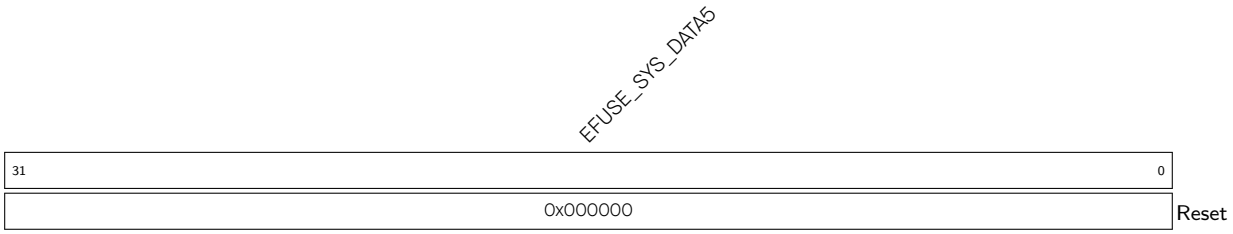
EFUSE_SYS_DATA3 储存第 2 个 32 位系统数据。(RO)

Register 4.22. EFUSE_RD_BLK2_DATA5_REG (0x0054)



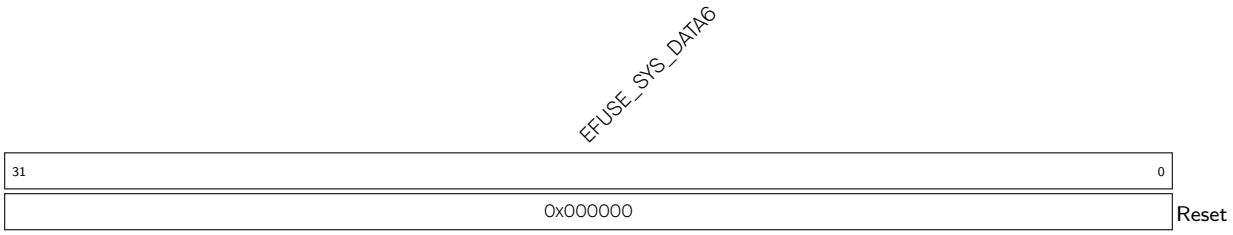
EFUSE_SYS_DATA4 储存第 3 个 32 位系统数据。(RO)

Register 4.23. EFUSE_RD_BLK2_DATA6_REG (0x0058)



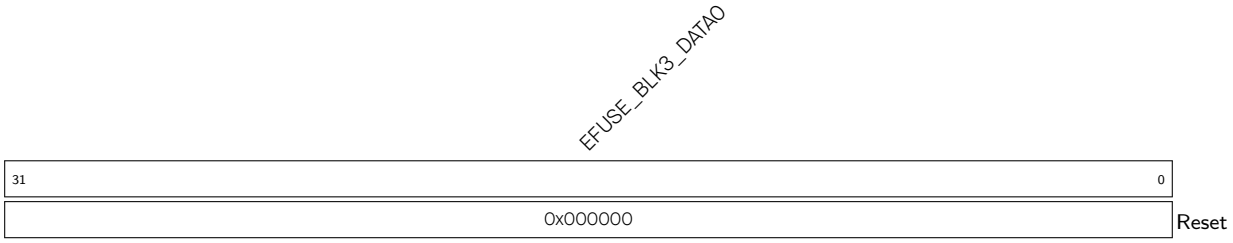
EFUSE_SYS_DATA5 储存第 4 个 32 位系统数据。(RO)

Register 4.24. EFUSE_RD_BLK2_DATA7_REG (0x005C)



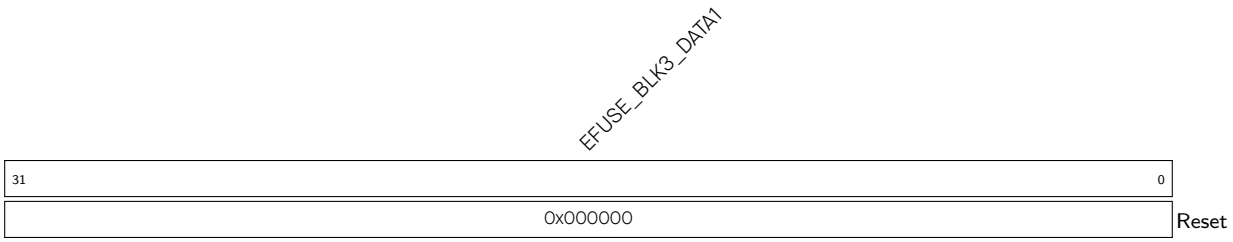
EFUSE_SYS_DATA6 储存第 5 个 32 位系统数据。(RO)

Register 4.25. EFUSE_RD_BLK3_DATA0_REG (0x0060)



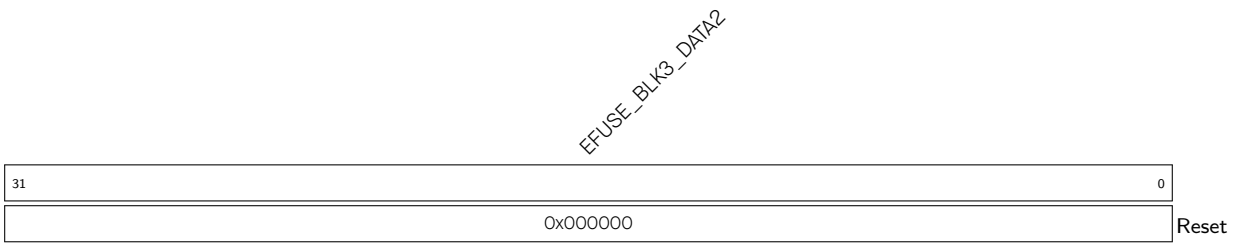
EFUSE_BLK3_DATA0 存储 BLOCK3 第 0 个 32 位。(RO)

Register 4.26. EFUSE_RD_BLK3_DATA1_REG (0x0064)



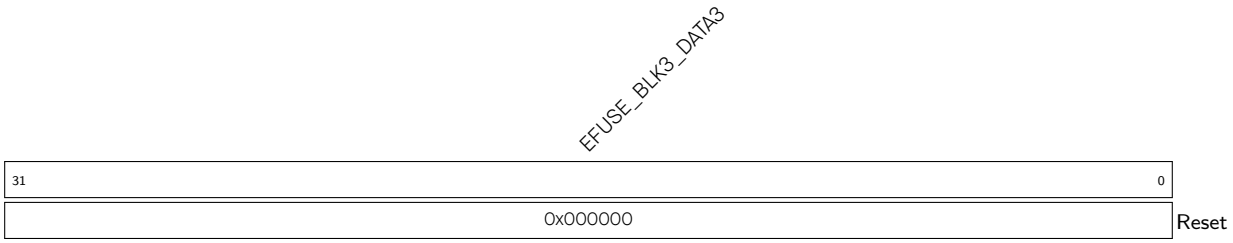
EFUSE_BLK3_DATA1 存储 BLOCK3 第 1 个 32 位。(RO)

Register 4.27. EFUSE_RD_BLK3_DATA2_REG (0x0068)



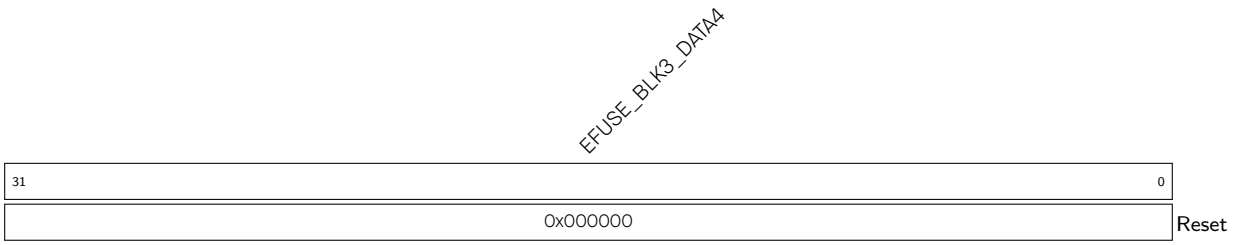
EFUSE_BLK3_DATA2 存储 BLOCK3 第 2 个 32 位。(RO)

Register 4.28. EFUSE_RD_BLK3_DATA3_REG (0x006C)



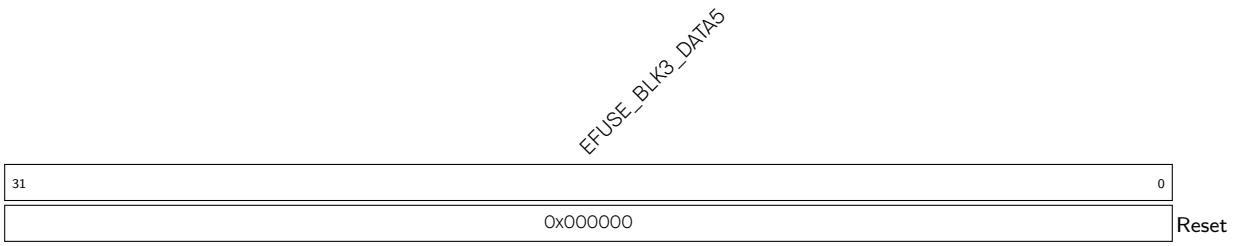
EFUSE_BLK3_DATA3 存储 BLOCK3 第 3 个 32 位。(RO)

Register 4.29. EFUSE_RD_BLK3_DATA4_REG (0x0070)



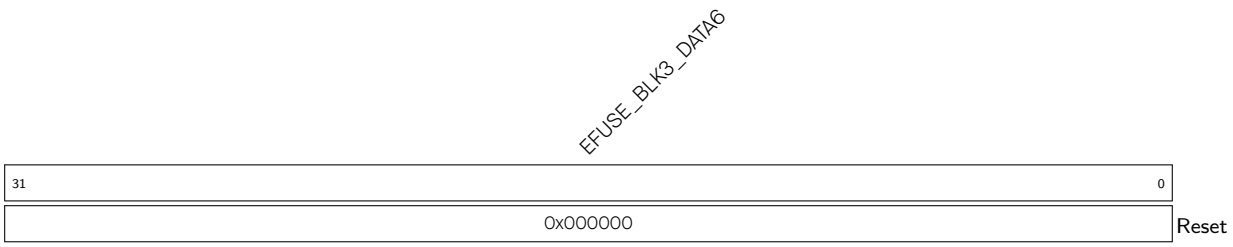
EFUSE_BLK3_DATA4 存储 BLOCK3 第 4 个 32 位。(RO)

Register 4.30. EFUSE_RD_BLK3_DATA5_REG (0x0074)



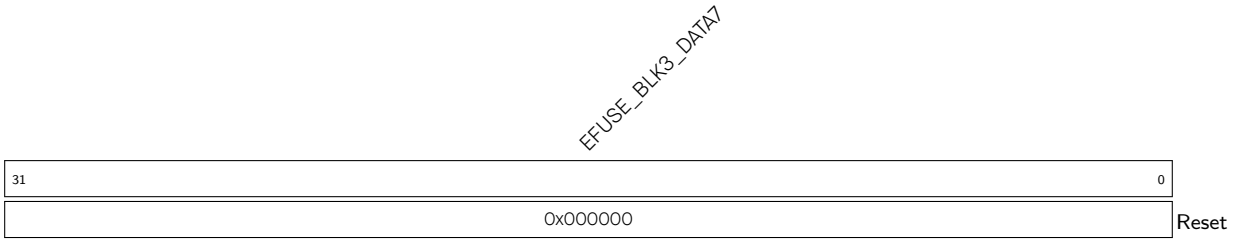
EFUSE_BLK3_DATA5 存储 BLOCK3 第 5 个 32 位。(RO)

Register 4.31. EFUSE_RD_BLK3_DATA6_REG (0x0078)



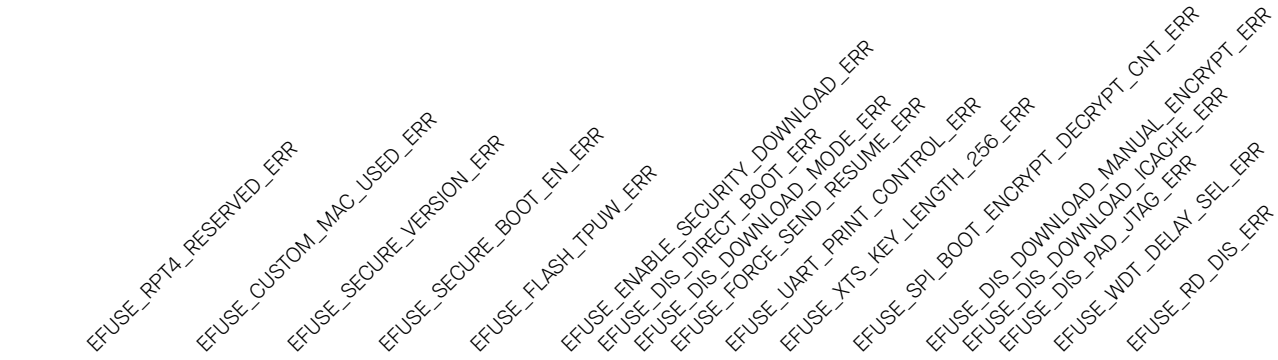
EFUSE_BLK3_DATA6 存储 BLOCK3 第 6 个 32 位。(RO)

Register 4.32. EFUSE_RD_BLK3_DATA7_REG (0x007C)



EFUSE_BLK3_DATA7 存储 BLOCK3 第 7 个 32 位。(RO)

Register 4.33. EFUSE_RD_REPEAT_ERR_REG (0x0080)



EFUSE_RD_DIS_ERR 若 RD_DIS_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_WDT_DELAY_SEL_ERR 若 WDT_DELAY_SEL_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_DIS_PAD_JTAG_ERR 若 DIS_PAD_JTAG_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_DIS_DOWNLOAD_ICACHE_ERR 若 DIS_DOWN_ICACHE_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR 若 DIS_DOWNLOAD_MANUAL_ENCRYPT_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT_ERR 若 SPI_BOOT_ENCRYPT_DECRYPT_CNT_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_XTS_KEY_LENGTH_256_ERR 若 XTS_KEY_LENGTH_256_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_UART_PRINT_CONTROL_ERR 若 UART_PRINT_CONTROL_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_FORCE_SEND_RESUME_ERR 若 FORCE_SEND_RESUME_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_DIS_DOWNLOAD_MODE_ERR 若 DIS_DOWNLOAD_MODE_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_DIS_DIRECT_BOOT_ERR 若 DIS_DIRECT_BOOT_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

见下页……

Register 4.33. EFUSE_RD_REPEAT_ERR_REG (0x0080)

接上页……

EFUSE_ENABLE_SECURITY_DOWNLOAD_ERR 若 ENABLE_SECURITY_DOWNLOAD_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_FLASH_TPUW_ERR 若 FLASH_TPUW_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

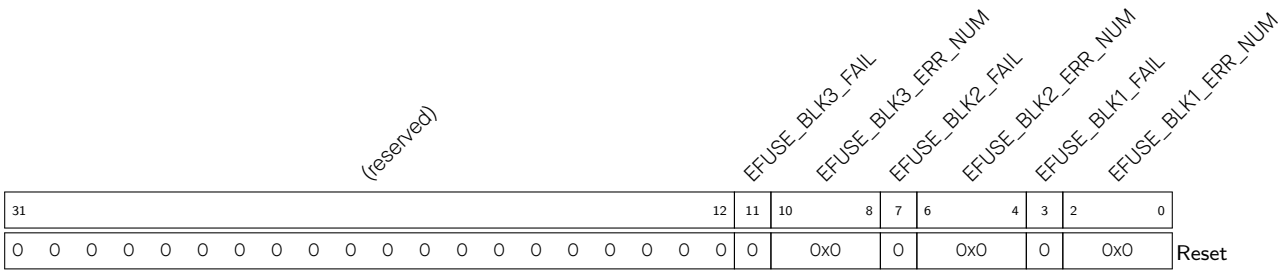
EFUSE_SECURE_BOOT_EN_ERR 若 SECURE_BOOT_EN_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_SECURE_VERSION_ERR 若 SECURE_VERSION_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_CUSTOM_MAC_USED_ERR 若 CUSTOM_MAC_USED_ERR 中的任何位为 1，则表示该参数烧写错误。(RO)

EFUSE_RPT4_RESERVED_ERR 保留。(RO)

Register 4.34. EFUSE_RD_RS_ERR_REG (0x0084)



EFUSE_BLK1_ERR_NUM 该信号的值表示 BLOCK1 中的错误字节数。(RO)

EFUSE_BLK1_FAIL 0：无烧写错误，BLOCK1 数据可靠。1：烧写用户数据失败，错误字节数大于 6。(RO)

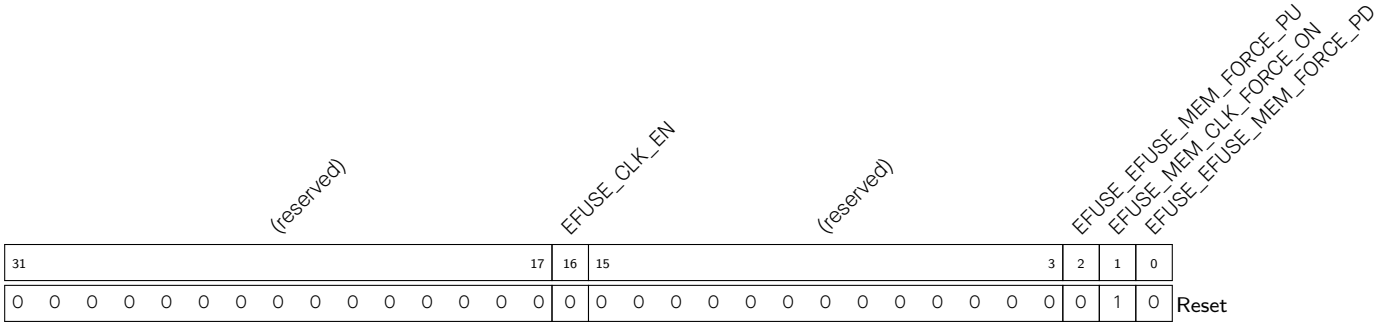
EFUSE_BLK2_ERR_NUM 该信号的值表示 BLOCK2 中的错误字节数。(RO)

EFUSE_BLK2_FAIL 0：无烧写错误，BLOCK2 数据可靠。1：烧写用户数据失败，错误字节数大于 6。(RO)

EFUSE_BLK3_ERR_NUM 该信号的值表示 BLOCK3 中的错误字节数。(RO)

EFUSE_BLK3_FAIL 0：无烧写错误，BLOCK3 数据可靠。1：烧写用户数据失败，错误字节数大于 6。(RO)

Register 4.35. EFUSE_CLK_REG (0x0088)



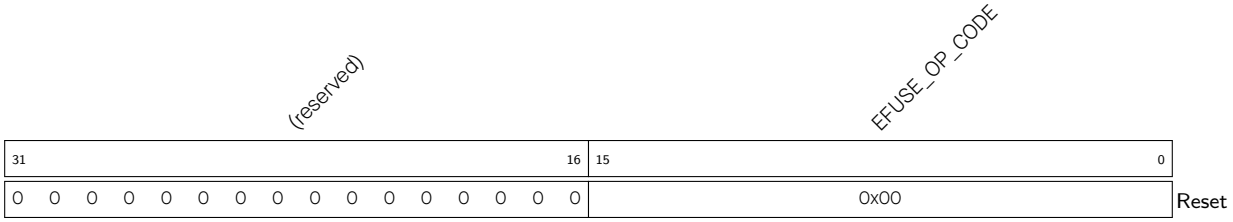
EFUSE_EFUSE_MEM_FORCE_PD 置位可强制 eFuse 控制器的 SRAM 进入省电模式。(R/W)

EFUSE_MEM_CLK_FORCE_ON 置位可强制激活 eFuse 控制器的 SRAM 的时钟信号。(R/W)

EFUSE_EFUSE_MEM_FORCE_PU 置位可强制 eFuse 控制器的 SRAM 进入工作模式。(R/W)

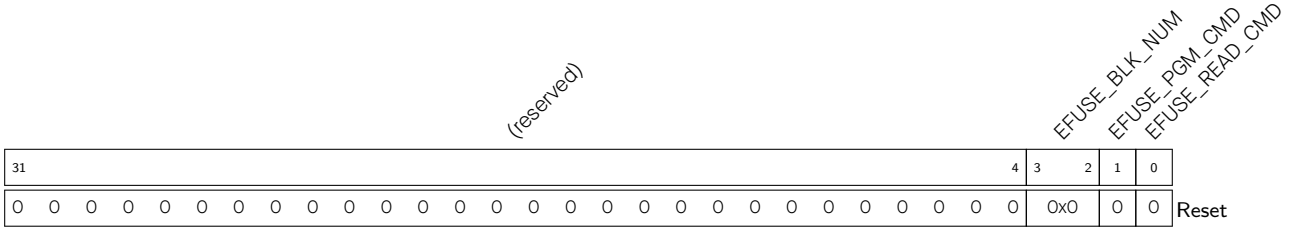
EFUSE_CLK_EN 置位可强制使能 eFuse 控制器的寄存器配置时钟信号。(R/W)

Register 4.36. EFUSE_CONF_REG (0x008C)



EFUSE_OP_CODE 0x5A5A：运行烧写指令；0x5AA5：运行读指令。(R/W)

Register 4.37. EFUSE_CMD_REG (0x0094)

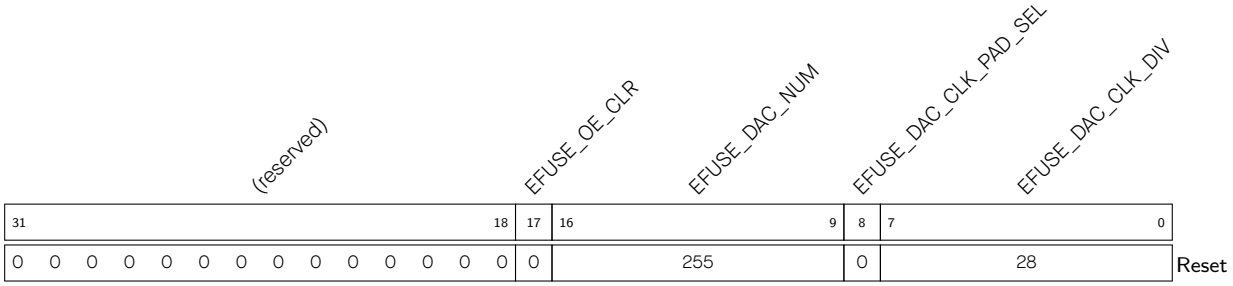


EFUSE_READ_CMD 置位发送读取指令。(R/W/SC)

EFUSE_PGM_CMD 置位可发送烧写指令。(R/W/SC)

EFUSE_BLK_NUM 待烧写的 BLOCK 的序列号。值 0-3 分别对应 BLOCK0-BLOCK3。(R/W)

Register 4.38. EFUSE_DAC_CONF_REG (0x0108)



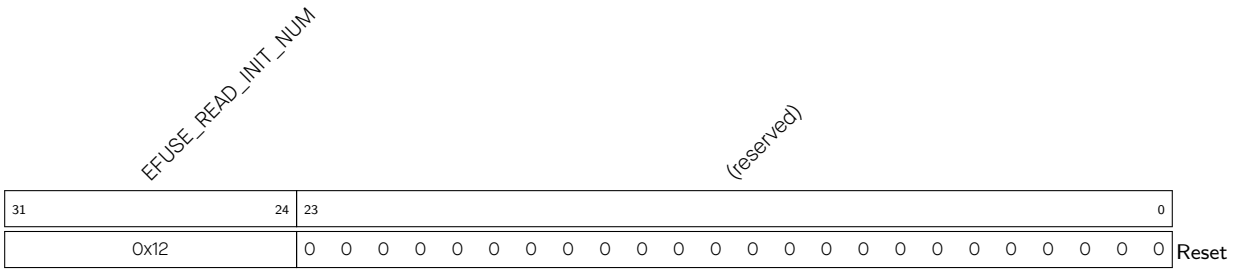
EFUSE_DAC_CLK_DIV 控制烧写电压爬升时钟的分频系数。(R/W)

EFUSE_DAC_CLK_PAD_SEL 无关项。(R/W)

EFUSE_DAC_NUM 烧写供电的上升周期。(R/W)

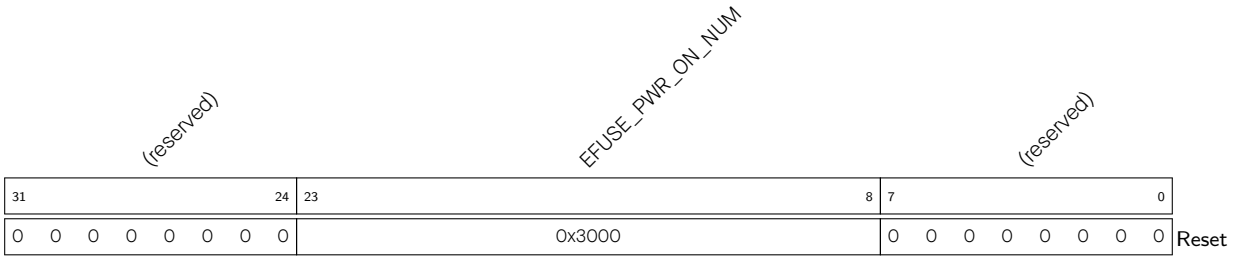
EFUSE_OE_CLR 降低烧写电压的供电能力。(R/W)

Register 4.39. EFUSE_RD_TIM_CONF_REG (0x010C)



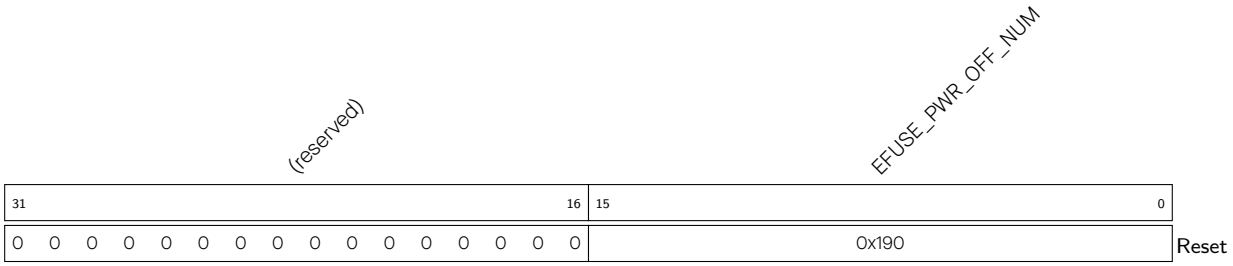
EFUSE_READ_INIT_NUM 配置读取 eFuse 存储器的等待时间。(R/W)

Register 4.40. EFUSE_WR_TIM_CONF1_REG (0x0114)



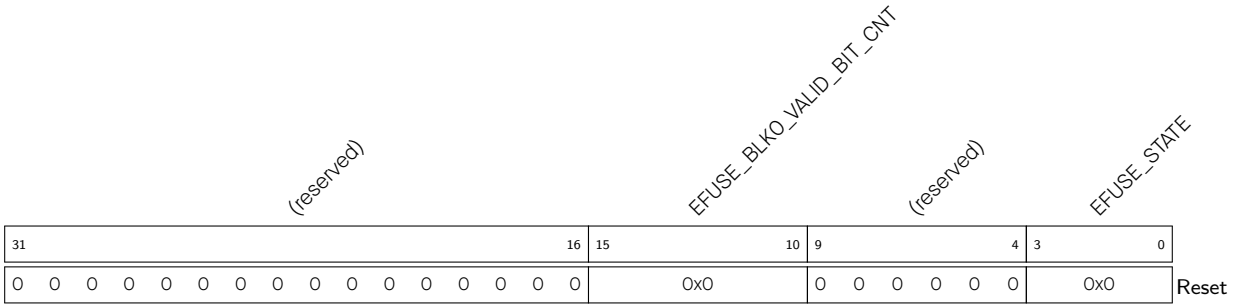
EFUSE_PWR_ON_NUM 配置 VDDQ 的上电时间。(R/W)

Register 4.41. EFUSE_WR_TIM_CONF2_REG (0x0118)



EFUSE_PWR_OFF_NUM 配置 VDDQ 的掉电时间。(R/W)

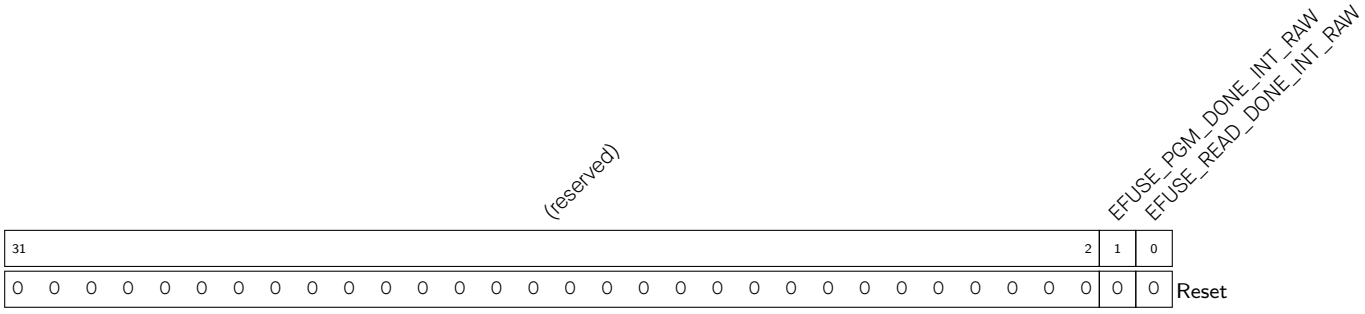
Register 4.42. EFUSE_STATUS_REG (0x0090)



EFUSE_STATE 表示 eFuse 控制器状态机的状态。(RO)

EFUSE_BLK0_VALID_BIT_CNT 表示 BLOCK0 中值为“1”的比特个数。(RO)

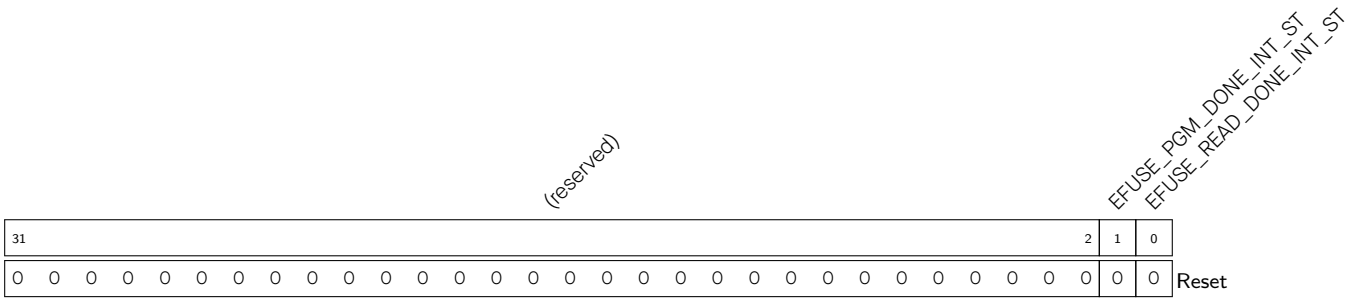
Register 4.43. EFUSE_INT_RAW_REG (0x0098)



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

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

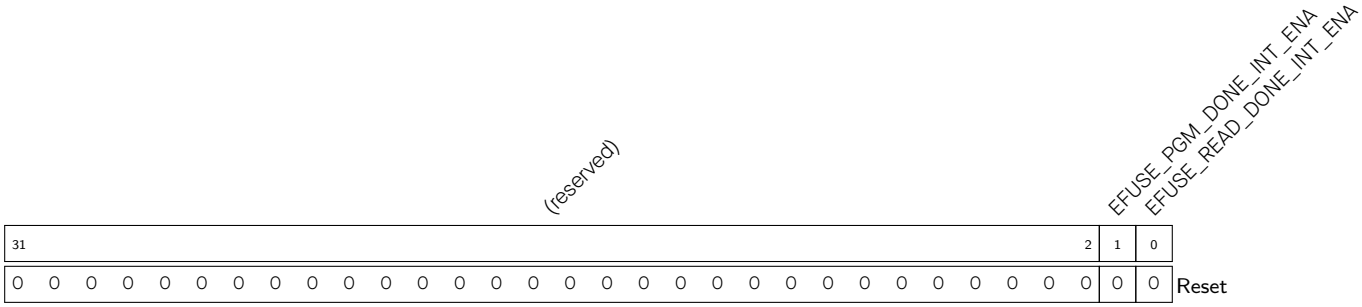
Register 4.44. EFUSE_INT_ST_REG (0x009C)



EFUSE_READ_DONE_INT_ST 读取完成中断的状态位。(RO)

EFUSE_PGM_DONE_INT_ST 烧写完成中断的状态位。(RO)

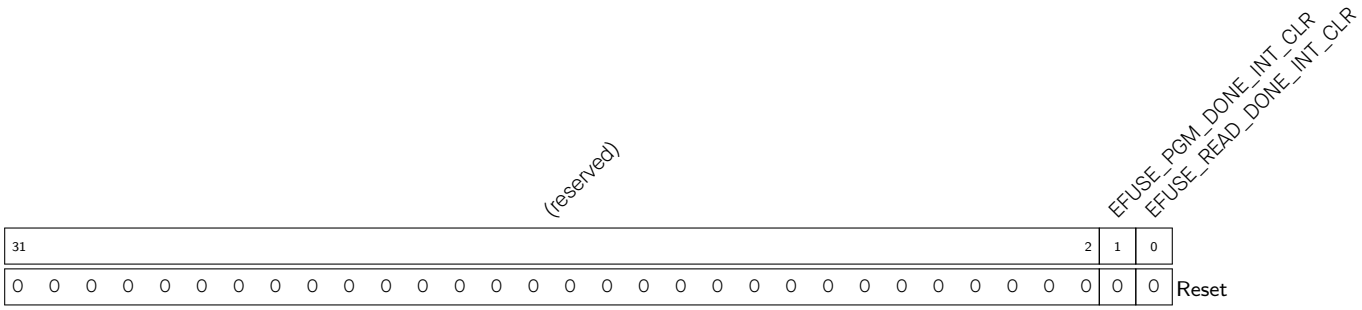
Register 4.45. EFUSE_INT_ENA_REG (0x0100)



EFUSE_READ_DONE_INT_ENA 读取完成中断的使能位。(R/W)

EFUSE_PGM_DONE_INT_ENA 烧写完成中断的使能位。(R/W)

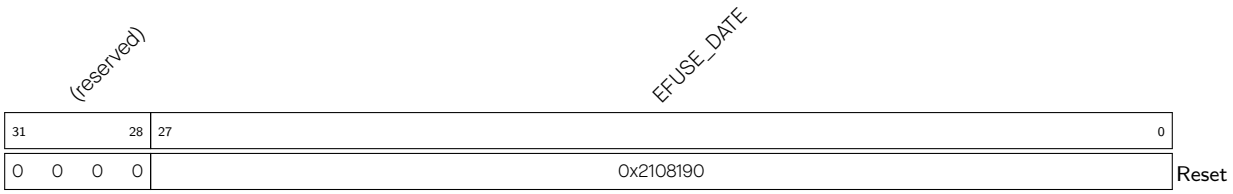
Register 4.46. EFUSE_INT_CLR_REG (0x0104)



EFUSE_READ_DONE_INT_CLR 读取完成中断的清除位。(WT)

EFUSE_PGM_DONE_INT_CLR 烧写完成中断的清除位。(WT)

Register 4.47. EFUSE_DATE_REG (0x01FC)



EFUSE_DATE 版本控制寄存器。(R/W)

5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)

5.1 概述

ESP8684 芯片有 21 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。

说明：

- 上述 21 个 GPIO 管脚的编号为：0 ~ 20；
- 如果选用的芯片版本内置了 SiP flash，则有 7 个 GPIO 管脚专用于连接 SiP flash，编号为：11 ~ 17，不可用作他用；用户可配置使用其它剩余的 14 个 GPIO 管脚，编号为：0 ~ 10、18 ~ 20。

5.2 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；
- 33 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 61 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 APB 时钟总线；
- 支持输入信号滤波；
- 支持 GPIO 简单输入输出。

IO MUX 具有如下特性：

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

5.3 结构概览

图 5-1 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。

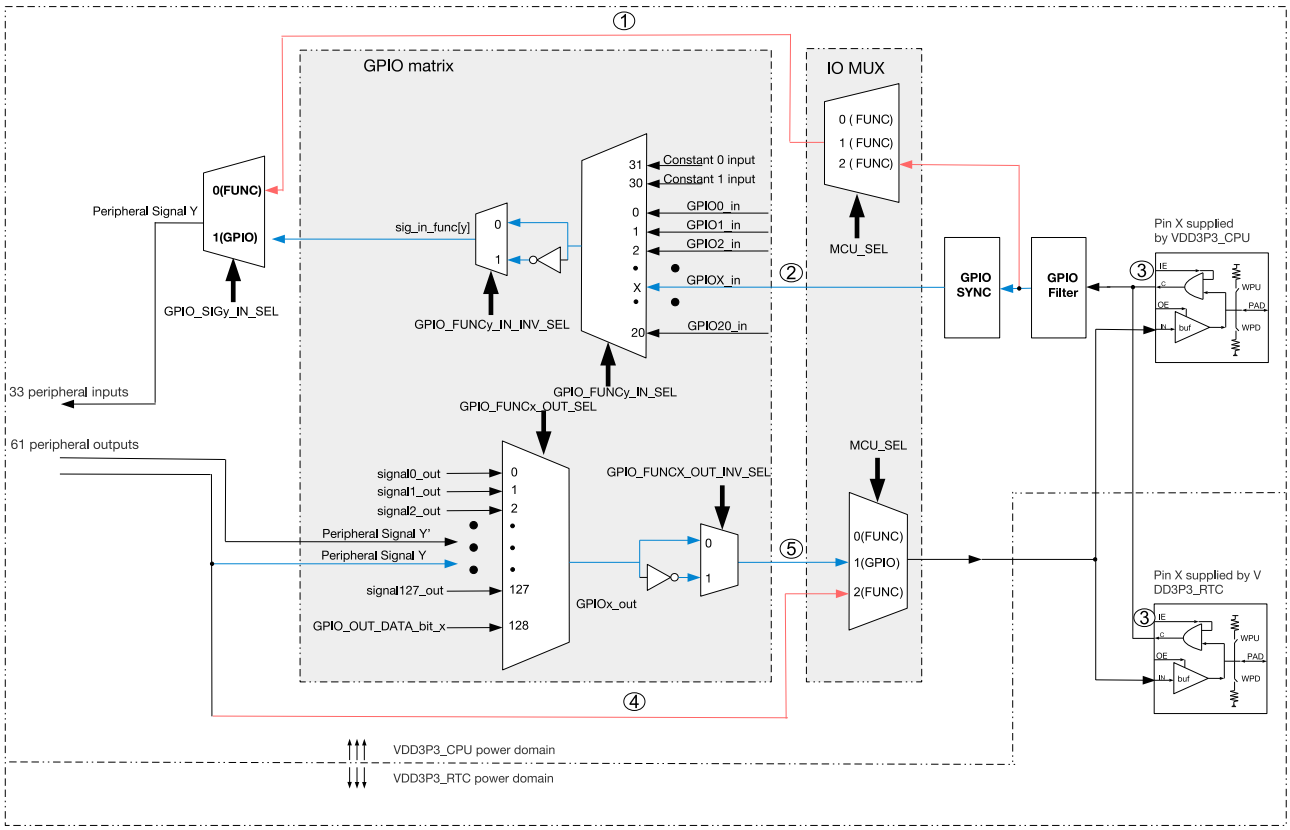


图 5-1. IO MUX 和 GPIO 交换矩阵框图

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 5-2 “信号可经由 IO MUX 直接输入”一栏中被标为“yes”。剩余其它信号只能通过 GPIO 交换矩阵连接至外设；
2. ESP8684 共有 21 个 GPIO 管脚，因此从 GPIO SYNC 进入到 GPIO 交换矩阵的输入共有 21 个。注意：在内置 SiP flash 的芯片版本中，从 GPIO SYNC 进入到 GPIO 交换矩阵的输入只有 14 个；
3. 位于 VDD3P3_CPU 电源域和 VDD3P3_RTC 电源域的管脚由 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 5-2 “信号可经由 IO MUX 直接输出”一栏中被标为“yes”。剩余其它信号只能通过 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 21 个，对应 GPIO *X*：0 ~ 20。注意：在内置 SiP flash 的芯片版本中，从 GPIO 交换矩阵到 IO MUX 的输出只有 14 个，对应 GPIO *X*：0 ~ 10、18 ~ 20。

图 5-2 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。21 个 GPIO 管脚均采用这一结构，且由 IE、OE、WPU 和 WPD 信号控制。

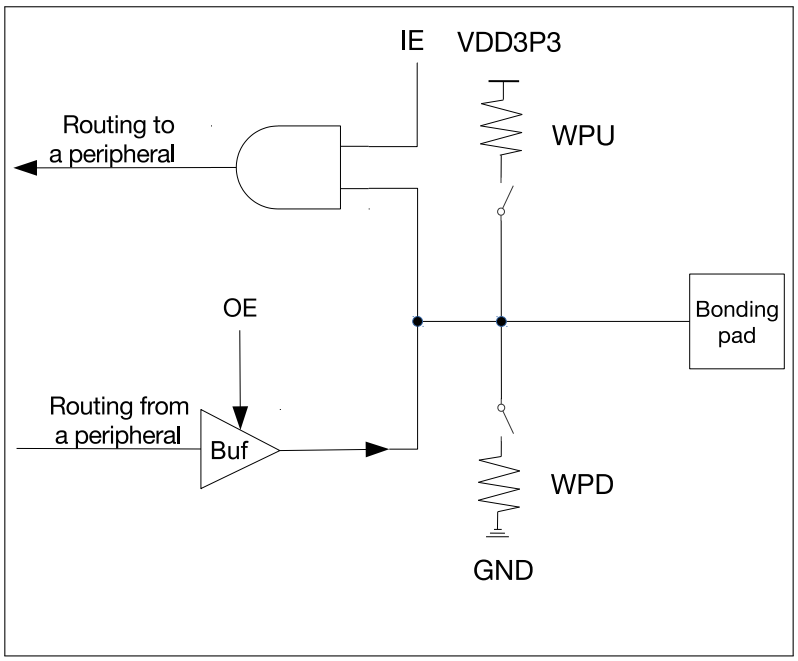


图 5-2. 焊盘内部结构

说明：

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

5.4 通过 GPIO 交换矩阵的外设输入

5.4.1 概述

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 21 个 GPIO (0 ~ 20) 中获取外部输入信号，见交换矩阵表格 5-2。并需要配置外设输入选择通过 GPIO 交换矩阵接收输入信号。

5.4.2 信号同步

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

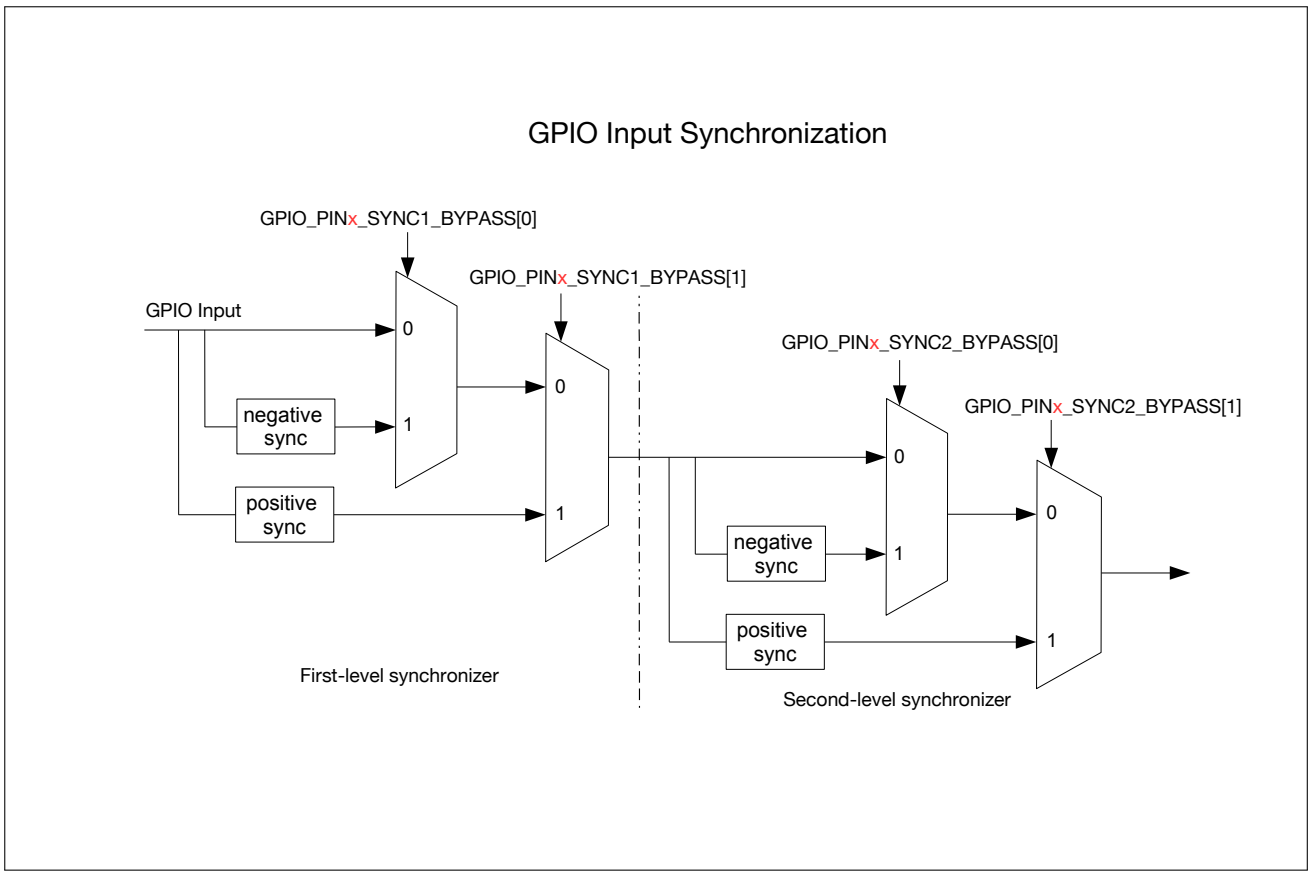


图 5-3. GPIO 输入经 APB 时钟上升沿或下降沿同步

GPIO SYNC 模块的功能如图 5-3 所示。其中，negative sync 为 GPIO 输入经过 APB 时钟的下降沿同步，positive sync 为 GPIO 输入经过 APB 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 3。

5.4.3 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*n*_FILTER_EN 使能 GPIO 管脚的输入信号滤波功能，如图 5-4 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。

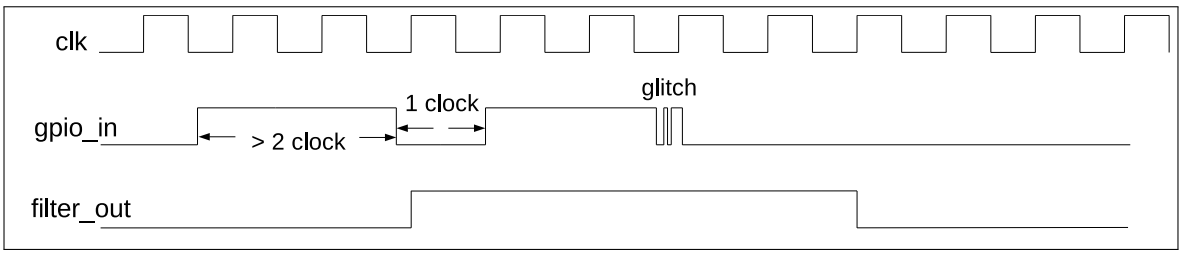


图 5-4. GPIO 输入信号滤波时序图

3. 同步 GPIO 输入信号。配置 GPIO 管脚 *X* 的 GPIO_PIN*x*_REG 来同步 GPIO 输入信号，过程如下：
   - 如图 5-3 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 5-3 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。
4. 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 UART0 DSR 输入信号 [3]（U0DSR_in，信号索引号 8）绑定到 GPIO7，请按照以下步骤操作。注意，GPIO7 也叫做 MTDO 管脚。

1. 置位 GPIO_FUNC8_IN_SEL_CFG_REG 寄存器的 GPIO_SIG8_IN_SEL 位，使能通过 GPIO 交换矩阵接收外部输入信号；
2. 配置 GPIO_FUNC8_IN_SEL_CFG_REG 寄存器中的 GPIO_FUNC8_IN_SEL 为 7，即选择管脚 GPIO7；
3. 置位 IO_MUX_GPIO7_REG 寄存器中 IO_MUX_GPIO7_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 0x1F，则输入信号恒为 0；
   - 设置 GPIO_FUNC*y*_IN_SEL 为 0x1E，则输入信号恒为 1。

5.4.4 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。

5.5 通过 GPIO 交换矩阵的外设输出

5.5.1 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 5-2 中“输出信号”一栏所列出的信号）输出到 21 个 GPIO (0 ~ 20) 管脚。注意：在内置 SiP flash 的芯片版本中，只可配置 GPIO 交换矩阵将外设信号输出到 14 个 GPIO 管脚，即 GPIO 0 ~ 10、GPIO 18 ~ 20。

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 5-2 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）。

5.5.2 功能描述

如图 5-1 所示，对于信号输出，61 个输出信号（即在表 5-2 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 5-2 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 Function 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。

5.5.3 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[20] 对应 GPIO0 ~ GPIO20，GPIO_OUT_REG[24:21] 无效。
- 推荐使用相应的 W1TS 和 W1TC 寄存器，例如 GPIO_OUT_W1TS/GPIO_OUT_W1TC 来置位/清零 GPIO_OUT_REG。

5.6 IO MUX 的直接输入输出功能

5.6.1 概述

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

这样比使用 GPIO 交换矩阵的灵活度要低，即每个 GPIO 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。

5.6.2 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寄存器：

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 5.12 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。

5.7 GPIO 管脚的模拟功能

ESP8684 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 5-4 列出了 ESP8684 管脚的模拟功能。

5.8 Light-sleep 模式管脚功能

当 ESP8684 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_*n*_REG 寄存器中 IO_MUX_SLP_SEL 位置为 1，芯片处于 Light-sleep 模式下将由另一组不同的寄存器控制管脚。

表 5-1. 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_SLP_SEL = 0	Light-sleep 模式 AND IO_MUX_SLP_SEL = 1
输出驱动强度	IO_MUX_FUN_DRV	IO_MUX_MCU_DRV
上拉电阻	IO_MUX_FUN_WPU	IO_MUX_MCU_WPU
下拉电阻	IO_MUX_FUN_WPD	IO_MUX_MCU_WPD
输出使能	由 GPIO 交换矩阵的 OEN_SEL 位控制 *	IO_MUX_MCU_OE

说明：

如果 IO_MUX_SLP_SEL 置为 0，则芯片在正常工作和 Light-sleep 模式下，管脚的功能一样。此时，具体的输出使能配置请参考 5.5.2 章节。

5.9 GPIO 管脚的 Hold 特性

每个 GPIO 管脚（包括 RTC 管脚 GPIO0 ~ GPIO5）都有单独的 Hold 功能，由 RTC 寄存器控制。管脚的 Hold 功能被置上后，管脚在置上 Hold 那一刻的状态被强制保持，无论内部信号如何变化，修改 IO MUX 配置或者 GPIO 配置，都不会改变管脚的状态。应用如果希望在看门狗超时触发内核复位时或 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。

说明：

- 对于数字管脚 (GPIO6 ~ 20)，若要在 Deep-sleep 中保持管脚输入输出的状态值，需要在掉电之前将寄存器 RTC_CNTL_DIG_PAD_HOLD_REG 中的 RTC_CNTL_DIG_PAD_HOLD*[n]* 位置 1。在芯片被唤醒后，若要关闭 Hold 功能，可将寄存器 RTC_CNTL_DIG_PAD_HOLD*[n]* 设置为 0。
- 对于 RTC 管脚 (GPIO0 ~ 5)，管脚的输入输出值由寄存器 RTC_CNTL_RTC_PAD_HOLD_REG 中的相应位控制。用户可置位或清除相应位来实现 Hold 或 Unhold 管脚输入输出值。

5.10 GPIO 管脚供电和电源管理

5.10.1 GPIO 管脚供电

GPIO 管脚供电请参考《*ESP8684* 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 VDD3P3_RTC 域中的管脚 (GPIO0 ~ GPIO5) 可用于将芯片从 Deep-sleep 唤醒。

5.10.2 电源管理

ESP8684 的管脚可分为如下两种不同的电源域。

- VDD3P3_RTC：RTC 和 CPU 的输入电源
- VDD3P3_CPU：CPU 的输入电源

5.11 外设信号列表

表 5-2 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 5-2 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，则表示寄存器 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 5-2 “输入信号” 一栏中有名字的信号均为有效输入信号；
- 表 5-2 “输出信号” 一栏中有名字的信号均为有效输出信号。

表 5-2. 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_FUNC*n*_OEN_SEL = 0 时输出信号的输出使能信号	信号可经由 IO MUX 直接输出
0	SPIQ_in	0	yes	SPIQ_out	SPIQ_oe	yes
1	SPID_in	0	yes	SPID_out	SPID_oe	yes
2	SPIHD_in	0	yes	SPIHD_out	SPIHD_oe	yes
3	SPIWP_in	0	yes	SPIWP_out	SPIWP_oe	yes
4	-	-	-	SPICLK_out_mux	SPICLK_oe	yes
5	-	-	-	SPICS0_out	SPICS0_oe	yes
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	0	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	-	-	-	-	-	-
13	-	-	-	-	-	-
14	-	-	-	-	-	-
15	-	-	-	SPIQ_monitor	1'd1	no
16	-	-	-	SPID_monitor	1'd1	no
17	-	-	-	SPIHD_monitor	1'd1	no
18	-	-	-	SPIWP_monitor	1'd1	no
19	-	-	-	SPICS1_out	SPICS1_oe	no
20	-	-	-	-	-	-
21	-	-	-	-	-	-
22	-	-	-	-	-	-
23	-	-	-	-	-	-
24	-	-	-	-	-	-

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
25	-	-	-	-	-	-
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	ext_adc_start	0	no	ledc_ls_sig_out0	1'd1	no
46	-	-	-	ledc_ls_sig_out1	1'd1	no
47	-	-	-	ledc_ls_sig_out2	1'd1	no
48	-	-	-	ledc_ls_sig_out3	1'd1	no
49	-	-	-	ledc_ls_sig_out4	1'd1	no
50	-	-	-	ledc_ls_sig_out5	1'd1	no
51	-	-	-	-	-	-

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
52	-	-	-	-	-	-
53	I2CEXT0_SCL_in	1	no	I2CEXT0_SCL_out	I2CEXT0_SCL_oe	no
54	I2CEXT0_SDA_in	1	no	I2CEXT0_SDA_out	I2CEXT0_SDA_oe	no
55	-	-	-	-	-	-
56	-	-	-	-	-	-
57	-	-	-	-	-	-
58	-	-	-	-	-	-
59	-	-	-	-	-	-
60	-	-	-	-	-	-
61	-	-	-	-	-	-
62	-	-	-	-	-	-
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	-	-	-	FSPICS1_out	FSPICS1_oe	no
70	-	-	-	FSPICS2_out	FSPICS2_oe	no
71	-	-	-	FSPICS3_out	FSPICS3_oe	no
72	-	-	-	FSPICS4_out	FSPICS4_oe	no
73	-	-	-	FSPICS5_out	FSPICS5_oe	no
74	-	-	-	-	-	-
75	-	-	-	-	-	-
76	-	-	-	-	-	-
77	-	-	-	-	-	-
78	-	-	-	-	-	-

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
79	-	-	-	-	-	-
80	-	-	-	-	-	-
81	-	-	-	-	-	-
82	-	-	-	-	-	-
83	-	-	-	-	-	-
84	-	-	-	-	-	-
85	-	-	-	-	-	-
86	-	-	-	-	-	-
87	-	-	-	-	-	-
88	-	-	-	-	-	-
89	-	-	-	ant_sel0	1'd1	no
90	-	-	-	ant_sel1	1'd1	no
91	-	-	-	ant_sel2	1'd1	no
92	-	-	-	ant_sel3	1'd1	no
93	-	-	-	ant_sel4	1'd1	no
94	-	-	-	ant_sel5	1'd1	no
95	-	-	-	ant_sel6	1'd1	no
96	-	-	-	ant_sel7	1'd1	no
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	-	-	-	-	-	-
102	-	-	-	-	-	-
103	-	-	-	-	-	-
104	-	-	-	-	-	-
105	-	-	-	-	-	-

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
106	-	-	-	-	-	-
107	-	-	-	-	-	-
108	-	-	-	-	-	-
109	-	-	-	-	-	-
110	-	-	-	-	-	-
111	-	-	-	-	-	-
112	-	-	-	-	-	-
113	-	-	-	-	-	-
114	-	-	-	-	-	-
115	-	-	-	-	-	-
116	-	-	-	-	-	-
117	-	-	-	-	-	-
118	-	-	-	-	-	-
119	-	-	-	-	-	-
120	-	-	-	-	-	-
121	-	-	-	-	-	-
122	-	-	-	-	-	-
123	-	-	-	CLK_OUT_out1	1'd1	no
124	-	-	-	CLK_OUT_out2	1'd1	no
125	-	-	-	CLK_OUT_out3	1'd1	no
126	-	-	-	-	-	-
127	-	-	-	-	-	-

5.12 IO MUX 管脚功能列表

表 5-3 列出了所有 GPIO 管脚的 IO MUX 功能。

表 5-3. 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	-	-	2	0	R
1	GPIO1	GPIO1	GPIO1	-	-	2	0	R
2	GPIO2	GPIO2	GPIO2	FSPIQ	-	2	1	R
3	GPIO3	GPIO3	GPIO3	-	-	2	1	R
4	MTMS	MTMS	GPIO4	FSPIHD	-	2	1	R
5	MTDI	MTDI	GPIO5	FSPIWP	-	2	1	R
6	MTCK	MTCK	GPIO6	FSPICLK	-	2	1*	-
7	MTDO	MTDO	GPIO7	FSPID	-	2	1	-
8	GPIO8	GPIO8	GPIO8	-	-	2	1	-
9	GPIO9	GPIO9	GPIO9	-	-	2	3	-
10	GPIO10	GPIO10	GPIO10	FSPICS0	-	2	1	-
11	VDD_SPI	GPIO11	GPIO11	-	-	2	0	S
12	SPIHD	SPIHD	GPIO12	-	-	2	3	S
13	SPIWP	SPIWP	GPIO13	-	-	2	3	S
14	SPICS0	SPICS0	GPIO14	-	-	2	3	S
15	SPICLK	SPICLK	GPIO15	-	-	2	3	S
16	SPID	SPID	GPIO16	-	-	2	3	S
17	SPIQ	SPIQ	GPIO17	-	-	2	3	S
18	GPIO18	GPIO18	GPIO18	-	-	2	0	-
19	U0RXD	U0RXD	GPIO19	-	-	2	3	-
20	U0TXD	U0TXD	GPIO20	-	-	2	4	-

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（默认值）
- **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。

说明

- **R** - 代表位于 VDD3P3_RTC 电源域的管脚，部分具有模拟功能，见表 5-4。
- **S** - 在内置 SiP flash 的芯片版本中，代表该管脚专用于连接 SiP flash，且只能使用功能 0；在没有内置 SiP flash 的芯片版本中，用户可正常使用该管脚的所有功能。

5.13 IO MUX 管脚模拟功能列表

表 5-4 列出了具有模拟功能的 IO MUX 管脚。

表 5-4. IO MUX 管脚的模拟功能

GPIO 编号	管脚名称	模拟功能
0	GPIO0	ADC1_CH0
1	GPIO1	ADC1_CH1
2	GPIO2	ADC1_CH2
3	GPIO3	ADC1_CH3
4	MTMS	ADC1_CH4
5	MTDI	ADC2_CH0

5.14 寄存器列表

5.14.1 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 3 *系统和存储器* 中的表 3-3。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

注意：在内置 SiP flash 的芯片版本中，可配置使用 14 个 GPIO 管脚，即 GPIO0 ~ GPIO10、GPIO18 ~ GPIO20 管脚，因此：

- **配置寄存器**只可以配置 GPIO0 ~ GPIO10 和 GPIO18 ~ GPIO20；
- **管脚配置寄存器**只可以使用 GPIO_PIN0_REG ~ GPIO_PIN10_REG 和 GPIO_PIN18_REG ~ GPIO_PIN20_REG 寄存器；
- **输入配置寄存器**只可以配置选择 GPIO0 ~ GPIO10 和 GPIO18 ~ 20；
- **输出配置寄存器**只可以使用 GPIO_FUNC0_OUT_SEL_CFG_REG ~ GPIO_FUNC10_OUT_SEL_CFG_REG 和 GPIO_PIN18_OUT_SEL_CFG_REG ~ GPIO_PIN20_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/SS
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

名称	描述	地址	访问
GPIO_ENABLE_REG	GPIO 输出使能寄存器	0x0020	R/W/SS
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
GPIO_STATUS_REG	GPIO 中断状态寄存器	0x0044	R/W/SS
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 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	配置 GPIO0 管脚	0x0074	R/W
GPIO_PIN1_REG	配置 GPIO1 管脚	0x0078	R/W
GPIO_PIN2_REG	配置 GPIO2 管脚	0x007C	R/W
GPIO_PIN3_REG	配置 GPIO3 管脚	0x0080	R/W
GPIO_PIN4_REG	配置 GPIO4 管脚	0x0084	R/W
GPIO_PIN5_REG	配置 GPIO5 管脚	0x0088	R/W
GPIO_PIN6_REG	配置 GPIO6 管脚	0x008C	R/W
GPIO_PIN7_REG	配置 GPIO7 管脚	0x0090	R/W
GPIO_PIN8_REG	配置 GPIO8 管脚	0x0094	R/W
GPIO_PIN9_REG	配置 GPIO9 管脚	0x0098	R/W
GPIO_PIN10_REG	配置 GPIO10 管脚	0x009C	R/W
GPIO_PIN11_REG	配置 GPIO11 管脚	0x00A0	R/W
GPIO_PIN12_REG	配置 GPIO12 管脚	0x00A4	R/W
GPIO_PIN13_REG	配置 GPIO13 管脚	0x00A8	R/W
GPIO_PIN14_REG	配置 GPIO14 管脚	0x00AC	R/W
GPIO_PIN15_REG	配置 GPIO15 管脚	0x00B0	R/W
GPIO_PIN16_REG	配置 GPIO16 管脚	0x00B4	R/W
GPIO_PIN17_REG	配置 GPIO17 管脚	0x00B8	R/W
GPIO_PIN18_REG	配置 GPIO18 管脚	0x00BC	R/W
GPIO_PIN19_REG	配置 GPIO19 管脚	0x00C0	R/W
GPIO_PIN20_REG	配置 GPIO20 管脚	0x00C4	R/W
输入配置寄存器			
GPIO_FUNC0_IN_SEL_CFG_REG	外设输入信号 0 配置寄存器	0x0154	R/W
GPIO_FUNC1_IN_SEL_CFG_REG	外设输入信号 1 配置寄存器	0x0158	R/W
GPIO_FUNC2_IN_SEL_CFG_REG	外设输入信号 2 配置寄存器	0x015C	R/W
...	...	...	...
GPIO_FUNC125_IN_SEL_CFG_REG	外设输入信号 125 配置寄存器	0x0348	R/W
GPIO_FUNC126_IN_SEL_CFG_REG	外设输入信号 126 配置寄存器	0x034C	R/W
GPIO_FUNC127_IN_SEL_CFG_REG	外设输入信号 127 配置寄存器	0x0350	R/W
输出配置寄存器			
GPIO_FUNC0_OUT_SEL_CFG_REG	GPIO0 管脚的输出配置寄存器	0x0554	R/W
GPIO_FUNC1_OUT_SEL_CFG_REG	GPIO1 管脚的输出配置寄存器	0x0558	R/W

名称	描述	地址	访问
GPIO_FUNC2_OUT_SEL_CFG_REG	GPIO2 管脚的输出配置寄存器	0x055C	R/W
GPIO_FUNC3_OUT_SEL_CFG_REG	GPIO3 管脚的输出配置寄存器	0x0560	R/W
GPIO_FUNC4_OUT_SEL_CFG_REG	GPIO4 管脚的输出配置寄存器	0x0564	R/W
GPIO_FUNC5_OUT_SEL_CFG_REG	GPIO5 管脚的输出配置寄存器	0x0568	R/W
GPIO_FUNC6_OUT_SEL_CFG_REG	GPIO6 管脚的输出配置寄存器	0x056C	R/W
GPIO_FUNC7_OUT_SEL_CFG_REG	GPIO7 管脚的输出配置寄存器	0x0570	R/W
GPIO_FUNC8_OUT_SEL_CFG_REG	GPIO8 管脚的输出配置寄存器	0x0574	R/W
GPIO_FUNC9_OUT_SEL_CFG_REG	GPIO9 管脚的输出配置寄存器	0x0578	R/W
GPIO_FUNC10_OUT_SEL_CFG_REG	GPIO10 管脚的输出配置寄存器	0x057C	R/W
GPIO_FUNC11_OUT_SEL_CFG_REG	GPIO11 管脚的输出配置寄存器	0x0580	R/W
GPIO_FUNC12_OUT_SEL_CFG_REG	GPIO12 管脚的输出配置寄存器	0x0584	R/W
GPIO_FUNC13_OUT_SEL_CFG_REG	GPIO13 管脚的输出配置寄存器	0x0588	R/W
GPIO_FUNC14_OUT_SEL_CFG_REG	GPIO14 管脚的输出配置寄存器	0x058C	R/W
GPIO_FUNC15_OUT_SEL_CFG_REG	GPIO15 管脚的输出配置寄存器	0x0590	R/W
GPIO_FUNC16_OUT_SEL_CFG_REG	GPIO16 管脚的输出配置寄存器	0x0594	R/W
GPIO_FUNC17_OUT_SEL_CFG_REG	GPIO17 管脚的输出配置寄存器	0x0598	R/W
GPIO_FUNC18_OUT_SEL_CFG_REG	GPIO18 管脚的输出配置寄存器	0x059C	R/W
GPIO_FUNC19_OUT_SEL_CFG_REG	GPIO19 管脚的输出配置寄存器	0x05A0	R/W
GPIO_FUNC20_OUT_SEL_CFG_REG	GPIO20 管脚的输出配置寄存器	0x05A4	R/W
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

5.14.2 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 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

注意：在内置 SiP flash 的芯片版本中，可配置使用 14 个 GPIO，即 GPIO 0 ~ 10 和 GPIO 18 ~ GPIO20 管脚，因此**配置寄存器**不可以配置 IO_MUX_GPIO11_REG ~ IO_MUX_GPIO17_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 管脚配置寄存器	0x0004	R/W
IO_MUX_GPIO1_REG	GPIO1 的 IO MUX 管脚配置寄存器	0x0008	R/W
IO_MUX_GPIO2_REG	GPIO2 的 IO MUX 管脚配置寄存器	0x000C	R/W
IO_MUX_GPIO3_REG	GPIO3 的 IO MUX 管脚配置寄存器	0x0010	R/W
IO_MUX_GPIO4_REG	MTMS 的 IO MUX 管脚配置寄存器	0x0014	R/W
IO_MUX_GPIO5_REG	MTDI 的 IO MUX 管脚配置寄存器	0x0018	R/W
IO_MUX_GPIO6_REG	MTCK 的 IO MUX 管脚配置寄存器	0x001C	R/W
IO_MUX_GPIO7_REG	MTDO 的 IO MUX 管脚配置寄存器	0x0020	R/W

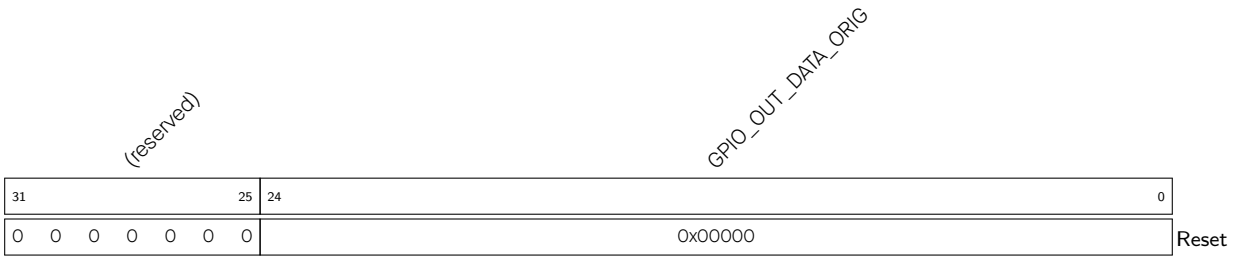
名称	描述	地址	访问
IO_MUX_GPIO8_REG	GPIO8 的 IO MUX 管脚配置寄存器	0x0024	R/W
IO_MUX_GPIO9_REG	GPIO9 的 IO MUX 管脚配置寄存器	0x0028	R/W
IO_MUX_GPIO10_REG	GPIO10 的 IO MUX 管脚配置寄存器	0x002C	R/W
IO_MUX_GPIO11_REG	VDD_SPI 的 IO MUX 管脚配置寄存器	0x0030	R/W
IO_MUX_GPIO12_REG	SPIHD 的 IO MUX 管脚配置寄存器	0x0034	R/W
IO_MUX_GPIO13_REG	SPIWP 的 IO MUX 管脚配置寄存器	0x0038	R/W
IO_MUX_GPIO14_REG	SPICS0 的 IO MUX 管脚配置寄存器	0x003C	R/W
IO_MUX_GPIO15_REG	SPICLK 的 IO MUX 管脚配置寄存器	0x0040	R/W
IO_MUX_GPIO16_REG	SPID 的 IO MUX 管脚配置寄存器	0x0044	R/W
IO_MUX_GPIO17_REG	SPIQ 的 IO MUX 管脚配置寄存器	0x0048	R/W
IO_MUX_GPIO18_REG	GPIO18 的 IO MUX 管脚配置寄存器	0x004C	R/W
IO_MUX_GPIO19_REG	U0RXD 的 IO MUX 管脚配置寄存器	0x0050	R/W
IO_MUX_GPIO20_REG	U0TXD 的 IO MUX 管脚配置寄存器	0x0054	R/W
版本寄存器			
IO_MUX_DATE_REG	版本控制寄存器	0x00FC	R/W

5.15 寄存器

5.15.1 GPIO 交换矩阵寄存器

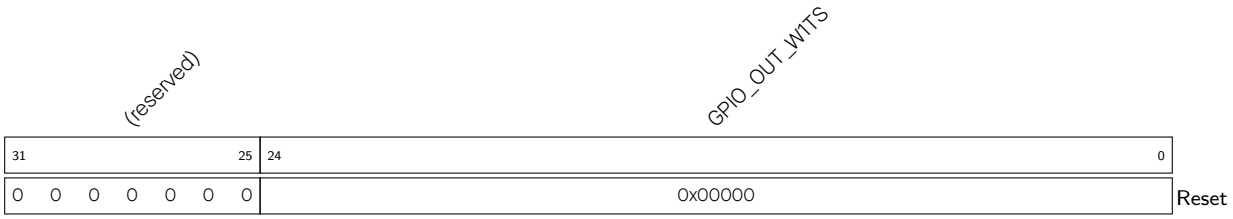
本小节的所有地址均为相对于 GPIO 基地址的地址偏移量（相对地址），具体基地址请见章节 3 *系统和存储器* 中的表 3-3。

Register 5.1. GPIO_OUT_REG (0x0004)



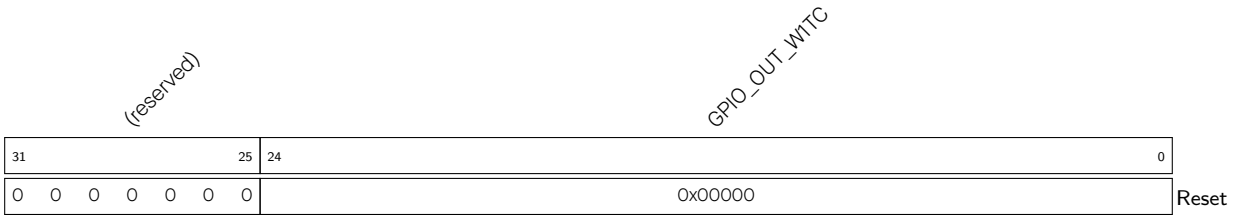
GPIO_OUT_DATA_ORIG 简单 GPIO 输出模式下 GPIO0 ~ 20 的输出值。bit0 ~ bit20 的值分别对应 GPIO0 ~ GPIO20 的值。bit21 ~ bit24 无效。(R/W/SS)

Register 5.2. GPIO_OUT_W1TS_REG (0x0008)



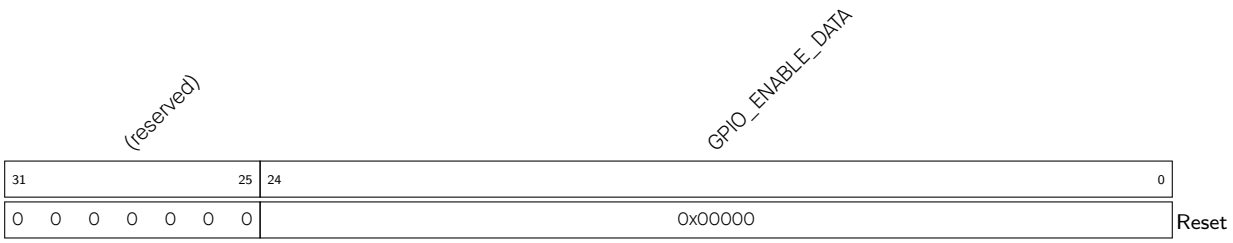
GPIO_OUT_W1TS GPIO0 ~ 20 输出置位寄存器，bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位置 1，则 GPIO_OUT_REG 中相应位也将置 1。注：推荐使用此寄存器来置位 GPIO_OUT_REG。(WT)

Register 5.3. GPIO_OUT_W1TC_REG (0x000C)



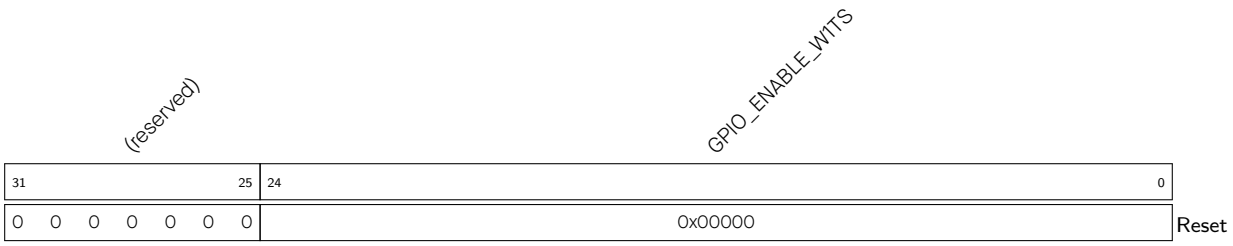
GPIO_OUT_W1TC GPIO0 ~ 20 输出清零寄存器，bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位置 1，则 GPIO_OUT_REG 中相应位会清零。注：推荐使用此寄存器来清零 GPIO_OUT_REG。(WT)

Register 5.4. GPIO_ENABLE_REG (0x0020)



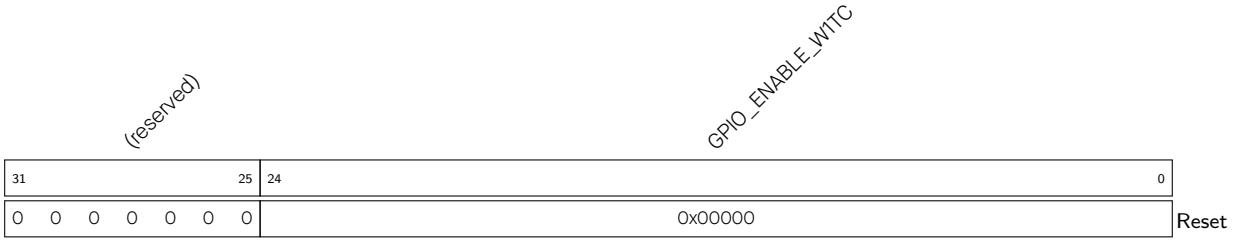
GPIO_ENABLE_DATA GPIO0 ~ 20 输出使能寄存器，bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。(R/W/SS)

Register 5.5. GPIO_ENABLE_W1TS_REG (0x0024)



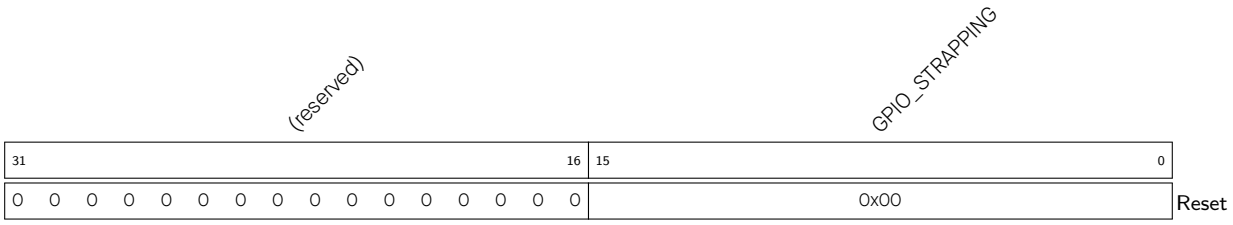
GPIO_ENABLE_W1TS GPIO0 ~ 20 输出使能置位寄存器。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位置 1，则 GPIO_ENABLE_REG 中相应位也将置 1。注：推荐使用此寄存器来置位 GPIO_ENABLE_REG。(WT)

Register 5.6. GPIO_ENABLE_W1TC_REG (0x0028)



GPIO_ENABLE_W1TC GPIO0 ~ 20 输出使能清零寄存器。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位置 1，则 GPIO_ENABLE_REG 中相应位会清零。注：推荐使用此寄存器清零 GPIO_ENABLE_REG。(WT)

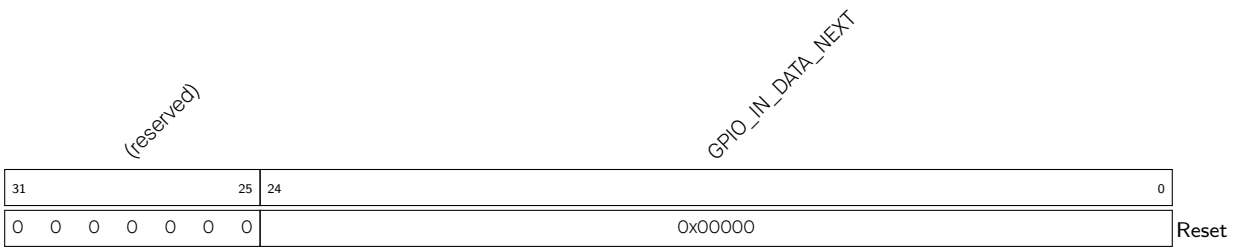
Register 5.7. GPIO_STRAP_REG (0x0038)



GPIO_STRAPPING GPIO Strapping 值。(RO)

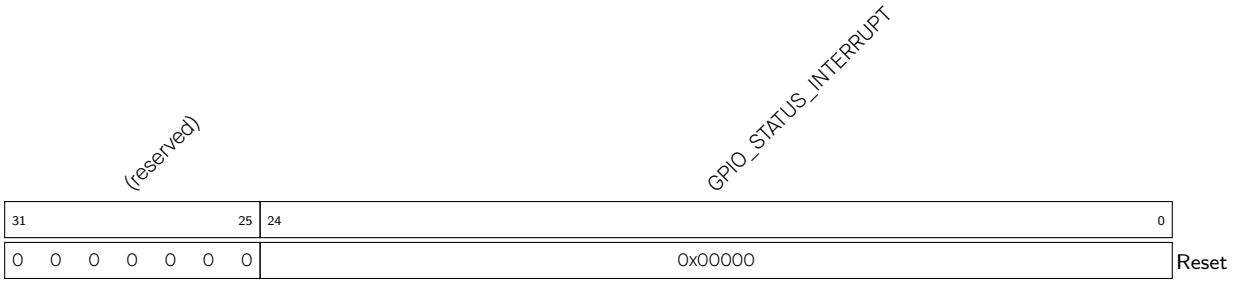
- bit 2：对应 GPIO8
- bit 3：对应 GPIO9

Register 5.8. GPIO_IN_REG (0x003C)



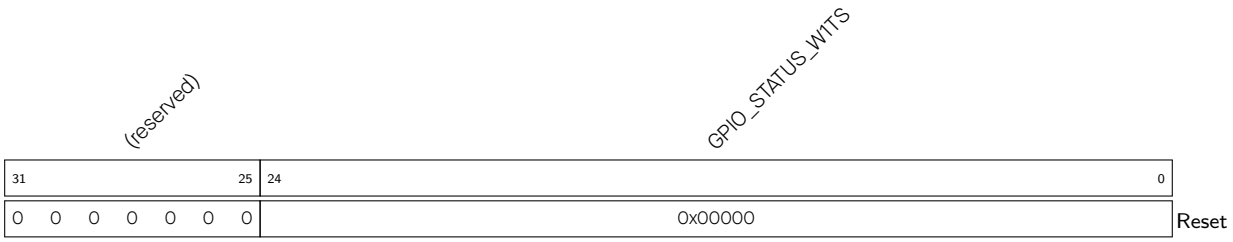
GPIO_IN_DATA_NEXT GPIO0 ~ 20 输入值。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位代表一个管脚的片外输入值，0 表示低电平，1 表示高电平。(RO)

Register 5.9. GPIO_STATUS_REG (0x0044)



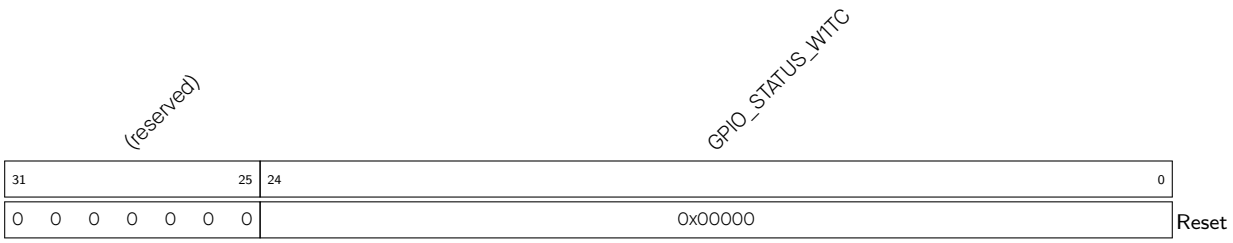
GPIO_STATUS_INTERRUPT GPIO0 ~ 20 中断状态寄存器。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。(R/W/SS)

Register 5.10. GPIO_STATUS_W1TS_REG (0x0048)



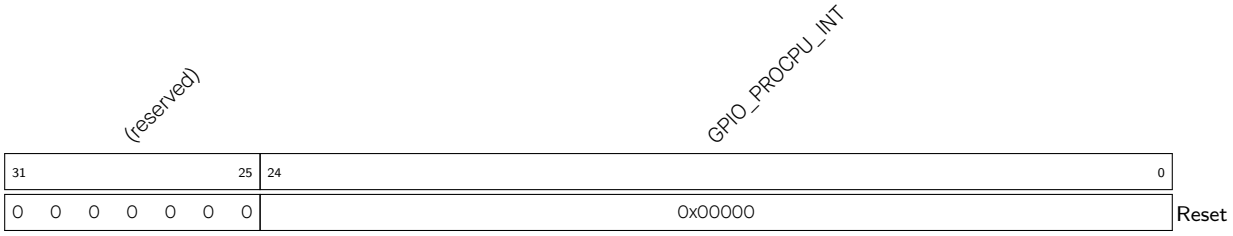
GPIO_STATUS_W1TS GPIO0 ~ 20 中断状态置位寄存器。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位置 1，则 GPIO_STATUS_INTERRUPT 中相应位也将置 1。注：推荐使用此寄存器来置位 GPIO_STATUS_INTERRUPT。(WT)

Register 5.11. GPIO_STATUS_W1TC_REG (0x004C)



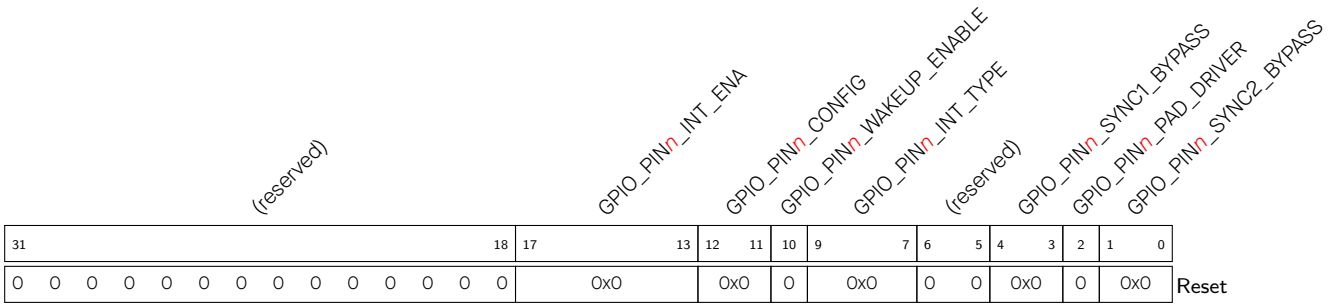
GPIO_STATUS_W1TC GPIO0 ~ 20 中断状态清除寄存器。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。每一位置 1，则 GPIO_STATUS_INTERRUPT 中相应位会清零。注：推荐使用此寄存器来清零 GPIO_STATUS_INTERRUPT。(WT)

Register 5.12. GPIO_PCPU_INT_REG (0x005C)



GPIO_PROCPU_INT GPIO0 ~ 20 CPU 中断状态。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 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。(RO)

Register 5.13. GPIO_PINn_REG (n: 0-20) (0x0074+4*n)



GPIO_PINn_SYNC2_BYPASS 使能 GPIO 输入信号在第二级同步中为上升沿或下降沿同步。0：关闭同步；1：下降沿同步；2 或 3：上升沿同步。(R/W)

GPIO_PINn_PAD_DRIVER 管脚驱动选择。0：正常输出；1：开漏输出。(R/W)

GPIO_PINn_SYNC1_BYPASS 使能 GPIO 输入信号在第一级同步中为上升沿或下降沿同步。0：关闭同步；1：下降沿同步；2 或 3：上升沿同步。(R/W)

GPIO_PINn_INT_TYPE 中断类型选择。(R/W)

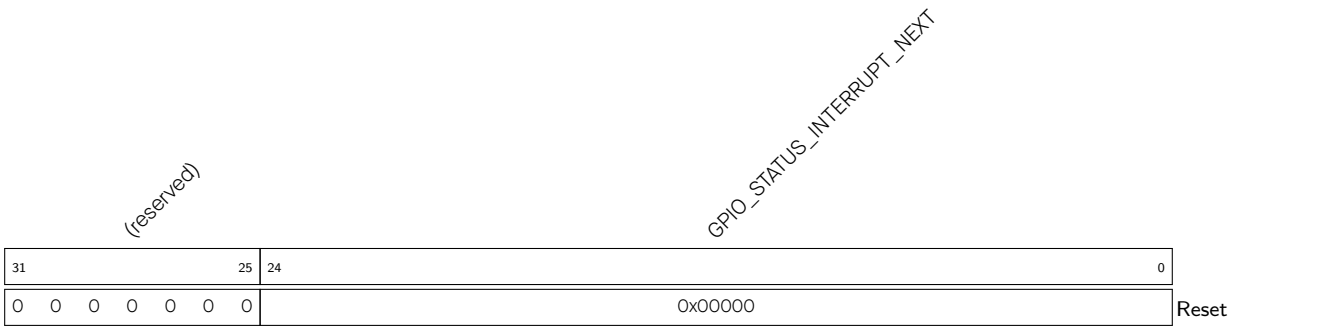
- 0：禁用 GPIO 中断
- 1：上升沿触发
- 2：下降沿触发
- 3：任一沿触发
- 4：低电平触发
- 5：高电平触发

GPIO_PINn_WAKEUP_ENABLE 使能 GPIO 唤醒，仅能将 CPU 从 Light-sleep 模式唤醒。(R/W)

GPIO_PINn_CONFIG 保留。(R/W)

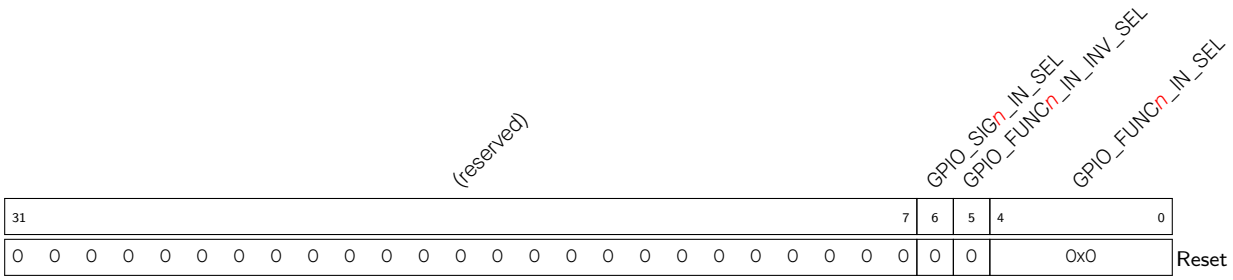
GPIO_PINn_INT_ENA 中断使能位。bit13：使能 CPU 中断；bit14：使能 CPU 非屏蔽中断。(R/W)

Register 5.14. GPIO_STATUS_NEXT_REG (0x014C)



GPIO_STATUS_INTERRUPT_NEXT GPIO0 ~ 20 中断源信号，可以设置为上升沿中断、下降沿中断、电平敏感中断或任一沿中断。bit0 ~ bit20 对应 GPIO0 ~ 20，bit21 ~ bit24 无效。(RO)

Register 5.15. GPIO_FUNC*n*_IN_SEL_CFG_REG (*n*: 0-127) (0x0154+4**n*)

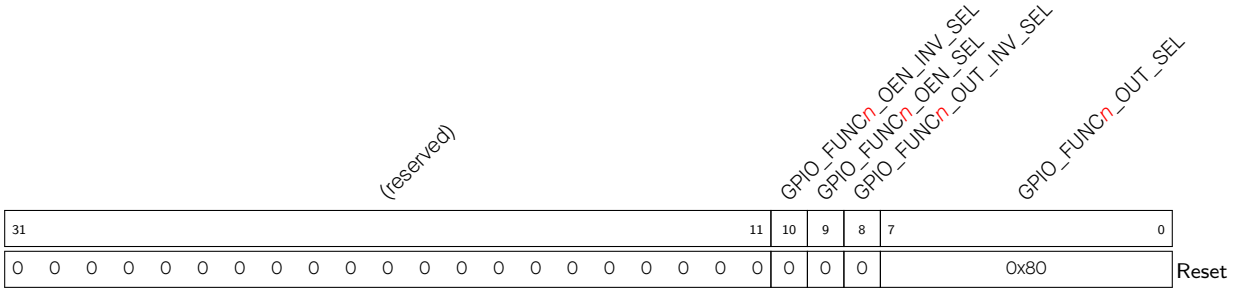


GPIO_FUNC*n*_IN_SEL 外设输入信号 *n* 的选择控制位。此位选择 1 个 GPIO 交换矩阵输入管脚与信号连接，或者选择 0x1E 与恒高电平输入信号连接，或者选择 0x1F 与恒低电平输入信号连接。(R/W)

GPIO_FUNC*n*_IN_INV_SEL 反转输入值。1：反转；0：不反转。(R/W)

GPIO_SIG*n*_IN_SEL 旁路 GPIO 交换矩阵。1：通过 GPIO 交换矩阵；0：直接通过 IO MUX 连接信号与外设。(R/W)

Register 5.16. GPIO_FUNC*n*_OUT_SEL_CFG_REG (*n*: 0-20) (0x0554+4**n*)



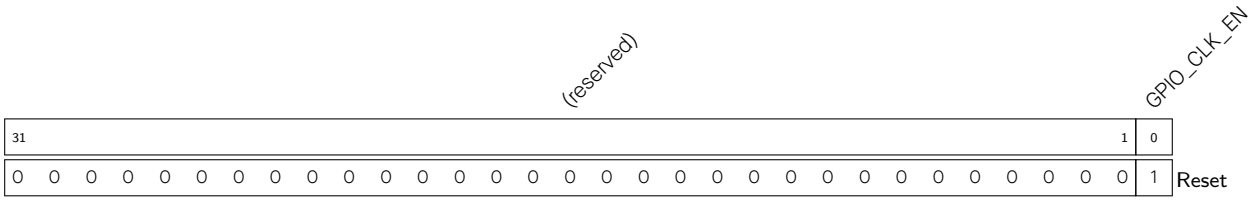
GPIO_FUNC*n*_OUT_SEL GPIO 管脚输出 *n* 的选择控制位。如果该字段设置为 *Y* (0<=Y<128)，则外设输出信号 *Y* 将连接至 GPIO *n* 输出。如果该字段设置为 128，则寄存器 GPIO_OUT_REG 和 GPIO_ENABLE_REG 中的 bit*n* 将用作输出值和输出使能。(R/W)

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

GPIO_FUNC*n*_OEN_SEL 0：采用外设的输出使能信号；1：强制使用 GPIO_ENABLE_REG 的 bit*n* 用作输出使能信号。(R/W)

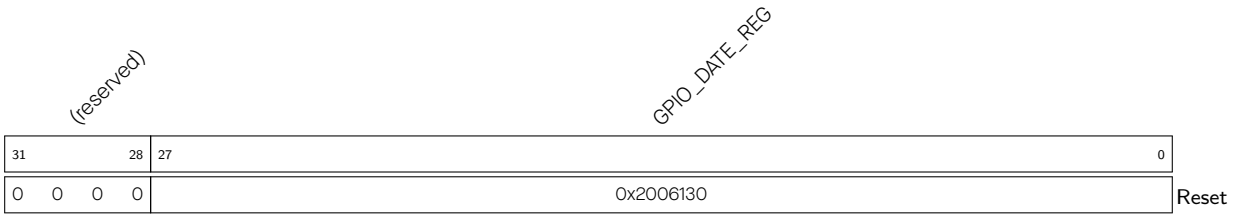
GPIO_FUNC*n*_OEN_INV_SEL 0：不反转输出使能信号；1：反转输出使能信号。(R/W)

Register 5.17. GPIO_CLOCK_GATE_REG (0x062C)



GPIO_CLK_EN 时钟门控使能。此位置 1，则时钟自由运转。(R/W)

Register 5.18. GPIO_DATE_REG (0x06FC)

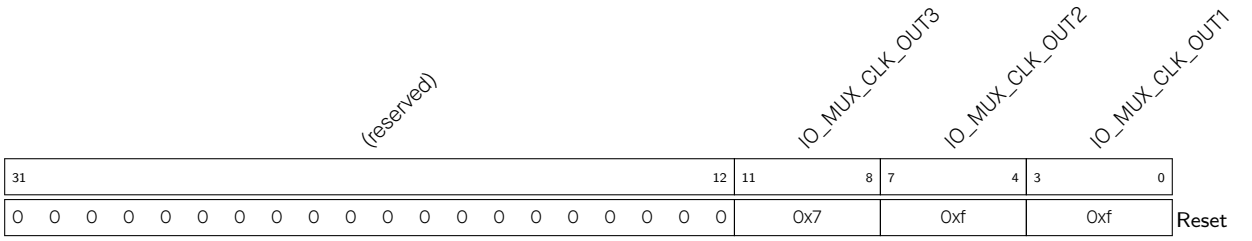


GPIO_DATE_REG 版本控制寄存器。(R/W)

5.15.2 IO MUX 寄存器

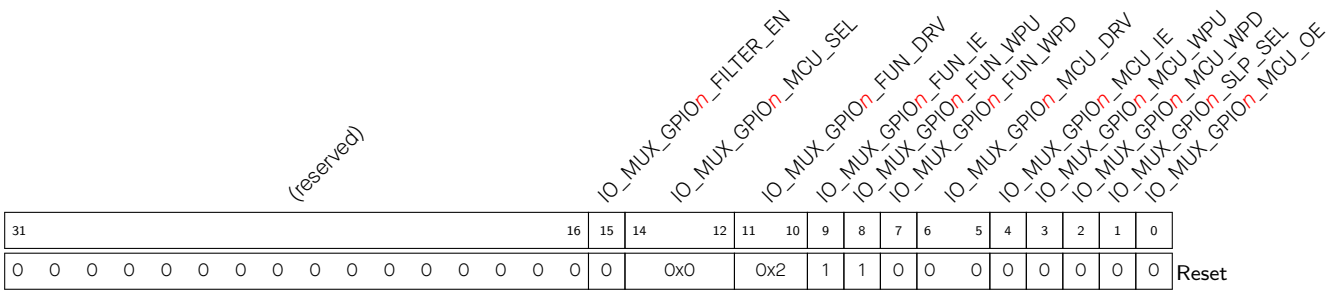
本小节的所有地址均为相对于 IO MUX 基地址的地址偏移量（相对地址），具体基地址请见章节 3 *系统和存储器* 中的表 3-3。

Register 5.19. IO_MUX_PIN_CTRL_REG (0x0000)



IO_MUX_CLK_OUT*x* 配置 I2S 外设时钟输出到 CLK_OUT_out*x*，需要设置 IO_MUX_CLK_OUT*x* 为 0x0。有关 CLK_OUT_out*x* 的信息，见表 5-2。(R/W)

Register 5.20. IO_MUX_GPIO*n*_REG (*n*: 0-20) (0x0004+4**n*)



IO_MUX_GPIO*n*_MCU_OE 睡眠模式下，管脚的输出使能位。1：输出使能；0：输出关闭。(R/W)

IO_MUX_GPIO*n*_SLP_SEL 管脚睡眠模式选择。置 1 进入睡眠模式。(R/W)

IO_MUX_GPIO*n*_MCU_WPD 睡眠模式下，管脚的下拉电阻使能位。1：使能内部下拉电阻；0：关闭内部下拉电阻。(R/W)

IO_MUX_GPIO*n*_MCU_WPU 睡眠模式下，管脚的上拉电阻使能位。1：使能内部上拉电阻；0：关闭内部上拉电阻。(R/W)

IO_MUX_GPIO*n*_MCU_IE 睡眠模式下，管脚的输入使能位。1：使能输入；0：关闭输入。(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

(读/写)

IO_MUX_GPIO*n*_FUN_WPD 管脚的下拉电阻使能位。1：使能内部下拉电阻；0：关闭内部下拉电阻。(R/W)

IO_MUX_GPIO*n*_FUN_WPU 管脚的上拉电阻使能位。1：使能内部上拉电阻；0：关闭内部上拉电阻。(R/W)

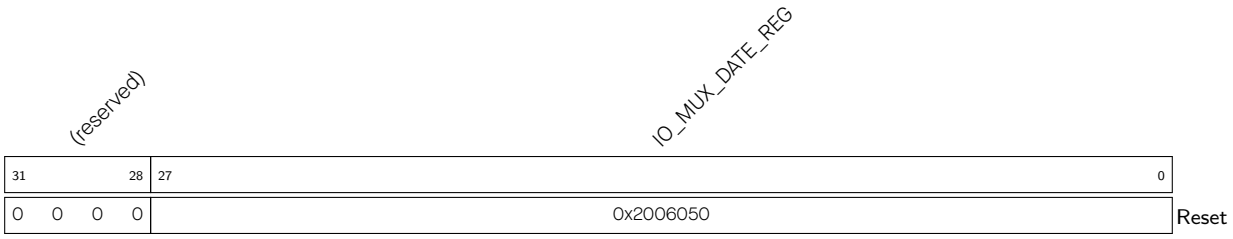
IO_MUX_GPIO*n*_FUN_IE 管脚的输入使能位。1：使能输入；0：关闭输入。(R/W)

IO_MUX_GPIO*n*_FUN_DRV 选择管脚驱动强度。0：~5 mA；1：~10 mA；2：~20 mA；3：~40 mA。(R/W)

IO_MUX_GPIO*n*_MCU_SEL 选择管脚功能。0：选择 Function 0；1：选择 Function 1；以此类推。(R/W)

IO_MUX_GPIO*n*_FILTER_EN 使能管脚输入信号滤波。1：滤波使能；0：滤波关闭。(R/W)

Register 5.21. IO_MUX_DATE_REG (0x00FC)



IO_MUX_DATE_REG 版本控制寄存器。(R/W)

6 复位和时钟

6.1 复位

6.1.1 概述

ESP8684 提供四种级别的复位类型，分别是 CPU 复位、内核复位、系统复位和芯片复位。除芯片复位外其它复位类型不影响片上内存存储的数据。图 6-1 展示了整个芯片系统的结构以及四种复位类型。

6.1.2 结构图

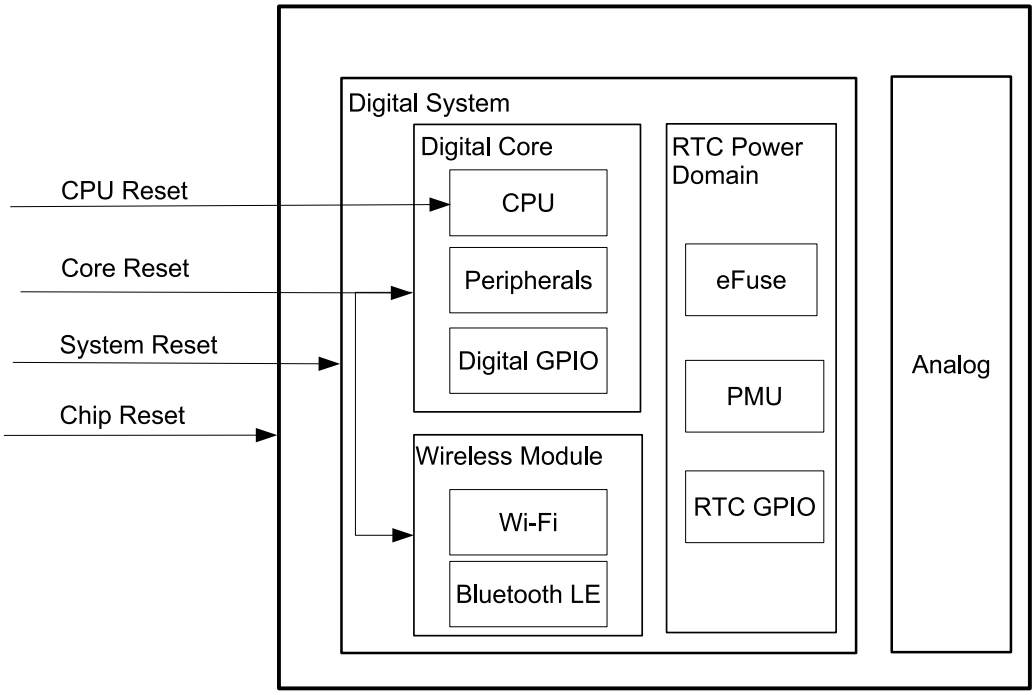


图 6-1. 四种复位类型

6.1.3 特性

- 支持四种复位类型：
  - CPU 复位：复位 CPU 核。复位释放后，程序将从 CPU Reset Vector 开始执行；
  - 内核复位：复位除 RTC 以外的其它数字系统，包括 CPU、外设、Wi-Fi、Bluetooth® LE 及数字 GPIO；
  - 系统复位：复位包括 RTC 在内的整个数字系统；
  - 芯片复位：复位整个芯片。
- 支持软件复位和硬件复位：
  - 软件复位：配置 CPU 相关寄存器可触发软件复位，见章节 *9 低功耗管理 (RTC_CNTL)*；
  - 硬件复位：硬件复位直接由硬件电路触发。

说明：

如果 CPU 发生复位，则 SENSITIVE 寄存器也将复位。

6.1.4 功能描述

上述任一复位发生时，CPU 将立刻复位。复位释放后，CPU 可通过读取寄存器 RTC_CNTL_RESET_CAUSE _PROCPU 获取复位源。

表 6-1 列出了从上述寄存器中可能读出的复位源以及触发的复位类型。

表 6-1. 复位源

编码	复位源	复位类型	说明
0x01	芯片复位	芯片复位	见表下方说明[1]
0x0F	欠压系统复位	芯片复位或系统复位	欠压检测器触发的系统复位，见表下方说明[2]
0x10	RWDT 系统复位	系统复位	详见章节 12 看门狗定时器 (WDT)
0x12	模拟超级看门狗复位	系统复位	详见章节 12 看门狗定时器 (WDT)
0x13	时钟毛刺复位	系统复位	详见章节 1 时钟毛刺检测 [to be added later]
0x03	软件系统复位	内核复位	配置 RTC_CNTL_SW_SYS_RST 寄存器触发
0x05	Deep-sleep 复位	内核复位	详见章节 9 低功耗管理 (RTC_CNTL)
0x07	MWDT0 内核复位	内核复位	详见章节 12 看门狗定时器 (WDT)
0x09	RWDT 内核复位	内核复位	详见章节 12 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
0x18	JTAG 复位	CPU 复位	JTAG 触发复位
0x0B	MWDT0 CPU 复位	CPU 复位	详见章节 12 看门狗定时器 (WDT)
0x0C	软件 CPU 复位	CPU 复位	配置 RTC_CNTL_SW_PROCPU_RST 寄存器触发
0x0D	RWDT CPU 复位	CPU 复位	详见章节 12 看门狗定时器 (WDT)

[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 RTC_CNTL_BROWN_OUT_RST_SEL 的配置，选择触发系统复位或者芯片复位。详见章节 9 低功耗管理 (RTC_CNTL)。

6.2 时钟

6.2.1 概述

ESP8684 的时钟主要来源于外部晶体振荡器 (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以及选择对应频率的工作时钟。图 6-2 为系统时钟结构。

6.2.2 结构图

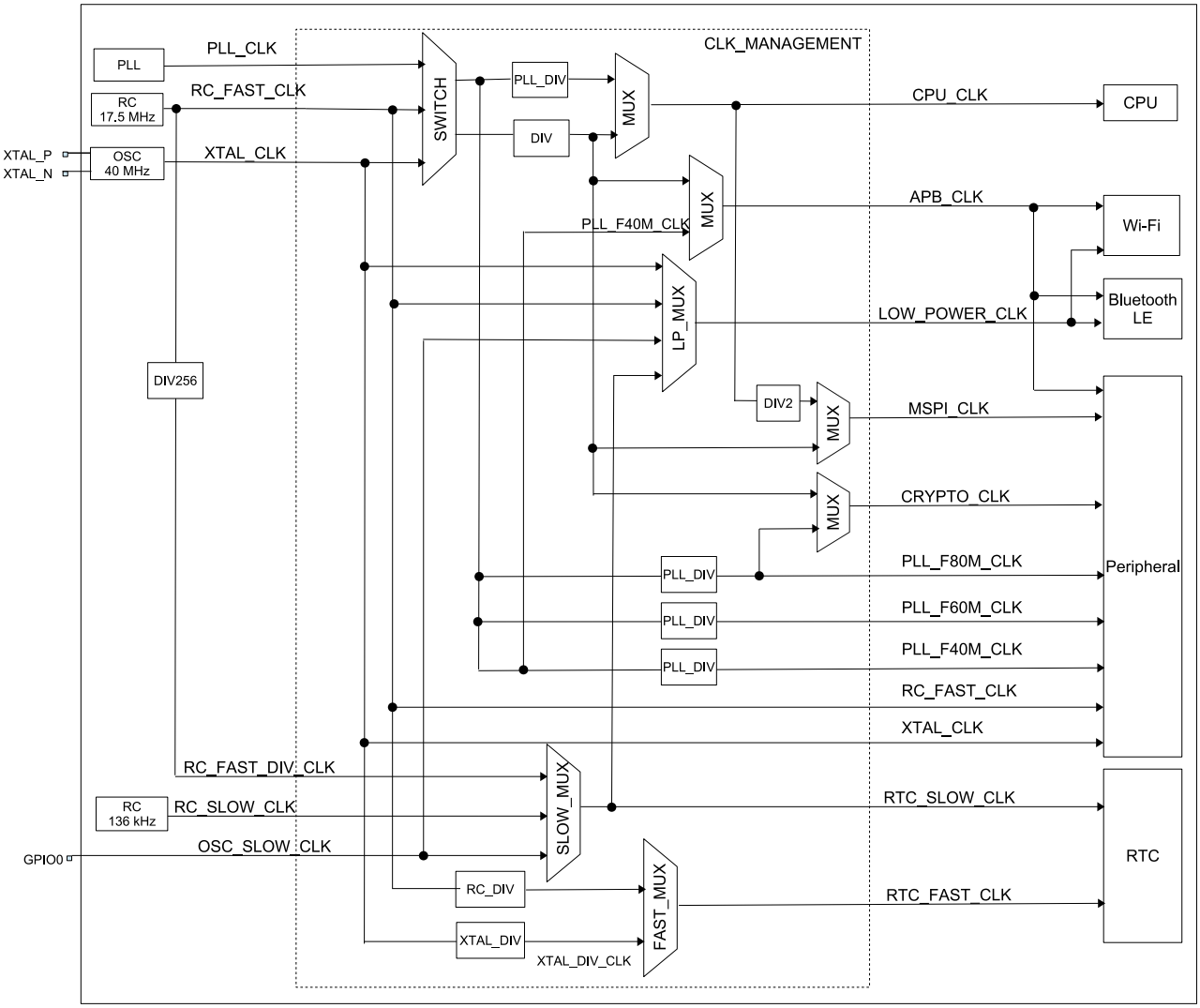


图 6-2. 系统时钟

6.2.3 特性

ESP8684 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_CLK：480 MHz 内部 PLL 时钟
  - XTAL_CLK：40 MHz 外部晶振时钟
- 低功耗时钟，主要为 RTC 模块以及部分处于低功耗模式的外设提供工作时钟
  - OSC_SLOW_CLK：来自 GPIO0 的外部低速时钟，频率通常为 32 kHz
  - RC_FAST_CLK：内置快速 RC 振荡器时钟，频率可调节（通常为 17.5 MHz）
  - RC_FAST_DIV_CLK：内置快速 RC 振荡器分频时钟，由内置快速 RC 振荡器时钟经 256 分频生成
  - RC_SLOW_CLK：内置慢速 RC 振荡器，频率可调节（通常为 136 kHz）

6.2.4 功能描述

6.2.4.1 CPU 时钟

如图 6-2 所示，CPU_CLK 为 CPU 主时钟。CPU 在最高效工作模式下，主频可以达到 120 MHz。同时，CPU 能够在超低频下工作（通常为 2 MHz），以减少功耗。CPU_CLK 由 SYSTEM_SOC_CLK_SEL 来选择时钟源，允许选择 PLL_CLK、RC_FAST_CLK 或 XTAL_CLK 作为 CPU_CLK 的时钟源。具体请参考表 6-2 和表 6-3。默认状态下，CPU 的时钟为 XTAL_CLK，且分频系数为 2 分频，即 20 MHz。

表 6-2. CPU_CLK 时钟源选择

SYSTEM_SOC_CLK_SEL 值	时钟源
0	XTAL_CLK
1	PLL_CLK
2	RC_FAST_CLK

表 6-3. CPU_CLK 时钟频率

时钟源	SEL_0*	SEL_1*	CPU 时钟频率
XTAL_CLK	0	-	CPU_CLK = XTAL_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT 默认值为 1，范围 0 ~ 1023。
PLL_CLK	1	0	CPU_CLK = PLL_CLK/6 CPU_CLK 频率为 80 MHz。
PLL_CLK	1	1	CPU_CLK = PLL_CLK/4 CPU_CLK 频率为 120 MHz。
RC_FAST_CLK	2	-	CPU_CLK = RC_FAST_CLK/(SYSTEM_PRE_DIV_CNT + 1) SYSTEM_PRE_DIV_CNT 默认值为 1，范围 0 ~ 1023。

* 寄存器 SYSTEM_SOC_CLK_SEL 的值；
* 寄存器 SYSTEM_CPUPERIOD_SEL 的值。

6.2.4.2 外设时钟

外设所需要的时钟可分为总线时钟和功能时钟。

- 总线时钟：APB_CLK
- 功能时钟：CRYPTO_CLK、PLL_F80M_CLK、PLL_F60M_CLK、PLL_F40M_CLK、MSPI_CLK、XTAL_CLK 和 RC_FAST_CLK

表 6-4 列出了接入各个外设的功能时钟。

表 6-4. 外设时钟

外设	XTAL_CLK	RC_FAST_CLK	PLL_F40M_CLK	PLL_F60M_CLK	PLL_F80M_CLK	RTC_FAST_CLK	CRYPTO_CLK	MSPI_CLK
定时器组	Y		Y					
UART	Y	Y	Y					
I2C	Y	Y						
SPI	Y		Y					
LEDC	Y	Y		Y				
SAR ADC	Y				Y			
温度传感器	Y	Y						
系统定时器	Y							
Crypto							Y	
MSPI								Y
eFuse						Y		

GoBack

APB_CLK 时钟

如表 6-5 所示，APB_CLK 的频率由 CPU_CLK 的时钟源决定。

表 6-5. APB_CLK 时钟

CPU_CLK 时钟源	APB_CLK 频率
PLL_CLK	40 MHz
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

CRYPTO_CLK 时钟

如表 6-6 所示，CRYPTO_CLK 的频率由 CPU_CLK 的时钟源决定。

表 6-6. CRYPTO_CLK 时钟

CPU_CLK 时钟源	CRYPTO_CLK 频率
PLL_CLK	80 MHz
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

MSPI_CLK 时钟

如表 6-7 所示，MSPI_CLK 的频率由 CPU_CLK 的时钟源决定。

表 6-7. MSPI_CLK 时钟

CPU_CLK 时钟源	MSPI_CLK 频率
PLL_CLK	CPU_CLK/2
XTAL_CLK	CPU_CLK
RC_FAST_CLK	CPU_CLK

PLL_F80M_CLK、PLL_F60M_CLK、PLL_F40M_CLK 时钟

PLL_F80M_CLK、PLL_F60M_CLK、PLL_F40M_CLK 是 PLL_CLK 根据当前 PLL 的频率分频所得。

6.2.4.3 Wireless 时钟

ESP8684 中的 Wireless 时钟为 LOW_POWER_CLK 时钟，用于 Wi-Fi 和 Bluetooth LE 的低功耗模式。LOW_POWER_CLK 可选择 OSC_SLOW_CLK、XTAL_CLK、RC_FAST_CLK、RTC_SLOW_CLK（RTC 当前所选的慢速时钟）作为时钟源。

注意： Wi-Fi 和 Bluetooth LE 必须在 CPU_CLK 时钟源选择 PLL_CLK 下才能工作。只有当 Wi-Fi 和 Bluetooth LE 进入低功耗模式时，才能暂时关闭 PLL_CLK。

6.2.4.4 RTC 时钟

RTC 模块能够在大多数时钟源关闭的状态下工作。RTC 时钟包括 RTC_SLOW_CLK 时钟和 RTC_FAST_CLK 时钟。

RTC_SLOW_CLK 和 RTC_FAST_CLK 的时钟源为低频时钟，其中：

- RTC_SLOW_CLK 时钟用于 RTC 计数器、RTC 看门狗和低功耗控制器，有三种可能的时钟源：
  - OSC_SLOW_CLK
  - RC_SLOW_CLK
  - RC_FAST_DIV_CLK
- RTC_FAST_CLK 用于 RTC 外设和传感器控制器，有两种可能的时钟源：
  - XTAL_CLK 的 2 分频时钟
  - RC_FAST_CLK 的 N 分频时钟

7 芯片 Boot 控制

7.1 概述

Strapping 管脚是指 ESP8684 芯片的特定管脚，可用于控制 ESP8684 芯片上电或硬件复位时的一些功能，包括：

- 控制 Boot 模式
- 控制 ROM 代码日志打印到 UART

ESP8684 共有两个 Strapping 管脚：

- GPIO8
- GPIO9

在上电复位、RTC 看门狗复位和欠压复位（请参考 6 复位和时钟 章节）过程中，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。Strapping 管脚锁存的状态可以通过软件从 GPIO_STRAPPING 中读取。

7.2 特性

- 共两个 Strapping 管脚：
  - GPIO8
  - GPIO9
- 可控制芯片 Boot 模式：
  - SPI Boot 模式
  - Download Boot 模式
- 控制 ROM 代码日志是否打印到 UART
- Strapping 管脚的锁存值可通过软件从 GPIO_STRAPPING 中读取

7.3 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 组合模式。

> **注意：** 请使用本章节所介绍的组合，其它组合可能会导致不可控结果。

7.3.1 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 7-1 所示。

表 7-1. 管脚默认上拉/下拉

管脚	默认值
GPIO8	N/A
GPIO9	上拉

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 ESP8684 上电复位时的 Strapping 管脚电平。复位释放后，Strapping 管脚和普通管脚功能相同。

7.3.2 Boot 模式控制

复位释放后，GPIO2、GPIO3、GPIO8 和 GPIO9 在复位时的值将共同决定 Boot 模式。表 7-2 列出了 GPIO2、GPIO3、GPIO8 和 GPIO9 的 Strapping 值及其对应的系统启动模式。

表 7-2. 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 模式下支持 UART Download Boot 下载方式。

[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 7-1。

在 Joint Download Boot 模式下，用户可通过 UART0 接口将二进制文件下载至 flash，或将二进制文件下载至 SRAM 并从 SRAM 中运行程序。

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 7-1。

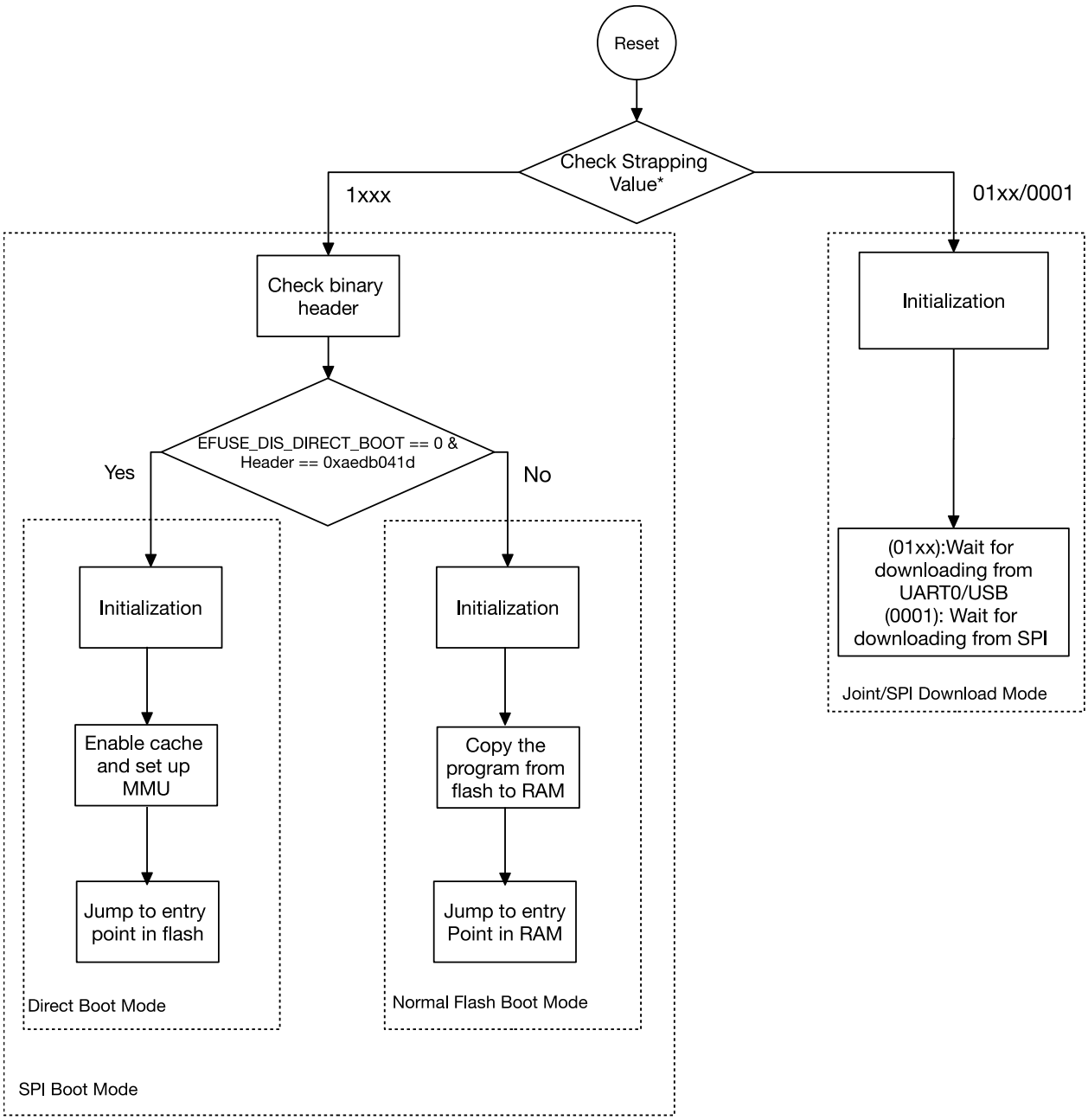


图 7-1. 芯片启动流程

下面几个寄存器可用于控制启动模式的具体行为：

- RTC_CNTL_FORCE_DOWNLOAD_BOOT

  软件可通过设置 RTC_CNTL_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 GPIO_STRAPPING[3:2] 的值从 “1x” 覆盖为 “01”。

- EFUSE_DIS_DOWNLOAD_MODE

  如果此 eFuse 设置为 1，则禁用 Joint Download Boot 模式。GPIO_STRAPPING 的值则不受 RTC_CNTL_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。

7.3.3 ROM 代码日志打印控制

系统在 SPI 启动模式下的早期阶段，GPIO8 与 EFUSE_UART_PRINT_CONTROL 一起控制 ROM 代码日志打印。

表 7-3. ROM 代码日志打印控制

eFuse[1]	GPIO8	ROM 代码日志打印
0	x	启动过程中，ROM 代码日志始终打印至 UART，此时 GPIO8 的值被忽略
1	0	启动过程中使能打印
	1	启动过程中关闭打印
2	0	启动过程中关闭打印
	1	启动过程中使能打印
3	x	启动过程中始终关闭打印，此时 GPIO8 的值被忽略

[1] eFuse: EFUSE_UART_PRINT_CONTROL

8 中断矩阵 (INTMTRX)

8.1 概述

ESP8684 中断矩阵将任一外部中断源单独映射到 ESP-RISC-V CPU 的任一外部中断上，以便在外设中断信号产生后，及时通知 CPU 进行处理。

ESP8684 有 43 个外部中断源，但 CPU 只支持 31 个中断。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*。

8.2 特性

中断矩阵具有如下特性：

- 接收 43 个外部中断源作为输入
- 生成 31 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
- 支持配置 CPU 的中断优先级、中断类型、中断阈值以及中断使能

中断矩阵的结构如图 8-1 所示。

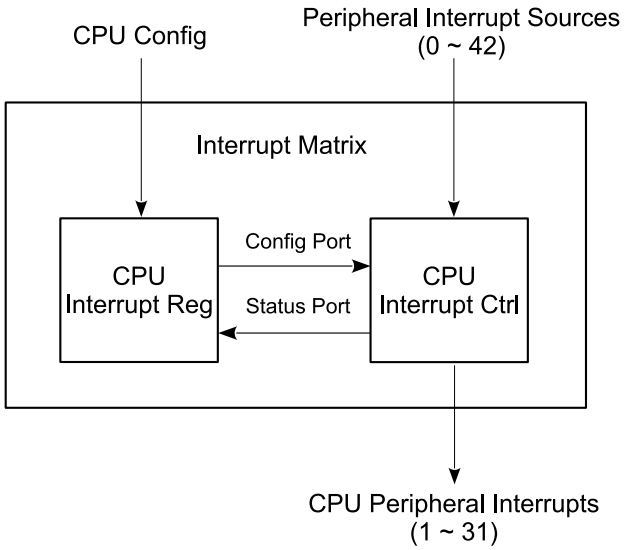


图 8-1. 中断矩阵结构图

8.3 功能描述

8.3.1 外部中断源

ESP8684 共有 43 个外部中断源。表 8-1 列出了所有外部中断源，以及对应的中断配置寄存器与中断状态寄存器。

- “索引”：表示外部中断源的索引号，范围：0 ~ 42
- “章节”：详细描述外部中断源的章节
- “中断源”：外部中断源名称
- “配置寄存器”：用于将外部中断源分配至 CPU 外部中断
- “状态寄存器”：用于读取中断源的中断状态
  - “状态寄存器 - 位”：表示在状态寄存器中的比特位置，用于记录相应中断源的状态
  - “状态寄存器 - 名称”：表示状态寄存器的名称

表 8-1. CPU 外部中断配置寄存器、外部中断状态寄存器、外部中断源

索引号	章节	中断源	配置寄存器	状态寄存器	
				位	名称
0	N/A	保留	保留	0	INTERRUPT_COREO_INTR_STATUS_0_REG
1	N/A	保留	保留	1	
2	N/A	保留	保留	2	
3	N/A	保留	保留	3	
4	N/A	保留	保留	4	
5	N/A	保留	保留	5	
6	N/A	保留	保留	6	
7	N/A	保留	保留	7	
8	N/A	保留	保留	8	
9	N/A	保留	保留	9	
10	N/A	保留	保留	10	
11	N/A	保留	保留	11	
12	N/A	保留	保留	12	
13	*IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)*	GPIO_PROCPU_INTR	INTERRUPT_COREO_GPIO_INTERRUPT_PRO_MAP_REG	13	
14	N/A	保留	保留	14	
15	N/A	保留	保留	15	
16	*SPI 控制器 (SPI)*	GPSPI2_INTR_2	INTERRUPT_COREO_SPI_INTR_2_MAP_REG	16	
17	*UART 控制器 (UART)*	UART_INTR	INTERRUPT_COREO_UART_INTR_MAP_REG	17	
18	*UART 控制器 (UART)*	UART1_INTR	INTERRUPT_COREO_UART1_INTR_MAP_REG	18	
19	*LED PWM 控制器 (LEDC)*	LEDC_INTR	INTERRUPT_COREO_LEDC_INT_MAP_REG	19	
20	*eFuse 控制器 (eFuse)*	EFUSE_INTR	INTERRUPT_COREO_EFUSE_INT_MAP_REG	20	
21	*低功耗管理 (RTC_CNTL)*	RTC_CNTL_INTR	INTERRUPT_COREO_RTC_CORE_INTR_MAP_REG	21	
22	*I2C 主机控制器 (I2C)*	I2C_EXTO_INTR	INTERRUPT_COREO_I2C_EXTO_INTR_MAP_REG	22	
23	*定时器组 (TIMG)*	TG_TO_INTR	INTERRUPT_COREO_TG_TO_INT_MAP_REG	23	
24	*定时器组 (TIMG)*	TG_WDT_INTR	INTERRUPT_COREO_TG_WDT_INT_MAP_REG	24	
25	N/A	保留	保留	25	
26	*系统定时器 (SYSTIMER)*	SYSTIMER_TARGET0_INTR	INTERRUPT_COREO_SYSTIMER_TARGET0_INT_MAP_REG	26	
27	*系统定时器 (SYSTIMER)*	SYSTIMER_TARGET1_INTR	INTERRUPT_COREO_SYSTIMER_TARGET1_INT_MAP_REG	27	
28	*系统定时器 (SYSTIMER)*	SYSTIMER_TARGET2_INTR	INTERRUPT_COREO_SYSTIMER_TARGET2_INT_MAP_REG	28	
29	N/A	保留	保留	29	
30	N/A	保留	保留	30	
31	N/A	保留	保留	31	
32	*片上传感器与模拟信号处理*	DIGTAL_ADC_INTR	INTERRUPT_COREO_APB_ADC_INT_MAP_REG	0	INTERRUPT_COREO_INTR_STATUS_1_REG

GoBack

索引号	章节	中断源	配置寄存器	状态寄存器	
				位	名称
33	通用 DMA 控制器 (GDMA)	GDMA_CH0_INTR	INTERRUPT_COREO_DMA_CH0_INT_MAP_REG	1	
34	SHA 加速器 (SHA)	SHA_INTR	INTERRUPT_COREO_SHA_INTR_MAP_REG	2	
35	ECC 硬件加速器 (ECC)	ECC_INTR	INTERRUPT_COREO_ECC_INTR_MAP_REG	3	
36	系统寄存器 (SYSTEM)	SW_INTR_0	INTERRUPT_COREO_CPU_INTR_FROM_CPU_0_MAP_REG	4	
37	系统寄存器 (SYSTEM)	SW_INTR_1	INTERRUPT_COREO_CPU_INTR_FROM_CPU_1_MAP_REG	5	
38	系统寄存器 (SYSTEM)	SW_INTR_2	INTERRUPT_COREO_CPU_INTR_FROM_CPU_2_MAP_REG	6	
39	系统寄存器 (SYSTEM)	SW_INTR_3	INTERRUPT_COREO_CPU_INTR_FROM_CPU_3_MAP_REG	7	
40	辅助调试 (ASSIST_DEBUG)	ASSIST_DEBUG_INTR	INTERRUPT_COREO_ASSIST_DEBUG_INTR_MAP_REG	8	
41	N/A	PERI_VIO_SIZE_INTR	INTERRUPT_COREO_PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP_REG	9	
42	N/A	保留	保留	10	

GoBack

8.3.2 CPU 中断

ESP8684 采用非 RISC-V 标准规范中断机制，CPU 共有 31 个中断号 (ID: 1 ~ 31)，每个中断：

- 优先级可设置为 1 ~ 15（数字越大优先级越高）；
- 可配置为高电平触发或者上升沿触发；
- 可通过设置中断阈值，屏蔽低优先级的中断。

说明：
CPU 中断的具体配置见章节 1 *ESP-RISC-V CPU*。

8.3.3 分配外部中断源至 CPU 外部中断

在本小节中，我们将使用以下术语描述中断矩阵相关操作：

- Source_*X*：代表某个外部中断源，其中 *X* 为中断源索引号，详见表 8-1。
- INTERRUPT_COREO_SOURCE_*X*_MAP_REG：外部中断源 (Source_*X*) 的中断映射配置寄存器。
- NUM_P：表示 CPU 中断 ID，范围：1 ~ 31。
- Interrupt_P：表示中断 ID 为 Num_P 的 CPU 中断。

8.3.3.1 分配一个外部中断源 Source_X 至 CPU 外部中断

将外部中断源 Source_X 对应的寄存器 INTERRUPT_COREO_SOURCE_*X*_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)。

8.3.3.2 分配多个外部中断源 Source_X*n* 至 CPU 外部中断

将各个中断源对应的寄存器 INTERRUPT_COREO_SOURCE_X*n*_MAP_REG 均配置成相同的 Num_P，即可将多个中断源 Source_X*n* 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，需要在中断服务程序中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*。

8.3.3.3 关闭 CPU 外部中断源 Source_X

将中断源对应的寄存器 INTERRUPT_COREO_SOURCE_*X*_MAP_REG 配置成 0，即可关闭外部中断源。

8.3.4 查询外部中断源当前的中断状态

读取寄存器 INTERRUPT_COREO_INTR_STATUS_*n*_REG（只读）中特定位的值可以获取 CPU 外部中断源当前的中断状态。寄存器 INTERRUPT_COREO_INTR_STATUS_*n*_REG 与外部中断源的对应关系如表 8-1 所示。

8.4 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 3 系统和存储器 中的表 3-3。

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
中断源映射寄存器			
INTERRUPT_COREO_GPIO_INTERRUPT_PRO_MAP_REG	GPIO_INTERRUPT_PRO 中断源映射寄存器	0x0034	R/W
INTERRUPT_COREO_SPI_INTR_2_MAP_REG	SPI_INTR_2 中断源映射寄存器	0x0040	R/W
INTERRUPT_COREO_UART_INTR_MAP_REG	UART_INTR 中断源映射寄存器	0x0044	R/W
INTERRUPT_COREO_UART1_INTR_MAP_REG	UART1_INTR 中断源映射寄存器	0x0048	R/W
INTERRUPT_COREO_LEDC_INT_MAP_REG	LEDC_INT 中断源映射寄存器	0x004C	R/W
INTERRUPT_COREO_EFUSE_INT_MAP_REG	EFUSE_INT 中断源映射寄存器	0x0050	R/W
INTERRUPT_COREO_RTC_CORE_INTR_MAP_REG	RTC_CORE_INTR 中断源映射寄存器	0x0054	R/W
INTERRUPT_COREO_I2C_EXTO_INTR_MAP_REG	I2C_EXTO_INTR 中断源映射寄存器	0x0058	R/W
INTERRUPT_COREO_TG_TO_INT_MAP_REG	TG_TO_INT 中断源映射寄存器	0x005C	R/W
INTERRUPT_COREO_TG_WDT_INT_MAP_REG	TG_WDT_INT 中断源映射寄存器	0x0060	R/W
INTERRUPT_COREO_SYSTIMER_TARGETO_INT_MAP_REG	SYSTIMER_TARGETO_INT 中断源映射寄存器	0x0068	R/W
INTERRUPT_COREO_SYSTIMER_TARGET1_INT_MAP_REG	SYSTIMER_TARGET1_INT 中断源映射寄存器	0x006C	R/W
INTERRUPT_COREO_SYSTIMER_TARGET2_INT_MAP_REG	SYSTIMER_TARGET2_INT 中断源映射寄存器	0x0070	R/W
INTERRUPT_COREO_APB_ADC_INT_MAP_REG	APB_ADC_INT 中断源映射寄存器	0x0080	R/W
INTERRUPT_COREO_DMA_CHO_INT_MAP_REG	DMA_CHO_INT 中断源映射寄存器	0x0084	R/W
INTERRUPT_COREO_SHA_INT_MAP_REG	SHA_INT 中断源映射寄存器	0x0088	R/W
INTERRUPT_COREO_ECC_INT_MAP_REG	ECC_INT 中断源映射寄存器	0x008C	R/W
INTERRUPT_COREO_CPU_INTR_FROM_CPU_0_MAP_REG	CPU_INTR_FROM_CPU_0 中断源映射寄存器	0x0090	R/W
INTERRUPT_COREO_CPU_INTR_FROM_CPU_1_MAP_REG	CPU_INTR_FROM_CPU_1 中断源映射寄存器	0x0094	R/W
INTERRUPT_COREO_CPU_INTR_FROM_CPU_2_MAP_REG	CPU_INTR_FROM_CPU_2 中断源映射寄存器	0x0098	R/W
INTERRUPT_COREO_CPU_INTR_FROM_CPU_3_MAP_REG	CPU_INTR_FROM_CPU_3 中断源映射寄存器	0x009C	R/W
INTERRUPT_COREO_ASSIST_DEBUG_INTR_MAP_REG	ASSIST_DEBUG_INTR 中断源映射寄存器	0x00A0	R/W
INTERRUPT_COREO_SIZE_INTR_MAP_REG	PIF_PMS_MONITOR_VIOLATE_SIZE_INTR 中断源映射寄存器	0x00A4	R/W

名称	描述	地址	访问
中断源状态寄存器			
INTERRUPT_COREO_INTR_STATUS_0_REG	中断源状态寄存器 0	0x00AC	RO
INTERRUPT_COREO_INTR_STATUS_1_REG	中断源状态寄存器 1	0x00B0	RO
时钟寄存器			
INTERRUPT_COREO_CLOCK_GATE_REG	时钟寄存器	0x00B4	R/W
CPU 中断寄存器			
INTERRUPT_COREO_CPU_INT_ENABLE_REG	CPU 中断使能配置寄存器	0x00B8	R/W
INTERRUPT_COREO_CPU_INT_TYPE_REG	CPU 中断类型配置寄存器	0x00BC	R/W
INTERRUPT_COREO_CPU_INT_CLEAR_REG	CPU 中断清零寄存器	0x00C0	R/W
INTERRUPT_COREO_CPU_INT_EIP_STATUS_REG	CPU 中断阻塞状态寄存器	0x00C4	RO
INTERRUPT_COREO_CPU_INT_PRI_1_REG	CPU 中断 1 的优先级配置寄存器	0x00CC	R/W
INTERRUPT_COREO_CPU_INT_PRI_2_REG	CPU 中断 2 的优先级配置寄存器	0x00D0	R/W
INTERRUPT_COREO_CPU_INT_PRI_3_REG	CPU 中断 3 的优先级配置寄存器	0x00D4	R/W
INTERRUPT_COREO_CPU_INT_PRI_4_REG	CPU 中断 4 的优先级配置寄存器	0x00D8	R/W
INTERRUPT_COREO_CPU_INT_PRI_5_REG	CPU 中断 5 的优先级配置寄存器	0x00DC	R/W
INTERRUPT_COREO_CPU_INT_PRI_6_REG	CPU 中断 6 的优先级配置寄存器	0x00E0	R/W
INTERRUPT_COREO_CPU_INT_PRI_7_REG	CPU 中断 7 的优先级配置寄存器	0x00E4	R/W
INTERRUPT_COREO_CPU_INT_PRI_8_REG	CPU 中断 8 的优先级配置寄存器	0x00E8	R/W
INTERRUPT_COREO_CPU_INT_PRI_9_REG	CPU 中断 9 的优先级配置寄存器	0x00EC	R/W
INTERRUPT_COREO_CPU_INT_PRI_10_REG	CPU 中断 10 的优先级配置寄存器	0x00F0	R/W
INTERRUPT_COREO_CPU_INT_PRI_11_REG	CPU 中断 11 的优先级配置寄存器	0x00F4	R/W
INTERRUPT_COREO_CPU_INT_PRI_12_REG	CPU 中断 12 的优先级配置寄存器	0x00F8	R/W
INTERRUPT_COREO_CPU_INT_PRI_13_REG	CPU 中断 13 的优先级配置寄存器	0x00FC	R/W
INTERRUPT_COREO_CPU_INT_PRI_14_REG	CPU 中断 14 的优先级配置寄存器	0x0100	R/W
INTERRUPT_COREO_CPU_INT_PRI_15_REG	CPU 中断 15 的优先级配置寄存器	0x0104	R/W
INTERRUPT_COREO_CPU_INT_PRI_16_REG	CPU 中断 16 的优先级配置寄存器	0x0108	R/W
INTERRUPT_COREO_CPU_INT_PRI_17_REG	CPU 中断 17 的优先级配置寄存器	0x010C	R/W
INTERRUPT_COREO_CPU_INT_PRI_18_REG	CPU 中断 18 的优先级配置寄存器	0x0110	R/W
INTERRUPT_COREO_CPU_INT_PRI_19_REG	CPU 中断 19 的优先级配置寄存器	0x0114	R/W

GoBack

名称	描述	地址	访问
INTERRUPT_COREO_CPU_INT_PRI_20_REG	CPU 中断 20 的优先级配置寄存器	0x0118	R/W
INTERRUPT_COREO_CPU_INT_PRI_21_REG	CPU 中断 21 的优先级配置寄存器	0x011C	R/W
INTERRUPT_COREO_CPU_INT_PRI_22_REG	CPU 中断 22 的优先级配置寄存器	0x0120	R/W
INTERRUPT_COREO_CPU_INT_PRI_23_REG	CPU 中断 23 的优先级配置寄存器	0x0124	R/W
INTERRUPT_COREO_CPU_INT_PRI_24_REG	CPU 中断 24 的优先级配置寄存器	0x0128	R/W
INTERRUPT_COREO_CPU_INT_PRI_25_REG	CPU 中断 25 的优先级配置寄存器	0x012C	R/W
INTERRUPT_COREO_CPU_INT_PRI_26_REG	CPU 中断 26 的优先级配置寄存器	0x0130	R/W
INTERRUPT_COREO_CPU_INT_PRI_27_REG	CPU 中断 27 的优先级配置寄存器	0x0134	R/W
INTERRUPT_COREO_CPU_INT_PRI_28_REG	CPU 中断 28 的优先级配置寄存器	0x0138	R/W
INTERRUPT_COREO_CPU_INT_PRI_29_REG	CPU 中断 29 的优先级配置寄存器	0x013C	R/W
INTERRUPT_COREO_CPU_INT_PRI_30_REG	CPU 中断 30 的优先级配置寄存器	0x0140	R/W
INTERRUPT_COREO_CPU_INT_PRI_31_REG	CPU 中断 31 的优先级配置寄存器	0x0144	R/W
INTERRUPT_COREO_CPU_INT_THRESH_REG	CPU 中断阈值配置寄存器	0x0148	R/W
版本寄存器			
INTERRUPT_COREO_INTERRUPT_DATE_REG	版本控制寄存器	0x07FC	R/W

GoBack

8.5 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 3 系统和存储器 中的表 3-3。

Register 8.1. INTERRUPT_COREO_*GPIO_INTERRUPT_PRO_MAP*_REG (0x0034)

Register 8.2. INTERRUPT_COREO_*SPI_INTR_2_MAP*_REG (0x0040)

Register 8.3. INTERRUPT_COREO_*UART_INTR_MAP*_REG (0x0044)

Register 8.4. INTERRUPT_COREO_*UART1_INTR_MAP*_REG (0x0048)

Register 8.5. INTERRUPT_COREO_*LEDC_INT_MAP*_REG (0x004C)

Register 8.6. INTERRUPT_COREO_*EFUSE_INT_MAP*_REG (0x0050)

Register 8.7. INTERRUPT_COREO_*RTC_CORE_INTR_MAP*_REG (0x0054)

Register 8.8. INTERRUPT_COREO_*I2C_EXT0_INTR_MAP*_REG (0x058)

Register 8.9. INTERRUPT_COREO_*TG_T0_INT_MAP*_REG (0x005C)

Register 8.10. INTERRUPT_COREO_*TG_WDT_INT_MAP*_REG (0x0060)

Register 8.11. INTERRUPT_COREO_*SYSTIMER_TARGET0_INT_MAP*_REG (0x0068)

Register 8.12. INTERRUPT_COREO_*SYSTIMER_TARGET1_INT_MAP*_REG (0x006C)

Register 8.13. INTERRUPT_COREO_*SYSTIMER_TARGET2_INT_MAP*_REG (0x0070)

Register 8.14. INTERRUPT_COREO_*APB_ADC_INT_MAP*_REG (0x0080)

Register 8.15. INTERRUPT_COREO_*DMA_CH0_INT_MAP*_REG (0x0084)

Register 8.16. INTERRUPT_COREO_*SHA_INT_MAP*_REG (0x0088)

Register 8.17. INTERRUPT_COREO_*ECC_INT_MAP*_REG (0x008C)

Register 8.18. INTERRUPT_COREO_*CPU_INTR_FROM_CPU_0_MAP*_REG (0x0090)

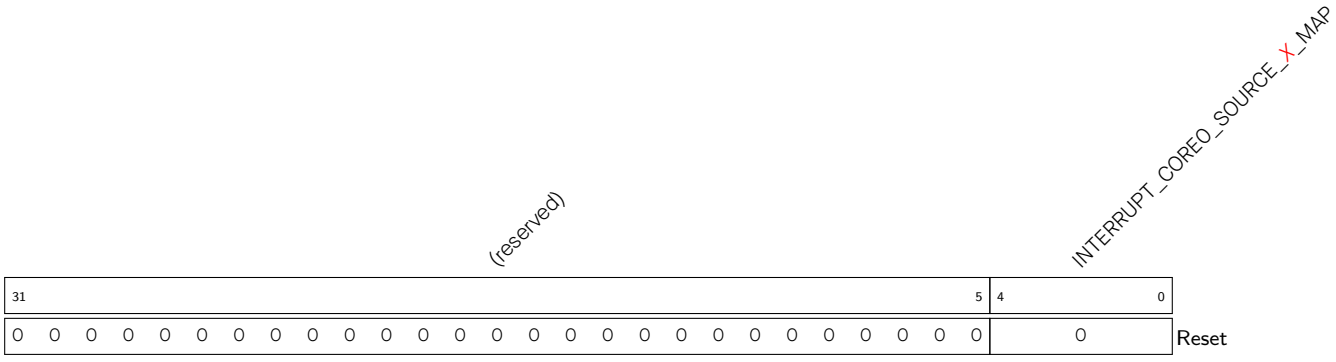
Register 8.19. INTERRUPT_COREO_*CPU_INTR_FROM_CPU_1_MAP*_REG (0x0094)

Register 8.20. INTERRUPT_COREO_*CPU_INTR_FROM_CPU_2_MAP*_REG (0x0098)

Register 8.21. INTERRUPT_COREO_*CPU_INTR_FROM_CPU_3_MAP*_REG (0x009C)

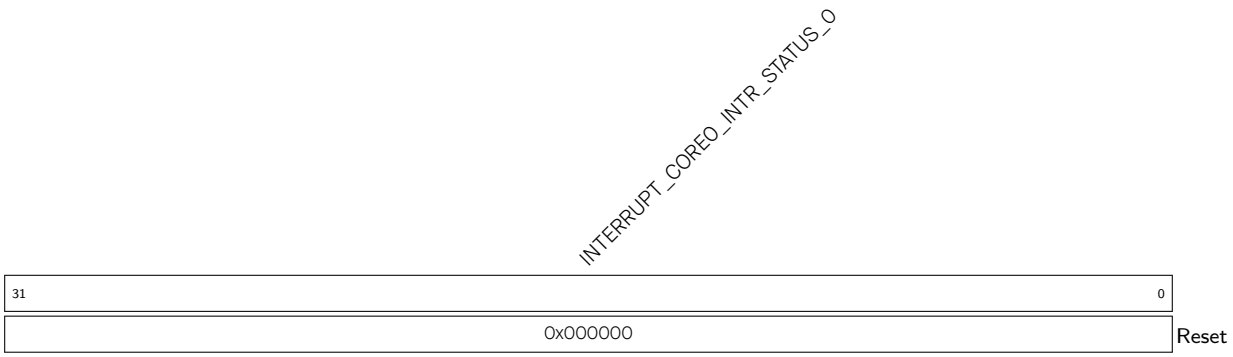
Register 8.22. INTERRUPT_COREO_*ASSIST_DEBUG_INTR_MAP*_REG (0x00A0)

Register 8.23. INTERRUPT_COREO_*PIF_PMS_MONITOR_VIOLATE_SIZE_INTR_MAP*_REG (0x00A4)



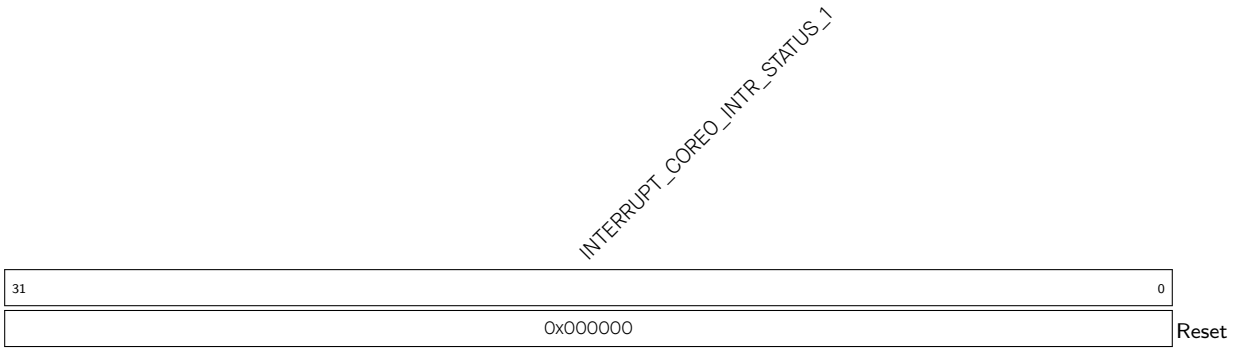
INTERRUPT_COREO_SOURCE_*X*_MAP 将中断源 SOURCE_*X* 映射至 CPU 外部中断。中断源 SOURCE_*X* 见表 8-1。(R/W)

Register 8.24. INTERRUPT_COREO_INTR_STATUS_0_REG (0x00AC)



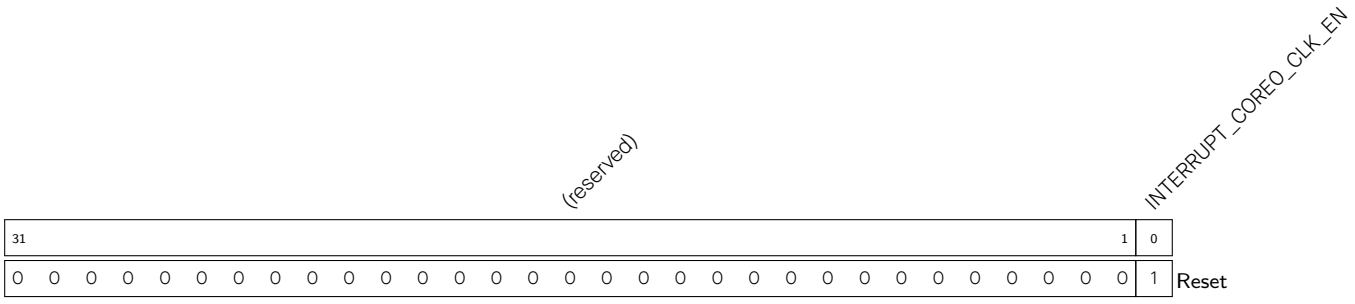
INTERRUPT_COREO_INTR_STATUS_0 用于存储外部中断源的状态，每一位均代表一个外部中断源的状态，对应中断编号源：0 ~ 31。如果对应的位为 1，则表示该中断源触发了中断。(RO)

Register 8.25. INTERRUPT_COREO_INTR_STATUS_1_REG (0x00B0)



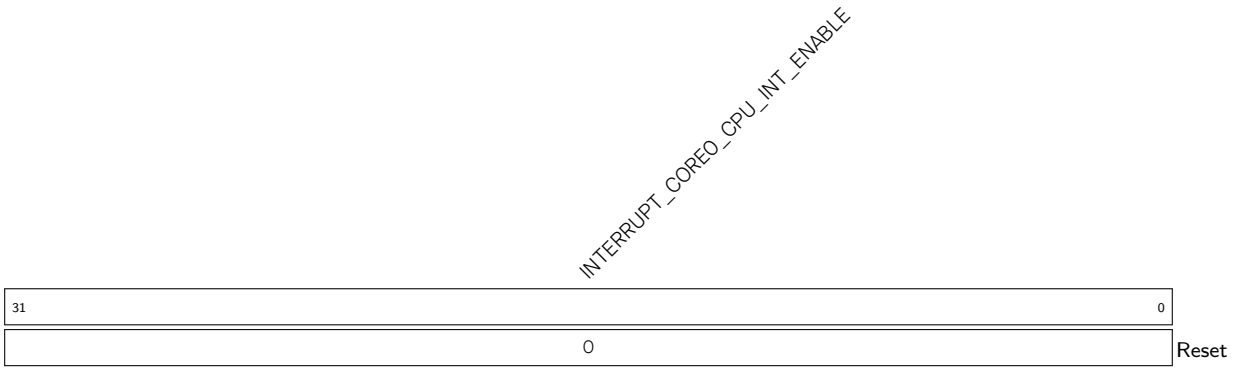
INTERRUPT_COREO_INTR_STATUS_1 用于存储外部中断源的状态，每一位均代表一个外部中断源的状态，对应中断编号源：32 ~ 42。如果对应的位为 1，则表示该中断源触发了中断。(RO)

Register 8.26. INTERRUPT_COREO_CLOCK_GATE_REG (0x00B4)



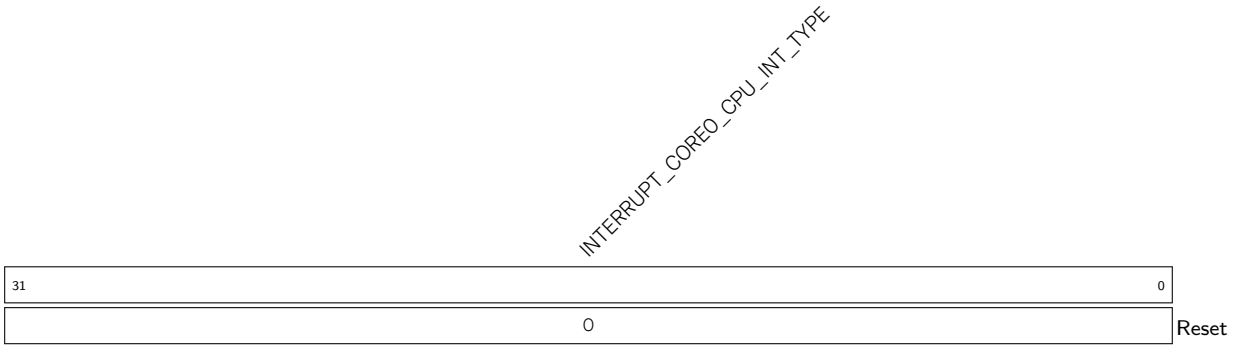
INTERRUPT_COREO_CLK_EN 置 1 强制使能中断寄存器的时钟门控。(R/W)

Register 8.27. INTERRUPT_CORE0_CPU_INT_ENABLE_REG (0x00B8)



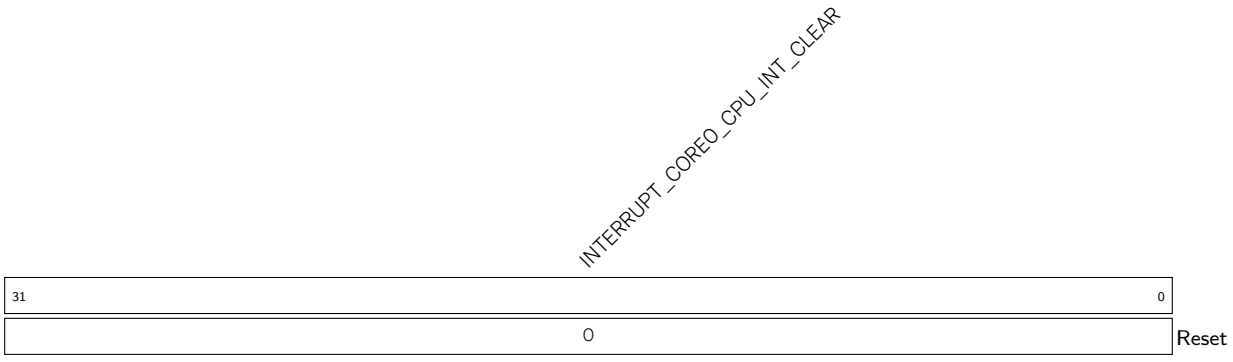
INTERRUPT_CORE0_CPU_INT_ENABLE 在相应位写 1，即可使能对应 CPU 中断。更多配置信息，见章节 1 *ESP-RISC-V CPU*。(R/W)

Register 8.28. INTERRUPT_CORE0_CPU_INT_TYPE_REG (0x00BC)



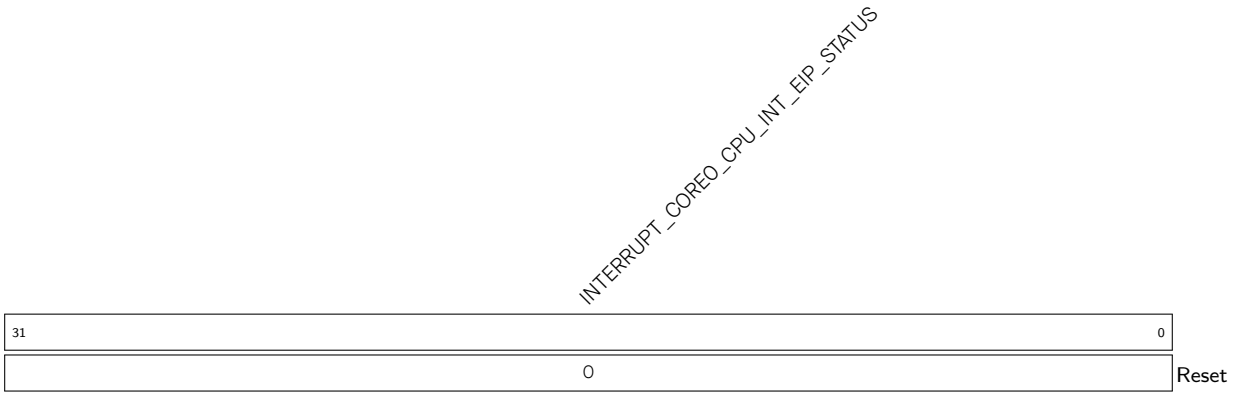
INTERRUPT_CORE0_CPU_INT_TYPE 配置 CPU 中断类型。0：电平触发；1：边沿触发。更多配置信息，见章节 1 *ESP-RISC-V CPU*。(R/W)

Register 8.29. INTERRUPT_CORE0_CPU_INT_CLEAR_REG (0x00C0)



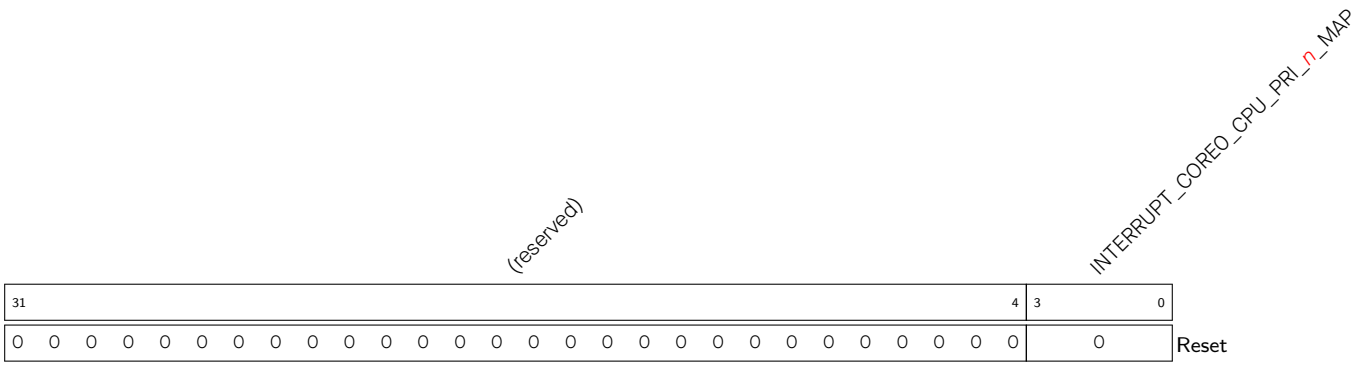
INTERRUPT_CORE0_CPU_INT_CLEAR 在相应位写 1，即可清除对应的 CPU 中断。更多配置信息，见章节 1 *ESP-RISC-V CPU*。(R/W)

Register 8.30. INTERRUPT_COREO_CPU_INT_EIP_STATUS_REG (0x00C4)



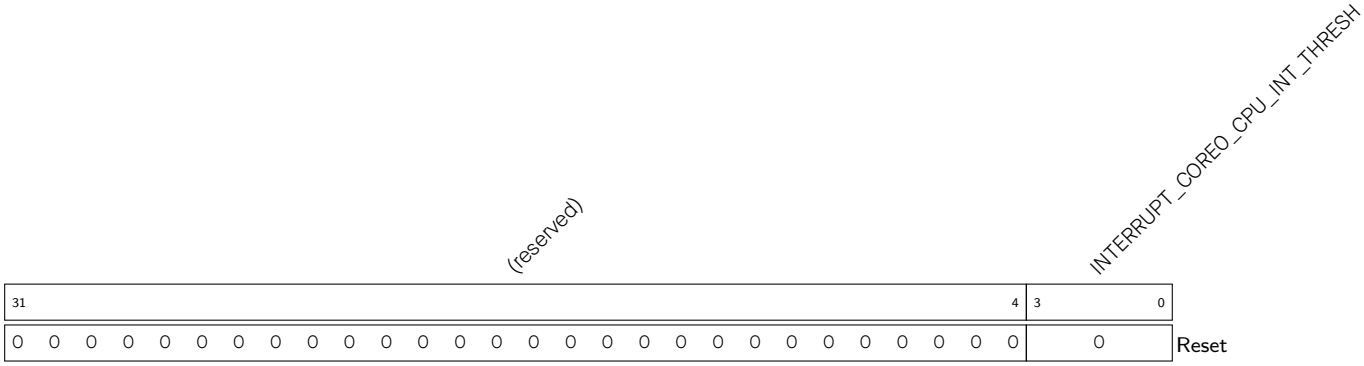
INTERRUPT_COREO_CPU_INT_EIP_STATUS 用于存储 CPU 中断的阻塞状态。更多信息，请参考章节 1 *ESP-RISC-V CPU*。(RO)

Register 8.31. INTERRUPT_COREO_CPU_INT_PRI_*n*_REG (*n*: 1 - 31)(0x00C8 + 0x4**n*)



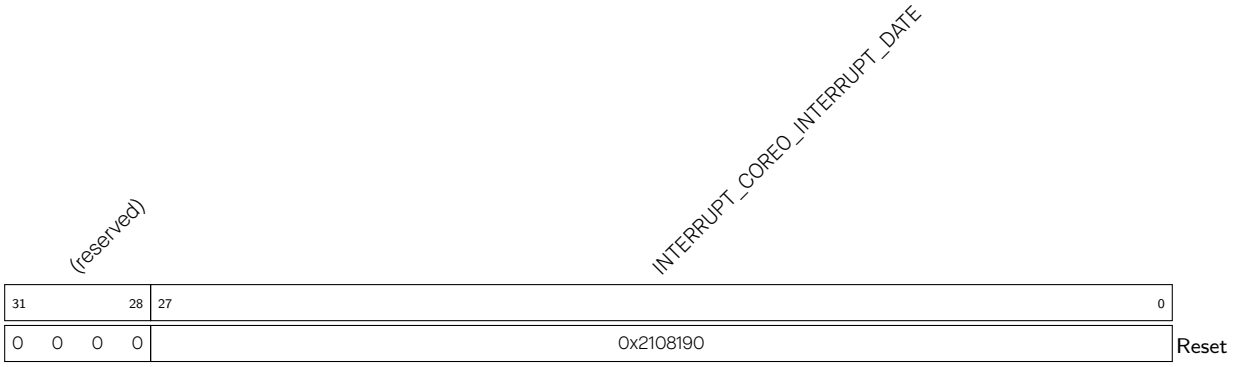
INTERRUPT_COREO_CPU_PRI_*n*_MAP 用于设置 CPU 中断 *n* 的优先级，可配置为 1 ~ 15。数字越大，优先级越高。更多配置信息，见章节 1 *ESP-RISC-V CPU*。(R/W)

Register 8.32. INTERRUPT_COREO_CPU_INT_THRESH_REG (0x0148)



INTERRUPT_COREO_CPU_INT_THRESH 用于设置 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。更多配置信息，见章节 1 *ESP-RISC-V CPU*。(R/W)

Register 8.33. INTERRUPT_COREO_INTERRUPT_DATE_REG (0x07FC)



INTERRUPT_COREO_INTERRUPT_DATE 版本寄存器。(R/W)

9 低功耗管理 (RTC_CNTL)

9.1 概述

ESP8684 拥有一个先进的电源管理单元，灵活打开或关闭芯片的不同电源域，协助客户在芯片工作性能、功耗控制和唤醒延迟之间实现最佳平衡。为了便利用户的使用，ESP8684 定义了四种最常见的电源域设置组合，对应四种预设功耗模式，可满足用户的常见场景需求，但也同时支持用户对某个电源域的独立控制，以满足一些复杂场景的功耗需求。

9.2 主要特性

ESP8684 的低功耗管理有如下特性：

- 4 种预设功耗模式，可满足多种典型应用场景需求
- 8 个 32 位保留寄存器 (retention register)

在本章节中，我们将首先介绍 ESP8684 低功耗管理的工作过程，然后介绍芯片的预设低功耗工作模式。

9.3 功能描述

ESP8684 的低功耗管理主要由以下模块实现：

- 功耗管理单元 (PMU)：控制向以下三大类电源域供电：
  - 实时控制器 RTC 类
  - 模拟类
  - 数字类

  有关以上三大类中所有电源域的完整列表，请见章节 9.4.1。
- 电源隔离单元：保证各电源域的独立工作，防止掉电电源域影响其他电源域的工作；
- 低功耗时钟：为低功耗模式下工作的电源域提供时钟信号；
- RTC 定时器：一个工作在 RTC 时钟下的 “always-on” 的定时器，可记录特定事件的发生时刻；
- 8 个 32 位 “always-on” 保留寄存器：即这 8 个寄存器永远处于工作状态，不受 Deep-sleep 等低功耗模式的影响，可用于存储一些不能丢失的数据。
- 6 个 “always-on” 管脚：即这 6 个管脚永远处于工作状态，不受 deep-sleep 等低功耗模式的影响，可用作低功耗模式下的唤醒源（详见第 9.4.3 节），也作为正常 GPIO 使用（详见 *5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)* 章节）。
- 调压器：调节向不同电源域的供电电压。

ESP8684 低功耗管理的原理图可见图 9-1。

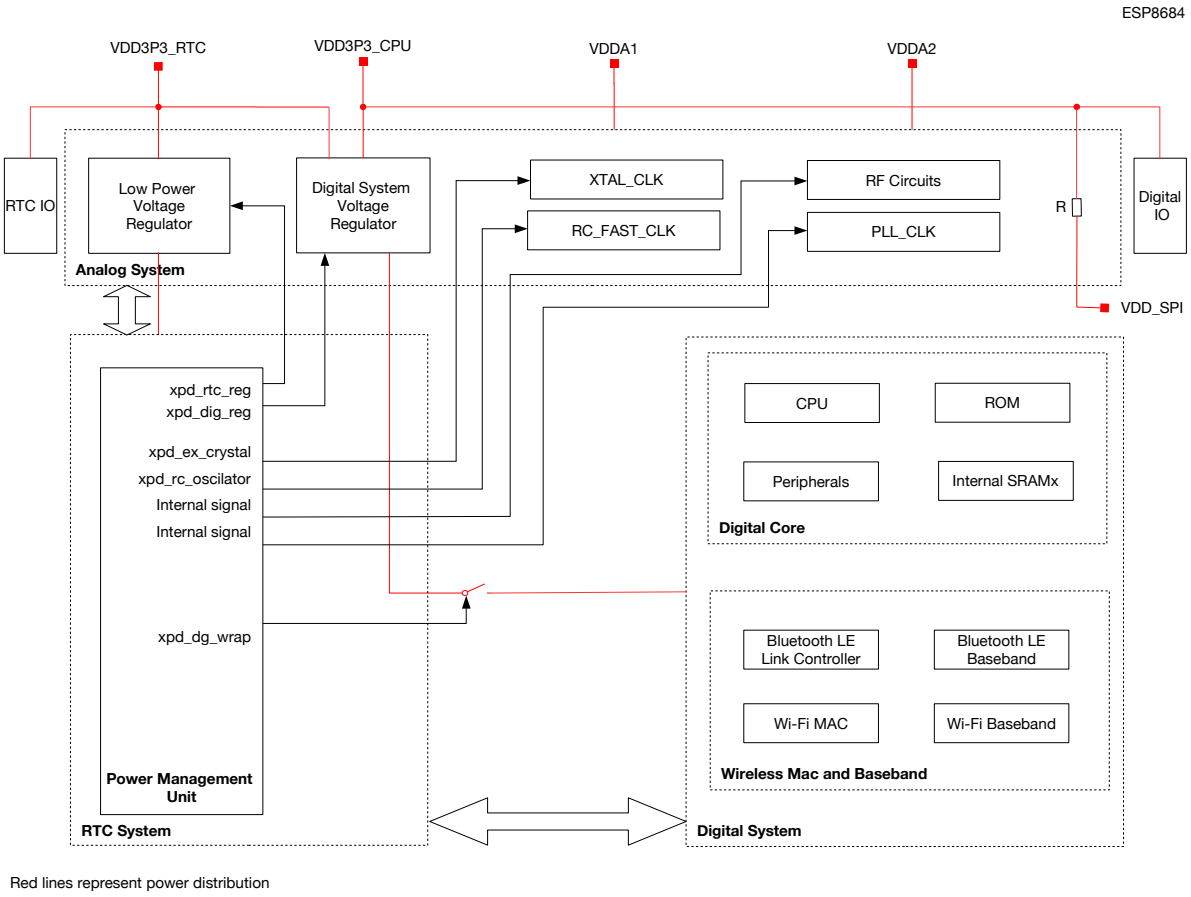


图 9-1. 低功耗管理原理图

说明：

- 上图中的主要电源域已用虚线框出。有关各电源域的具体描述，请见第 9.4.1 节。
- 上图中所有开关均由寄存器控制，详见 RTC_CNTL_DIG_PWC_REG。
- 上图中开关之外的信号描述如下：
  - xpd_rtc_reg:
    - 当 RTC_CNTL_REGULATOR_FORCE_PU 置 1 时，低功耗调压器常开；
    - 否则，低功耗调压器在芯片进入 Light-sleep 和 Deep-sleep 时关闭。此时，RTC 电路由一个极低功耗的内置电源供电。
  - xpd_dig_reg:
    - 当 RTC_CNTL_DG_WRAP_PD_EN 使能时，数字系统调压器在芯片进入 Light-sleep 和 Deep-sleep 时关闭；
    - 否则，数字系统调压器常开。
  - xpd_ex_crystal:
    - 当 RTC_CNTL_XTL_FORCE_PU 置 1 时，外部主晶振常开；
    - 否则，外部主晶振在芯片进入 Light-sleep 和 Deep-sleep 时关闭。
  - xpd_rc_oscilator:
    - 当 RTC_CNTL_FOSC_FORCE_PU 置 1 时，快速 RC 振荡器常开；
    - 否则，快速 RC 振荡器在芯片进入 Light-sleep 和 Deep-sleep 时关闭。
  - 其他信号不对客户开放。如有具体需求，请联系我们的 销售人员。

9.3.1 功耗管理单元 (PMU)

ESP8684 功耗管理单元可以控制向不同电源域的供电，其主要组成部分包括：

- RTC 主状态机 (RTC Main State Machine)：产生电源门控、时钟门控和复位信号。
- 电源控制器 (Power Controller)：根据 RTC 主状态机产生的电源门控信号，打开或关闭各电源域。
- 睡眠和唤醒控制器 (Sleep Controller, Wakeup Controller)：向 RTC 主状态机发送睡眠或唤醒请求。
- 时钟控制器 (Clock Controller)：选择并打开或关闭时钟源。
- 保护定时器 (Protection Timer)：控制主状态机切换状态的等待时间。

在 ESP8684 的电源管理单元中，睡眠和唤醒控制器向 RTC 主状态机发送睡眠或唤醒请求，RTC 主状态机接着产生电源门控、时钟门控和复位信号。此后，电源控制器和时钟控制器会根据 RTC 主状态机产生的信号，打开或关闭不同的电源域和时钟信号，从而让芯片进入或退出低功耗模式。电源管理单元的主要工作流程可见图 9-2。

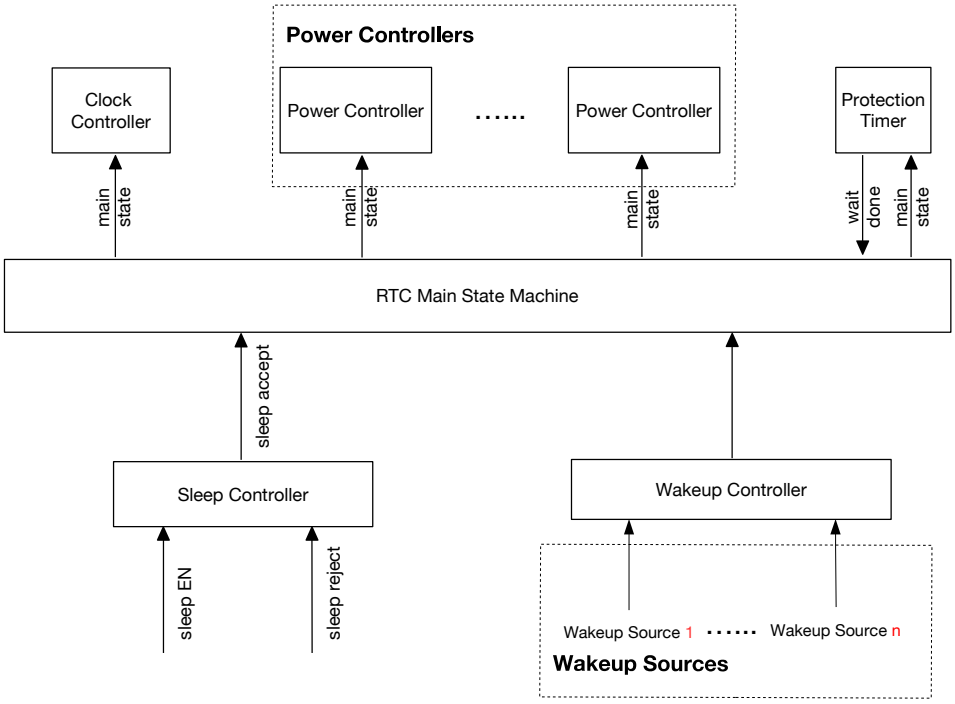


图 9-2. 电源管理单元的主要工作流程

说明：

1. 每个电源控制器均可控制一个具体的电源域。因此，有关具体电源控制器的完整列表，具体请见第 9.4.1 节。
2. 有关各唤醒源的具体描述，请见表 9-4。

9.3.2 低功耗时钟

通常情况下，当 ESP8684 处于低功耗模式下，芯片的外部主晶振 XTAL_CLK 和 PLL 时钟 (PLL_CLK) 将被断电以降低功耗，但低功耗时钟仍保持开启，为芯片的低功耗管理系统提供时钟，以确保芯片在低功耗模式下的正常工作。

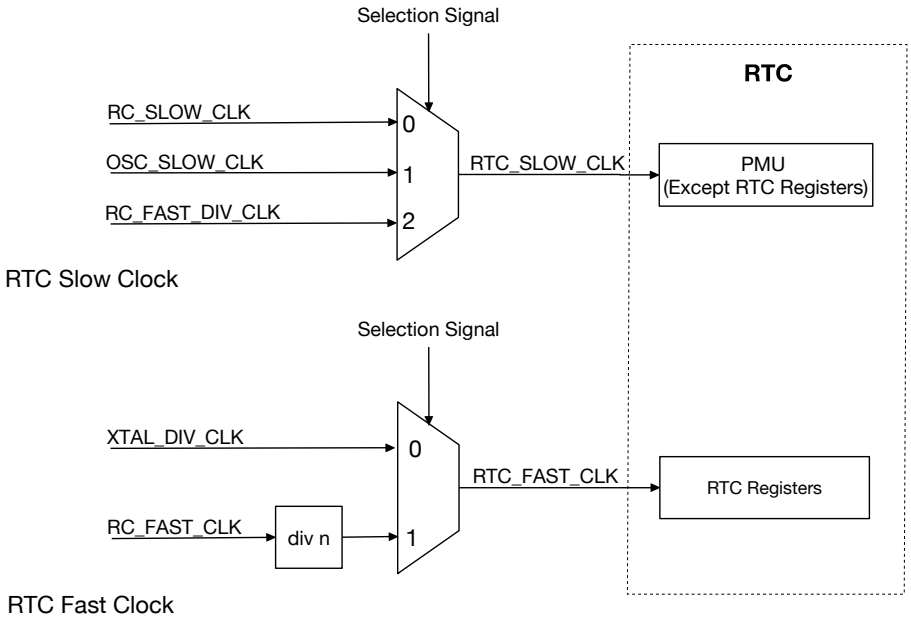


图 9-3. RTC_SLOW_CLOCK 和 RTC_FAST_CLOCK

表 9-1. 低功耗时钟

时钟类型	可选时钟源	时钟选择寄存器	作用电源域
RTC_SLOW_CLK	OSC_SLOW_CLK	RTC_CNTL_ANA_CLK_RTC_SEL	功耗管理系统（RTC 寄存器除外）
	RC_FAST_DIV_CLK		
	RC_SLOW_CLK (default)		
RTC_FAST_CLK	RC_FAST_CLK divided by n (default)	RTC_CNTL_FAST_CLK_RTC_SEL	RTC 寄存器
	XTAL_DIV_CLK		

更多有关时钟的描述，请见章节 6 复位和时钟。

9.3.3 定时器

ESP8684 的低功耗管理使用 RTC 定时器。RTC 定时器是一个 48 位的可读计数器，可通过配置，使用 RTC 慢速时钟记录以下任一事件发生的时刻。更多详情，请见表 9-2。

表 9-2. RTC 定时器的触发条件

使能条件	描述
RTC_CNTL_TIMER_XTL_OFF	1. RTC 主状态机关闭，或 2. 打开 XTAL_CLK 时均触发。
RTC_CNTL_TIMER_SYS_STALL	CPU 进入或退出 stall 状态时触发。该设置可保证 SYS_TIMER 的时间连续性。
RTC_CNTL_TIMER_SYS_RST	系统复位时触发。
RTC_CNTL_TIME_UPDATE	配置寄存器 RTC_CNTL_TIME_UPDATE 时触发。该触发由 CPU 产生（比如用户）。

RTC 定时器会在每次触发时更新两组寄存器。其中第一组寄存器记录本次触发的时间，第二组寄存器记录之前触发的时间。这两组寄存器的具体情况见下：

- 寄存器组 0 用于记录 RTC 定时器在当前触发下的计数值。
  - RTC_CNTL_TIME_HIGH0_REG
  - RTC_CNTL_TIME_LOW0_REG
- 寄存器组 1 用于记录 RTC 定时器在上一次触发下的计数值。
  - RTC_CNTL_TIME_HIGH1_REG
  - RTC_CNTL_TIME_LOW1_REG

每次有新的触发，上一次触发时的记录将从寄存器组 0 移至寄存器组 1（寄存器组 1 中之前的记录将被覆盖），而本次触发的记录将存储在寄存器组 0。因此，RTC 定时器最多可同时记录两次触发的值。

值得注意的是，除上电复位外的其余任何复位 / 睡眠均不会使 RTC 定时器停止或复位。

此外，RTC 定时器还能用作唤醒源。更多详情，请见第 9.4.3 节。

9.3.4 调压器

ESP8684 共有两个调压器，负责调节向不同电源域的供电：

- 数字系统调压器：负责数字类电源域；
- 低功耗调压器：负责 RTC 类电源域；

> **说明：**
> 更多有关不同电源域的描述，请见第 9.4.1 节。

9.3.4.1 数字系统调压器

ESP8684 的内置数字系统调压器可以将外部电源电压（通常为 3.3 V）转换为 1.1 V，支持数字类电源域的正常工作。该调压器主要由 xpd_dig_reg 信号控制，详见 9-1。具体结构示意图可见下方图 9-4。

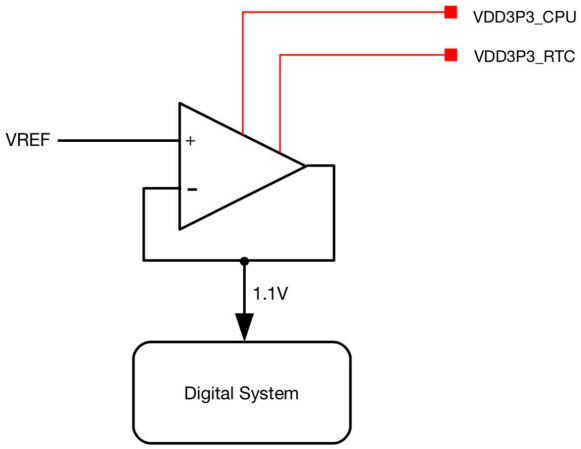


图 9-4. 数字系统调压器

9.3.4.2 低功耗调压器

ESP8684 的内置低功耗调压器可以将外部电源电压（通常为 3.3 V）转换为 1.1 V，支持 RTC 类电源域的正常工作。当管脚 CHIP_EN 为高电平时，低功耗调压器无法关闭。否则，低功耗调压器在芯片进入 Light-sleep 和 Deep-sleep 时关闭。此时，RTC 电路由一个极低功耗的内置电源供电。具体结构示意图可见下方图 9-5。

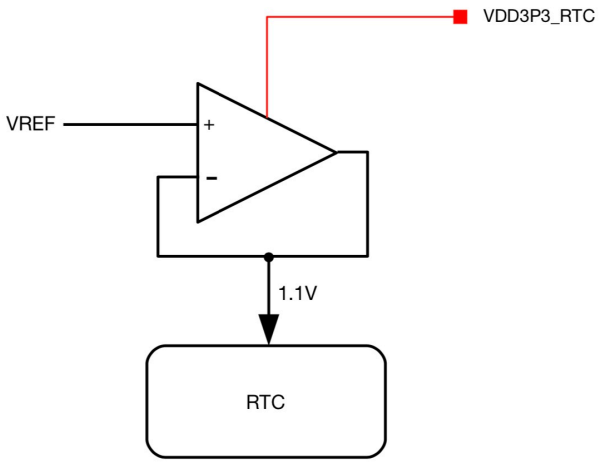


图 9-5. 低功耗调压器

9.3.4.3 欠压检测器

ESP8684 的欠压检测器可以检查管脚 VDDA, VDDA3P3, VDD3P3_RTC 和 VDD3P3 _CPU 的电压，在电压快速下落至预设阈值（默认为 2.7 V）以下时发出触发信号，并在触发信号持续一定时间后进行芯片或系统复位，从而关闭部分耗电模块（比如 LNA 和 PA 等），为数字模块争取更多时间，用以保存、转移重要数据。

欠压检测器的功耗非常低，在芯片开启时将永远保持开启。ESP8684 欠压检测器的具体结构示意图可见下方图 9-6。

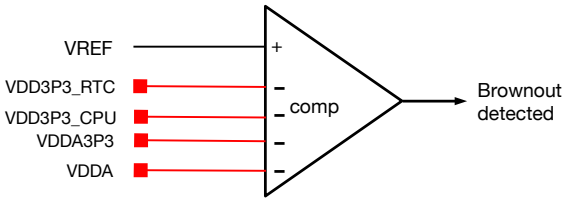


图 9-6. 欠压检测器

RTC_CNTL_BROWN_OUT_DET 可用于指示欠压检测器的输出电平，默认为低电平，可在检测管脚电压下降至阈值以下时跳至高电平。

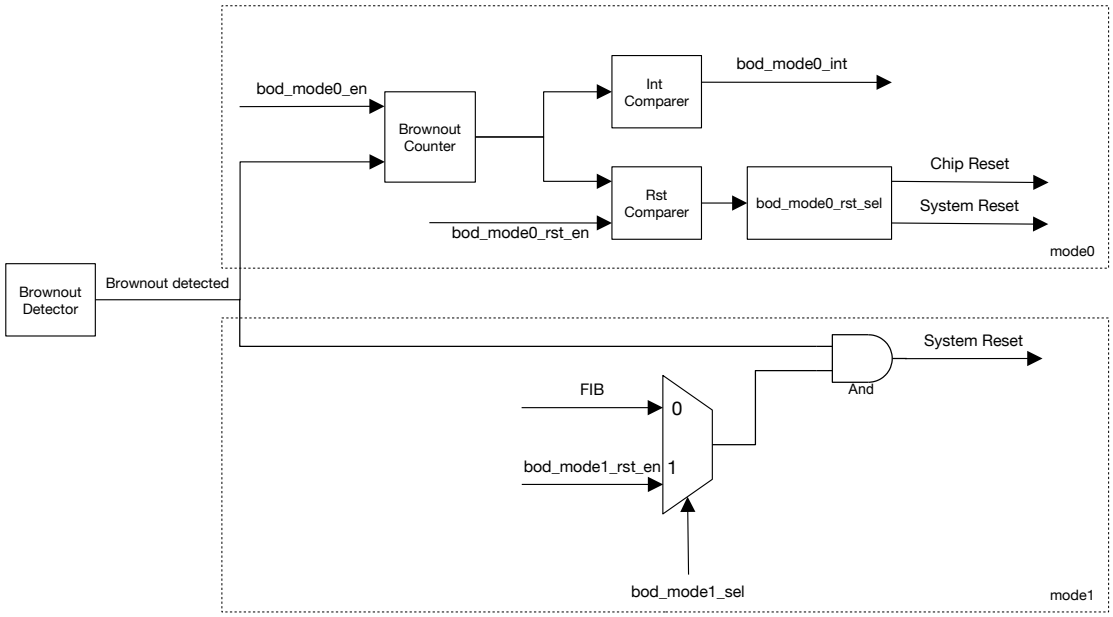


图 9-7. 欠压处理

如上方图 9-7 所示，欠压检测器可以根据用户配置，采用以下两种方式之一处理检测到的欠压信号：

- mode0：当欠压计数器达到中断比较器中配置的阈值（通过 RTC_CNTL_BROWN_OUT_INT_WAIT 配置）时触发中断。
  - 此外，还可以在 bod_mode0_rst_en（通过 RTC_CNTL_BROWN_OUT_RST_ENA 配置）使能的条件下，在欠压计数器达到复位比较器中配置的阈值（通过 RTC_CNTL_BROWN_OUT_RST_WAIT 配置）时触发复位，具体复位方式由 rst_sel 决定（通过 RTC_CNTL_BROWN_OUT_RST_SEL 配置）：
    * 0：芯片复位
    * 1：系统复位

    更多有关复位的信息，请见章节 6 复位和时钟。
- mode1：直接进行系统复位

如何选择处理方式：

- mode0：设置 bod_mode0_en 信号（配置 RTC_CNTL_BROWN_OUT_ENA）。
- mode1：取决于 bod_mode1_sel 信号
  - 0：设置 bod_mode1_rst_en 信号（配置 RTC_CNTL_BROWN_OUT_ANA_RST_EN）
  - 1：由 FIB 总线决定
- 注意，当同时使能 mode0 和 mode1 时，仅 mode1 生效。

9.4 功耗模式管理

9.4.1 电源域

ESP8684 共有三大类共 6 个电源域：

- RTC 类
  - 功耗管理单元（含 RTC 计时器和 Always-on 保留寄存器等）
- 数字类
  - 数字系统（包括数字内核和 Wireless 数字电路）
- 模拟类
  - RC_FAST_CLK
  - XTAL_CLK
  - PLL_CLK
  - RF 电路

9.4.2 预设功耗模式

如上文所示，ESP8684 定义了四种最常见的电源域设置组合，对应四种预设功耗模式，可满足用户的常见场景需求，详见表 9-3。

表 9-3. 预设功耗模式

功耗模式	PMU	数字系统	RC_FAST_CLK	XTAL_CLK	PLL_CLK	RF 电路
Active	ON	ON	ON	ON	ON	ON
Modem-sleep	ON	ON	ON	ON	ON	OFF
Light-sleep	ON	ON	OFF	OFF	OFF	OFF
Deep-sleep	ON	OFF	OFF	OFF	OFF	OFF

默认情况下，ESP8684 系统复位后将进入 Active 模式。此后，CPU 在停止工作一段时间后（比如等待外部活动唤醒时），可以进入 Modem-sleep、Light-sleep 和 Deep-sleep 等低功耗模式。从 Active 到 Deep-sleep，可用功能递减 [1]、功耗递减[2]、唤醒延迟时间递增。此外，这些模式可支持的唤醒源 [3] 不同。用户根据具体需求，从功能要求、功耗高低、唤醒延迟及可用唤醒源等方面考虑，选择合适的功耗模式。

说明：

1. 更多详情，请见表 9-3。
2. 具体功耗数据可见《ESP8684 技术规格书》中的功耗特性章节。
3. 具体可支持的唤醒源，请见第 9.4.3 节。

9.4.3 唤醒源

ESP8684 可支持多种唤醒源将 CPU 从不同睡眠模式中唤醒。唤醒源的选择由 RTC_CNTL_WAKEUP_ENA 决定，见表 9-4。

表 9-4. 唤醒源

WAKEUP_ENA	唤醒源	Light-sleep	Deep-sleep
0x4	GPIO[1]	Y	Y
0x8	RTC 定时器	Y	Y
0x20	Wi-Fi[2]	Y	-
0x40	UART0[3]	Y	-
0x80	UART1[3]	Y	-
0x400	Bluetooth	Y	-

[1] 在 Deep-sleep 模式下，仅有 RTC GPIO 可以作为唤醒源。

[2] 为了通过 Wi-Fi 唤醒芯片，芯片将在 Active、Modem-sleep 和 Light-sleep 之间进行切换，CPU 和 RF 模块均将在预设间隔中唤醒，保证 Wi-Fi 的正常连接和数据通信。

[3] 当接收到的 RX 脉冲数量超过阈值寄存器 UART_SLEEP_CONF_REG 中的设置时，即触发唤醒。详情请见章节 19 *UART 控制器 (UART)*。

9.4.4 拒绝睡眠

ESP8684 提供了硬件拒绝睡眠的机制，防止系统在某些外设仍在工作但是未被 CPU 检测到时进入睡眠，最终导致该外设不能正常工作。

表 9-5. 拒绝睡眠

REJECT_ENA	拒绝睡眠源
0x4	GPIO
0x8	RTC Timer
0x20	Wi-Fi
0x400	Bluetooth

用户可以根据上方表 9-5，配置以下寄存器使能或禁用拒绝睡眠功能。

- 配置 RTC_CNTL_SLEEP_REJECT_ENA 整体使能或关闭拒绝休眠功能：
  - 进一步配置 RTC_CNTL_LIGHT_SLP_REJECT_EN，具体使能拒绝进入 Light_sleep；
  - 进一步配置 RTC_CNTL_DEEP_SLP_REJECT_EN，具体使能拒绝进入 Deep_sleep；
- 读取 RTC_CNTL_SLP_REJECT_CAUSE_REG 了解拒绝睡眠的原因。

9.5 寄存器列表

本小节的所有地址均为相对于低功耗管理基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

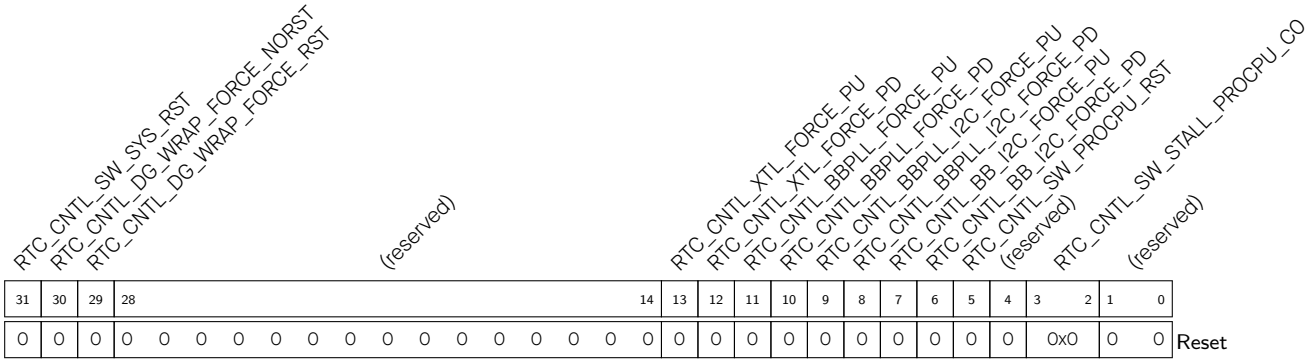
名称	描述	地址	访问权限
控制 / 配置寄存器			
RTC_CNTL_OPTIONS0_REG	配置晶振和 PLL 时钟的电源选项，并启动软件复位	0x0000	varies
RTC_CNTL_SLP_TIMER0_REG	RTC 定时器阈值寄存器 0	0x0004	R/W
RTC_CNTL_SLP_TIMER1_REG	RTC 定时器阈值寄存器 1	0x0008	R/W
RTC_CNTL_TIME_UPDATE_REG	RTC 定时器更新控制寄存器	0x000C	R/W
RTC_CNTL_TIME_LOW0_REG	存储 RTC 定时器 0 的低 32 位	0x0010	R/W
RTC_CNTL_TIME_HIGH0_REG	存储 RTC 定时器 0 的高 16 位	0x0014	R/W
RTC_CNTL_STATE0_REG	配置 sleep / reject / wakeup 状态	0x0018	R/W
RTC_CNTL_TIMER1_REG	配置 CPU stall 选项	0x001C	R/W
RTC_CNTL_TIMER2_REG	配置 RTC_SLOW_CLK 和触摸控制器	0x0020	R/W
RTC_CNTL_ANA_CONF_REG	配置 I2C 和 PLLA 的电源选项	0x002C	R/W
RTC_CNTL_WAKEUP_STATE_REG	唤醒位图使能寄存器	0x0034	R/W
RTC_CNTL_STORE0_REG	保留寄存器 0	0x0048	R/W
RTC_CNTL_STORE1_REG	保留寄存器 1	0x004C	R/W
RTC_CNTL_STORE2_REG	保留寄存器 2	0x0050	R/W
RTC_CNTL_STORE3_REG	保留寄存器 3	0x0054	R/W
RTC_CNTL_EXT_WAKEUP_CONF_REG	GPIO 唤醒配置寄存器	0x005C	R/W
RTC_CNTL_SLP_REJECT_CONF_REG	配置睡眠 / 拒绝睡眠选项	0x0060	R/W
RTC_CNTL_CLK_CONF_REG	RTC 定时器配置寄存器	0x0068	R/W
RTC_CNTL_REG	RTC 配置寄存器	0x0074	R/W
RTC_CNTL_PWC_REG	RTC 电源配置寄存器	0x0078	R/W
RTC_CNTL_DIG_PWC_REG	数字系统电源配置寄存器	0x007C	R/W
RTC_CNTL_DIG_ISO_REG	数字系统 ISO 配置寄存器	0x0080	R/W
RTC_CNTL_WDTCONFIG0_REG	RTC 看门狗配置寄存器	0x0084	R/W
RTC_CNTL_WDTCONFIG1_REG	配置 0 级 RTC 看门狗的保持时间	0x0088	R/W
RTC_CNTL_WDTCONFIG2_REG	配置 1 级 RTC 看门狗的保持时间	0x008C	R/W
RTC_CNTL_WDTCONFIG3_REG	配置 2 级 RTC 看门狗的保持时间	0x0090	R/W
RTC_CNTL_WDTCONFIG4_REG	配置 3 级 RTC 看门狗的保持时间	0x0094	R/W
RTC_CNTL_WDTFEED_REG	RTC 看门狗软件喂狗配置寄存器	0x0098	R/W
RTC_CNTL_WDTWPROTECT_REG	RTC 看门狗写保护配置寄存器	0x009C	R/W
RTC_CNTL_SWD_CONF_REG	超级看门狗配置寄存器	0x00A0	R/W
RTC_CNTL_SWD_WPROTECT_REG	超级看门狗写保护配置寄存器	0x00A4	R/W
RTC_CNTL_SW_CPU_STALL_REG	CPU stall 配置寄存器	0x00A8	R/W
RTC_CNTL_STORE4_REG	保留寄存器 4	0x00AC	R/W
RTC_CNTL_STORE5_REG	保留寄存器 5	0x00B0	R/W
RTC_CNTL_STORE6_REG	保留寄存器 6	0x00B4	R/W
RTC_CNTL_STORE7_REG	保留寄存器 7	0x00B8	R/W

名称	描述	地址	访问权限
RTC_CNTL_PAD_HOLD_REG	配置 RTC GPIO 的保持选项	0x00C4	R/W
RTC_CNTL_DIG_PAD_HOLD_REG	配置数字 GPIO 的保持选项	0x00C8	R/W
RTC_CNTL_BROWN_OUT_REG	欠压监测配置寄存器	0x00CC	R/W
RTC_CNTL_TIME_LOW1_REG	存储 RTC 定时器 1 的低 32 位	0x00D0	R/W
RTC_CNTL_TIME_HIGH1_REG	存储 RTC 定时器 1 的高 16 位	0x00D4	R/W
RTC_CNTL_USB_CONF_REG	IO_MUX 配置寄存器	0x00D8	R/W
RTC_CNTL_SLP_REJECT_CAUSE_REG	存储拒绝睡眠原因	0x00DC	R/W
RTC_CNTL_OPTION1_REG	RTC 选项寄存器	0x00E0	R/W
RTC_CNTL_SLP_WAKEUP_CAUSE_REG	存储唤醒原因	0x00E4	R/W
RTC_CNTL_CNTL_GPIO_WAKEUP_REG	GPIO 唤醒配置寄存器	0x00FC	R/W
RTC_CNTL_CNTL_SENSOR_CTRL_REG	SAR ADC 控制寄存器	0x0108	R/W
RTC_CNTL_FIB_SEL_REG	欠压监测配置寄存器	0x00F8	R/W
状态寄存器			
RTC_CNTL_RESET_STATE_REG	存储 CPU 复位原因	0x0030	R/W
RTC_CNTL_LOW_POWER_ST_REG	存储 RTC 状态	0x00BC	R/W
中断寄存器			
RTC_CNTL_INT_ENA_RTC_REG	RTC 中断使能寄存器	0x0038	R/W
RTC_CNTL_INT_RAW_RTC_REG	RTC 中断原始寄存器	0x003C	R/W
RTC_CNTL_INT_ST_RTC_REG	RTC 中断状态寄存器	0x0040	R/W
RTC_CNTL_INT_CLR_RTC_REG	RTC 中断清除寄存器	0x0044	R/W
RTC_CNTL_INT_ENA_RTC_W1TS_REG	RTC 中断使能寄存器 (W1TS)	0x00EC	R/W
RTC_CNTL_INT_ENA_RTC_W1TC_REG	RTC 中断清除寄存器 (W1TC)	0x00F0	R/W

9.6 寄存器

本小节的所有地址均为相对于低功耗管理基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 9.1. RTC_CNTL_OPTIONS0_REG (0x0000)



RTC_CNTL_SW_STALL_PROCPU_C0 写 0x2 通过软件使 CPU 进入 stall 状态。仅当 RTC_CNTL_SW_STALL_PROCPU_C1 配置为 0x21 时有效。(R/W)

RTC_CNTL_SW_PROCPU_RST 写 1 软件复位 CPU。(WO)

RTC_CNTL_BB_I2C_FORCE_PD 写 1 强制关闭 BB_I2C。(R/W)

RTC_CNTL_BB_I2C_FORCE_PU 写 1 强制打开 BB_I2C。(R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PD 写 1 强制关闭 BB_PLL_I2C。(R/W)

RTC_CNTL_BBPLL_I2C_FORCE_PU 写 1 强制打开 BB_PLL_I2C。(R/W)

RTC_CNTL_BBPLL_FORCE_PD 写 1 强制关闭 BB_PLL。(R/W)

RTC_CNTL_BBPLL_FORCE_PU 写 1 强制打开 BB_PLL。(R/W)

RTC_CNTL_XTL_FORCE_PD 写 1 强制关闭 XTAL_CLK。(R/W)

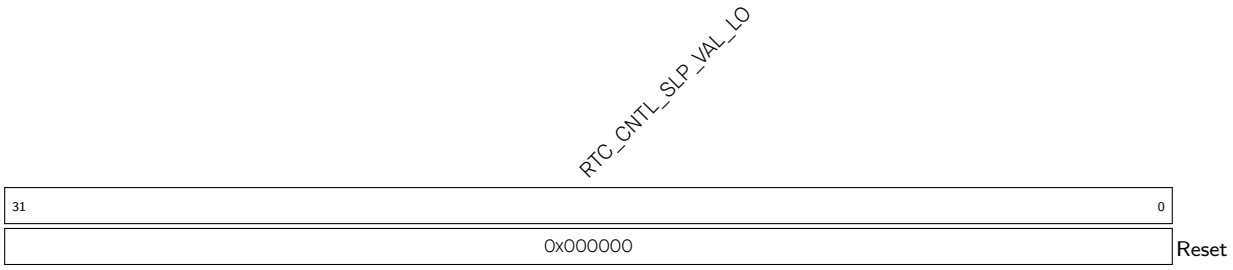
RTC_CNTL_XTL_FORCE_PU 写 1 强制打开 XTAL_CLK。(R/W)

RTC_CNTL_DG_WRAP_FORCE_RST 写 1 强制 Deep-sleep 中的数字系统复位。(R/W)

RTC_CNTL_DG_WRAP_FORCE_NORST 写 1 禁止强制 Deep-sleep 中的数字系统复位。(R/W)

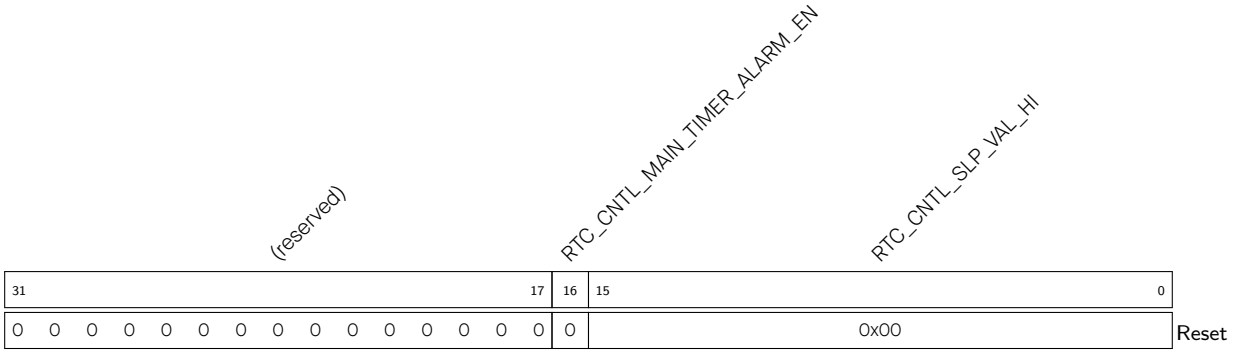
RTC_CNTL_SW_SYS_RST 写 1 通过软件复位数字类电源域。(WO)

Register 9.2. RTC_CNTL_SLP_TIMER0_REG (0x0004)



RTC_CNTL_SLP_VAL_LO 配置 RTC 定时器触发阈值的低 32 位。(R/W)

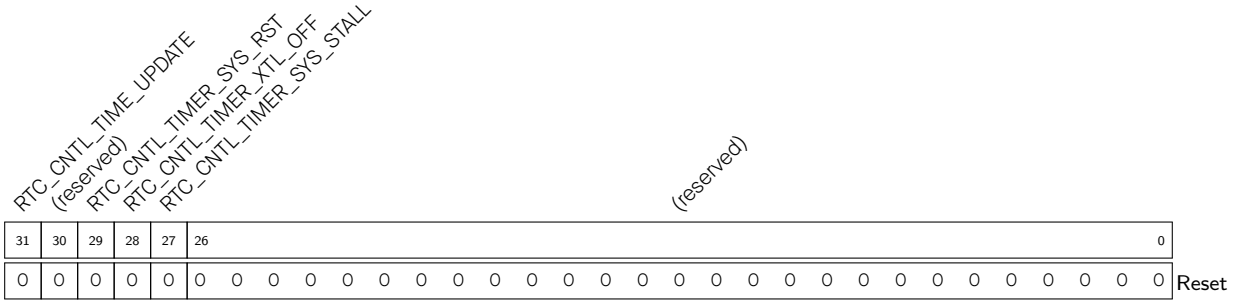
Register 9.3. RTC_CNTL_SLP_TIMER1_REG (0x0008)



RTC_CNTL_SLP_VAL_HI 配置 RTC 定时器触发阈值的高 16 位。(R/W)

RTC_CNTL_MAIN_TIMER_ALARM_EN 写 1 使能定时器警报。(R/W)

Register 9.4. RTC_CNTL_TIME_UPDATE_REG (0x000C)



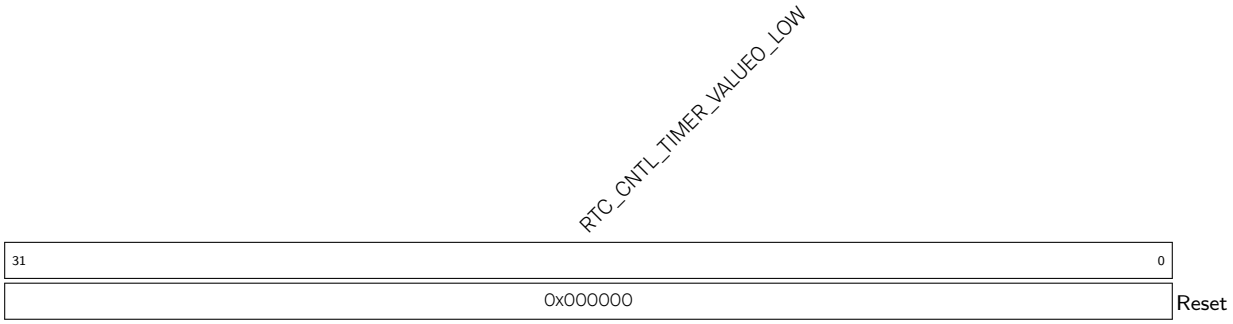
RTC_CNTL_TIMER_SYS_STALL 写 1 使能记录数字系统 stall 时间。(R/W)

RTC_CNTL_TIMER_XTL_OFF 写 1 使能记录 XTAL_CLK 掉电时间。(R/W)

RTC_CNTL_TIMER_SYS_RST 写 1 使能记录数字系统复位时间。(R/W)

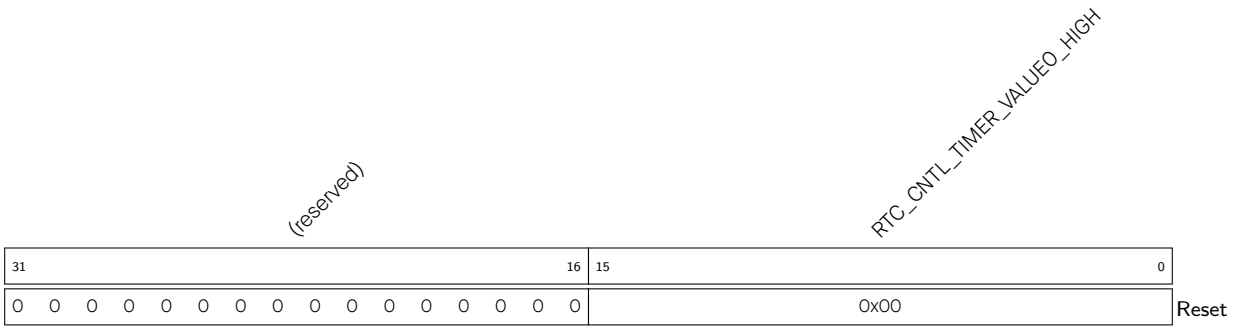
RTC_CNTL_TIME_UPDATE 写 1 使用 RTC 定时器更新寄存器。(R/W)

Register 9.5. RTC_CNTL_TIME_LOW0_REG (0x0010)



RTC_CNTL_TIMER_VALUE0_LOW 存储 RTC 定时器 0 的低 32 位。(R/W)

Register 9.6. RTC_CNTL_TIME_HIGH0_REG (0x0014)



RTC_CNTL_TIMER_VALUE0_HIGH 存储 RTC 定时器 0 的高 16 位。(R/W)

Register 9.7. RTC_CNTL_STATE0_REG (0x0018)

Bits	Field	Reset
31	RTC_CNTL_SLEEP_EN	0
30	RTC_CNTL_SLP_REJECT	0
29	RTC_CNTL_SLP_WAKEUP	0
28	RTC_CNTL_SDIO_ACTIVE_IND	0
27–23	(reserved)	0 0 0 0 0
22	RTC_CNTL_APB2RTC_BRIDGE_SEL	0
21–2	(reserved)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1	RTC_CNTL_SLP_REJECT_CAUSE_CLR	0
0	RTC_CNTL_SW_CPU_INT	0

RTC_CNTL_SW_CPU_INT 写 1 向 CPU 发送软件 RTC 中断。(R/W)

RTC_CNTL_SLP_REJECT_CAUSE_CLR 写 1 清除 RTC 拒绝进入睡眠状态的原因。(R/W)

RTC_CNTL_APB2RTC_BRIDGE_SEL 配置 APB 至 RTC 选项。

0x0：使用同步

0x1: 使用 bridge

(R/W)

RTC_CNTL_SDIO_ACTIVE_IND 表示 SDIO 处于活动状态。(R/W)

RTC_CNTL_SLP_WAKEUP 表示唤醒事件。(R/W)

RTC_CNTL_SLP_REJECT 表示拒绝入睡事件。(R/W)

RTC_CNTL_SLEEP_EN 写 1 使芯片进入睡眠状态。(R/W)

Register 9.8. RTC_CNTL_TIMER1_REG (0x001C)

Bits	Field	Reset
31–24	RTC_CNTL_PLL_BUF_WAIT	40
23–14	RTC_CNTL_XTL_BUF_WAIT	80
13–6	RTC_CNTL_FOSC_WAIT	0x10
5–1	RTC_CNTL_CPU_STALL_WAIT	1
0	RTC_CNTL_CPU_STALL_EN	1

RTC_CNTL_CPU_STALL_EN 写 1 使能 CPU stall。(R/W)

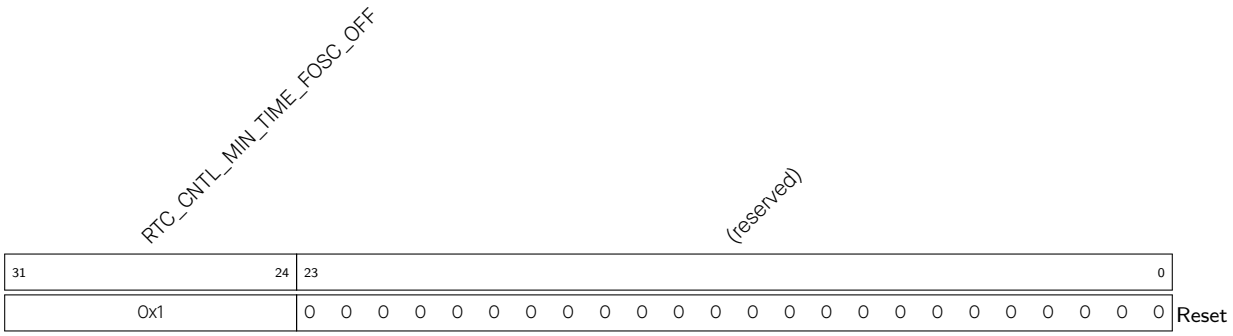
RTC_CNTL_CPU_STALL_WAIT 配置 CPU stall 等待周期（单位：RTC_FAST_CLK）。(R/W)

RTC_CNTL_FOSC_WAIT 配置 RC_FAST_CLK 等待周期（单位：RTC_SLOW_CLK）。(R/W)

RTC_CNTL_XTL_BUF_WAIT 配置 XTAL_CLK 等待周期（单位：RTC_SLOW_CLK）。(R/W)

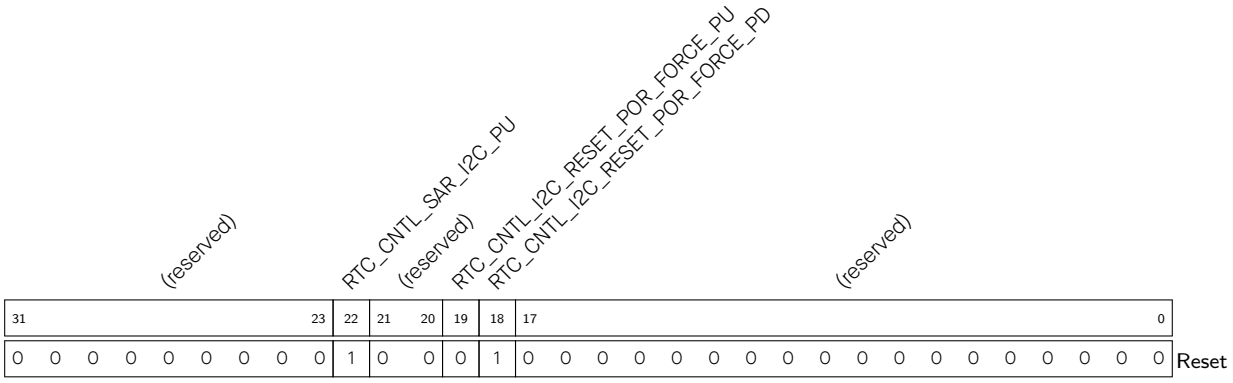
RTC_CNTL_PLL_BUF_WAIT 配置 PLL_CLK 等待周期（单位：RTC_SLOW_CLK）。(R/W)

Register 9.9. RTC_CNTL_TIMER2_REG (0x0020)



RTC_CNTL_MIN_TIME_FOSC_OFF 配置掉电时 RC_FAST_CLK 的最小睡眠周期（单位：RTC_SLOW_CLK）。(RW)

Register 9.10. RTC_CNTL_ANA_CONF_REG (0x002C)

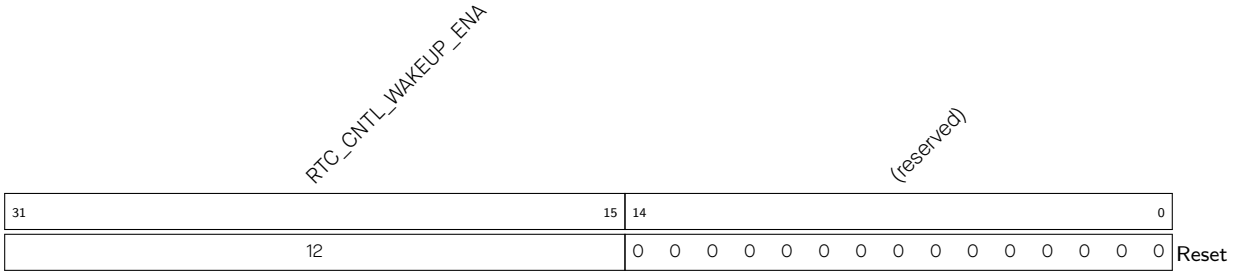


RTC_CNTL_I2C_RESET_POR_FORCE_PD 写 1 强制不许绕过 I2C 上电复位。(R/W)

RTC_CNTL_I2C_RESET_POR_FORCE_PU 写 1 强制绕过 I2C 上电复位。(R/W)

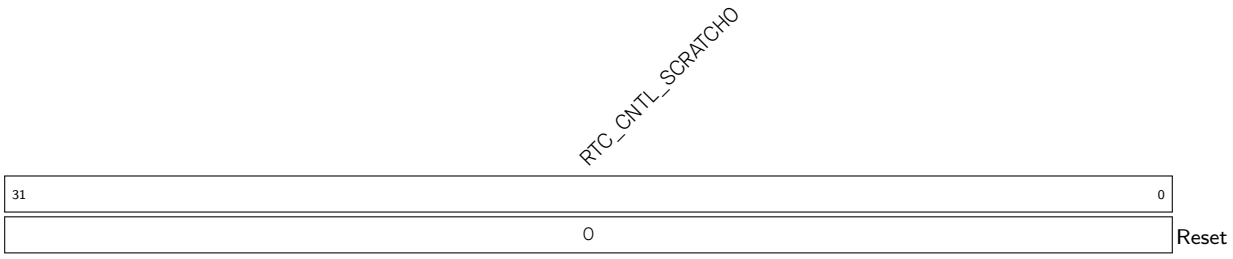
RTC_CNTL_SAR_I2C_PU 写 1 强制打开 SAR_I2C。(R/W)

Register 9.11. RTC_CNTL_WAKEUP_STATE_REG (0x0034)



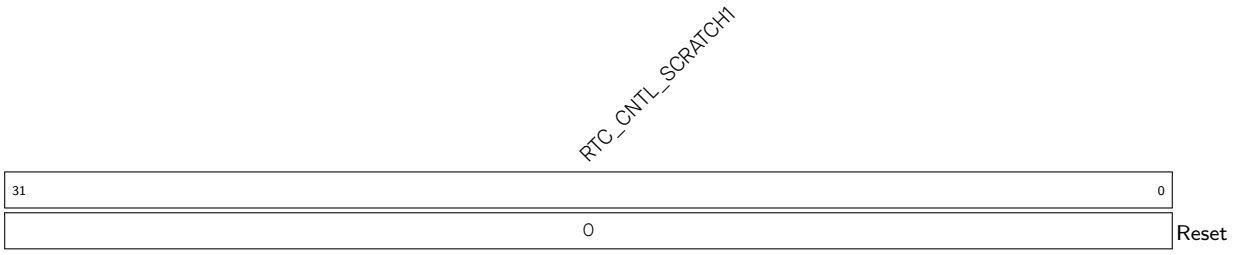
RTC_CNTL_WAKEUP_ENA 配置唤醒源。详见表 9-4。(R/W)

Register 9.12. RTC_CNTL_STORE0_REG (0x0048)



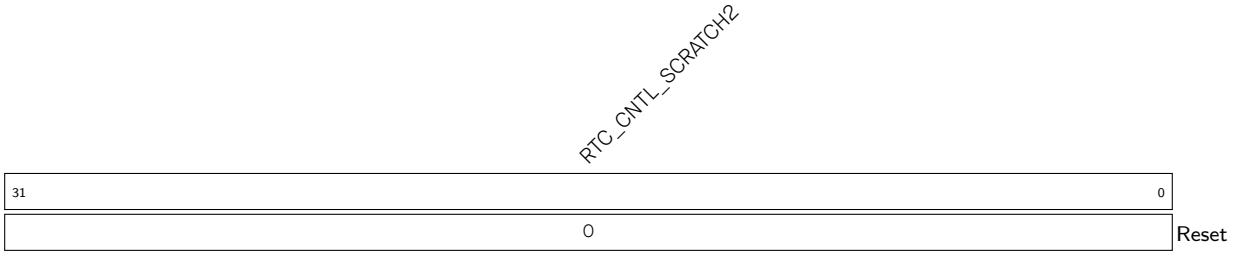
RTC_CNTL_SCRATCH0 保留寄存器 0。(R/W)

Register 9.13. RTC_CNTL_STORE1_REG (0x004C)



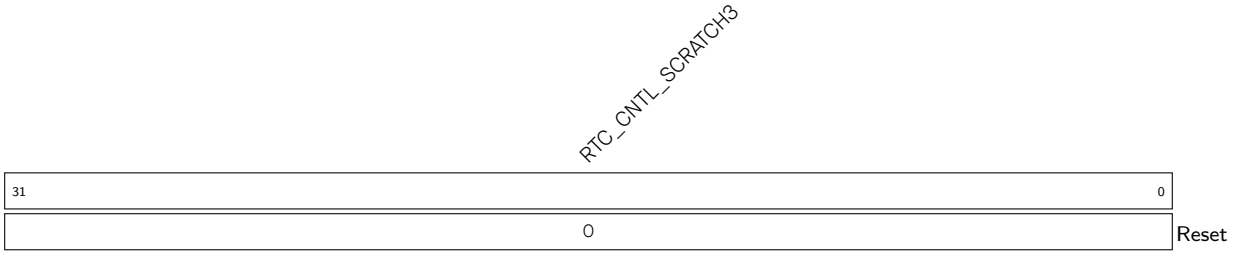
RTC_CNTL_SCRATCH1 保留寄存器 1。(R/W)

Register 9.14. RTC_CNTL_STORE2_REG (0x0050)



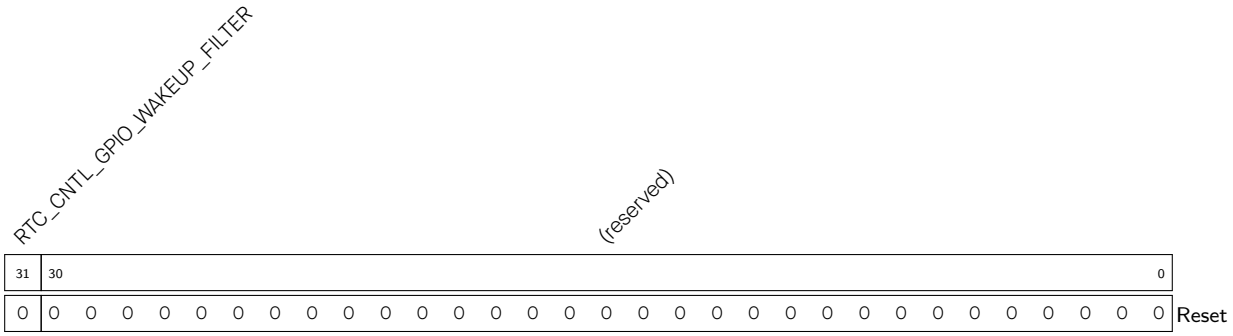
RTC_CNTL_SCRATCH2 保留寄存器 2。(R/W)

Register 9.15. RTC_CNTL_STORE3_REG (0x0054)



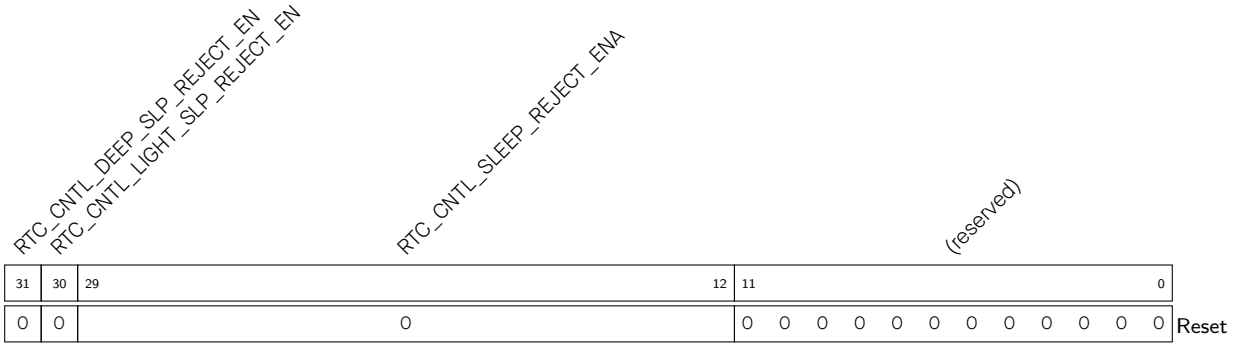
RTC_CNTL_SCRATCH3 保留寄存器 3。(R/W)

Register 9.16. RTC_CNTL_EXT_WAKEUP_CONF_REG (0x005C)



RTC_CNTL_GPIO_WAKEUP_FILTER 写 1 使能 GPIO 唤醒事件过滤器。(R/W)

Register 9.17. RTC_CNTL_SLP_REJECT_CONF_REG (0x0060)

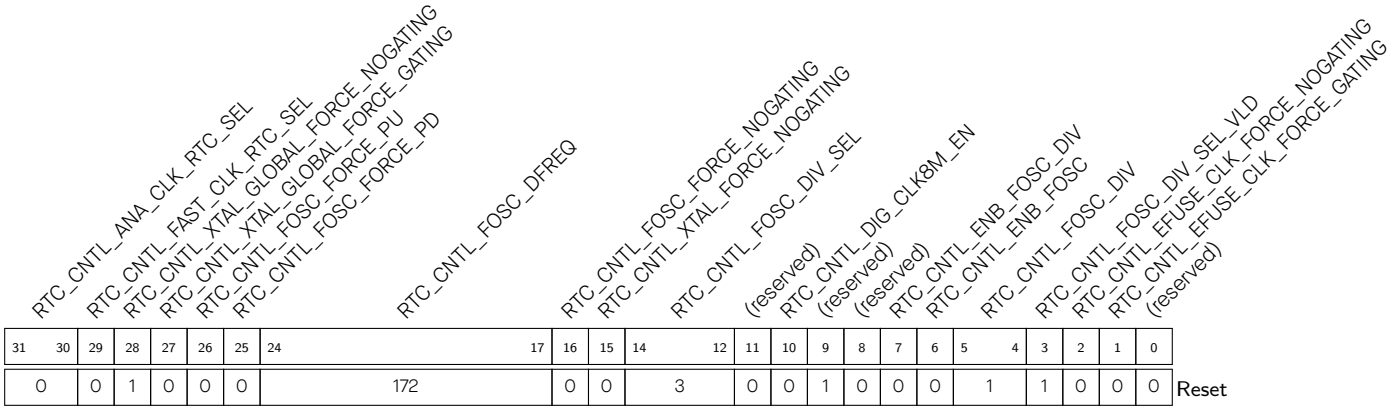


RTC_CNTL_SLEEP_REJECT_ENA 写 1 使能拒绝入睡。(R/W)

RTC_CNTL_LIGHT_SLP_REJECT_EN 写 1 使能拒绝进入 Light-sleep。(R/W)

RTC_CNTL_DEEP_SLP_REJECT_EN 写 1 使能拒绝进入 Deep-sleep。(R/W)

Register 9.18. RTC_CNTL_CLK_CONF_REG (0x0068)



RTC_CNTL_EFUSE_CLK_FORCE_GATING 写 1 强制打开 eFuse 时钟门控。(R/W)

RTC_CNTL_EFUSE_CLK_FORCE_NOGATING 写 1 强制关闭 eFuse 时钟门控。(R/W)

RTC_CNTL_FOSC_DIV_SEL_VLD 写 1 同步 RTC_CNTL_FOSC_DIV_SEL。注意在修改分频器前必须先使总线无效，然后重新使分频器时钟生效。(R/W)

RTC_CNTL_FOSC_DIV 配置 RC_FAST_DIV_CLK 分频数。

0x0：128 分频

0x1：256 分频

0x2：512 分频

0x3：1024 分频

(R/W)

RTC_CNTL_ENB_FOSC 写 1 禁用 RC_FAST_CLK 和 RC_FAST_DIV_CLK。(R/W)

RTC_CNTL_ENB_FOSC_DIV 配置 RC_FAST_CLK 分频数。

0x0：RC_FAST_CLK 的 256 分频

0x1：RC_FAST_CLK

(R/W)

RTC_CNTL_DIG_FOSC_EN 写 1 为数字系统选择 RC_FAST_CLK。(R/W)

RTC_CNTL_FOSC_DIV_SEL 表示 RC_FAST_CLK 分频数，即 RTC_CNTL_FOSC_DIV_SEL + 1。(RW)

接下页...

Register 9.18. RTC_CNTL_CLK_CONF_REG (0x0068)

接上页...

RTC_CNTL_XTAL_FORCE_NOGATING 写 1 在睡眠期间强制打开 XTAL_CLK。(R/W)

RTC_CNTL_FOSC_FORCE_NOGATING 写 1 在睡眠期间强制绕过 XTAL_CLK。(R/W)

RTC_CNTL_FOSC_DFREQ 配置 RC_FAST_CLK 频率。(R/W)

RTC_CNTL_FOSC_FORCE_PD 写 1 强制关闭 RC_FAST_CLK。(R/W)

RTC_CNTL_FOSC_FORCE_PU 写 1 强制打开 RC_FAST_CLK。(R/W)

RTC_CNTL_XTAL_GLOBAL_FORCE_GATING 写 1 强制打开 XTAL_CLK 时钟门控。(R/W)

RTC_CNTL_XTAL_GLOBAL_FORCE_NOGATING 写 1 强制绕过 XTAL_CLK 时钟门控。(R/W)

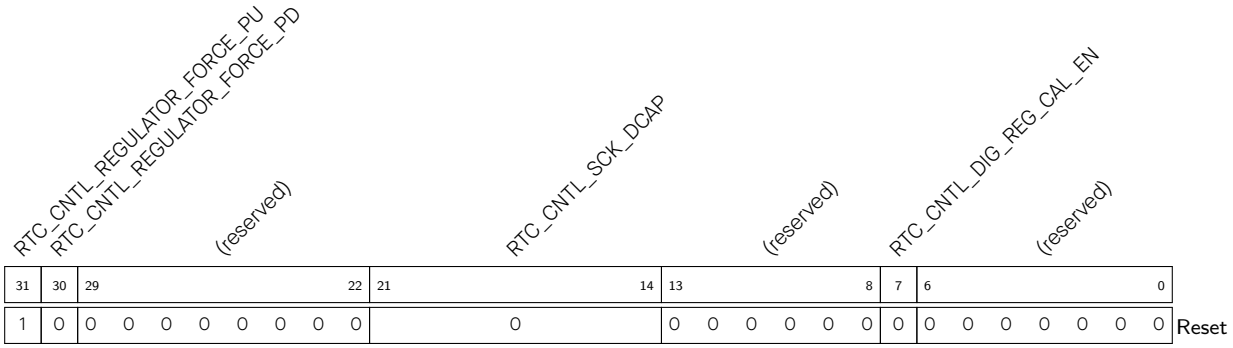
RTC_CNTL_FAST_CLK_RTC_SEL 配置 RTC_FAST_CLK。

0x0：XTAL_DIV_CLK
0x1：FOSC_DIV
(R/W)

RTC_CNTL_ANA_CLK_RTC_SEL 配置 RC_SLOW_CLK。

0x0：RC_SLOW_CLK
0x1：OSC_SLOW_CLK
0x2：RC_FAST_DIV_CLK
0x3：保留
(R/W)

Register 9.19. RTC_CNTL_REG (0x0074)



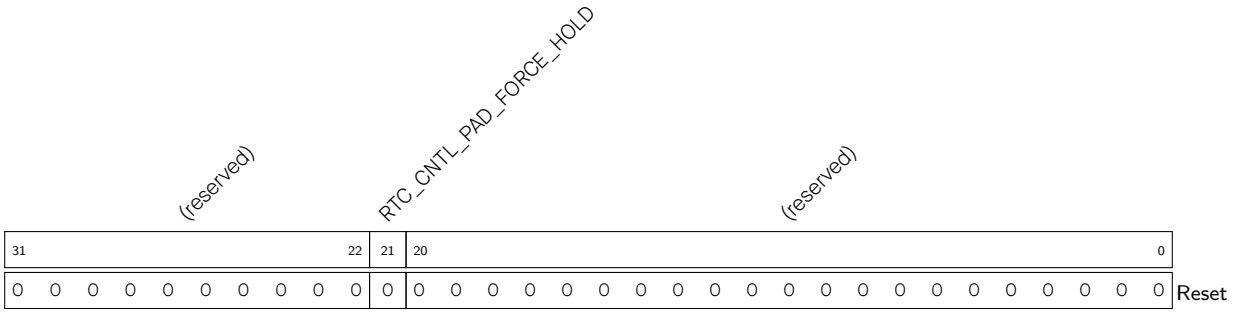
RTC_CNTL_DIG_REG_CAL_EN 写 1 使能数字调压器的软件校准。(R/W)

RTC_CNTL_SCK_DCAP 配置 RC_SLOW_CLK 频率。(R/W)

RTC_CNTL_REGULATOR_FORCE_PD 写 1 强制关闭低功耗调压器（即将电压降低至 0.8 V 或以下）。(R/W)

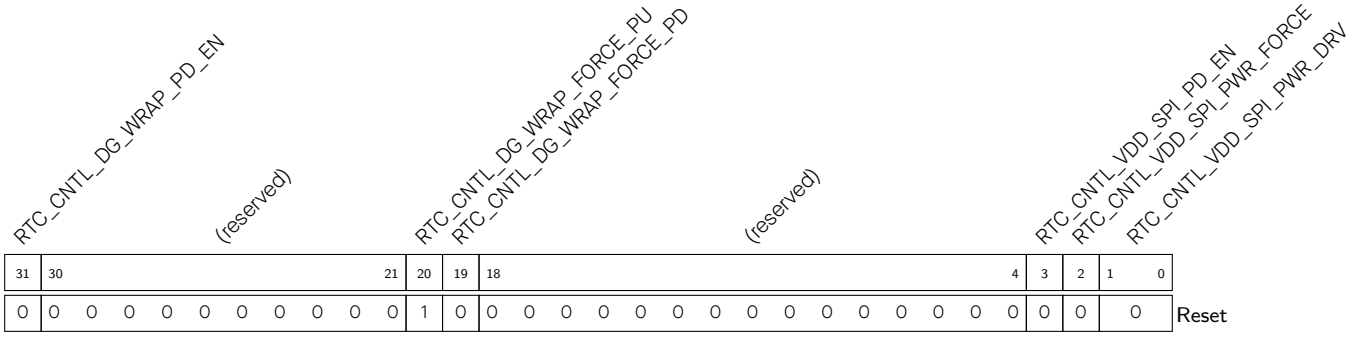
RTC_CNTL_REGULATOR_FORCE_PU 写 1 强制打开低功耗调压器（即将电压升至 0.8 V 或以上）。(R/W)

Register 9.20. RTC_CNTL_PWC_REG (0x0078)



RTC_CNTL_PAD_FORCE_HOLD 写 1 强制 RTC pad 进入 hold 状态。(R/W)

Register 9.21. RTC_CNTL_DIG_PWC_REG (0x007C)



RTC_CNTL_VDD_SPI_PWR_DRV 配置 vdd_spi 的驱动能力。(R/W)

RTC_CNTL_VDD_SPI_PWR_FORCE 写 1 允许软件配置 vdd_spi 的驱动能力。(R/W)

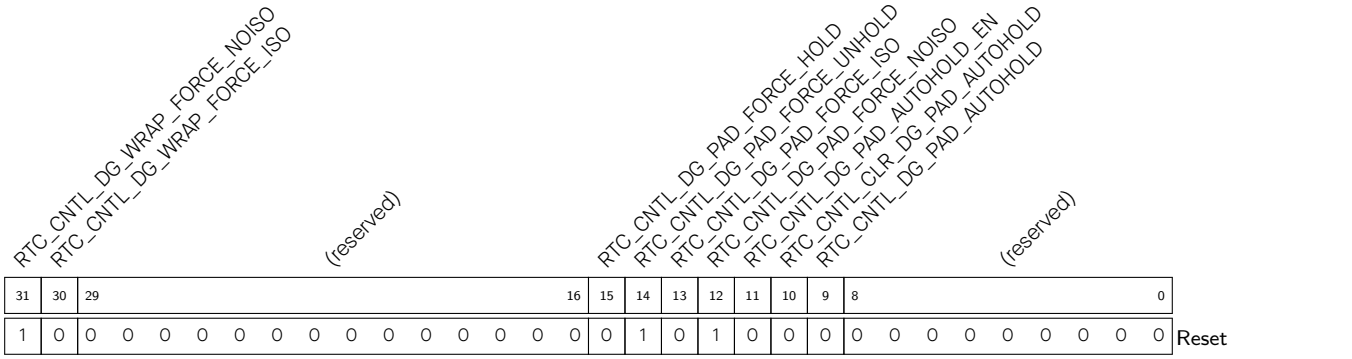
RTC_CNTL_VDD_SPI_PD_EN 写 1 在睡眠状态关闭 VDD_SPI。(R/W)

RTC_CNTL_DG_WRAP_FORCE_PD 写 1 强制关闭数字系统。(R/W)

RTC_CNTL_DG_WRAP_FORCE_PU 写 1 强制打开数字系统。(R/W)

RTC_CNTL_DG_WRAP_PD_EN 写 1 使能在睡眠中强制打开数字系统。(R/W)

Register 9.22. RTC_CNTL_DIG_ISO_REG (0x0080)



RTC_CNTL_DG_PAD_AUTOHOLD 存储数字 GPIO 的 auto-hold 状态。(R/W)

RTC_CNTL_CLR_DG_PAD_AUTOHOLD 写 1 取消数字 GPIO 的 auto-hold 状态。(R/W)

RTC_CNTL_DG_PAD_AUTOHOLD_EN 写 1 允许数字 GPIO 进入 auto-hold 状态。(R/W)

RTC_CNTL_DG_PAD_FORCE_NOISO 写 1 强制不隔离数字 GPIO。(R/W)

RTC_CNTL_DG_PAD_FORCE_ISO 写 1 强制隔离数字 GPIO。(R/W)

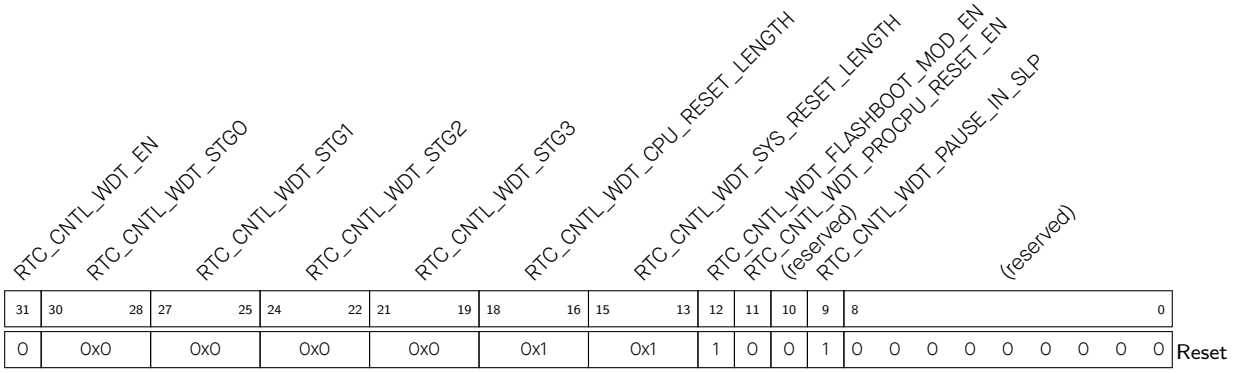
RTC_CNTL_DG_PAD_FORCE_UNHOLD 写 1 强制数字 GPIO 进入 unhold 状态。(R/W)

RTC_CNTL_DG_PAD_FORCE_HOLD 写 1 强制数字 GPIO 进入 hold 状态。(R/W)

RTC_CNTL_DG_WRAP_FORCE_ISO 写 1 强制隔离数字系统。(R/W)

RTC_CNTL_DG_WRAP_FORCE_NOISO 写 1 强制不隔离数字系统。(R/W)

Register 9.23. RTC_CNTL_WDTCONFIG0_REG (0x0084)



RTC_CNTL_WDT_PAUSE_IN_SLP 写 1 在睡眠中暂停看门狗。(R/W)

RTC_CNTL_WDT_PROCPU_RESET_EN 写 1 允许通过 RTC 看门狗复位 CPU。(R/W)

RTC_CNTL_WDT_FLASHBOOT_MOD_EN 写 1 在芯片从 flash 重启时使能看门狗。(R/W)

RTC_CNTL_WDT_SYS_RESET_LENGTH 配置数字系统复位计数器的长度。(R/W)

RTC_CNTL_WDT_CPU_RESET_LENGTH 配置 CPU 复位计数器的长度。(R/W)

RTC_CNTL_WDT_STG3 配置 3 级 RTC 看门狗的超时动作。

0x1：触发中断
0x2：复位 CPU 内核
0x3：复位数字系统（除 RTC）
0x4：复位数字系统（包括 RTC）
(R/W)

RTC_CNTL_WDT_STG2 配置 2 级 RTC 看门狗的超时动作。

0x1：触发中断
0x2：复位 CPU 内核
0x3：复位数字系统（除 RTC）
0x4：复位数字系统（包括 RTC）
(R/W)

接下页...

Register 9.23. RTC_CNTL_WDTCONFIG0_REG (0x0084)

接上页...

RTC_CNTL_WDT_STG1 配置 1 级 RTC 看门狗的超时动作。

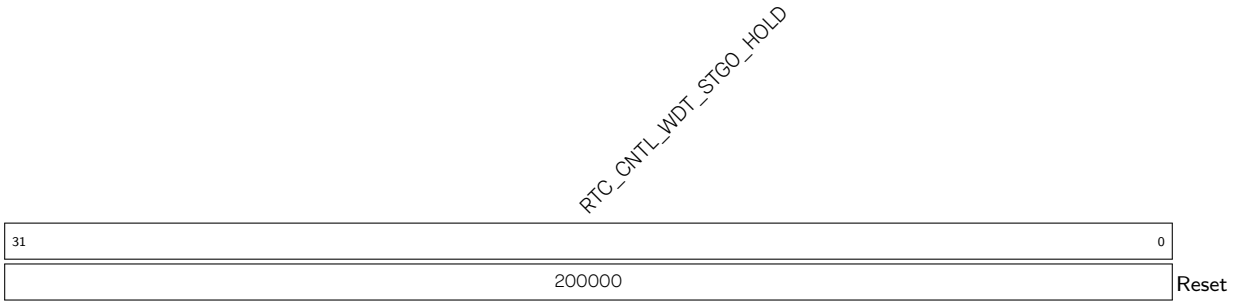
0x1：触发中断
0x2：复位 CPU 内核
0x3：复位数字系统（除 RTC）
0x4：复位数字系统（包括 RTC）
(R/W)

RTC_CNTL_WDT_STG0 配置 0 级 RTC 看门狗的超时动作。

0x1：触发中断
0x2：复位 CPU 内核
0x3：复位数字系统（除 RTC）
0x4：复位数字系统（包括 RTC）
(R/W)

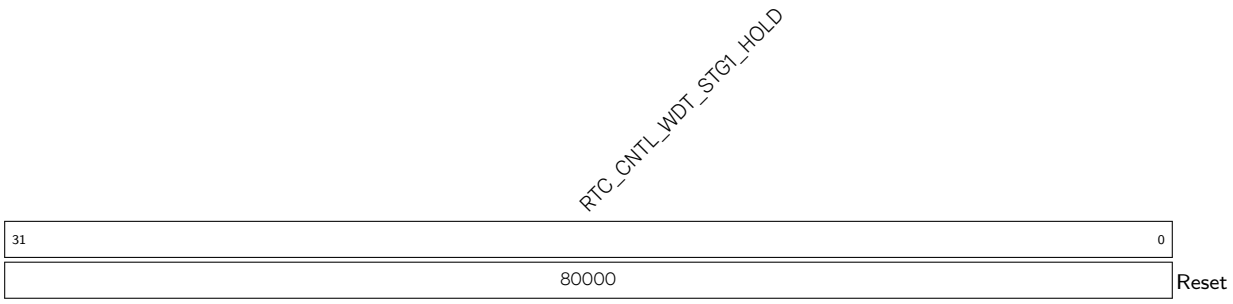
RTC_CNTL_WDT_EN 写 1 使能 RTC 看门狗。(R/W)

Register 9.24. RTC_CNTL_WDTCONFIG1_REG (0x0088)



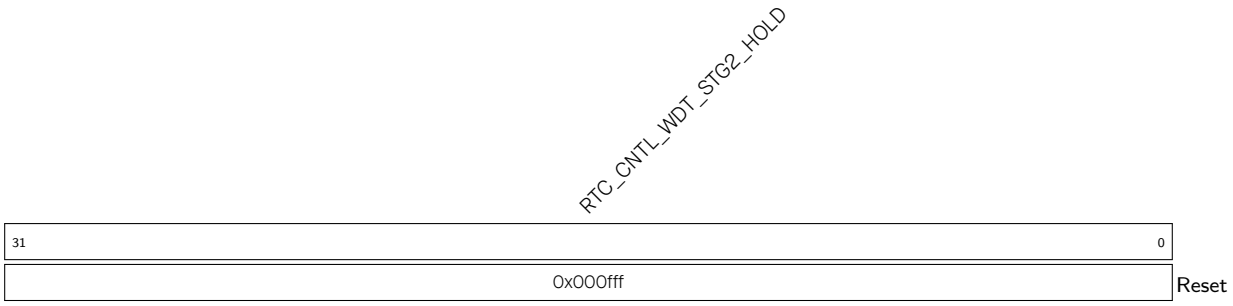
RTC_CNTL_WDT_STG0_HOLD 配置 0 级 RTC 看门狗的 hold 时间。(RW)

Register 9.25. RTC_CNTL_WDTCONFIG2_REG (0x008C)



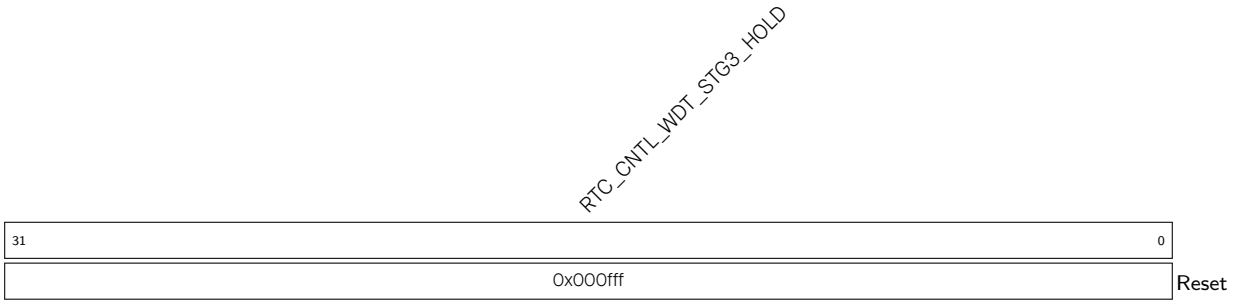
RTC_CNTL_WDT_STG1_HOLD 配置 1 级 RTC 看门狗的 hold 时间。(R/W)

Register 9.26. RTC_CNTL_WDTCONFIG3_REG (0x0090)



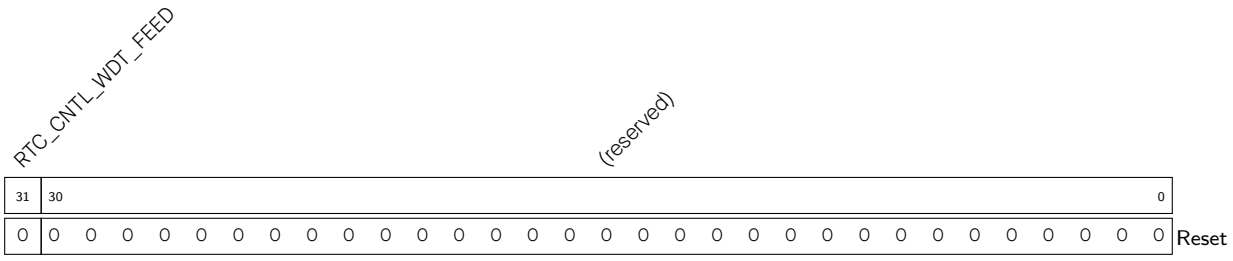
RTC_CNTL_WDT_STG2_HOLD 配置 2 级 RTC 看门狗的 hold 时间。(R/W)

Register 9.27. RTC_CNTL_WDTCONFIG4_REG (0x0094)



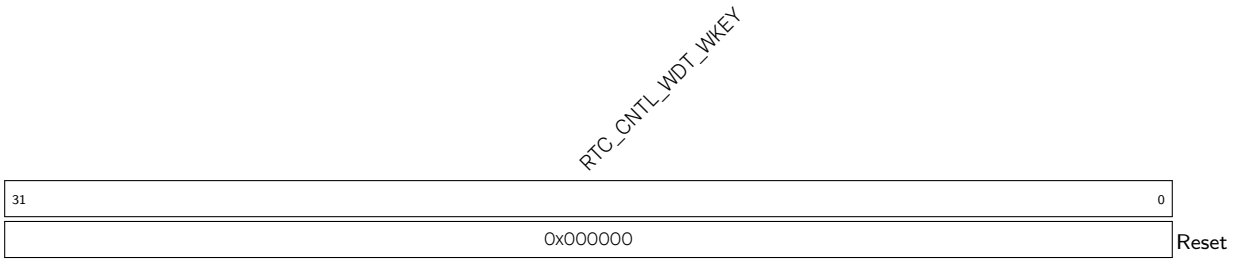
RTC_CNTL_WDT_STG3_HOLD 配置 3 级 RTC 看门狗的 hold 时间。(R/W)

Register 9.28. RTC_CNTL_WDTFEED_REG (0x0098)



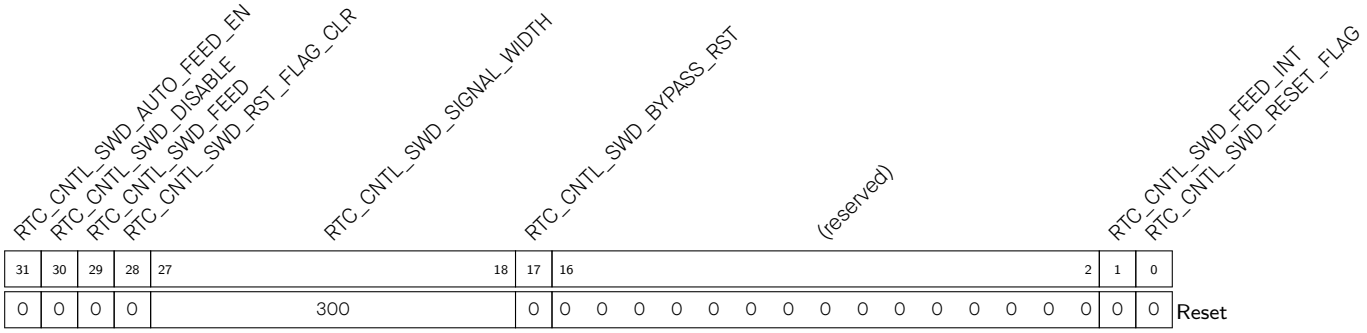
RTC_CNTL_WDT_FEED 写 1 开始喂 RTC 看门狗。(R/W)

Register 9.29. RTC_CNTL_WDTWPROTECT_REG (0x009C)



RTC_CNTL_WDT_WKEY 当该寄存器中的值不为 0x50d83aa1 时，使能 RTC 看门狗 (RWDT) 的写保护。(R/W)

Register 9.30. RTC_CNTL_SWD_CONF_REG (0x00A0)



RTC_CNTL_SWD_RESET_FLAG 表示超级看门狗的重置标志。(R/W)

RTC_CNTL_SWD_FEED_INT 表示将软件喂 RTC 看门狗。(R/W)

RTC_CNTL_SWD_BYPASS_RST 写 1 绕过超级看门狗复位。(R/W)

RTC_CNTL_SWD_SIGNAL_WIDTH 配置发送到超级看门狗的信号宽度。(R/W)

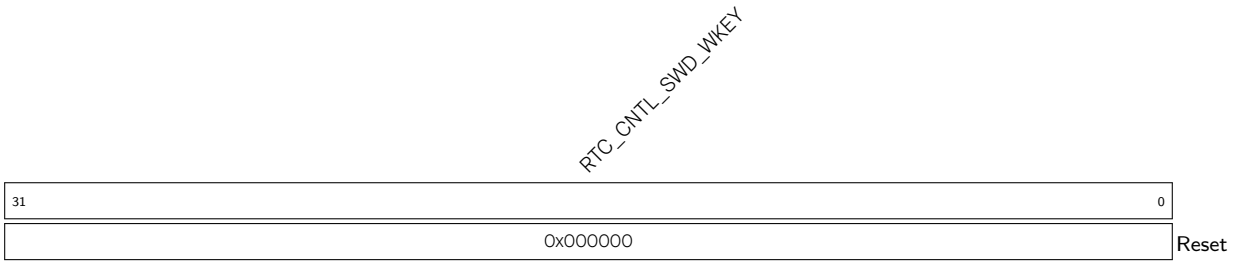
RTC_CNTL_SWD_RST_FLAG_CLR 写 1 复位超级看门狗的复位标志。(R/W)

RTC_CNTL_SWD_FEED 写 1 开始喂超级看门狗。(R/W)

RTC_CNTL_SWD_DISABLE 写 1 禁用超级看门狗。(R/W)

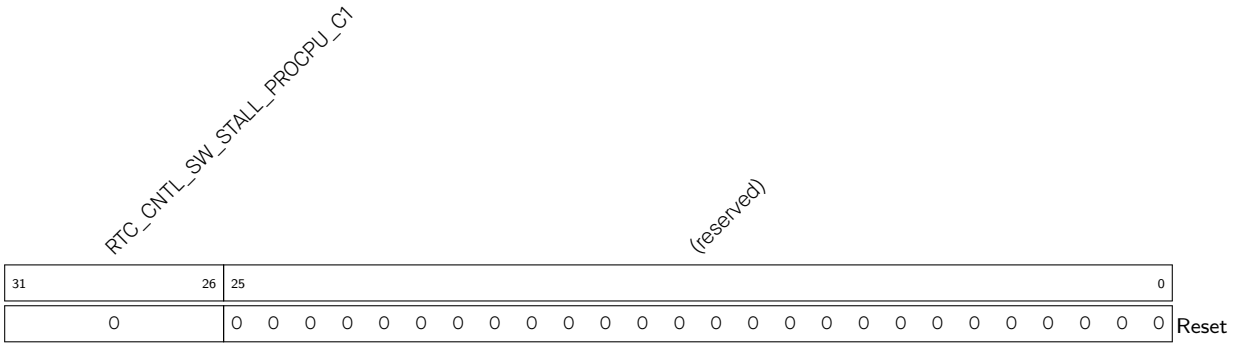
RTC_CNTL_SWD_AUTO_FEED_EN 写 1 使能中断时自动喂狗。(R/W)

Register 9.31. RTC_CNTL_SWD_WPROTECT_REG (0x00A4)



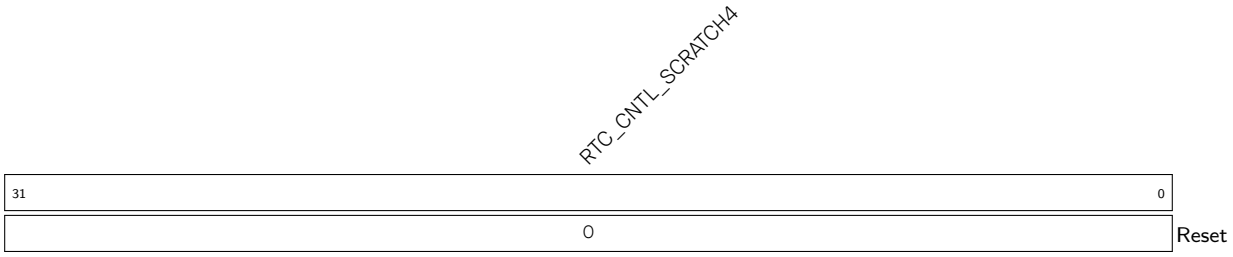
RTC_CNTL_SWD_WKEY 配置超级看门狗的写保护密钥。(R/W)

Register 9.32. RTC_CNTL_SW_CPU_STALL_REG (0x00A8)



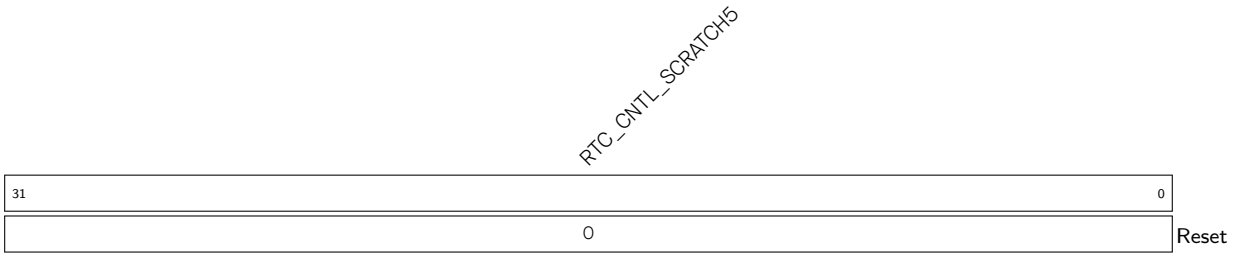
RTC_CNTL_SW_STALL_PROCPU_C1 写 0x21 通过软件使 CPU 进入 stall 状态。仅当 RTC_CNTL_SW_STALL_PROCPU_C0 配置为 0x2 时有效。(R/W)

Register 9.33. RTC_CNTL_STORE4_REG (0x00AC)



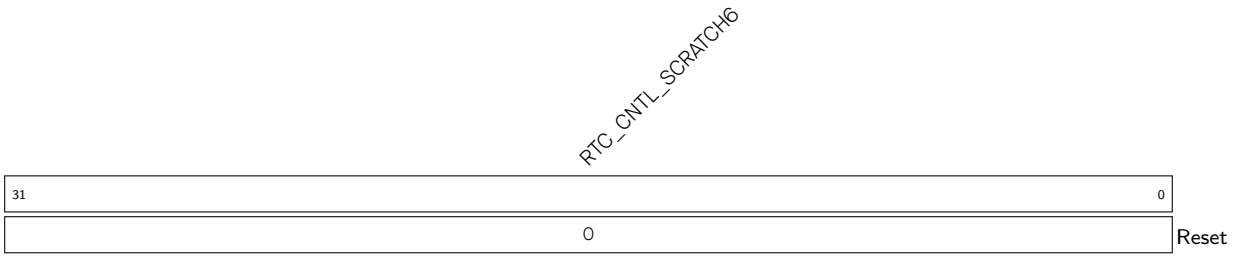
RTC_CNTL_SCRATCH4 保留寄存器 4。(R/W)

Register 9.34. RTC_CNTL_STORE5_REG (0x00B0)



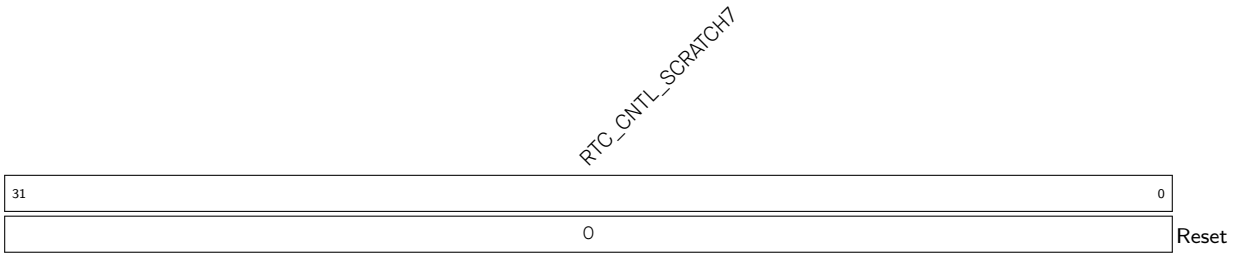
RTC_CNTL_SCRATCH5 保留寄存器 5。(R/W)

Register 9.35. RTC_CNTL_STORE6_REG (0x00B4)



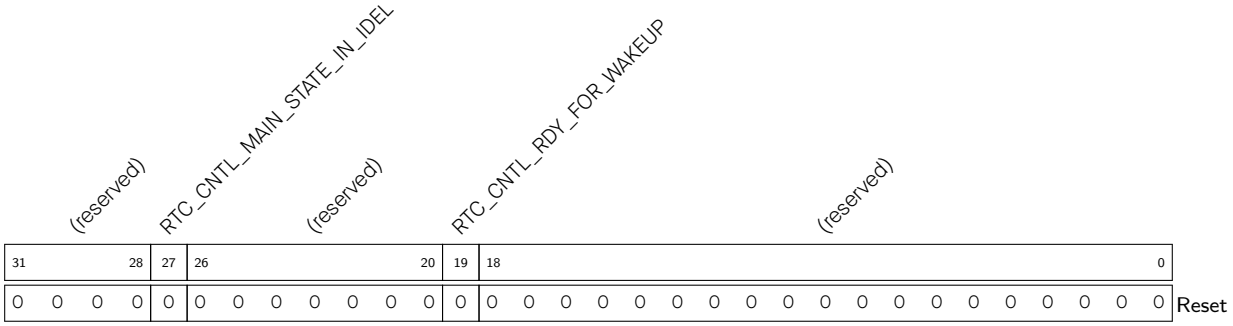
RTC_CNTL_SCRATCH6 保留寄存器 6。(R/W)

Register 9.36. RTC_CNTL_STORE7_REG (0x00B8)



RTC_CNTL_SCRATCH7 保留寄存器 7。(R/W)

Register 9.37. RTC_CNTL_LOW_POWER_ST_REG (0x00BC)

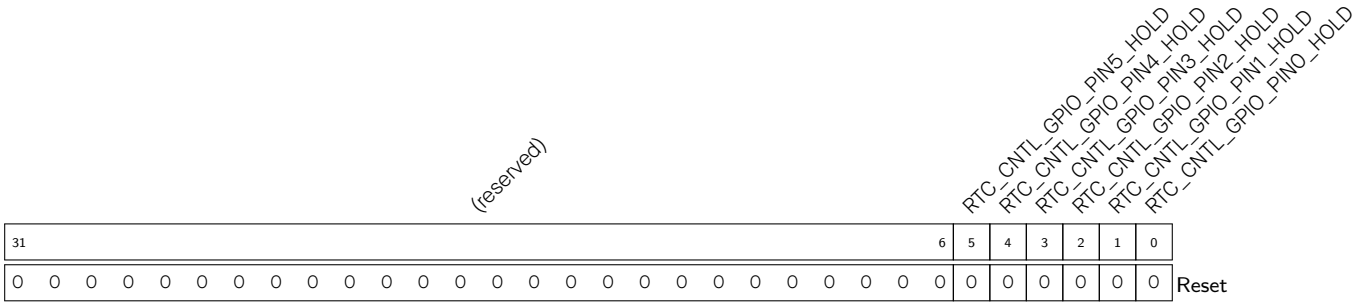


RTC_CNTL_RDY_FOR_WAKEUP 表示 RTC 已准备好由任何唤醒源触发。(RO)

RTC_CNTL_MAIN_STATE_IN_IDLE 代表 RTC 状态。

- 0：芯片可能处于以下任一情况
  - 已经进入睡眠模式
  - 正在进入睡眠模式。此时，需等待 RTC_CNTL_RDY_FOR_WAKEUP 变为 1，然后可以唤醒芯片。
  - 正在退出睡眠模式。此时，RTC_CNTL_MAIN_STATE_IN_IDLE 终将变为 1。
- 1：芯片不处于睡眠模式（比如正常运行中）。

Register 9.38. RTC_CNTL_PAD_HOLD_REG (0x00C4)



RTC_CNTL_GPIO_PIN0_HOLD 置 1 使 GPIO 0 进入 hold 状态。(R/W)

RTC_CNTL_GPIO_PIN1_HOLD 置 1 使 GPIO 1 进入 hold 状态。(R/W)

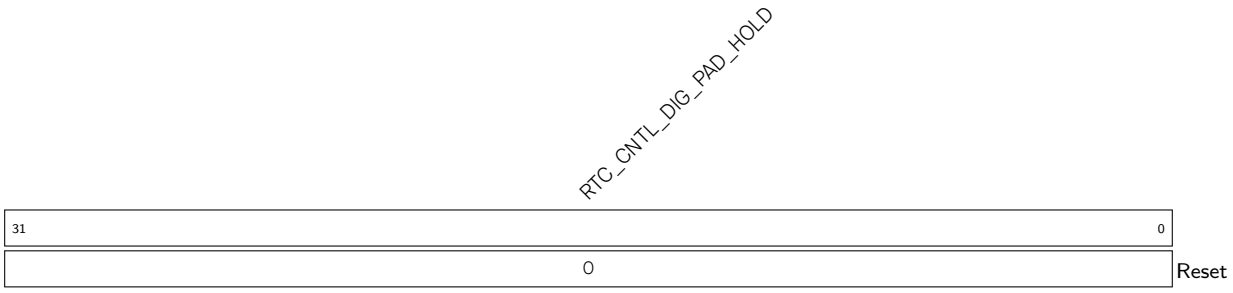
RTC_CNTL_GPIO_PIN2_HOLD 置 1 使 GPIO 2 进入 hold 状态。(R/W)

RTC_CNTL_GPIO_PIN3_HOLD 置 1 使 GPIO 3 进入 hold 状态。(R/W)

RTC_CNTL_GPIO_PIN4_HOLD 置 1 使 GPIO 4 进入 hold 状态。(R/W)

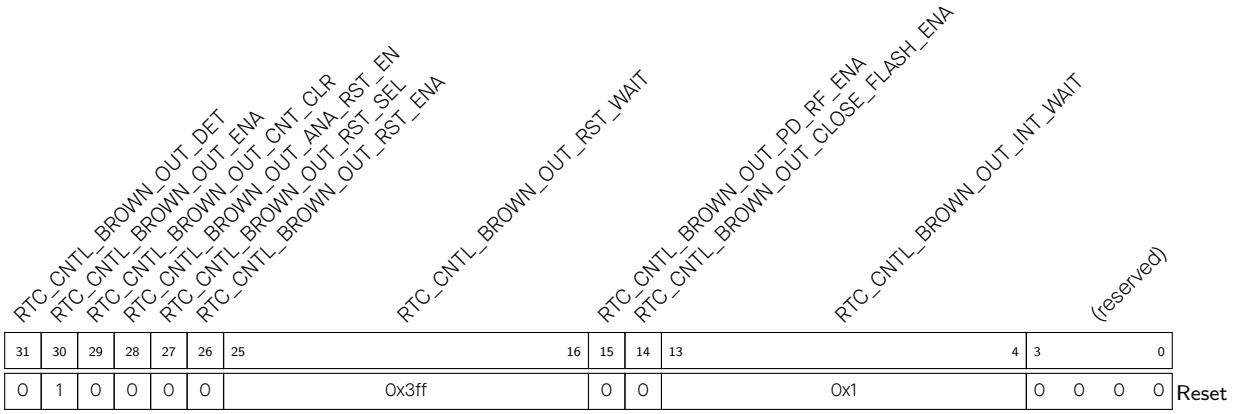
RTC_CNTL_GPIO_PIN5_HOLD 置 1 使 GPIO 5 进入 hold 状态。(R/W)

Register 9.39. RTC_CNTL_DIG_PAD_HOLD_REG (0x00C8)



RTC_CNTL_DIG_PAD_HOLD 置 1 使 GPIO 6 - GPIO 20 进入 hold 状态。其中，GPIO 的位置可见芯片位图。(R/W)

Register 9.40. RTC_CNTL_BROWN_OUT_REG (0x00CC)



RTC_CNTL_BROWN_OUT_INT_WAIT 配置发送欠压掉电中断前的等待周期。(R/W)

RTC_CNTL_BROWN_OUT_CLOSE_FLASH_ENA 写 1 使能在发生欠压掉电时强制关闭 flash。(R/W)

RTC_CNTL_BROWN_OUT_PD_RF_ENA 写 1 使能在发生欠压掉电时强制关闭 RF 电路。(R/W)

RTC_CNTL_BROWN_OUT_RST_WAIT 配置发生欠压掉电复位前的等待周期。(R/W)

RTC_CNTL_BROWN_OUT_RST_ENA 写 1 复位欠压监测器。(R/W)

RTC_CNTL_BROWN_OUT_RST_SEL 配置欠压检测 mode0 下的复位类型。

0x0：系统复位

0x1：芯片复位

(R/W)

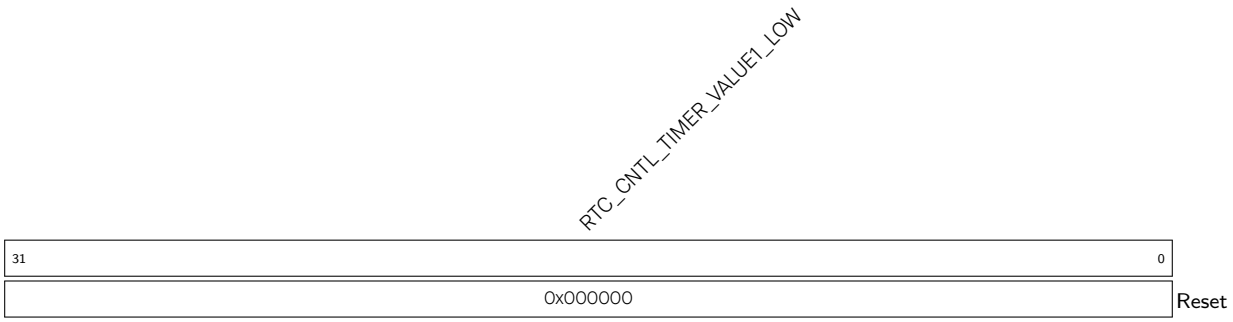
RTC_CNTL_BROWN_OUT_ANA_RST_EN 写 1 使能欠压监测 mode1。(R/W)

RTC_CNTL_BROWN_OUT_CNT_CLR 写 1 清除欠压监测计数器。(R/W)

RTC_CNTL_BROWN_OUT_ENA 写 1 使能欠压监测 mode0。(R/W)

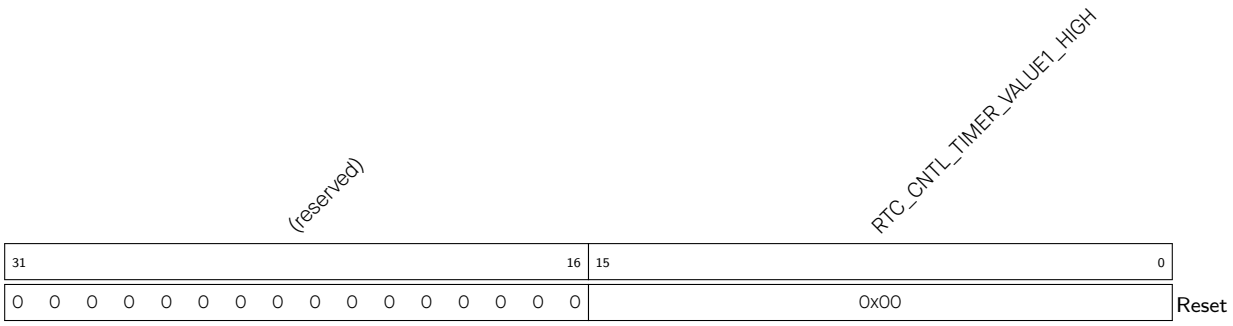
RTC_CNTL_BROWN_OUT_DET 表示欠压信号的状态。(R/W)

Register 9.41. RTC_CNTL_TIME_LOW1_REG (0x00D0)



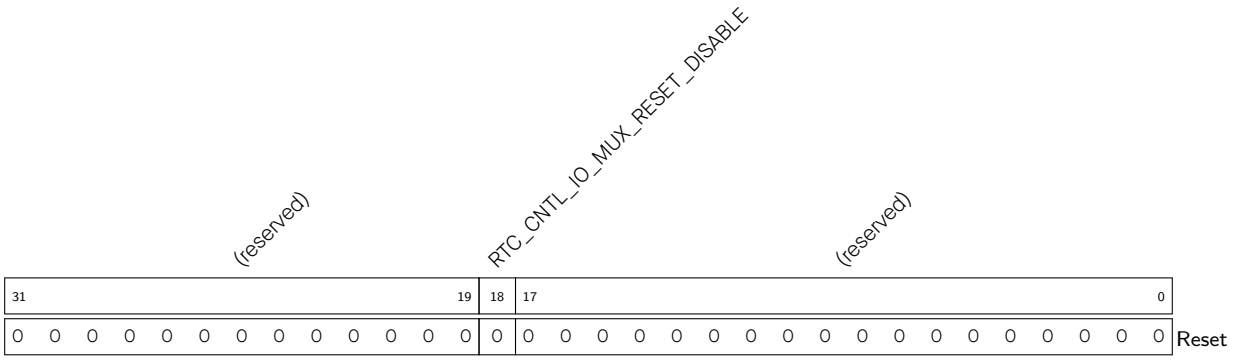
RTC_CNTL_TIMER_VALUE1_LOW 表示 RTC 定时器 1 的低 32 位。(R/W)

Register 9.42. RTC_CNTL_TIME_HIGH1_REG (0x00D4)



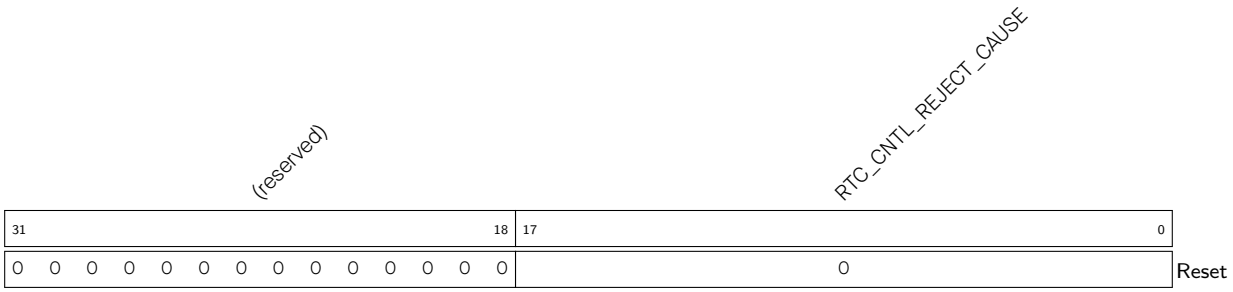
RTC_CNTL_TIMER_VALUE1_HIGH 存储 RTC 定时器 1 的高 16 位 (R/W)

Register 9.43. RTC_CNTL_USB_CONF_REG (0x00D8)



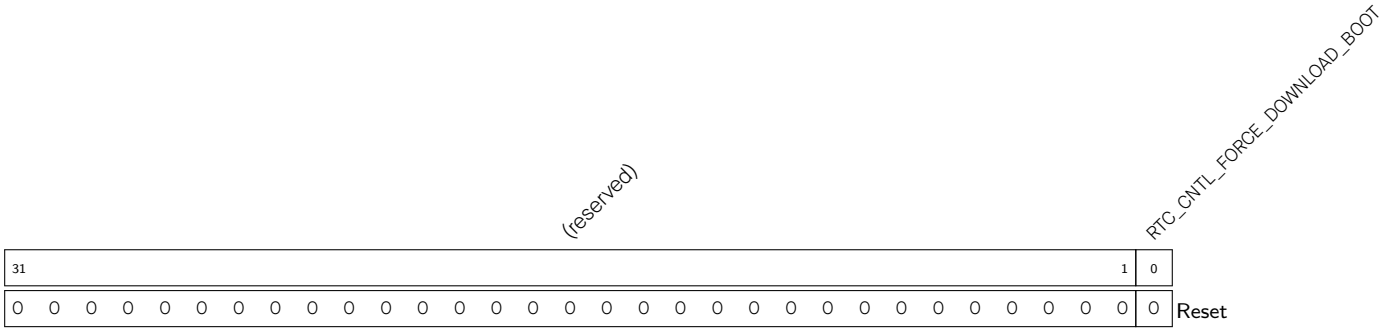
RTC_CNTL_IO_MUX_RESET_DISABLE 写 1 禁用 io_mux 复位。(R/W)

Register 9.44. RTC_CNTL_SLP_REJECT_CAUSE_REG (0x00DC)



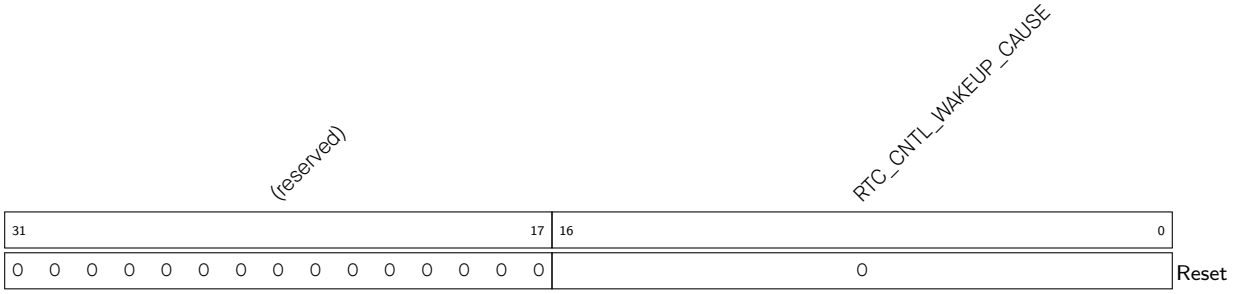
RTC_CNTL_REJECT_CAUSE 存储拒绝睡眠原因。(R/W)

Register 9.45. RTC_CNTL_OPTION1_REG (0x00E0)



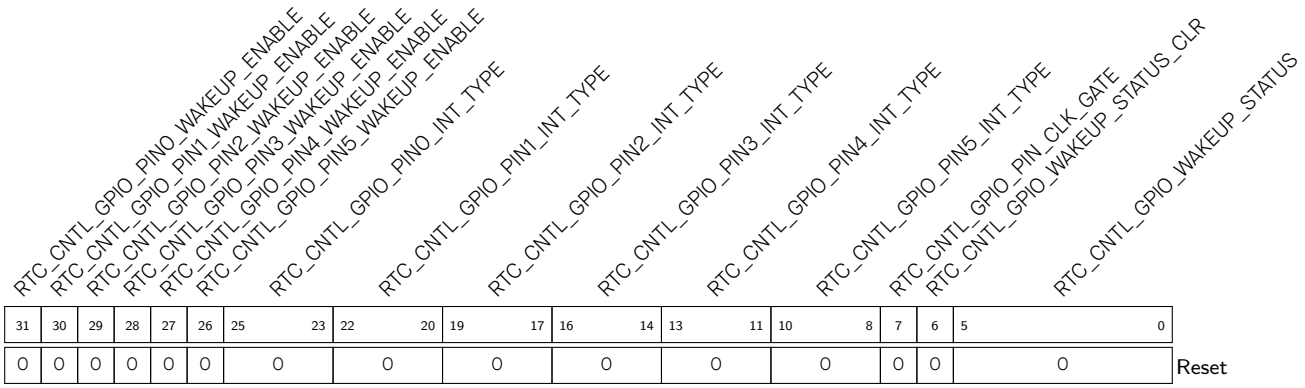
RTC_CNTL_FORCE_DOWNLOAD_BOOT 写 1 强制芯片从下载模式启动。(R/W)

Register 9.46. RTC_CNTL_SLP_WAKEUP_CAUSE_REG (0x00E4)



RTC_CNTL_WAKEUP_CAUSE 存储唤醒原因。(R/W)

Register 9.47. RTC_CNTL_CNTL_GPIO_WAKEUP_REG (0x00FC)



RTC_CNTL_GPIO_WAKEUP_STATUS 写 1 设置 RTC GPIO 唤醒标志。(R/W)

RTC_CNTL_GPIO_WAKEUP_STATUS_CLR 写 1 清除 RTC GPIO 标志。(R/W)

RTC_CNTL_GPIO_PIN_CLK_GATE 写 1 使能 RTC GPIO 时钟门控。(R/W)

RTC_CNTL_GPIO_PIN5_INT_TYPE 配置 GPIO 5 的唤醒类型。(R/W)

RTC_CNTL_GPIO_PIN4_INT_TYPE 配置 GPIO 4 的唤醒类型。(R/W)

RTC_CNTL_GPIO_PIN3_INT_TYPE 配置 GPIO 3 的唤醒类型。(R/W)

RTC_CNTL_GPIO_PIN2_INT_TYPE 配置 GPIO 2 的唤醒类型。(R/W)

RTC_CNTL_GPIO_PIN1_INT_TYPE 配置 GPIO 1 的唤醒类型。(R/W)

RTC_CNTL_GPIO_PIN0_INT_TYPE 配置 GPIO 0 的唤醒类型。(R/W)

RTC_CNTL_GPIO_PIN5_WAKEUP_ENABLE 写 1 使能从 RTC GPIO 5 唤醒。(R/W)

RTC_CNTL_GPIO_PIN4_WAKEUP_ENABLE 写 1 使能从 RTC GPIO 4 唤醒。(R/W)

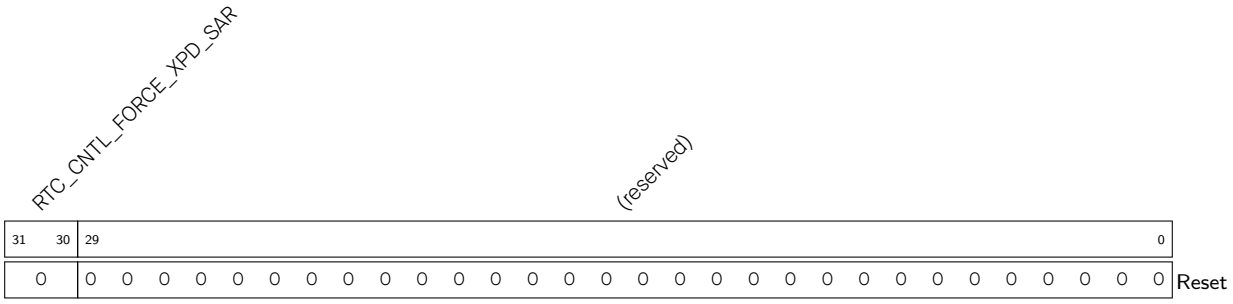
RTC_CNTL_GPIO_PIN3_WAKEUP_ENABLE 写 1 使能从 RTC GPIO 3 唤醒。(R/W)

RTC_CNTL_GPIO_PIN2_WAKEUP_ENABLE 写 1 使能从 RTC GPIO 2 唤醒。(R/W)

RTC_CNTL_GPIO_PIN1_WAKEUP_ENABLE 写 1 使能从 RTC GPIO 1 唤醒。(R/W)

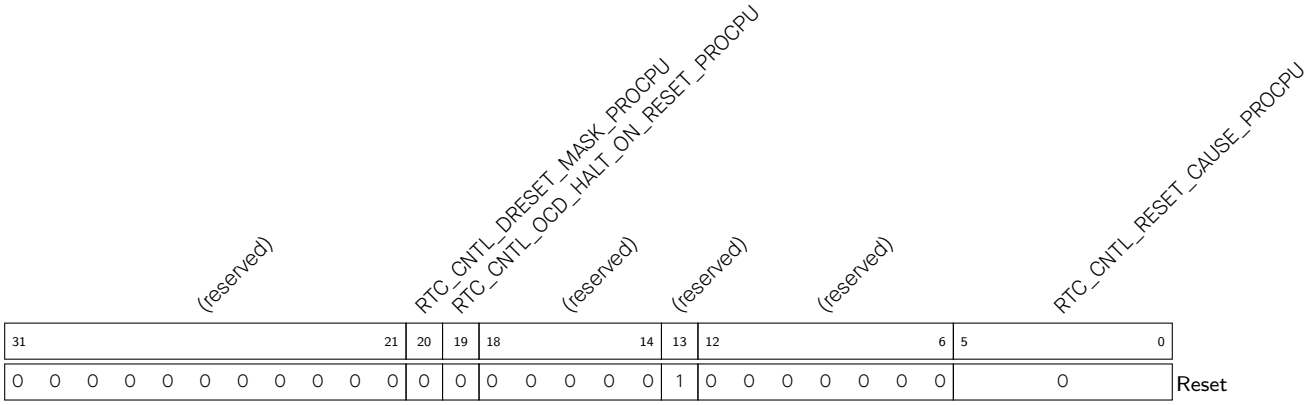
RTC_CNTL_GPIO_PIN0_WAKEUP_ENABLE 写 1 使能从 RTC GPIO 0 唤醒。(R/W)

Register 9.48. RTC_CNTL_CNTL_SENSOR_CTRL_REG (0x0108)



RTC_CNTL_FORCE_XPD_SAR 配置此字段强制打开 SAR ADC。(R/W)

Register 9.49. RTC_CNTL_RESET_STATE_REG (0x0030)

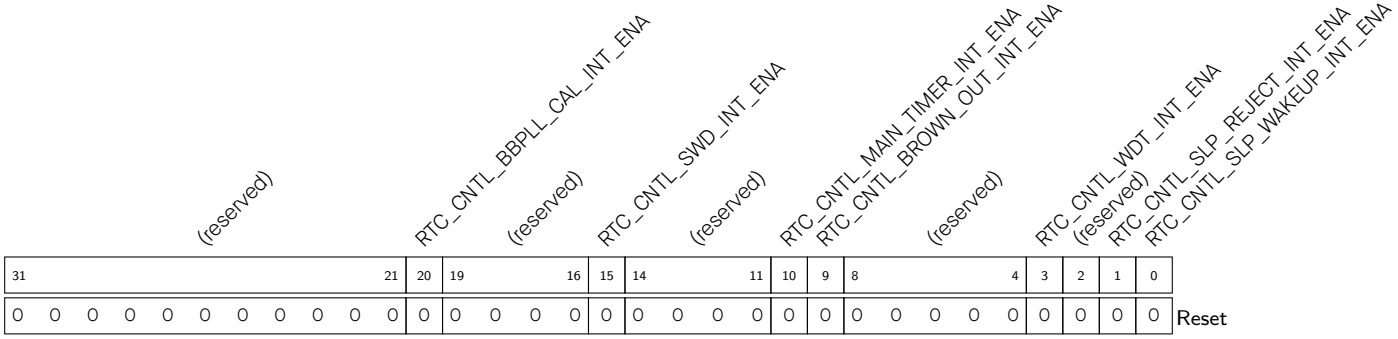


RTC_CNTL_RESET_CAUSE_PROCPU 表示 CPU 复位原因。(R/W)

RTC_CNTL_OCD_HALT_ON_RESET_PROCPU 写 1 在 CPU 重置时将 CPU 置于暂停状态。(R/W)

RTC_CNTL_DRESET_MASK_PROCPU 写 1 绕过 D-reset。(R/W)

Register 9.50. RTC_CNTL_INT_ENA_RTC_REG (0x0038)



RTC_CNTL_SLP_WAKEUP_INT_ENA 写 1 使能在芯片从睡眠中唤醒时发送中断。(R/W)

RTC_CNTL_SLP_REJECT_INT_ENA 写 1 使能在芯片拒绝睡眠时发送中断。(R/W)

RTC_CNTL_WDT_INT_ENA 写 1 使能 RTC WDT 中断。(R/W)

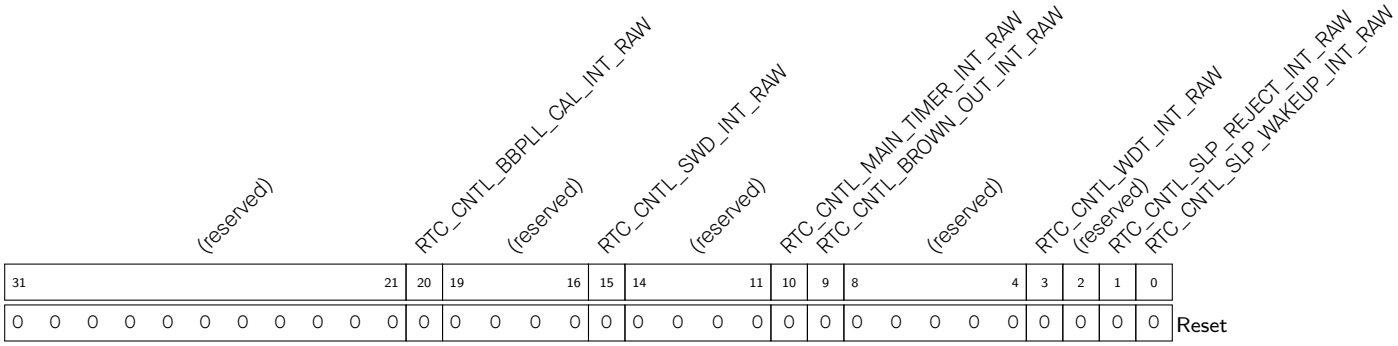
RTC_CNTL_BROWN_OUT_INT_ENA 写 1 使能欠压掉电中断。(R/W)

RTC_CNTL_MAIN_TIMER_INT_ENA 写 1 使能 RTC 定时器中断。(R/W)

RTC_CNTL_SWD_INT_ENA 写 1 使能超级看门狗中断。(R/W)

RTC_CNTL_BBPLL_CAL_INT_ENA 写 1 使能在 BB_PLL 调用结束时发送中断。(R/W)

Register 9.51. RTC_CNTL_INT_RAW_RTC_REG (0x003C)



RTC_CNTL_SLP_WAKEUP_INT_RAW 表示芯片从睡眠状态唤醒时触发中断的原始中断位。(R/W)

RTC_CNTL_SLP_REJECT_INT_RAW 表示芯片拒绝睡眠时触发中断的原始中断位。(R/W)

RTC_CNTL_WDT_INT_RAW 表示看门狗中断的原始中断位。(R/W)

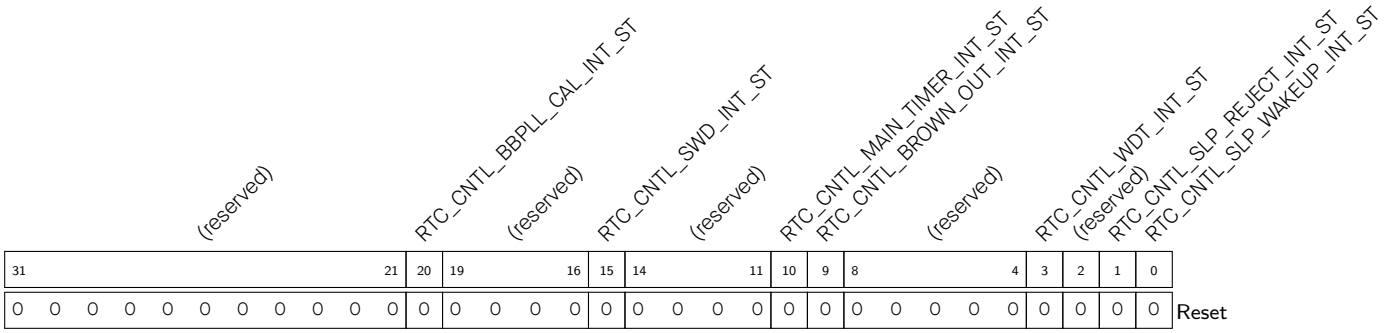
RTC_CNTL_BROWN_OUT_INT_RAW 表示欠压监测中断的原始中断位。(R/W)

RTC_CNTL_MAIN_TIMER_INT_RAW 表示 RTC 主定时器中断的原始中断位。(R/W)

RTC_CNTL_SWD_INT_RAW 表示超级看门狗中断的原始中断位。(R/W)

RTC_CNTL_BBPLL_CAL_INT_RAW 表示在 BB_PLL 调用结束时发送中断的原始中断位。(R/W)

Register 9.52. RTC_CNTL_INT_ST_RTC_REG (0x0040)



RTC_CNTL_SLP_WAKEUP_INT_ST 表示芯片从睡眠状态唤醒时触发中断的中断状态。(R/W)

RTC_CNTL_SLP_REJECT_INT_ST 表示芯片拒绝进入睡眠时触发中断的中断状态。(R/W)

RTC_CNTL_WDT_INT_ST 表示 RTC 看门狗中断的中断状态。(R/W)

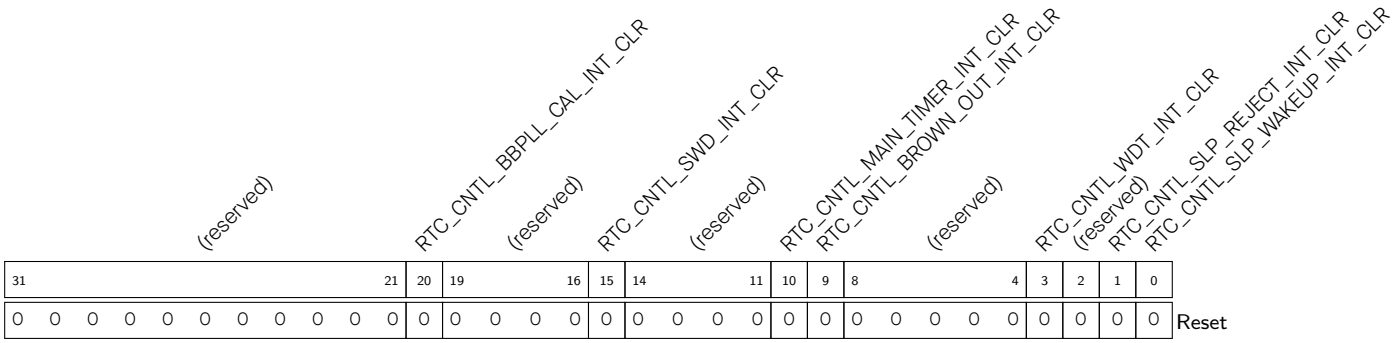
RTC_CNTL_BROWN_OUT_INT_ST 表示欠压掉电中断的中断状态。(R/W)

RTC_CNTL_MAIN_TIMER_INT_ST 表示 RTC 主定时器中断的中断状态。(R/W)

RTC_CNTL_SWD_INT_ST 表示超级看门狗中断的中断状态。(R/W)

RTC_CNTL_BBPLL_CAL_INT_ST 表示在 BB_PLL 调用结束时发送中断的中断状态。(R/W)

Register 9.53. RTC_CNTL_INT_CLR_RTC_REG (0x0044)



RTC_CNTL_SLP_WAKEUP_INT_CLR 写 1 清除在芯片从睡眠状态唤醒时触发的中断。(R/W)

RTC_CNTL_SLP_REJECT_INT_CLR 写 1 清除芯片拒绝进入睡眠时触发的中断。(R/W)

RTC_CNTL_WDT_INT_CLR 写 1 清除 RTC 看门狗中断。(R/W)

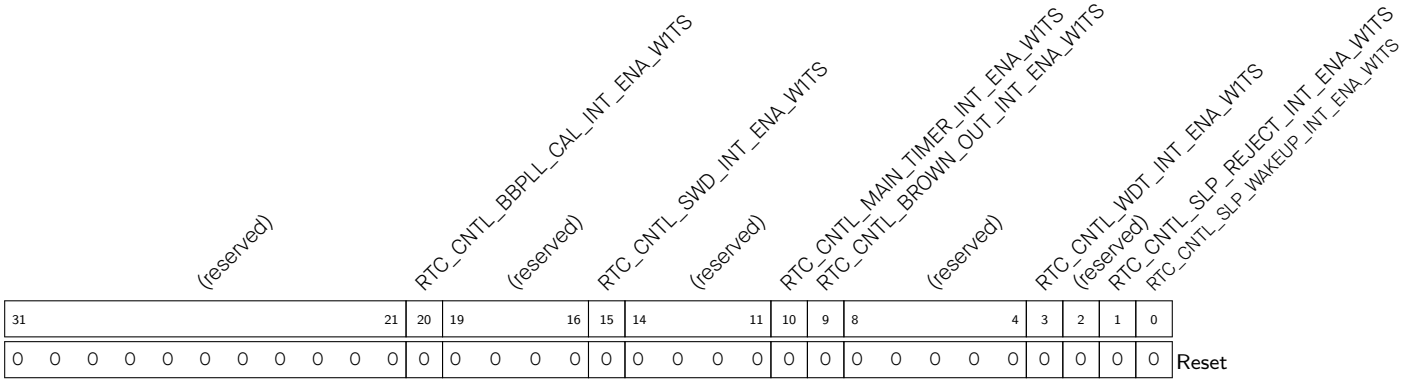
RTC_CNTL_BROWN_OUT_INT_CLR 写 1 清除欠压监测中断。(R/W)

RTC_CNTL_MAIN_TIMER_INT_CLR 写 1 清除 RTC 主定时器中断。(R/W)

RTC_CNTL_SWD_INT_CLR 写 1 清除超级看门狗中断。(R/W)

RTC_CNTL_BBPLL_CAL_INT_CLR 写 1 清除在 BB_PLL 调用结束时触发的中断。(R/W)

Register 9.54. RTC_CNTL_INT_ENA_RTC_W1TS_REG (0x00EC)



RTC_CNTL_SLP_WAKEUP_INT_ENA_W1TS 写 1 使能在芯片从睡眠中唤醒时发送中断。此位一旦置 1，则 RTC_CNTL_SLP_WAKEUP_INT_ENA 也将置 1。(R/W)

RTC_CNTL_SLP_REJECT_INT_ENA_W1TS 写 1 使能在芯片拒绝睡眠时发送中断。此位一旦置 1，则 RTC_CNTL_SLP_REJECT_INT_ENA 也将置 1。(R/W)

RTC_CNTL_WDT_INT_ENA_W1TS 写 1 使能 RTC 看门狗。此位一旦置 1，则 RTC_CNTL_WDT_INT_ENA 也将置 1。(R/W)

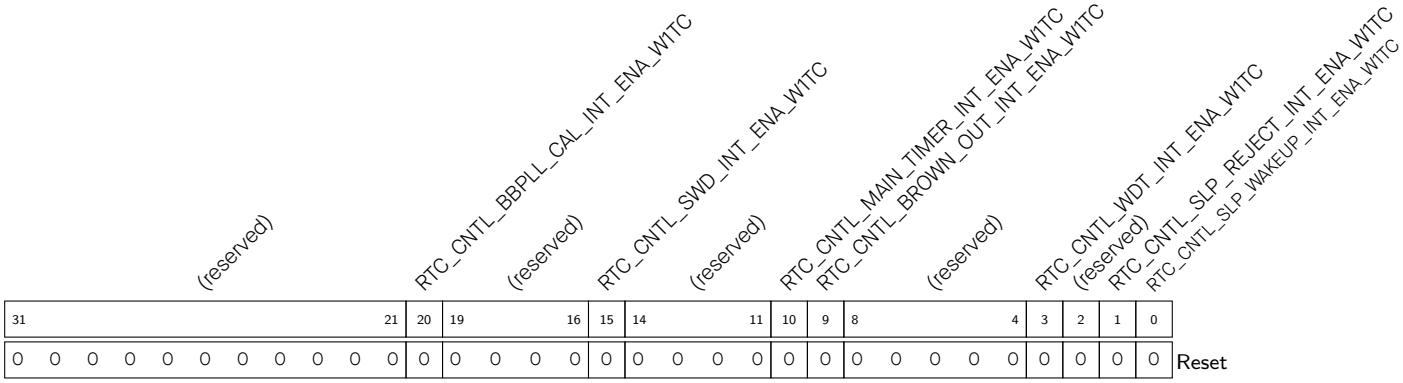
RTC_CNTL_BROWN_OUT_INT_ENA_W1TS 写 1 使能欠压掉电中断。此位一旦置 1，则 RTC_CNTL_BROWN_OUT_INT_ENA 也将置 1。(R/W)

RTC_CNTL_MAIN_TIMER_INT_ENA_W1TS 写 1 使能 RTC 主定时器中断。此位一旦置 1，则 RTC_CNTL_MAIN_TIMER_INT_ENA 也将置 1。(R/W)

RTC_CNTL_SWD_INT_ENA_W1TS 写 1 使能超级看门狗中断。此位一旦置 1，则 RTC_CNTL_SWD_INT_ENA 也将置 1。(R/W)

RTC_CNTL_BBPLL_CAL_INT_ENA_W1TS 写 1 使能在 BB_PLL 调用结束时发送中断。此位一旦置 1，则 RTC_CNTL_BBPLL_CAL_INT_ENA 也将置 1。(R/W)

Register 9.55. RTC_CNTL_INT_ENA_RTC_W1TC_REG (0x00F0)



RTC_CNTL_SLP_WAKEUP_INT_ENA_W1TC 写 1 禁用在芯片从睡眠中唤醒时发送中断。此位一旦置 1，则 RTC_CNTL_SLP_WAKEUP_INT_CLR 将清零。(R/W)

RTC_CNTL_SLP_REJECT_INT_ENA_W1TC 写 1 禁用在芯片拒绝睡眠时发送中断。此位一旦置 1，则 RTC_CNTL_SLP_REJECT_INT_CLR 将清零。(R/W)

RTC_CNTL_WDT_INT_ENA_W1TC 写 1 禁用 RTC 看门狗。此位一旦置 1，则 RTC_CNTL_WDT_INT_CLR 将清零。(R/W)

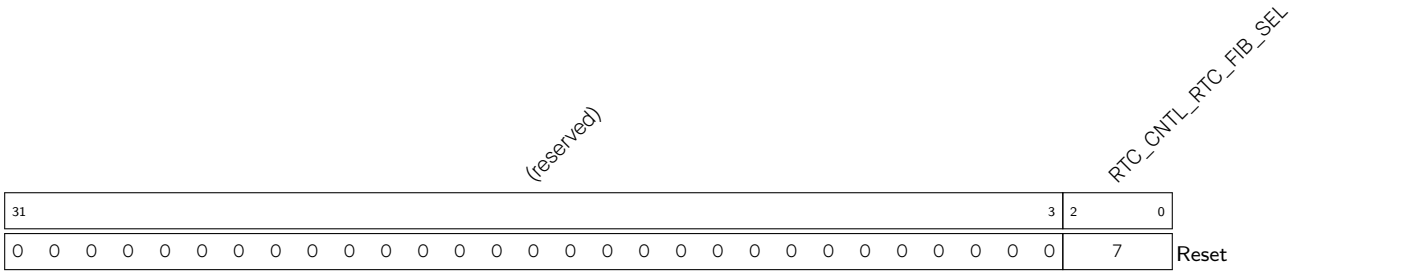
RTC_CNTL_BROWN_OUT_INT_ENA_W1TC 写 1 禁用欠压掉电中断。此位一旦置 1，则 RTC_CNTL_BROWN_OUT_INT_CLR 将清零。(R/W)

RTC_CNTL_MAIN_TIMER_INT_ENA_W1TC 写 1 禁用 RTC 主定时器中断。此位一旦置 1，则 RTC_CNTL_MAIN_TIMER_INT_CLR 将清零。(R/W)

RTC_CNTL_SWD_INT_ENA_W1TC 写 1 禁用超级看门狗中断。此位一旦置 1，则 RTC_CNTL_SWD_INT_CLR 将清零。(R/W)

RTC_CNTL_BBPLL_CAL_INT_ENA_W1TC 写 1 禁用在 BB_PLL 调用结束时发送中断。此位一旦置 1，则 RTC_CNTL_BBPLL_CAL_INT_CLR 将清零。(R/W)

Register 9.56. RTC_CNTL_FIB_SEL_REG (0x00F8)



RTC_CNTL_RTC_FIB_SEL 配置欠压监测器。(R/W)

10 系统定时器 (SYSTIMER)

10.1 概述

ESP8684 芯片内置一组 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。

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 10-1。

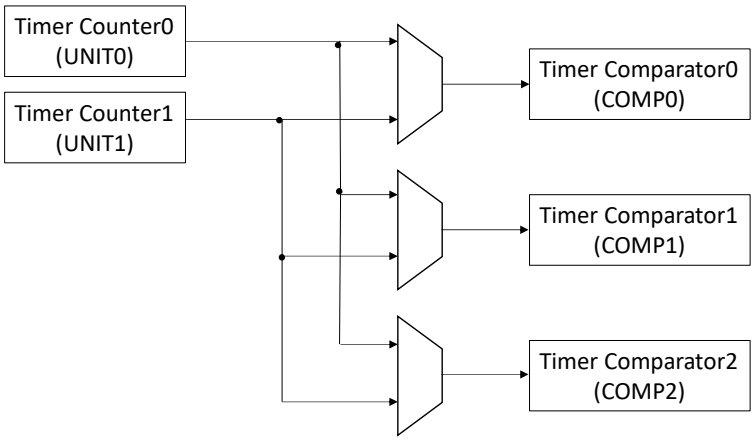


图 10-1. 系统定时器结构图

10.2 主要特性

系统定时器具有如下特性：

- 由两个 52 位计数器和三个 52 位比较器组成
- 软件通过 APB_CLK 访问寄存器
- 计数采用 CNT_CLK 时钟，两次计数周期的平均频率为 16 MHz
- CNT_CLK 的时钟源为 XTAL_CLK (40 MHz)
- 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
- 三个比较器可根据设置的报警值 (t) 或报警周期 (δt) 生成三个独立中断
- 支持软件配置基准计数值。例如，支持从 Light-sleep 唤醒之后，系统定时器通过软件加载 RTC 定时器记录的睡眠时间，并进行补偿
- CPU 处于停止状态或处于在线调试状态时，系统定时器可选择停止运行或继续运行

10.3 时钟源选择

计数器和比较器使用 XTAL_CLK 用作时钟源。XTAL_CLK 经分数分频后，在一个计数周期生成频率为 $f_{XTAL\_CLK}/3$ 的时钟信号，然后在另一个计数周期生成频率为 $f_{XTAL\_CLK}/2$ 的时钟信号。因此，计数器使用的时钟 CNT_CLK，其实际平均频率为 $f_{XTAL\_CLK}/2.5$，即 16 MHz，见图 10-2。每个 CNT_CLK 时钟周期，计数递增 1/16 μs，即 16 个周期递增 1 μs。

配置寄存器等软件操作则是由 APB_CLK 提供时钟信号。更多有关 APB_CLK 的信息，见章节 6 复位和时钟。

用户可使用以下系统寄存器的相关位来控制系统定时器：

- 置位寄存器 SYSTEM_PERIP_CLK_ENO_REG 中 SYSTEM_SYSTIMER_CLK_EN 位使能系统定时器的 APB_CLK 信号；
- 置位寄存器 SYSTEM_PERIP_CLK_ENO_REG 中 SYSTEM_SYSTIMER_RST 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 13 系统寄存器 (SYSTEM) 中表：外设时钟门控与复位控制位。

10.4 功能描述

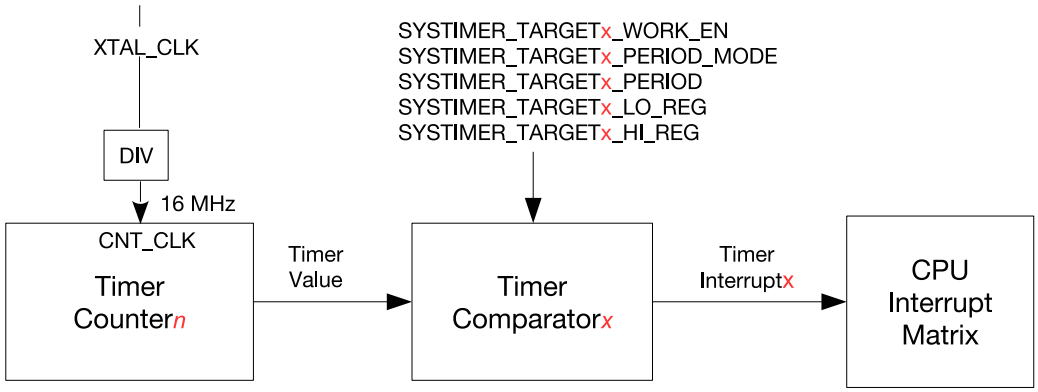


图 10-2. 系统定时器生成报警

图 10-2 展示了系统定时器生成报警的过程。在上述过程中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。

10.4.1 计数器

系统定时器提供两个 52 位计数器，下文用 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 中下面两个位来控制计数器 UNIT*n*：

- SYSTIMER_TIMER_UNIT*n*_WORK_EN：置位此位，使能计数器 UNIT*n*；
- SYSTIMER_TIMER_UNIT*n*_COREO_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。

UNIT*n* 的具体配置见下表，其中假设 CPU 当前状态为停止工作。

表 10-1. UNIT*n* 配置控制位

SYSTIMER_TIMER_UNIT*n*_WORK_EN	SYSTIMER_TIMER_UNIT*n*_COREO_STALL_EN	计数器 UNIT*n*
0	x *	未处于工作状态。
1	1	暂停计数，但 CPU 苏醒后，会继续计数。
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，则计数值将保持不变，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 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，则将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。SYSTIMER_TIMER_UNIT*n*_VALUE_LO 和 SYSTIMER_TIMER_UNIT*n*_VALUE_HI 寄存器中的值保持不变，直至下次更新事件发生。

10.4.2 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：选择周期报警模式
- 0：选择单次报警模式

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，将再次触发一次报警中断，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，则触发一次报警。与周期报警模式不同，单次报警模式仅生成一次报警中断。

用户可配置寄存器 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：选择与计数器 UNIT*1* 的计数值进行比较
- 0：选择与计数器 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...)。

实际计数值等于报警值 (t)，或等于 t1 + n*δt (n = 1,2,3...)，则触发一次报警中断。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，或当前计数值超过设定的报警值 (t) 一定范围 (0 ~ 2^{51} -1)，则也将立即触发中断。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 10-2. 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

10.4.3 同步操作

软件操作与计数器和比较器工作在不同时钟频率下，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写入合适的值，见表 10-3 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 10-3 第二列。

表 10-3. 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

10.4.4 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。比较器开始触发报警，即拉高中断信号。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。

10.5 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。

10.5.1 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）。

10.5.2 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；
2. 读取当前计数器的值，步骤见章节 10.5.1。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) 则触发一次报警中断 SYSTIMER_TARGET*x*_INT。

10.5.3 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...)，则触发一次 SYSTIMER_TARGET*x*_INT 中断。

10.5.4 唤醒后时间补偿

1. 在芯片进入 Light-sleep 之前，用户需配置 RTC 定时器用于精确记录睡眠时间，见低功耗管理章节；
2. 系统从 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 10.5.1；
4. 将 RTC 记录的睡眠时间，单位：RTC_SLOW_CLK 周期，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 即可。
5. 将 RTC 定时器转换后的值加到系统定时器当前计数值：
   - 将计算所得值，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。这样即可完成系统定时器更新。

10.6 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 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

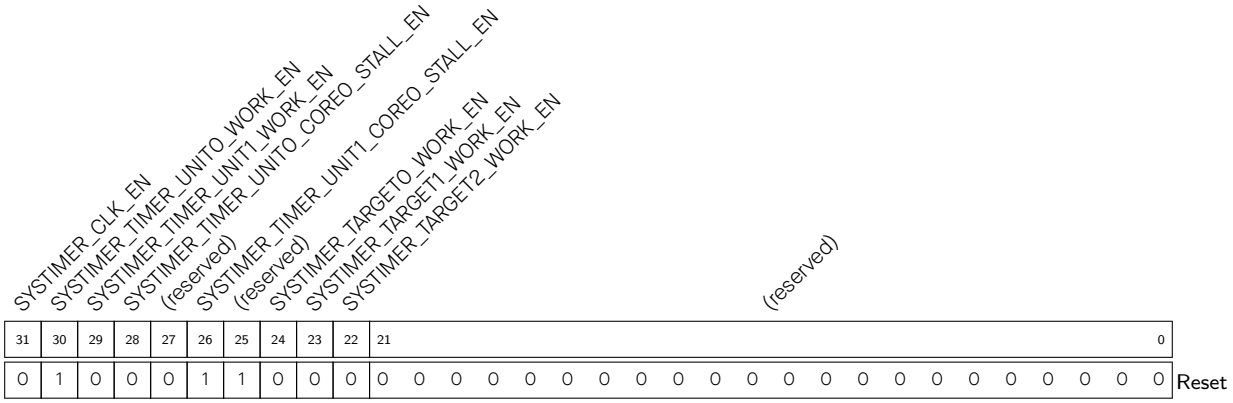
名称	描述	地址	访问
时钟控制寄存器			
SYSTIMER_CONF_REG	配置系统定时器的时钟	0x0000	R/W
UNIT*0* 控制和配置寄存器			
SYSTIMER_UNIT0_OP_REG	读取 UNIT0 的值到相应寄存器	0x0004	varies
SYSTIMER_UNIT0_LOAD_HI_REG	待装载至 UNIT0 的值，高 20 位	0x000C	R/W
SYSTIMER_UNIT0_LOAD_LO_REG	待装载至 UNIT0 的值，低 32 位	0x0010	R/W
SYSTIMER_UNIT0_VALUE_HI_REG	UNIT0 的读值，高 20 位	0x0040	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

名称	描述	地址	访问
SYSTIMER_INT_ST_REG	系统定时器的中断状态寄存器	0x0070	RO
版本寄存器			
SYSTIMER_DATE_REG	版本控制寄存器	0x00FC	R/W

10.7 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 3 系统和存储器 中的表 3-3。

Register 10.1. SYSTIMER_CONF_REG (0x0000)



SYSTIMER_TARGET2_WORK_EN 置位此位，使能比较器 COMP2。(R/W)

SYSTIMER_TARGET1_WORK_EN 置位此位，使能比较器 COMP1。(R/W)

SYSTIMER_TARGET0_WORK_EN 置位此位，使能比较器 COMP0。(R/W)

SYSTIMER_TIMER_UNIT1_CORE0_STALL_EN 置位此位，则如果 CPU 停止工作，计数器 UNIT1 也将停止工作。(R/W)

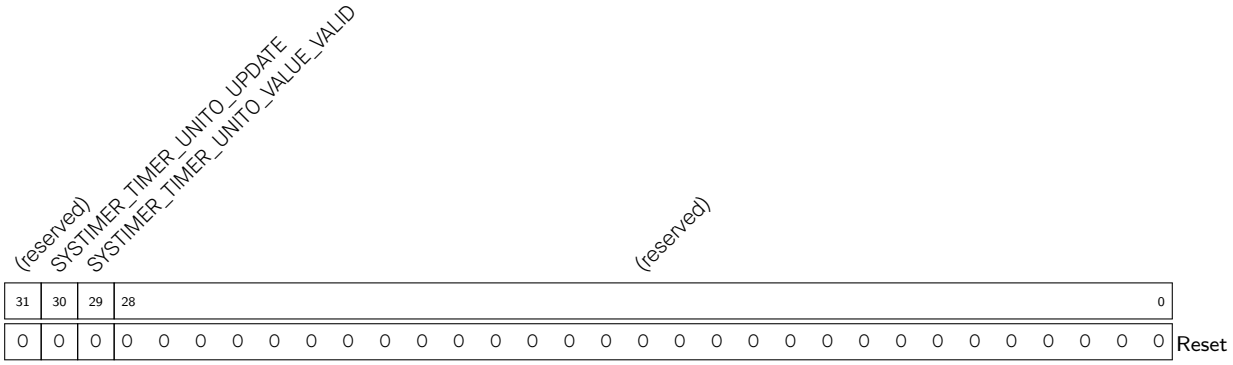
SYSTIMER_TIMER_UNIT0_CORE0_STALL_EN 置位此位，则如果 CPU 停止工作，计数器 UNIT0 也将停止工作。(R/W)

SYSTIMER_TIMER_UNIT1_WORK_EN 使能计数器 UNIT1。(R/W)

SYSTIMER_TIMER_UNIT0_WORK_EN 使能计数器 UNIT0。(R/W)

SYSTIMER_CLK_EN 寄存器时钟门控。1：持续开启读写寄存器的时钟；0：只在读写寄存器时打开所需时钟。(R/W)

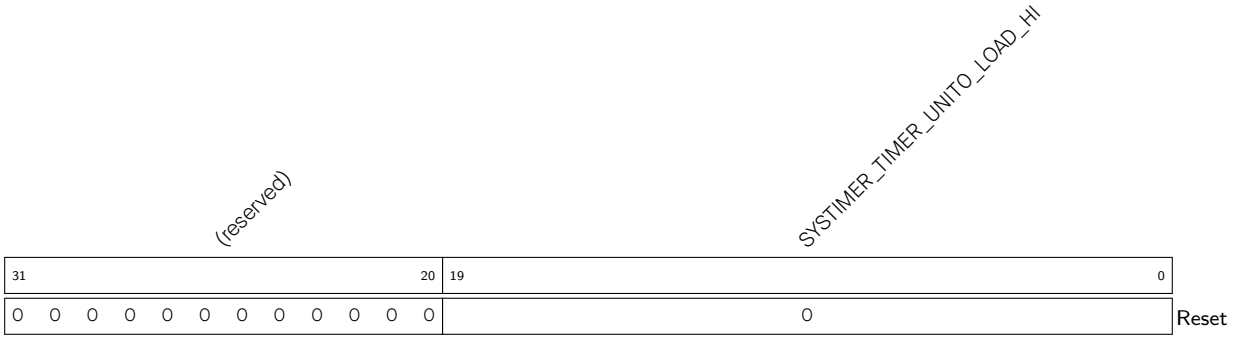
Register 10.2. SYSTIMER_UNIT0_OP_REG (0x0004)



SYSTIMER_TIMER_UNIT0_VALUE_VALID 计数器 UNIT0 的值已同步读取至寄存器，且读值有效。(R/SS/WTC)

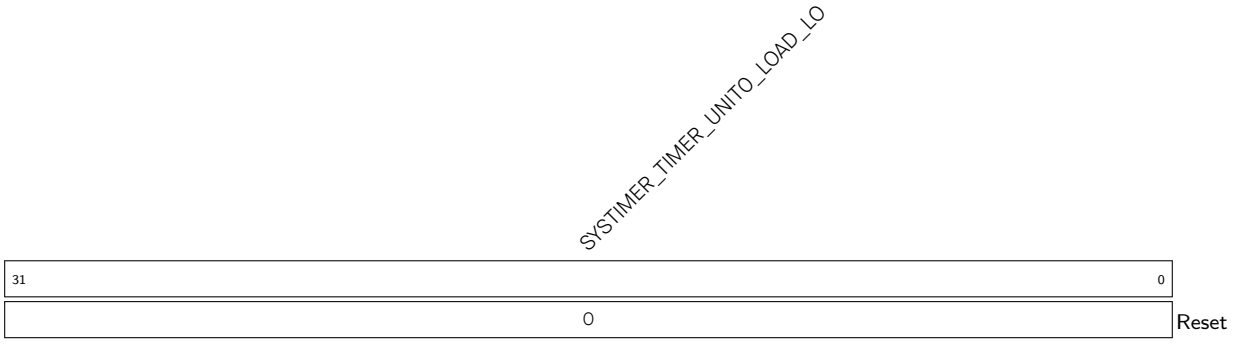
SYSTIMER_TIMER_UNIT0_UPDATE 读取计数器 UNIT0 的值到寄存器 SYSTIMER_TIMER_UNIT0_VALUE_HI 和 SYSTIMER_TIMER_UNIT0_VALUE_LO。(WT)

Register 10.3. SYSTIMER_UNIT0_LOAD_HI_REG (0x000C)



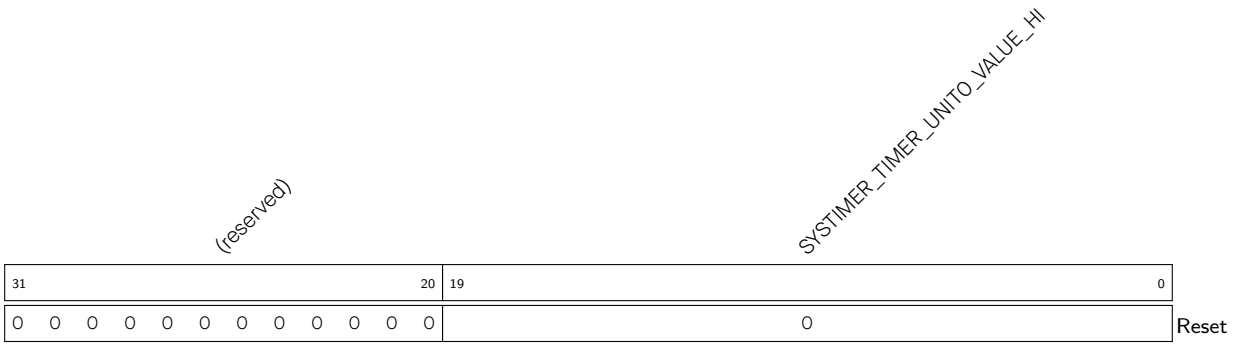
SYSTIMER_TIMER_UNIT0_LOAD_HI 待装载至计数器 UNIT0 的值，高 20 位。(R/W)

Register 10.4. SYSTIMER_UNIT0_LOAD_LO_REG (0x0010)



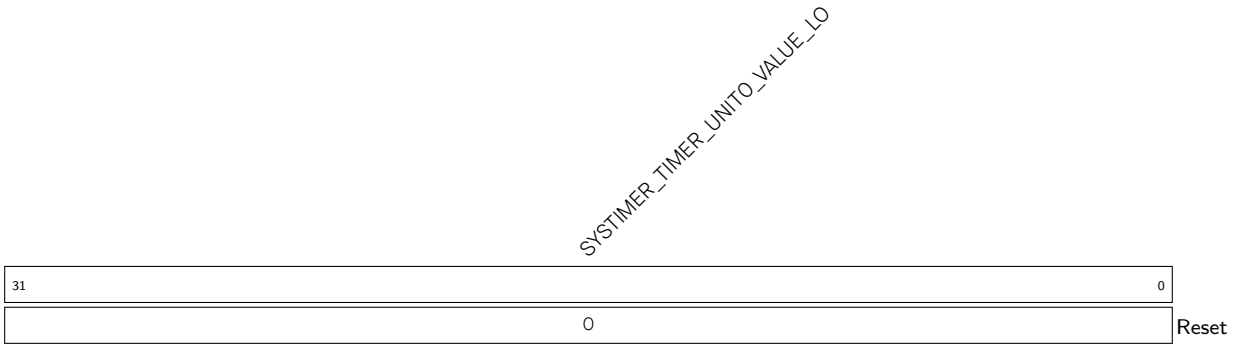
SYSTIMER_TIMER_UNIT0_LOAD_LO 待装载至计数器 UNIT0 的值，低 32 位。(R/W)

Register 10.5. SYSTIMER_UNIT0_VALUE_HI_REG (0x0040)



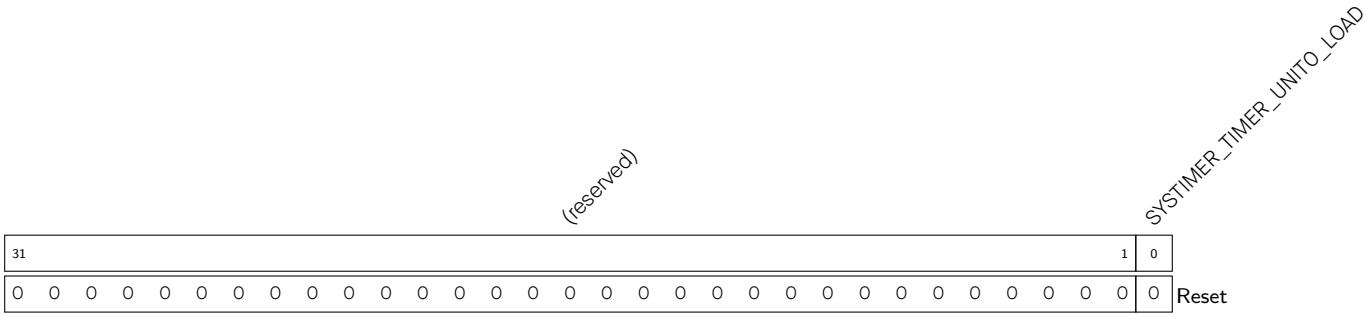
SYSTIMER_TIMER_UNIT0_VALUE_HI 计数器 UNIT0 的读数，高 20 位。(RO)

Register 10.6. SYSTIMER_UNIT0_VALUE_LO_REG (0x0044)



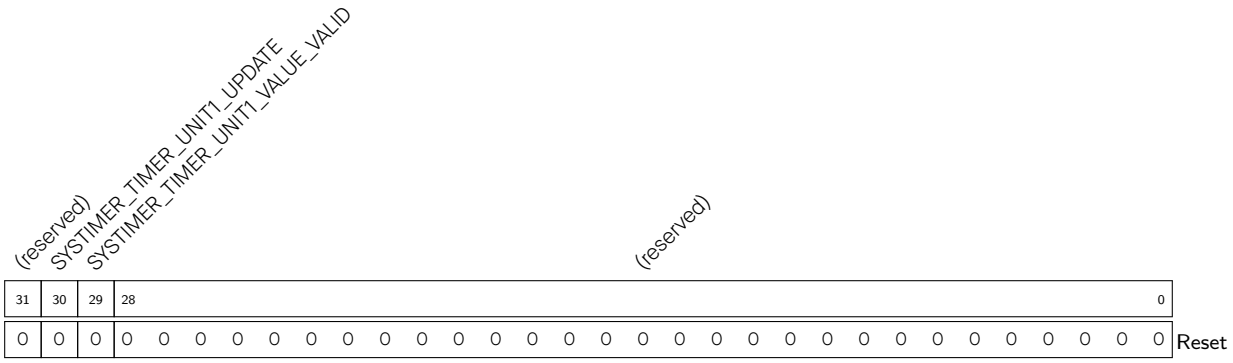
SYSTIMER_TIMER_UNIT0_VALUE_LO 计数器 UNIT0 的读数，低 32 位。(RO)

Register 10.7. SYSTIMER_UNIT0_LOAD_REG (0x005C)



SYSTIMER_TIMER_UNIT0_LOAD 计数器 UNIT0 的同步使能信号。置位此位，将重新装载寄存器 SYSTIMER_TIMER_UNIT0_LOAD_HI 和 SYSTIMER_TIMER_UNIT0_LOAD_LO 的值到计数器 UNIT0。(WT)

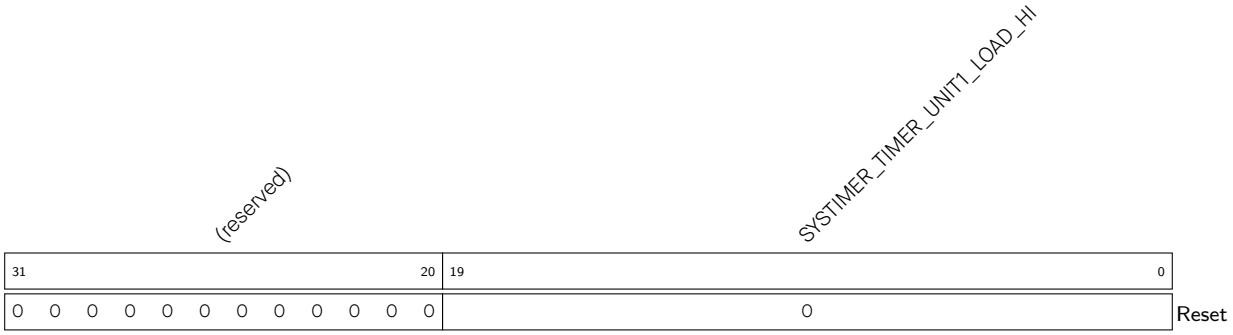
Register 10.8. SYSTIMER_UNIT1_OP_REG (0x0008)



SYSTIMER_TIMER_UNIT1_VALUE_VALID 计数器 UNIT1 的值已同步读取至寄存器，且读值有效。(R/SS/WTC)

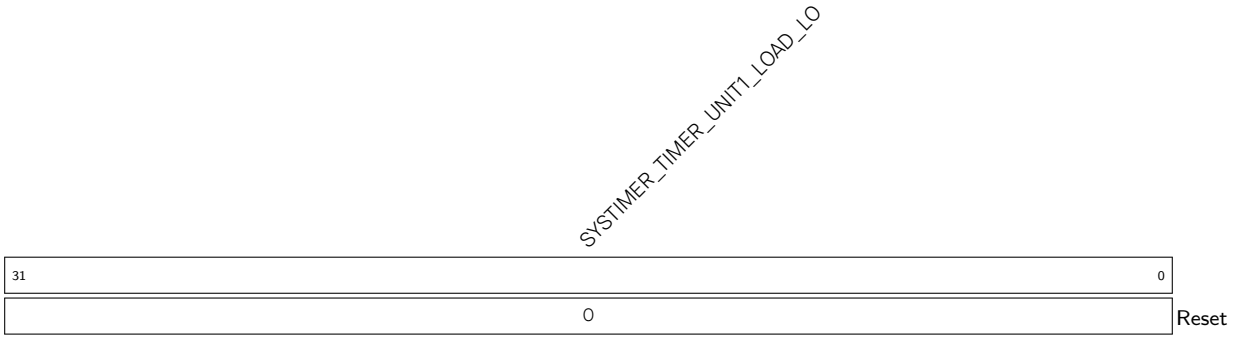
SYSTIMER_TIMER_UNIT1_UPDATE 将计数器 UNIT1 的值读取到寄存器 SYSTIMER_TIMER_UNIT1_VALUE_HI 和 SYSTIMER_TIMER_UNIT1_VALUE_LO。(WT)

Register 10.9. SYSTIMER_UNIT1_LOAD_HI_REG (0x0014)



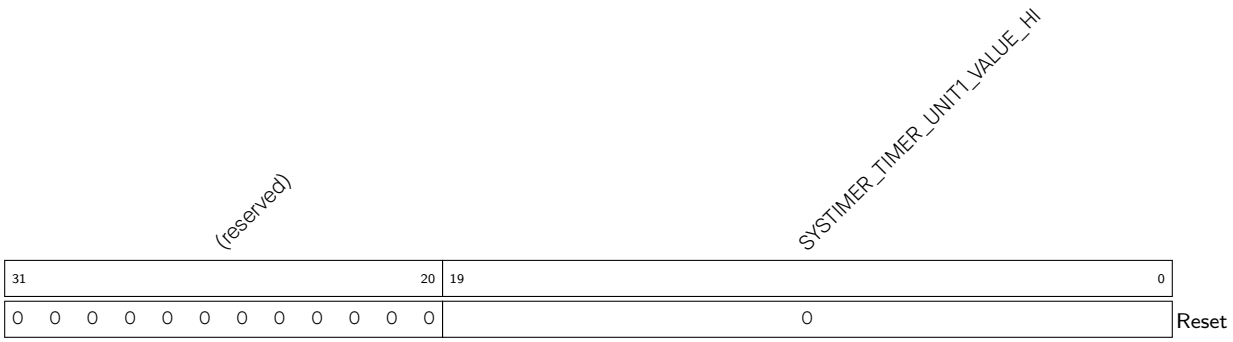
SYSTIMER_TIMER_UNIT1_LOAD_HI 待装载至计数器 UNIT1 的值，高 20 位。(R/W)

Register 10.10. SYSTIMER_UNIT1_LOAD_LO_REG (0x0018)



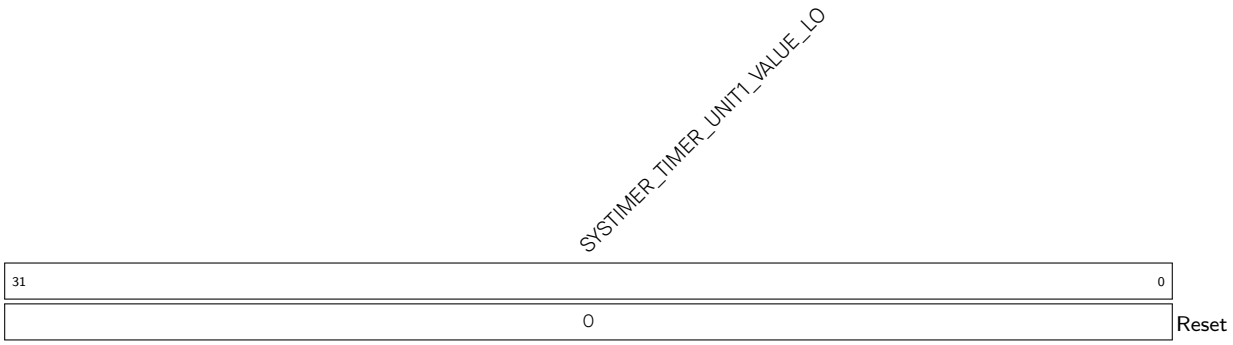
SYSTIMER_TIMER_UNIT1_LOAD_LO 待装载至计数器 UNIT1 的值，低 32 位。(R/W)

Register 10.11. SYSTIMER_UNIT1_VALUE_HI_REG (0x0048)



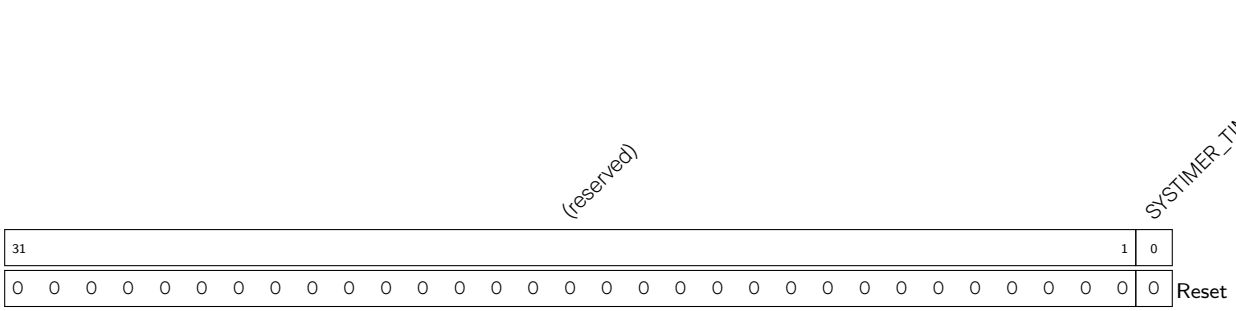
SYSTIMER_TIMER_UNIT1_VALUE_HI 计数器 UNIT1 的读数，高 20 位。(RO)

Register 10.12. SYSTIMER_UNIT1_VALUE_LO_REG (0x004C)



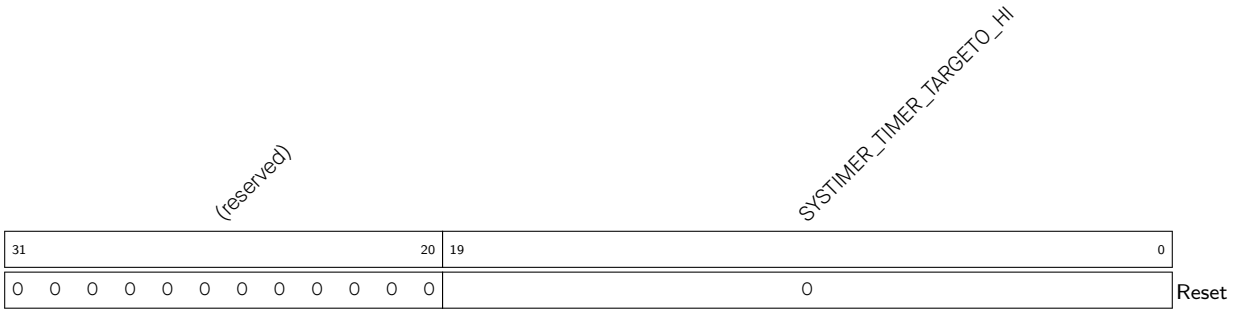
SYSTIMER_TIMER_UNIT1_VALUE_LO 计数器 UNIT1 的读数，低 32 位。(RO)

Register 10.13. SYSTIMER_UNIT1_LOAD_REG (0x0060)



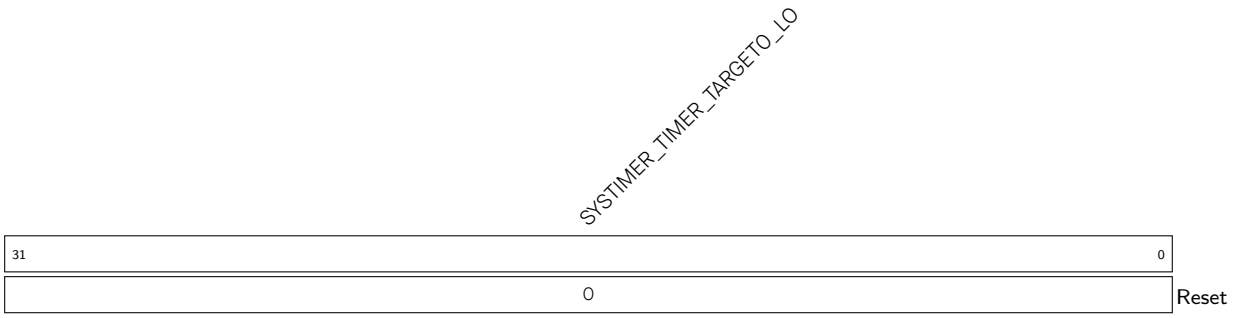
SYSTIMER_TIMER_UNIT1_LOAD 计数器 UNIT1 的同步使能信号。置位此位，将重新装载寄存器 SYSTIMER_TIMER_UNIT1_LOAD_HI 和 SYSTIMER_TIMER_UNIT1_LOAD_LO 的值到计数器 UNIT1。(WT)

Register 10.14. SYSTIMER_TARGET0_HI_REG (0x001C)



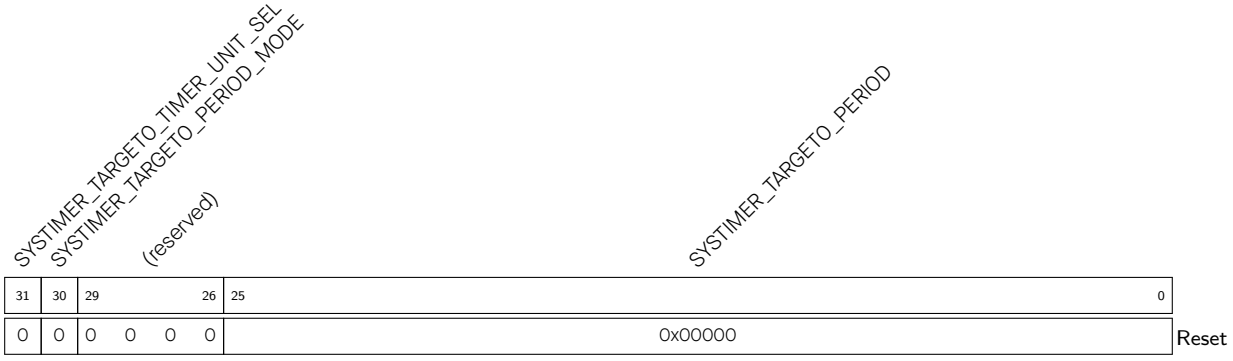
SYSTIMER_TIMER_TARGET0_HI 待装载至 COMP0 的报警值，高 20 位。(R/W)

Register 10.15. SYSTIMER_TARGET0_LO_REG (0x0020)



SYSTIMER_TIMER_TARGET0_LO 待装载至 COMP0 的报警值，低 32 位。(R/W)

Register 10.16. SYSTIMER_TARGET0_CONF_REG (0x0034)

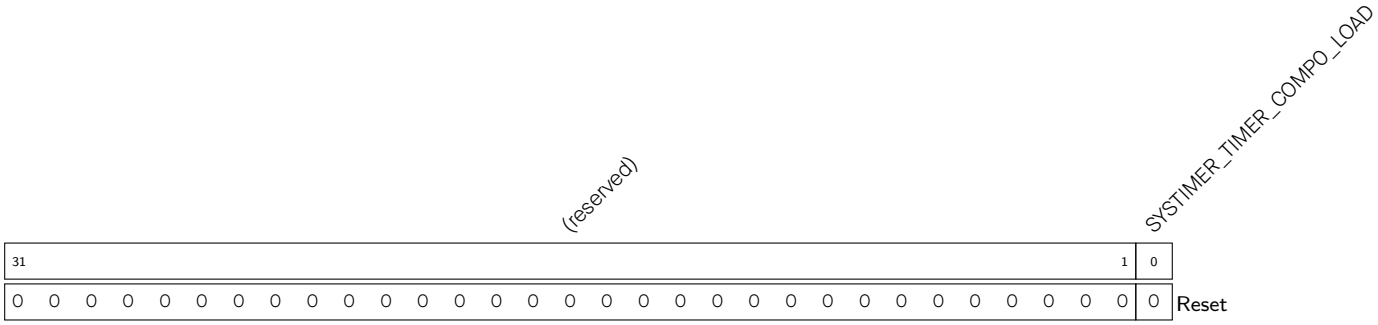


SYSTIMER_TARGET0_PERIOD 待装载至 COMP0 的报警周期。(R/W)

SYSTIMER_TARGET0_PERIOD_MODE 设置 COMP0 为周期报警模式。(R/W)

SYSTIMER_TARGET0_TIMER_UNIT_SEL 选择要与 COMP0 比较的计数器。(R/W)

Register 10.17. SYSTIMER_COMP0_LOAD_REG (0x0050)

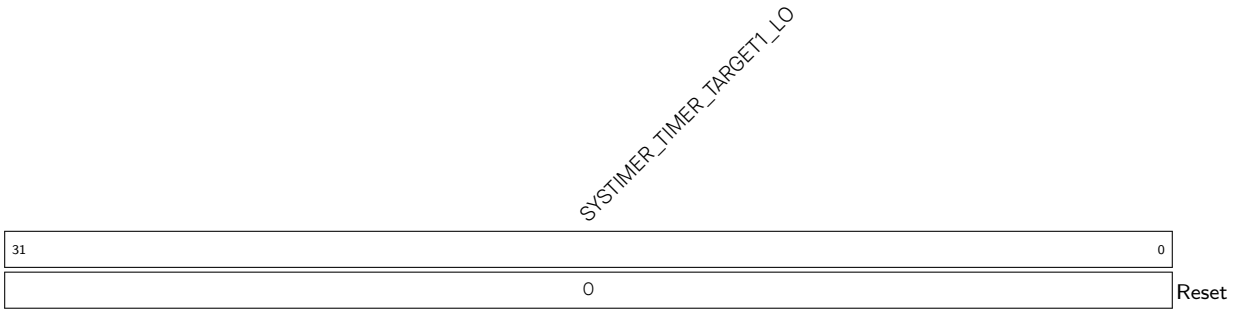


SYSTIMER_TIMER_COMP0_LOAD 比较器 COMP0 的同步使能信号。置位此位，将重新装载报警值或报警周期到 COMP0。(WT)

Register 10.18. SYSTIMER_TARGET1_HI_REG (0x0024)

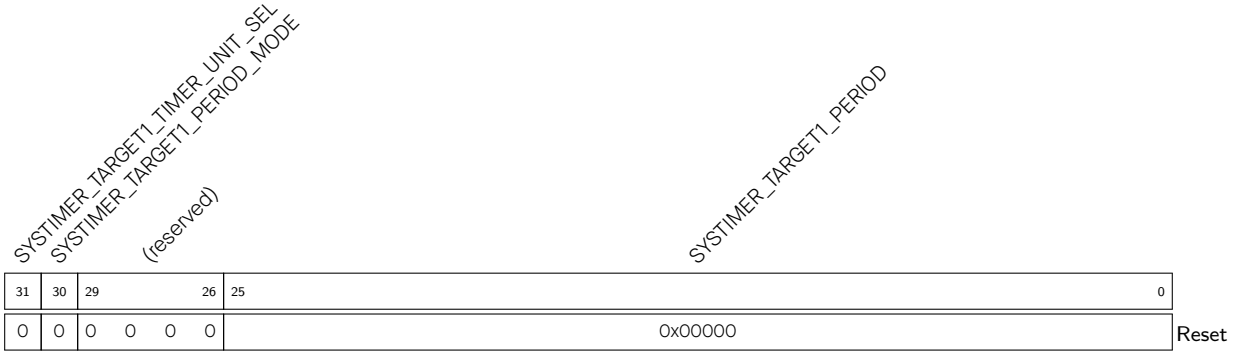
SYSTIMER_TIMER_TARGET1_HI 待装载至 COMP1 的报警值，高 20 位。(R/W)

Register 10.19. SYSTIMER_TARGET1_LO_REG (0x0028)



SYSTIMER_TIMER_TARGET1_LO 待装载至 COMP1 的报警值，低 32 位。(R/W)

Register 10.20. SYSTIMER_TARGET1_CONF_REG (0x0038)

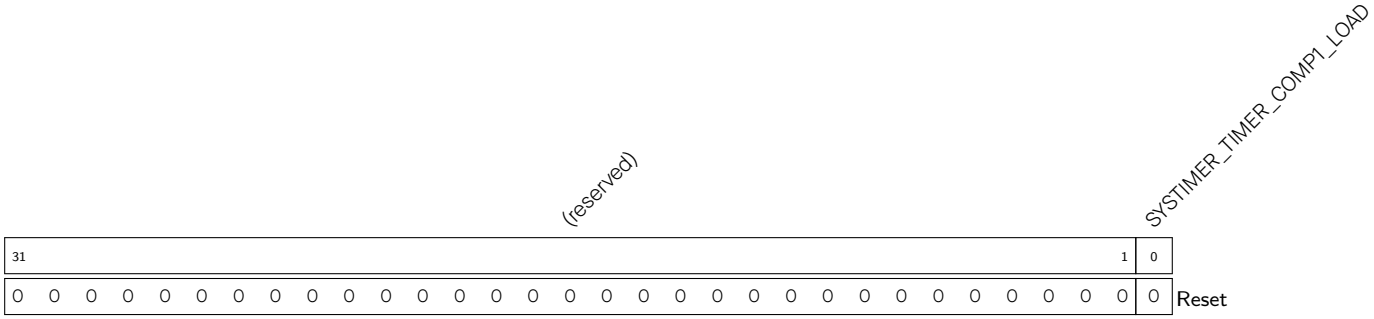


SYSTIMER_TARGET1_PERIOD 待装载至 COMP1 的报警周期。(R/W)

SYSTIMER_TARGET1_PERIOD_MODE 设置 COMP1 为周期报警模式。(R/W)

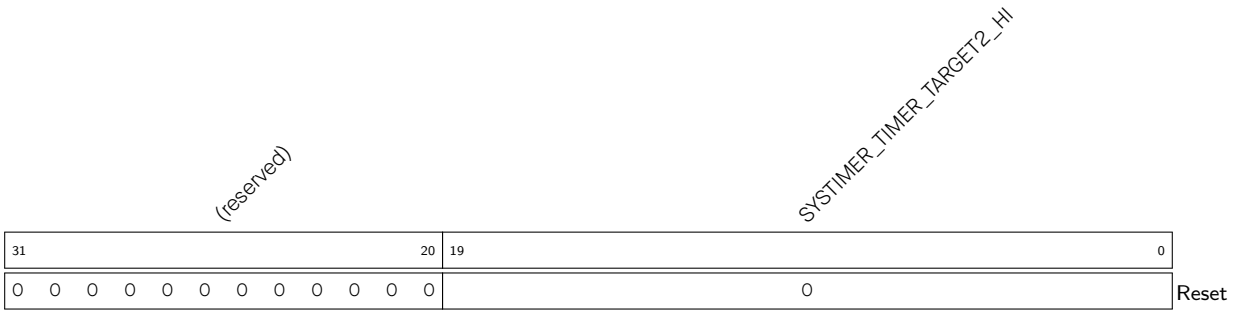
SYSTIMER_TARGET1_TIMER_UNIT_SEL 选择要与 COMP1 比较的计数器。(R/W)

Register 10.21. SYSTIMER_COMP1_LOAD_REG (0x0054)



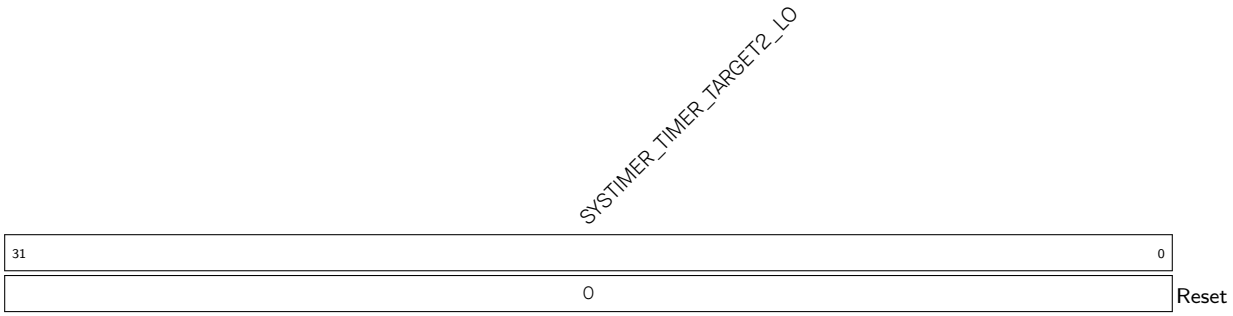
SYSTIMER_TIMER_COMP1_LOAD 比较器 COMP1 的同步使能信号。置位此位，将重新装载报警值或报警周期到 COMP1。(WT)

Register 10.22. SYSTIMER_TARGET2_HI_REG (0x002C)



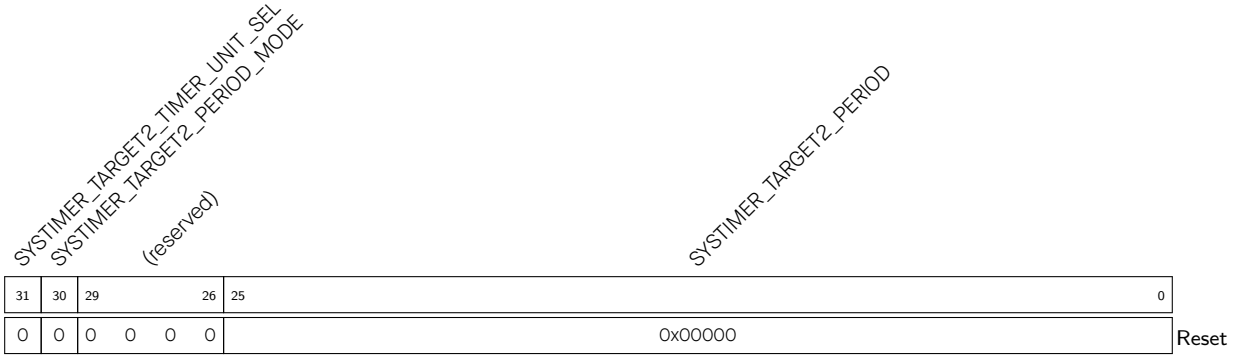
SYSTIMER_TIMER_TARGET2_HI 待装载至比较器 COMP2 的报警值，高 20 位。(R/W)

Register 10.23. SYSTIMER_TARGET2_LO_REG (0x0030)



SYSTIMER_TIMER_TARGET2_LO 待装载至比较器 COMP2 的报警值，低 32 位。(R/W)

Register 10.24. SYSTIMER_TARGET2_CONF_REG (0x003C)

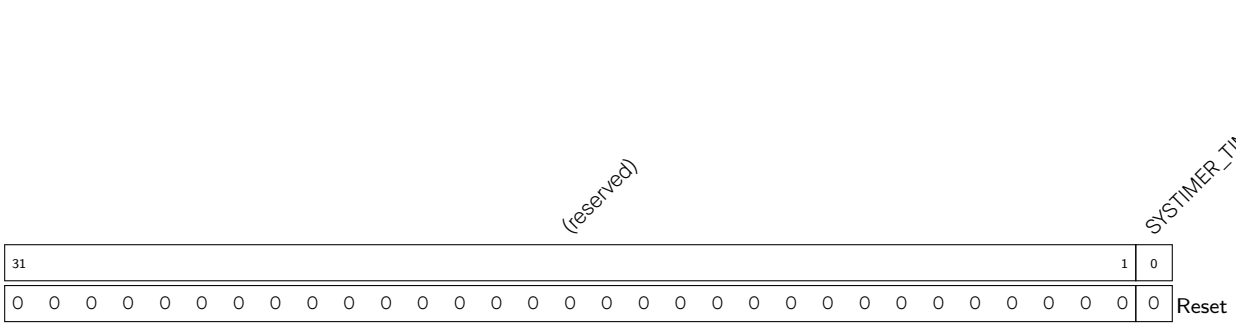


SYSTIMER_TARGET2_PERIOD 待装载至 COMP2 的报警周期。(R/W)

SYSTIMER_TARGET2_PERIOD_MODE 设置 COMP2 为周期报警模式。(R/W)

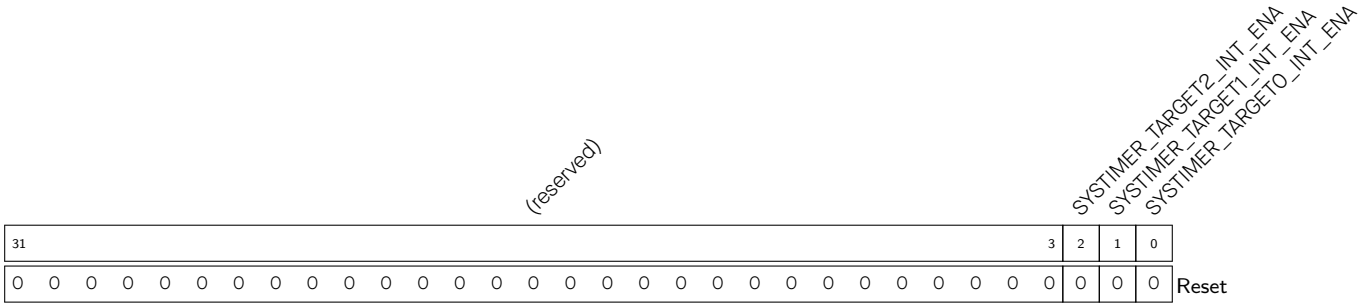
SYSTIMER_TARGET2_TIMER_UNIT_SEL 选择要与 COMP2 比较的计数器。(R/W)

Register 10.25. SYSTIMER_COMP2_LOAD_REG (0x0058)



SYSTIMER_TIMER_COMP2_LOAD 比较器 COMP2 的同步使能信号。置位此位，将重新装载报警值或报警周期至 COMP2。(WT)

Register 10.26. SYSTIMER_INT_ENA_REG (0x0064)

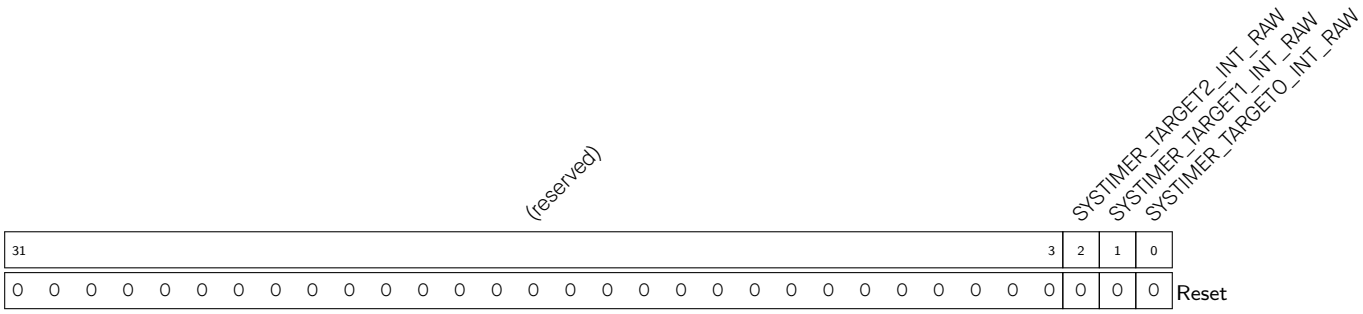


SYSTIMER_TARGET0_INT_ENA SYSTIMER_TARGET0_INT 中断使能位。(R/W)

SYSTIMER_TARGET1_INT_ENA SYSTIMER_TARGET1_INT 中断使能位。(R/W)

SYSTIMER_TARGET2_INT_ENA SYSTIMER_TARGET2_INT 中断使能位。(R/W)

Register 10.27. SYSTIMER_INT_RAW_REG (0x0068)

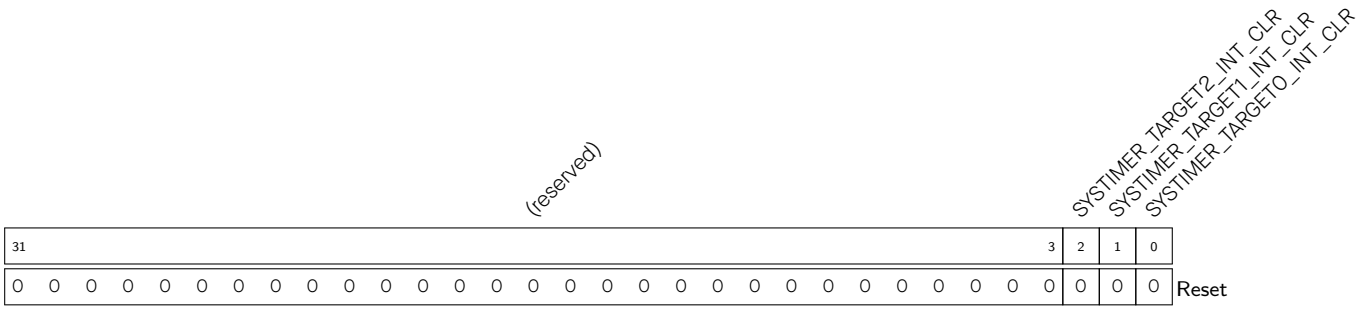


SYSTIMER_TARGET0_INT_RAW SYSTIMER_TARGET0_INT 中断原始位。(R/WTC/SS)

SYSTIMER_TARGET1_INT_RAW SYSTIMER_TARGET1_INT 中断原始位。(R/WTC/SS)

SYSTIMER_TARGET2_INT_RAW SYSTIMER_TARGET2_INT 中断原始位。(R/WTC/SS)

Register 10.28. SYSTIMER_INT_CLR_REG (0x006C)

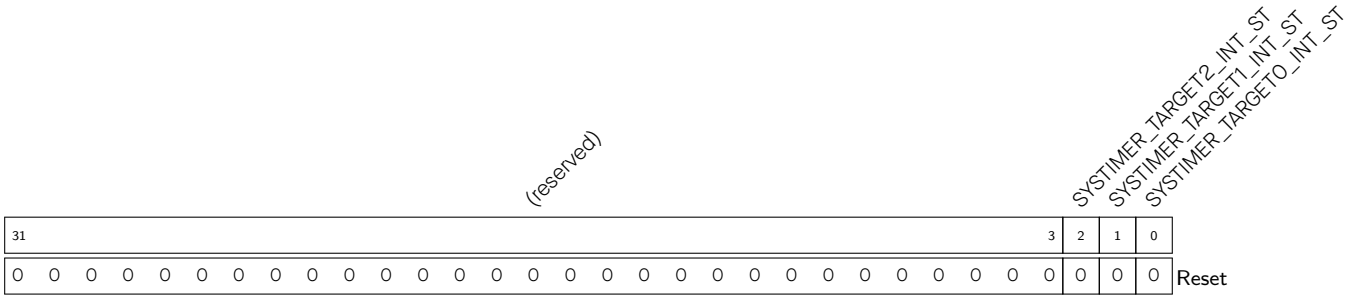


SYSTIMER_TARGET0_INT_CLR SYSTIMER_TARGET0_INT 中断清除位。(WT)

SYSTIMER_TARGET1_INT_CLR SYSTIMER_TARGET1_INT 中断清除位。(WT)

SYSTIMER_TARGET2_INT_CLR SYSTIMER_TARGET2_INT 中断清除位。(WT)

Register 10.29. SYSTIMER_INT_ST_REG (0x0070)

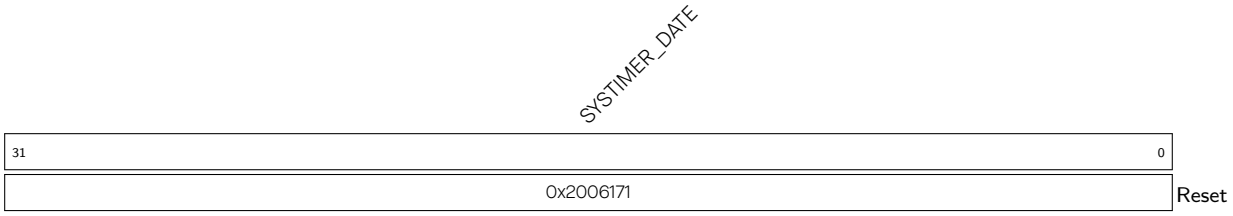


SYSTIMER_TARGET0_INT_ST SYSTIMER_TARGET0_INT 中断状态位。(RO)

SYSTIMER_TARGET1_INT_ST SYSTIMER_TARGET1_INT 中断状态位。(RO)

SYSTIMER_TARGET2_INT_ST SYSTIMER_TARGET2_INT 中断状态位。(RO)

Register 10.30. SYSTIMER_DATE_REG (0x00FC)



SYSTIMER_DATE 版本控制寄存器。(R/W)

11 定时器组 (TIMG)

11.1 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 11-1 所示，ESP8684 包含一个定时器组，即定时器组 0。该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。

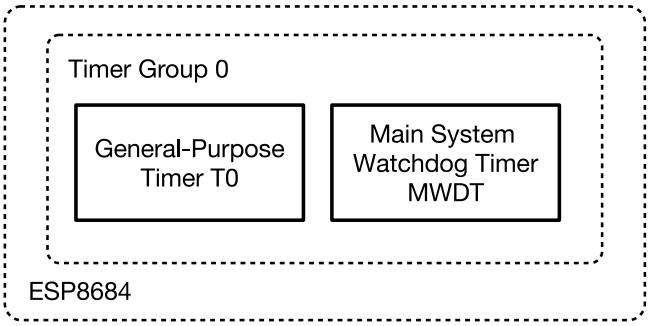


图 11-1. 定时器组概览

本章包含主系统看门狗定时器的寄存器描述，其功能描述请参阅章节 12 看门狗定时器 (WDT)。本章中“定时器”指代通用定时器。

11.2 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
- 两个时钟源：40 MHz PLL_F40M_CLK 时钟或 XTAL_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

11.3 功能描述

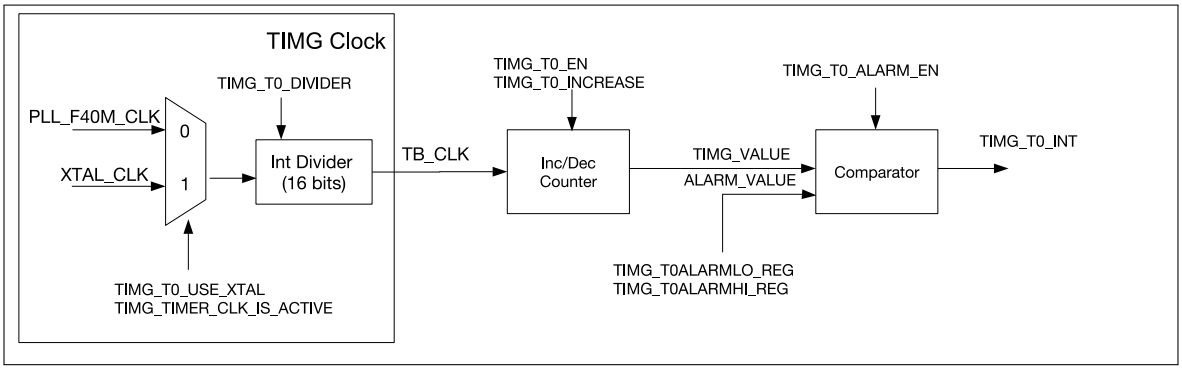


图 11-2. 定时器组架构

图 11-2 为定时器组的 T0。T0 包含一个时钟选择器、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。

11.3.1 16 位预分频器与时钟选择器

每个定时器可通过配置寄存器 TIMG_TOCONFIG_REG 的 TIMG_T0_USE_XTAL 字段，选择 PLL_F40M_CLK 时钟或外部时钟 (XTAL_CLK) 作为时钟源。注意，当芯片处于低功耗模式 CPU_CLK 且时钟源不为 PLL_CLK 时（即 SYSTEM_SOC_CLK_SEL 不为 1 时，详见章节 6 复位和时钟 的表 6-2），定时器仅能选择 XTAL_CLK。

要开启所选时钟，需置位 TIMG_REGCLK_REG 寄存器的 TIMG_TIMER_CLK_IS_ACTIVE 字段，要关闭时钟需清零该字段。所选时钟经 16 位预分频器分频，产生时基计数器使用的时基计数器时钟 (TB_CLK)。16 位预分频器的分频系数可通过 TIMG_T0_DIVIDER 字段配置，选取从 2 到 65536 之间的任意值。注意，将 TIMG_T0_DIVIDER 置 0 后，分频系数会变为 65536。TIMG_T0_DIVIDER 置 1 时，实际分频系数为 2，计数器的值为实际时间的一半。

要更改 16 位预分频器，需先重新配置TIMG_T0_DIVIDER 字段，再将TIMG_T0_DIVIDER_RST 置 1，同时需要关闭定时器（即 TIMG_T0_EN 必须清零），否则会造成不可预知的结果。

11.3.2 54 位时基计数器

54 位时基计数器基于 TB_CLK，可通过 TIMG_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_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_T0_EN 是否置位，TIMG_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_T0UPDATE_REG 写入任意值时，54 位定时器的值开始被锁入寄存器 TIMG_T0LO_REG 和 TIMG_T0HI_REG，两个寄存器分别锁存低 32 位和高 22 位。当 TIMG_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_T0UPDATE_REG 被写入新值之前，保持寄存器 TIMG_T0LO_REG 和 TIMG_T0HI_REG 的值不变，以供 32 位的 CPU 读值。

11.3.3 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，（可选择）让定时器的当前值自动重新加载（详见第 11.3.4节）。

54 位报警值可在 TIMG_TOALARMLO_REG 和 TIMG_TOALARMHI_REG 配置，两者分别代表报警值的低 32 位和高 22 位。但是，只有置位 TIMG_TO_ALARM_EN 字段使能报警功能后，配置的报警值才会生效。为解决报警使能“过晚”（即报警使能时，定时器的值已过报警值），出现以下情况时硬件会立即触发报警：

- 可逆计数器向上计数时，定时器的当前值高于报警值（在一定范围内）
- 可逆计数器向下计数时，定时器的当前值低于报警值（在一定范围内）

表 11-1 和表 11-2 说明了定时器当前值、报警值与报警触发的关系。假设定时器当前值和报警值如下：

- TIMG_VALUE = {TIMG_TOHI_REG, TIMG_TOLO_REG}
- ALARM_VALUE = {TIMG_TOALARMHI_REG, TIMG_TOALARMLO_REG}

表 11-1. 可逆计数器向上计数时的报警触发场景

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 11-2. 可逆计数器向下计数时的报警触发场景

场景	范围	报警
5	TIMG_VALUE − ALARM_VALUE $> 2^{53}$	触发
6	$0 <$ TIMG_VALUE − ALARM_VALUE $\leq 2^{53}$	可逆计数器向下计数，TIMG_VALUE 达到 ALARM_VALUE 时报警
7	$0 \leq$ ALARM_VALUE − TIMG_VALUE $< 2^{53}$	触发
8	ALARM_VALUE − TIMG_VALUE $\geq 2^{53}$	可逆计数器向下计数达到最小值时，重新开始从最大值向下计数，TIMG_VALUE 达到 ALARM_VALUE 时触发报警

报警时，TIMG_TO_ALARM_EN 字段自动清零，在下次置位 TIMG_TO_ALARM_EN 前不会再次报警。

11.3.4 定时器重新加载

定时器重新加载指将定时器的低 32 位和高 22 位分别更新为 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 字段存储的重新加载值。但是，把重新加载值写入 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 字段不会改变定时器的当前值。写入的重新加载值会被定时器忽视，直到重新加载事件被触发。重新加载事件可由软件即时重新加载或报警时自动重新加载触发。

CPU 在寄存器 TIMG_TOLOAD_REG 写任意值会触发软件即时重新加载，定时器的当前值会立即改变。若此时 TIMG_TO_EN 是置位状态，定时器会继续从新数值开始递增或递减计数。若此时 TIMG_TO_ALARM_EN置位，仍会根据表 11-1 或表 11-2 中的关系在对应时刻产生报警。若 TIMG_TO_EN 是清零状态，定时器将保持当前值，直至计数重新使能。

报警时，自动重新加载功能可让定时器在报警时重新加载，从重新加载值开始继续递增或递减计数。该功能通常用于周期性报警时重置定时器的值。要使能报警时自动重新加载，需将 TIMG_TO_AUTORELOAD 字段置 1。如未使能该功能，报警后定时器的值会在过报警值后继续递增或递减。

11.3.5 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。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 11.4.4）；
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：$f\_rtc = \frac{C0 \times f\_XTAL\_CLK}{C1}$

11.3.6 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_TO_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_TO_INT_RAW：报警时置 1。该位在写值到对应的 TIMG_TO_INT_CLR 位后才会被清零。
- TIMG_WDT_INT_RAW：阶段超时时置 1。该位在写值到对应的 TIMG_WDT_INT_CLR 位后才会被清零。
- TIMG_TO_INT_ST：反映每个定时器中断的状态，通过用 TIMG_TO_INT_ENA 屏蔽 TIMG_TO_INT_RAW 位来生成。
- TIMG_WDT_INT_ST：反映每个看门狗定时器中断的状态，通过用 TIMG_WDT_INT_ENA 屏蔽 TIMG_WDT_INT_RAW 位来生成。
- TIMG_TO_INT_ENA：用于使能或屏蔽组内定时器的中断状态位。
- TIMG_WDT_INT_ENA：用于使能或屏蔽组内看门狗定时器的中断状态位。
- TIMG_TO_INT_CLR：置 1 此位清除定时器中断，定时器在 TIMG_TO_INT_RAW 和 TIMG_TO_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_WDT_INT_CLR：置 1 此位清除定时器中断，看门狗定时器在 TIMG_WDT_INT_RAW 和 TIMG_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。

11.4 配置与使用

11.4.1 定时器用作简单时钟

1. 配置时基计数器。
   - 置位或清除 TIMG_TO_USE_XTAL 字段选择时钟源。CPU 处于高效工作模式时，这个字段可以任意配置。CPU 处于低功耗模式时（即 SYSTEM_SOC_CLK_SEL 不为 1）时，该字段必须置 1。
   - 置位 TIMG_TO_DIVIDER 配置 16 位预分频器。

- 置位或清除 TIMG_TO_INCREASE 配置定时器方向。
- 在 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 上写初始值设置定时器的初始值，然后在 TIMG_TOLOAD_REG 上写任意值将初始值重新加载进定时器。

2. 置位 TIMG_TO_EN 开启定时器。
3. 获得定时器的当前值。
   - 在 TIMG_TOUPDATE_REG 上写任意值锁存定时器的当前值。
   - 等待硬件将 TIMG_TOUPDATE_REG 清 0。
   - 从 TIMG_TOLO_REG 和 TIMG_TOHI_REG 读取锁存的定时器值。

11.4.2 定时器用于单次报警

1. 按照第 11.4.1 节的第 1 步配置时基计数器。
2. 配置报警。
   - 置位 TIMG_TOALARMLO_REG 和 TIMG_TOALARMHI_REG 配置报警值。
   - 置位 TIMG_TO_INT_ENA 使能中断。
3. 清零 TIMG_TO_AUTORELOAD 关闭自动重新加载。
4. 置位 TIMG_TO_ALARM_EN 开启报警。
5. 处理报警中断。
   - 置位定时器在 TIMG_TO_INT_CLR 的对应位清除中断。
   - 清零 TIMG_TO_EN 关闭定时器。

11.4.3 定时器用于周期性报警

1. 按照第 11.4.1 节的第 1 步配置时基计数器。
2. 按照第 11.4.2节的第 2 步配置报警。
3. 置位 TIMG_TO_AUTORELOAD 使能自动重新加载，将重新加载值写入 TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI。
4. 置位 TIMG_TO_ALARM_EN 开启报警。
5. 处理报警中断（每次报警时重复）。
   - 置位定时器在 TIMG_TO_INT_CLR 的对应位清除中断。
   - 如下一次报警需要新的报警值和重新加载值（即每次都有不同的报警间隔），则应根据需要重新配置 TIMG_TOALARMLO_REG、TIMG_TOALARMHI_REG、TIMG_TO_LOAD_LO 和 TIMG_TO_LOAD_HI 。否则，上述寄存器应保持不变。
   - 置位 TIMG_TO_ALARM_EN 重新使能报警。
6. （最后一次报警时）关闭定时器。
   - 置位定时器在 TIMG_TO_INT_CLR 的对应位清除中断。
   - 清零 TIMG_TO_EN 关闭定时器。

11.4.4 RTC_SLOW_CLK 频率计算

1. 单次计算
   - 设置 TIMG_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_RTC_CALI_MAX 配置频率计算时间。
   - 清空 TIMG_RTC_CALI_START_CYCLING 选择单次校准模式，然后配置 TIMG_RTC_CALI_START 开启两个计数器。
   - 等待 TIMG_RTC_CALI_RDY 的值变为 1，读取TIMG_RTC_CALI_VALUE 获取 XTAL_CLK 计数器值，根据章节 11.3.5 的公式计算 RTC_SLOW_CLK 频率。
2. 周期性计算
   - 设置 TIMG_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_RTC_CALI_MAX 配置频率计算时间。
   - 使能 TIMG_RTC_CALI_START_CYCLING，硬件将不间断进行频率计算过程。
   - 只要 TIMG_RTC_CALI_CYCLING_DATA_VLD 为 1，即表示TIMG_RTC_CALI_VALUE 有效。
3. 超时
   如果 RTC_SLOW_CLK 的计数器没有在TIMG_RTC_CALI_TIMEOUT_RST_CNT 的 XTAL_CLK 计数器内完成计数，将置位 TIMG_RTC_CALI_TIMEOUT 标记计算超时。

11.5 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 3 系统和存储器 中的表 3-3。

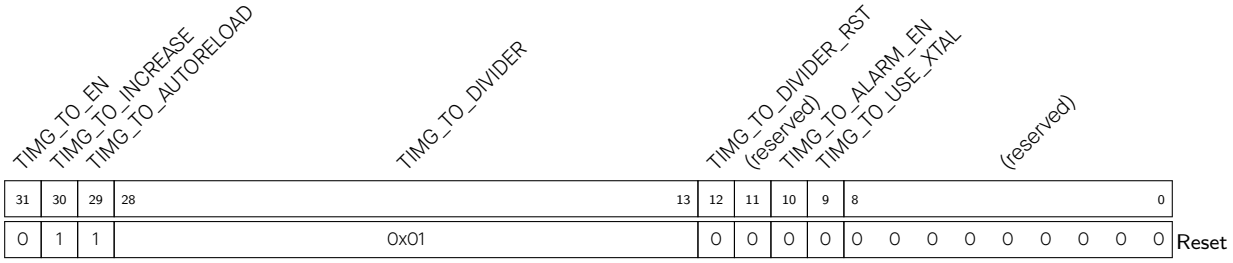
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
定时器 0 控制和配置寄存器			
TIMG_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 TIMG_T0LO_REG 或 TIMG_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

11.6 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 3 系统和存储器 中的表 3-3。

Register 11.1. TIMG_T0CONFIG_REG (0x0000)



TIMG_T0_USE_XTAL 0：使用 PLL_F40M_CLK 作为定时器组的源时钟；1：使用 XTAL_CLK 作为定时器组的源时钟。(R/W)

TIMG_T0_ALARM_EN 置 1 后，报警使能。报警时，此位自动清零。(R/W/SC)

TIMG_T0_DIVIDER_RST 置 1 后，复位定时器 0 时钟分频器的计数器。(WT)

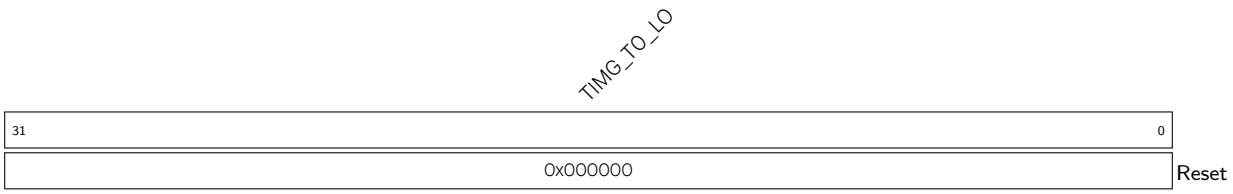
TIMG_T0_DIVIDER 定时器 0 时钟 (T0_clk) 的预分频器值。(R/W)

TIMG_T0_AUTORELOAD 置 1 后，定时器 0 报警时自动重新加载使能。(R/W)

TIMG_T0_INCREASE 置 1 后，定时器 0 的时基计数器会在每个时钟周期后递增。清零后，定时器 0 的时基计数器会递减。(R/W)

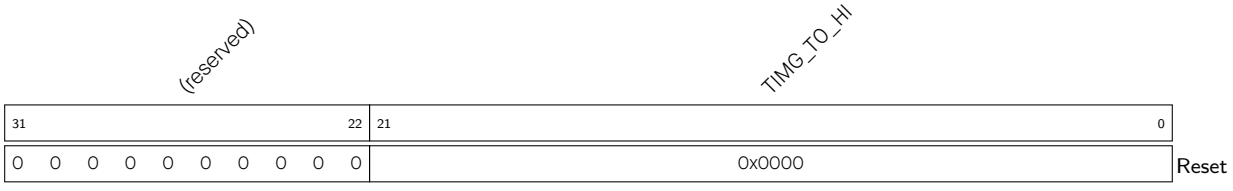
TIMG_T0_EN 置 1 后，定时器 0 时基计数器使能。(R/W)

Register 11.2. TIMG_T0LO_REG (0x0004)



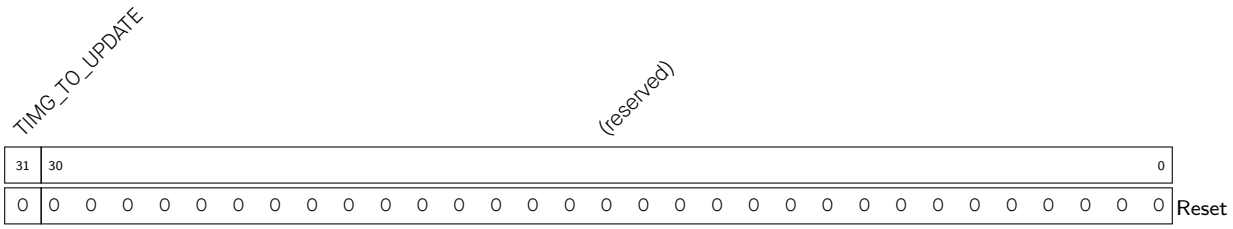
TIMG_T0_LO 在 TIMG_T0UPDATE_REG 上写值后，可读取定时器 0 时基计数器的低 32 位。(RO)

Register 11.3. TIMG_T0HI_REG (0x0008)



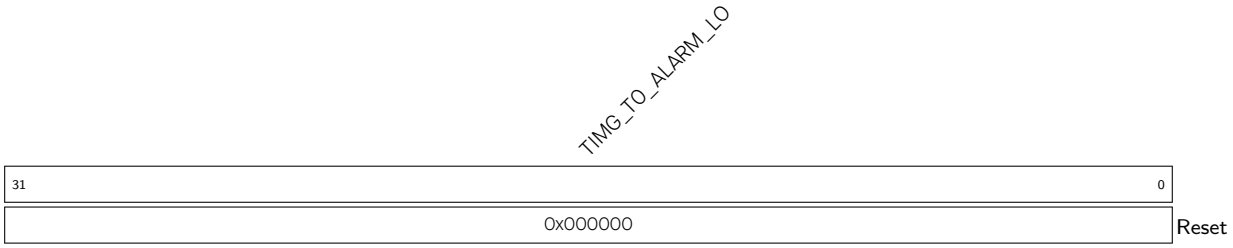
TIMG_T0_HI 在 TIMG_T0UPDATE_REG 上写值后，可读取定时器 0 时基计数器的高 22 位。(RO)

Register 11.4. TIMG_T0UPDATE_REG (0x000C)



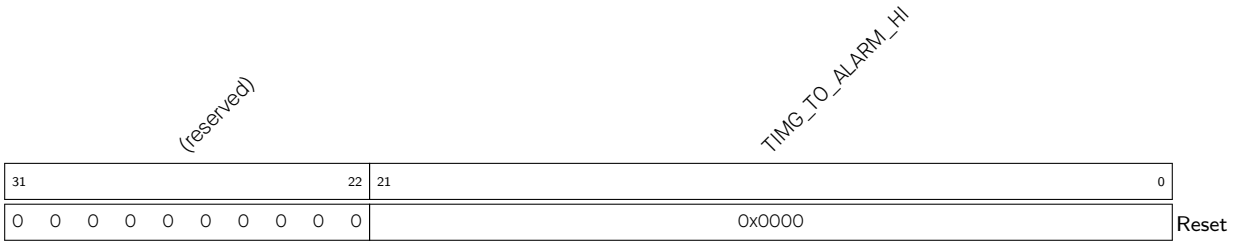
TIMG_T0_UPDATE 在 TIMG_T0UPDATE_REG 上写 0 或 1，计数器的值被锁住。(R/W/SC)

Register 11.5. TIMG_T0ALARMLO_REG (0x0010)



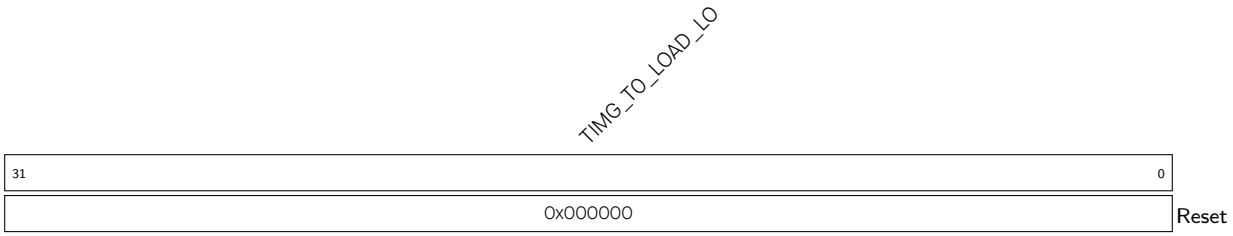
TIMG_T0_ALARM_LO 定时器 0 时基计数器触发警报值的低 32 位。(R/W)

Register 11.6. TIMG_T0ALARMHI_REG (0x0014)



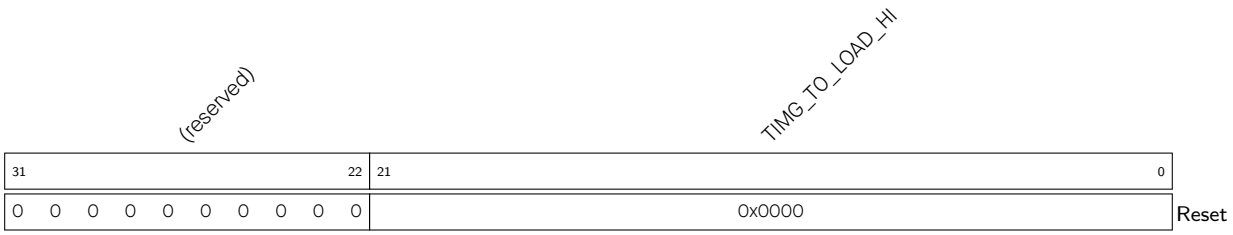
TIMG_T0_ALARM_HI 定时器 0 时基计数器触发警报值的高 22 位。(R/W)

Register 11.7. TIMG_T0LOADLO_REG (0x0018)



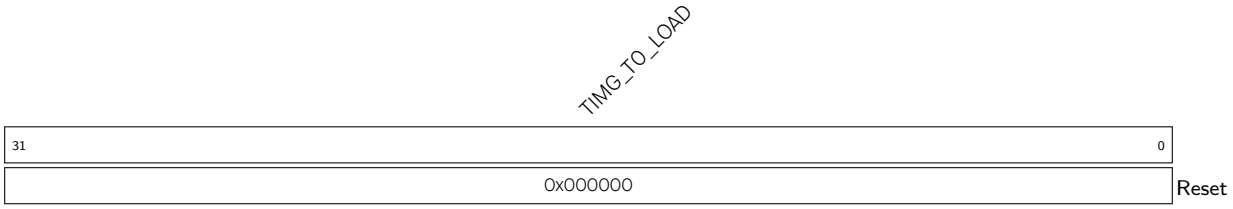
TIMG_T0_LOAD_LO 定时器 0 时基计数器重新加载的低 32 位值。(R/W)

Register 11.8. TIMG_TOLOADHI_REG (0x001C)



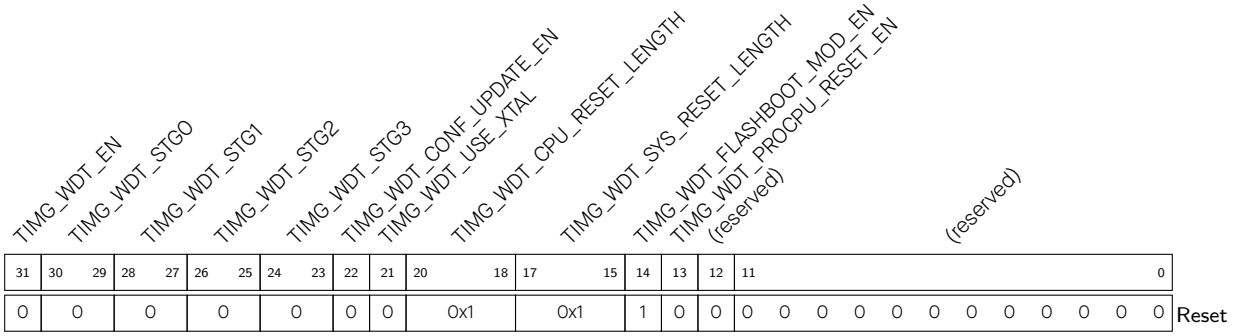
TIMG_TO_LOAD_HI 定时器 0 时基计数器重新加载的高 22 位值。(R/W)

Register 11.9. TIMG_TOLOAD_REG (0x0020)



TIMG_TO_LOAD 写任意值触发定时器 0 时基计数器重新加载。(WT)

Register 11.10. TIMG_WDTCONFIG0_REG (0x0048)



TIMG_WDT_PROCPU_RESET_EN WDT 复位 CPU 使能。(R/W)

TIMG_WDT_FLASHBOOT_MOD_EN 置 1 后, flash 启动保护使能。(R/W)

TIMG_WDT_SYS_RESET_LENGTH 系统复位信号长度选择。0：100 ns；1：200 ns；2：300 ns；3：400 ns；4：500 ns；5：800 ns；6：1.6 μs；7：3.2 μs。(R/W)

TIMG_WDT_CPU_RESET_LENGTH CPU 复位信号长度选择。0：100 ns；1：200 ns；2：300 ns；3：400 ns；4：500 ns；5：800 ns；6：1.6 μs；7：3.2 μs。(R/W)

TIMG_WDT_USE_XTAL 选择看门狗定时器的时钟。0：PLL_F40M_CLK；1：XTAL_CLK。(R/W)

TIMG_WDT_CONF_UPDATE_EN 更新看门狗定时器配置寄存器。(WT)

TIMG_WDT_STG3 阶段 3 配置。0：关闭；1：中断；2：复位 CPU；3：复位系统。(R/W)

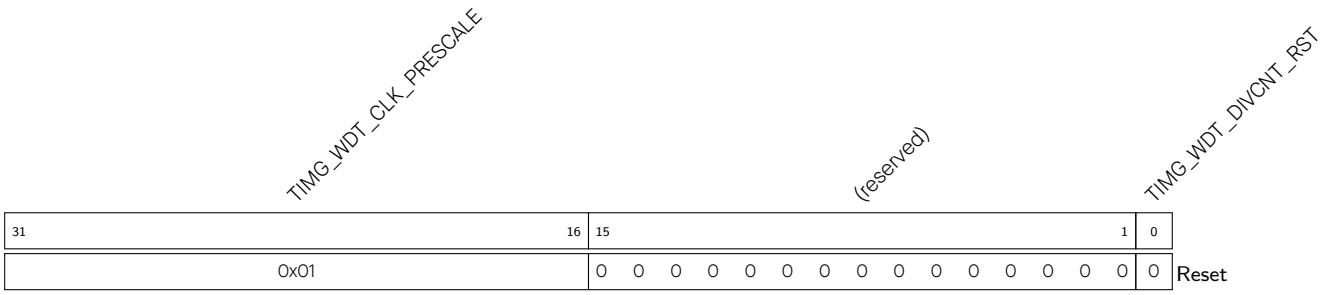
TIMG_WDT_STG2 阶段 2 配置。0：关闭；1：中断；2：复位 CPU；3：复位系统。(R/W)

TIMG_WDT_STG1 阶段 1 配置。0：关闭；1：中断；2：复位 CPU；3：复位系统。(R/W)

TIMG_WDT_STG0 阶段 0 配置。0：关闭；1：中断；2：复位 CPU；3：复位系统。(R/W)

TIMG_WDT_EN 置 1 后，MWDT 使能。(R/W)

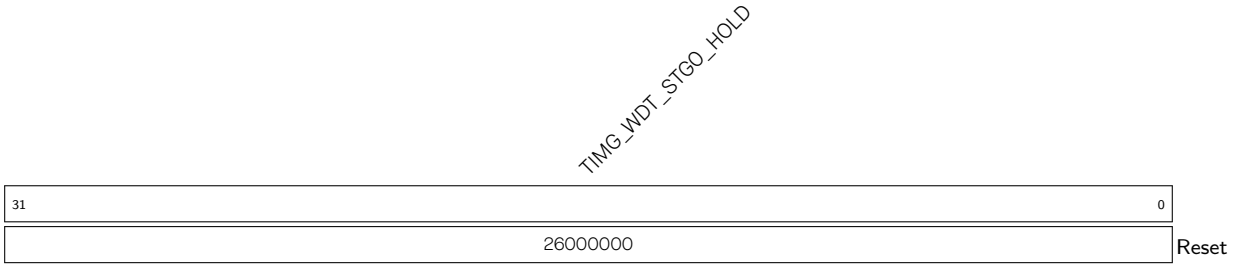
Register 11.11. TIMG_WDTCONFIG1_REG (0x004C)



TIMG_WDT_DIVCNT_RST 置 1 后，复位看门狗定时器时钟分频器的计数器。(WT)

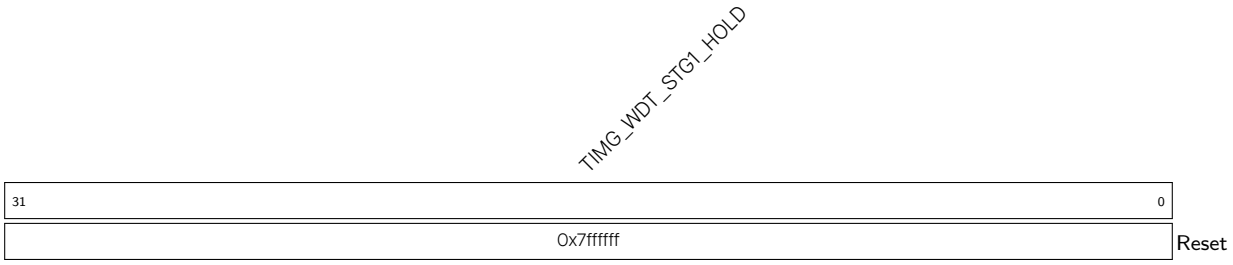
TIMG_WDT_CLK_PRESCALE MWDT 时钟预分频器值。MWDT 时钟周期 = MWDT 时钟源周期 * TIMG_WDT_CLK_PRESCALE。(R/W)

Register 11.12. TIMG_WDTCONFIG2_REG (0x0050)



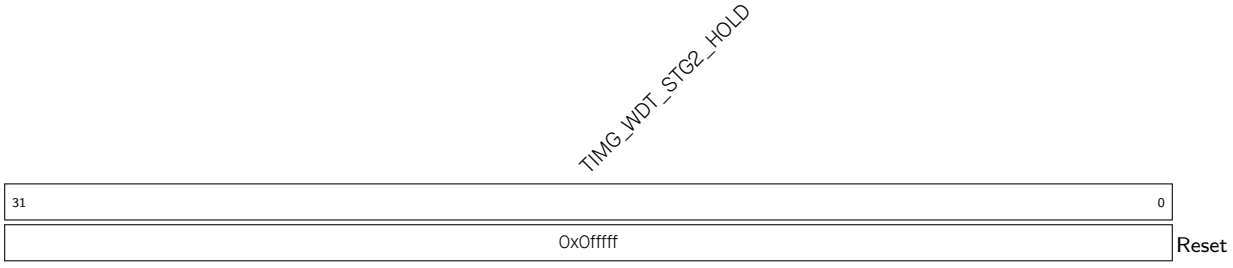
TIMG_WDT_STG0_HOLD 阶段 0 超时时间，单位是 MWDT 时钟周期。(R/W)

Register 11.13. TIMG_WDTCONFIG3_REG (0x0054)



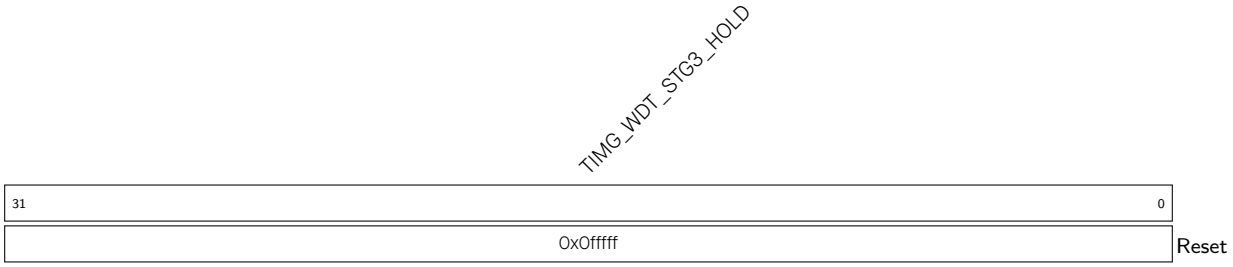
TIMG_WDT_STG1_HOLD 阶段 1 超时时间，单位是 MWDT 时钟周期。(R/W)

Register 11.14. TIMG_WDTCONFIG4_REG (0x0058)



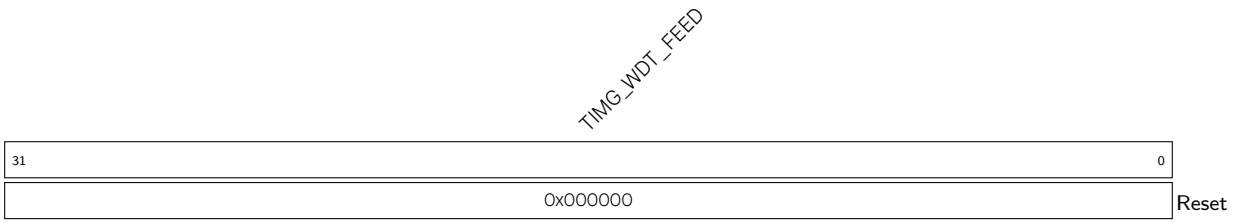
TIMG_WDT_STG2_HOLD 阶段 2 超时时间，单位是 MWDT 时钟周期。(R/W)

Register 11.15. TIMG_WDTCONFIG5_REG (0x005C)



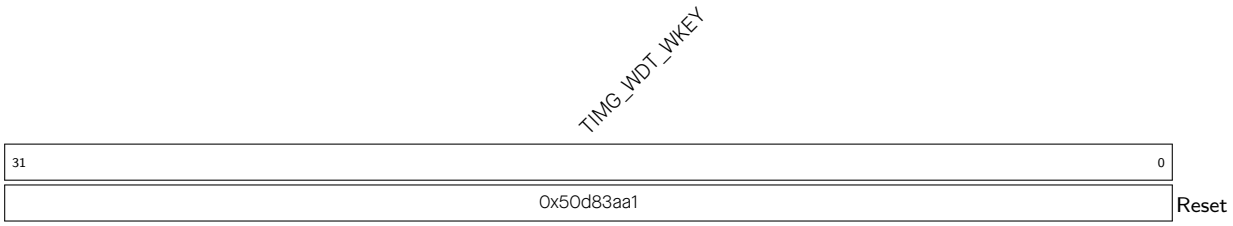
TIMG_WDT_STG3_HOLD 阶段 3 超时时间，单位是 MWDT 时钟周期。(R/W)

Register 11.16. TIMG_WDTFEED_REG (0x0060)



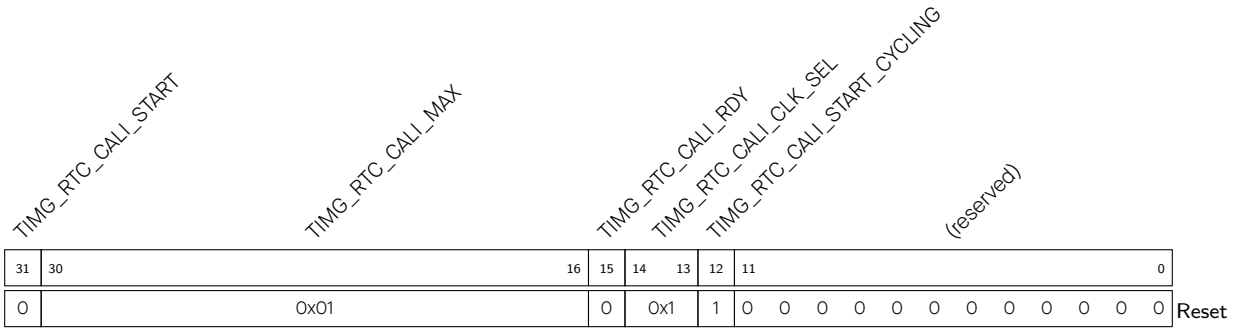
TIMG_WDT_FEED 写任意值喂 MWDT。(WT)

Register 11.17. TIMG_WDTWPROTECT_REG (0x0064)



TIMG_WDT_WKEY 如果寄存器的值与复位值不同，写保护使能。(R/W)

Register 11.18. TIMG_RTCCALICFG_REG (0x0068)



TIMG_RTC_CALI_START_CYCLING 0：单次频率计算模式；1：周期性频率计算模式。(R/W)

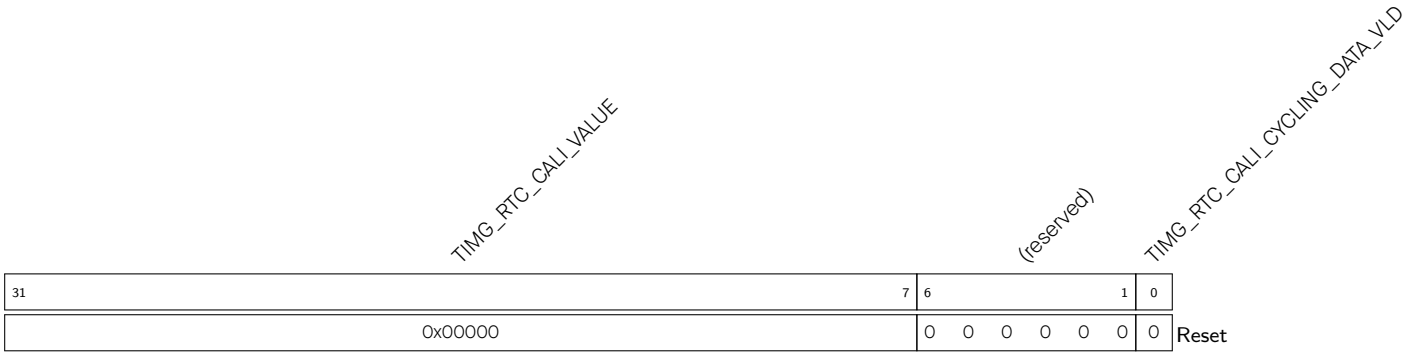
TIMG_RTC_CALI_CLK_SEL 0：RC_SLOW_CLK；1：RC_FAST_DIV_CLK；2：XTAL32K_CLK。(R/W)

TIMG_RTC_CALI_RDY 标记单次频率计算完成。(RO)

TIMG_RTC_CALI_MAX 配置计算 RTC 慢速时钟 RTC_SLOW_CLK 频率的时间。单位：RTC_SLOW_CLK 时钟周期。(R/W)

TIMG_RTC_CALI_START 置位此位，开始单次频率计算。(R/W)

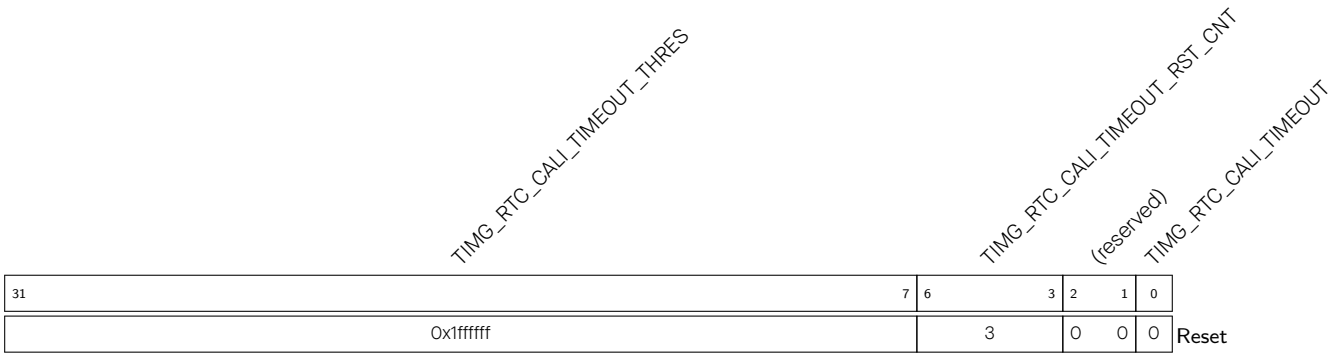
Register 11.19. TIMG_RTCCALICFG1_REG (0x006C)



TIMG_RTC_CALI_CYCLING_DATA_VLD 标记周期性频率计算完成。(RO)

TIMG_RTC_CALI_VALUE 单次或周期性频率计算完成时，读取此位计算 RTC 慢速时钟 RTC_SLOW_CLK 的频率。单位：XTAL_CLK 时钟周期。(RO)

Register 11.20. TIMG_RTCCALICFG2_REG (0x0080)

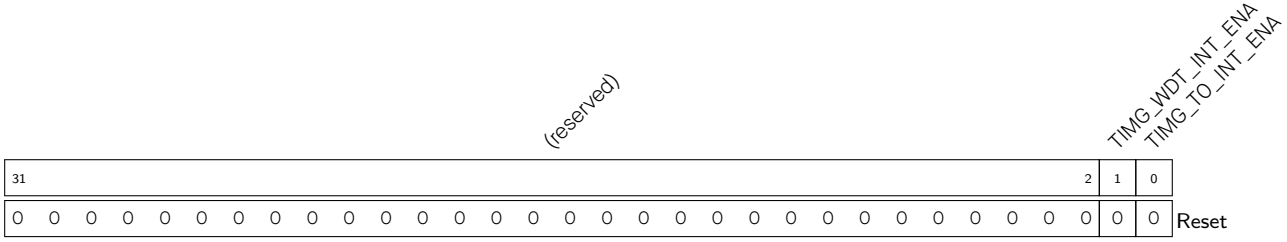


TIMG_RTC_CALI_TIMEOUT 表示频率计算超时。(RO)

TIMG_RTC_CALI_TIMEOUT_RST_CNT 频率计算超时复位的周期。(R/W)

TIMG_RTC_CALI_TIMEOUT_THRES 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。(R/W)

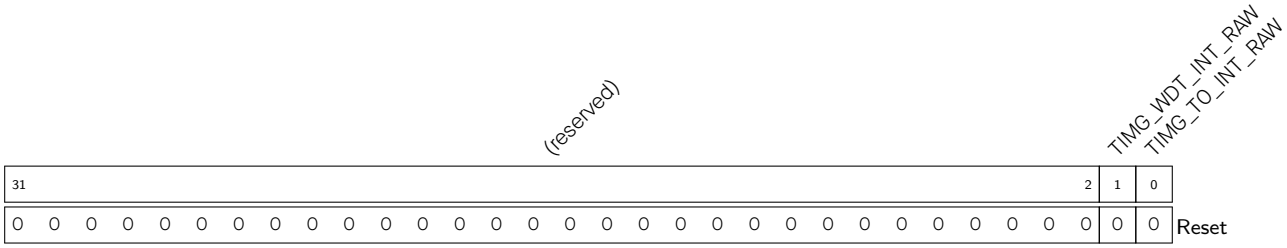
Register 11.21. TIMG_INT_ENA_TIMERS_REG (0x0070)



TIMG_TO_INT_ENA TIMG_TO_INT 中断的使能位。(R/W)

TIMG_WDT_INT_ENA TIMG_WDT_INT 中断的使能位。(R/W)

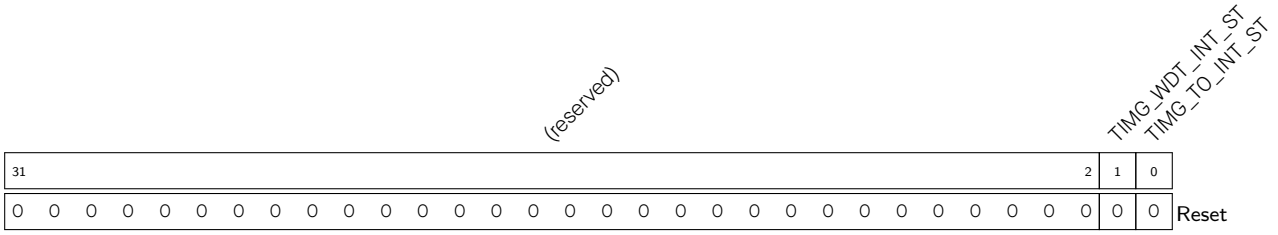
Register 11.22. TIMG_INT_RAW_TIMERS_REG (0x0074)



TIMG_TO_INT_RAW TIMG_TO_INT 中断的原始中断状态位。(R/SS/WTC)

TIMG_WDT_INT_RAW TIMG_WDT_INT 中断的原始中断状态位。(R/SS/WTC)

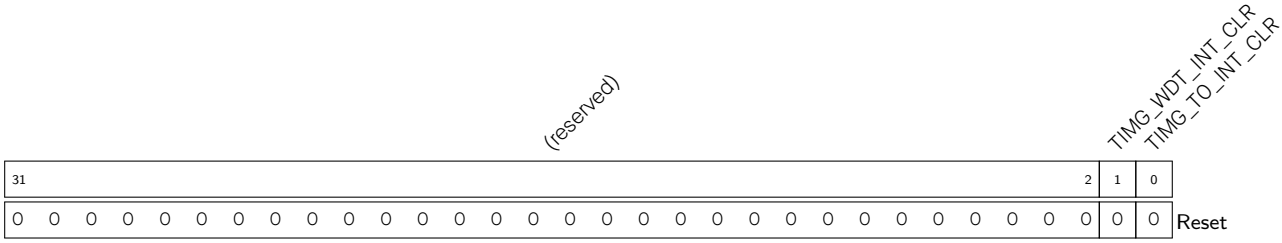
Register 11.23. TIMG_INT_ST_TIMERS_REG (0x0078)



TIMG_T0_INT_ST TIMG_T0_INT 中断的屏蔽中断状态位。(RO)

TIMG_WDT_INT_ST TIMG_WDT_INT 中断的屏蔽中断状态位。(RO)

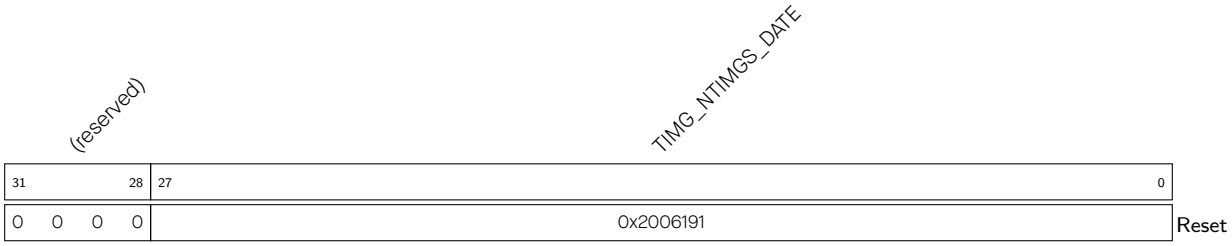
Register 11.24. TIMG_INT_CLR_TIMERS_REG (0x007C)



TIMG_T0_INT_CLR 置位此位，清除 TIMG_T0_INT 中断。(WT)

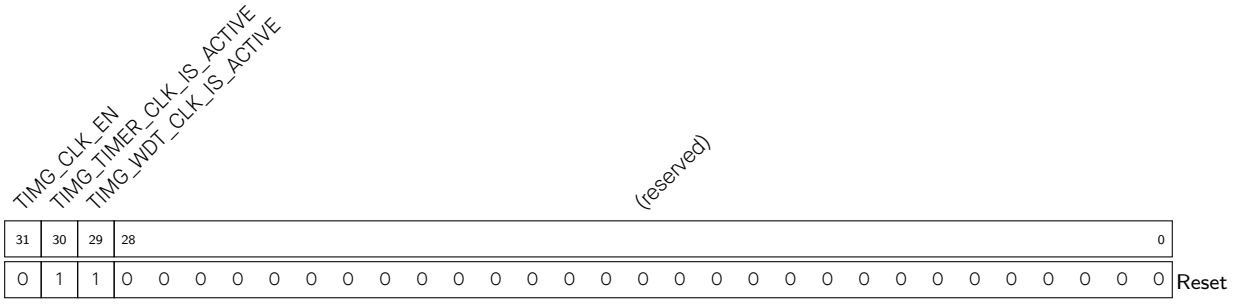
TIMG_WDT_INT_CLR 置位此位，清除 TIMG_WDT_INT 中断。(WT)

Register 11.25. TIMG_NTIMERS_DATE_REG (0x00F8)



TIMG_NTIMGS_DATE 版本控制寄存器。(R/W)

Register 11.26. TIMG_REGCLK_REG (0x00FC)



TIMG_WDT_CLK_IS_ACTIVE 使能看门狗定时器的时钟。(R/W)

TIMG_TIMER_CLK_IS_ACTIVE 使能定时器 0 的时钟。(R/W)

TIMG_CLK_EN 寄存器时钟门控信号。1：软件可以读写寄存器；0：软件不能读写寄存器。(R/W)

12 看门狗定时器 (WDT)

12.1 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 12-1 所示，ESP8684 中有两个数字看门狗定时器：章节 11 定时器组 (TIMG) 描述的定时器组中有一个（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 12.2.2.2 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。

ESP8684 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。

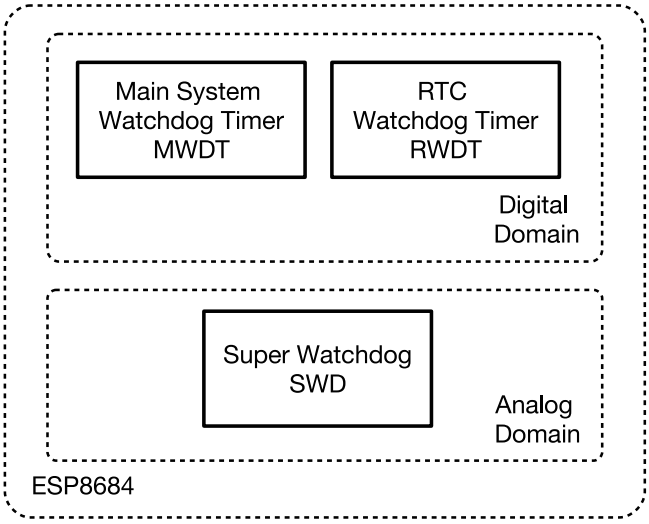


图 12-1. 看门狗定时器概览

请注意，本章节仅包含看门狗定时器的功能描述，其寄存器部分详见章节 11 定时器组 (TIMG) 和章节 9 低功耗管理 (RTC_CNTL)。

12.2 数字看门狗定时器

12.2.1 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：
  - MWDT：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：40 MHz PLL_F40M_CLK 或 XTAL_CLK
  - RWDT：RTC_SLOW_CLK

12.2.2 功能描述

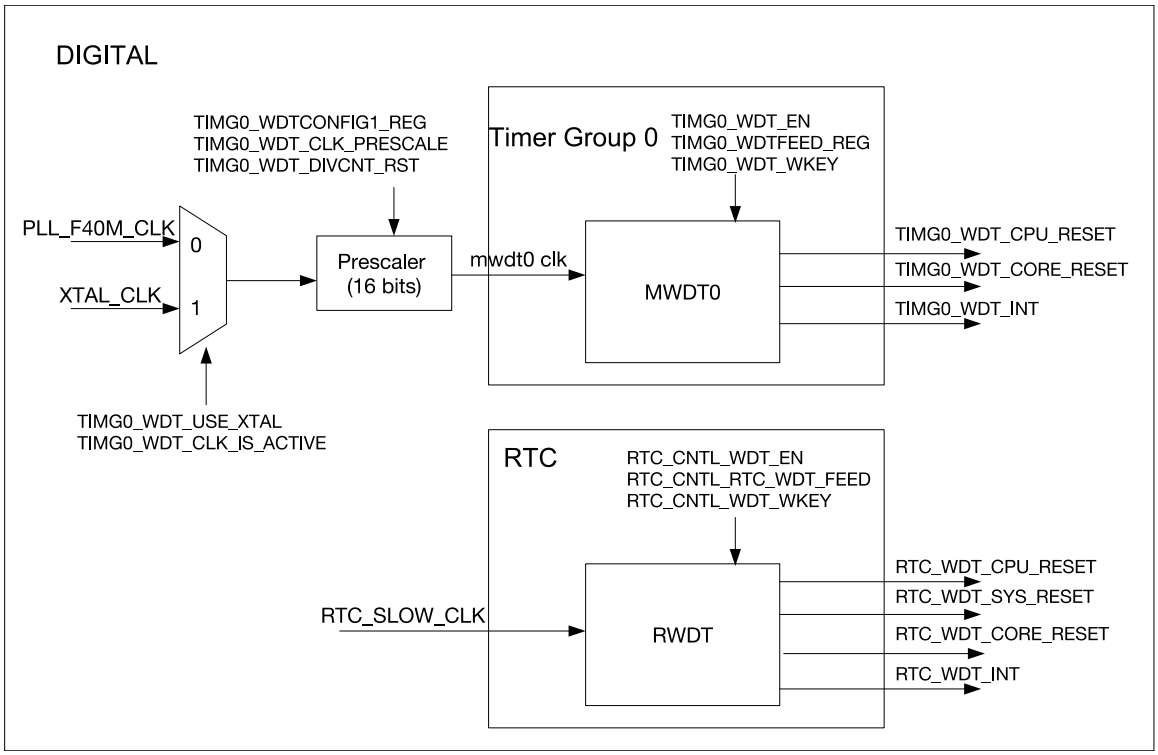


图 12-2. ESP8684 的数字看门狗定时器

图 12-2 为 ESP8684 数字系统中的两个看门狗定时器。

12.2.2.1 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。

MWDT 可通过设置 TIMG_WDTCONFIG0_REG 寄存器的 TIMG_WDT_USE_XTAL 字段选择 PLL_F40M_CLK 时钟或外部时钟 (XTAL_CLK) 作为时钟源。注意，当芯片处于低功耗模式 CPU_CLK 且时钟源不为 PLL_CLK 时（即 SYSTEM_SOC_CLK_SEL 不为 1 时，详见章节 6 复位和时钟 的表 6-2），MWDT 仅能选择 XTAL_CLK。将 TIMG_REGCLK_REG 寄存器的 TIMG_WDT_CLK_IS_ACTIVE 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。MWDT 的 16 位预分频器可通过 TIMG_WDTCONFIG1_REG 寄存器的

TIMG_WDT_CLK_PRESCALE 字段配置。TIMG_WDT_DIVCNT_RST 字段置位时，预分频器复位，可立即重新配置。

RWDT 直接将 RTC 慢速时钟 RTC_SLOW_CLK（详见章节 6 复位和时钟）用作时钟源。

MWDT 和 RWDT 看门狗可分别通过设置 TIMG_WDT_EN 和 RTC_CNTL_WDT_EN 字段使能。看门狗使能后，其内部 32 位计数器的值会在每个时钟源周期内累加 1，直到达到该阶段的超时时间（即在该阶段发生超时）。如发生超时，计数器的值会重置为 0，同时看门狗进入下一阶段。如果软件成功喂狗，看门狗定时器会回到阶段 0，并将计数器的值重置为 0。软件向 TIMG_WDTFEED_REG 和 RTC_CNTL_RTC_WDT_FEED 寄存器内写入任意值，便可分别为 MDWT 和 RWDT 喂狗。

12.2.2.2 阶段与超时动作

定时器在各阶段可以配置不同的超时时间和对应的超时动作。某一阶段超时会触发对应的超时动作，同时计数器的值被重置为 0，看门狗进入下一阶段。MWDT 和 RWDT 有四个阶段（称为阶段 0 至阶段 3）。看门狗定时器会循环工作（即从阶段 0 至阶段 3，再回到阶段 0）。

MWDT 每个阶段的超时时间可用 TIMG_WDTCONFIG*i*_REG （i 的范围是 2 到 5）寄存器配置，RWDT 的超时时间可用 RTC_CNTL_WDT_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 ($Thold_0$) 受 eFuse 寄存器 EFUSE_RD_REPEAT_DATA0_REG 的
EFUSE_WDT_DELAY_SEL 字段和 RTC_CNTL_WDT_STG0_HOLD 字段共同影响，关系如下：

$$T_{hold0} = RTC\_CNTL\_WDT\_STG0\_HOLD << (EFUSE\_WDT\_DELAY\_SEL + 1)$$

其中，<< 为左移运算符。

如某个阶段超时，下列超时动作之一将会执行：

表 12-1. 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	复位主系统（包括 MWDT、CPU 和所有外设），功耗管理单元和 RTC 外设不会复位
系统复位	复位主系统、功耗管理单元和 RTC 外设（详见章节 9 低功耗管理 (RTC_CNTL)），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_WDTCONFIG0_REG 寄存器中配置。RWDT 的超时动作可在 RTC_CNTL_WDTCONFIG0_REG 寄存器配置。

12.2.2.3 写保护

看门狗定时器对于检测和处理系统或软件错误而言至关重要，不应轻易关闭（例如，因写寄存器位置错误而误将看门狗关闭）。因此，MWDT 和 RWDT 引入写保护机制，防止看门狗因无意的写操作而被关闭或篡改。

写保护机制通过每个看门狗定时器的写密钥字段运行（MWDT 看门狗使用 TIMG_WDT_WKEY，RWDT 看门狗使用 RTC_CNTL_WDT_WKEY）。必须向看门狗定时器的写密钥字段写入 0x50D83AA1，才能修改其它看门狗寄存器。如果写密钥字段的值不是 0x50D83AA1，任何试图向看门狗定时器寄存器（除了向写密钥字段本身）写值的操作都会被忽略。推荐按以下步骤访问看门狗定时器：

1. 将 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。

12.2.2.4 Flash 引导保护

在 flash 引导模式下，MWDT 和 RWDT 会默认使能。MWDT 的阶段 0 的默认超时动作为内核复位（复位主系统）。RWDT 的阶段 0 超时动作为系统复位（复位主系统和 RTC）。引导后，应将 TIMG_WDT_FLASHBOOT_MOD_EN 和 RTC_CNTL_WDT_FLASHBOOT_MOD_EN 位清零，分别关闭 MWDT 和 RWDT 的 flash 引导保护。然后，软件可以配置 MWDT 和 RWDT。

12.3 模拟看门狗定时器

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。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。

SWD 的时钟源固定，不可选择。

12.3.1 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

12.3.2 SWD 控制器

12.3.2.1 结构

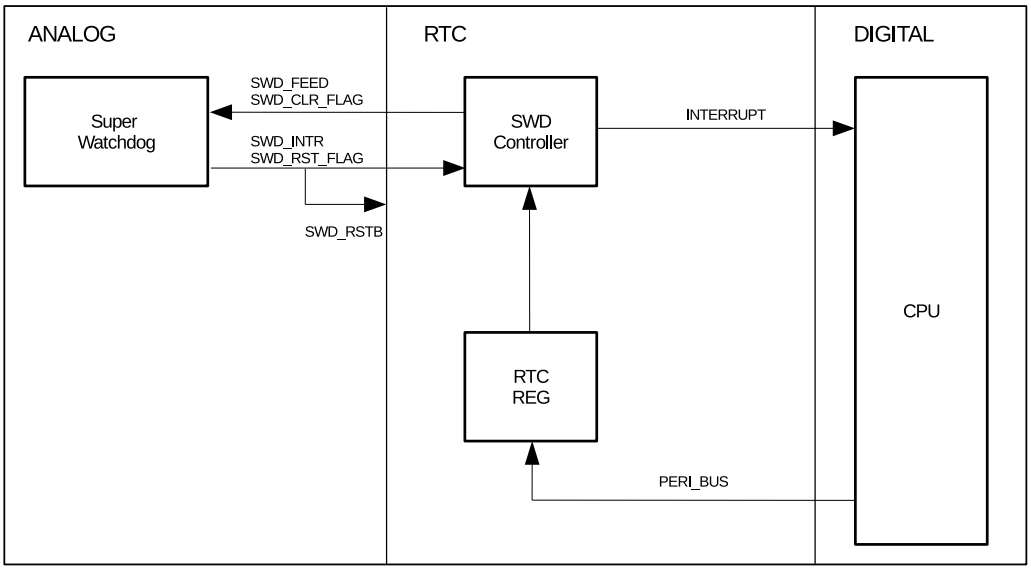


图 12-3. SWD 控制器结构

12.3.2.2 工作流程

正常状态下：

- SWD 控制器收到 SWD 的喂狗请求。
- SWD 控制器可以向主 CPU 发送中断。
- 主 CPU 可以通过置位 RTC_CNTL_SWD_FEED 直接喂狗。
- CPU 喂狗时，需要先向 RTC_CNTL_SWD_WKEY 写 0x8F1D312A 关闭 SWD 控制器的写保护。这样做可以防止系统在数字电路异常状态下运行时误喂 SWD。
- 如将 RTC_CNTL_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 RTC_CNTL_RESET_CAUSE_PROCPU[5:0] 获知 CPU 复位原因。
  如 RTC_CNTL_RESET_CAUSE_PROCPU[5:0] == 0x12，则表示上一次复位的原因是 SWD 复位。
- 置位 RTC_CNTL_SWD_RST_FLAG_CLR 清除 SWD 复位标志。

12.4 中断

看门狗定时器中断，请前往章节 11 定时器组 *(TIMG)* 的第 11.3.6 节 中断 查看。

12.5 寄存器

MWDT 寄存器是定时器组模块的一部分，在章节 11 定时器组 *(TIMG)* 的第 11.5 节 寄存器列表 中有详细描述。
RWDT 和 SWD 寄存器是 RTC 模块的一部分，在章节 9 低功耗管理 *(RTC_CNTL)* 的第 9.5 节 寄存器列表 中有详

细描述。

13 系统寄存器 (SYSTEM)

13.1 概述

ESP8684 集成了丰富的外设，且允许对不同外设模块进行独立控制，从而在保持最佳性能的同时将功耗降至最低。具体来说，ESP8684 设计了一系列系统配置寄存器，用于芯片的时钟管理（时钟门控）、功耗管理、外设模块及核心模块配置。本章将简要例举这些系统配置寄存器及其功能。

13.2 主要特性

ESP8684 的系统寄存器可用于控制以下外设和模块：

- 系统和存储器
- 时钟
- 软件中断
- 外设时钟门控和复位

13.3 功能描述

13.3.1 系统和存储器寄存器

13.3.1.1 内部存储器

以下寄存器用以控制 ESP8684 内存的功耗，具体来说：

- 在寄存器 SYSCON_CLKGATE_FORCE_ON_REG 中：
  - 设置 SYSCON_ROM_CLKGATE_FORCE_ON 的相应位可分别控制 Internal ROM 0 和 Internal ROM 1 的时钟门控；
  - 设置 SYSCON_SRAM_CLKGATE_FORCE_ON 的相应位可分别控制 Internal SRAM 的时钟门控。
  - 配置为 1 时，ROM 或 SRAM 内存的时钟门控始终开启；配置为 0 时，则 ROM 或 SRAM 内存的时钟门控在被访问时自动打开，没有访问时自动关闭。因此，建议将本寄存器配置为 0，以降低功耗。
- 在寄存器 SYSCON_MEM_POWER_DOWN_REG 中：
  - 设置 SYSCON_ROM_POWER_DOWN 的相应位可分别控制 Internal ROM 0 和 Internal ROM 1 进入 Retention 状态；
  - 设置 SYSCON_SRAM_POWER_DOWN 的相应位可分别控制 Internal SRAM 进入 Retention 状态。
  - Retention 状态是存储器的一种低功耗模式。在此状态下，存储器中的数据不会丢失，但是不允许访问，因此可降低功耗。所以，如果用户在一段时间内不会访问某些存储器，也可以配置此寄存器让这些存储器进入 Retention 状态，以降低功耗。
- 在寄存器 SYSCON_MEM_POWER_UP_REG 中：
  - 默认情况下，芯片进入 Light-sleep 时会让所有的存储器进入 Retention 状态。
  - 设置 SYSCON_ROM_POWER_UP 的相应位可分别控制 Internal ROM 0 和 Internal ROM 1 在芯片进入 Light-sleep 时不会进入 Retention 状态；

- 设置 SYSCON_SRAM_POWER_UP 的相应位可分别控制 Internal SRAM 在芯片进入 Light-sleep 时不会进入 Retention 状态。

有关上述所有寄存器中各控制位和对应内存的控制关系，请见下方表 13-1。

表 13-1. 内存功耗控制位

内存	指令低地址	指令高地址	数据低地址	数据高地址	控制域
ROM 0	0x4000_0000	0x4003_FFFF	-	-	Bit0
ROM 1	0x4004_0000	0x4007_FFFF	0x3FF0_0000	0x3FF3_FFFF	Bit1
	0x4008_0000	0x4008_FFFF	0x3FF4_0000	0x3FF4_FFFF	Bit2
SRAM Block 0	0x4037_C000	0x4037_FFFF	-	-	Bit0
SRAM Block 1	0x4038_0000	0x4038_FFFF	0x3FCA_0000	0x3FCA_FFFF	Bit1
SRAM Block 2	0x4039_0000	0x4039_FFFF	0x3FCB_0000	0x3FCB_FFFF	Bit2
SRAM Block 3	0x403A_0000	0x403B_FFFF	0x3FCC_0000	0x3FCD_FFFF	Bit3

更多信息，请见章节 3 *系统和存储器*。

13.3.1.2 片外存储器

SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 可用于控制外部存储的加解密配置，详情请见章节 17 *片外存储器加密与解密 (XTS_AES)*。

13.3.2 时钟配置寄存器

以下系统寄存器用于系统和外设时钟源和时钟频率的配置。更多信息，请见章节 6 *复位和时钟*。

- SYSTEM_CPU_PER_CONF_REG
- SYSTEM_SYSCLK_CONF_REG

13.3.3 中断信号寄存器

以下系统寄存器用于产生中断信号（软件中断），经过配置可通过中断矩阵，产生不同的 CPU 外设中断。当以下寄存器写为 1 时，会产生中断信号，可用于软件自己控制中断的产生；清 0 则会清除对应的中断信号。以下寄存器与中断源 SW_INTR_0/1/2/3 一一对应。更多信息，请见章节 8 *中断矩阵 (INTMTRX)*。

- SYSTEM_CPU_INTR_FROM_CPU_0_REG
- SYSTEM_CPU_INTR_FROM_CPU_1_REG
- SYSTEM_CPU_INTR_FROM_CPU_2_REG
- SYSTEM_CPU_INTR_FROM_CPU_3_REG

13.3.4 外设时钟门控和复位寄存器

以下系统寄存器用于控制外设时钟门控和复位，相应位分别为不同外设的门控使能和复位使能，详见下方表 13-2。

- SYSTEM_CACHE_CONTROL_REG
- SYSTEM_GDMA_CTRL_REG

- SYSTEM_PERIP_CLK_ENO_REG
- SYSTEM_PERIP_RST_ENO_REG
- SYSTEM_PERIP_CLK_EN1_REG
- SYSTEM_PERIP_RST_EN1_REG

ESP8684 具有低功耗特性，因此有些外设时钟默认为关闭状态。在启用这些外设之前，必须将外设的时钟打开，并且解除外设的复位状态，具体见下表：

表 13-2. 外设时钟门控与复位控制位

组件	时钟使能位 [1]	复位使能位 [2, 3]
Cache 控制	**SYSTEM_CACHE_CONTROL_REG**	
DCACHE	SYSTEM_DCACHE_CLK_ON	SYSTEM_DCACHE_RESET
ICACHE	SYSTEM_ICACHE_CLK_ON	SYSTEM_ICACHE_RESET
GDMA	**SYSTEM_GDMA_CTRL_REG**	
GDMA	SYSTEM_GDMA_CLK_ON	SYSTEM_GDMA_RESET
CPU	**SYSTEM_CPU_PERI_CLK_EN_REG**	**SYSTEM_CPU_PERI_RST_EN_REG**
DEBUG_ASSIST	SYSTEM_CLK_EN_ASSIST_DEBUG	SYSTEM_RST_EN_ASSIST_DEBUG
外设	**SYSTEM_PERIP_CLK_ENO_REG**	**SYSTEM_PERIP_RST_ENO_REG**
SPI0 / SPI1	SYSTEM_SPI01_CLK_EN	SYSTEM_SPI01_RST
UART0	SYSTEM_UART_CLK_EN	SYSTEM_UART_RST
UART1	SYSTEM_UART1_CLK_EN	SYSTEM_UART1_RST
SPI2	SYSTEM_SPI2_CLK_EN	SYSTEM_SPI2_RST
I2C0	SYSTEM_I2C_EXT0_CLK_EN	SYSTEM_I2C_EXT0_RST
LED PWM 控制器	SYSTEM_LEDC_CLK_EN	SYSTEM_LEDC_RST
Timer Group0	SYSTEM_TIMERGROUP_CLK_EN	SYSTEM_TIMERGROUP_RST
UART MEM	SYSTEM_UART_MEM_CLK_EN [4]	SYSTEM_UART_MEM_RST
APB SARADC	SYSTEM_APB_SARADC_CLK_EN	SYSTEM_APB_SARADC_RST
System 定时器	SYSTEM_SYSTIMER_CLK_EN	SYSTEM_SYSTIMER_RST
ADC 控制器	SYSTEM_ADC2_ARB_CLK_EN	SYSTEM_ADC2_ARB_RST
加速器	**SYSTEM_PERIP_CLK_EN1_REG**	**SYSTEM_PERIP_RST_EN1_REG**
SHA 加速器	SYSTEM_CRYPTO_SHA_CLK_EN	SYSTEM_CRYPTO_SHA_RST
ECC 加速器	SYSTEM_CRYPTO_ECC_CLK_EN	SYSTEM_CRYPTO_ECC_RST
DMA	SYSTEM_DMA_CLK_EN	SYSTEM_DMA_RST[5]
TSENS	SYSTEM_TSENS_CLK_EN	SYSTEM_TSENS_RST

[1] 时钟控制寄存器相应位置 1 表示打开对应时钟，置 0 表示关闭对应时钟。

[2] 复位寄存器相应位置 1 表示使能复位状态，对应外设进行复位，置 0 表示关闭复位状态，对应外设正常工作。

[3] 复位寄存器无法通过硬件清除，因此软件将外设复位后需要清除复位寄存器。

[4] UART 存储器为所有 UART 外设所共用，因此只要有一个 UART 在工作，UART 存储器就不能处于门控状态。

[5] 当外设需要通过 DMA 进行数据传输时，比如 SPI、SHA 等，需要同时将 DMA 的时钟打开。

13.4 寄存器列表

下表的所有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），具体基地址请见章节 3 系统和存储器 中的表 3-3。

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
外设时钟控制寄存器			
SYSTEM_CPU_PERI_CLK_EN_REG	CPU 外设时钟使能寄存器	0x0000	读写
SYSTEM_CPU_PERI_RST_EN_REG	CPU 外设时钟复位寄存器	0x0004	读写
SYSTEM_PERIP_CLK_EN0_REG	系统外设时钟使能寄存器 0	0x0010	读写
SYSTEM_PERIP_CLK_EN1_REG	系统外设时钟使能寄存器 1	0x0014	读写
SYSTEM_PERIP_RST_EN0_REG	系统外设时钟复位寄存器 0	0x0018	读写
SYSTEM_PERIP_RST_EN1_REG	系统外设时钟复位寄存器 1	0x001C	读写
SYSTEM_GDMA_CTRL_REG	GDMA 时钟控制寄存器	0x003C	读写
SYSTEM_CACHE_CONTROL_REG	Cache 时钟控制寄存器	0x0040	读写
时钟配置寄存器			
SYSTEM_CPU_PER_CONF_REG	CPU 时钟配置寄存器	0x0008	可变
SYSTEM_SYSCLK_CONF_REG	系统时钟配置寄存器	0x0058	读写
CPU 中断控制寄存器			
SYSTEM_CPU_INTR_FROM_CPU_0_REG	CPU 中断控制寄存器 0	0x0028	读写
SYSTEM_CPU_INTR_FROM_CPU_1_REG	CPU 中断控制寄存器 1	0x002C	读写
SYSTEM_CPU_INTR_FROM_CPU_2_REG	CPU 中断控制寄存器 2	0x0030	读写
SYSTEM_CPU_INTR_FROM_CPU_3_REG	CPU 中断控制寄存器 3	0x0034	读写
系统和内存控制寄存器			
SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG	外部内存加解密控制寄存器	0x0044	读写
时钟门控控制寄存器			
SYSTEM_CLOCK_GATE_REG	时钟门控寄存器	0x0054	读写
日期寄存器			
SYSTEM_DATE_REG	版本寄存器	0x0FFC	读写

下表的所有地址均为相对于 APB 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 3 系统和存储器 中的表 3-3。

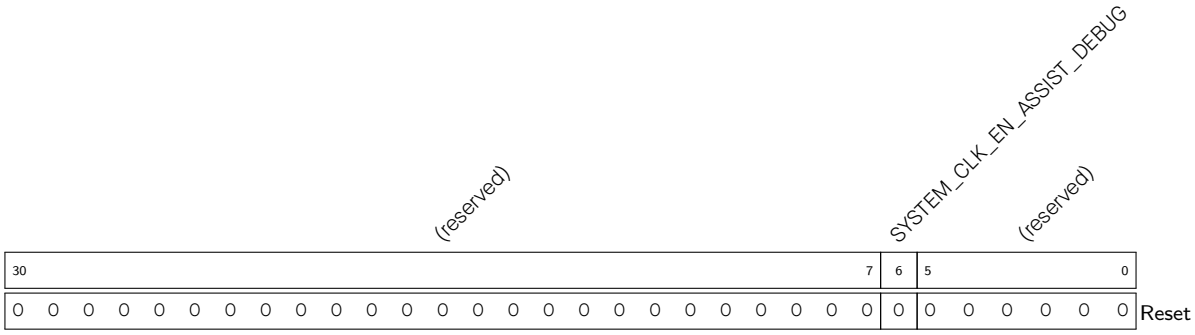
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问权限
配置寄存器			
SYSCON_CLKGATE_FORCE_ON_REG	内存时钟门控使能寄存器	0x00A4	R/W
SYSCON_MEM_POWER_DOWN_REG	内存控制寄存器	0x00A8	R/W
SYSCON_MEM_POWER_UP_REG	内存控制寄存器	0x00AC	R/W

13.5 寄存器

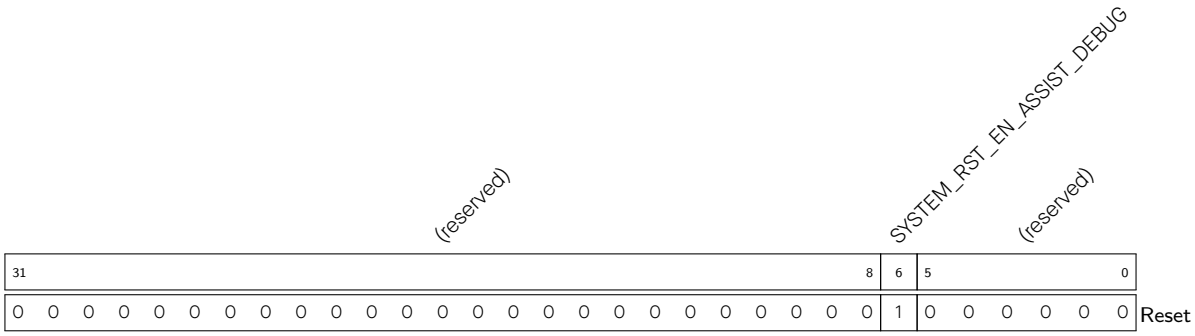
以下所有地址均为相对于系统寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 13.1. SYSTEM_CPU_PERI_CLK_EN_REG (0x0000)



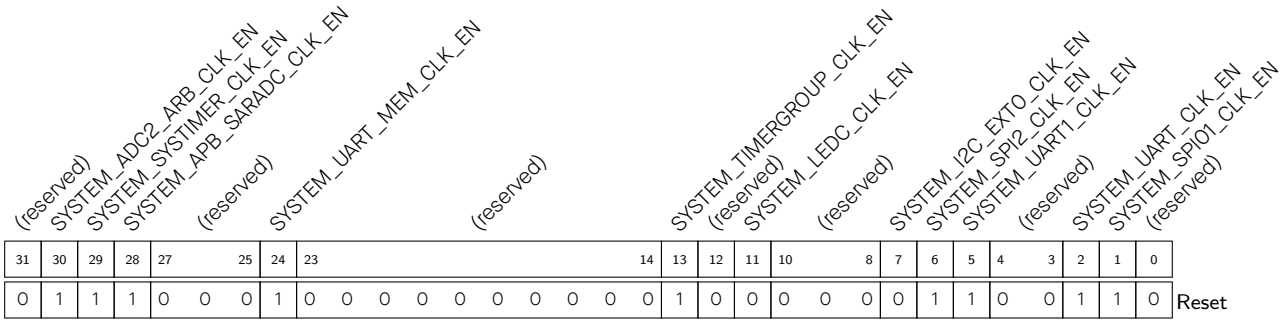
SYSTEM_CLK_EN_ASSIST_DEBUG 置 1 使能 ASSIST_DEBUG 时钟。更多信息，请见章节 14 辅助调试 *(ASSIST_DEBUG)*。(R/W)

Register 13.2. SYSTEM_CPU_PERI_RST_EN_REG (0x0004)



SYSTEM_RST_EN_ASSIST_DEBUG 置 1 复位 ASSIST_DEBUG 时钟。更多信息，请见章节 14 辅助调试 *(ASSIST_DEBUG)*。(R/W)

Register 13.3. SYSTEM_PERIP_CLK_EN0_REG (0x0010)



SYSTEM_SPI01_CLK_EN 置 1 使能 SPI0 / SPI1 时钟。(R/W)

SYSTEM_UART_CLK_EN 置 1 使能 UART 时钟。(R/W)

SYSTEM_UART1_CLK_EN 置 1 使能 UART1 时钟。(R/W)

SYSTEM_SPI2_CLK_EN 置 1 使能 SPI2 时钟。(R/W)

SYSTEM_I2C_EXT0_CLK_EN 置 1 使能 I2C_EXT0 时钟。(R/W)

SYSTEM_LEDC_CLK_EN 置 1 使能 LEDC 时钟。(R/W)

SYSTEM_TIMERGROUP_CLK_EN 置 1 使能 TIMER GROUP 时钟。(R/W)

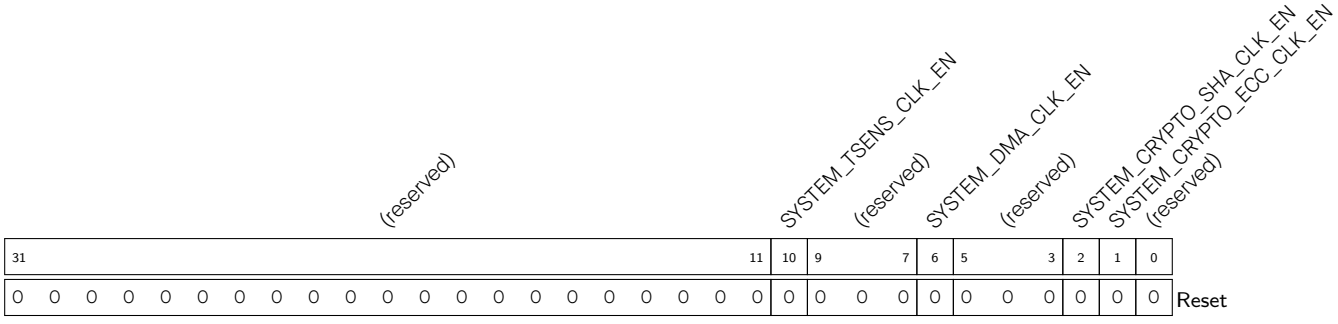
SYSTEM_UART_MEM_CLK_EN 置 1 使能 UART_MEM 时钟。(R/W)

SYSTEM_APB_SARADC_CLK_EN 置 1 使能 APB_SARADC 时钟。(R/W)

SYSTEM_SYSTIMER_CLK_EN 置 1 使能 SYSTEMTIMER 时钟。(R/W)

SYSTEM_ADC2_ARB_CLK_EN 置 1 使能 ADC2_ARB 时钟。(R/W)

Register 13.4. SYSTEM_PERIP_CLK_EN1_REG (0x0014)



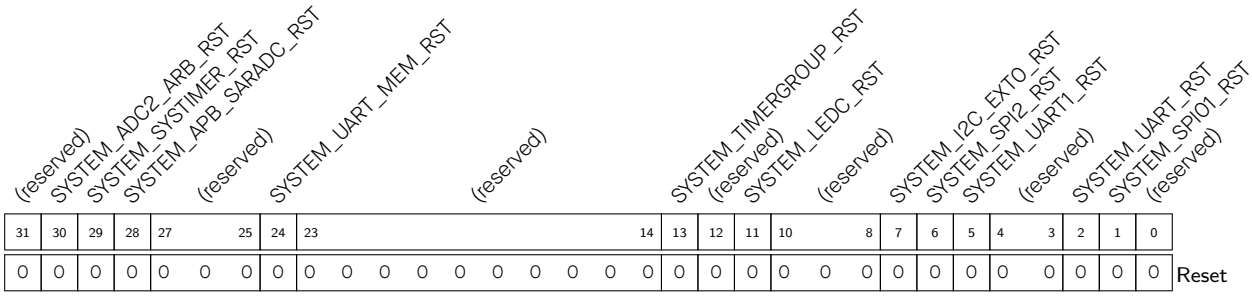
SYSTEM_CRYPTO_ECC_CLK_EN 置 1 使能 ECC 时钟。(R/W)

SYSTEM_CRYPTO_SHA_CLK_EN 置 1 使能 SHA 时钟。(R/W)

SYSTEM_DMA_CLK_EN 置 1 使能 DMA 时钟。(R/W)

SYSTEM_TSENS_CLK_EN 置 1 使能 TSENS 时钟。(R/W)

Register 13.5. SYSTEM_PERIP_RST_EN0_REG (0x0018)



SYSTEM_SPI01_RST 置 1 复位 SPI01。(R/W)

SYSTEM_UART_RST 置 1 复位 UART。(R/W)

SYSTEM_UART1_RST 置 1 复位 UART1。(R/W)

SYSTEM_SPI2_RST 置 1 复位 SPI2。(R/W)

SYSTEM_I2C_EXT0_RST 置 1 复位 I2C_EXT0。(R/W)

SYSTEM_RMT_RST 置 1 复位 RMT。(R/W)

SYSTEM_LEDC_RST 置 1 复位 LEDC。(R/W)

SYSTEM_TIMERGROUP_RST 置 1 复位 TIMERGROUP。(R/W)

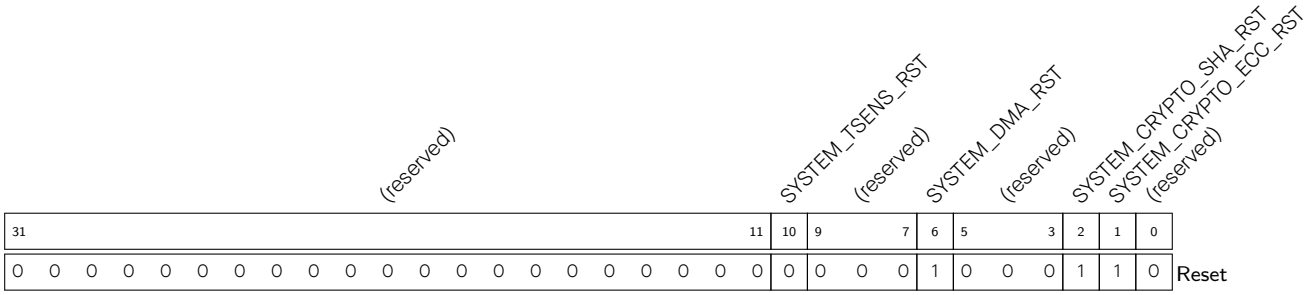
SYSTEM_UART_MEM_RST 置 1 复位 UART_MEM。(R/W)

SYSTEM_APB_SARADC_RST 置 1 复位 APB_SARADC。(R/W)

SYSTEM_SYSTIMER_RST 置 1 复位 SYSTIMER。(R/W)

SYSTEM_ADC2_ARB_RST 置 1 复位 ADC2_ARB。(R/W)

Register 13.6. SYSTEM_PERIP_RST_EN1_REG (0x001C)



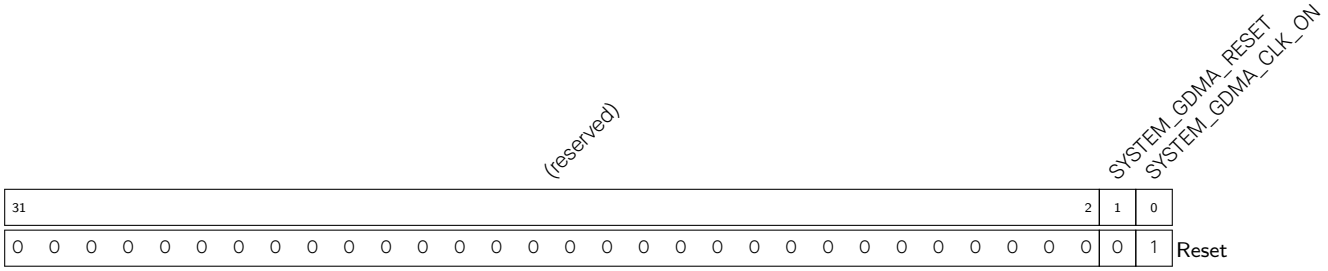
SYSTEM_CRYPTO_ECC_RST 置 1 复位 CRYPTO_ECC。(R/W)

SYSTEM_CRYPTO_SHA_RST 置 1 复位 CRYPTO_SHA。(R/W)

SYSTEM_DMA_RST 置 1 复位 DMA。(R/W)

SYSTEM_TSENS_RST 置 1 复位 TSENS。(R/W)

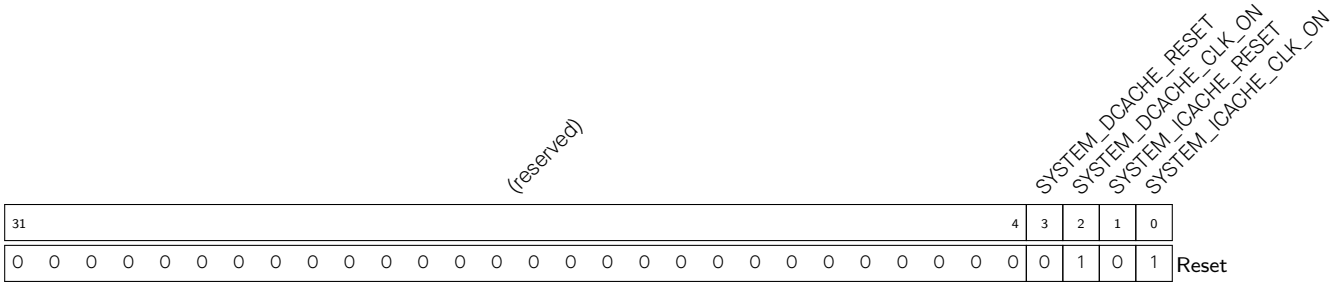
Register 13.7. SYSTEM_GDMA_CTRL_REG (0x003C)



SYSTEM_GDMA_CLK_ON 置 1 使能 GDMA 时钟. (R/W)

SYSTEM_GDMA_RESET 置 1 复位 GDMA. (R/W)

Register 13.8. SYSTEM_CACHE_CONTROL_REG (0x0040)



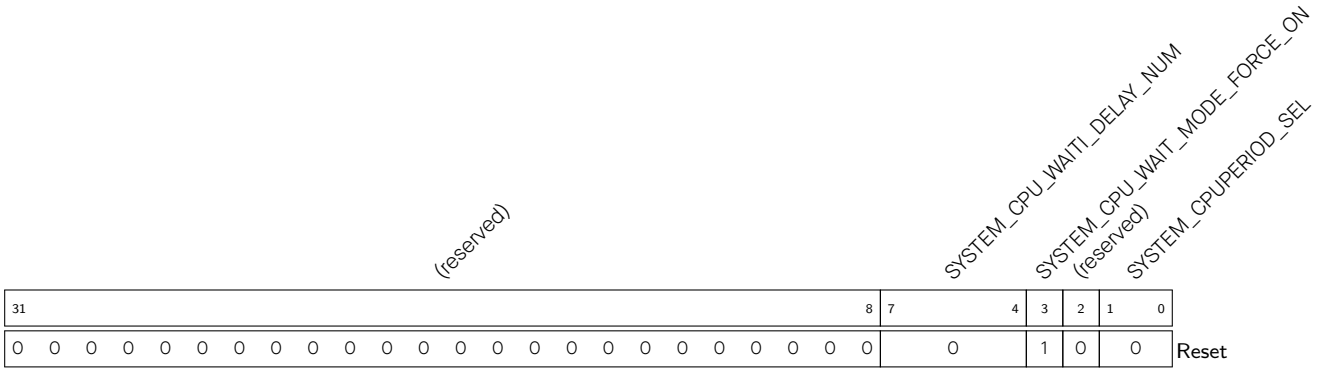
SYSTEM_ICACHE_CLK_ON 置 1 使能 i-cache 时钟。(R/W)

SYSTEM_ICACHE_RESET 置 1 复位 i-cache。(R/W)

SYSTEM_DCACHE_CLK_ON 置 1 使能 d-cache 时钟。(R/W)

SYSTEM_DCACHE_RESET 置 1 复位 d-cache。(R/W)

Register 13.9. SYSTEM_CPU_PER_CONF_REG (0x0008)

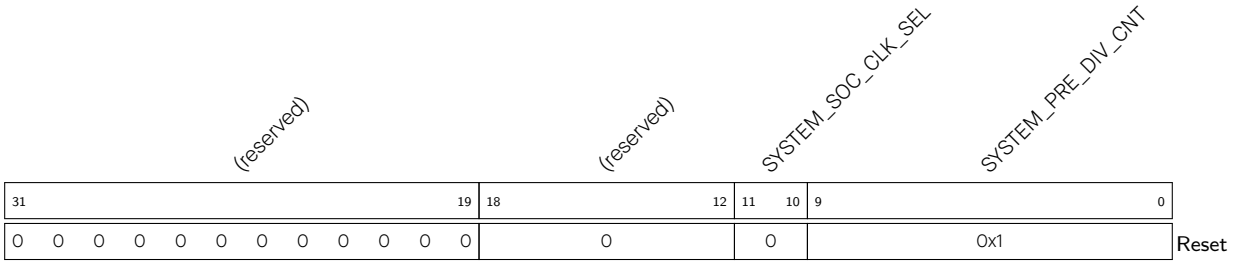


SYSTEM_CPUPERIOD_SEL 选择 CPU 时钟频率。具体配置，请见章节 6 复位和时钟 中的表 6-4。(R/W)

SYSTEM_CPU_WAIT_MODE_FORCE_ON 置 1 强制打开 CPU 等待中断模式下的门控时钟。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)

SYSTEM_CPU_WAITI_DELAY_NUM 设置 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。(R/W)

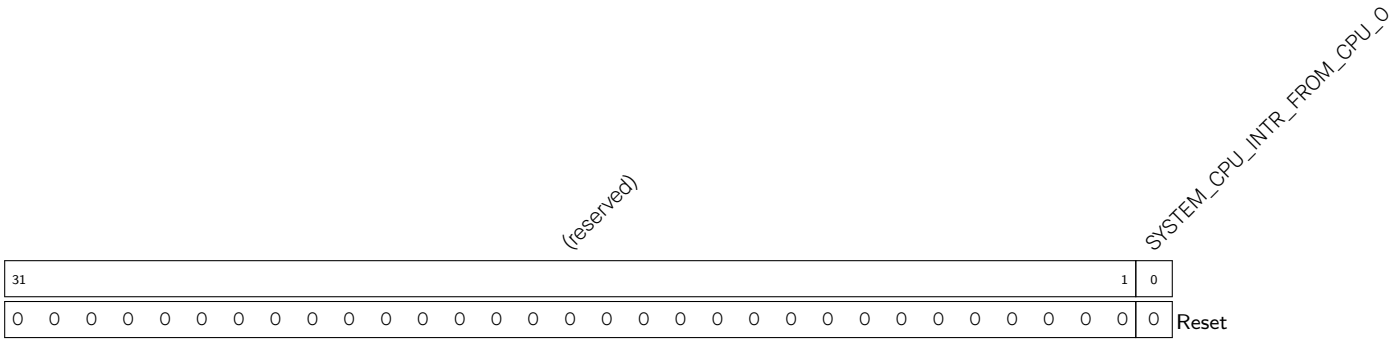
Register 13.10. SYSTEM_SYSCLK_CONF_REG (0x0058)



SYSTEM_PRE_DIV_CNT 设置预分频器计数器。具体配置，请见章节 6 复位和时钟 中的表 6-3。(R/W)

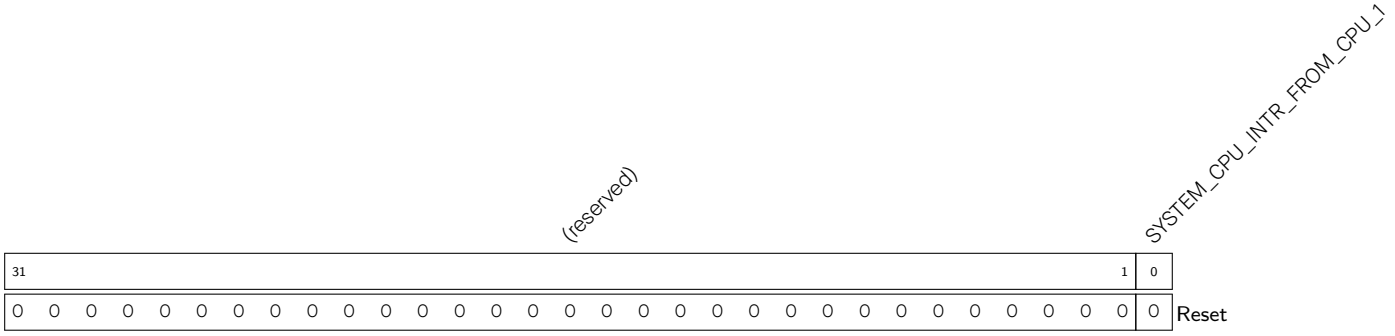
SYSTEM_SOC_CLK_SEL 选择 SoC 时钟。具体配置，请见章节 6 复位和时钟 中的表 6-4。(R/W)

Register 13.11. SYSTEM_CPU_INTR_FROM_CPU_0_REG (0x0028)



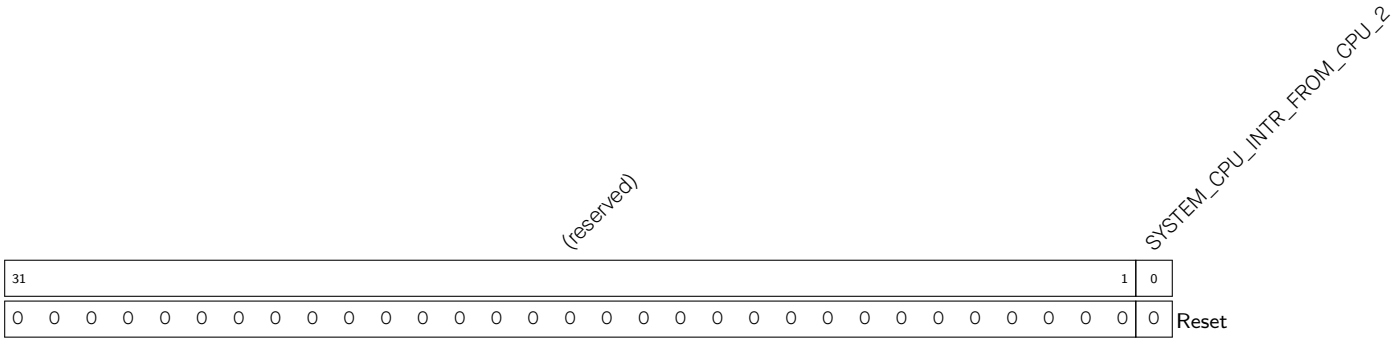
SYSTEM_CPU_INTR_FROM_CPU_0 置 1 生成 CPU 中断 0。该位需在 ISR 过程中由软件清 0。(R/W)

Register 13.12. SYSTEM_CPU_INTR_FROM_CPU_1_REG (0x002C)



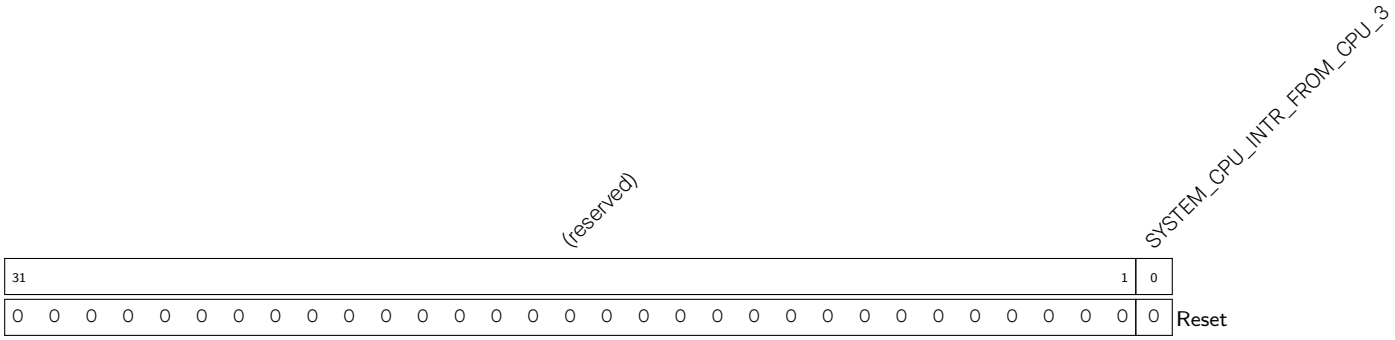
SYSTEM_CPU_INTR_FROM_CPU_1 置 1 生成 CPU 中断 1。该位需在 ISR 过程中由软件清 0。(R/W)

Register 13.13. SYSTEM_CPU_INTR_FROM_CPU_2_REG (0x0030)



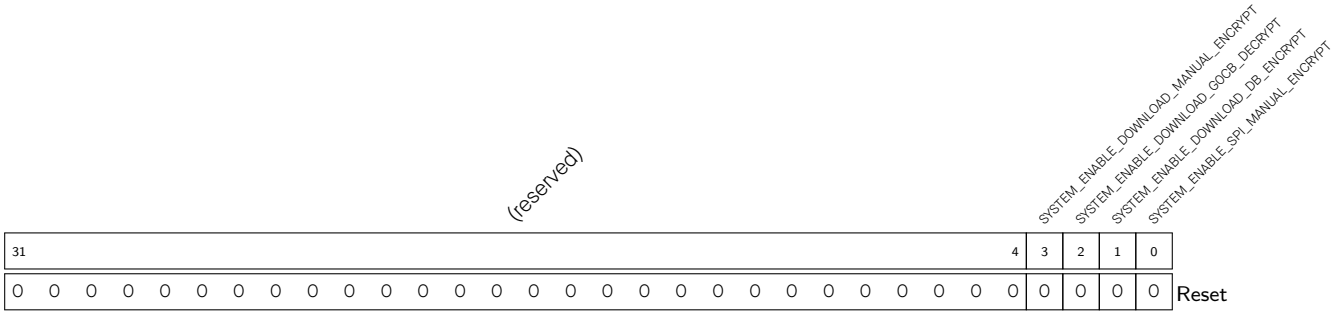
SYSTEM_CPU_INTR_FROM_CPU_2 置 1 生成 CPU 中断 2。该位需在 ISR 过程中由软件清 0。(R/W)

Register 13.14. SYSTEM_CPU_INTR_FROM_CPU_3_REG (0x0034)



SYSTEM_CPU_INTR_FROM_CPU_3 置 1 生成 CPU 中断 3。该位需在 ISR 过程中由软件清 0。(R/W)

Register 13.15. SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG (0x0044)



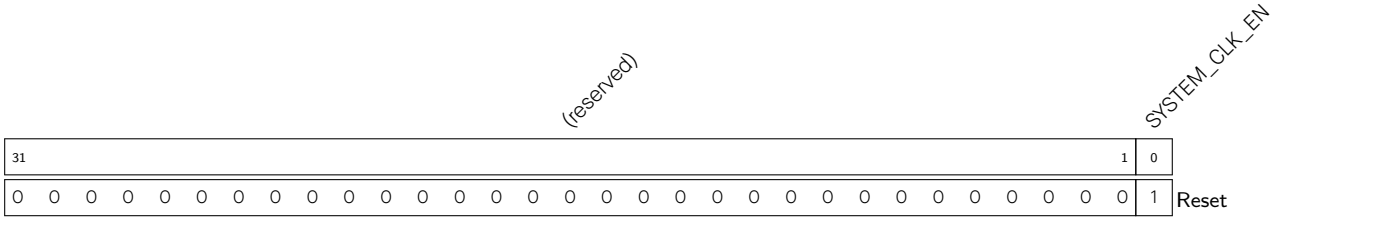
SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 置 1 在 SPI Boot 模式下使能手动加密 (Manual Encryption)。(R/W)

SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT 置 1 在 Download Boot 模式下使能自动加密 (Auto Encryption)。(R/W)

SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 置 1 在 Download Boot 模式下使能自动解密 (Auto Decryption)。(R/W)

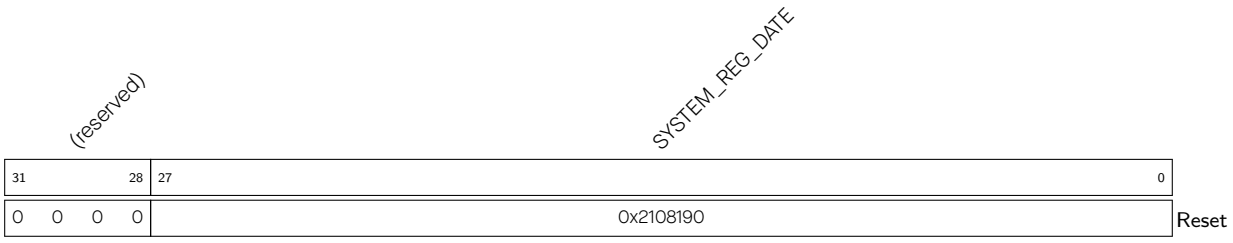
SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 置 1 在 Download Boot 模式下使能手动加密 (Manual Encryption)。(R/W)

Register 13.16. SYSTEM_CLOCK_GATE_REG (0x0054)



SYSTEM_CLK_EN 置 1 使能系统时钟。(R/W)

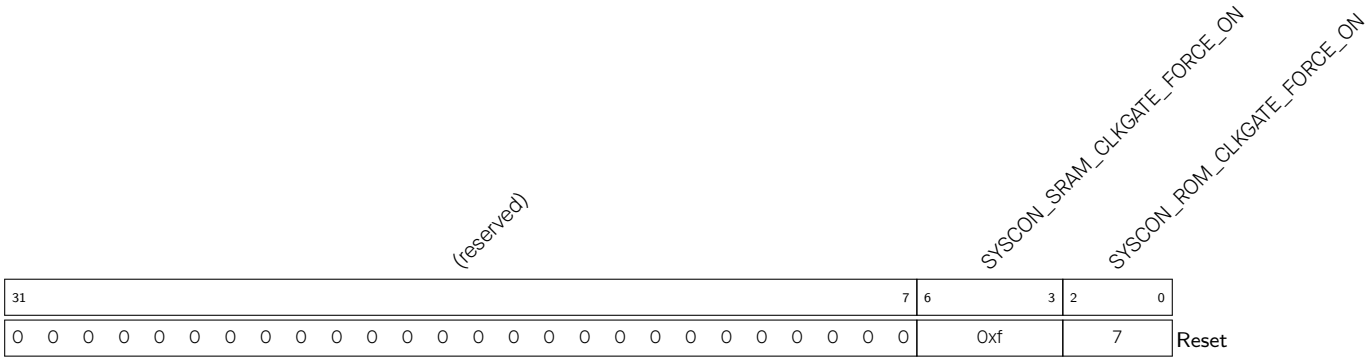
Register 13.17. SYSTEM_DATE_REG (0x0FFC)



SYSTEM_REG_DATE 版本控制寄存器。(R/W)

以下所有地址均为相对于 APB 控制寄存器基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

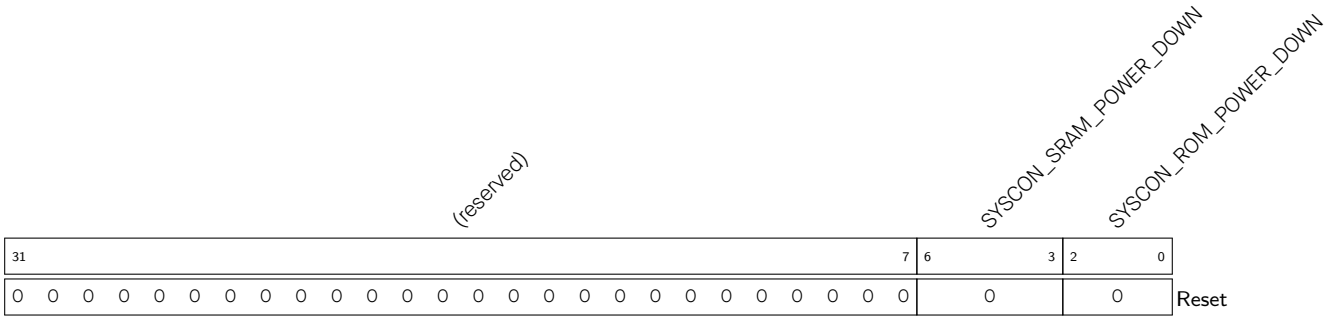
Register 13.18. SYSCON_CLKGATE_FORCE_ON_REG (0x00A4)



SYSCON_ROM_CLKGATE_FORCE_ON 置 1 配置 ROM 内存的时钟门控始终打开；置 0 则配置 ROM 内存的时钟门控在被访问时自动打开，没有访问时自动关闭。(R/W)

SYSCON_SRAM_CLKGATE_FORCE_ON 置 1 配置 SRAM 内存的时钟门控始终打开；置 0 则配置 SRAM 内存的时钟门控在被访问时自动打开，没有访问时自动关闭。(R/W)

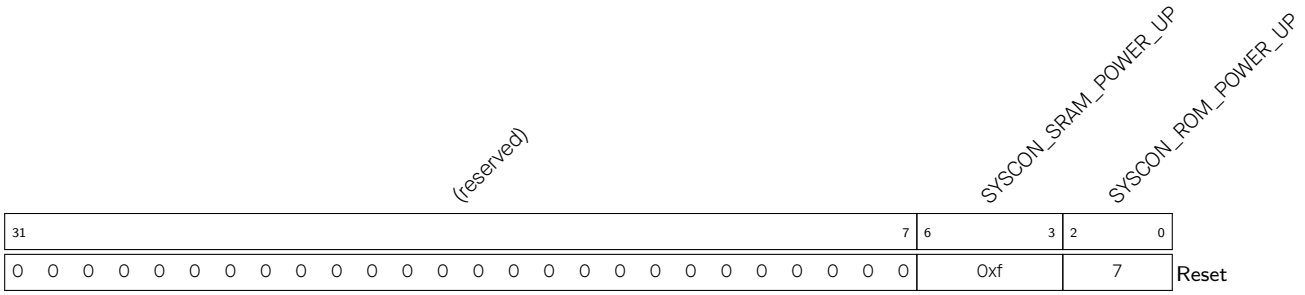
Register 13.19. SYSCON_MEM_POWER_DOWN_REG (0x00A8)



SYSCON_ROM_POWER_DOWN 控制 Internal ROM 进入 Retention 状态。(R/W)

SYSCON_SRAM_POWER_DOWN 控制 Internal SRAM 进入 Retention 状态。(R/W)

Register 13.20. SYSCON_MEM_POWER_UP_REG (0x00AC)



SYSCON_ROM_POWER_UP 控制 Internal ROM 在芯片进入 Light-sleep 时不会进入 Retention 状态。(R/W)

SYSCON_SRAM_POWER_UP 控制 Internal SRAM 在芯片进入 Light-sleep 时不会进入 Retention 状态。(R/W)

14 辅助调试 (ASSIST_DEBUG)

14.1 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。

14.2 主要特性

辅助调试模块具有如下特性：

- 支持监测栈指针 (SP)
- 支持记录 CPU 复位前的程序计数器 (PC)
- 支持查看 CPU 调试状态信息

14.3 功能描述

14.3.1 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，然后用户可以读取记录的 PC 值来确定导致越界访问的指令。上下边界值必须由软件进行配置。

14.3.2 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。

14.3.3 CPU 调试状态记录

辅助调试模块提供一组只读寄存器，可获取 CPU 的调试状态信息，详情请参考 1 *ESP-RISC-V CPU* 章节。

14.4 工作流程

14.4.1 栈监测配置

栈指针监测：

- 监测栈指针是否越过上限
- 监测栈指针是否越过下限

栈监测的配置流程如下：

1. 配置栈指针监测范围 ASSIST_DEBUG_CORE_0_SP_MIN_REG 和 ASSIST_DEBUG_CORE_0_SP_MAX_REG
2. 配置中断
   - 配置 ASSIST_DEBUG_CORE_0_INTR_EN_REG 用于使能不同监测模式的中断。
   - 查询 ASSIST_DEBUG_CORE_0_INTR_RAW_REG 获取不同监测模式的中断状态。

- 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_SP_MONITOR_EN_REG 使能不同的监测模式，可同时使能。

读取 ASSIST_DEBUG_CORE_0_SP_PC 可获取触发中断时刻的 PC 值。

辅助调试模块的中断对应中断矩阵的中断源 ASSIST_DEBUG_INTR，关于如何将该中断源映射到 CPU 中断，请参考 8 中断矩阵 *(INTMTRX)*中断矩阵章节。

14.4.2 PC 记录配置

CPU 输出一个 PC 值给辅助调试模块，当 ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 配置为 1 时，该 PC 才有效，否则一直为 0。同时当 ASSIST_DEBUG_CORE_0_RCD_RECORDEN 配置为 1 时，ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 会去采样 CPU PC，否则保持原值。

寄存器 ASSIST_DEBUG_CORE_0_RCD_EN_REG、ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 描述见 14.8、14.9。

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 不会被复位，因此后者会一直保持复位时刻的 PC 值。ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG 记录复位时刻的 SP 值。

14.5 寄存器列表

本小节的所有地址均为相对于辅助调试基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

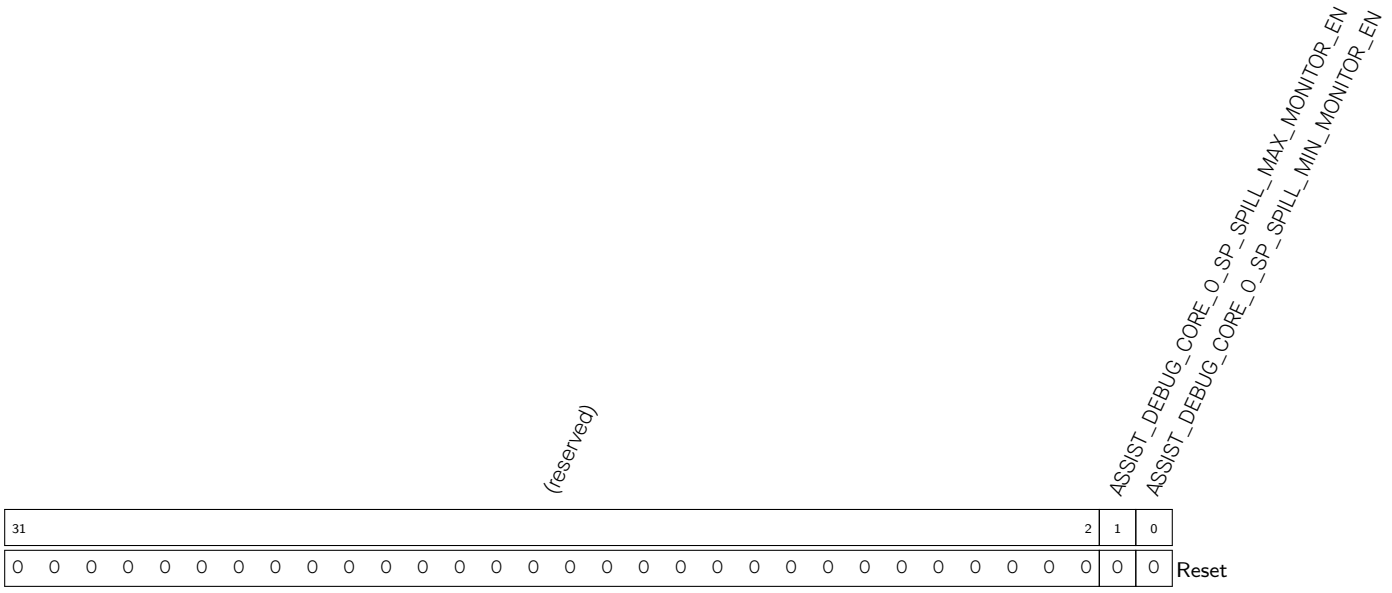
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
监测配置寄存器			
ASSIST_DEBUG_CORE_0_SP_MONITOR_EN_REG	配置监测模式	0x0000	R/W
ASSIST_DEBUG_CORE_0_SP_MIN_REG	配置栈指针下边界	0x0010	R/W
ASSIST_DEBUG_CORE_0_SP_MAX_REG	配置栈指针上边界	0x0014	R/W
ASSIST_DEBUG_CORE_0_SP_PC_REG	储存中断产生时的 PC 值	0x0018	RO
中断配置寄存器			
ASSIST_DEBUG_CORE_0_INTR_RAW_REG	储存监测模式下的中断状态	0x0004	RO
ASSIST_DEBUG_CORE_0_INTR_EN_REG	使能监测模式下的中断	0x0008	R/W
ASSIST_DEBUG_CORE_0_INTR_CLR_REG	清除监测模式下的中断	0x000C	WT
PC 记录配置寄存器			
ASSIST_DEBUG_CORE_0_RCD_EN_REG	使能 PC 记录	0x001C	R/W
PC 记录状态寄存器			
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG	记录 PC 值	0x0020	RO
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG	记录 SP 值	0x0024	RO
CPU 状态寄存器			
ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG	储存 CPU 异常前的最后一条指令的 PC	0x0028	RO
ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG	储存 CPU 调试模式的状态	0x002C	RO
时钟门寄存器			
ASSIST_DEBUG_CLOCK_GATE_REG	时钟门寄存器	0x0030	R/W
版本寄存器			
ASSIST_DEBUG_DATE_REG	版本控制寄存器	0x01FC	R/W

14.6 寄存器

本小节的所有地址均为相对于辅助调试基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

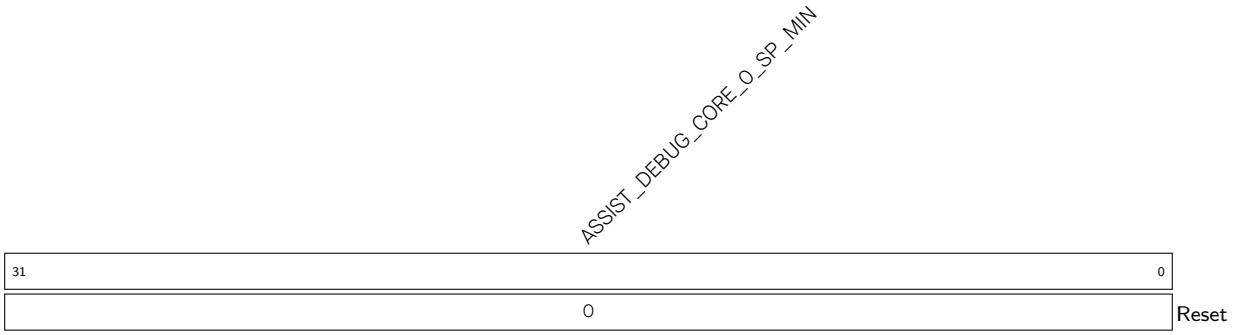
Register 14.1. ASSIST_DEBUG_CORE_0_SP_MONITOR_EN_REG (0x0000)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_MONITOR_EN 置 1 使能栈指针下溢监测。(R/W)

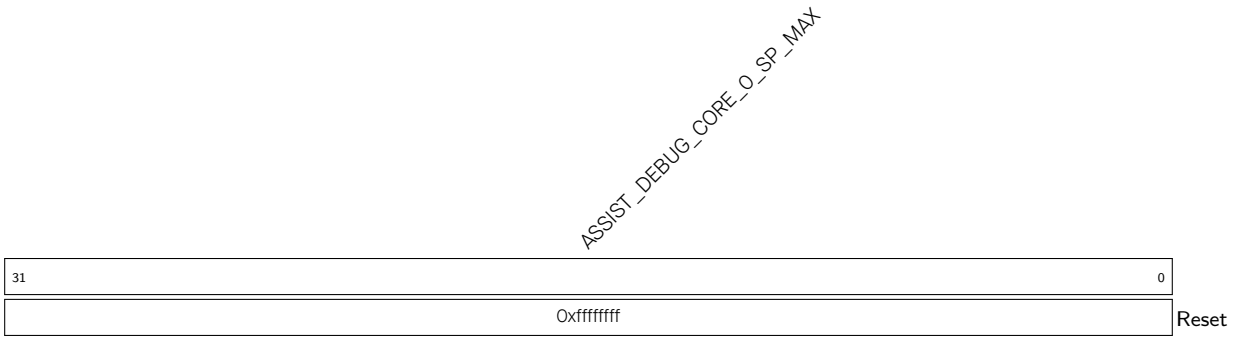
ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_MONITOR_EN 置 1 使能栈指针上溢监测。(R/W)

Register 14.2. ASSIST_DEBUG_CORE_0_SP_MIN_REG (0x0010)



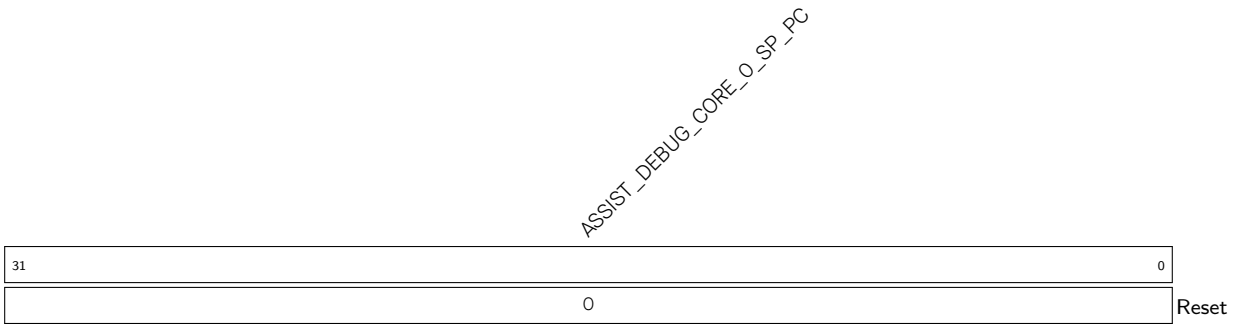
ASSIST_DEBUG_CORE_0_SP_MIN 记录栈指针的下边界。(R/W)

Register 14.3. ASSIST_DEBUG_CORE_0_SP_MAX_REG (0x0014)



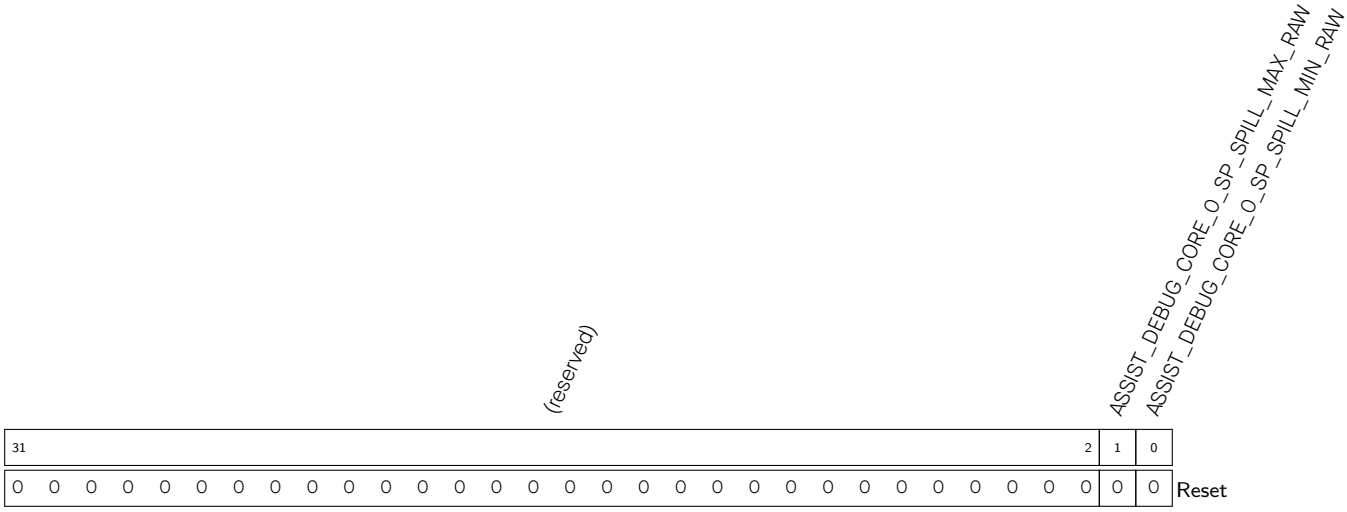
ASSIST_DEBUG_CORE_0_SP_MAX 栈指针的上边界。(R/W)

Register 14.4. ASSIST_DEBUG_CORE_0_SP_PC_REG (0x0018)



ASSIST_DEBUG_CORE_0_SP_PC 记录栈指针监测的 PC 值。(RO)

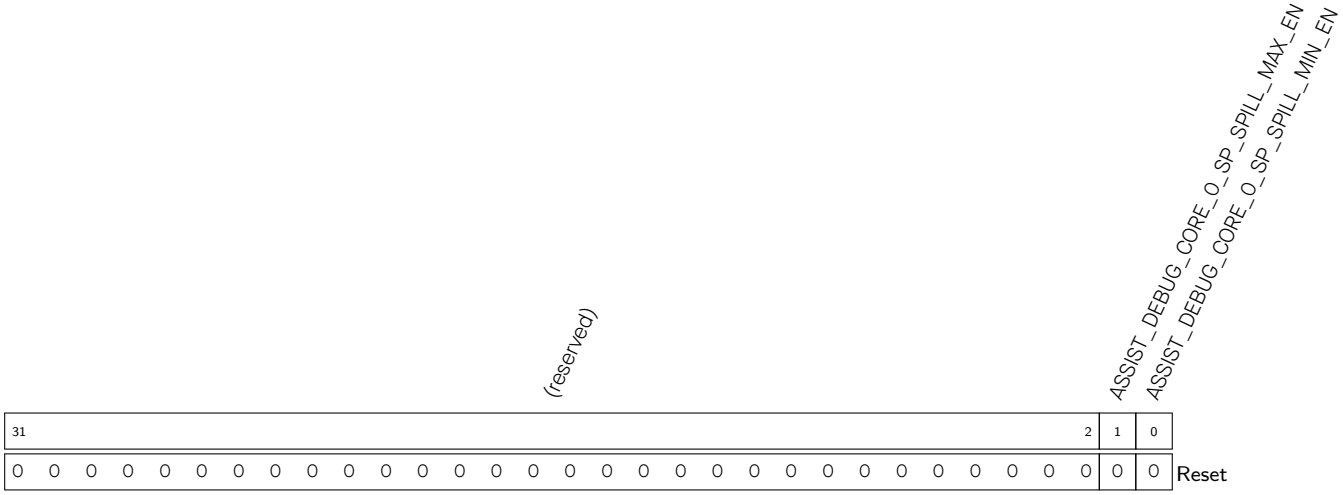
Register 14.5. ASSIST_DEBUG_CORE_0_INTR_RAW_REG (0x0004)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_RAW 储存栈指针下溢监测的中断状态。(RO)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_RAW 储存栈指针上溢监测的中断状态。(RO)

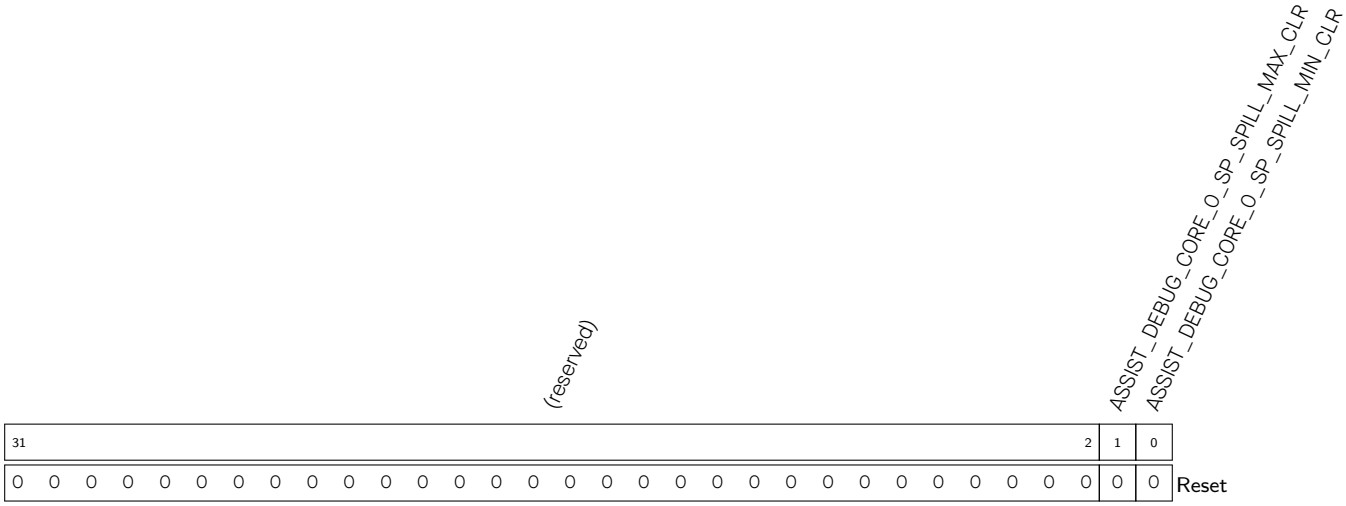
Register 14.6. ASSIST_DEBUG_CORE_0_INTR_EN_REG (0x0008)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_EN 使能栈指针下溢监测中断，1：使能，0：不使能。(R/W)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_EN 使能栈指针上溢监测中断，1：使能，0：不使能。(R/W)

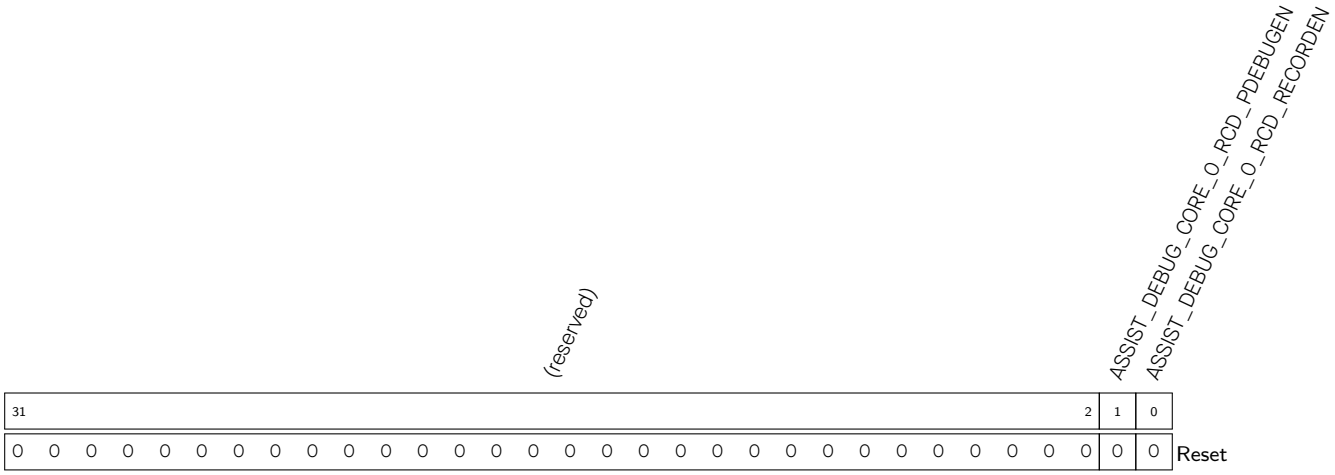
Register 14.7. ASSIST_DEBUG_CORE_0_INTR_CLR_REG (0x000C)



ASSIST_DEBUG_CORE_0_SP_SPILL_MIN_CLR 写 1 清除栈指针下溢监测中断。(WT)

ASSIST_DEBUG_CORE_0_SP_SPILL_MAX_CLR 写 1 清除栈指针上溢监测中断。(WT)

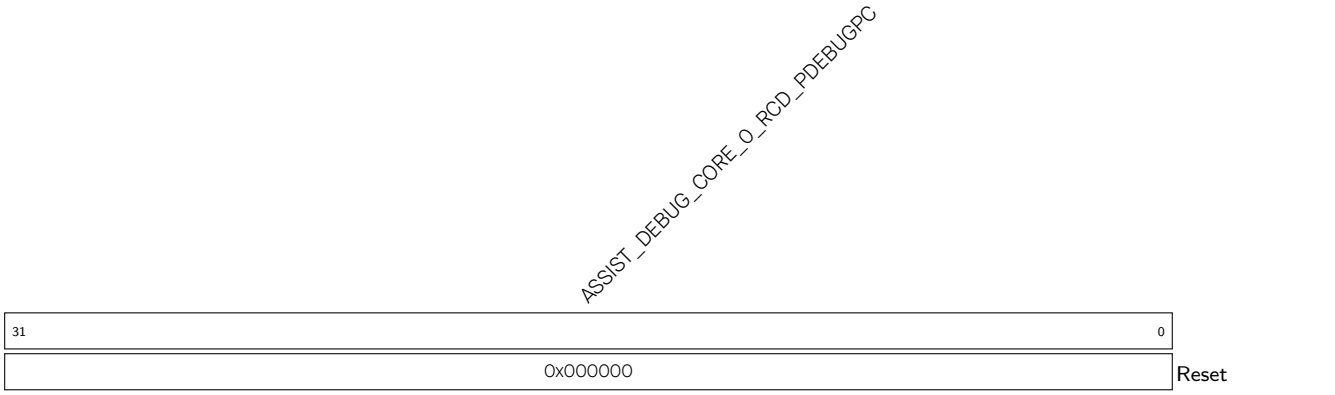
Register 14.8. ASSIST_DEBUG_CORE_0_RCD_EN_REG (0x001C)



ASSIST_DEBUG_CORE_0_RCD_RECORDEN 使能 PC 记录，配置为 1 时，ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG 开始实时记录 PC。(R/W)

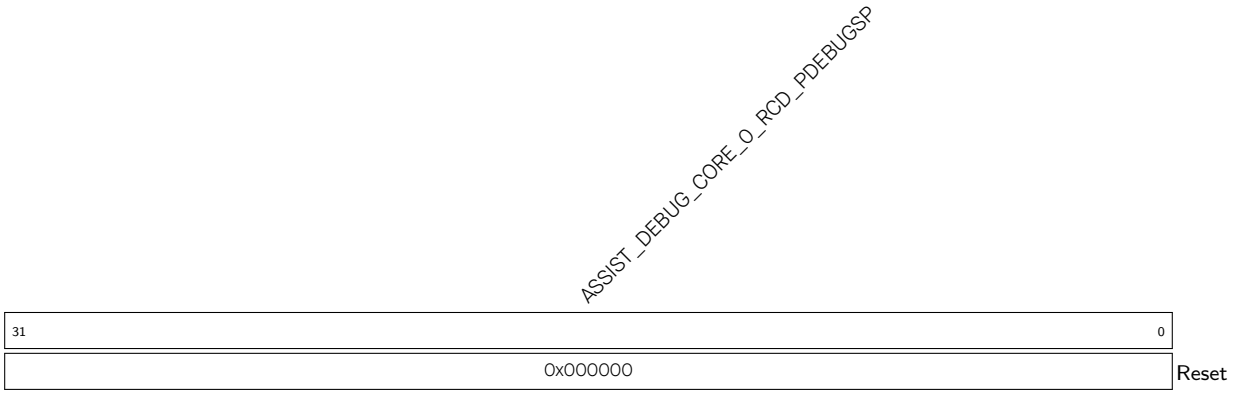
ASSIST_DEBUG_CORE_0_RCD_PDEBUGEN 使能 CPU 调试，配置为 1 时，CPU 才会输出 PC。(R/W)

Register 14.9. ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC_REG (0x0020)



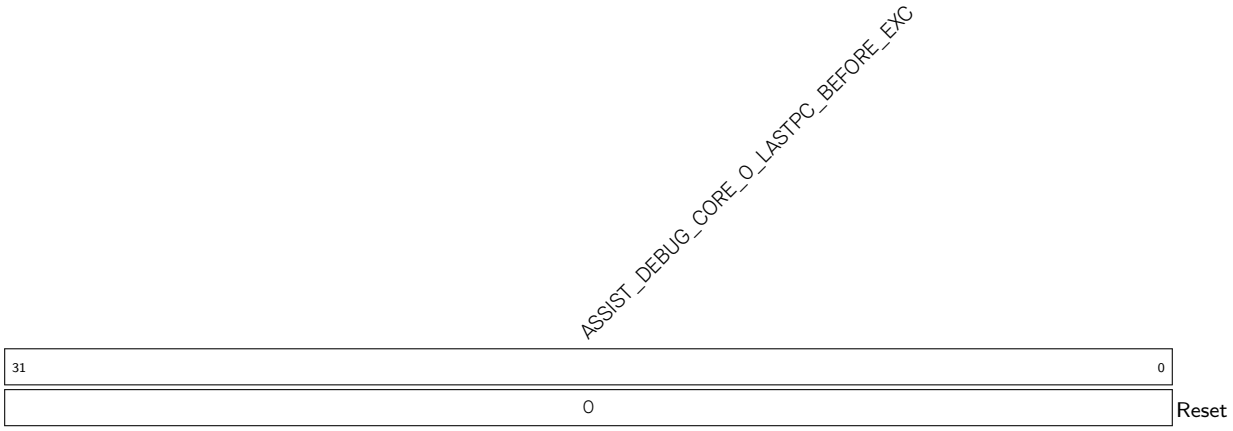
ASSIST_DEBUG_CORE_0_RCD_PDEBUGPC 记录复位时刻的 PC 值。(RO)

Register 14.10. ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP_REG (0x0024)



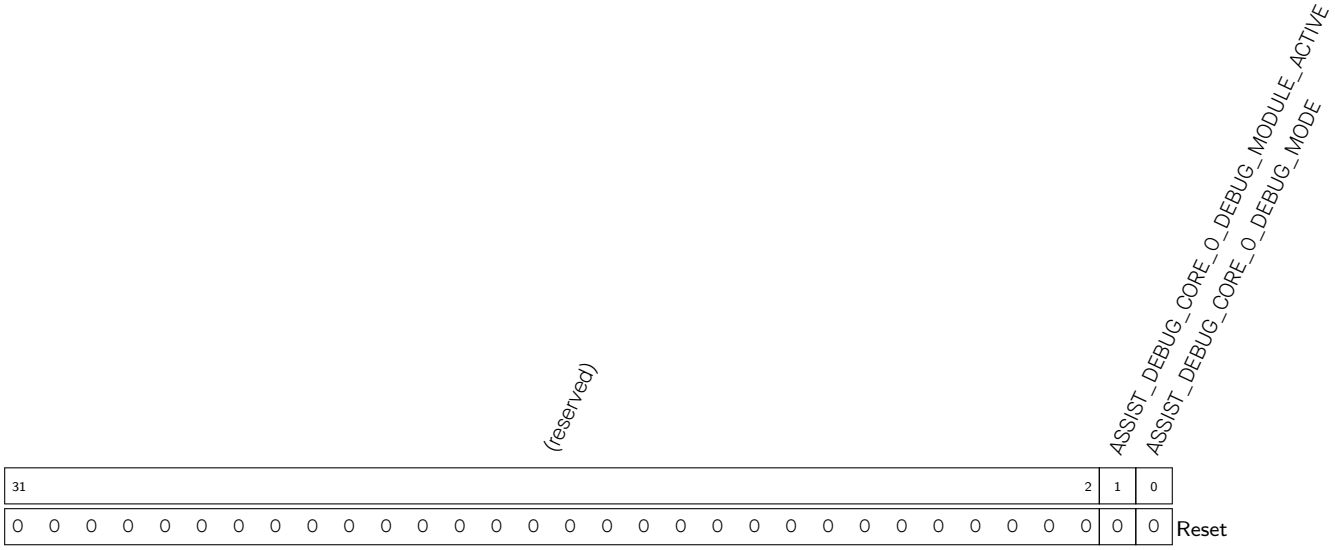
ASSIST_DEBUG_CORE_0_RCD_PDEBUGSP 记录 SP。(RO)

Register 14.11. ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXCEPTION_REG (0x0028)



ASSIST_DEBUG_CORE_0_LASTPC_BEFORE_EXC 记录 CPU 异常前的最后一条指令的 PC。(RO)

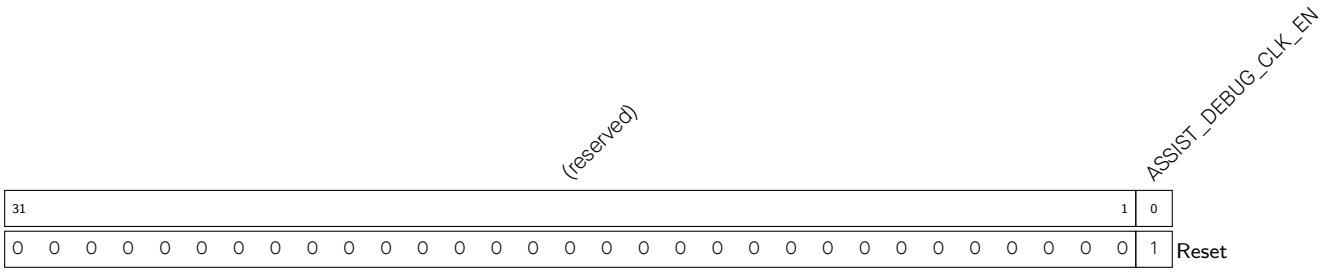
Register 14.12. ASSIST_DEBUG_CORE_0_DEBUG_MODE_REG (0x002C)



ASSIST_DEBUG_CORE_0_DEBUG_MODE 指示 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)

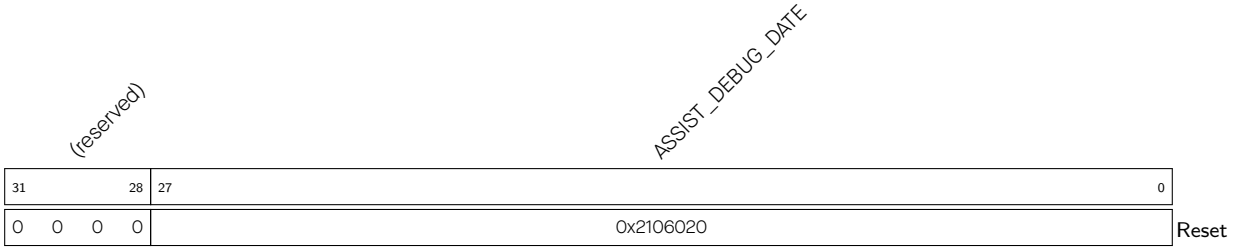
ASSIST_DEBUG_CORE_0_DEBUG_MODULE_ACTIVE 指示 RISC-V CPU 调试模块的工作状态。1：处于工作状态；其它：未处于工作状态。(RO)

Register 14.13. ASSIST_DEBUG_CLOCK_GATE_REG (0x0030)



ASSIST_DEBUG_CLK_EN 时钟门寄存器。(R/W)

Register 14.14. ASSIST_DEBUG_DATE_REG (0x01FC)



ASSIST_DEBUG_DATE 版本控制寄存器。(R/W)

15 ECC 硬件加速器 (ECC)

15.1 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。

ESP8684 ECC 硬件加速器支持对于可选曲线的多种基础运算，用以实现对 ECC 基本运算、衍生算法（如 ECDSA 等算法）的加速。

15.2 主要特性

ESP8684 ECC 硬件加速器支持以下功能:

- 支持 2 种可选 ECC 曲线，包括 FIPS 186-3 中定义的 P-192 和 P-256。
- 提供 7 种可选工作模式。
- 提供计算完成的中断和中断控制。

15.3 专业名词定义

为方便叙述说明，我们在此首先对 ECC 硬件加速器章节中会应用到的专业名词进行定义。

15.3.1 ECC 背景知识

15.3.1.1 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

15.3.1.2 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：

$$X = x$$

$$Y = y$$

$$Z = 1$$

15.3.2 ESP8684 ECC 相关定义

15.3.2.1 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输出数据。

表 15-1. ECC 硬件加速器内存块

内存块	大小 (byte)	起始地址*	结束地址*	访问属性
ECC_Mem_k	32	0x100	0x11F	R/W
ECC_Mem_Px	32	0x120	0x13F	R/W
ECC_Mem_Py	32	0x140	0x15F	R/W

* 采用相对于 ECC 加速器基地址的偏移量。详见章节 3 系统和存储器 中的表 3-3。

15.3.2.2 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 8 个 32 bit 位宽的数据块 $D[n][31:0](n = 0, 1, \cdots, 7)$，序号数低的数据块对应二进制低位，即：

$$D[255:0] = D[7][31:0], D[6][31:0], D[5][31:0], D[4][31:0], D[3][31:0], D[2][31:0], D[1][31:0], D[0][31:0]$$

15.3.2.3 数据存储

数据存储即将一个数据存储进一个内存块的操作，也可以说该数据为输入数据。具体来说，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”
- $\cdots$
- 写入 $D[7]$ 至 “起始地址 + 28”

说明：
在 192 bit 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 bit 位宽。

15.3.2.4 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，也可以说该数据为输出数据。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 $D[1]$
- $\cdots$
- 从“起始地址 + 28”读出 $D[7]$

说明：
在 192 bit 模式下，进行数据读取操作时，只需要读取 192 bit（即 6 个数据块）的数据。

15.3.2.5 标准运算与 Jacobian 运算

ESP8684 ECC 硬件加速器中，所有标准运算（包括标准点验证和标准点乘）的输入数据以及输出数据中的点均在仿射坐标系中；相对应的，所有 Jacobian 运算（包括 Jacobian 点验证和 Jacobian 点乘）的输入数据以及输出数据中的点均在 Jacobian 坐标系中。

15.4 功能描述

15.4.1 密钥长度模式

ESP8684 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。用户通过配置寄存器 ECC_KEY_LENGTH 来选定密钥长度模式，其与椭圆曲线的对应关系如表 15-2。

表 15-2. ECC 加速器密钥长度模式控制

ECC_KEY_LENGTH	对应椭圆曲线
1'b0	FIPS P-192
1'b1	FIPS P-256

[1] FIPS P-192/P-256 的曲线定义在 FIPS 186-3 中描述。

15.4.2 工作模式

ESP8684 ECC 硬件加速器共有 7 种工作模式，每种工作模式进行基于选定曲线的不同操作。用户通过配置寄存器 ECC_WORK_MODE 来选定工作模式，其与工作模式的对应关系如表15-3。

表 15-3. ECC 硬件加速器工作模式控制

ECC_WORK_MODE	对应模式	ECC_WORK_MODE	对应模式
3'd0	标准点乘	3'd4	Jacobian 点乘
3'd1	有限域除法	3'd5	**保留项，不可用**
3'd2	标准点验证	3'd6	Jacobian 点验证
3'd3	标准点验证 + 标准点乘	3'd7	标准点验证 + Jacobian 点乘

每个工作模式的具体计算和输入/输出数据请参照下述子章节。

15.4.2.1 标准点乘模式

该模式计算如下公式：

$$(Q_x, Q_y) = k \cdot (P_x, P_y)$$

其中，

- 输入数据：P_x，P_y，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 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。

15.4.2.2 有限域除法模式

该模式计算如下公式：

$$\text{Result} = P_y \cdot k^{-1}$$

其中，

- 输入数据：P_y，k 对应的内存块为 ECC_Mem_Py 和 ECC_Mem_k。
- 输出数据：Result 对应的内存块为 ECC_Mem_Py。

15.4.2.3 标准点验证模式

该模式用于计算点 (P_x, P_y) 是否在选定的椭圆曲线上。其中，

- 输入数据：P_x，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寄存器 ECC_VERIFICATION_RESULT 中。

15.4.2.4 标准点验证 + 标准点乘模式

该模式先计算点 (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：

$$(Q_x, Q_y) = k \cdot (P_x, P_y)$$

其中，

- 输入数据：P_x，P_y，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寄存器 ECC_VERIFICATION_RESULT 中；Q_x、Q_y 对应的内存块为 ECC_Mem_Px 和 ECC_Mem_Py。

15.4.2.5 Jacobian 点乘模式

该模式计算如下公式：

$$(Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

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, ECC_Mem_k。
- 输出数据：Q_x、Q_y 和 Q_z 对应的内存块为 ECC_Mem_Px, ECC_Mem_Py, ECC_Mem_k。

15.4.2.6 Jacobian 点验证模式

该模式用于计算点 (Q_x, Q_y, Q_z) 是否在选定的椭圆曲线上。其中，

- (Q_x, Q_y, Q_z) 为 Jacobian 表示的点。
- 输入数据：Q_x、Q_y 和 Q_z 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寄存器 ECC_VERIFICATION_RESULT 中。

15.4.2.7 标准点验证 + Jacobian 点乘模式

该模式先计算点 (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：

$$(Q_x, Q_y, Q_z) = k \cdot (P_x, P_y, 1)$$

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, ECC_Mem_k。
- 输出数据：点验证的结果存储在寄存器 ECC_VERIFICATION_RESULT 中；Q_x、Q_y 和 Q_z 对应的内存块为 ECC_Mem_Px，ECC_Mem_Py，ECC_Mem_k。

15.5 时钟与复位

ESP8684 ECC 硬件加速器模块仅有一个模块时钟 crypo_ecc_clk 和一个模块复位 crypto_ecc_rst 。在使用 ECC 硬件加速器之前，需要开启 ECC 时钟，关闭 ECC 复位。如何配置 ECC 时钟和复位，请参考章节 6 复位和时钟。

15.6 中断

ESP8684 ECC 硬件加速器共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的状态位共同产生，即任意一个其包含中断的状态位触发，该中断信号就会被触发。

ECC 硬件加速器的中断信号 ECC_INTR 包含以下中断：

- ECC_CALC_DONE_INT：ECC 硬件加速器运算完成即触发该中断。

中断 ECC_CALC_DONE_INT 由以下寄存器控制：

- 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。

15.7 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。
2. 根据 15.4 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 15.6 小节的描述，使能 ECC_CALC_DONE_INT 中断。
4. 置位寄存器 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 15.4 小节的描述，查看运算结果。

15.8 寄存器列表

本小节的所有地址均为相对于 ECC 硬件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 3 系统和存储器 中的表 3-3。

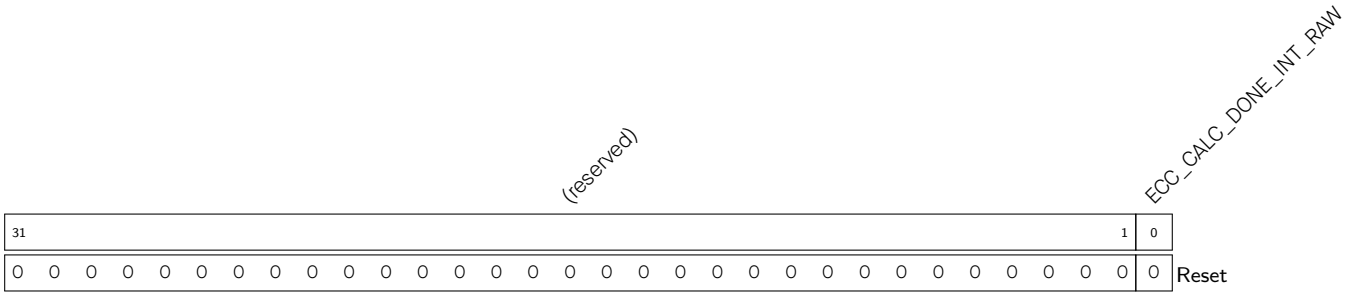
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

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	RO/WTC/SS
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

15.9 寄存器

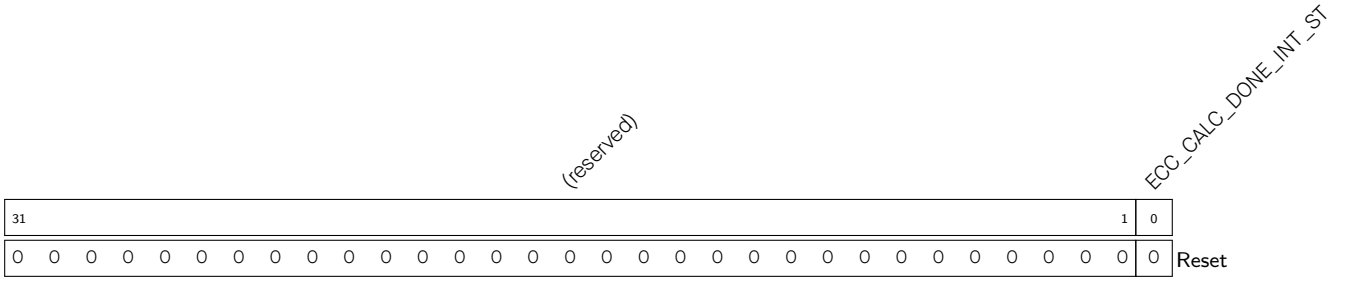
本小节的所有地址均为相对于 ECC 硬件加速器基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 15.1. ECC_MULT_INT_RAW_REG (0x000C)



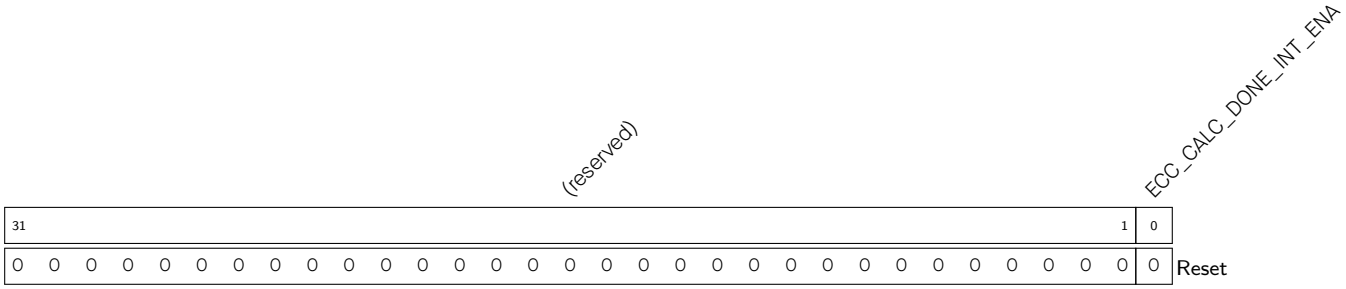
ECC_CALC_DONE_INT_RAW 存储 ECC_CALC_DONE_INT 中断的原始中断位。(RO/WTC/SS)

Register 15.2. ECC_MULT_INT_ST_REG (0x0010)



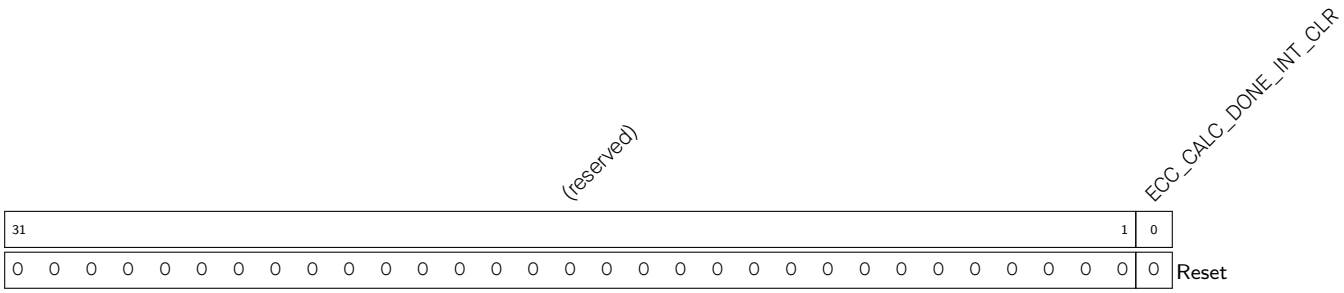
ECC_CALC_DONE_INT_ST 存储 ECC_CALC_DONE_INT 中断的屏蔽状态。(RO)

Register 15.3. ECC_MULT_INT_ENA_REG (0x0014)



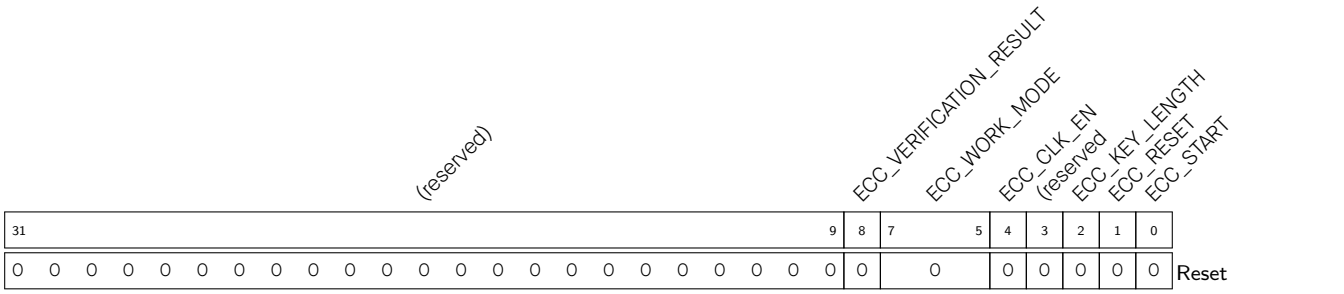
ECC_CALC_DONE_INT_ENA 置 1 使能 ECC_CALC_DONE_INT 中断。(R/W)

Register 15.4. ECC_MULT_INT_CLR_REG (0x0018)



ECC_CALC_DONE_INT_CLR 置 1 清除 ECC_CALC_DONE_INT 中断。(WT)

Register 15.5. ECC_MULT_CONF_REG (0x001C)



ECC_START 置 1 启动 ECC 加速器。此位运算结束后自动清 0。(R/W/SC)

ECC_RESET 置 1 复位 ECC 加速器。(WT)

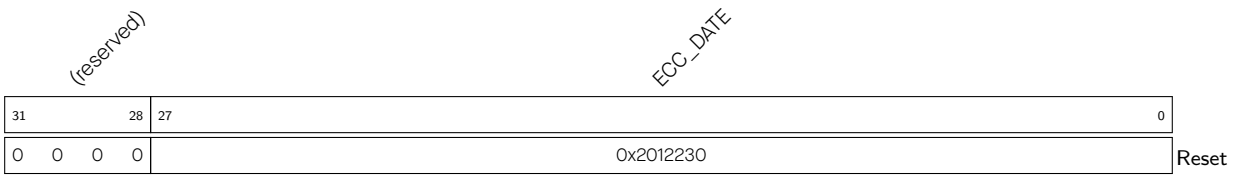
ECC_KEY_LENGTH 选择 ECC 加速的密钥长度。1'b0：P-192；1'b1：P-256。(R/W)

ECC_CLK_EN 置 1 强制打开寄存器门控。(R/W)

ECC_WORK_MODE 选择 ECC 加速器的工作模式。3'd0：标准点乘；3'd1：有限域除法；3'd2：标准点验证；3'd3：标准点验证 + 标准点乘；3'd4：Jacobian 点乘；3'd5：保留项，不可用；3'd6：Jacobian 点验证；3'd7：标准点验证 + Jacobian 点乘。(R/W)

ECC_VERIFICATION_RESULT 存储 ECC 加速器的验证结果，仅在运算完成时有效。(RO/SS)

Register 15.6. ECC_MULT_DATE_REG (0x00FC)



ECC_DATE ECC 加速器版本控制寄存器。(R/W)

16 SHA 加速器 (SHA)

16.1 概述

ESP8684 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。

16.2 主要特性

ESP8684 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
- 允许中断功能（仅限 DMA-SHA 工作模式）

16.3 工作模式简介

ESP8684 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。

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 16-1。

表 16-1. 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 16-2。

表 16-2. 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

16.4 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。

16.4.1 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。

16.4.1.1 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，则填充步骤见下：

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 中的 “5.1 Padding the Message” 章节。

16.4.1.2 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 中 “2.1 Glossary of Terms and Acronyms” 章节。

16.4.1.3 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。

16.4.2 哈希运算流程

在完成信息预处理后，ESP8684 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，ESP8684 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。

16.4.2.1 Typical SHA 模式下的运算流程

通常情况下，ESP8684 的 SHA 会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，ESP8684 SHA 加速器还支持“interleaved”运算，即在完成当前信息的所有运算前，允许插入其他运算任务。

- 在 Typical SHA 工作模式下，用户每计算完一个信息块后均可插入新的运算；
- 而在 DMA-SHA 工作模式下，用户必须等待本次 DMA 运算全部完成才可以插入新的运算。

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 16-2。
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，ESP8684 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。具体按照插入运行类型的不同，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。

16.4.2.2 DMA-SHA 模式下的运算流程

ESP8684 SHA 加速器在 DMA-SHA 工作模式下不支持在完成每个“信息块”运算后插入新的运算，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 *2 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 16-2。
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。

16.4.3 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 16-3：

表 16-3. 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] 信息摘要从左至右存放，第一个 word 存放在寄存器 SHA_H_0_REG 中，第二个 word 存放在寄存器 SHA_H_1_REG 中，以此类推。

16.4.4 中断

SHA 加速器在 DMA-SHA 工作模式下允许中断发生。用户可通过将 SHA_INT_ENA_REG 寄存器配置为 1 开启中断。如开启中断功能，SHA 加速器在完成运算时，中断发生。注意，该中断必须由软件将 SHA_INT_CLEAR_REG 寄存器置为 1 进行清除。由于 SHA 加速器在 Typical SHA 工作模式下的时间开销较小，因此不支持中断功能。

16.5 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 3 系统和存储器 中的表 3-3。

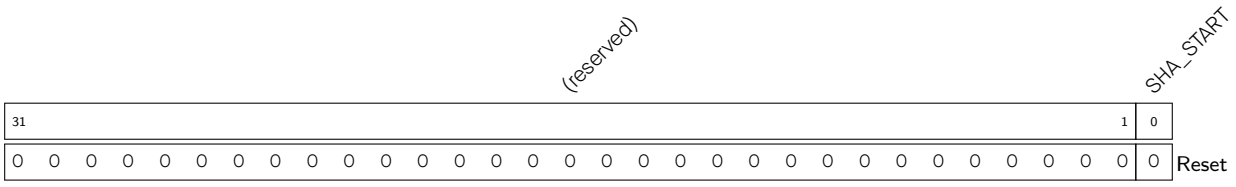
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	权限
控制与状态寄存器			
SHA_CONTINUE_REG	继续 SHA 运算（仅用于 Typical SHA 模式）	0x0014	WO
SHA_BUSY_REG	指示 SHA 加速器是否处于“忙碌”状态	0x0018	RO
SHA_DMA_START_REG	启动 SHA 加速器的 DMA-SHA 模式	0x001C	WO
SHA_START_REG	启动 SHA 加速器的 Typical SHA 模式	0x0010	WO
SHA_DMA_CONTINUE_REG	继续 SHA 运算（仅用于 DMA-SHA 模式）	0x0020	WO
SHA_INT_CLEAR_REG	DMA-SHA 中断清除寄存器	0x0024	WO
SHA_INT_ENA_REG	DMA-SHA 中断使能寄存器	0x0028	R/W
版本寄存器			
SHA_DATE_REG	版本控制寄存器	0x002C	R/W
配置寄存器			
SHA_MODE_REG	配置 SHA 加速器的运算标准	0x0000	R/W
数据寄存器			
SHA_DMA_BLOCK_NUM_REG	信息块个数寄存器(仅用于 DMA-SHA 工作模式)	0x000C	R/W
SHA_H_0_REG	哈希值	0x0040	R/W
SHA_H_1_REG	哈希值	0x0044	R/W
SHA_H_2_REG	哈希值	0x0048	R/W
SHA_H_3_REG	哈希值	0x004C	R/W
SHA_H_4_REG	哈希值	0x0050	R/W
SHA_H_5_REG	哈希值	0x0054	R/W
SHA_H_6_REG	哈希值	0x0058	R/W
SHA_H_7_REG	哈希值	0x005C	R/W
SHA_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

16.6 寄存器

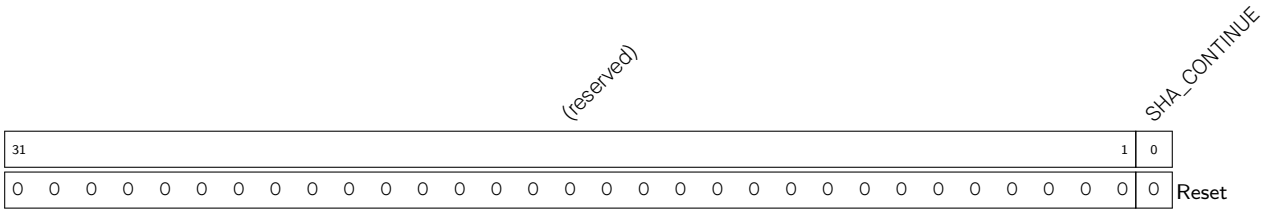
本小节的所有地址均为相对于 SHA 加速器基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 16.1. SHA_START_REG (0x0010)



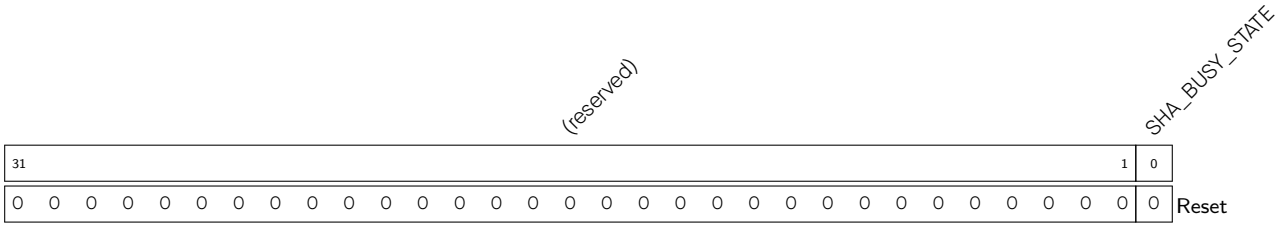
SHA_START 置 1 启动 SHA 加速器的 Typical SHA 模式。(WO)

Register 16.2. SHA_CONTINUE_REG (0x0014)



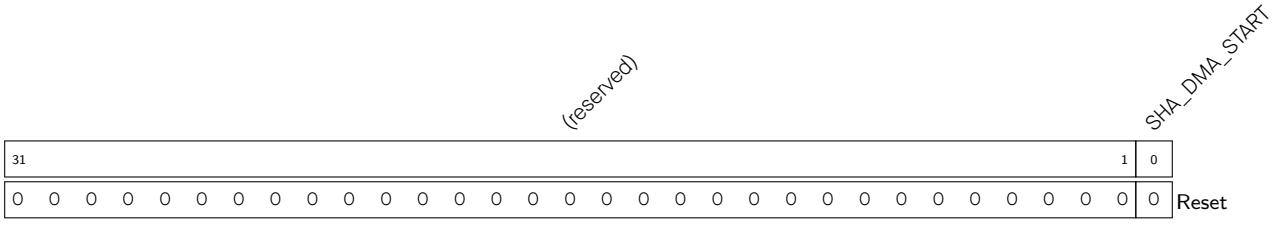
SHA_CONTINUE 置 1 继续 SHA 加速器的 Typical SHA 运算。(WO)

Register 16.3. SHA_BUSY_REG (0x0018)



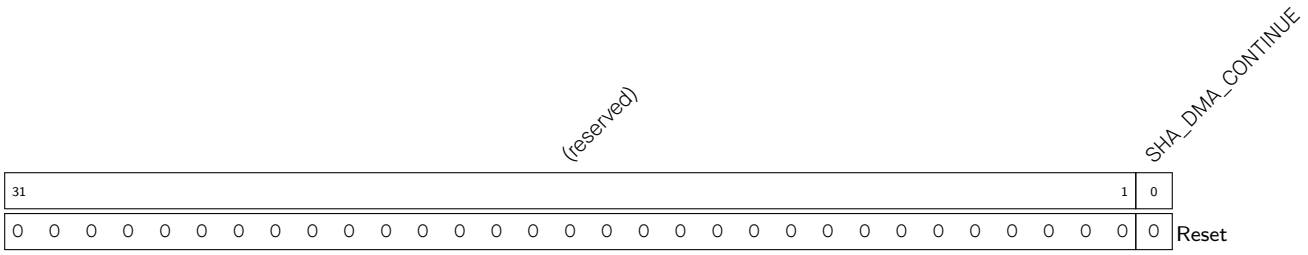
SHA_BUSY_STATE 指示 SHA 是否处于“忙碌”状态。(RO) 1'h0: 空闲 1'h1: 忙碌

Register 16.4. SHA_DMA_START_REG (0x001C)



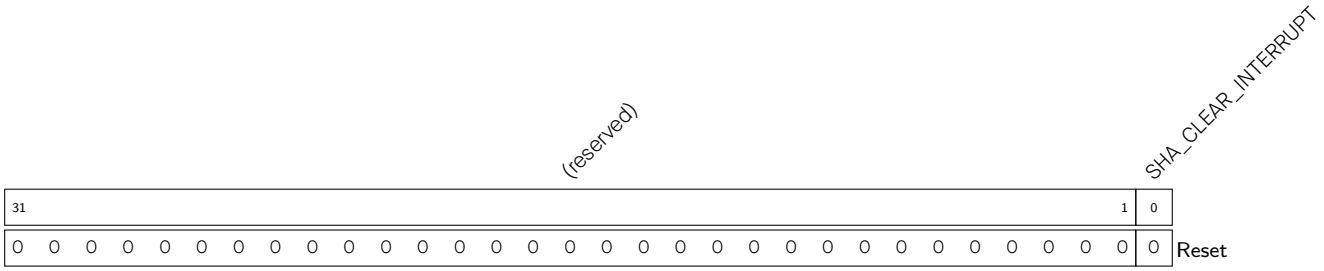
SHA_DMA_START 置 1 启动 SHA 加速器的 DMA-SHA 模式。(WO)

Register 16.5. SHA_DMA_CONTINUE_REG (0x0020)



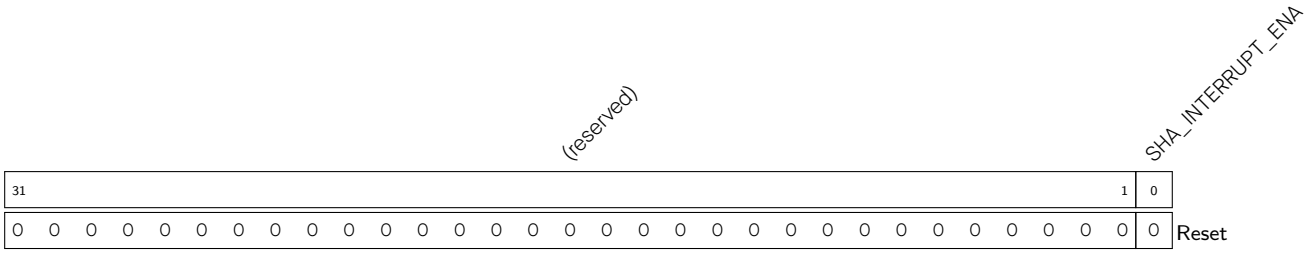
SHA_DMA_CONTINUE 置 1 继续 SHA 加速器的 DMA-SHA 运算。(WO)

Register 16.6. SHA_INT_CLEAR_REG (0x0024)



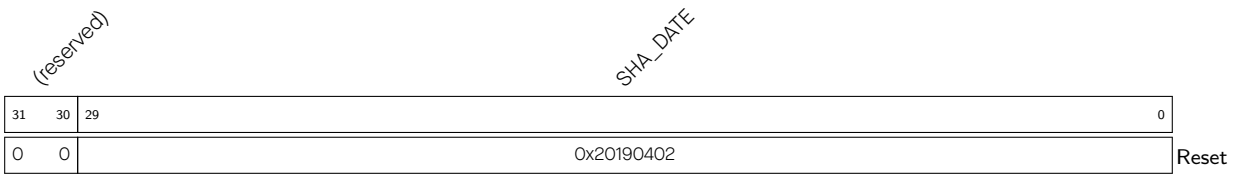
SHA_CLEAR_INTERRUPT 清除 DMA-SHA 中断。(WO)

Register 16.7. SHA_INT_ENA_REG (0x0028)



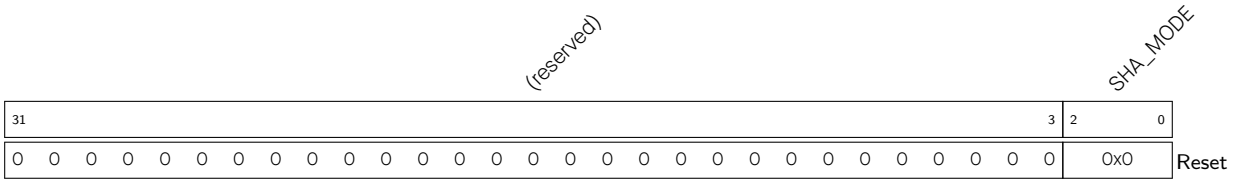
SHA_INTERRUPT_ENA 使能 DMA-SHA 中断。(R/W)

Register 16.8. SHA_DATE_REG (0x002C)



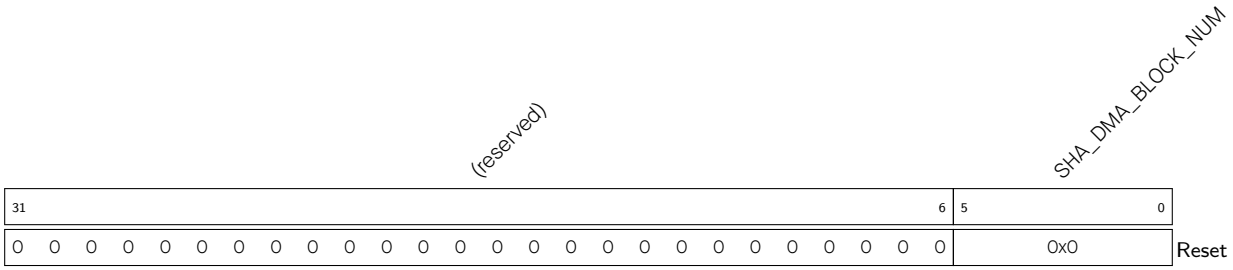
SHA_DATE 版本控制寄存器。(R/W)

Register 16.9. SHA_MODE_REG (0x0000)



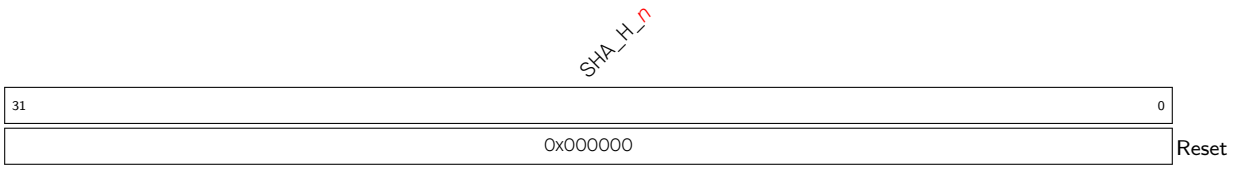
SHA_MODE 选择 SHA 加速器的运算标准，详见表 16-2。(R/W)

Register 16.10. SHA_DMA_BLOCK_NUM_REG (0x000C)



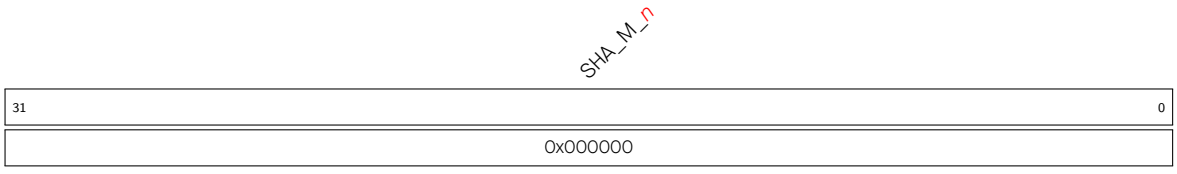
SHA_DMA_BLOCK_NUM 定义 DMA-SHA 工作模式下的信息块个数。(R/W)

Register 16.11. SHA_H_*n*_REG (*n*: 0-7) (0x0040+4**n*)



SHA_H_*n* 存储第 *n* 个 32 位哈希值。(R/W)

Register 16.12. SHA_M_*n*_REG (*n*: 0-15) (0x0080+4**n*)



SHA_M_*n* 存储第 *n* 个 32 位输入信息。(R/W)

17 片外存储器加密与解密 (XTS_AES)

17.1 概述

ESP8684 芯片集成了片外存储器加密与解密模块，采用符合 IEEE Std 1619-2007 指定的 XTS-AES 标准算法，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。

17.2 主要特性

该模块支持以下功能：

- 通用 XTS-AES 算法，符合 IEEE Std 1619-2007
- 手动加密过程需要软件参与
- 高速的自动解密过程，无需软件参与
- 寄存器配置、eFuse 参数、启动 (Boot) 模式共同决定加解密功能

17.3 模块结构

片外存储器加解密模块包含两个部分：手动加密 (Manual Encryption) 模块和自动解密 (Auto Decryption) 模块。结构图如图 17-1 所示。

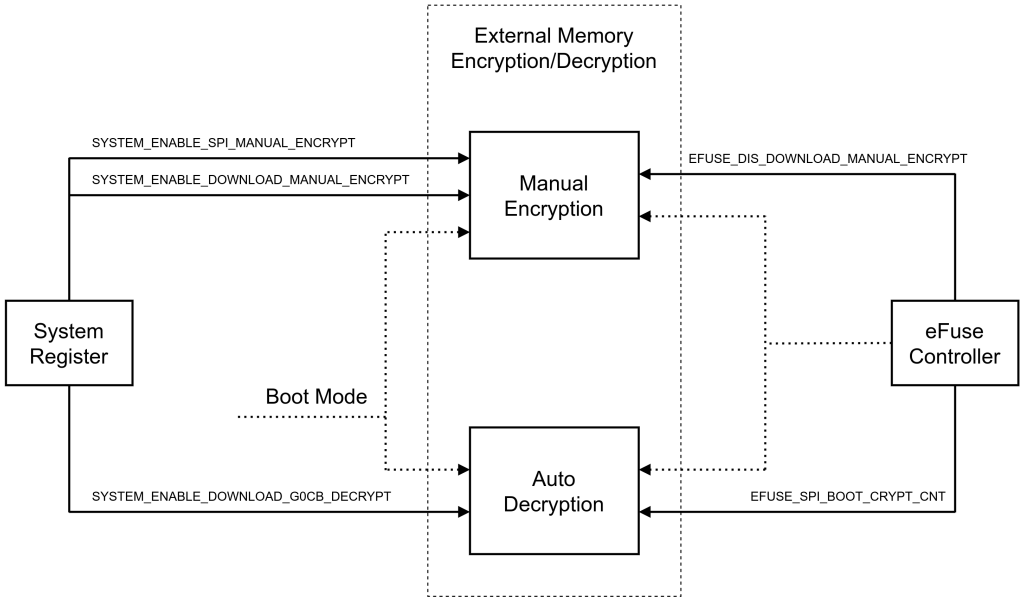


图 17-1. 片外存储器加解密结构

手动加密模块能够对指令/数据进行加密，指令/数据将以密文状态通过 SPI1 被写入片外 flash。

系统寄存器 (SYSREG) 外设中（请参见 13 *系统寄存器 (SYSTEM)*）SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 寄存器内的以下 4 个位与片外存储器加解密相关：

- SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT

- SYSTEM_ENABLE_DOWNLOAD_GCB_DECRYPT
- SYSTEM_ENABLE_DOWNLOAD_DB_ENCRYPT
- SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT

片外存储器加解密模块还会从外设 eFuse 控制器中获取 2 个参数：EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 和 EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_CNT。更多详细信息，请参考章节 *4 eFuse 控制器 (eFuse)*。

17.4 功能描述

17.4.1 XTS 算法

不论是手动加密，还是自动解密，该模块使用的都是 XTS 算法。根据算法特征，在具体实现中，使用 1024 位为一个数据单元 (data unit)，此处的“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。

17.4.2 密钥

在执行 XTS 运算时，手动加密模块和自动解密模块使用完全相同的密钥 Key 。密钥 Key 来自硬件 eFuse，且无法被用户访问获取。

密钥 Key 的长度为 256 位。Key 的值完全由 eFuse 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，现约定：

- Key_A：eFuse BLOCK3 中的低 128 位。
- Key_B：eFuse BLOCK3 中的高 128 位。

根据 eFuse 参数 EFUSE_XTS_KEY_LENGTH_256 的值存在两种可能性。不同情况下，Key 值可以由 Key_A，Key_B 的值唯一确定，如表 17-1 所示。

表 17-1. 根据 Key_A□Key_B 生成的 Key 值

EFUSE_XTS_KEY_LENGTH_256	Key	Key 长度（位）
1	$\{Key_B, Key_A\}$	256
0	$SHA-256(Key_A)$[1]	256

[1] "SHA-256" 表示 SHA-256 算法，参考 *16 SHA 加速器 (SHA)* 章节。

17.4.3 目标空间

目标空间指：片外存储器 (flash) 中存放首次加密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这两个参数唯一确定。这两个参数的定义如下：

- 目标大小：目标空间的大小 ($size$)，以字节为单位，即单次对多少数据进行加密，仅支持 16 和 32 字节。
- 目标基地址：目标空间的基地址 ($base\_addr$)，这是一个 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 $base\_addr\%size == 0$。

如某一次加密操作，要将 16 字节的指令数据加密后存放在片外 flash 中的地址段 0x130 ~ 0x13F 中，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/数据的加密，可以将整个加密过程拆分成多次进行，每

次都有各自的目标空间和相应参数。

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，目标空间等参数需要用户主动配置。

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。

17.4.4 数据写入

对于自动解密模块，数据的写入由硬件自动完成。对于手动加密模块，数据的写入需要用户主动配置。手动加密模块中包含 8 个寄存器 XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 7) 构成的寄存器块，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，手动加密模块不在乎明文来自什么地方，只注重密文将要存放在什么地方。考虑到明文和密文之间呈严格的对应关系，为了更好地描述明文如何存放在寄存器块中，现假设明文从一开始就放在目标空间中，并在加密完成后被密文替换。因此，接下来的描述不再出现 “明文” 这个概念，而用 “目标空间” 来代替。但请注意，在真正使用时，明文可以来自任何地方，但用户必须清晰知道明文如何存放在寄存器块中。

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 $offset = address\%32$，n = $\frac{offset}{4}$，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 32 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 17-2 所示。

表 17-2. 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	0x10	XTS_AES_PLAIN_4_REG
0x04	XTS_AES_PLAIN_1_REG	0x14	XTS_AES_PLAIN_5_REG
0x08	XTS_AES_PLAIN_2_REG	0x18	XTS_AES_PLAIN_6_REG
0x0C	XTS_AES_PLAIN_3_REG	0x1C	XTS_AES_PLAIN_7_REG

17.4.5 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 (SYSREG) 外设、eFuse 参数、boot 模式共同配置并使用这一模块。请注意，手动加密模块只能加密片外 flash。

当且仅当手动加密模块拥有工作权限时，才允许手动加密。手动加密模块是否拥有工作权限取决于：

- SPI Boot 模式下

  当寄存器 SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 SYSTEM_ENABLE_SPI_MANUAL_ENCRYPT 位为 1 时，手动加密模块拥有工作权限，否则无法工作。

- Download Boot 模式下

  当寄存器 SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的

SYSTEM_ENABLE_DOWNLOAD_MANUAL_ENCRYPT 位为 1，且 eFuse 参数 EFUSE_DIS_DOWNLOAD_MANUAL_ENCRYPT 为 0 时，手动加密模块拥有工作权限，否则无法工作。

说明：

- 即使 CPU 可以越过 cache，直接读片外存储器从而得到加密指令/数据，但用户还是绝对无法获取到密钥 Key。

17.4.6 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 (SYSREG) 外设、eFuse 参数、boot 模式共同配置并使用这一模块。

当且仅当自动解密模块拥有工作权限时，才允许自动解密。自动解密模块是否拥有工作权限取决于：

- SPI Boot 模式下

  当 eFuse 参数 EFUSE_SPI_BOOT_ENCRYPT_DECRYPT_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 SYSTEM_EXTERNAL_DEVICE_ENCRYPT_DECRYPT_CONTROL_REG 的 SYSTEM_ENABLE_DOWNLOAD_GOCB_DECRYPT 位为 1 时，自动解密模块拥有工作权限，否则无法工作。

说明：

- 当自动解密模块拥有工作权限时，如果 CPU 通过 cache 读取片外存储器中的指令/数据，自动解密将自动对读取到的密文进行解密以恢复指令/数据。解密的整个过程无需软件参与并且对 cache 是透明的。解密算法过程中密钥 Key 绝对无法被用户获取。
- 当自动解密模块没有工作权限时，自动解密模块不对片外存储器中的数据产生作用，无论是加密内容还是未加密内容，因此 CPU 通过 cache 读取到的是片外存储器中的原始内容。

17.5 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 17.4.3。
2. 将明文数据写入至寄存器块 XTS_AES_PLAIN_*n*_REG (*n*: 0 ~ 7)。更多详细信息，请参考章节 17.4.4。请根据您的实际需求写入寄存器，未使用的寄存器可为任意值。
3. 等待手动加密模块成为空闲状态。轮询寄存器 XTS_AES_STATE_REG 直到软件读取到 0。
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。然后，轮询寄存器 XTS_AES_STATE_REG，直到软件读取到 3。
7. 调用 SPI1，将密文写入片外 flash（请参阅章节 *20 SPI 控制器 (SPI)*）。
8. 向寄存器 XTS_AES_DESTROY_REG 写入 1，销毁密文。然后，寄存器 XTS_AES_STATE_REG 的值将为 0。

重复上述步骤，即可满足明文指令/数据的加密需求。

17.6 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 3 系统和存储器 中的表 3-3。

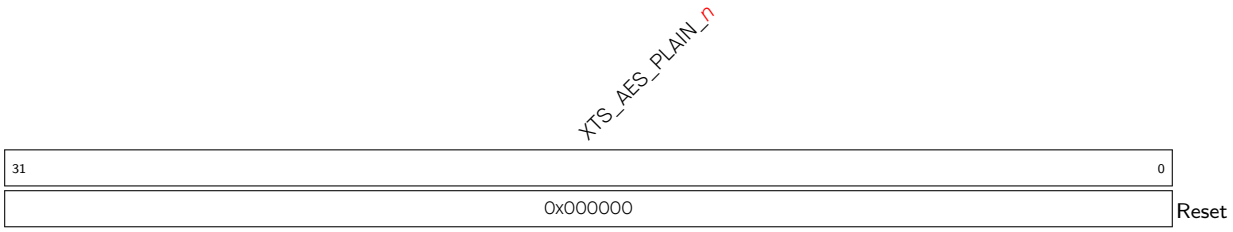
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

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	0x0000	读/写
XTS_AES_PLAIN_1_REG	明文寄存器 1	0x0004	读/写
XTS_AES_PLAIN_2_REG	明文寄存器 2	0x0008	读/写
XTS_AES_PLAIN_3_REG	明文寄存器 3	0x000C	读/写
XTS_AES_PLAIN_4_REG	明文寄存器 4	0x0010	读/写
XTS_AES_PLAIN_5_REG	明文寄存器 5	0x0014	读/写
XTS_AES_PLAIN_6_REG	明文寄存器 6	0x0018	读/写
XTS_AES_PLAIN_7_REG	明文寄存器 7	0x001C	读/写
配置寄存器			
XTS_AES_LINESIZE_REG	配置目标空间的大小	0x0040	读/写
XTS_AES_DESTINATION_REG	配置片外存储器的类型	0x0044	读/写
XTS_AES_PHYSICAL_ADDRESS_REG	物理地址	0x0048	读/写
控制/状态寄存器			
XTS_AES_TRIGGER_REG	启动 AES 算法	0x004C	只写
XTS_AES_RELEASE_REG	释放控制	0x0050	只写
XTS_AES_DESTROY_REG	销毁控制	0x0054	只写
XTS_AES_STATE_REG	状态寄存器	0x0058	只读
版本寄存器			
XTS_AES_DATE_REG	版本控制寄存器	0x005C	只读

17.7 寄存器

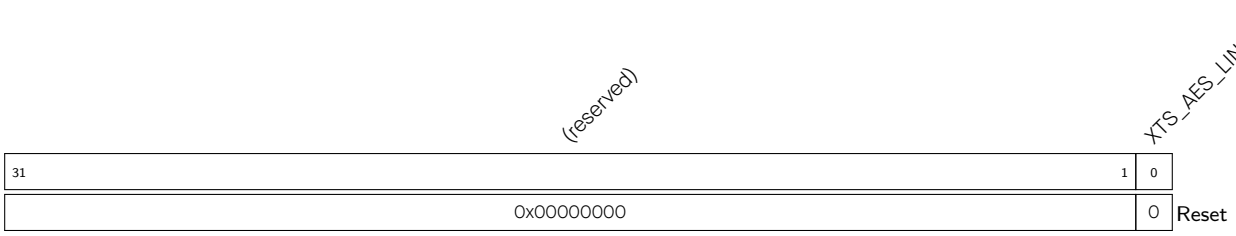
本小节的所有地址均为相对于片外存储器加密与解密基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 17.1. XTS_AES_PLAIN_*n*_REG (*n*: 0-7) (0x0000+4**n*)



XTS_AES_PLAIN_*n* 存储明文的第 *n* 个 32 位部分。（读/写）

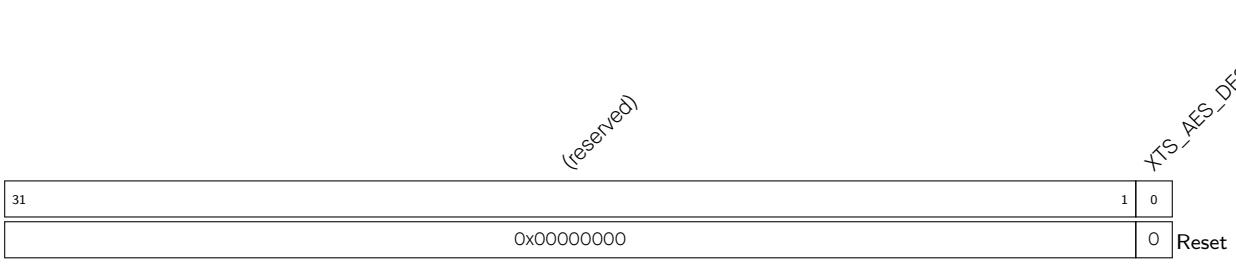
Register 17.2. XTS_AES_LINESIZE_REG (0x0040)



XTS_AES_LINESIZE 配置单次加密的数据大小。（读/写）

- 0：加密 16 字节；
- 1：加密 32 字节。

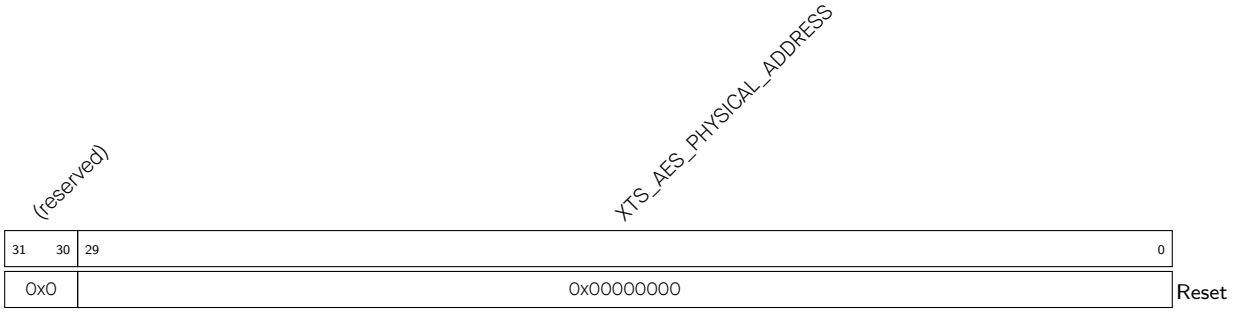
Register 17.3. XTS_AES_DESTINATION_REG (0x0044)



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。（读/写）

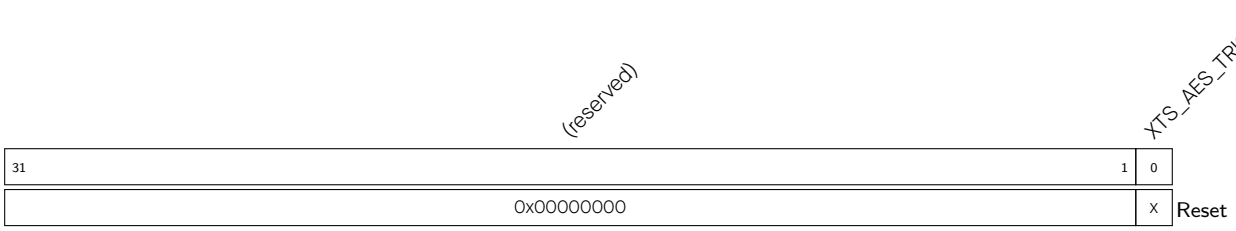
- 0：加密 flash；
- 1：加密片外 RAM。

Register 17.4. XTS_AES_PHYSICAL_ADDRESS_REG (0x0048)



XTS_AES_PHYSICAL_ADDRESS 物理地址（请注意，该值范围必须为 0x0000_0000 ~ 0x00FF_FFFF)。（读/写）

Register 17.5. XTS_AES_TRIGGER_REG (0x004C)

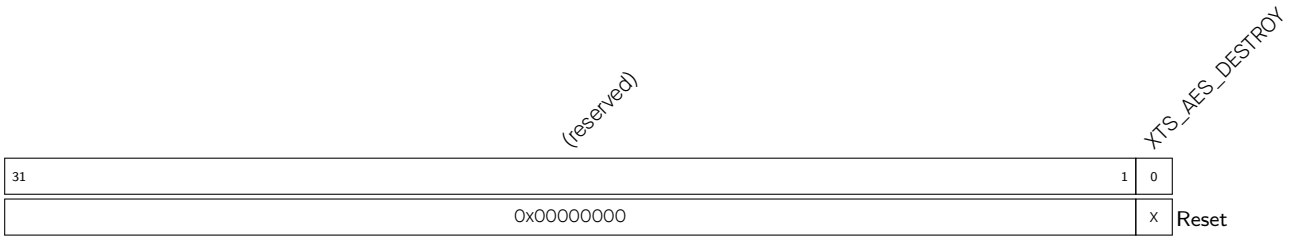


XTS_AES_TRIGGER 置位使能手动加密运算。(只写)

Register 17.6. XTS_AES_RELEASE_REG (0x0050)

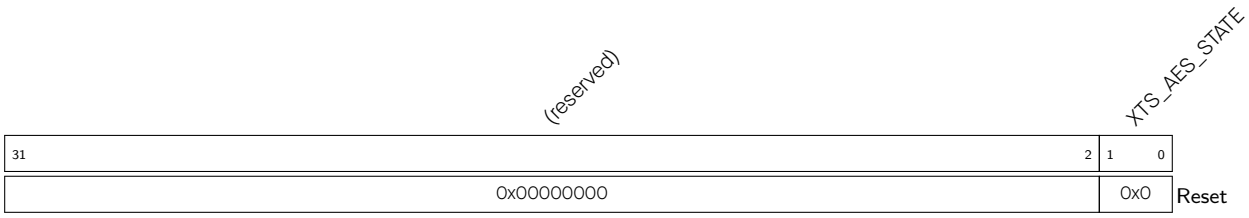
XTS_AES_RELEASE 置位使 SPI1 获取密文访问权限。(只写)

Register 17.7. XTS_AES_DESTROY_REG (0x0054)



XTS_AES_DESTROY 置位销毁加密结果。(只写)

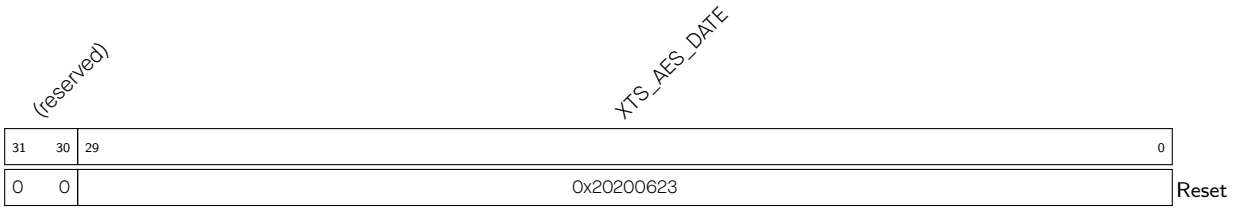
Register 17.8. XTS_AES_STATE_REG (0x0058)



XTS_AES_STATE 手动加密模块状态寄存器。(只读)

- 0x0 (XTS_AES_IDLE)：空闲；
- 0x1 (XTS_AES_BUSY)：计算中；
- 0x2 (XTS_AES_DONE)：计算完成，但手动加密结果数据对 SPI 不可见；
- 0x3 (XTS_AES_RELEASE)：手动加密结果对 SPI 可见。

Register 17.9. XTS_AES_DATE_REG (0x005C)



XTS_AES_DATE 版本控制寄存器。(读/写)

18 随机数发生器 (RNG)

18.1 概述

ESP8684 内置一个真随机数发生器，其生成的 32 位随机数可作为加密等操作的基础。

18.2 主要特性

ESP8684 的随机数发生器可通过物理过程而非算法生成真随机数，所有生成的随机数在特定范围内出现的概率完全一样。

18.3 功能描述

系统可以从随机数发生器的寄存器 RNG_DATA_REG 中读取随机数，每个读到的 32 位随机数都是真随机数，噪声源为系统中的**热噪声**和**异步时钟**。

- **热噪声**可以来自 SAR ADC 或高速 ADC 或两者兼有。当芯片的 SAR ADC 或高速 ADC 工作时，就会产生比特流，并通过异或 (XOR) 逻辑运算作为随机数种子进入随机数生成器。
- 内部快速 RC 振荡器时钟 RC_FAST_CLK（通常为 17.5 MHz，频率可调节）是一种**异步时钟源**，会产生电路亚稳态。这种亚稳态也可以作为随机数种子，进入随机数生成器，提高随机数发生器的熵值。

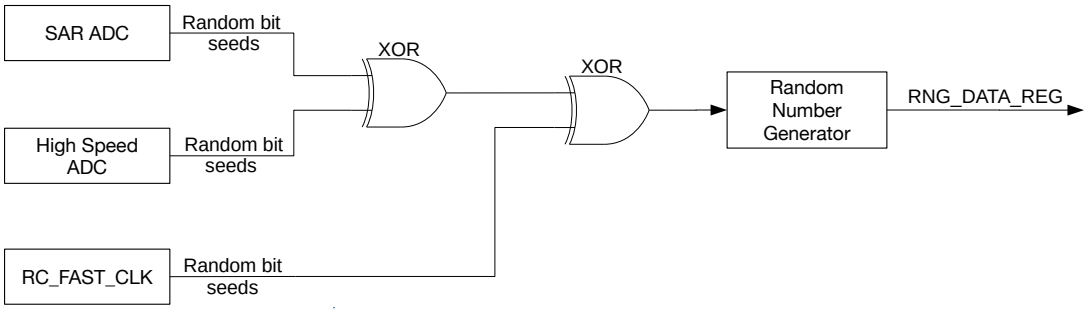


图 18-1. 噪声源

当 SAR ADC 打开时，每个 ADC 采样周期中，随机数发生器将获得 1 位的熵。由于 ADC 采样频率不超过 128 KHz，因此建议读取 RNG_DATA_REG 寄存器时的速率也不超过 128 kHz。

当高速 ADC 打开时，每个 APB 时钟周期（通常为 80 MHz）内，随机数发生器将获得 2 位的熵。因此，为了获得最大的熵值，建议读取 RNG_DATA_REG 寄存器时的速率不超过 5 MHz。

18.4 编程指南

在使用 ESP8684 的随机数生成器时，应该至少保证 SAR ADC 或高速 ADC[1] 打开，否则可能会导致产生伪随机数，应注意避免。其中，

- SAR ADC 受控于 DIG ADC 控制器。详见 23 片上传感器与模拟信号处理 章节。
- 高速 ADC 在 wireless 开启时自动打开。
- RC_FAST_CLK[2] 时钟在 Active 状态下始终打开，无需专门使能。

说明：

1. 注意，在 wireless 模块开启时，极端情况下高速 ADC 有读值饱和的可能，这会降低熵值。因此，建议在 wireless 模块开启时，同时通过 DIG ADC 控制器打开 SAR ADC 产生随机数。
2. RC_FAST_CLK 时钟仅可以提高随机数发生器的熵值。然而，为了保证随机数发生器可以获得足够大的，仍建议在使用随机数发生器时至少保证 SAR ADC 或高速 ADC 处于工作状态。

在使用随机数生成器时，请多次读取 RNG_DATA_REG 寄存器的值，直至获得足够多的随机数。在读取寄存器时，注意控制速率不要超过上方第 18.3 小节的介绍。

18.5 寄存器列表

请注意，下表中的地址都是相对于随机数发生器基地址的地址偏移量（相对地址），详见章节 3 系统和存储器 中的表 3-3。

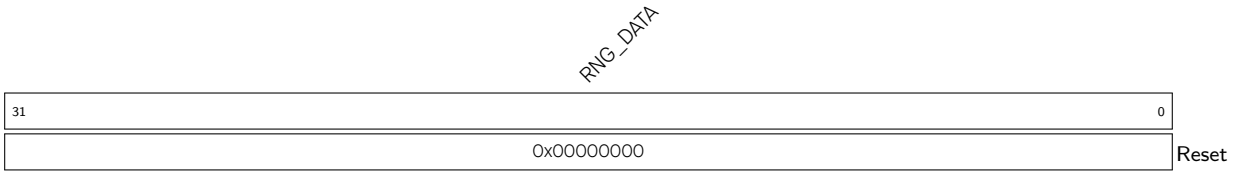
请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
RNG_DATA_REG	随机数数据	0x00B0	只读

18.6 寄存器

请注意，这里的地址都是相对于随机数发生器基地址的地址偏移量（相对地址），相见章节 3 系统和存储器 中的表 3-3。

Register 18.1. RNG_DATA_REG (0x00B0)



RNG_DATA 随机数来源。(只读)

19 UART 控制器 (UART)

19.1 概述

嵌入式应用通常要求一个简单的并且占用系统资源少的方法来传输数据。通用异步收发传输器 (UART) 即可以满足这些要求，它能够灵活地与外部设备进行全双工数据交换。芯片中有两个 UART 控制器可供使用，并且兼容不同的 UART 设备。另外，UART 还可以用作红外数据交换 (IrDA) 或 RS485 调制解调器。

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。芯片上的 UART 控制器支持多种字符长度和停止位。另外，控制器还支持软硬件流控。

19.2 主要特性

- 全双工异步通信
- 可配置波特率，最高 2.5 Mbaud
- 输入信号波特率自检功能
- 数据帧格式：
  - 一个 START 位
  - 数据位，长度为 5 ~ 8
  - 一个奇偶校验位
  - STOP 位，长度为 1、1.5 或 2
- AT_CMD 特殊字符检测
- 支持协议：RS485、IrDA
- UART 唤醒模式
- 软件流控和硬件流控
- 三个可预分频的时钟源：
  - 40 MHz PLL_F40M_CLK
  - 内置快速 RC 振荡器时钟 RC_FAST_CLK
  - 外部晶振时钟 XTAL_CLK
- 两个 UART 的发送 FIFO 以及接收 FIFO 共享 512 x 8-bit RAM

19.3 UART 架构

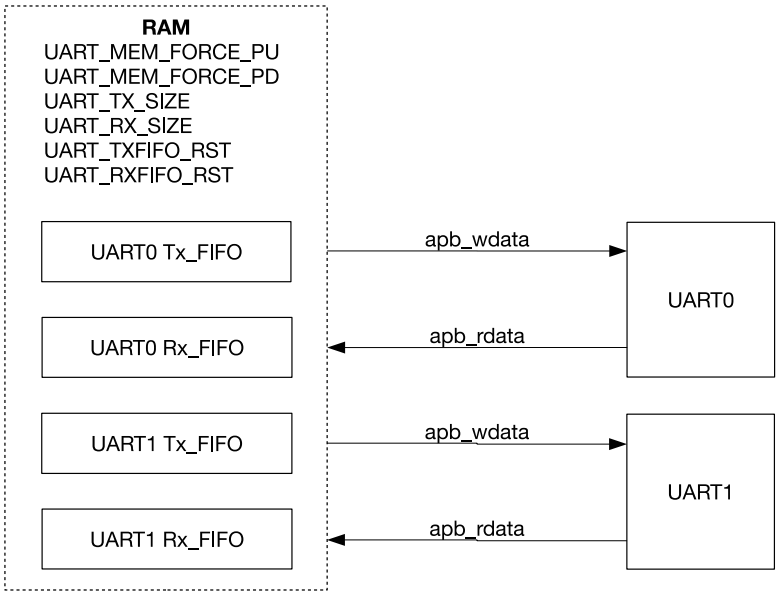


图 19-1. UART 架构概况

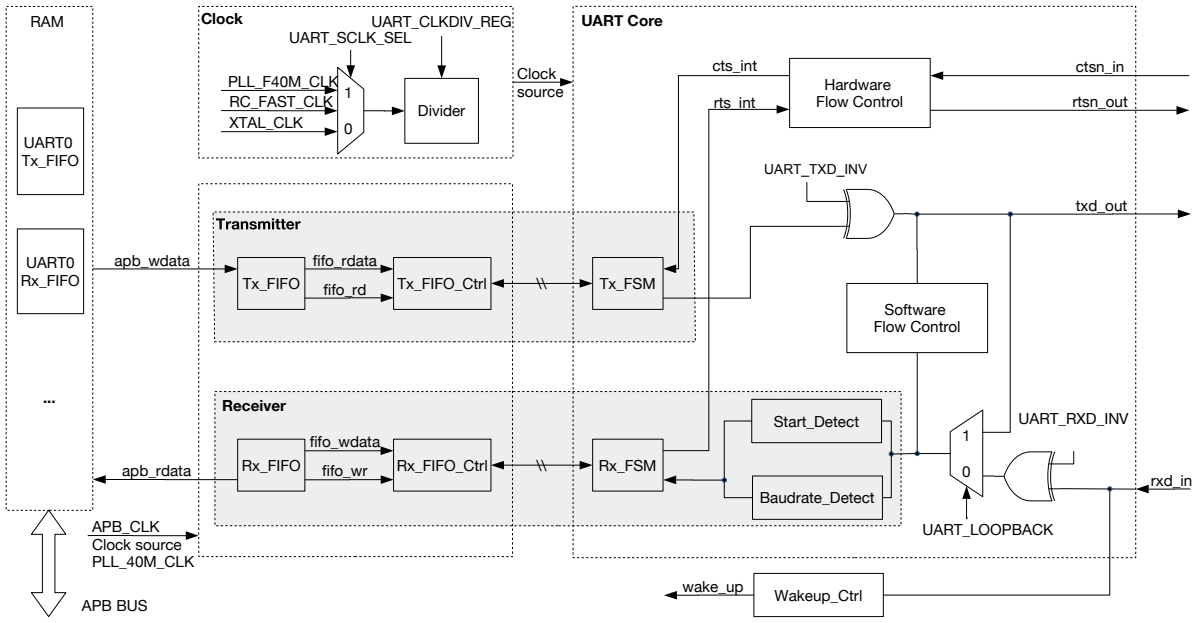


图 19-2. UART 基本架构图

图 19-2 为 UART 基本架构图。UART 模块工作在两个时钟域：APB_CLK 时钟域和 Core 时钟域。

APB_CLK 的时钟源是 PLL_F40M_CLK。

UART Core 有三个时钟源：40 MHz PLL_F40M_CLK、RC_FAST_CLK 以及晶振时钟 XTAL_CLK （详情请参考章节 6 复位和时钟）。可以通过配置 UART_SCLK_SEL 来选择时钟源。分频器用于对时钟源进行分频，然后产生时钟信号来驱动 UART Core 模块。UART_CLKDIV_REG 将分频系数分成两个部分：UART_CLKDIV 用于配置整数部分，UART_CLKDIV_FRAG 用于配置小数部分。

UART 控制器可以分为两个功能块：发送块和接收块。

发送块包含一个发送 FIFO 用于缓存待发送的数据。软件可以通过 APB 总线向 Tx_FIFO 写数据。Tx_FIFO_Ctrl 用于控制 Tx_FIFO 的读写过程，当 Tx_FIFO 非空时，Tx_FSM 通过 Tx_FIFO_Ctrl 读取数据，并将数据按照配置的帧格式转化成比特流。比特流输出信号 txd_out 可以通过配置 UART_TXD_INV 寄存器实现取反功能。

接收块包含一个接收 FIFO 用于缓存待处理的数据。输入比特流 rxd_in 可以输入到 UART 控制器。可以通过 UART_RXD_INV 寄存器实现取反。Baudrate_Detect 通过检测最小比特流输入信号的脉宽来测量输入信号的波特率。Start_Detect 用于检测数据的 START 位，当检测到 START 位之后，Rx_FSM 通过 Rx_FIFO_Ctrl 将帧解析后的数据存入 Rx_FIFO 中。软件可以通过 APB 总线读取 Rx_FIFO 中的数据。

HW_Flow_Ctrl 通过标准 UART RTS 和 CTS（rtsn_out 和 ctsn_in）流控信号来控制 rxd_in 和 txd_out 的数据流。SW_Flow_Ctrl 通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来进行数据流的控制。当 UART 处于 Light-sleep 状态（详情请参考章节 9 低功耗管理 (RTC_CNTL)）时，Wakeup_Ctrl 开始计算 rxd_in 的上升沿个数，当上升沿个数大于等于（UART_ACTIVE_THRESHOLD + 3）时产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒芯片。

19.4 功能描述

19.4.1 时钟与复位

UART 为异步外设。其寄存器配置模块与 TX/RX FIFO 工作在 APB_CLK 时钟域，而控制 UART 发送与接收的 Core 模块工作在 UART Core 时钟域。UART Core 有三个时钟源：PLL_F40M_CLK，RC_FAST_CLK 以及晶振时钟 XTAL_CLK，可通过配置 UART_SCLK_SEL 字段来选择时钟源。选择后的时钟源通过预分频器分频后进入 UART Core 模块。该预分频器支持小数分频，UART_SCLK_DIV_NUM 字段为整数部分，UART_SCLK_DIV_B 字段为小数部分的分子，UART_SCLK_DIV_A 为小数部分的分母。支持的分频范围为：1 ~ 256。

若分频之后的 Core 时钟频率还能满足生成波特率的需求，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。通常情况下，UART Core 模块时钟小于 APB_CLK 时钟，并且在满足 UART 波特率的情况下，UART Core 时钟分频系数可以配置到最大值。UART 也支持 UART Core 模块时钟大于 APB_CLK 时钟，此时，UART Core 模块时钟最大为 APB_CLK 的 3 倍。另外，UART 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 19.5。

对整个 UART 的复位，需要遵循如下配置流程：

- 将 SYSTEM_UART_MEM_CLK_EN 置 1 打开 UART RAM 时钟；
- 将 SYSTEM_UART*n*_CLK_EN 置 1 打开 UART*n* APB_CLK；
- 将 SYSTEM_UART*n*_RST 位清 0；
- 向寄存器 UART_RST_CORE 写 1；

- 向寄存器 SYSTEM_UART*n*_RST 写 1；
- 将寄存器 SYSTEM_UART*n*_RST 清 0；
- 将寄存器 UART_RST_CORE 清 0。

说明：
不推荐单独复位 UART APB 模块 (SYSTEM_UART*n*_RST) 或者 UART Core (UART_RST_CORE) 模块。

19.4.2 UART RAM

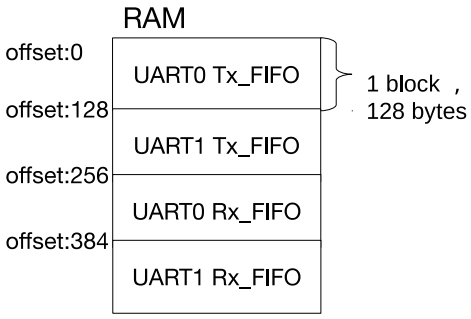


图 19-3. UART 共享 RAM 图

芯片中两个 UART 控制器共用 512×8-bit RAM 空间。如图 19-3 所示，RAM 以 block 为单位进行分配，1 block 为 128×8 bits，共 4 个 block。图 19-3 所示为默认情况下两个 UART 控制器的 Tx_FIFO 和 Rx_FIFO 占用 RAM 的情况。通过配置 UART_TX_SIZE 可以对 UART*n* 的 Tx_FIFO 以 1 block 为单位进行扩展，通过配置 UART_RX_SIZE 可以对 UART*n* 的 Rx_FIFO 以 1 block 为单位进行扩展：

- UART0 Tx_FIFO 可以从地址 0 扩展到整个 RAM 空间；
- UART1 Tx_FIFO 可以从地址 128 扩展到 RAM 的尾地址；
- UART0 Rx_FIFO 可以从地址 256 扩展到 RAM 的尾地址；
- UART1 Rx_FIFO 则不支持地址空间扩展。

需要注意的是所有 UART 的 FIFO 起始地址是固定的，因此前一个 UART 的 FIFO 空间向后扩展会占用后面 UART 的 FIFO 空间。比如，设置 UART0 的 UART_TX_SIZE 为 2，则 UART0 Tx_FIFO 的地址从 0 扩展到 255。这时，UART1 Tx_FIFO 的默认空间被占用，这时将不能使用 UART1 发送器功能。

当两个 UART 控制器都不工作时，可以通过置位 UART_MEM_FORCE_PD 来使 RAM 进入低功耗状态。

UART0 和 UART1 的 Tx_FIFO 可以通过置位 UART_TXFIFO_RST 来复位，UART0 和 UART1 的 Rx_FIFO 可以通过置位 UART_RXFIFO_RST 来复位。

对于 TX FIFO，可以通过 APB 总线向其写入数据，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。

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。

19.4.3 波特率产生与检测

19.4.3.1 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_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 40 MHz 的情况下，UART 能支持的最大波特率为 2.5 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 19-4 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 19-4 所示，输出脉冲相互交错，使得输出时序更加统一。

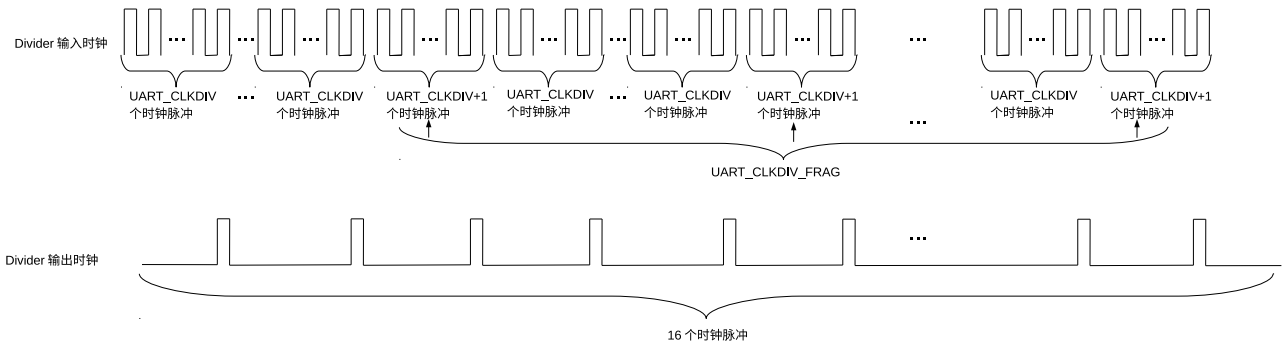


图 19-4. UART 控制器分频

为了支持 IrDA （详情见章节 19.4.7），IrDA 小数分频器会产生 16 × UART_CLKDIV_REG 分频的时钟用于 IrDA 数据传输。产生 IrDA 数据传输时钟的小数分频器原理与上述小数分频器一样，取 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。

19.4.3.2 波特率检测

置位 UART_AUTOBAUD_EN 可以开启 UART 波特率自检测功能。图 19-2 中的 Baudrate_Detect 可以滤除信号脉宽小于 UART_GLITCH_FILT 的噪声。

在 UART 双方进行通信之前，可以通过发送几个随机数据让具有波特率检测功能的数据接收方进行波特率分析。UART_LOWPULSE_MIN_CNT 存储了最小低电平脉冲宽度，UART_HIGHPULSE_MIN_CNT 存储了最小高电平脉

冲宽度，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。

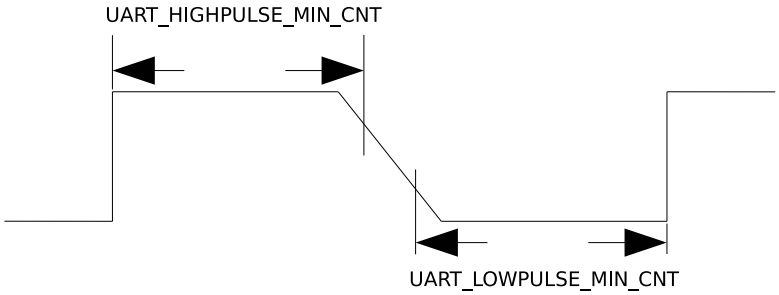


图 19-5. UART 信号下降沿较差时序图

波特率 B_{uart} 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：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_LOWPULSE\_MIN\_CNT} + \text{UART\_HIGHPULSE\_MIN\_CNT} + 2)/2}$$

   其中，f_{clk} 代表时钟频率。

2. 对于 UART 信号的下降沿信号比较差的情况，如图19-5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：

$$B_{\text{uart}} = \frac{f_{\text{clk}}}{(\text{UART\_NEGEDGE\_MIN\_CNT} + 1)/2}$$

19.4.4 UART 数据帧

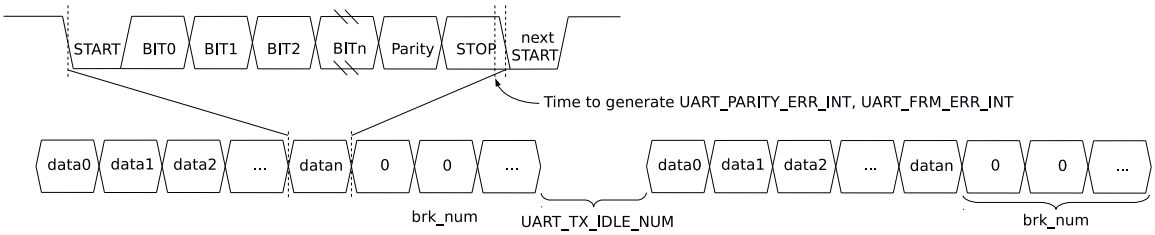


图 19-6. UART 数据帧结构

图 19-6 所示为基本数据帧格式，数据帧从 START 位开始以 STOP 位结束。START 占用 1 bit，STOP 位可以通过配置 UART_STOP_BIT_NUM 实现 1、1.5、2 位宽（RS485 模式下可增加转换延时，详见章节 19.4.6.2）。START 为低电平，STOP 为高电平。

数据位宽 (BIT0 ~ BITn) 为 5 ~ 8 bit，可以通过 UART_BIT_NUM 进行配置。当置位 UART_PARITY_EN 时，数据帧会在数据之后添加一位奇偶校验位。UART_PARITY 用于选择奇校验或是偶校验。当接收器检测到输入数据的

校验位错误时会产生 UART_PARITY_ERR_INT 中断，输入数据仍会存入 Rx_FIFO。当接收器检测到数据数据帧格式错误（即采样到的 STOP 位不为 1）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，RX 数据线若一直输出低电平，接收端会检测为终止状态，并触发 UART_BRK_DET_INT 中断表示终止状态已结束。

接收端通过 UART_RXFIFO_TOUT_INT 中断检测总线状态。接收端接收到至少一个字节数据后，总线处于空闲状态超过 UART_RX_TOUT_THRHD 位时间时，触发 UART_RXFIFO_TOUT_INT 中断。您可用此中断检测发送端是否已经发送所有数据。

19.4.5 AT_CMD 字符格式

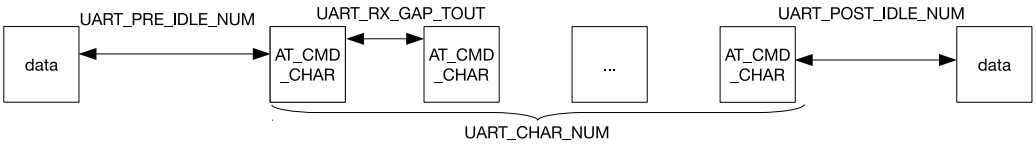


图 19-7. AT_CMD 字符格式

图 19-7 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 字符之间间隔至少 UART_POST_IDLE_NUM 个波特率周期。

19.4.6 RS485

UART 支持 RS485 协议，RS485 因使用差分信号传输数据，相比于 RS232 具有更远的传输距离及更高的传输速率。RS485 有两线半双工及四线全双工模式，UART 模块采用两线半双工模式，并支持侦听总线的功能。RS485 两线 multidrop 模式，最大可支持 32 个 slave。

19.4.6.1 驱动控制

如图 19-8 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 (D) 与一个接收器 (R)。当 UART 不作为发送器时，通过关闭驱动器 (D) 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 接收端通过接收器 (R) 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（软件仍能控制）。图 19-8 所示，DE 与 UART 的 dtrn_out 相连（详见 19.4.9.1 小节）。

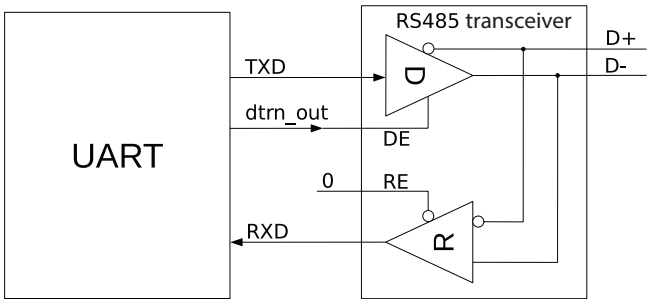


图 19-8. RS485 模式驱动控制结构图

19.4.6.2 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 协议推荐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 Start 位之前或在停止位之后增加一个波特率的延时。置位 UART_DLO_EN，在 Start 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。

19.4.6.3 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，配合外部 RS485 传输器的配置，UART 保持侦听传输总线。另外，默认情况下，不允许 UART 在接收数据时发送数据。置位 UART_RS485RXBY_TX_EN，允许在接收数据时发送数据。

UART 支持侦听 UART 发送的数据。UART 处于发送状态下，当侦听到 UART 发送的数据与 UART 接收的数据不同时，触发 UART_RS485_CLASH_INT 中断；侦听到发送的数据帧错误时，触发 UART_RS485_FRM_ERR_INT 中断；侦听到发送数据极性错误时，触发 UART_RS485_PARITY_ERR_INT 中断。

19.4.7 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 19-9 所示，IrDA 编码器将来自 UART 的非归零编码（NRZ）信号采用反向归零编码（RZI）并输出给外部驱动和红外 LED，用 3/16 Bit Time 的脉宽调制信号表示逻辑 “0”，用低电平表示逻辑 “1”。IrDA 解码器接收来自红外接收器的信号并输出为 UART 的 NRZ 编码。一般情况下，接收端信号空闲时为高电平，编码器输出极性与解码器输入极性相反。当检测到低脉冲表示接收到开始信号。

IrDA 使能时，一个比特被划分为 16 个时钟周期，在其第 9、10、11 个时钟周期中，当需要发送的比特为 0 时，IrDA 输出为高。

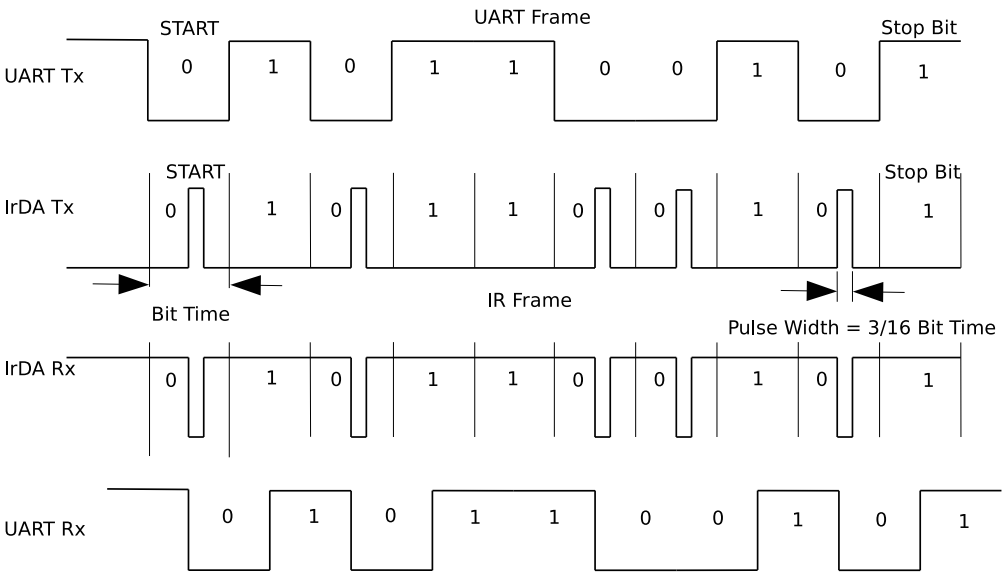


图 19-9. SIR 模式编解码时序图

IrDA 为半双工传输协议，不允许同时进行收发。如图 19-10所示，置位 UART_IRDA_EN 使能 IrDA 功能。置位 UART_IRDA_TX_EN （拉高）使能 IrDA 发送数据，这时不允许 IrDA 接收数据；复位 UART_IRDA_TX_EN（拉低）使能 IrDA 接收数据，这时不允许 IrDA 发送数据。

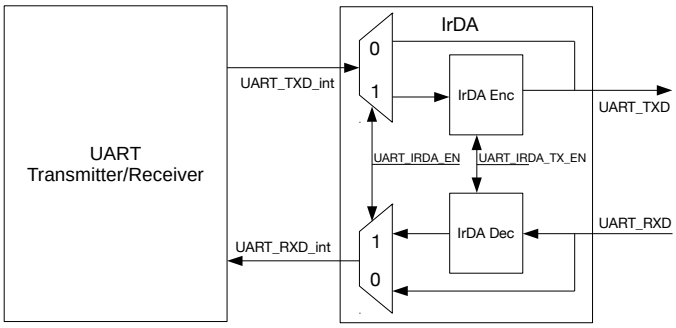


图 19-10. IrDA 编解码结构图

19.4.8 唤醒

UART0 和 UART1 支持唤醒功能。当 UART 处于 Light-sleep 状态时，Wakeup_Ctrl 开始计算 rxd_in 的上升沿个数，当上升沿个数大于等于（UART_ACTIVE_THRESHOLD + 3）时产生 wake_up 信号给 RTC 模块，由 RTC 来唤醒芯片。

19.4.9 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 dsrn_in 进行数据流控制。软件流控主要通过在发送数据流中插入特殊字符以及在接收数据流中检测特殊字符来实现数据流控功能。

19.4.9.1 硬件流控

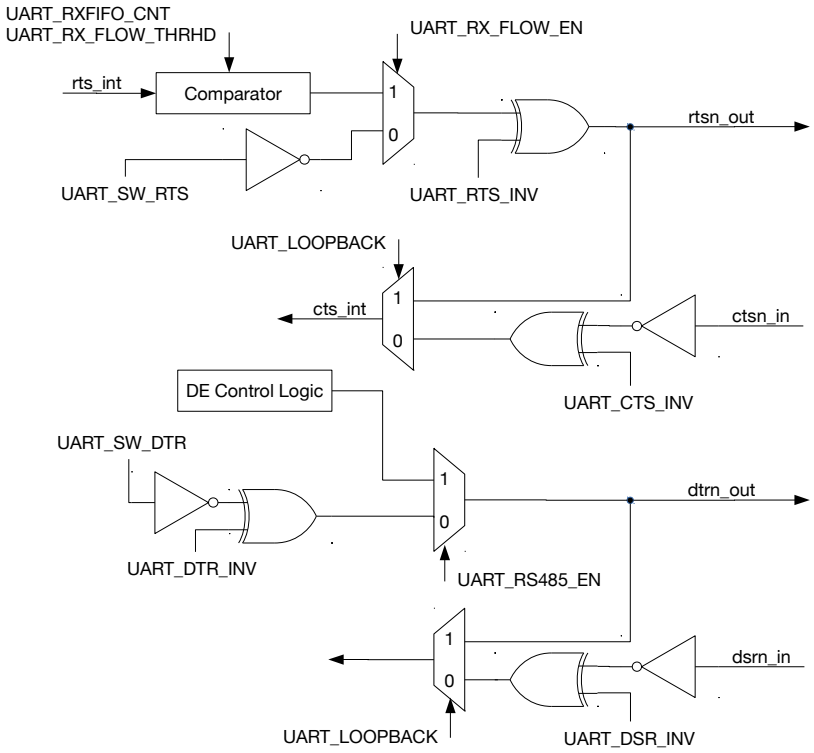


图 19-11. 硬件流控图

图 19-11 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 19-12 为两个 UART 之间硬件流控信号连接图。记 ESP8684 UART 为 IUO，External UART 为 EUO，下文将使用这两个标记来区分两个 UART。输出信号 rtsn_out （IUO）为低电平表示允许对方（EUO）发送数据，rtsn_out（IUO）为高电平表示通知对方（EUO）中止数据发送直到 rtsn_out （IUO）恢复低电平。rtsn_out 输出信号的控制有两种方式。

- 软件控制：将 UART_RX_FLOW_EN 置 0 进入该模式。该模式下通过软件配置 UART_SW_RTS 改变 rtsn_out 的电平。
- 硬件控制：将 UART_RX_FLOW_EN 置 1 进入该模式。该模式下硬件会当 Rx_FIFO 中的数据大于 UART_RX_FLOW_THRHD 时拉高 rtsn_out 的电平。

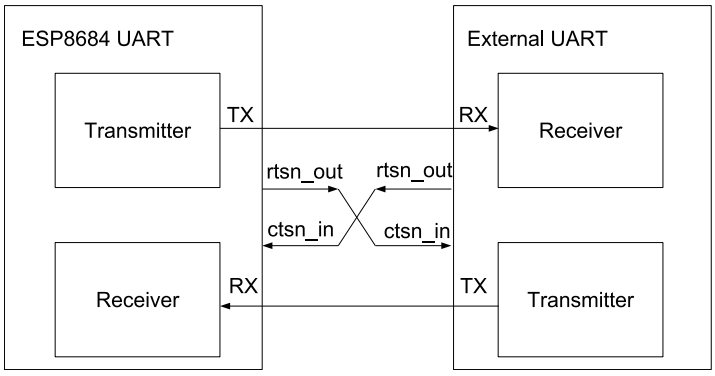


图 19-12. 硬件流控信号连接图

输入信号 ctsn_in（IUO）为低电平表示允许发送端（IUO）发送数据；ctsn_in（IUO）为高电平表示禁止发送端（IUO）发送数据。当 UART 检测到输入信号 ctsn_in （IUO）的沿变化时会产生 UART_CTS_CHG_INT 中断。

UART 发送设备（IUO）输出信号 dtrn_out 为高电平表示发送数据已经准备完毕，处于可用状态。dtrn_out 通过配置寄存器 UART_SW_DTR 产生。UART 接收设备（IUO）在检测到输入信号 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（EUO）处于可用状态。

对于 RS485 两线 multidrop 系统，使用 dtrn_out 来收发转换。置位 UART_RS485_EN 使能 RS485 功能，dtrn_out 由硬件（即图 19-12 中的 DE control logic）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。

19.4.9.2 软件流控

软件流控不使用硬件的 ctsn_in 和 rtsn_out 信号，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。

19.4.10 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：当接收器接收一个字节的时间大于 UART_RX_TOUT_THRHD 时触发此中断。
- UART_BRK_DET_INT：当接收器在停止位之后检测到一个 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。

19.5 编程流程

19.5.1 寄存器类型

UART 的所有寄存器都处于 APB_CLK 时钟域。对于软件可配置的寄存器，根据其作用的时钟域及同步处理，将其分为三类：立即寄存器，同步寄存器及静态寄存器。立即寄存器作用于 APB_CLK 时钟域，通过 APB 总线配置后立即生效；同步寄存器作用于 Core 时钟域，这些寄存器需要经过同步之后才能生效；静态寄存器也作用于 Core 时钟域，但这些寄存器不会在 UART 工作过程中动态修改。静态寄存器没有同步处理，软件可以通过开关 UART TX/RX Core 时钟的方式保证 UART Core 时钟域采样到正确的配置信息。

19.5.1.1 同步寄存器

为了确保作用于 UART Core 时钟域的寄存器被正确采样，他们中大多数都做了跨时钟域处理，这部分即为同步寄存器。同步寄存器如表19-1所示。对这些寄存器的配置流程如下：

- 将 UART_UPDATE_CTRL 清 0 使能寄存器同步功能；
- 等待 UART_REG_UPDATE 为 0，确保上一次同步已经完成；
- 配置同步寄存器；
- 向 UART_REG_UPDATE 写 1，将配置的值同步到 Core 时钟域。

表 19-1. UART*n*同步寄存器

寄存器	域名
UART_CLKDIV_REG	UART_CLKDIV_FRAG[3:0]

见下页

表 19-1 – 接上页

寄存器	域名
	UART_CLKDIV[11:0]
UART_CONF0_REG	UART_AUTOBAUD_EN
	UART_ERR_WR_MASK
	UART_TXD_INV
	UART_RXD_INV
	UART_IRDA_EN
	UART_TX_FLOW_EN
	UART_LOOPBACK
	UART_IRDA_RX_INV
	UART_IRDA_TX_EN
	UART_IRDA_WCTL
	UART_IRDA_TX_EN
	UART_IRDA_DPLX
	UART_STOP_BIT_NUM
	UART_BIT_NUM
	UART_PARITY_EN
	UART_PARITY
UART_FLOW_CONF_REG	UART_SEND_XOFF
	UART_SEND_XON
	UART_FORCE_XOFF
	UART_FORCE_XON
	UART_XONOFF_DEL
	UART_SW_FLOW_CON_EN
UART_TXBRK_CONF_REG	UART_RS485_TX_DLY_NUM[3:0]
	UART_RS485_RX_DLY_NUM
	UART_RS485RXBY_TX_EN
	UART_RS485TX_RX_EN
	UART_DL1_EN
	UART_DL0_EN
	UART_RS485_EN

19.5.1.2 静态寄存器

在作用于 UART Core 时钟域的寄存器中，有一部分寄存器不会在 UART 工作过程中动态修改，被认为是静态的，称为静态寄存器。静态寄存器没有做跨时钟域处理。静态寄存器的配置一定是 UART TX/RX 停止工作阶段，因此可以通过关闭 UART TX/RX 时钟的方式，保证配置寄存器的亚稳态不会被采样到。当 UART TX/RX 时钟打开时，软件配置的值已经稳定，从而确保配置的值被正确采样。表 19-2 列出了这些寄存器。对这些寄存器的配置流程如下：

- 根据将要停止工作的模块为 UART TX 还是 RX，将 UART_TX_SCLK_EN 或 UART_RX_SCLK_EN 清 0 关闭 UART TX 或 RX 时钟；
- 配置静态寄存器；
- 向 UART_TX_SCLK_EN 或 UART_RX_SCLK_EN 写 1 打开 UART TX 或 RX 时钟。

表 19-2. UART*n*静态寄存器

寄存器	域名
UART_RX_FILT_REG	UART_GLITCH_FILT_EN
	UART_GLITCH_FILT[7:0]
UART_SLEEP_CONF_REG	UART_ACTIVE_THRESHOLD[9:0]
UART_SWFC_CONF0_REG	UART_XOFF_CHAR[7:0]
UART_SWFC_CONF1_REG	UART_XON_CHAR[7:0]
UART_IDLE_CONF_REG	UART_TX_IDLE_NUM[9:0]
UART_AT_CMD_PRECNT_REG	UART_PRE_IDLE_NUM[15:0]
UART_AT_CMD_POSTCNT_REG	UART_POST_IDLE_NUM[15:0]
UART_AT_CMD_GAPTOUT_REG	UART_RX_GAP_TOUT[15:0]
UART_AT_CMD_CHAR_REG	UART_CHAR_NUM[7:0]
	UART_AT_CMD_CHAR[7:0]

19.5.1.3 立即寄存器

除表19-1与19-2 外的所有软件可配置寄存器作用于 APB_CLK 时钟域，即为立即寄存器，例如，中断及 FIFO 配置寄存器等。

19.5.2 具体步骤

图 19-13 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。

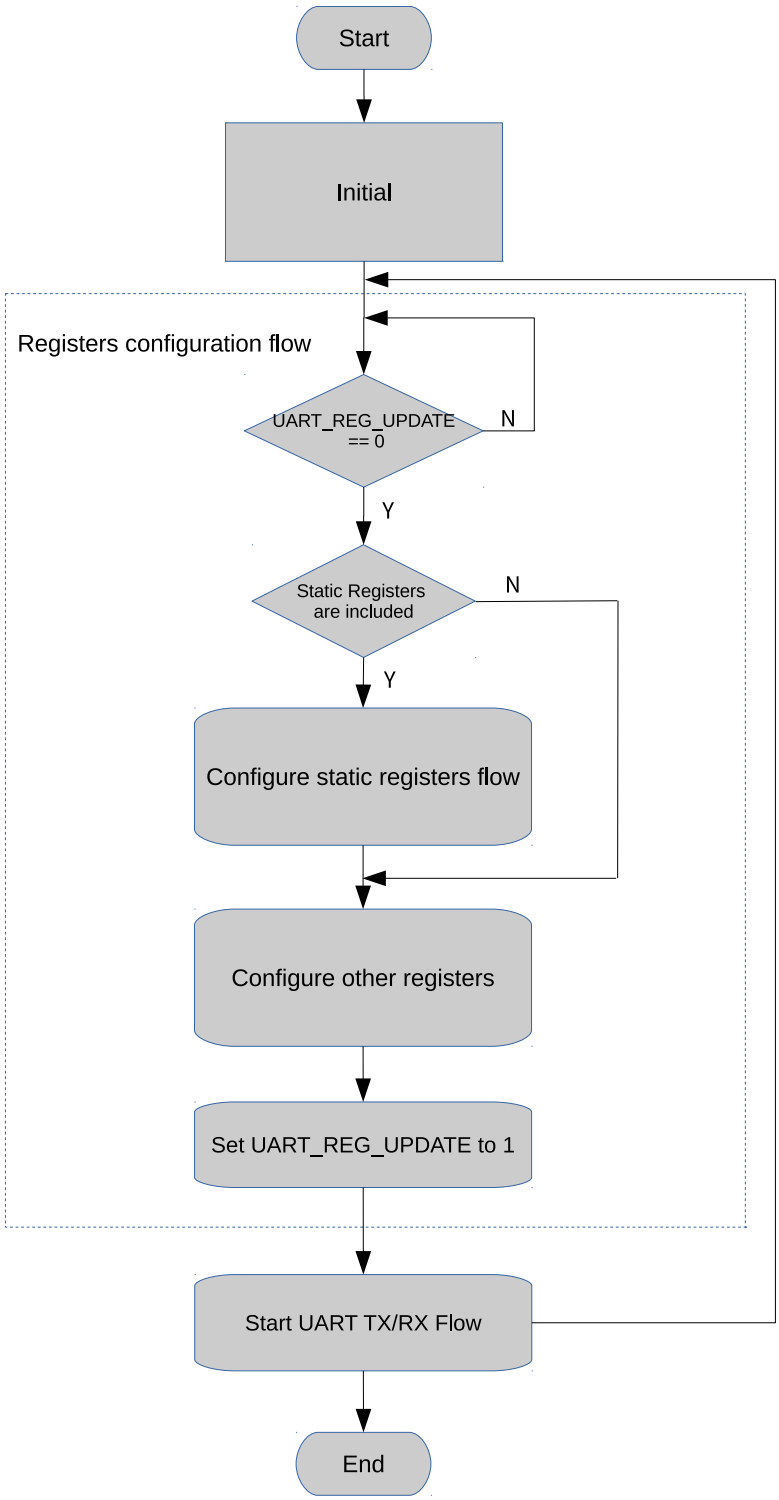


图 19-13. UART 编程流程

19.5.2.1 UART*n* 模块初始化

UART*n* 模块初始化流程如下：

- 将 SYSTEM_UART_MEM_CLK_EN 置 1 打开 UART RAM 时钟；
- 将 SYSTEM_UART*n*_CLK_EN 置 1 打开 UART*n* APB_CLK；
- 将寄存器 SYSTEM_UART*n*_RST 清 0；

- 向寄存器 UART_RST_CORE 写 1；
- 向寄存器 SYSTEM_UART*n*_RST 写 1；
- 将寄存器 SYSTEM_UART*n*_RST 清 0；
- 将寄存器 UART_RST_CORE 清 0；
- 将 UART_UPDATE_CTRL 清 0 使能寄存器同步功能。

19.5.2.2 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；
- 如果配置寄存器中包含静态寄存器，配置流程参考章节 19.5.1.2 完成配置；
- 配置 UART_SCLK_SEL 选择时钟源；
- 配置 UART_SCLK_DIV_NUM、UART_SCLK_DIV_A、UART_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。

19.5.2.3 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，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_TXFIFO_FULL_INT，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。

19.6 寄存器列表

19.6.1 UART 寄存器列表

本小节的所有地址均为相对地址 (相对于 UART 控制器 基地址的地址偏移量)，具体基地址请见章节 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

名称	描述	地址	访问
FIFO 配置			
UART_FIFO_REG	FIFO 数据寄存器	0x0000	RO
UART_MEM_CONF_REG	UART 阈值和分配配置	0x0060	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_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_REG	配置寄存器 0	0x0020	R/W
UART_CONF1_REG	配置寄存器 1	0x0024	R/W
UART_FLOW_CONF_REG	软件流控配置	0x0034	varies
UART_SLEEP_CONF_REG	睡眠模式配置	0x0038	R/W
UART_SWFC_CONF0_REG	软件流控字符配置寄存器 0	0x003C	R/W
UART_SWFC_CONF1_REG	软件流控字符配置寄存器 1	0x0040	R/W
UART_TXBRK_CONF_REG	TX 断开字符配置	0x0044	R/W
UART_IDLE_CONF_REG	帧结束空闲配置	0x0048	R/W
UART_RS485_CONF_REG	RS485 模式配置	0x004C	R/W
UART_CLK_CONF_REG	UART core 时钟配置	0x0078	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	0x0064	RO
UART_MEM_RX_STATUS_REG	RX FIFO 写入、读取偏移地址	0x0068	RO
UART_FSM_STATUS_REG	UART 发送和接收状态	0x006C	RO
自动波特率检测寄存器			
UART_LOWPULSE_REG	自动波特率检测最短低电平脉冲持续时间寄存器	0x0028	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	0x002C	RO
UART_RXD_CNT_REG	自动波特率检测沿变化计数寄存器	0x0030	RO
UART_POSPULSE_REG	自动波特率检测高电平脉冲寄存器	0x0070	RO
UART_NEGPULSE_REG	自动波特率检测低电平脉冲寄存器	0x0074	RO
AT 转义序列检测配置			
UART_AT_CMD_PRECNT_REG	序列发送前的时序配置	0x0050	R/W

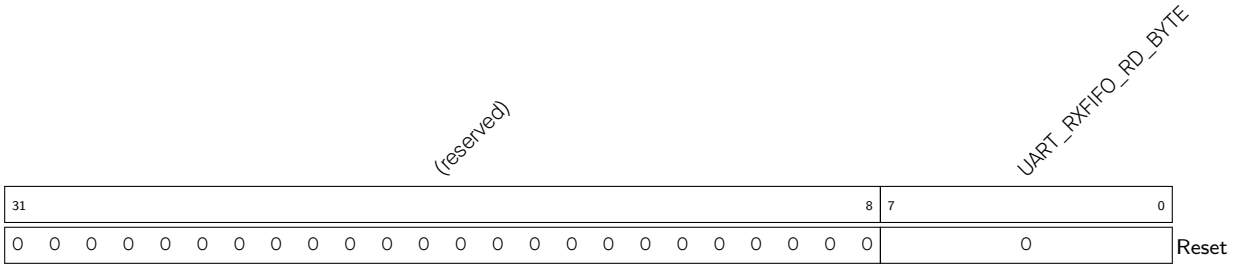
名称	描述	地址	访问
UART_AT_CMD_POSTCNT_REG	序列发送后的时序配置	0x0054	R/W
UART_AT_CMD_GAPTOUT_REG	超时配置	0x0058	R/W
UART_AT_CMD_CHAR_REG	AT 转义序列检测配置	0x005C	R/W
版本寄存器			
UART_DATE_REG	UART 版本控制寄存器	0x007C	R/W
UART_ID_REG	UART ID 寄存器	0x0080	varies

19.7 寄存器

19.7.1 UART 寄存器

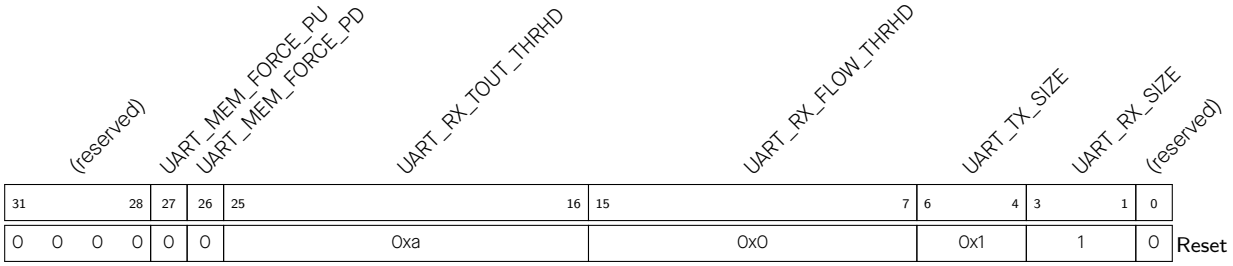
本小节的所有地址均为相对地址 (相对于 UART 控制器 基地址的地址偏移量)，具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 19.1. UART_FIFO_REG (0x0000)



UART_RXFIFO_RD_BYTE UART*n* 通过此字段访问 FIFO。(RO)

Register 19.2. UART_MEM_CONF_REG (0x0060)



UART_RX_SIZE 配置 RAM 分配给 RX FIFO 的空间大小。默认为 128 字节。(R/W)

UART_TX_SIZE 配置 RAM 分配给 TX FIFO 的空间大小。默认为 128 字节。(R/W)

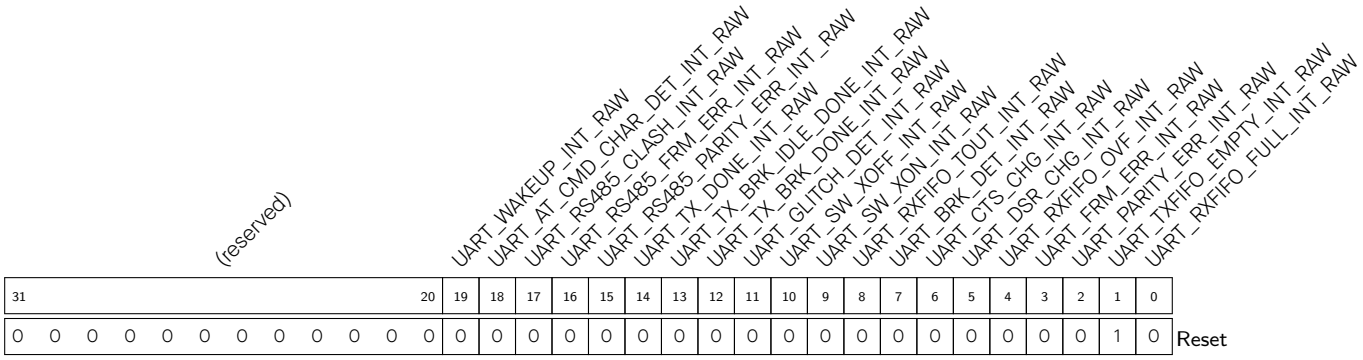
UART_RX_FLOW_THRHD 配置使用硬件流控时接收数据的最大值。(R/W)

UART_RX_TOUT_THRHD 配置接收器接收一个字节所需时间的阈值，单位是比特时间（即传输一个比特所需的时间）。接收器接收一个字节所需时间超过阈值且 UART_RX_TOUT_EN 置 1 时触发 UART_RXFIFO_TOUT_INT 中断。(R/W)

UART_MEM_FORCE_PD 置位此位强制关闭 UART RAM。(R/W)

UART_MEM_FORCE_PU 置位此位强制开启 UART RAM。(R/W)

Register 19.3. UART_INT_RAW_REG (0x0004)



UART_RXFIFO_FULL_INT_RAW 接收器接收数据多于 UART_RXFIFO_FULL_THRHD 的值时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_TXFIFO_EMPTY_INT_RAW TX FIFO 中的数据少于 UART_TXFIFO_EMPTY_THRHD 的值时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_PARITY_ERR_INT_RAW 接收器检测到数据奇偶检验位错误时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_FRM_ERR_INT_RAW 接收器检测到数据帧错误时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_RXFIFO_OVF_INT_RAW 接收器接收数据超过 RX FIFO 的存储容量时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_DSR_CHG_INT_RAW 接收器检测到 DSRn 信号的沿变化时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_CTS_CHG_INT_RAW 接收器检测到 CTSn 信号的沿变化时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_BRK_DET_INT_RAW 接收器在停止位后检测到 0 时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_RXFIFO_TOUT_INT_RAW 接收器接收一个字节所需时间超过 UART_RX_TOUT_THRHD 时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_SW_XON_INT_RAW 接收器接收到 XON 字符且 UART_SW_FLOW_CON_EN 置 1 时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_SW_XOFF_INT_RAW 接收器接收到 XOFF 字符且 UART_SW_FLOW_CON_EN 置 1 时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_GLITCH_DET_INT_RAW 接收器在起始位的中点处检测到毛刺时，该原始中断位翻转至高电平。(R/WTC/SS)

见下页...

Register 19.3. UART_INT_RAW_REG (0x0004)

接上页...

UART_TX_BRK_DONE_INT_RAW 发送器在发送完 TX FIFO 中所有数据后完成 NULL 字符的发送时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_TX_BRK_IDLE_DONE_INT_RAW 发送器发送完最后一个数据后的间隔时间达到阈值时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_TX_DONE_INT_RAW 发送器发完 FIFO 中的所有数据后，该原始中断位翻转至高电平。(R/WTC/SS)

UART_RS485_PARITY_ERR_INT_RAW RS485 模式下接收器检测到发送器回音的数据检验位错误时，该原始中断位翻转至高电平。(R/WTC/SS)

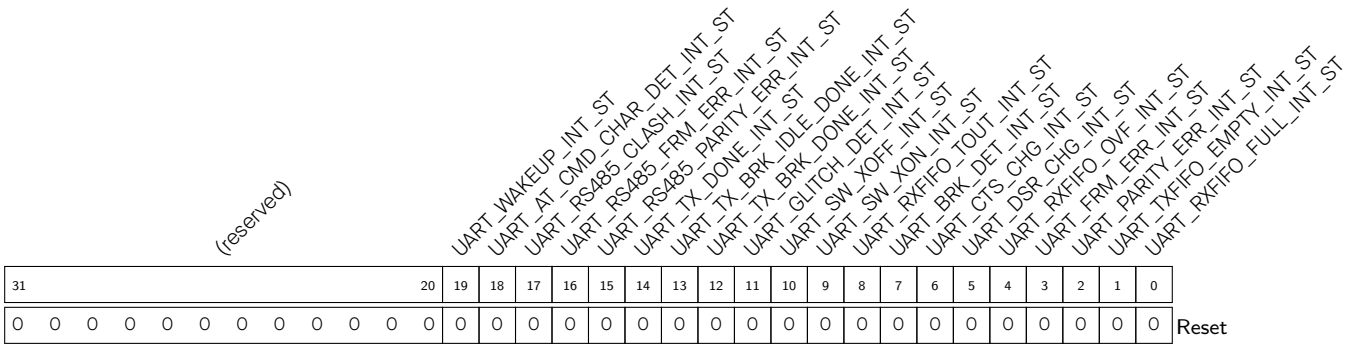
UART_RS485_FRM_ERR_INT_RAW RS485 模式下接收器检测到发送器回音的数据帧错误时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_RS485_CLASH_INT_RAW RS485 模式下检测到发送器与接收器冲突时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_AT_CMD_CHAR_DET_INT_RAW 接收器检测到配置的 UART_AT_CMD_CHAR 时，该原始中断位翻转至高电平。(R/WTC/SS)

UART_WAKEUP_INT_RAW 输入 RXD 沿变化次数超过 Light-sleep 模式指定的 UART_ACTIVE_THRESHOLD 值加 3 时，该原始中断位翻转至高电平。(R/WTC/SS)

Register 19.4. UART_INT_ST_REG (0x0008)



UART_RXFIFO_FULL_INT_ST UART_RXFIFO_FULL_INT_ENA 置 1 时 UART_RXFIFO_FULL_INT 中断的状态位。(RO)

UART_TXFIFO_EMPTY_INT_ST UART_TXFIFO_EMPTY_INT_ENA 置 1 时 UART_TXFIFO_EMPTY_INT 中断的状态位。(RO)

UART_PARITY_ERR_INT_ST UART_PARITY_ERR_INT_ENA 置 1 时 UART_PARITY_ERR_INT 中断的状态位。(RO)

UART_FRM_ERR_INT_ST UART_FRM_ERR_INT_ENA 置 1 时 UART_FRM_ERR_INT 中断的状态位。(RO)

UART_RXFIFO_OVF_INT_ST UART_RXFIFO_OVF_INT_ENA 置 1 时 UART_RXFIFO_OVF_INT 中断的状态位。(RO)

UART_DSR_CHG_INT_ST UART_DSR_CHG_INT_ENA 置 1 时 UART_DSR_CHG_INT 中断的状态位。(RO)

UART_CTS_CHG_INT_ST UART_CTS_CHG_INT_ENA 置 1 时 UART_CTS_CHG_INT 中断的状态位。(RO)

UART_BRK_DET_INT_ST UART_BRK_DET_INT_ENA 置 1 时 UART_BRK_DET_INT 中断的状态位。(RO)

UART_RXFIFO_TOUT_INT_ST UART_RXFIFO_TOUT_INT_ENA 置 1 时 UART_RXFIFO_TOUT_INT 中断的状态位。(RO)

UART_SW_XON_INT_ST UART_SW_XON_INT_ENA 置 1 时 UART_SW_XON_INT 中断的状态位。(RO)

UART_SW_XOFF_INT_ST UART_SW_XOFF_INT_ENA 置 1 时 UART_SW_XOFF_INT 中断的状态位。(RO)

UART_GLITCH_DET_INT_ST UART_GLITCH_DET_INT_ENA 置 1 时 UART_GLITCH_DET_INT 中断的状态位。(RO)

见下页...

Register 19.4. UART_INT_ST_REG (0x0008)

接上页...

UART_TX_BRK_DONE_INT_ST UART_TX_BRK_DONE_INT_ENA 置 1 时 UART_TX_BRK_DONE_INT 中断的状态位。(RO)

UART_TX_BRK_IDLE_DONE_INT_ST UART_TX_BRK_IDLE_DONE_INT_ENA 置 1 时 UART_TX_BRK_IDLE_DONE_INT 中断的状态位。(RO)

UART_TX_DONE_INT_ST UART_TX_DONE_INT_ENA 置 1 时 UART_TX_DONE_INT 中断的状态位。(RO)

UART_RS485_PARITY_ERR_INT_ST UART_RS485_PARITY_INT_ENA 置 1 时 UART_RS485_PARITY_ERR_INT 中断的状态位。(RO)

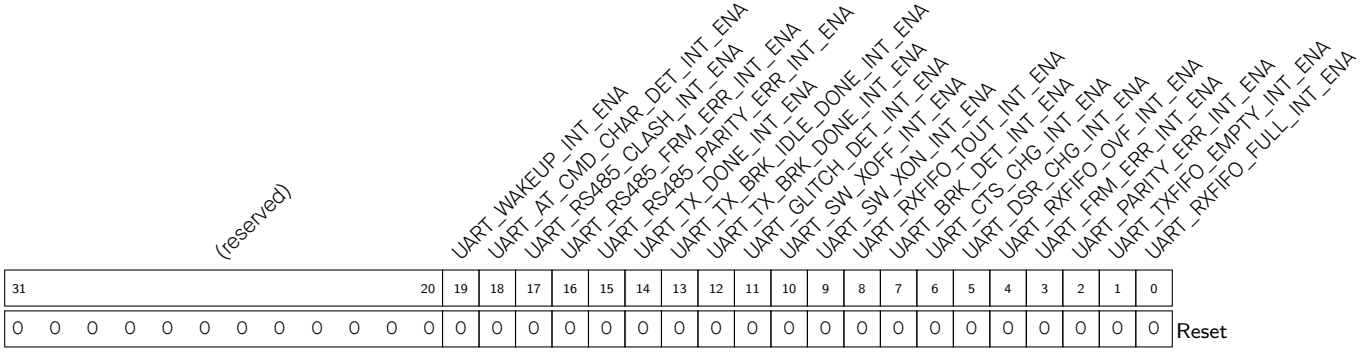
UART_RS485_FRM_ERR_INT_ST UART_RS485_FRM_ERR_INT_ENA 置 1 时 UART_RS485_FRM_ERR_INT 中断的状态位。(RO)

UART_RS485_CLASH_INT_ST UART_RS485_CLASH_INT_ENA 置 1 时 UART_RS485_CLASH_INT 中断的状态位。(RO)

UART_AT_CMD_CHAR_DET_INT_ST UART_AT_CMD_CHAR_DET_INT_ENA 置 1 时 UART_AT_CMD_CHAR_DET_INT 中断的状态位。(RO)

UART_WAKEUP_INT_ST UART_WAKEUP_INT_ENA 置 1 时 UART_WAKEUP_INT 中断的状态位。(RO)

Register 19.5. UART_INT_ENA_REG (0x000C)



UART_RXFIFO_FULL_INT_ENA UART_RXFIFO_FULL_INT 中断的使能位。(R/W)

UART_TXFIFO_EMPTY_INT_ENA UART_TXFIFO_EMPTY_INT 中断的使能位。(R/W)

UART_PARITY_ERR_INT_ENA UART_PARITY_ERR_INT 中断的使能位。(R/W)

UART_FRM_ERR_INT_ENA UART_FRM_ERR_INT 中断的使能位。(R/W)

UART_RXFIFO_OVF_INT_ENA UART_RXFIFO_OVF_INT 中断的使能位。(R/W)

UART_DSR_CHG_INT_ENA UART_DSR_CHG_INT 中断的使能位。(R/W)

UART_CTS_CHG_INT_ENA UART_CTS_CHG_INT 中断的使能位。(R/W)

UART_BRK_DET_INT_ENA UART_BRK_DET_INT 中断的使能位。(R/W)

UART_RXFIFO_TOUT_INT_ENA UART_RXFIFO_TOUT_INT 中断的使能位。(R/W)

UART_SW_XON_INT_ENA UART_SW_XON_INT 中断的使能位。(R/W)

UART_SW_XOFF_INT_ENA UART_SW_XOFF_INT 中断的使能位。(R/W)

UART_GLITCH_DET_INT_ENA UART_GLITCH_DET_INT 中断的使能位。(R/W)

UART_TX_BRK_DONE_INT_ENA UART_TX_BRK_DONE_INT 中断的使能位。(R/W)

UART_TX_BRK_IDLE_DONE_INT_ENA UART_TX_BRK_IDLE_DONE_INT 中断的使能位。(R/W)

UART_TX_DONE_INT_ENA UART_TX_DONE_INT 中断的使能位。(R/W)

UART_RS485_PARITY_ERR_INT_ENA UART_RS485_PARITY_ERR_INT 中断的使能位。(R/W)

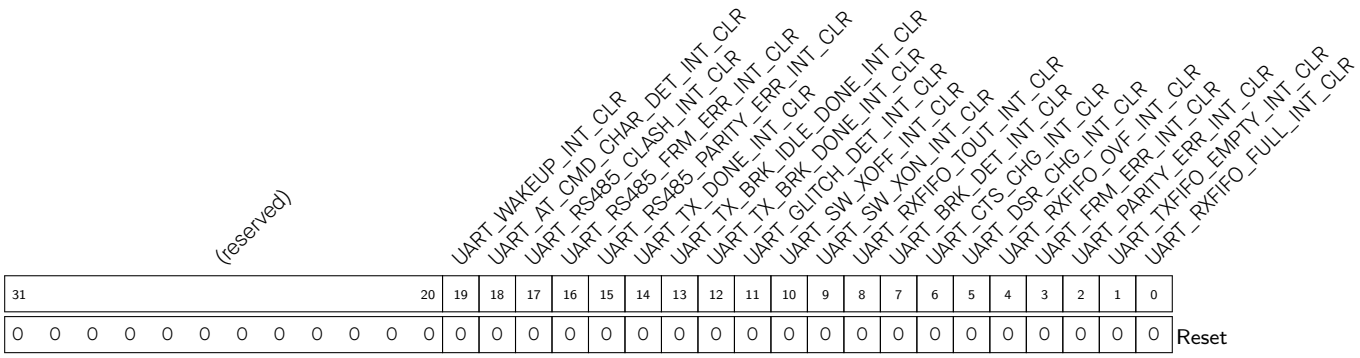
UART_RS485_FRM_ERR_INT_ENA UART_RS485_PARITY_ERR_INT 中断的使能位。(R/W)

UART_RS485_CLASH_INT_ENA UART_RS485_CLASH_INT 中断的使能位。(R/W)

UART_AT_CMD_CHAR_DET_INT_ENA UART_AT_CMD_CHAR_DET_INT 中断的使能位。(R/W)

UART_WAKEUP_INT_ENA UART_WAKEUP_INT 中断的使能位。(R/W)

Register 19.6. UART_INT_CLR_REG (0x0010)



UART_RXFIFO_FULL_INT_CLR 置位此位清除 UART_RXFIFO_FULL_INT 中断。(WT)

UART_TXFIFO_EMPTY_INT_CLR 置位此位清除 UART_TXFIFO_EMPTY_INT 中断。(WT)

UART_PARITY_ERR_INT_CLR 置位此位清除 UART_PARITY_ERR_INT 中断。(WT)

UART_FRM_ERR_INT_CLR 置位此位清除 UART_FRM_ERR_INT 中断。(WT)

UART_RXFIFO_OVF_INT_CLR 置位此位清除 UART_RXFIFO_OVF_INT 中断。(WT)

UART_DSR_CHG_INT_CLR 置位此位清除 UART_DSR_CHG_INT 中断。(WT)

UART_CTS_CHG_INT_CLR 置位此位清除 UART_CTS_CHG_INT 中断。(WT)

UART_BRK_DET_INT_CLR 置位此位清除 UART_BRK_DET_INT 中断。(WT)

UART_RXFIFO_TOUT_INT_CLR 置位此位清除 UART_RXFIFO_TOUT_INT 中断。(WT)

UART_SW_XON_INT_CLR 置位此位清除 UART_SW_XON_INT 中断。(WT)

UART_SW_XOFF_INT_CLR 置位此位清除 UART_SW_XOFF_INT 中断。(WT)

UART_GLITCH_DET_INT_CLR 置位此位清除 UART_GLITCH_DET_INT 中断。(WT)

UART_TX_BRK_DONE_INT_CLR 置位此位清除 UART_TX_BRK_DONE_INT 中断。(WT)

UART_TX_BRK_IDLE_DONE_INT_CLR 置位此位清除 UART_TX_BRK_IDLE_DONE_INT 中断。(WT)

UART_TX_DONE_INT_CLR 置位此位清除 UART_TX_DONE_INT 中断。(WT)

UART_RS485_PARITY_ERR_INT_CLR 置位此位清除 UART_RS485_PARITY_ERR_INT 中断。(WT)

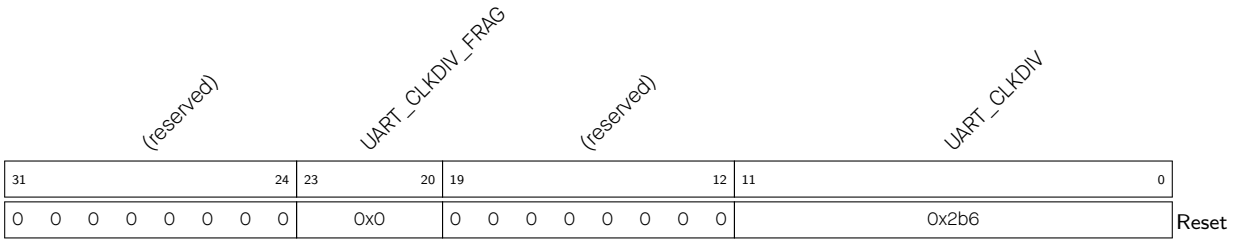
UART_RS485_FRM_ERR_INT_CLR 置位此位清除 UART_RS485_FRM_ERR_INT 中断。(WT)

UART_RS485_CLASH_INT_CLR 置位此位清除 UART_RS485_CLASH_INT 中断。(WT)

UART_AT_CMD_CHAR_DET_INT_CLR 置位此位清除 UART_AT_CMD_CHAR_DET_INT 中断。(WT)

UART_WAKEUP_INT_CLR 置位此位清除 UART_WAKEUP_INT 中断。(WT)

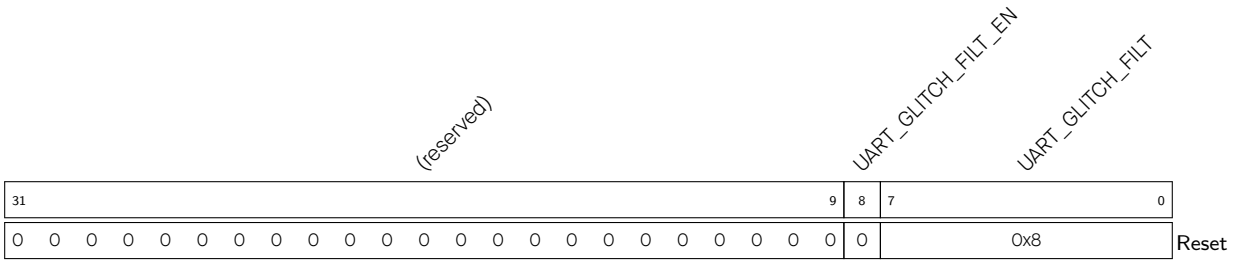
Register 19.7. UART_CLKDIV_REG (0x0014)



UART_CLKDIV 分频系数的整数部分。(R/W)

UART_CLKDIV_FRAG 分频系数的小数部分。(R/W)

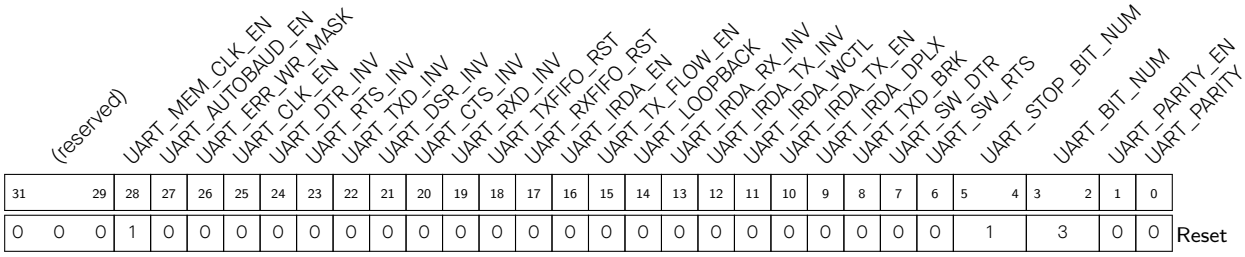
Register 19.8. UART_RX_FILT_REG (0x0018)



UART_GLITCH_FILT 宽度小于该字段值的输入脉冲会被忽略。(R/W)

UART_GLITCH_FILT_EN 置位此位，使能 RX 信号滤波器。(R/W)

Register 19.9. UART_CONF0_REG (0x0020)



UART_PARITY 配置奇偶检验方式。(R/W)

UART_PARITY_EN 置位此位使能 UART 奇偶检验。(R/W)

UART_BIT_NUM 设置数据长度。(R/W)

UART_STOP_BIT_NUM 设置停止位的长度。(R/W)

UART_SW_RTS 该位用于配置软件流控使用的软件 RTS 信号。(R/W)

UART_SW_DTR 该位用于配置软件流控使用的软件 DTR 信号。(R/W)

UART_TXD_BRK 置位此位，使能发送器在发完数据后发送 NULL。(R/W)

UART_IRDA_DPLX 置位此位开启 IrDA 回环测试模式。(R/W)

UART_IRDA_TX_EN IrDA 发送器的启动使能位。(R/W)

UART_IRDA_WCTL 1：IrDA 发送器的第 11 位与第 10 位相同；0：将 IrDA 发送器的第 11 位置 0。(R/W)

UART_IRDA_TX_INV 置位此位翻转 IrDA 发送器的电平。(R/W)

UART_IRDA_RX_INV 置位此位翻转 IrDA 接收器的电平。(R/W)

UART_LOOPBACK 置位此位开启 UART 回环测试模式。(R/W)

UART_TX_FLOW_EN 置位此位使能发送器的流控功能。(R/W)

UART_IRDA_EN 置位此位使能 IrDA 协议。(R/W)

UART_RXFIFO_RST 置位此位复位 UART RX FIFO。(R/W)

UART_TXFIFO_RST 置位此位复位 UART TX FIFO。(R/W)

UART_RXD_INV 置位此位翻转 UART RXD 信号电平。(R/W)

UART_CTS_INV 置位此位翻转 UART CTS 信号电平。(R/W)

UART_DSR_INV 置位此位翻转 UART DSR 信号电平。(R/W)

UART_TXD_INV 置位此位翻转 UART TXD 信号电平。(R/W)

UART_RTS_INV 置位此位翻转 UART RTS 信号电平。(R/W)

UART_DTR_INV 置位此位翻转 UART DTR 信号电平。(R/W)

见下页...

Register 19.9. UART_CONF0_REG (0x0020)

接上页...

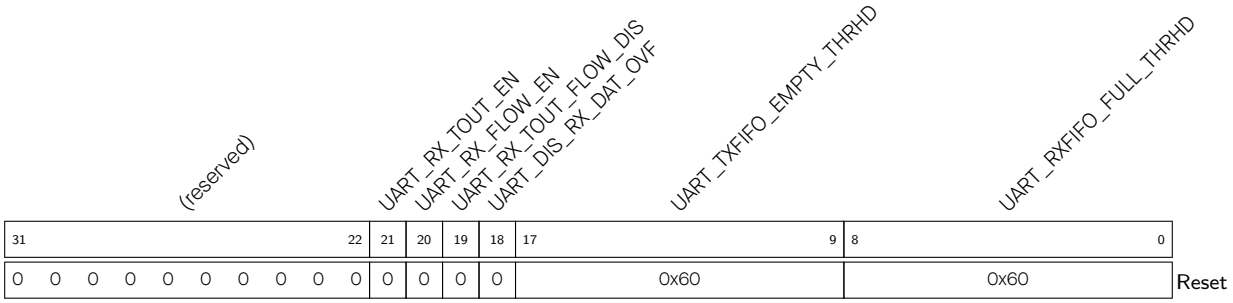
UART_CLK_EN 1：强制为寄存器开启时钟；0：仅在应用写寄存器时支持时钟。(R/W)

UART_ERR_WR_MASK 1：若数据错误，接收器不再将数据存入 FIFO；0：若数据错误，接收器仍存储。(R/W)

UART_AUTOBAUD_EN 波特率检测的使能信号。(R/W)

UART_MEM_CLK_EN UART RAM 门控使能信号。(R/W)

Register 19.10. UART_CONF1_REG (0x0024)



UART_RXFIFO_FULL_THRHD 接收器接收数据多于该字段的值时产生 UART_RXFIFO_FULL_INT 中断。(R/W)

UART_TXFIFO_EMPTY_THRHD TX FIFO 中的数据少于该字段的值时产生 UART_TXFIFO_EMPTY_INT 中断。(R/W)

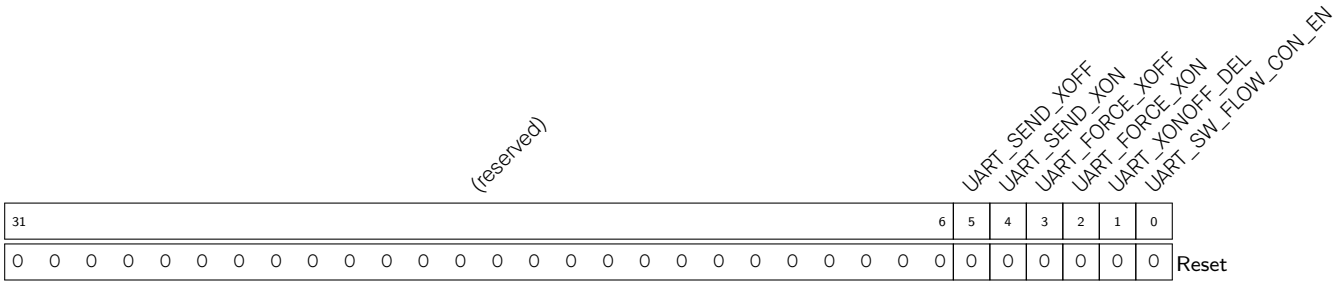
UART_DIS_RX_DAT_OVF 关闭 UART RX 数据溢出检测。(R/W)

UART_RX_TOUT_FLOW_DIS 使用硬件流控时置位此位停止堆积 idle_cnt。(R/W)

UART_RX_FLOW_EN UART 接收器流控功能的使能位。(R/W)

UART_RX_TOUT_EN UART 接收器超时功能的使能位。(R/W)

Register 19.11. UART_FLOW_CONF_REG (0x0034)



UART_SW_FLOW_CON_EN 置位此位使能软件流控。UART 接收到 UART_XON_CHAR 或 UART_XOFF_CHAR 配置的流控字符 XON 或 XOFF 时，UART_SW_XON_INT 或 UART_SW_XOFF_INT 中断可在使能时触发。(R/W)

UART_XONOFF_DEL 置位此位移除接收数据中的流控字符。(R/W)

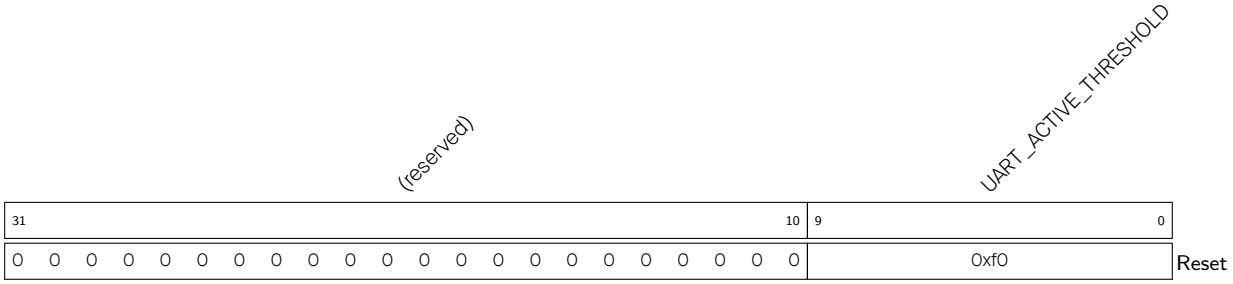
UART_FORCE_XON 置位此位让发送器继续发送数据。(R/W)

UART_FORCE_XOFF 置位此位让发送器停止发送数据。(R/W)

UART_SEND_XON 置位此位发送 XON 字符。此位由硬件自动清除。(R/W/SS/SC)

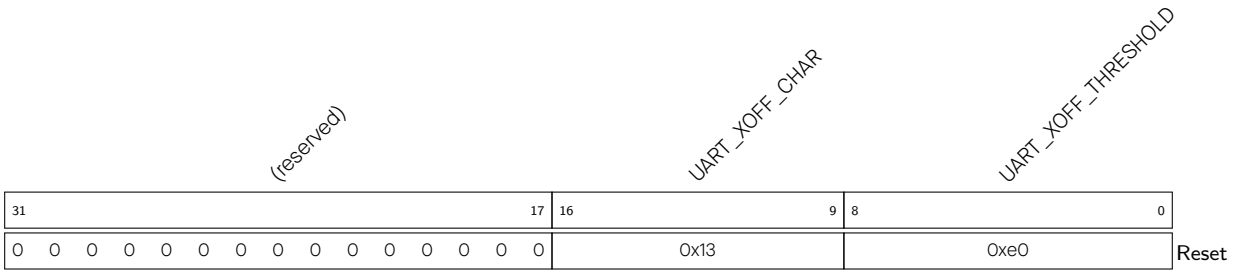
UART_SEND_XOFF 置位此位发送 XOFF 字符。此位由硬件自动清除。(R/W/SS/SC)

Register 19.12. UART_SLEEP_CONF_REG (0x0038)



UART_ACTIVE_THRESHOLD 输入 RXD 沿变化次数超过该字段的值加 3 时，UART 从 Light-sleep 模式唤醒。(R/W)

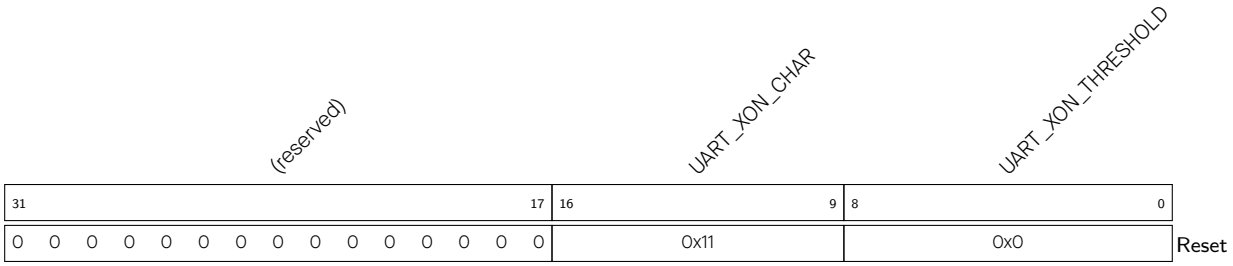
Register 19.13. UART_SWFC_CONF0_REG (0x003C)



UART_XOFF_THRESHOLD RX FIFO 中的数据超过该字段的值且 UART_SW_FLOW_CON_EN 置 1 时，发送 XOFF 字符。(R/W)

UART_XOFF_CHAR 存储 XOFF 流控字符。(R/W)

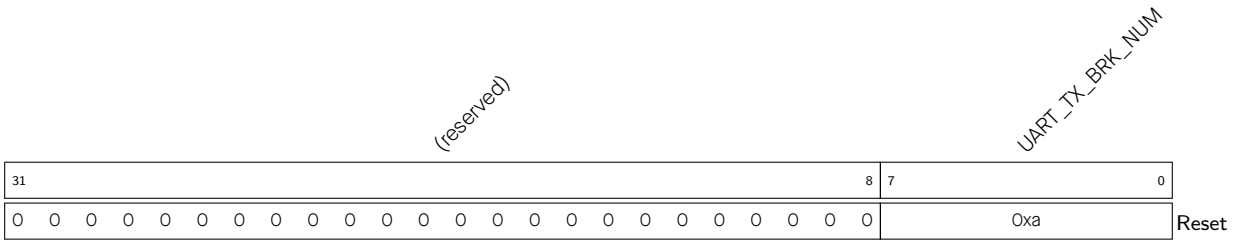
Register 19.14. UART_SWFC_CONF1_REG (0x0040)



UART_XON_THRESHOLD RX FIFO 中的数据小于该字段的值且 UART_SW_FLOW_CON_EN 置 1 时，发送 XON 字符。(R/W)

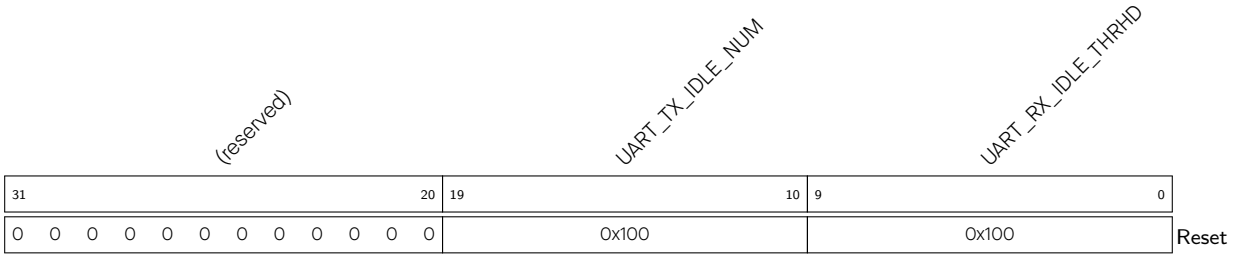
UART_XON_CHAR 存储 XON 流控字符。(R/W)

Register 19.15. UART_TXBRK_CONF_REG (0x0044)



UART_TX_BRK_NUM 配置数据发完后待发 NULL 字符的数量。UART_TXD_BRK 置 1 时有意义。(R/W)

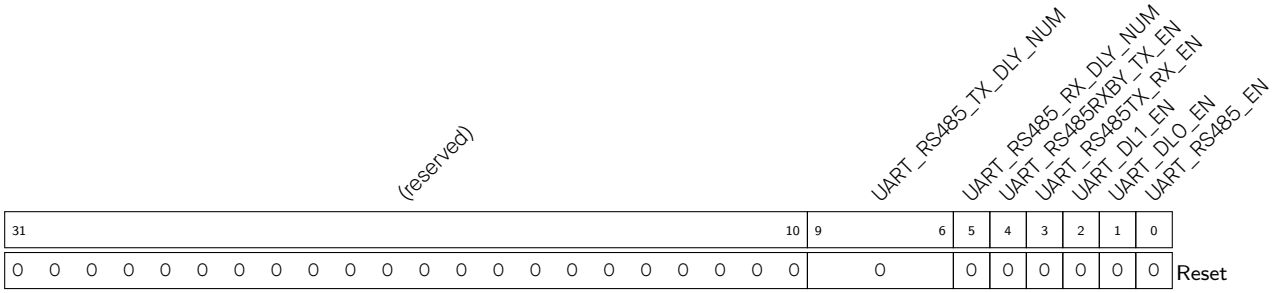
Register 19.16. UART_IDLE_CONF_REG (0x0048)



UART_RX_IDLE_THRHD 接收器接收一字节数据所需时间超过该字段的值时产生帧结束信号，单位是比特时间（即传输一个比特所需的时间）。(R/W)

UART_TX_IDLE_NUM 配置两次数据传输的间隔时间，单位是比特时间（即传输一个比特所需的时间）。(R/W)

Register 19.17. UART_RS485_CONF_REG (0x004C)



UART_RS485_EN 置位此位选择 RS485 模式。(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)

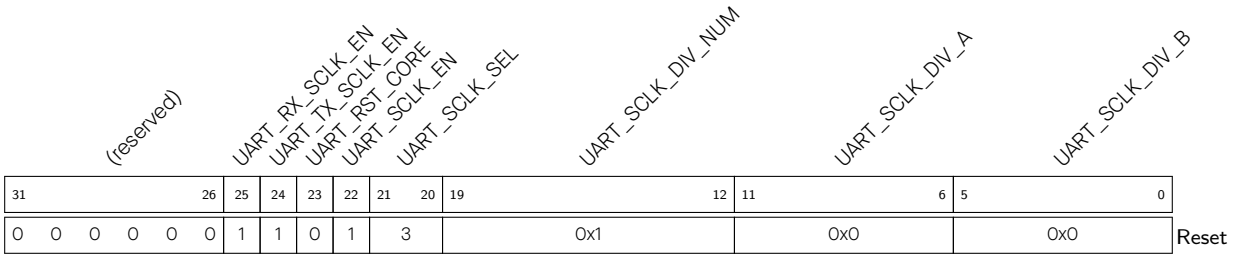
UART_RS485TX_RX_EN 发送器在 RS485 模式下发送数据时，置位此位使能接收器接收数据。(R/W)

UART_RS485RXBY_TX_EN 置位此位，在 RS485 接收器线路繁忙时使能 RS485 发送器发送数据。(R/W)

UART_RS485_RX_DLY_NUM 延迟接收器的内部数据信号。(R/W)

UART_RS485_TX_DLY_NUM 延迟发送器的内部数据信号。(R/W)

Register 19.18. UART_CLK_CONF_REG (0x0078)



UART_SCLK_DIV_B 分频系数的分母。(R/W)

UART_SCLK_DIV_A 分频系数的分子。(R/W)

UART_SCLK_DIV_NUM 分频系数的整数部分。(R/W)

UART_SCLK_SEL 选择 UART 时钟源。1：PLL_F40M_CLK；2：RC_FAST_CLK；3：XTAL_CLK。(R/W)

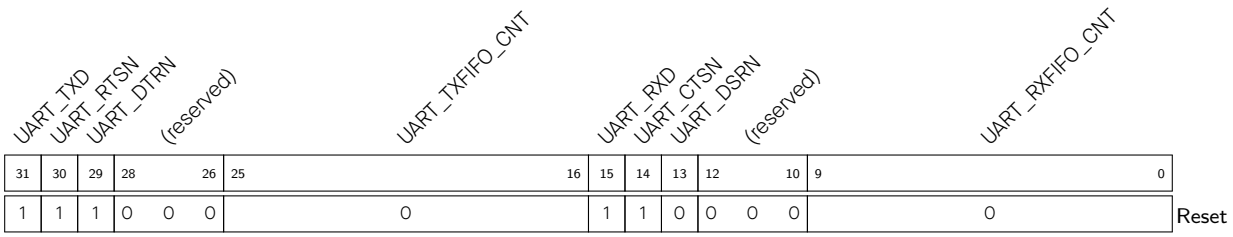
UART_SCLK_EN 置位此位，使能 UART TX/RX 使能。(R/W)

UART_RST_CORE 向此位先写 1 后写 0，复位 UART TX/RX。(R/W)

UART_TX_SCLK_EN 置位此位，使能 UART TX 时钟。(R/W)

UART_RX_SCLK_EN 置位此位，使能 UART RX 时钟。(R/W)

Register 19.19. UART_STATUS_REG (0x001C)



UART_RXFIFO_CNT 存储 RX FIFO 中有效数据的字节数。(RO)

UART_DSRN 该位表示内部 UART DSR 信号的电平值。(RO)

UART_CTSN 该位表示内部 UART CTS 信号的电平值。(RO)

UART_RXD 该位表示内部 UART RXD 信号的电平值。(RO)

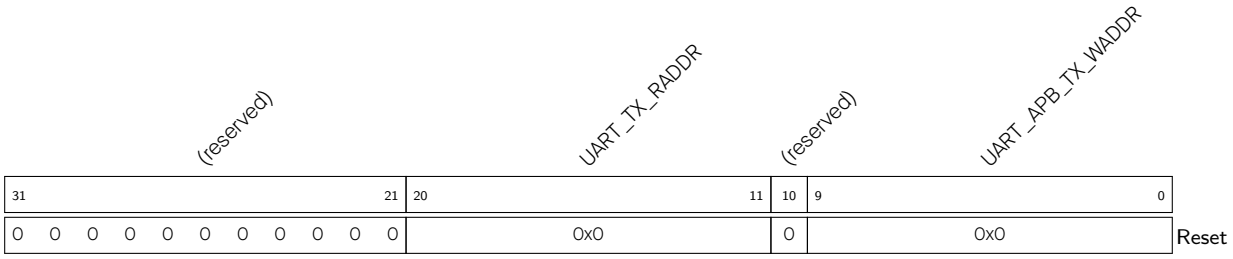
UART_TXFIFO_CNT 存储 TX FIFO 中数据的字节数。(RO)

UART_DTRN 此位表示内部 UART DTR 信号的电平。(RO)

UART_RTSN 此位表示内部 UART RTS 信号的电平。(RO)

UART_TXD 此位表示内部 UART TXD 信号的电平。(RO)

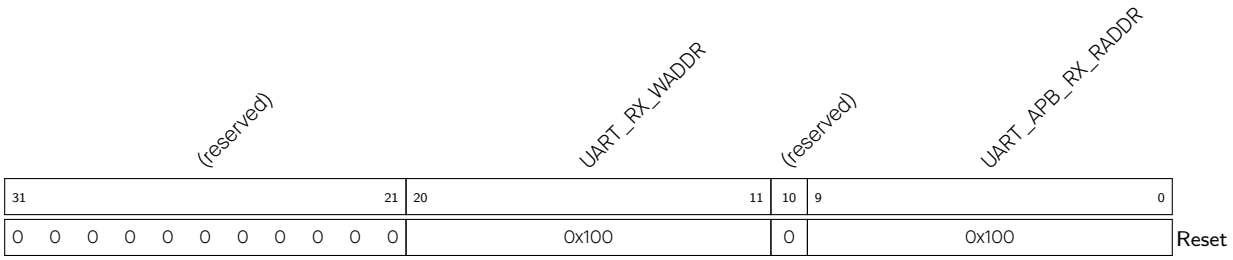
Register 19.20. UART_MEM_TX_STATUS_REG (0x0064)



UART_APB_TX_WADDR 在软件通过 APB 总线写 TX FIFO 时存储 TX FIFO 的偏移地址。(RO)

UART_TX_RADDR 在 TX FSM 通过 Tx_FIFO_Ctrl 读取数据时存储 TX FIFO 的偏移地址。(RO)

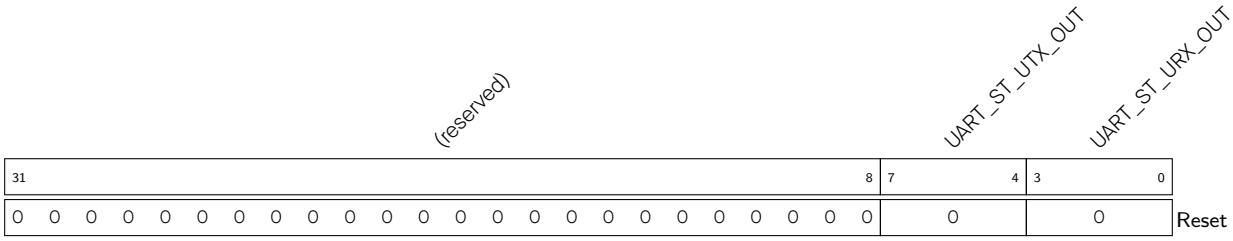
Register 19.21. UART_MEM_RX_STATUS_REG (0x0068)



UART_APB_RX_RADDR 在软件通过 APB 总线读取 RX FIFO 数据时存储 RX FIFO 的偏移地址。UART0 为 0x100，UART1 为 0x180。(RO)

UART_RX_WADDR 在 Rx_FIFO_Ctrl 写 RX FIFO 时存储 RX FIFO 的偏移地址。UART0 为 0x100，UART1 为 0x180。(RO)

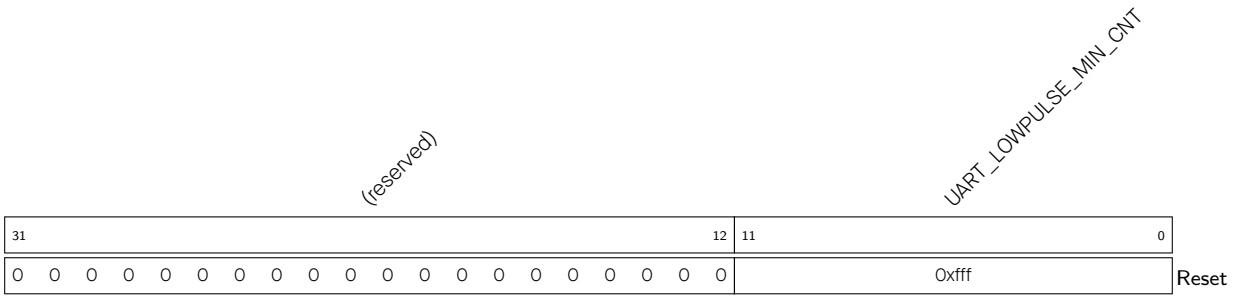
Register 19.22. UART_FSM_STATUS_REG (0x006C)



UART_ST_URX_OUT 接收器的状态字段。(RO)

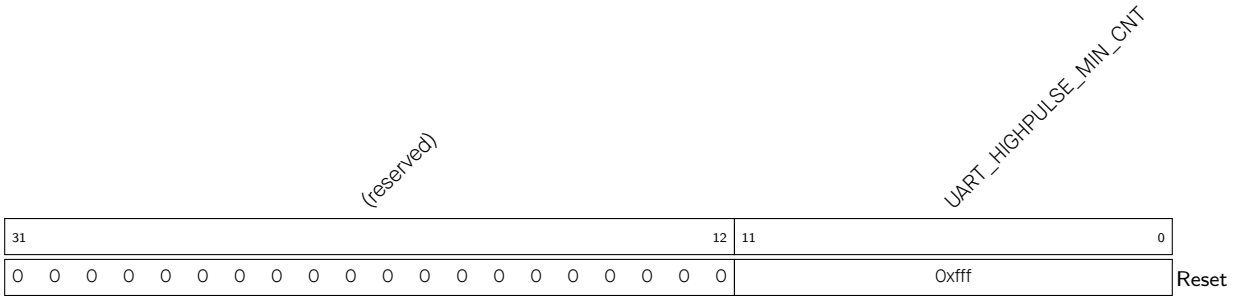
UART_ST_UTX_OUT 发送器的状态字段。(RO)

Register 19.23. UART_LOWPULSE_REG (0x0028)



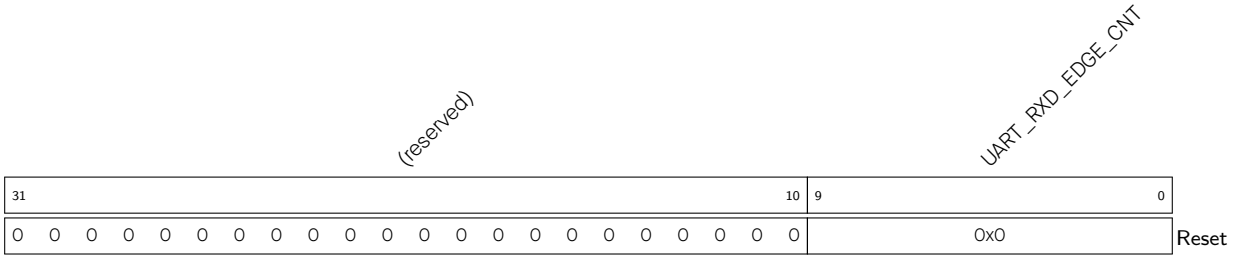
UART_LOWPULSE_MIN_CNT 存储低电平脉冲的最短持续时间，用于波特率检测，单位是 APB_CLK 时钟周期。(RO)

Register 19.24. UART_HIGHPULSE_REG (0x002C)



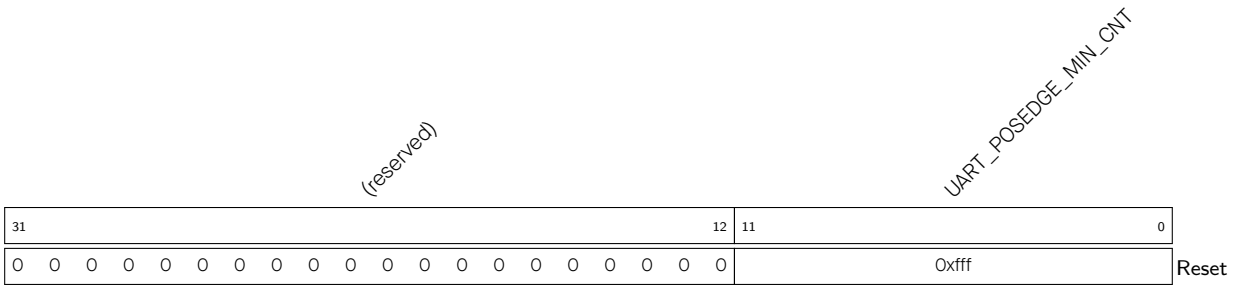
UART_HIGHPULSE_MIN_CNT 存储最长高电平脉冲持续时间。用于波特率检测，单位是 APB_CLK 时钟周期。(RO)

Register 19.25. UART_RXD_CNT_REG (0x0030)



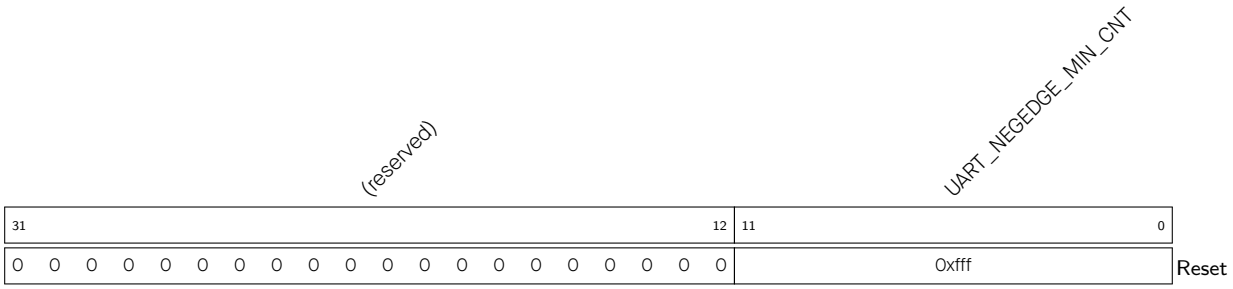
UART_RXD_EDGE_CNT 存储 RXD 沿变化的次数。用于波特率检测。(RO)

Register 19.26. UART_POSPULSE_REG (0x0070)



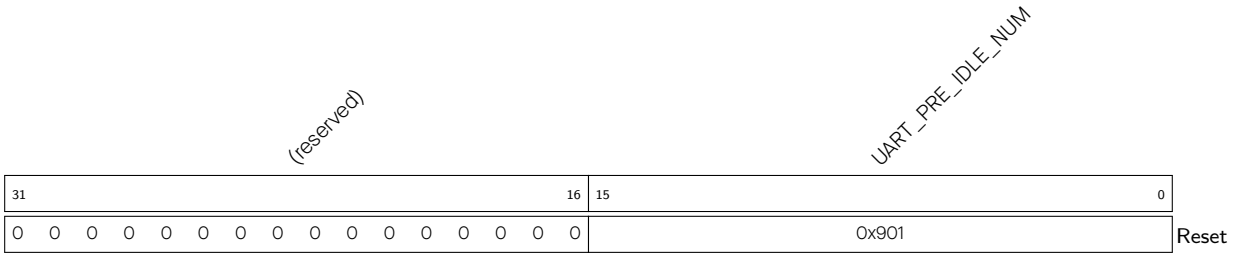
UART_POSEDGE_MIN_CNT 存储两个上升沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 19.27. UART_NEGPULSE_REG (0x0074)



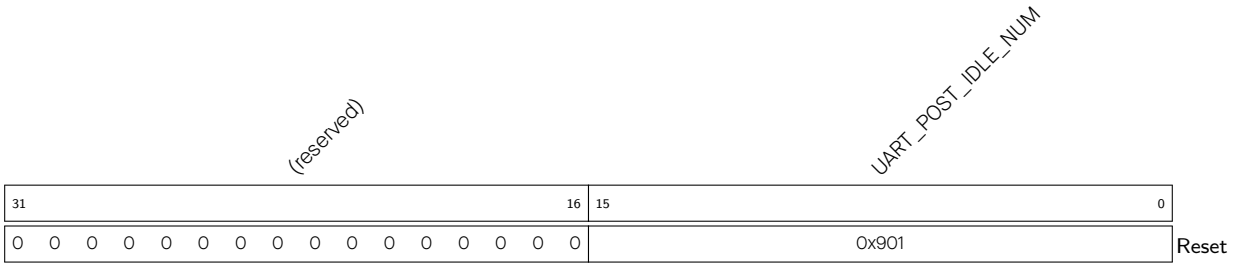
UART_NEGEDGE_MIN_CNT 存储两个下降沿之间的最小输入时钟计数值。用于波特率检测。(RO)

Register 19.28. UART_AT_CMD_PRECNT_REG (0x0050)



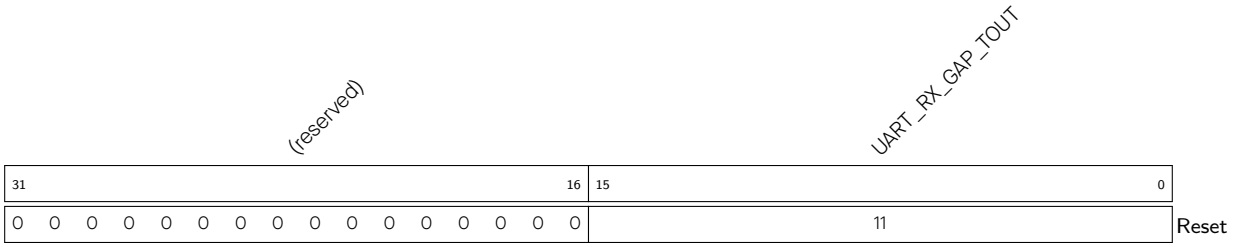
UART_PRE_IDLE_NUM 配置接收器接收第一个 AT_CMD 字符前的空闲时间，单位是比特时间（即传输一个比特所需的时间）。(R/W)

Register 19.29. UART_AT_CMD_POSTCNT_REG (0x0054)



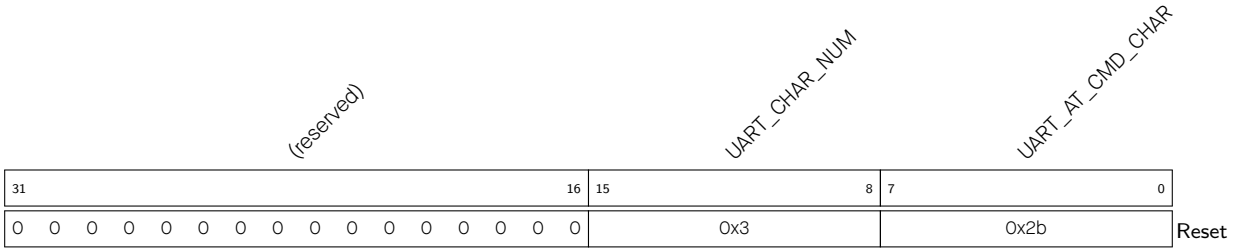
UART_POST_IDLE_NUM 配置最后一个 AT_CMD 字符和后续数据的间隔时间，单位是比特时间（即传输一个比特所需的时间）。(R/W)

Register 19.30. UART_AT_CMD_GAPTOUT_REG (0x0058)



UART_RX_GAP_TOUT 配置 AT_CMD 字符的间隔时间，单位是比特时间（即传输一个比特所需的时间）。(R/W)

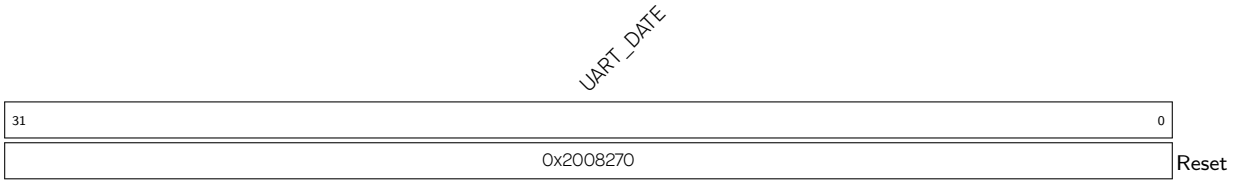
Register 19.31. UART_AT_CMD_CHAR_REG (0x005C)



UART_AT_CMD_CHAR 配置 AT_CMD 字符的内容。(R/W)

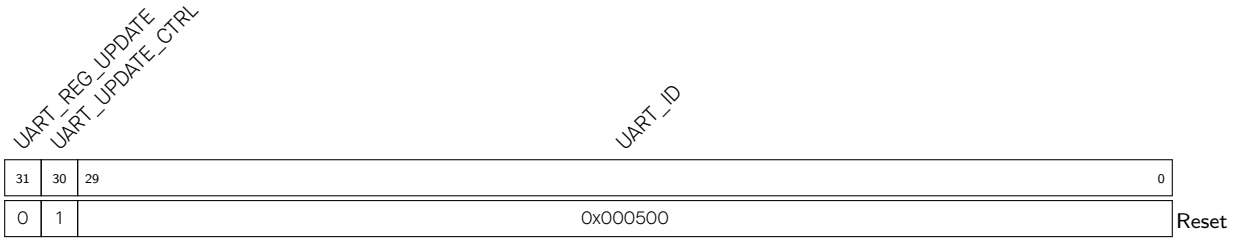
UART_CHAR_NUM 配置接收器接收连续 AT_CMD 字符的个数。(R/W)

Register 19.32. UART_DATE_REG (0x007C)



UART_DATE 版本控制寄存器。(R/W)

Register 19.33. UART_ID_REG (0x0080)



UART_ID 配置 UART_ID。(R/W)

UART_UPDATE_CTRL 用于控制同步模式。在向 UART_REG_UPDATE 写 1 同步配置寄存器至 UART Core 时钟域之前，该位必须配置为 0。(R/W)

UART_REG_UPDATE 软件向该位写 1，将寄存器值同步到 UART Core 时钟域。该字段在同步完成后由硬件自清。(R/W/SC)

20 SPI 控制器 (SPI)

20.1 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。ESP8684 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。

20.2 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。

20.3 特性

GP-SPI2 具有以下特性：

- 支持主机模式和从机模式
- 支持半双工通信和全双工通信
- 支持 CPU 控制的传输模式以及 DMA 控制的传输模式
- 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：
  - 在主机模式下：时钟频率可达 40 MHz
  - 在从机模式下：时钟频率可达 40 MHz
- 数据长度可配置：
  - 在主机和从机 CPU 控制的传输模式下：数据长度为 1 ~ 64 B
  - 在主机 DMA 控制的单次传输模式下：数据长度为 1 ~ 32 KB
  - 在主机 DMA 控制的分段配置传输模式下：数据长度字节数无限制
  - 在从机 DMA 控制的单次或连续传输模式下：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
- 在主机模式下，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

20.4 架构概览

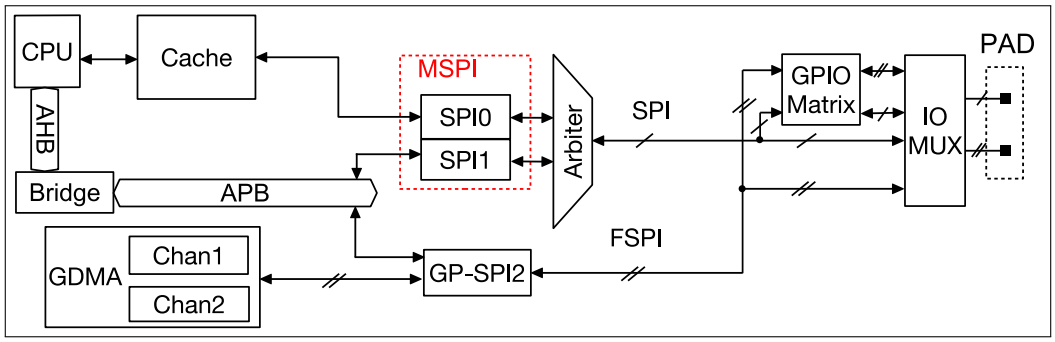


图 20-1. SPI 模块概览

图 20-1 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 控制的传输模式下：CPU <-> GP-SPI2 <-> SPI 设备
- 在 DMA 控制的传输模式下：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 *5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)*。

20.5 功能描述

20.5.1 数据模式

GP-SPI2 可配置成主机或从机模式，采用表 20-2 所示的数据模式与其它 SPI 设备进行通信。

表 20-2. 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

主机模式下，有效的阶段请参考第 20.5.8 小节；从机模式下，有效的阶段请参考第 20.5.9 小节。

20.5.2 FSPI 总线信号描述

FSPI 总线信号的功能描述如表 20-3。各种 SPI 模式下使用到的信号见表 20-4。

表 20-3. 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	主从机模式，输入输出时钟
FSPICS0	主从机模式，输入输出片选信号
FSPICS1 ~ 5	主机模式，输出片选信号

[a] SIO0：全称为 serial data input and output, bit0

表 20-4. 各种 SPI 模式下使用到的信号

FSPI 总线信号	主机模式						从机模式					
	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

20.5.3 数据位读/写顺序控制

在主机模式下：

- 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。

在从机模式下：

- 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 20-5 所示为 SPI_RD/WR_BIT_ORDER 的功能。

表 20-5. GP-SPI 主机模式和从机模式下的数据位控制

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

20.5.4 传输方式

GP-SPI2 在主机模式和从机模式下支持的传输方式见下表。

表 20-6. 主机模式和从机模式下支持的传输方式

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方式。

20.5.5 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 20-2。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。在这种传输方式下，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。

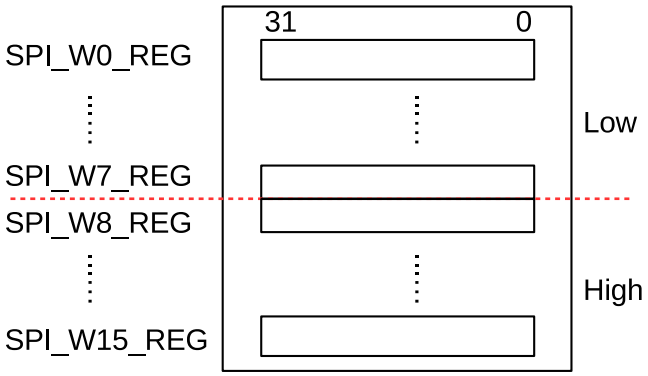


图 20-2. CPU 控制的传输中使用的数据 Buffer

20.5.5.1 CPU 控制的主机模式

主机模式下，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 中的数据可能会被多次发送。**
    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 中的数据将被多次发送**。
  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 中的数据可能被覆盖**。
    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 中的数据将被覆盖。**
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。

20.5.5.2 CPU 控制的从机模式

从机模式下，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

20.5.6 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。主机模式和从机模式均支持这种传输方式。

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 主机模式和从机模式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 主机模式支持这种分段配置传输。更多信息，见章节 20.5.8.5。
- 从机连续传输，包含多次传输事务。仅有 GP-SPI2 从机模式支持这种从机连续传输。更多信息，见章节 20.5.9.3。

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 20.5.8 和章节 20.5.9 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。

20.5.6.1 GDMA 配置

- 选择 GDMA 通道 *n*，并配置 GDMA TX/RX 描述符，见章节 *2 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。

主机模式下，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。

从机模式下，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 20-7. GP-SPI2 从机模式下数据传输中断触发条件

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

20.5.6.2 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。

20.5.7 GP-SPI2 主机模式和从机模式下的数据流控制

GP-SPI2 主机模式和从机模式均支持 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 来选择需要的传输方式。

20.5.7.1 GP-SPI2 功能块图

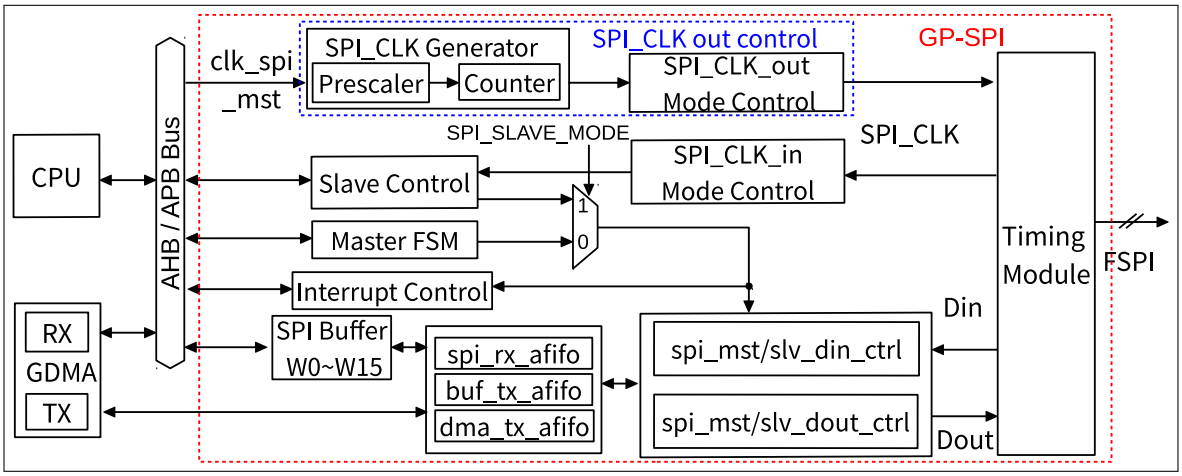


图 20-3. GP-SPI2 功能块图

图 20-3 所示为 GP-SPI2 主要的功能模块，包括：

- Master FSM：GP-SPI2 的主机状态机。主机模式下支持的所有功能，均由该状态机与寄存器共同控制。
- SPI Buffer: SPI_W0_REG ~ SPI_W15_REG，见图 20-2。CPU 控制模式下传输的数据在 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 主机模式下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 20.7。
- 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。

20.5.7.2 主机模式下的数据流控制

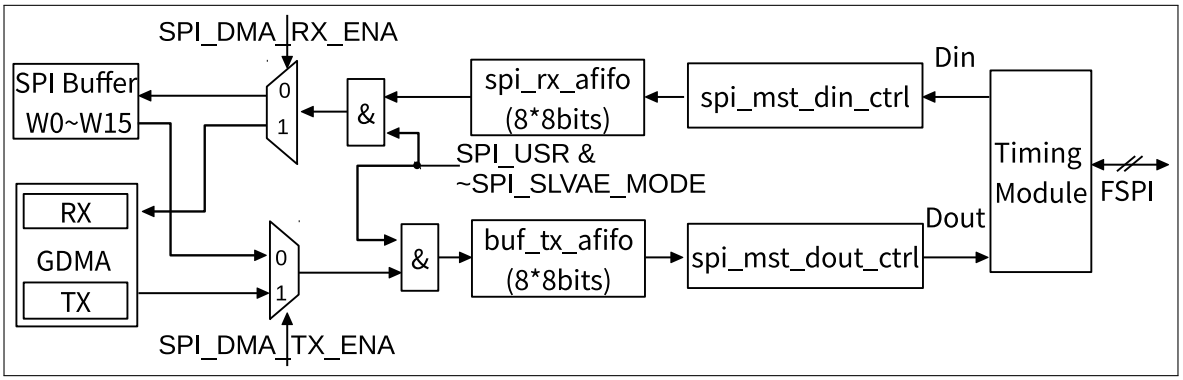


图 20-4. GP-SPI2 主机模式下的数据流控制

图 20-4 所示为 GP-SPI2 在主机模式下的数据流。主机模式下，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。更多信息，见章节 20.8。

20.5.7.3 从机模式下的数据流控制

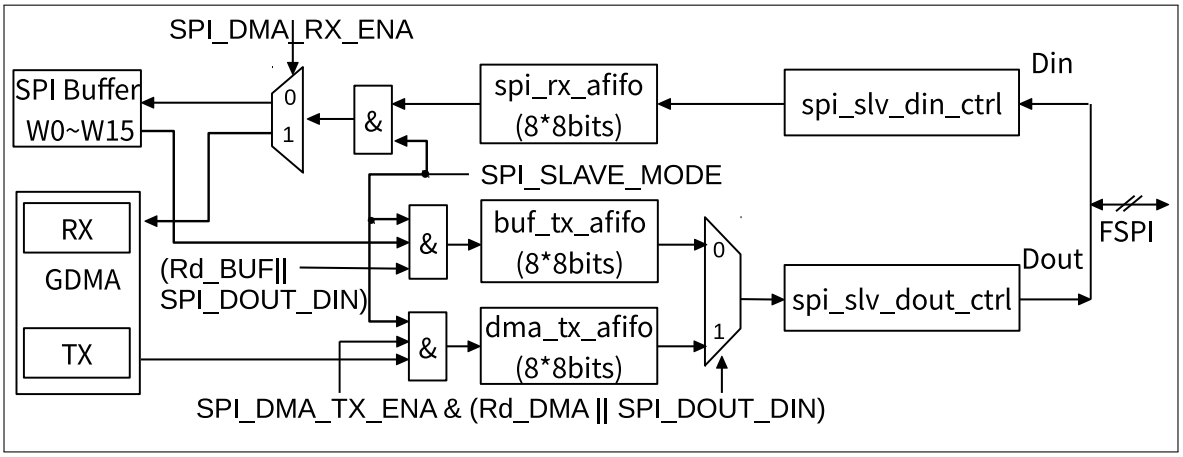


图 20-5. GP-SPI2 从机模式下的数据流控制

图 20-5 所示为 GP-SPI2 在从机模式下的数据流控制。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。

20.5.8 GP-SPI2 主机模式

清零 SPI_SLAVE_REG 中 SPI_SLAVE_MODE 位可将 GP-SPI2 配置成主机模式。在这种模式下，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。

20.5.8.1 主机模式状态机

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)、空闲阶段 (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 20-6。

1. 空闲阶段 (IDLE)：GP-SPI2 未处于工作状态或处于从机模式。
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。

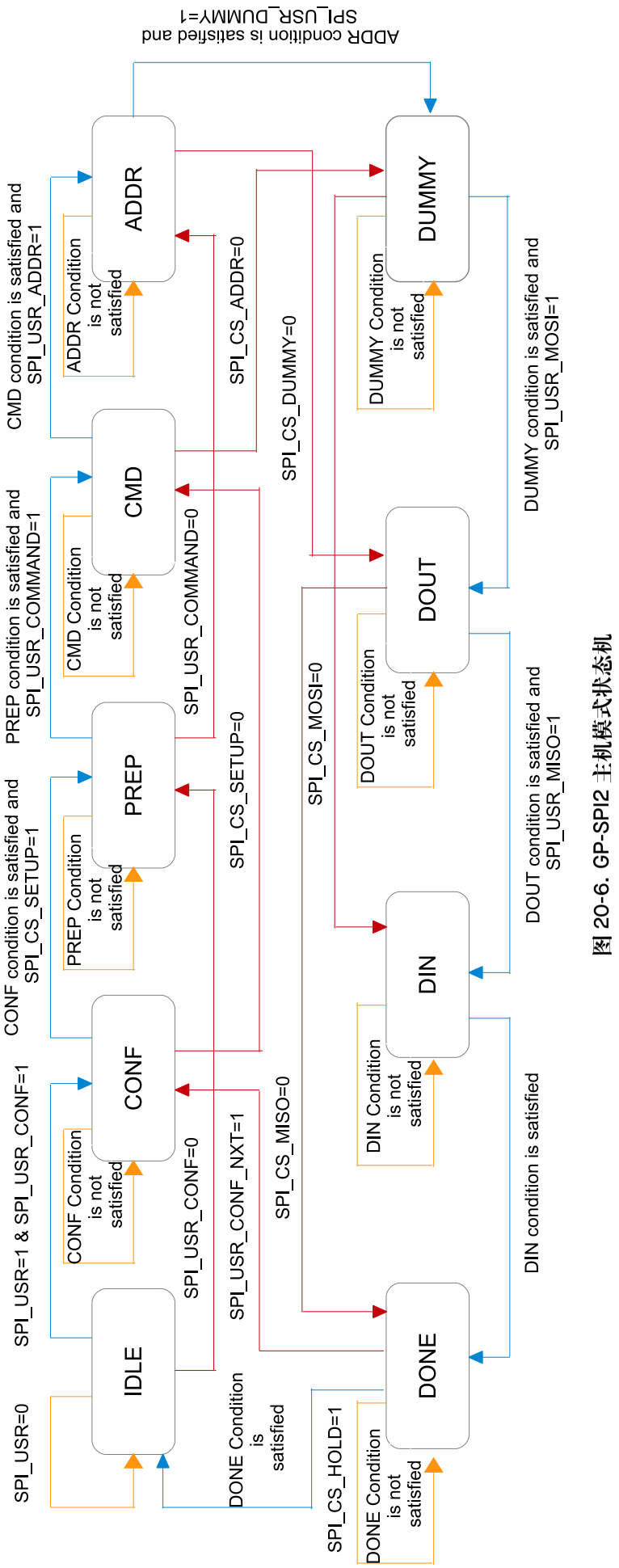


图 20-6. GP-SPI2 主机模式状态机

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。

20.5.8.2 状态控制和位模式控制寄存器

概述

表 20-8 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 20-8. 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 20-8 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 控制的传输模式，则需要配置 GDMA。如果选择了 CPU 控制的传输模式，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 控制的传输模式下，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 20-9. 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 20-10. 地址值的发送顺序

ADDR_BITLEN[1]	ADDR_VALUE[2]	BIT_ORDER[3]	地址值的发送顺序
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。

20.5.8.3 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。在这种模式下，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 20-7。

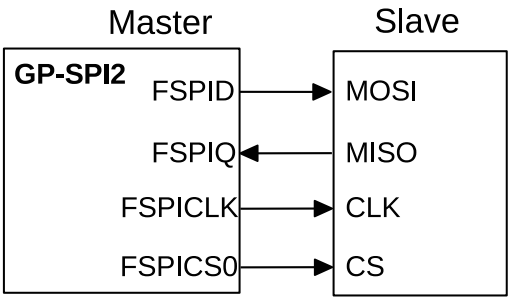


图 20-7. GP-SPI2 主机使用全双工模式与 SPI 从机通信框图

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，全双工模式跳过 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 时钟、APB 时钟（即 AHB_CLK、APB_CLK，见章节 6 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，使能主机模式下的全双工通信方式；
- 配置表 20-8 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 20.6；
- 设置 FSPICLK 的极性，见章节 20.7；
- 根据选定的传输模式准备数据：
  - 如果选择的传输模式为 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 控制的传输模式，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 20.5.6 和章节 20.5.7。
- 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。

20.5.8.4 主机半双工通信（支持 1/2/4-bit 模式）

概述

在半双工模式下，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 20.5.8.2 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 20-8。

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 控制的模式下，可传输 0 ~ 512 位（64 字节）数据；在 DMA 控制的模式下，可传输 0 ~ 256 Kbit (32 KB)。主机发送，从机接收。
5. DIN：在 CPU 控制的模式下，可传输 0 ~ 512 位（64 字节）数据；在 DMA 控制的模式下，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 时钟、APB 时钟（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，使能主机模式下的半双工通信方式；
4. 配置表 20-8 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 20.6；
6. 设置 FSPICLK 的极性，见章节 20.7；
7. 根据选定的传输模式准备数据：
   - 如果选择的传输模式为 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 控制的传输模式，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 20.5.6 和章节 20.5.7。
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。

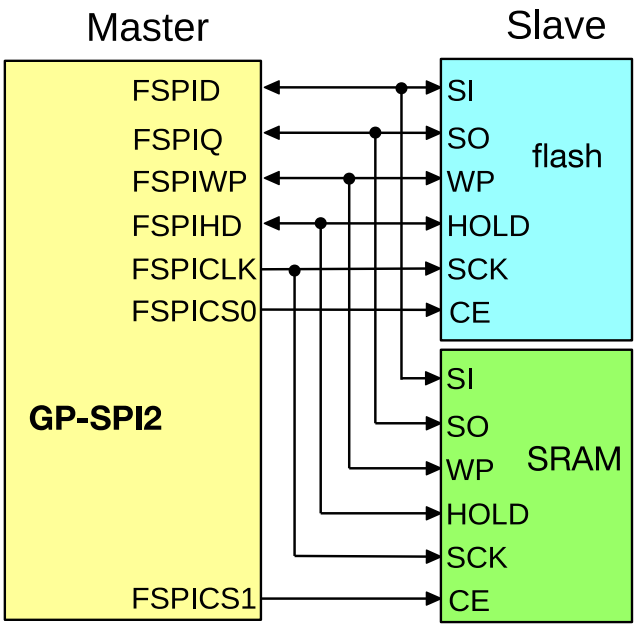


图 20-8. 4-bit 模式下 GP-SPI2 与 Flash 以及外部 RAM 的连接方式

图 20-9 所示为 GP-SPI2 按照标准 flash 规范进行 Quad I/O Read 操作。其它 GP-SPI2 命令序列可以根据 SPI 从机的要求实现。

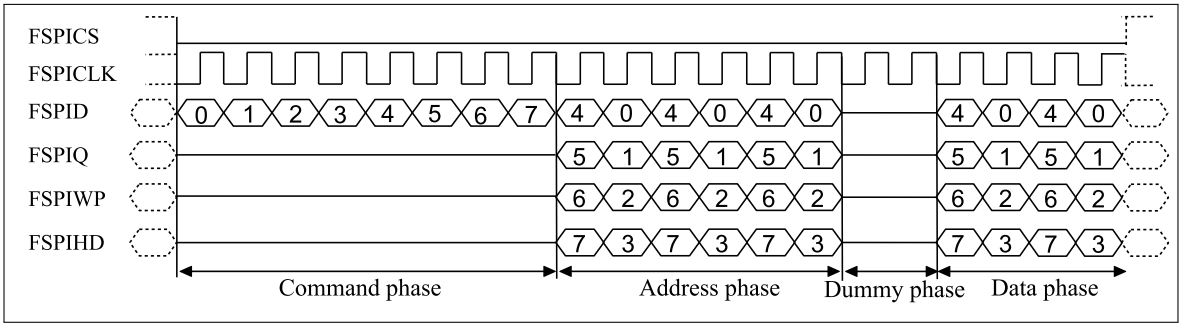


图 20-9. GP-SPI2 发送到 Flash 的 SPI Quad I/O 命令序列

20.5.8.5 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主机模式下配置单次传输。

概述

GP-SPI2 用作主机时，可采用 DMA 控制的分段配置传输模式。

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模式，则在每个分段中，寄存器均可单独配置。在分段配置传输模式下，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 20-10。

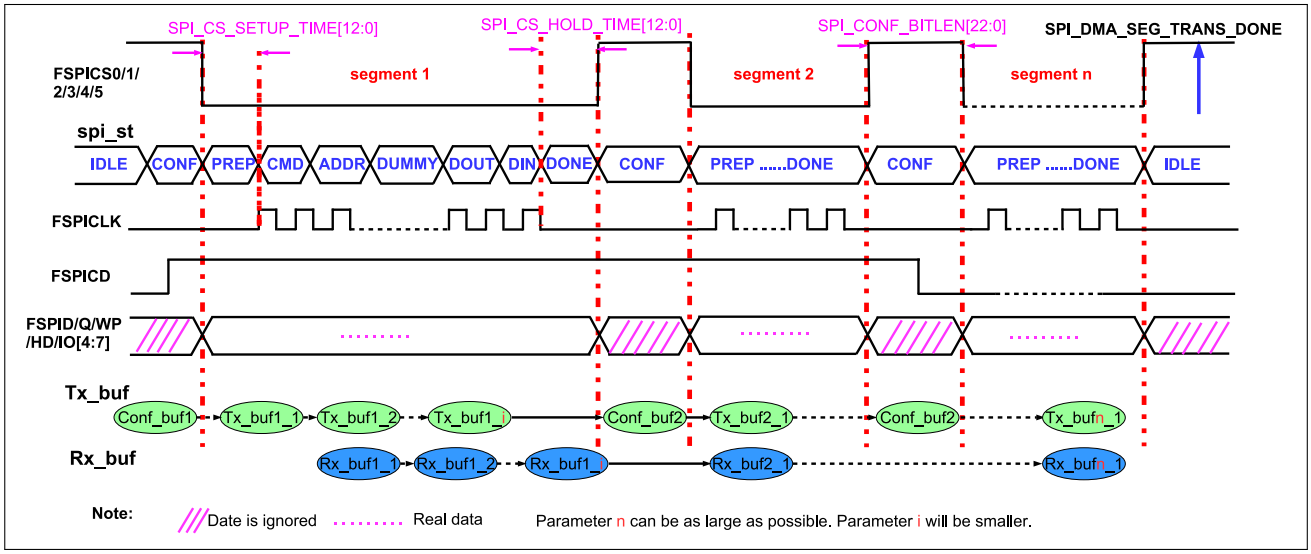


图 20-10. 主机模式下 DMA 控制的分段配置传输

如图 20-10 所示，在分段配置传输模式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 20-10 中的 Conf_buf*i*）。GDMA TX 链表将所有的 CONF buffer 和 TX data buffer（即图 20-10 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 时钟、APB 时钟（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，使能主机模式下的半双工通信方式；
4. 配置表 20-8 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 20.6；
6. 设置 FSPICLK 的相位和极性，见章节 20.7；
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。如需配置其它中断，请参考章节 20.9；
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，其它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 20-11，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。如果其它寄存器不需要修改，则位图表中相应位应置为 0。

表 20-11. 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	保留
2	SPI_CLOCK_REG	9	保留
3	SPI_USER_REG	10	保留
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 20-12 和表 20-13。

表 20-12. 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 20-13 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 20-13. 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	保留
2	1	SPI_CLOCK_REG	9	0	保留
3	1	SPI_USER_REG	10	0	保留
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 20.6。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_{AHB_CLK} 为 40 MHz 时，CONF 阶段的 CS 高电平时长可配置为 125 μs ~ 6.5536 ms。如果 SPI_CONF_BITLEN 大于 0x3FFFA，(SPI_CONF_BITLEN + 5) 将溢出 (0x40000 - SPI_CONF_BITLEN - 5)。

20.5.9 GP-SPI2 从机模式

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 从机模式。

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。

20.5.9.1 可配置的通信格式

GP-SPI2 从机模式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。

全双工模式下，传输一开始，则数据同时输入和输出。在此模式下，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。

在半双工通信模式下，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 20-14 和表 20-15 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，或在其它命令中用作占位符，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 控制的模式下，可传输 0 ~ 64 字节数据；在 DMA 控制的模式下，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：
半双工通信模式下，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 从机模式不支持传输的 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。

20.5.9.2 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 控制的写操作模式。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 控制的读操作模式。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 控制的写操作模式。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 控制的读操作模式。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。在从机连续传输模式下，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 20-14 和表 20-15。注意，DUMMY 仅支持 1-bit 模式，且持续八个 SPI_CLK 时钟周期。

表 20-14. 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
	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 20-15. 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 20-14 和表 20-15 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 20.9。

20.5.9.3 从机单次传输和从机连读传输

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 20-14 和表 20-15 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。

20.5.9.4 配置从机单次传输模式

在从机模式下，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 时钟、APB 时钟（AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 使能从机模式；
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：
   - 如果选择的传输模式为 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。
   - 如果选择的传输模式为 DMA 控制的传输模式，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 20.5.6 和章节 20.5.7。
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 控制模式下，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。其它中断见章节 20.9。

20.5.9.5 配置半双工模式下从机连续传输

此模式必须使用 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 时钟、APB 时钟（AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 使能从机模式；
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 20.5.6 和章节 20.5.7；
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。其它中断见章节 20.9。

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。

20.5.9.6 配置全双工模式下从机连续传输

在这一传输模式中，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。具体的配置程序程如下：

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 时钟、APB 时钟（AHB_CLK、APB_CLK）；
3. 置位 SPI_SLAVE_MODE 和 SPI_DOUTDIN，使能从机全双工通信模式；

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 20.5.6 和章节 20.5.7；
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。

20.6 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。

从机模式下，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。

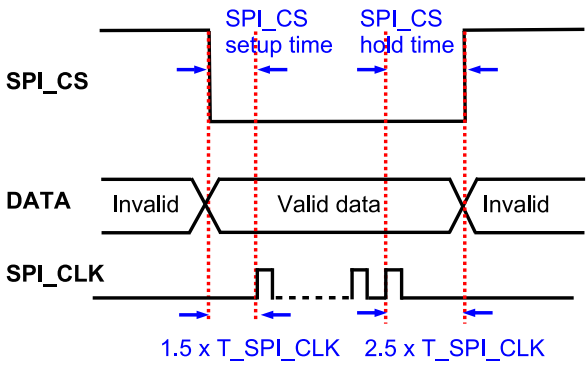
主机模式下，CS 建立时间由寄存器 SPI_USER_REG 中的 SPI_CS_SETUP 位和寄存器 SPI_USER1_REG 中的 SPI_CS_SETUP_TIME 位控制：

- 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 20-11 和图 20-12 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 20-11. GP-SPI2 访问外部 RAM 时推荐的 CS 时序配置

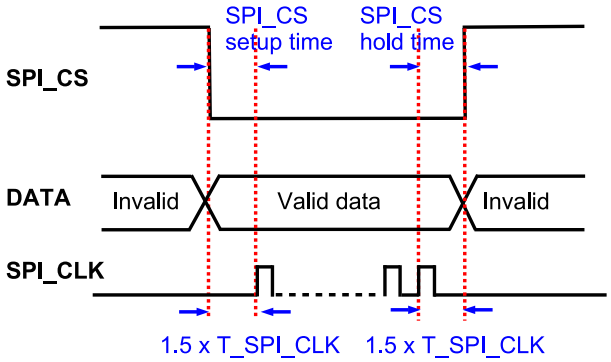


图 20-12. GP-SPI2 访问 Flash 时推荐的 CS 时序配置

20.7 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，由 PLL_CLK 分频所得。在 GP-SPI2 主机模式下用于生成数据传输以及从机所需的 SPI_CLK 信号；
- SPI_CLK：主机模式输出时钟；
- AHB_CLK/APB_CLK：用于寄存器配置的时钟。

主机模式下 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 $f_{clk\_spi\_mst}$。如果采用其它整数分频，则 SPI_CLK_EQU_SYSCLK 应置 0。

从机模式下，GP-SPI2 支持的输入时钟频率为：

$$f_{SPI\_CLK} <= 40MHz$$

20.7.1 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 20-13 和图 20-14。注，图片来源于 SPI 协议。

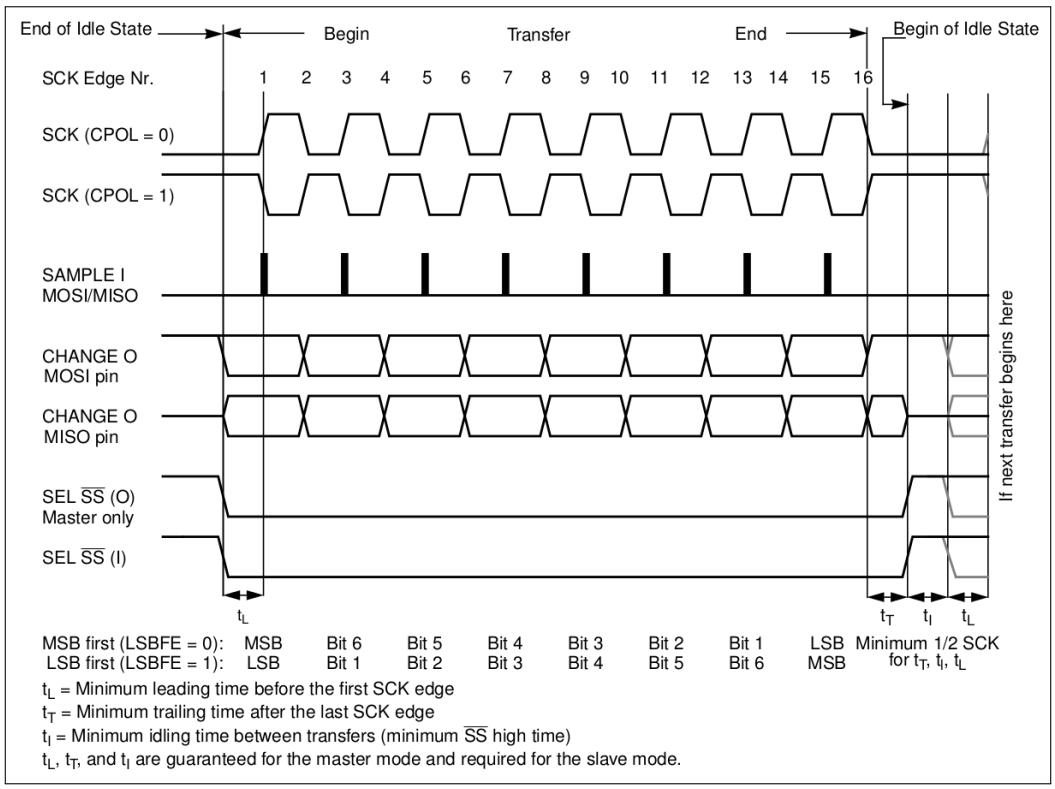


图 20-13. SPI 时钟模式 0 和时钟模式 2

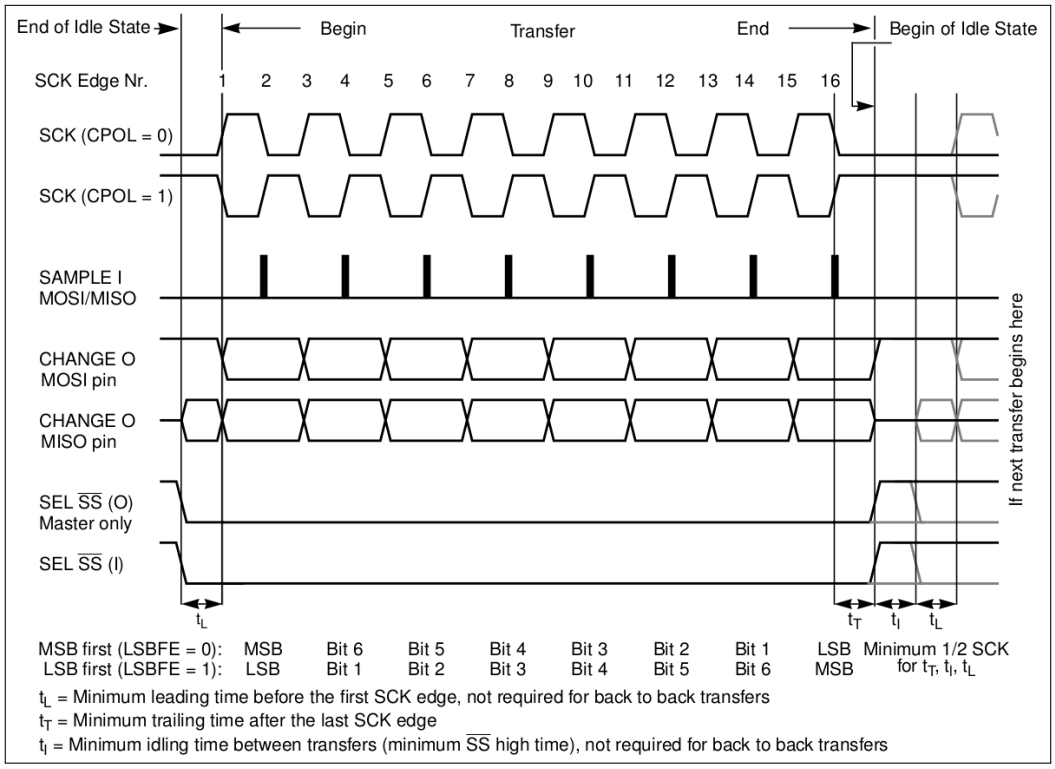


图 20-14. SPI 时钟模式 1 和时钟模式 3

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。

20.7.2 主机模式下的时钟控制

GP-SPI2 主机支持多种 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 20-16，可根据应用的路径延迟进行更改。

表 20-16. 主机模式下的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。

20.7.3 从机模式下的时钟控制

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 20-17。

表 20-17. 从机模式下的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

20.8 GP-SPI2 时序补偿

SPI 输入输出信号可通过 GPIO 矩阵或 IO MUX 映射到芯片管脚，但 IO MUX 不支持时序调整。输入输出数据在 GPIO 矩阵模块中，可在上升沿或下降沿延迟 1、或 2 个 APB_CLK 周期。更多寄存器配置信息，见章节 *5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)*。

在 GP-SPI2 从机模式下，如果寄存器 SPI_SLAVE_REG 中 SPI_RSCK_DATA_OUT 置 1，则在锁存沿发送输出数据，即提前半个 SPI 时钟周期。上述功能可用于从机模式时序补偿。

20.9 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 从机模式下，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 从机模式下，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 从机模式下，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 从机模式下，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 从机模式下，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 从机模式下，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：从机模式下，Rd_DMA 传输结束即触发此中断。
- SPI_SLV_WR_DMA_DONE_INT：从机模式下，Wr_DMA 传输结束即触发此中断。
- SPI_SLV_RD_BUF_DONE_INT：从机模式下，Rd_BUF 传输结束即触发此中断。
- SPI_SLV_WR_BUF_DONE_INT：从机模式下，Wr_BUF 传输结束即触发此中断。
- SPI_TRANS_DONE_INT：主从机模式下，SPI 总线传输结束均会触发此中断。
- SPI_DMA_SEG_TRANS_DONE_INT：GP-SPI2 从机连续传输模式下，End_SEG_TRANS 传输结束即触发此中断。主机模式下，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模式下，CONF buffer 中的 Magic 值有误即触发此中断。
- SPI_MST_RX_AFIFO_WFULL_ERR_INT：GP-SPI2 主机模式下，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 主机模式下，如果发生 TX AFIFO read-empty 错误即触发此中断。
- SPI_SLV_CMD_ERR_INT：GP-SPI2 从机模式下，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。

主机模式和从机模式分别用到的中断

表 20-18 和表 20-19 分别列出了 GP-SPI2 在主机模式下和从机模式下用到的中断。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 20-18. GP-SPI2 主机模式下用到的中断

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
	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 20-19. GP-SPI2 从机模式下用到的中断

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]

见下页

表 20-19 – 接上页

传输类型	通信模式	控制方式	中断

[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 End_SEG_TRAN 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。

20.10 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 3 *系统和存储器* 中的表 3-3。

请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

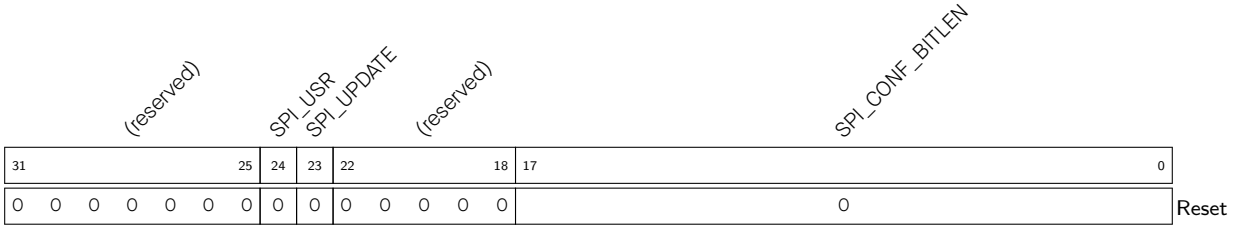
名称	描述	地址	访问
自定义控制寄存器			
SPI_CMD_REG	命令控制寄存器	0x0000	varies
SPI_ADDR_REG	地址值寄存器	0x0004	R/W
SPI_USER_REG	SPI 用户控制寄存器	0x0010	varies
SPI_USER1_REG	SPI 用户控制寄存器 1	0x0014	R/W
SPI_USER2_REG	SPI 用户控制寄存器 2	0x0018	R/W
控制和配置寄存器			
SPI_CTRL_REG	SPI 控制寄存器	0x0008	R/W
SPI_MS_DLEN_REG	SPI 数据位长控制寄存器	0x001C	R/W
SPI_MISC_REG	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
中断寄存器			
SPI_DMA_INT_ENA_REG	SPI DMA 中断使能寄存器	0x0034	R/W

名称	描述	地址	访问
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
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

20.11 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 3 系统和存储器 中的表 3-3。

Register 20.1. SPI_CMD_REG (0x0000)

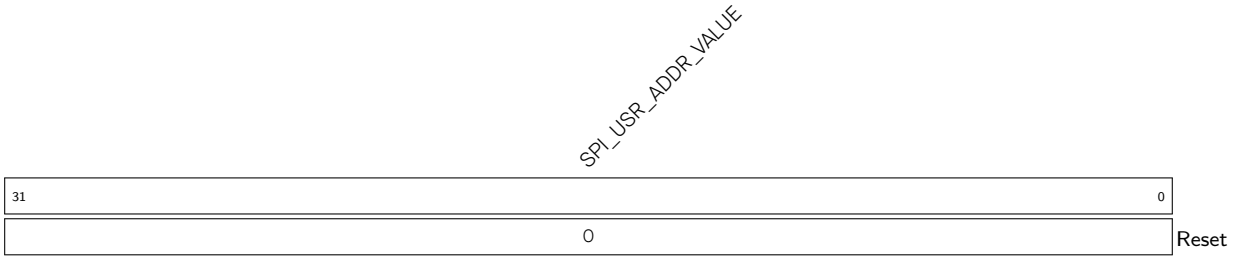


SPI_CONF_BITLEN 定义 SPI CONF 阶段的 SPI CLK 周期。可在 CONF 阶段配置。(R/W)

SPI_UPDATE 置位此位，将 SPI 寄存器从 APB 时钟域同步到 SPI 模块时钟域。该位仅用于 SPI 主机模式。(WT)

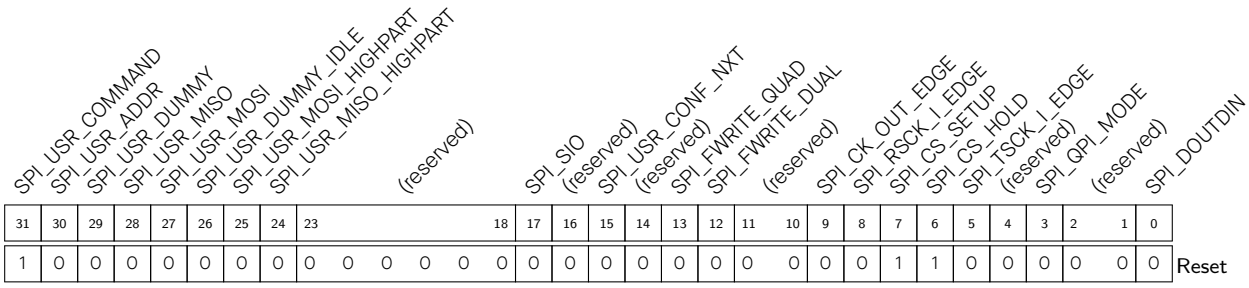
SPI_USR 使能用户自定义命令。置位此位将触发一次 SPI 操作。操作结束后此位被自动清零。1：使能此功能；0：禁用此功能。CONF_buf 不可更改该配置。(R/W/SC)

Register 20.2. SPI_ADDR_REG (0x0004)



SPI_USR_ADDR_VALUE 从机地址。可在 CONF 阶段配置。(R/W)

Register 20.3. SPI_USER_REG (0x0010)



SPI_DOUTDIN 置位此位，使能全双工通信。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_QPI_MODE 1：使能 QPI 模式。0：禁用 QPI 模式。SPI 主机模式和从机模式均支持该配置。可在 CONF 阶段配置。(R/W/SS/SC)

SPI_TSCK_I_EDGE 在从机模式下，此位可用于更改 TSCK 极性。0：TSCK = SPI_CK_I；1：TSCK = !SPI_CK_I。(R/W)

SPI_CS_HOLD 当 SPI 处于完成 (DONE) 阶段时，保持 SPI CS 拉低。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_CS_SETUP 当 SPI 处于准备 (PREP) 阶段时，使能 SPI CS。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_RSCK_I_EDGE 在从机模式下，此位可用于更改 RSCK 极性。0：RSCK = !SPI_CK_I；1：RSCK = SPI_CK_I。(R/W)

SPI_CK_OUT_EDGE 该位与 SPI_CK_IDLE_EDGE 一起用于控制 SPI 时钟模式。可在 CONF 阶段配置。更多信息见章节 20.7.2。(R/W)

SPI_FWRITE_DUAL 在写操作 (DOUT) 阶段，读数据的方式为 2-bit 方式。可在 CONF 阶段配置。(R/W)

SPI_FWRITE_QUAD 在写操作 (DOUT) 阶段，读数据的方式为 4-bit 方式。可在 CONF 阶段配置。(R/W)

SPI_USR_CONF_NXT 使能下一次传输事务的 CONF 阶段。可在 CONF 阶段配置。(R/W)

- 置位此位，则本次分段配置传输继续进行，开始下一次传输事务。
- 清除此位，则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。

SPI_SIO 置位此位，使能 3 线半双工通信，其中 MOSI 和 MISO 信号共享一个管脚。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_USR_MISO_HIGHPART 在读数据阶段，仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

见下页

Register 20.3. SPI_USER_REG (0x0010)

接上页

SPI_USR_MOSI_HIGHPART 在写数据阶段，仅访问高位 buffer：SPI_W8_REG ~ SPI_W15_REG。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_USR_DUMMY_IDLE 置位此位，在 DUMMY 阶段禁用 SPI 时钟。可在 CONF 阶段配置。(R/W)

SPI_USR_MOSI 置位此位，使能一次操作的写数据 (DOUT) 阶段。可在 CONF 阶段配置。(R/W)

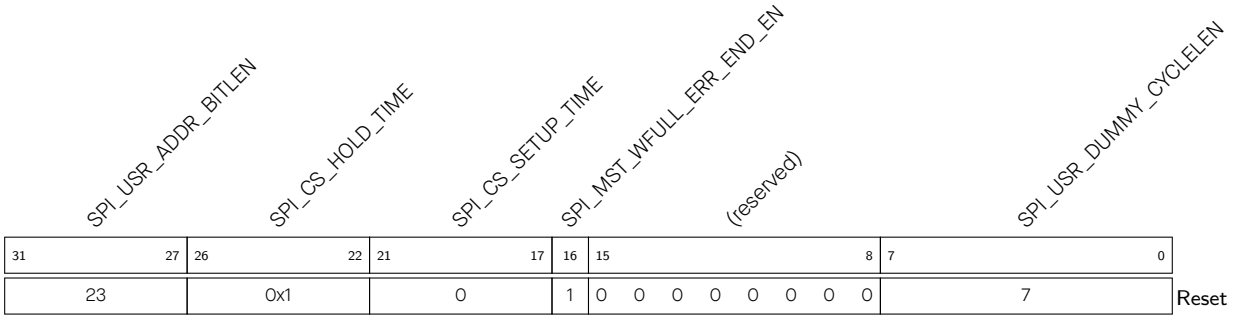
SPI_USR_MISO 置位此位，使能一次操作的读数据 (DIN) 阶段。可在 CONF 阶段配置。(R/W)

SPI_USR_DUMMY 置位此位，使能一次操作的 DUMMY 阶段。可在 CONF 阶段配置。(R/W)

SPI_USR_ADDR 置位此位，使能一次操作的地址 (ADDR) 阶段。可在 CONF 阶段配置。(R/W)

SPI_USR_COMMAND 置位此位，使能一次操作的命令 (CMD) 阶段。可在 CONF 阶段配置。(R/W)

Register 20.4. SPI_USER1_REG (0x0014)



SPI_USR_DUMMY_CYCLELEN 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)

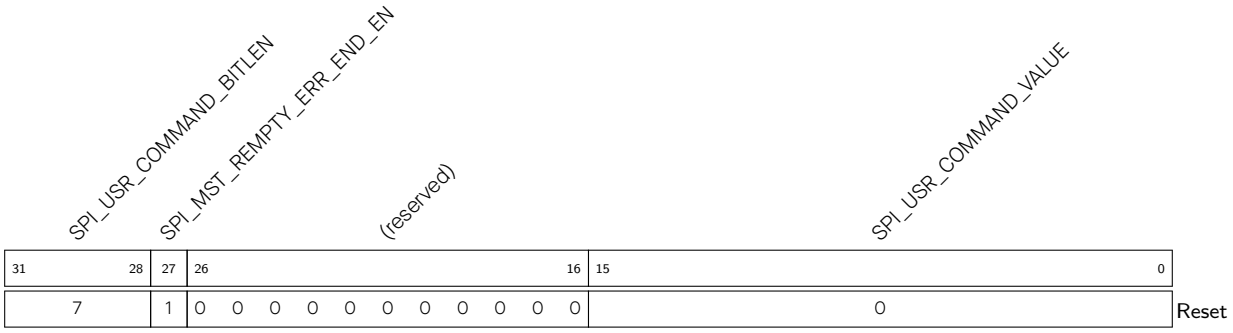
SPI_MST_WFULL_ERR_END_EN 1: 在 GP-SPI2 主机全双工或半双工模式下，如果发生 SPI RX AFIFO 满错误，则 SPI 传输将终止。0：在 GP-SPI2 主机全双工或半双工模式下，如果发生 SPI RX AFIFO 满错误，SPI 传输将不被终止。(R/W)

SPI_CS_SETUP_TIME 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 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 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 20.5. SPI_USER2_REG (0x0018)

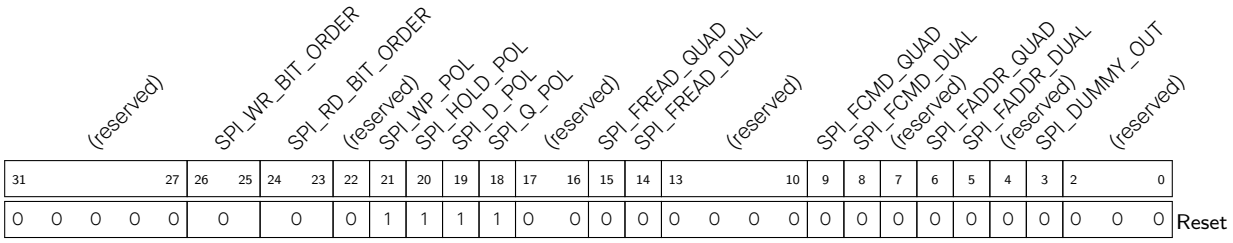


SPI_USR_COMMAND_VALUE 命令值。可在 CONF 阶段配置。(R/W)

SPI_MST_REMPTY_ERR_END_EN 1：在 GP-SPI2 主机全双工或半双工模式下，如果发生 SPI TX AFIFO 空错误，则 SPI 传输将终止。0：在 GP-SPI2 主机全双工或半双工模式下，如果发生 SPI TX AFIFO 空错误，则 SPI 传输将不会被终止。(R/W)

SPI_USR_COMMAND_BITLEN 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 20.6. SPI_CTRL_REG (0x0008)



SPI_DUMMY_OUT 0：在 DUMMY 阶段，不输出 FSPI 总线信号。1：在 DUMMY 阶段，输出 FSPI 总线信号。可在 CONF 阶段配置。(R/W)

SPI_FADDR_DUAL 在地址 (ADDR) 阶段采用 2-bit 模式。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_FADDR_QUAD 在地址 (ADDR) 阶段采用 4-bit 模式。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_FCMD_DUAL 在命令 (CMD) 阶段采用 2-bit 模式。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_FCMD_QUAD 在命令 (CMD) 阶段采用 4-bit 模式。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_FREAD_DUAL 在读数据 (DIN) 阶段采用 2-bit 模式。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_FREAD_QUAD 在读数据 (DIN) 阶段采用 4-bit 模式。1：使能此功能；0：禁用此功能。可在 CONF 阶段配置。(R/W)

SPI_Q_POL 此位用于设置 MISO 的极性。1：高；0：低。可在 CONF 阶段配置。(R/W)

SPI_D_POL 此位用于设置 MOSI 的极性。1：高；0：低。可在 CONF 阶段配置。(R/W)

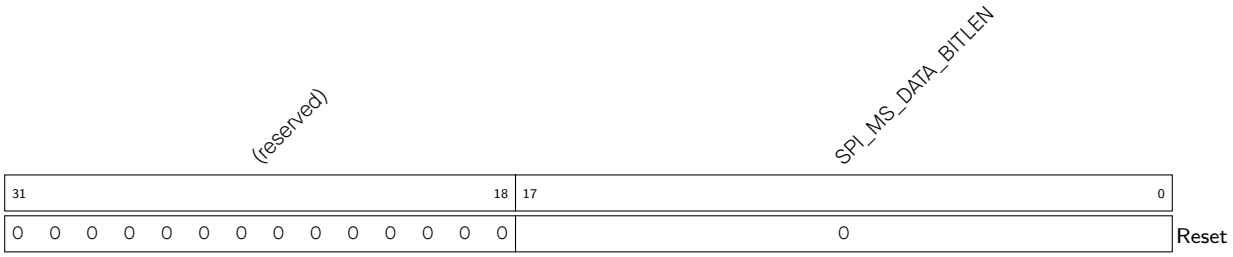
SPI_HOLD_POL 此位用于设置在 SPI 空闲状态下，SPI_HOLD 的输出值。1：输出高电平；0：输出低电平。可在 CONF 阶段配置。(R/W)

SPI_WP_POL 此位用于设置在 SPI 空闲状态下，WP 信号的输出值。1：输出高电平；0：输出低电平。可在 CONF 阶段配置。(R/W)

SPI_RD_BIT_ORDER 在读数据 (MISO) 阶段，1：先读低有效位；0：先读高有效位。可在 CONF 阶段配置。(R/W)

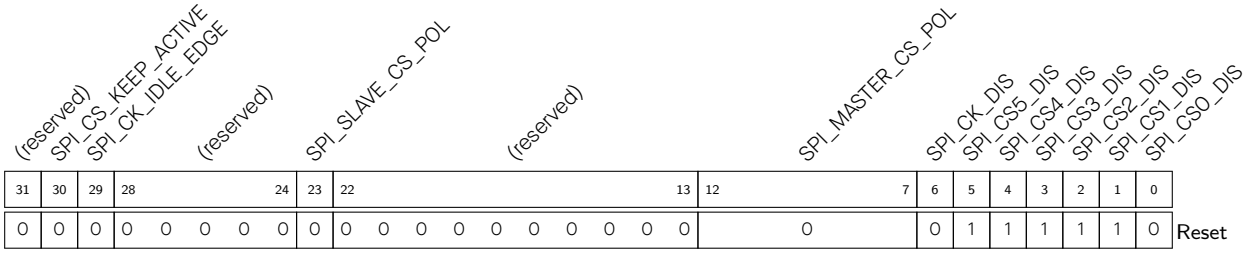
SPI_WR_BIT_ORDER 在命令 (CMD)、地址 (ADDR) 和写数据 (MOSI) 阶段，1：先读低有效位；0：先读高有效位。可在 CONF 阶段配置。(R/W)

Register 20.7. SPI_MS_DLEN_REG (0x001C)



SPI_MS_DATA_BITLEN 该字段用于配置主机模式下 DMA 控制或 CPU 控制的 SPI 传输的数据位长。也可用于配置从机模式下 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 20.8. SPI_MISC_REG (0x0020)



SPI_CS0_DIS SPI CS0 管脚使能。1：禁用 CS0，0：SPI CS0 信号来自 CS0 管脚或输出至 CS0 管脚。可在 CONF 阶段配置。(R/W)

SPI_CS1_DIS SPI CS1 管脚使能。1：禁用 CS1，0：SPI CS1 信号来自 CS1 管脚或输出至 CS1 管脚。可在 CONF 阶段配置。(R/W)

SPI_CS2_DIS SPI CS2 管脚使能。1：禁用 CS2，0：SPI CS2 信号来自 CS2 管脚或输出至 CS2 管脚。可在 CONF 阶段配置。(R/W)

SPI_CS3_DIS SPI CS3 管脚使能。1：禁用 CS3，0：SPI CS3 信号来自 CS3 管脚或输出至 CS3 管脚。可在 CONF 阶段配置。(R/W)

SPI_CS4_DIS SPI CS4 管脚使能。1：禁用 CS4，0：SPI CS4 信号来自 CS4 管脚或输出至 CS4 管脚。可在 CONF 阶段配置。(R/W)

SPI_CS5_DIS SPI CS5 管脚使能。1：禁用 CS5，0：SPI CS5 信号来自 CS5 管脚或输出至 CS5 管脚。可在 CONF 阶段配置。(R/W)

SPI_CK_DIS 1：停止 SPI_CLK 输出信号；0：使能 SPI_CLK 输出信号。可在 CONF 阶段配置。(R/W)

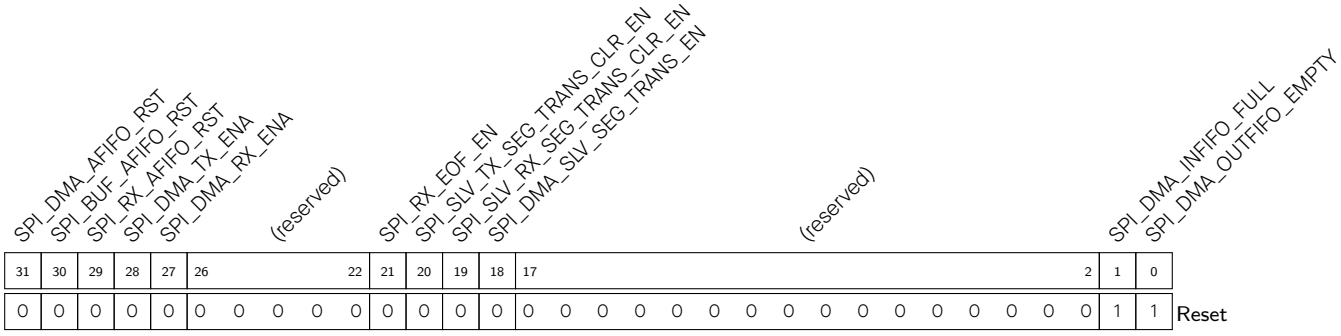
SPI_MASTER_CS_POL 主机模式下，SPI_MASTER_CS_POL[*i*] 用于配置 SPI CS*i* 的极性，*i* = 0 ~ 5。0：CS*i* 低电平有效。1：CS*i* 高电平有效。可在 CONF 阶段配置。(R/W)

SPI_SLAVE_CS_POL 选择 SPI 从机输入信号 CS 的极性。1：反相；0：保持不变。可在 CONF 阶段配置。(R/W)

SPI_CK_IDLE_EDGE 1：SPI CLK 线在 GP-SPI2 空闲状态时保持高电平；0：SPI CLK 线在 GP-SPI2 空闲状态时保持低电平。可在 CONF 阶段配置。(R/W)

SPI_CS_KEEP_ACTIVE 置位此位，则 SPI CS 线保持低电平。可在 CONF 阶段配置。(R/W)

Register 20.9. SPI_DMA_CONF_REG (0x0030)



SPI_DMA_OUTFIFO_EMPTY 记录 DMA TX FIFO 的状态。1：DMA TX FIFO 尚未就绪，不能发送数据。0：DMA TX FIFO 已就绪，可以发送数据。(RO)

SPI_DMA_INFIFO_FULL 记录 DMA RX FIFO 的状态。1：DMA RX FIFO 尚未就绪，不能接收数据。0：DMA RX FIFO 已就绪，可以接收数据。(RO)

SPI_DMA_SLV_SEG_TRANS_EN 1：使能半双工通信方式下，DMA 控制的从机连续传输。0：禁用。(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：(R/W)

- 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。

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：(R/W)

- 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。

SPI_RX_EOF_EN 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 置位此位，使能 SPI DMA 控制的接收数据模式。(R/W)

SPI_DMA_TX_ENA 置位此位，使能 SPI DMA 控制的发送数据模式。(R/W)

见下页

Register 20.9. SPI_DMA_CONF_REG (0x0030)

接上页

SPI_RX_AFIFO_RST 置位此位，复位图 20-4 和图 20-5 中的 spi_rx_afifo。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 置位此位，复位图 20-4 和图 20-5 中的 buf_tx_afifo。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 置位此位，复位图 20-4 和图 20-5 中的 dma_tx_afifo。dma_tx_afifo 在 DMA 控制的从机传输中用于发送数据。(WT)

Register 20.10. SPI_SLAVE_REG (0x00E0)

Field	Bits	Reset
(reserved)	31:29	0 0 0
SPI_USR_CONF	28	0
SPI_SOFT_RESET	27	0
SPI_SLAVE_MODE	26	0
SPI_DMA_SEG_MAGIC_VALUE	25:22	10
(reserved)	21:12	0 0 0 0 0 0 0 0 0 0
SPI_SLV_WRBUF_BITLEN_EN	11	0
SPI_SLV_RDBUF_BITLEN_EN	10	0
SPI_SLV_WRDMA_BITLEN_EN	9	0
SPI_SLV_RDDMA_BITLEN_EN	8	0
(reserved)	7:4	0 0 0 0
SPI_RSCK_DATA_OUT	3	0
SPI_CLK_MODE_13	2	0
SPI_CLK_MODE	1:0	0

SPI_CLK_MODE SPI 时钟模式控制位。可在 CONF 阶段配置。(R/W)

- 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。

SPI_CLK_MODE_13 配置时钟模式。(R/W)

- 1：支持 SPI 时钟模式 1 或 3，见表 20-17。
- 0：支持 SPI 时钟模式 0 或 2，见表 20-17。

SPI_RSCK_DATA_OUT TSCK 与 RSCK 相同时，将节省半个周期。1：在 RSCK 上升沿输出数据；0：在 TSCK 上升沿输出数据。(R/W)

SPI_SLV_RDDMA_BITLEN_EN 置位此位，则在 DMA 控制的 Rd_DMA 传输过程中，SPI_SLV_DATA_BITLEN 将用于存储主机读取从机的数据位长。(R/W)

SPI_SLV_WRDMA_BITLEN_EN 置位此位，则在 DMA 控制的 Wr_DMA 传输过程中，SPI_SLV_DATA_BITLEN 将用于存储主机向从机写数据的位长。(R/W)

SPI_SLV_RDBUF_BITLEN_EN 置位此位，则在 CPU 控制的 Rd_BUF 传输过程中，SPI_SLV_DATA_BITLEN 将用于存储主机读取从机的数据位长。(R/W)

SPI_SLV_WRBUF_BITLEN_EN 置位此位，则在 CPU 控制的 Wr_BUF 传输过程中，SPI_SLV_DATA_BITLEN 将用于存储主机向从机写数据的位长。(R/W)

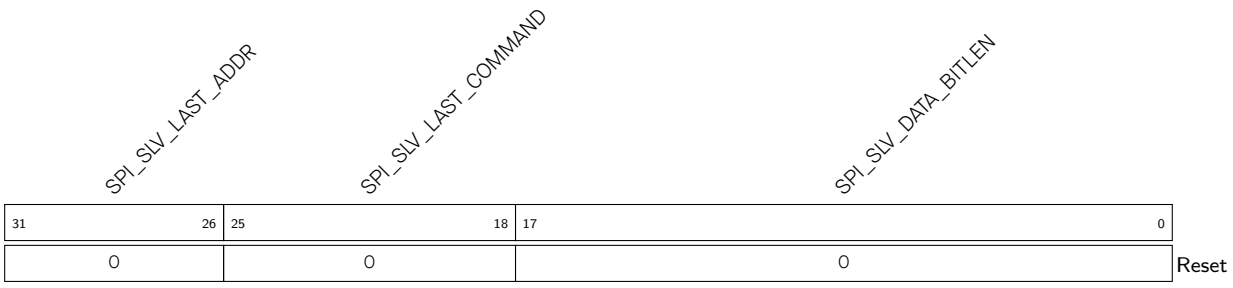
SPI_DMA_SEG_MAGIC_VALUE 在 DMA 控制的分段配置传输中，用于配置位图表的 Magic 值。(R/W)

SPI_SLAVE_MODE 配置 SPI 工作模式。1：从机模式；0：主机模式。(R/W)

SPI_SOFT_RESET 软件置位使能位。置位此位，可复位 SPI 时钟线、CS 线和数据线。可在 CONF 阶段配置。(WT)

SPI_USR_CONF 1：置位此位，使能当前 DMA 控制分段配置传输的 CONF 阶段，开始分段配置传输。0：清除此位，则表明当前传输不是分段配置传输。(R/W)

Register 20.11. SPI_SLAVE1_REG (0x00E4)

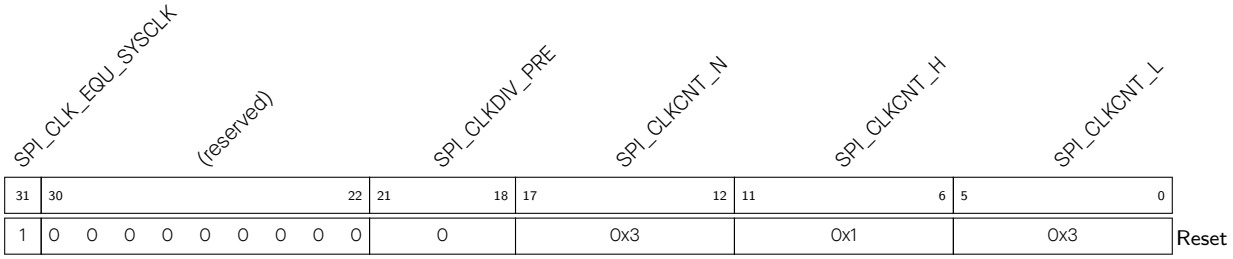


SPI_SLV_DATA_BITLEN 在 SPI 从机全双工和半双工传输中，配置传输的数据位长。(R/W/SS)

SPI_SLV_LAST_COMMAND 从机模式下的命令值。(R/W/SS)

SPI_SLV_LAST_ADDR 从机模式下的地址值。(R/W/SS)

Register 20.12. SPI_CLOCK_REG (0x000C)



SPI_CLKCNT_L 主机模式下，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)

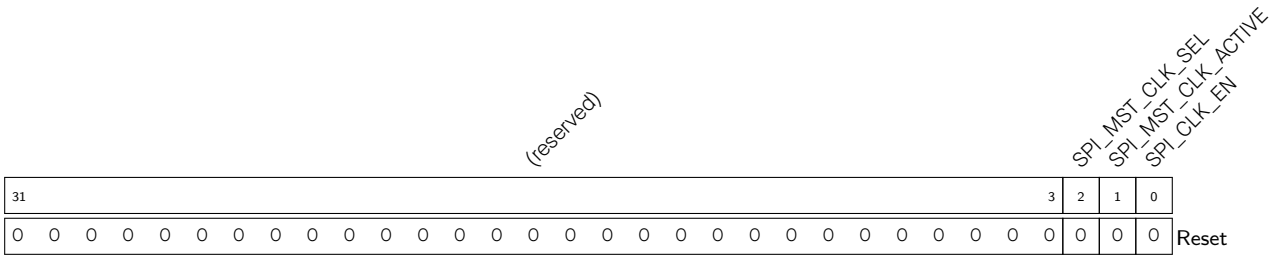
SPI_CLKCNT_H 主机模式下，此字段用于配置 SPI_CLK（高电平）的占空比。建议将此值配置为 floor ((SPI_CLKCNT_N + 1)/2 - 1)。floor() 表示向下取整值，例如 floor(2.2) = 2。从机模式下，必须为 0。可在 CONF 阶段配置。(R/W)

SPI_CLKCNT_N 主机模式下，SPI_CLK 的分频系数。因此 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 主机模式下，SPI_CLK 的预分频系数。可在 CONF 阶段配置。(R/W)

SPI_CLK_EQU_SYSCLK 主机模式下，1：SPI_CLK 与 clk_spi_mst 频率相同。0：SPI_CLK 为 clk_spi_mst 的分频时钟。可在 CONF 阶段配置。(R/W)

Register 20.13. SPI_CLK_GATE_REG (0x00E8)



SPI_CLK_EN 置位此位，使能时钟门控。(R/W)

SPI_MST_CLK_ACTIVE 置位此位，SPI 模块时钟上电。(R/W)

SPI_MST_CLK_SEL 该位用于选择主机模式下 SPI 模块的时钟源。1：选择 PLL_F80M_CLK；0：选择 XTAL_CLK。(R/W)

Register 20.14. SPI_DMA_INT_ENA_REG (0x0034)

SPI_DMA_INFIFO_FULL_ERR_INT_ENA SPI_DMA_INFIFO_FULL_ERR_INT 的中断使能位。(R/W)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_ENA SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断使能位。(R/W)

SPI_SLV_EX_QPI_INT_ENA SPI_SLV_EX_QPI_INT 的中断使能位。(R/W)

SPI_SLV_EN_QPI_INT_ENA SPI_SLV_EN_QPI_INT 的中断使能位。(R/W)

SPI_SLV_CMD7_INT_ENA SPI_SLV_CMD7_INT 的中断使能位。(R/W)

SPI_SLV_CMD8_INT_ENA SPI_SLV_CMD8_INT 的中断使能位。(R/W)

SPI_SLV_CMD9_INT_ENA SPI_SLV_CMD9_INT 的中断使能位。(R/W)

SPI_SLV_CMDA_INT_ENA SPI_SLV_CMDA_INT 的中断使能位。(R/W)

SPI_SLV_RD_DMA_DONE_INT_ENA SPI_SLV_RD_DMA_DONE_INT 的中断使能位。(R/W)

SPI_SLV_WR_DMA_DONE_INT_ENA SPI_SLV_WR_DMA_DONE_INT 的中断使能位。(R/W)

SPI_SLV_RD_BUF_DONE_INT_ENA SPI_SLV_RD_BUF_DONE_INT 的中断使能位。(R/W)

SPI_SLV_WR_BUF_DONE_INT_ENA SPI_SLV_WR_BUF_DONE_INT 的中断使能位。(R/W)

SPI_TRANS_DONE_INT_ENA SPI_TRANS_DONE_INT 的中断使能位。(R/W)

SPI_DMA_SEG_TRANS_DONE_INT_ENA SPI_DMA_SEG_TRANS_DONE_INT 的中断使能位。(R/W)

SPI_SEG_MAGIC_ERR_INT_ENA SPI_SEG_MAGIC_ERR_INT 的中断使能位。(R/W)

见下页

Register 20.14. SPI_DMA_INT_ENA_REG (0x0034)

接上页

SPI_SLV_CMD_ERR_INT_ENA SPI_SLV_CMD_ERR_INT 的中断使能位。(R/W)

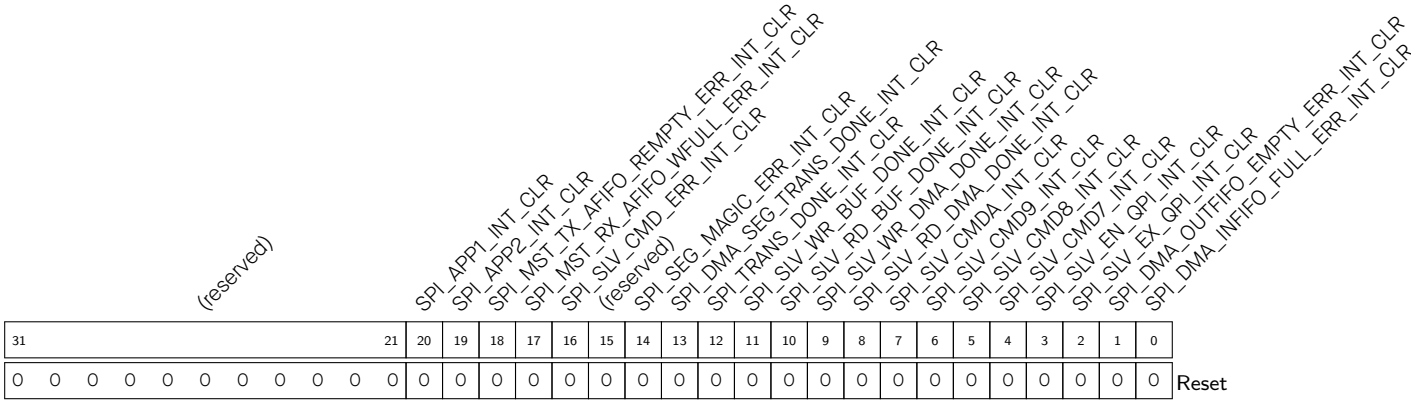
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ENA SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断使能位。(R/W)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_ENA SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断使能位。(R/W)

SPI_APP2_INT_ENA SPI_APP2_INT 的中断使能位。(R/W)

SPI_APP1_INT_ENA SPI_APP1_INT 的中断使能位。(R/W)

Register 20.15. SPI_DMA_INT_CLR_REG (0x0038)



SPI_DMA_INFIFO_FULL_ERR_INT_CLR SPI_DMA_INFIFO_FULL_ERR_INT 的中断清除位。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_CLR SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断清除位。(WT)

SPI_SLV_EX_QPI_INT_CLR SPI_SLV_EX_QPI_INT 的中断清除位。(WT)

SPI_SLV_EN_QPI_INT_CLR SPI_SLV_EN_QPI_INT 的中断清除位。(WT)

SPI_SLV_CMD7_INT_CLR SPI_SLV_CMD7_INT 的中断清除位。(WT)

SPI_SLV_CMD8_INT_CLR SPI_SLV_CMD8_INT 的中断清除位。(WT)

SPI_SLV_CMD9_INT_CLR SPI_SLV_CMD9_INT 的中断清除位。(WT)

SPI_SLV_CMDA_INT_CLR SPI_SLV_CMDA_INT 的中断清除位。(WT)

SPI_SLV_RD_DMA_DONE_INT_CLR SPI_SLV_RD_DMA_DONE_INT 的中断清除位。(WT)

SPI_SLV_WR_DMA_DONE_INT_CLR SPI_SLV_WR_DMA_DONE_INT 的中断清除位。(WT)

SPI_SLV_RD_BUF_DONE_INT_CLR SPI_SLV_RD_BUF_DONE_INT 的中断清除位。(WT)

SPI_SLV_WR_BUF_DONE_INT_CLR SPI_SLV_WR_BUF_DONE_INT 的中断清除位。(WT)

SPI_TRANS_DONE_INT_CLR SPI_TRANS_DONE_INT 的中断清除位。(WT)

SPI_DMA_SEG_TRANS_DONE_INT_CLR SPI_DMA_SEG_TRANS_DONE_INT 的中断清除位。(WT)

SPI_SEG_MAGIC_ERR_INT_CLR SPI_SEG_MAGIC_ERR_INT 的中断清除位。(WT)

见下页

Register 20.15. SPI_DMA_INT_CLR_REG (0x0038)

接上页

SPI_SLV_CMD_ERR_INT_CLR SPI_SLV_CMD_ERR_INT 的中断清除位。(WT)

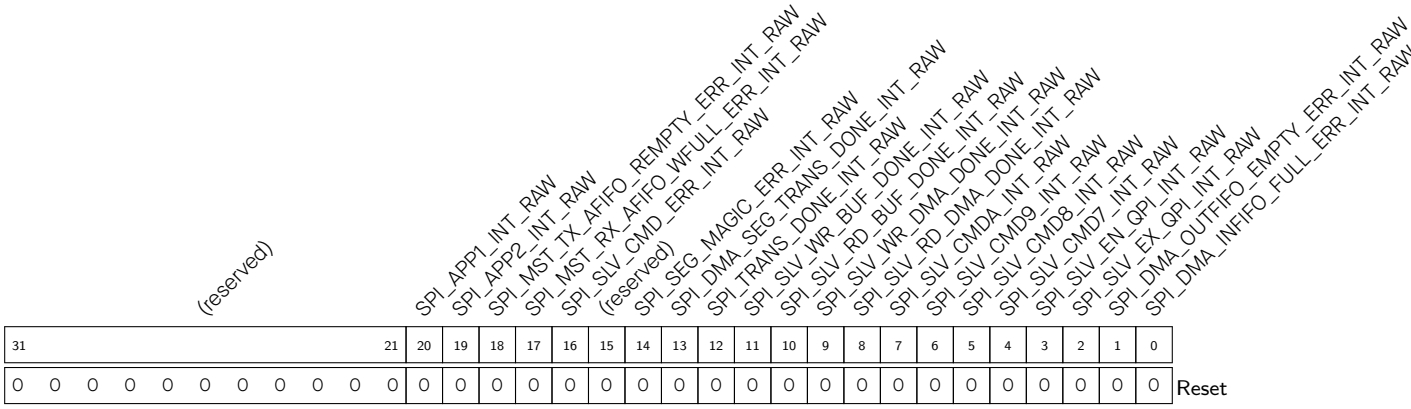
SPI_MST_RX_AFIFO_WFULL_ERR_INT_CLR SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断清除位。(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_CLR SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断清除位。(WT)

SPI_APP2_INT_CLR SPI_APP2_INT 的中断清除位。(WT)

SPI_APP1_INT_CLR SPI_APP1_INT 的中断清除位。(WT)

Register 20.16. SPI_DMA_INT_RAW_REG (0x003C)



SPI_DMA_INFIFO_FULL_ERR_INT_RAW SPI_DMA_INFIFO_FULL_ERR_INT 的原始中断位。(R/W/WTC/SS)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_RAW SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_EX_QPI_INT_RAW SPI_SLV_EX_QPI_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_EN_QPI_INT_RAW SPI_SLV_EN_QPI_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_CMD7_INT_RAW SPI_SLV_CMD7_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_CMD8_INT_RAW SPI_SLV_CMD8_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_CMD9_INT_RAW SPI_SLV_CMD9_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_CMDA_INT_RAW SPI_SLV_CMDA_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_RD_DMA_DONE_INT_RAW SPI_SLV_RD_DMA_DONE_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_WR_DMA_DONE_INT_RAW SPI_SLV_WR_DMA_DONE_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_RD_BUF_DONE_INT_RAW SPI_SLV_RD_BUF_DONE_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_WR_BUF_DONE_INT_RAW SPI_SLV_WR_BUF_DONE_INT 的原始中断位。(R/W/WTC/SS)

SPI_TRANS_DONE_INT_RAW SPI_TRANS_DONE_INT 的原始中断位。(R/W/WTC/SS)

见下页

Register 20.16. SPI_DMA_INT_RAW_REG (0x003C)

接上页

SPI_DMA_SEG_TRANS_DONE_INT_RAW SPI_DMA_SEG_TRANS_DONE_INT 的原始中断位。(R/W/WTC/SS)

SPI_SEG_MAGIC_ERR_INT_RAW SPI_SEG_MAGIC_ERR_INT 的原始中断位。(R/W/WTC/SS)

SPI_SLV_CMD_ERR_INT_RAW SPI_SLV_CMD_ERR_INT 的原始中断位。(R/W/WTC/SS)

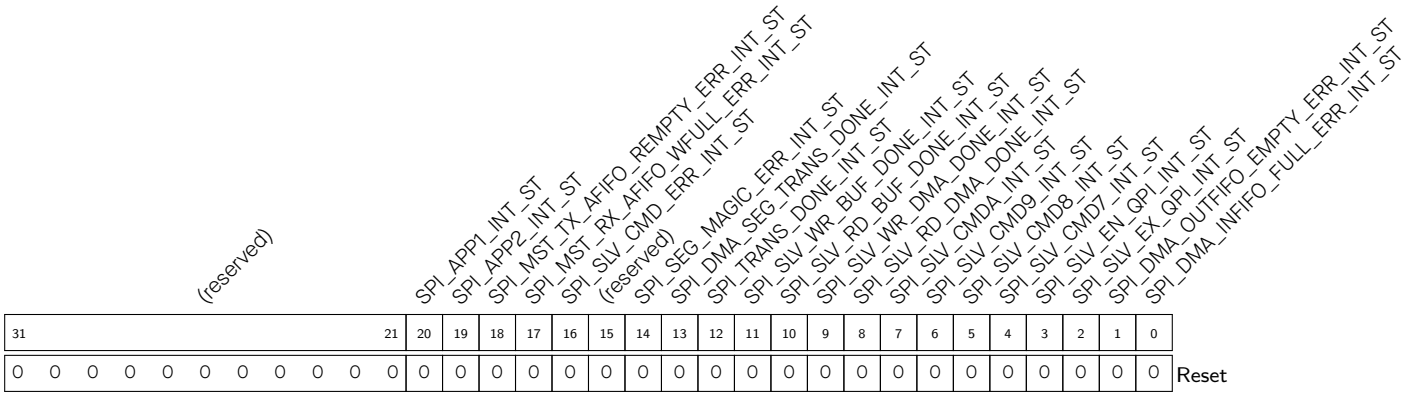
SPI_MST_RX_AFIFO_WFULL_ERR_INT_RAW SPI_MST_RX_AFIFO_WFULL_ERR_INT 的原始中断位。(R/W/WTC/SS)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_RAW SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的原始中断位。(R/W/WTC/SS)

SPI_APP2_INT_RAW SPI_APP2_INT 的原始中断位。该值仅由应用控制。(R/W/WTC)

SPI_APP1_INT_RAW SPI_APP1_INT 的原始中断位。该值仅由应用控制。(R/W/WTC)

Register 20.17. 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)

见下页

Register 20.17. SPI_DMA_INT_ST_REG (0x0040)

接上页

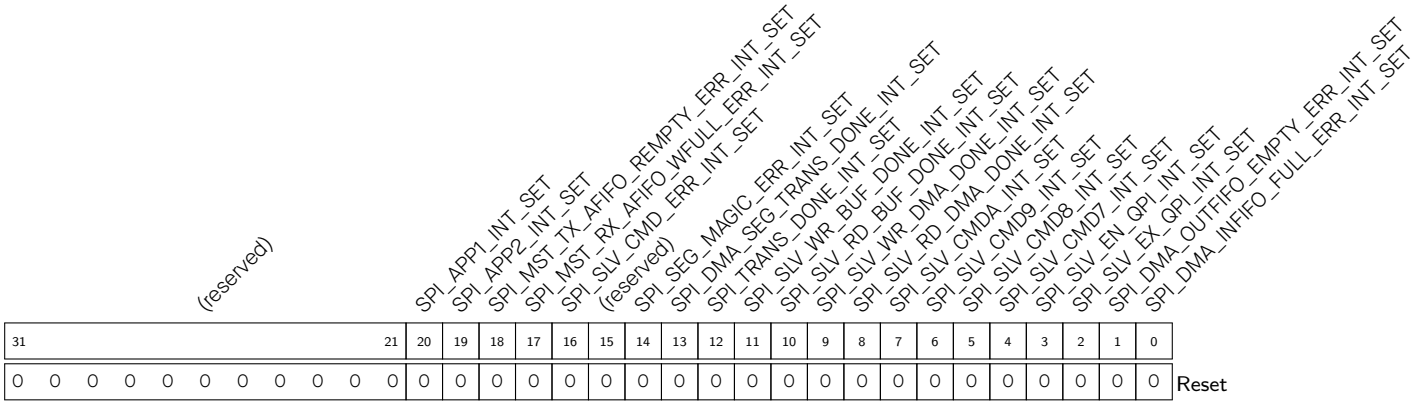
SPI_MST_RX_AFIFO_WFULL_ERR_INT_ST SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断状态位。(RO)

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 20.18. SPI_DMA_INT_SET_REG (0x0044)



SPI_DMA_INFIFO_FULL_ERR_INT_SET SPI_DMA_INFIFO_FULL_ERR_INT 的中断软件置位。(WT)

SPI_DMA_OUTFIFO_EMPTY_ERR_INT_SET SPI_DMA_OUTFIFO_EMPTY_ERR_INT 的中断软件置位。(WT)

SPI_SLV_EX_QPI_INT_SET SPI_SLV_EX_QPI_INT 的中断软件置位。(WT)

SPI_SLV_EN_QPI_INT_SET SPI_SLV_EN_QPI_INT 的中断软件置位。(WT)

SPI_SLV_CMD7_INT_SET SPI_SLV_CMD7_INT 的中断软件置位。(WT)

SPI_SLV_CMD8_INT_SET SPI_SLV_CMD8_INT 的中断软件置位。(WT)

SPI_SLV_CMD9_INT_SET SPI_SLV_CMD9_INT 的中断软件置位。(WT)

SPI_SLV_CMDA_INT_SET SPI_SLV_CMDA_INT 的中断软件置位。(WT)

见下页

Register 20.18. SPI_DMA_INT_SET_REG (0x0044)

接上页

SPI_SLV_RD_DMA_DONE_INT_SET SPI_SLV_RD_DMA_DONE_INT 的中断软件置位。(WT)

SPI_SLV_WR_DMA_DONE_INT_SET SPI_SLV_WR_DMA_DONE_INT 的中断软件置位。(WT)

SPI_SLV_RD_BUF_DONE_INT_SET SPI_SLV_RD_BUF_DONE_INT 的中断软件置位。(WT)

SPI_SLV_WR_BUF_DONE_INT_SET SPI_SLV_WR_BUF_DONE_INT 的中断软件置位。(WT)

SPI_TRANS_DONE_INT_SET SPI_TRANS_DONE_INT 的中断软件置位。(WT)

SPI_DMA_SEG_TRANS_DONE_INT_SET SPI_DMA_SEG_TRANS_DONE_INT 的中断软件置位。(WT)

SPI_SEG_MAGIC_ERR_INT_SET SPI_SEG_MAGIC_ERR_INT 的中断软件置位。(WT)

SPI_SLV_CMD_ERR_INT_SET SPI_SLV_CMD_ERR_INT 的中断软件置位。(WT)

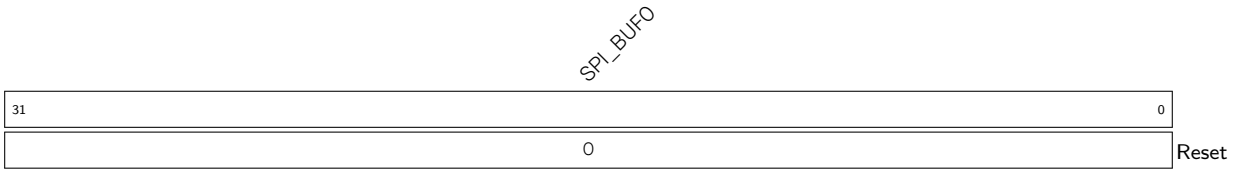
SPI_MST_RX_AFIFO_WFULL_ERR_INT_SET SPI_MST_RX_AFIFO_WFULL_ERR_INT 的中断软件置位。(WT)

SPI_MST_TX_AFIFO_REMPTY_ERR_INT_SET SPI_MST_TX_AFIFO_REMPTY_ERR_INT 的中断软件置位。(WT)

SPI_APP2_INT_SET SPI_APP2_INT 的中断软件置位。(WT)

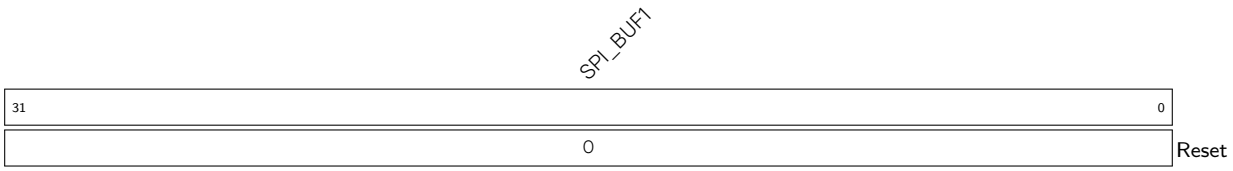
SPI_APP1_INT_SET SPI_APP1_INT 的中断软件置位。(WT)

Register 20.19. SPI_W0_REG (0x0098)



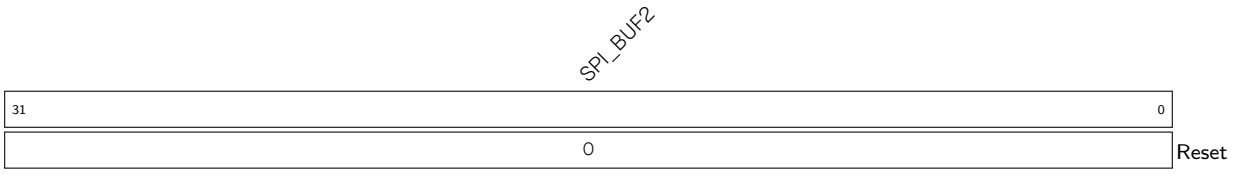
SPI_BUF0 数据 buffer 0，32 位。(R/W/SS)

Register 20.20. SPI_W1_REG (0x009C)



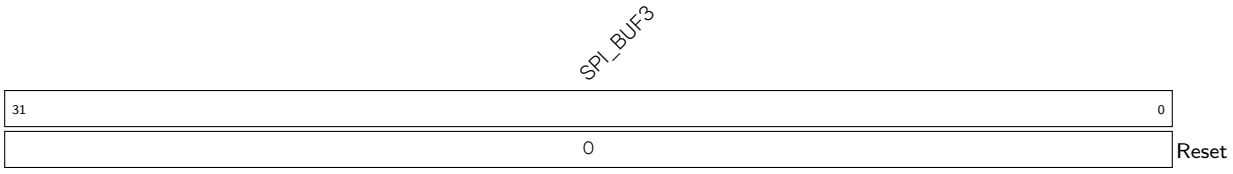
SPI_BUF1 数据 buffer 1，32 位。(R/W/SS)

Register 20.21. SPI_W2_REG (0x00A0)



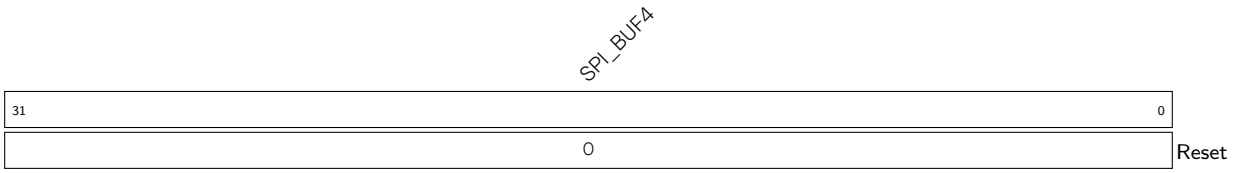
SPI_BUF2 数据 buffer 2，32 位。(R/W/SS)

Register 20.22. SPI_W3_REG (0x00A4)



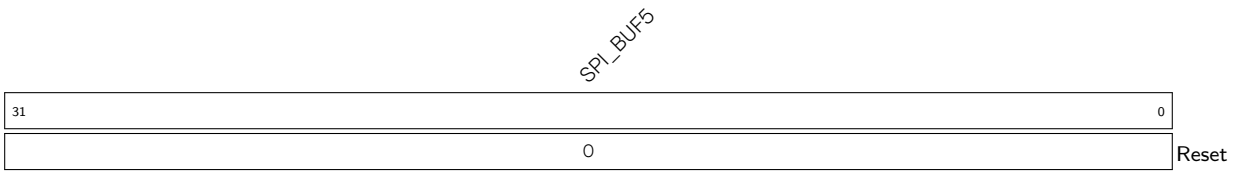
SPI_BUF3 数据 buffer 3，32 位。(R/W/SS)

Register 20.23. SPI_W4_REG (0x00A8)



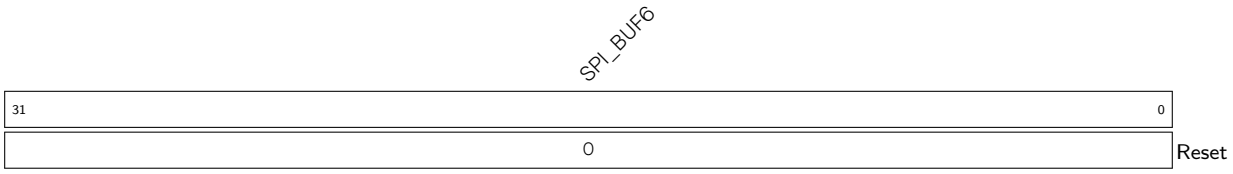
SPI_BUF4 数据 buffer 4，32 位。(R/W/SS)

Register 20.24. SPI_W5_REG (0x00AC)



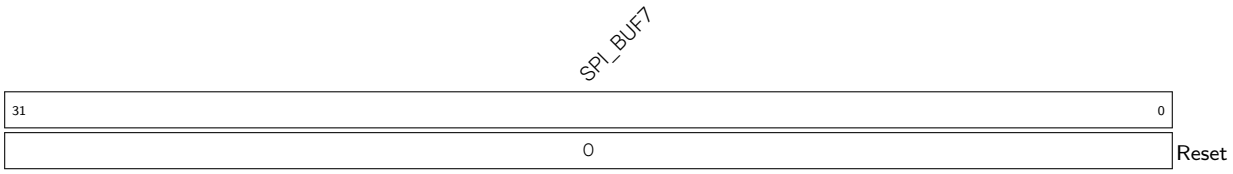
SPI_BUF5 数据 buffer 5，32 位。(R/W/SS)

Register 20.25. SPI_W6_REG (0x00B0)



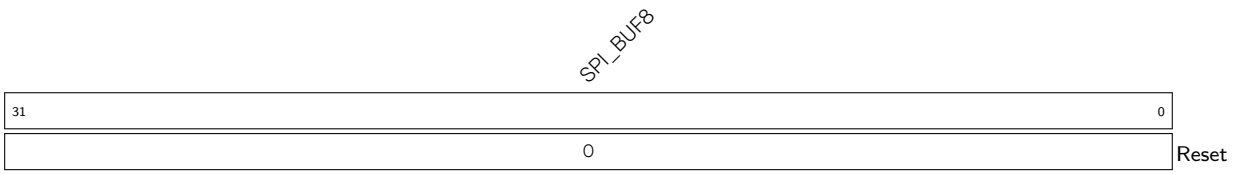
SPI_BUF6 数据 buffer 6，32 位。(R/W/SS)

Register 20.26. SPI_W7_REG (0x00B4)



SPI_BUF7 数据 buffer 7，32 位。(R/W/SS)

Register 20.27. SPI_W8_REG (0x00B8)

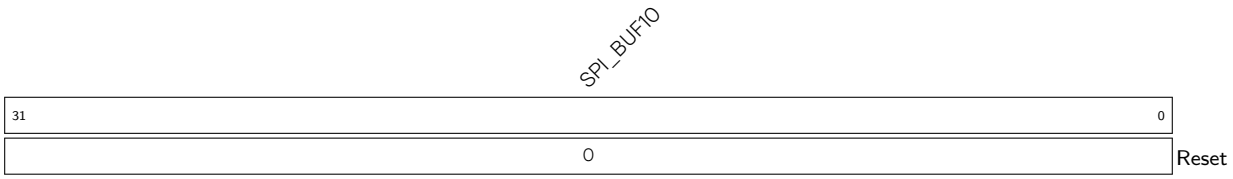


SPI_BUF8 数据 buffer 8，32 位。(R/W/SS)

Register 20.28. SPI_W9_REG (0x00BC)

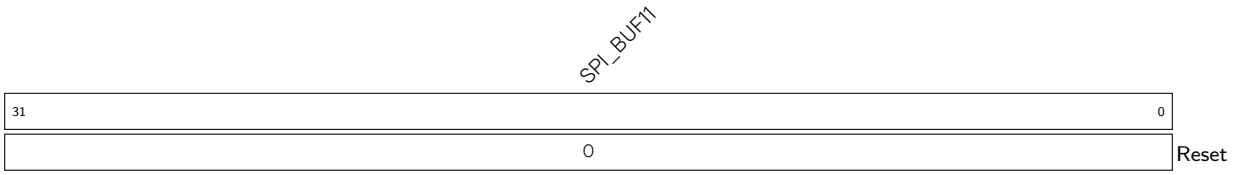
SPI_BUF9 数据 buffer 9，32 位。(R/W/SS)

Register 20.29. SPI_W10_REG (0x00C0)



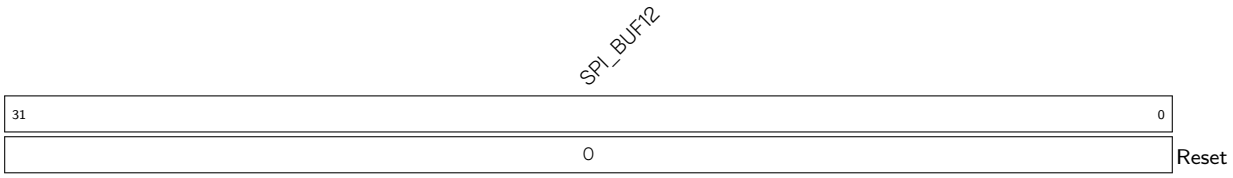
SPI_BUF10 数据 buffer 10，32 位。(R/W/SS)

Register 20.30. SPI_W11_REG (0x00C4)



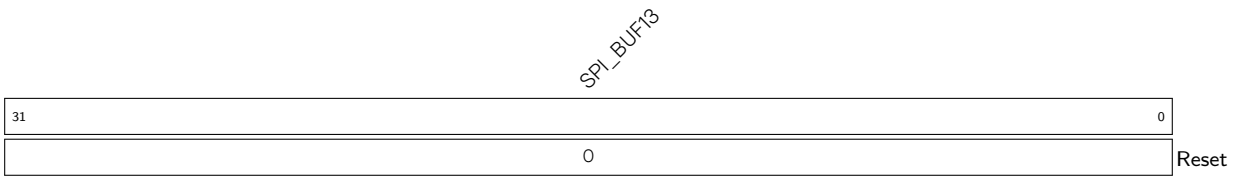
SPI_BUF11 数据 buffer 11，32 位。(R/W/SS)

Register 20.31. SPI_W12_REG (0x00C8)



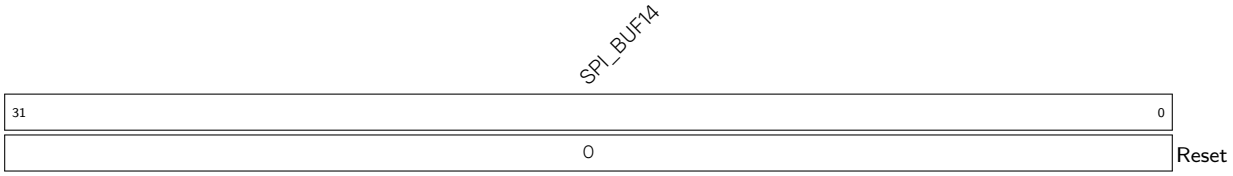
SPI_BUF12 数据 buffer 12，32 位。(R/W/SS)

Register 20.32. SPI_W13_REG (0x00CC)



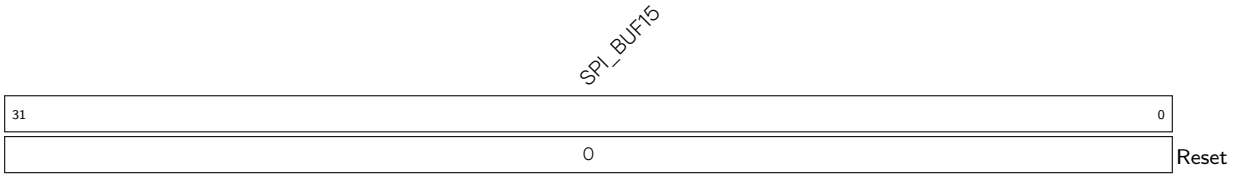
SPI_BUF13 数据 buffer 13，32 位。(R/W/SS)

Register 20.33. SPI_W14_REG (0x00D0)



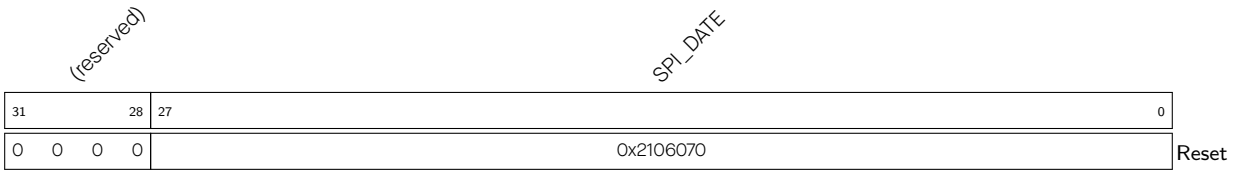
SPI_BUF14 数据 buffer 14，32 位。(R/W/SS)

Register 20.34. SPI_W15_REG (0x00D4)



SPI_BUF15 数据 buffer 15，32 位。(R/W/SS)

Register 20.35. SPI_DATE_REG (0x00F0)



SPI_DATE 版本寄存器。(R/W)

21 I2C 主机控制器 (I2C)

I2C (Inter-Integrated Circuit) 总线用于使 ESP8684 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。

ESP8684 有一个在主机模式工作的 I2C 控制器。

21.1 概述

I2C 是一个两线总线，由 SDA 线和 SCL 线构成。这些线设置为漏极开漏 (open-drain) 输出。因此，I2C 总线上可以挂载多个外设，通常是和一个或多个主机以及一个或多个从机。但同一时刻只有一个主机能占用总线访问一个从机。

主机发出开始信号，则通讯开始：在 SCL 为高电平时拉低 SDA 线，主机将通过 SCL 线发出 9 个时钟脉冲。前 8 个脉冲用于传输 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。根据应答位的逻辑电平决定是否停止发送数据。在数据传输中，SDA 线仅在 SCL 线为低电平时才发生变化。当主机完成通讯，发送一个停止标志：在 SCL 为高电平时，拉高 SDA 线。如果一次通信中主机既有写操作又有读操作，则主机需在读写操作变化前，发送一个重新开始信号、从机地址和读写标志位。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）。

21.2 主要特性

ESP8684 I2C 主机控制器具有以下几个特点：

- 支持主机模式
- 支持多主机通信
- 支持标准模式 (100 Kbit/s)
- 支持快速模式 (400 Kbit/s)
- 支持从机的 7 位以及 10 位地址寻址
- 支持拉低 SCL 时钟实现连续数据传输
- 支持可编程数字噪声滤波功能
- 支持从机地址和从机内存或寄存器地址的双寻址模式

21.3 I2C 架构

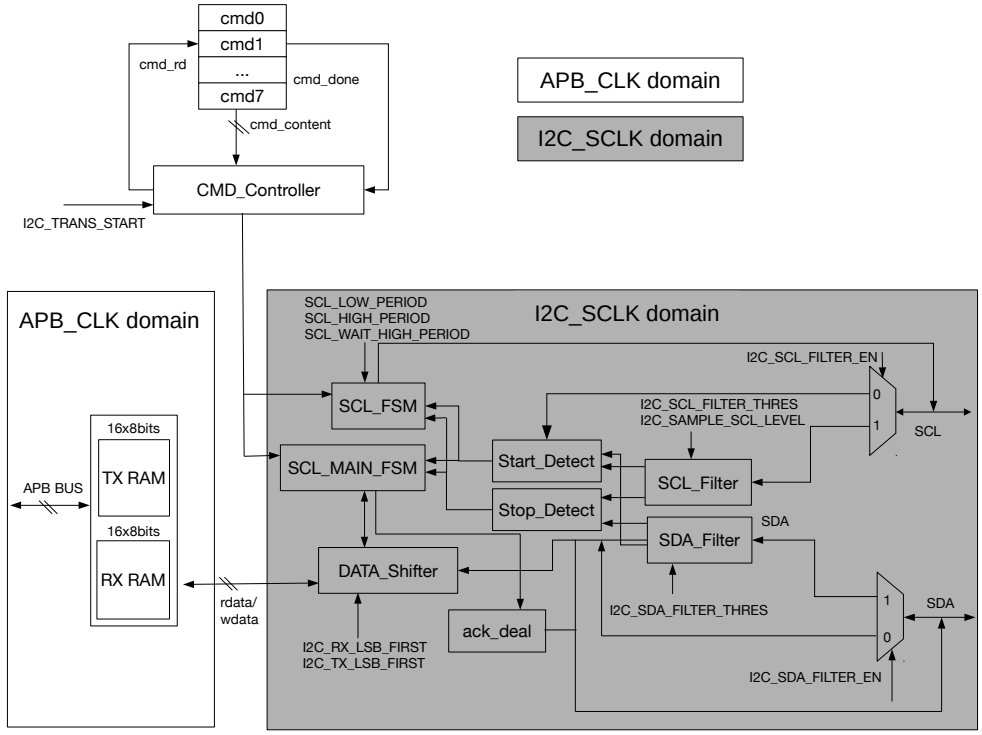


图 21-1. I2C 主机基本架构

图 21-1 为 I2C 主机基本架构图。I2C 主机控制器内部包括的模块主要有：

- 接收和发送存储器 TX/RX RAM
- 命令控制器 CMD_Controller
- SCL 时钟控制器 SCL_FSM
- SDA 数据控制器 SCL_MAIN_FSM
- 串并转换器 DATA_Shifter
- SCL 滤波器 SCL_Filter
- SDA 滤波器 SDA_Filter
- ACK 位控制器 ACK_deal

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。SCL_Filter 和 SDA_Filter 分别用于消除 SCL 及 SDA 输入信号上的噪声。同步模块用来同步不同时钟域之间信号的传输。

图 21-2 和图 21-3 是 I2C 协议的时序图和对应的参数表。SCL_FSM 用来产生满足 I2C 协议的 SCL 时钟。

SCL_MAIN_FSM 模块用来控制 I2C 指令的执行，和 SDA 线的序列。SCL_MAIN_FSM 还控制 ACK_deal 模块生成 ACK 位，检测 SDA 线上 ACK 位的电平。I2C 主机通过 CMD_Controller 产生 (R)START、STOP、WRITE、READ 和 END 指令。TX/RX RAM 分别用来存储 I2C 要发送和接收到的数据。DATA_Shifter 用来完成串行数据和并行数据之间的转换。

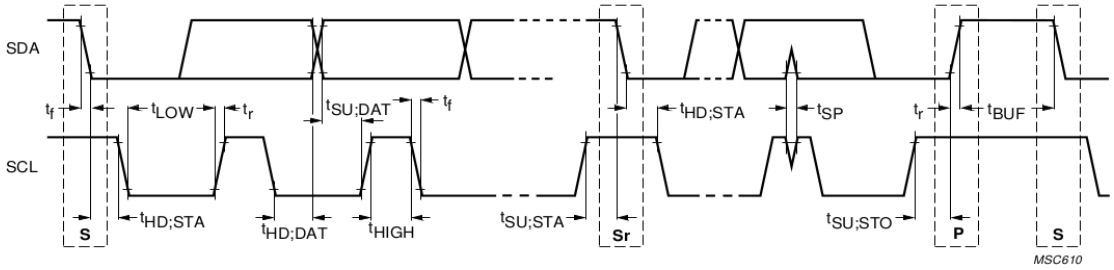


Fig.31 Definition of timing for F/S-mode devices on the I²C-bus.

图 21-2. 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

图 21-3. I2C 时序参数（引自 The I2C-bus specification Version 2.1 Table5）

21.4 功能描述

需要注意的是，I2C 总线上其他主机或者从机的操作可能与 ESP8684 I2C 外设有所不同，具体请参考各个 I2C 设备的技术规格书。

21.4.1 时钟配置

寄存器配置和 TX/RX RAM 部分的时钟域为 APB_CLK。I2C 主要逻辑部分，包括 SCL_FSM、SCL_MAIN_FSM、SCL_FILTER、SDA_FILTER 和 DATA_SHIFTER 都为 I2C_SCLK 时钟域。

用户可以通过配置 I2C_SCLK_SEL 选择 I2C_SCLK 的时钟源：XTAL_CLK 或 RC_FAST_CLK，I2C_SCLK_SEL 为 0 时选择时钟源 XTAL_CLK，I2C_SCLK_SEL 为 1 时选择时钟源 RC_FAST_CLK。配置 I2C_SCLK_ACTIVE 为高电平来打开 I2C_SCLK 的时钟源。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：

$$I2C\_SCLK\_DIV\_NUM + 1 + \frac{I2C\_SCLK\_DIV\_A}{I2C\_SCLK\_DIV\_B}$$

XTAL_CLK 的频率是 40 MHz，RC_FAST_CLK 的频率是 17.5 MHz。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。

21.4.2 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 滤波器会过滤脉冲宽度小于 I2C_SCL_FILTER_THRES 以及 I2C_SDA_FILTER_THRES 个 I2C_SCLK 时钟周期的线路毛刺。

21.4.3 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。ESP8684 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。置位 I2C_SCL_RST_SLV_EN，硬件会发送 I2C_SCL_RST_SLV_NUM 个 SCL 脉冲。一段时间后，软件读取到 I2C_SCL_RST_SLV_EN 位的值为 0 后（该位由硬件自动清零），再置位 I2C_CONF_UPGATE，来停止这个功能。

21.4.4 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 21-1。

表 21-1. 需同步的 I2C 寄存器

配置寄存器	配置参数	地址
I2C_CTR_REG	I2C_SLV_TX_AUTO_START_EN	0x0004
	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_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_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	

21.4.5 漏级开路输出

SCL 及 SDA 线采用漏级开路的驱动方式。I2C 主机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。

21.4.6 时序参数配置

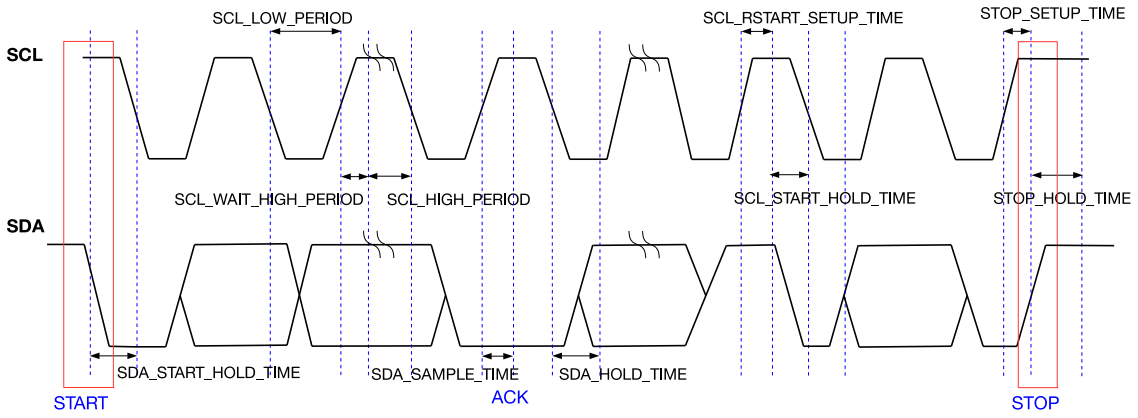


图 21-4. I2C 时序图

图 21-4 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 主机控制器的 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 21-4 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. $t_{SU:DAT} = (I2C\_SCL\_LOW\_PERIOD - I2C\_SDA\_HOLD\_TIME) \cdot T_{I2C\_SCLK}$

各时序寄存器的含义如下：

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。
2. I2C_SCL_LOW_PERIOD：SCL 低电平持续时间。SCL 低电平时间为（I2C_SCL_LOW_PERIOD + 1）个模块时钟周期。但是如果外设拉低 SCL ，I2C 主机控制器执行 END 命令拉低 SCL，或者控制器发生 SCL 时钟拉伸则可能会导致 SCL 低电平时间变长。
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。
4. I2C_SCL_HIGH_PERIOD：SCL 线拉高后维持高电平的模块时钟周期数。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：

$$f_{scl} = \frac{f_{\text{I2C\_SCLK}}}{\text{I2C\_SCL\_LOW\_PERIOD} + \text{I2C\_SCL\_HIGH\_PERIOD} + \text{I2C\_SCL\_WAIT\_HIGH\_PERIOD} + 3}$$

5. I2C_SDA_SAMPLE_TIME：SCL 上升沿到采样 SDA 线电平值的时间间隔。推荐设置在 SCL 高电平持续时间的中间值，以保证能够正确采样到 SDA 线上电平。
6. I2C_SDA_HOLD_TIME：SDA 输出数据变化与 SCL 下降沿的时间间隔。

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

21.4.7 超时控制

I2C 内部有三种超时控制，分别是对 SCL_FSM 状态的超时控制、SCL_MAIN_FSM 状态的超时控制和对 SCL 线状态的超时控制。其中前两种是一直打开的，第三种是可编程配置的。

当 SCL_FSM 长时间处于同一状态不变，且时间超过 $2^{I2C\_SCL\_ST\_TO\_I2C}$ 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，即最大在时间超过 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 I2C_SCLK 时钟周期后会产生超时中断。

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。

21.4.8 指令配置

I2C 主机的 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。

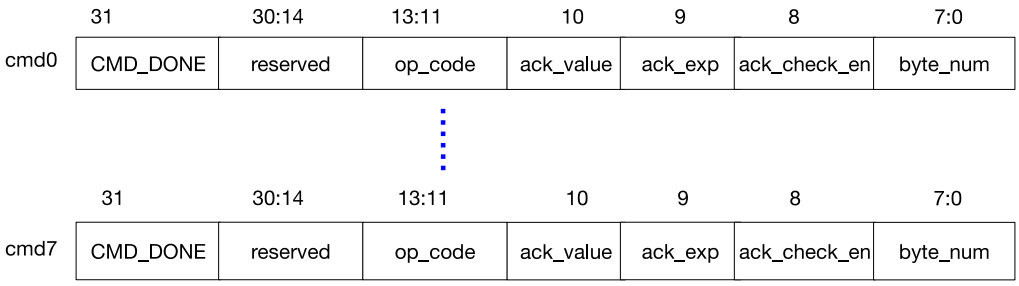


图 21-5. I2C 命令寄存器结构

命令寄存器的内部结构如图 21-5 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：
   - RSTART: op_code 等于 6 时为 RSTART 命令，该命令指示 I2C 主机控制器发送 I2C 协议中的 START 位或 RESTART 位。
   - WRITE: op_code 等于 1 时为 WRITE 命令，该命令指示 I2C 主机控制器向从机发送从机地址、被访问的寄存器地址（仅限双寻址模式）、数据。
   - READ: op_code 等于 3 时为 READ 命令，该命令指示 I2C 主机控制器从从机读取数据。
   - STOP: op_code 等于 2 时为 STOP 命令，该命令指示 I2C 主机控制器发送 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。
   - END: op_code 等于 4 时为 END 命令，该命令指示 I2C 主机控制器将 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。
3. ack_value: 该位设置读操作时 I2C 主机控制器在 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 主机控制器在 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 主机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。

21.4.9 TX/RX RAM 数据存储

TX/RX RAM 大小均为 16 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 主机控制器需要发送的数据。在 I2C 通信的过程中，当 I2C 主机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。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。

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，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 主机控制器接收到的数据。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 (ESP8684 为 16 字节) 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 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 (ESP8684 为 16 字节) 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 时，会产生 I2C_RXFIFO_WM_INT 中断。软件收到中断后，从 I2C_DATA_REG 中读数。

21.4.10 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。

21.4.11 寻址模式

除了 7 位寻址模式，ESP8684 I2C 还支持 10 位寻址模式和双寻址模式。10 位寻址和 7 位寻址可结合使用。

假设从机地址为 SLV_ADDR。ESP8684 I2C 主机控制器可以使用 7 位寻址（SLV_ADDR[6:0]），也可以使用 10 位寻址（SLV_ADDR[9:0]）。

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。当主机发送的地址为广播地址（0x00）且读写标志位为 0 时，此时支持广播地址的从机无论自己的地址是多少，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]。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。

部分 I2C 从机支持双地址访问方式。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。ESP8684 I2C 同样支持以双地址模式访问从机。

21.4.12 启动控制器

I2C 主机控制器配置成主机模式 (I2C_MS_MODE) 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。

21.5 编程示例

本节提供一些典型通信场景的编程示例。ESP8684 中有一个 I2C 主机控制器，为了便于描述，下文所有图示中的 I2C 主机假定为 ESP8684 I2C 外设控制器，从机为支持 The I2C-bus specification Version 2.1 的 I2C 控制器，并包含相应功能。

21.5.1 I2C 主机写入从机，7 位寻址，单次命令序列

21.5.1.1 场景介绍

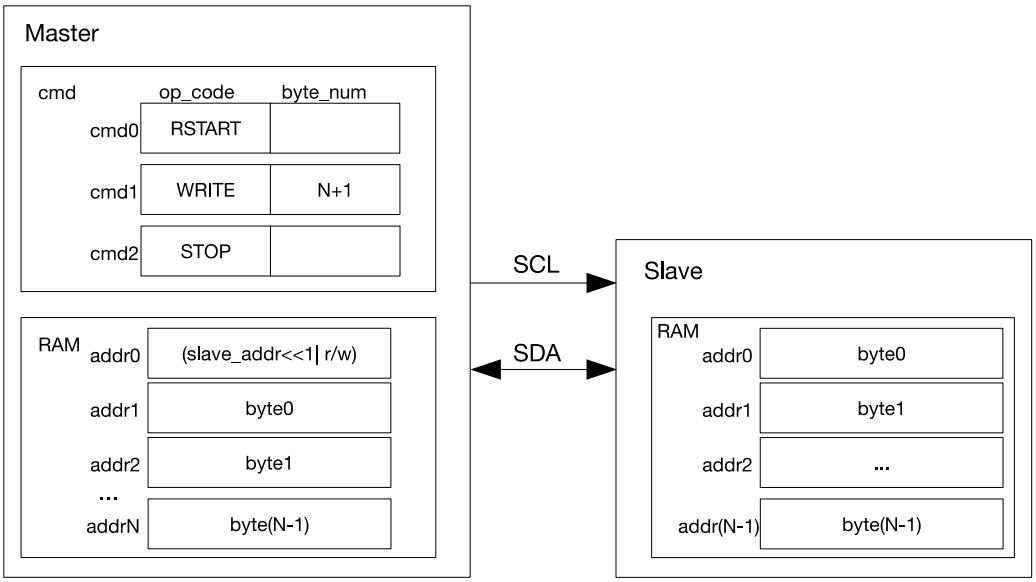


图 21-6. I2C 主机写 7 位寻址的从机

图 21-6 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 21-6 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. 执行 RSTART 命令发送 START 位。
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。

21.5.1.2 配置示例

1. 参照章节 21.4.6，配置 I2C 主机的时序参数寄存器。参照所使用的 I2C 从机产品手册，调节 I2C 从机的时序。
2. 设置 I2C_MS_MODE 为 1。

3. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2	STOP	—	—	—	—

5. 参考章节 21.4.9，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. 设置 I2C 从机的地址 I2C_SLAVE_ADDR[7:0]。
7. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的从机地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 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 配置的不同进行或不进行 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 21.4.9。
11. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT 中断。

21.5.2 I2C 主机写入从机，10 位寻址，单次命令序列

21.5.2.1 场景介绍

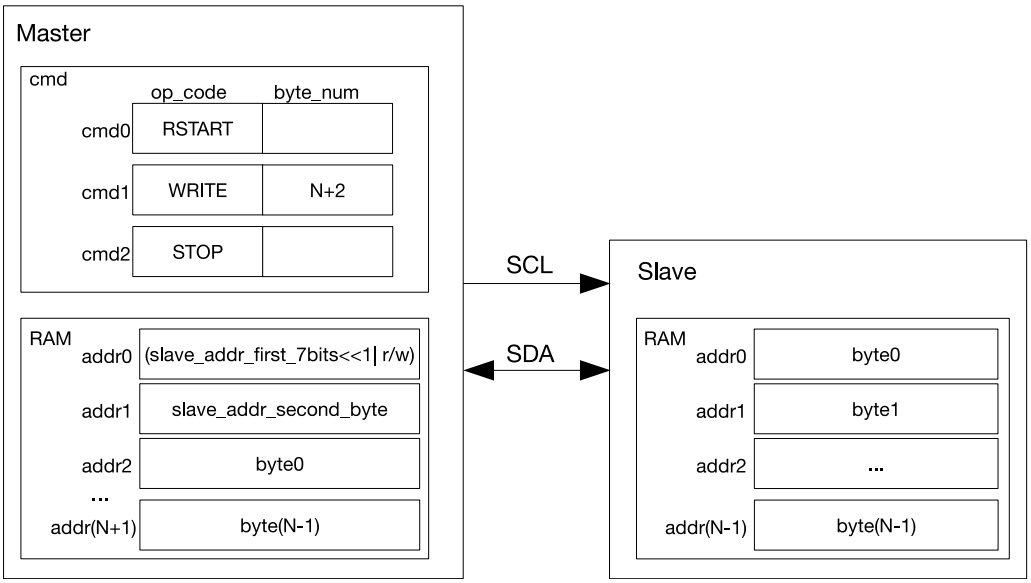


图 21-7. I2C 主机写 10 位寻址的从机

图 21-7 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 21.5.1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。

21.5.2.2 配置示例

1. 设置 I2C_MS_MODE 为 1。
2. 向 I2C_CONF_UPGATE 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_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2	STOP	—	—	—	—

4. 设置 I2C 从机的 10 位从机地址 I2C_SLAVE_ADDR[9:0]。
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 写 1 来同步寄存器。
7. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 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 配置的不同进行或不进行 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 21.4.9。
11. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT 中断。

21.5.3 I2C 主机写入从机，7 位双地址寻址，单次命令序列

21.5.3.1 场景介绍

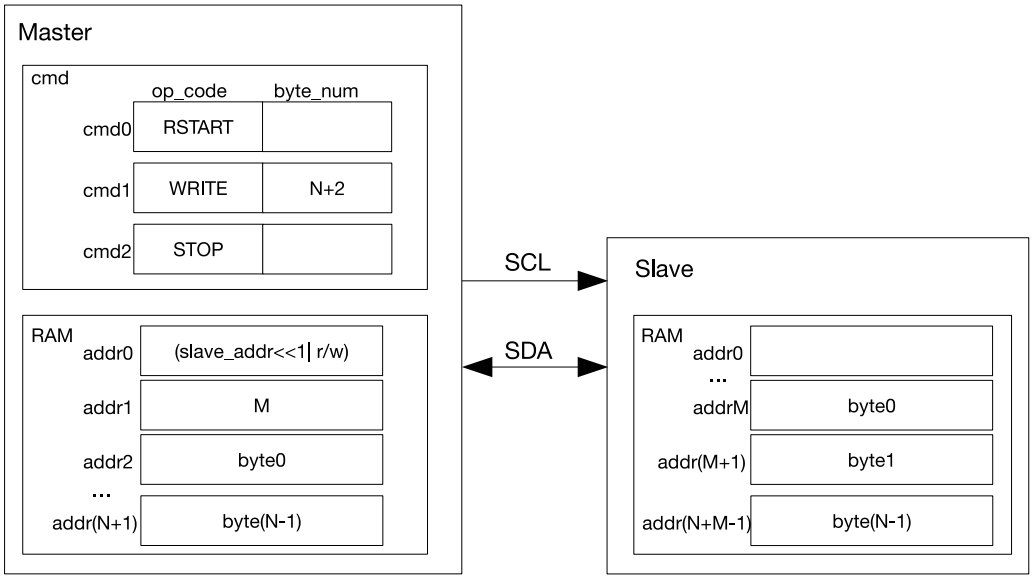


图 21-8. I2C 主机写 7 位双地址寻址从机

图21-8 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 21.5.1 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 21-8 中右侧的 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 的寄存器或 RAM 中的 addrM 开始依次存储，addrM 就是主机发送的 I2C 内存地址。

21.5.3.2 配置示例

1. 选择支持双寻址模式的 I2C 从机并打开双寻址模式。
2. 设置 I2C_MS_MODE 为 1。
3. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+2
I2C_COMMAND2	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. 设置 I2C 从机的地址 I2C_SLAVE_ADDR[7:0]。
7. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
8. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
- 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 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 配置的不同进行或不进行 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 21.4.9。
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 中断。

21.5.4 I2C 主机写入从机，7 位寻址，多次命令序列

21.5.4.1 场景介绍

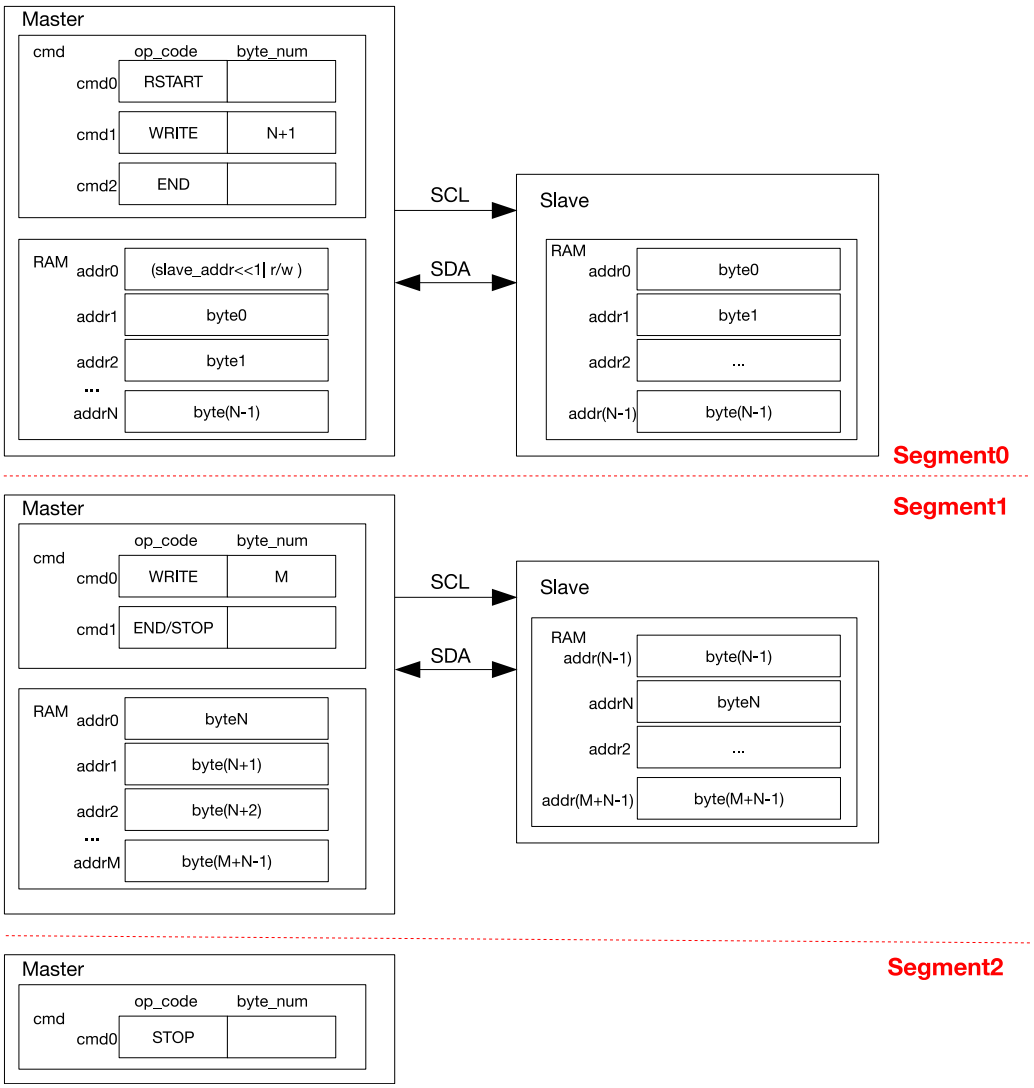


图 21-9. I2C 主机分段写 7 位寻址的从机

RAM 的大小只有 16 字节，对于大量的数据传输当 RAM 乒乓操作也不能满足要求时，建议使用多次命令序列进行分段传输。每次命令序列以 END 命令结尾，这样控制器会执行 END 命令拉低 SCL 线，软件此时可以更新命令序列寄存器和 RAM 的内容以用于下一次命令序列的传输。

以两段和三段传输为例，如图 21-9 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 信号后总线才会被释放。任何情况下，置位 I2C_FSM_RST 可复位 I2C 主机控制器，硬件自清 I2C_FSM_RST。

21.5.4.2 配置示例

1. 设置 I2C_MS_MODE 为 1。
2. 向 I2C_CONF_UPGATE 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_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	ack_value	ack_exp	1	N+1
I2C_COMMAND2	END	—	—	—	—

4. 参考章节 21.4.9，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. 设置 I2C 从机的地址 I2C_SLAVE_ADDR[7:0]。
6. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
7. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 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 配置的不同进行或不进行 ACK 检测。
10. 等到 I2C_END_DETECT_INT 中断产生后，设置I2C_END_DETECT_INT_CLR 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_er	byte_num

I2C_COMMAND0	WRITE	ack_value	ack_exp	1	M
I2C_COMMAND1	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 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 中断。
15. 若I2C_COMMAND1 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_er	byte_num
I2C_COMMAND1	STOP	—	—	—	—

17. 向 I2C_TRANS_START 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 中断。

21.5.5 I2C 主机读取从机，7 位寻址，单次命令序列

21.5.5.1 场景介绍

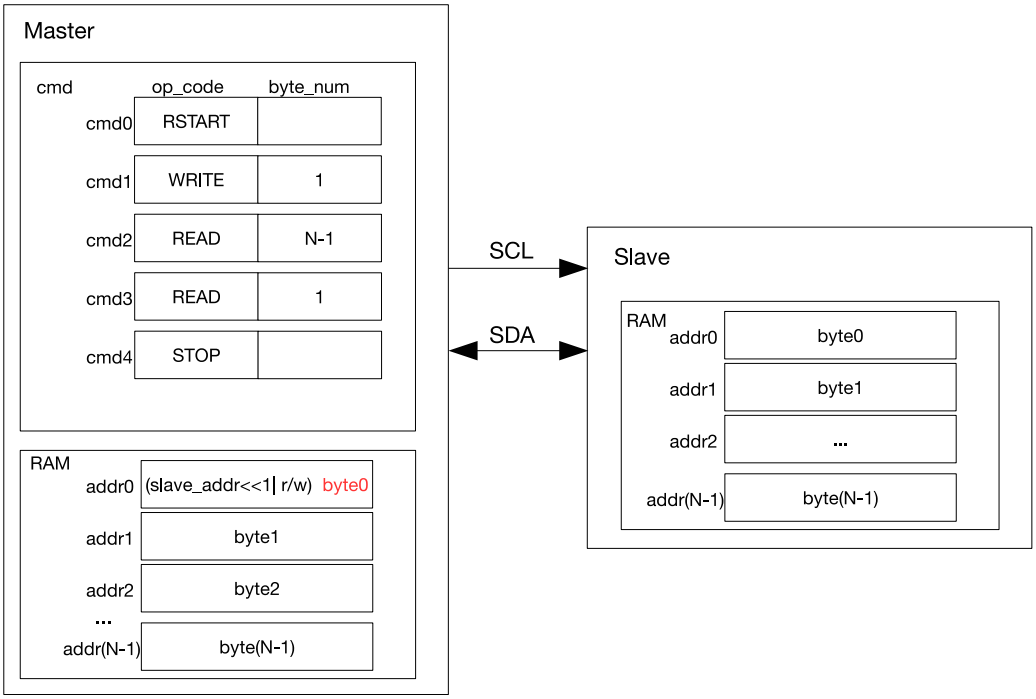


图 21-10. I2C 主机读 7 位寻址的从机

图 21-10 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 21-10 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。

21.5.5.2 配置示例

1. 设置 I2C_MS_MODE 为 1。
2. 向 I2C_CONF_UPGATE 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_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	1
I2C_COMMAND2	READ	0	0	1	N-1
I2C_COMMAND3	READ	1	0	1	1
I2C_COMMAND4	STOP	—	—	—	—

4. 参考章节 21.4.9，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。
5. 设置 I2C 从机的地址 I2C_SLAVE_ADDR[7:0]。
6. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
7. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 中断，停止发送数据并且产生 STOP。
9. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_value 配置的不同回复不同的 ACK 值。
10. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 RX RAM 进行乒乓操作，具体做法参照章节 21.4.9。
11. 当 I2C 主机接收最后一个数据时，将 ack_value 设成 1，I2C 从机接收到 NACK 中断，停止发送。
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 中断。

21.5.6 I2C 主机读取从机，10 位寻址，单次命令序列

21.5.6.1 场景介绍

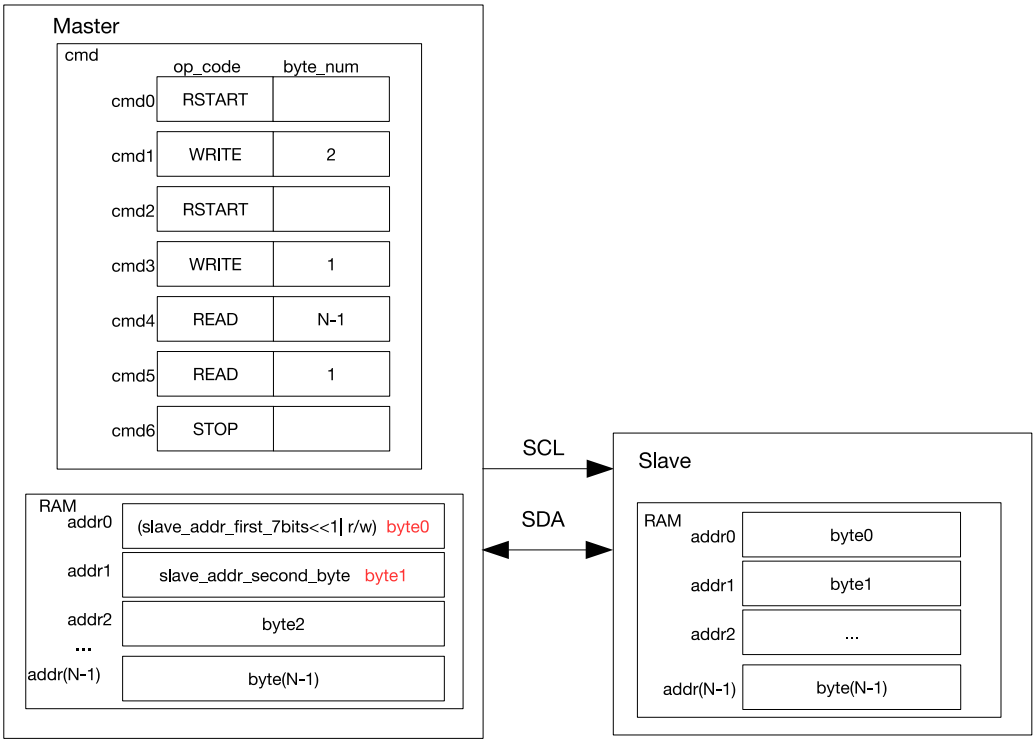


图 21-11. I2C 主机读 10 位寻址的从机

图 21-11 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。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 21.5.2 的相同。

21.5.6.2 配置示例

1. 设置 I2C_MS_MODE 为 1。
2. 向 I2C_CONF_UPGATE 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_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	2
I2C_COMMAND2	RSTART	—	—	—	—
I2C_COMMAND3	WRITE	0	0	1	1
I2C_COMMAND4	READ	0	0	1	N-1
I2C_COMMAND5	READ	1	0	1	1
I2C_COMMAND6	STOP	—	—	—	—

4. 设置 I2C 从机的 10 位从机地址 I2C_SLAVE_ADDR[9:0] 。
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，R/W 位为 W ；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。

6. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
7. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 中断，停止发送数据并且产生 STOP。
9. 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。
10. I2C 从机重复执行步骤 8，若地址匹配，继续后面的步骤。
11. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_value 配置的不同回复不同的 ACK 值。
12. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 RX RAM 进行乒乓操作，具体做法参照章节 21.4.9。
13. 当 I2C 主机接收最后一个数据时，将 ack_value 设成 1，I2C 从机接收到 NACK 中断，停止发送。
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 中断。

21.5.7 I2C 主机读取从机，7 位双寻址，单次命令序列

21.5.7.1 场景介绍

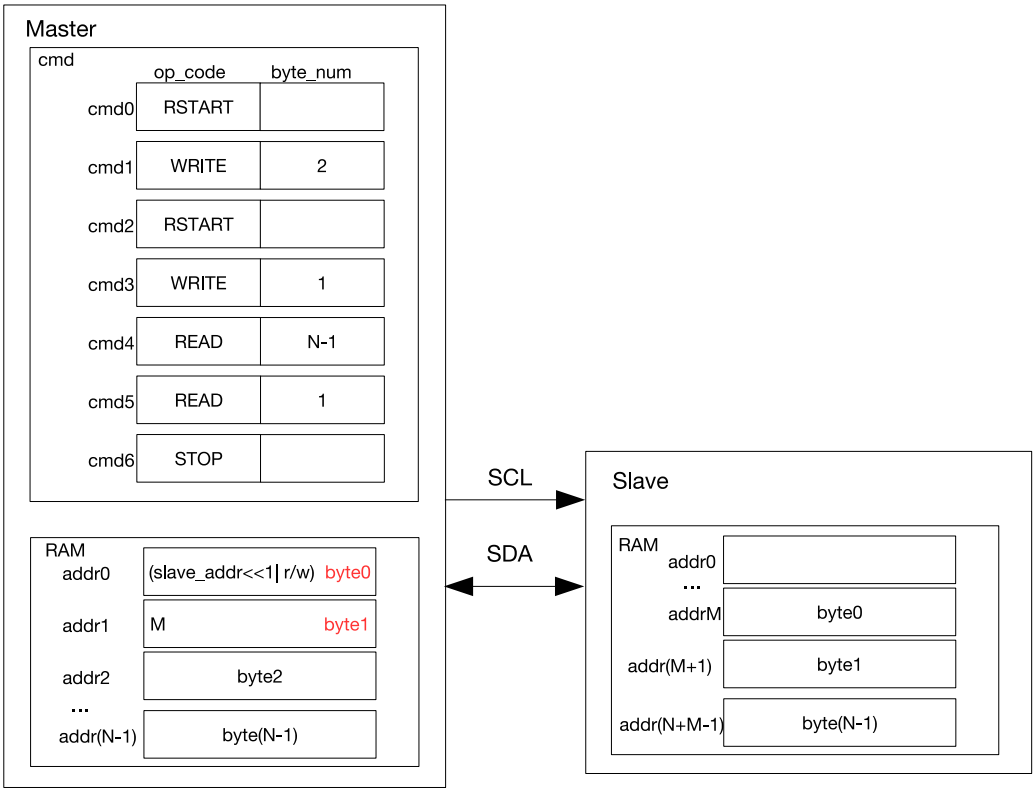


图 21-12. I2C 主机从 7 位寻址从机的 M 地址读取 N 个数据

图 21-12 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 ； 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 。之后主机从从机的 AddrM 地址开始读取数据。

21.5.7.2 配置示例

1. 设置 I2C_MS_MODE 为 1。
2. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
3. 选择支持双寻址模式的 I2C 从机并打开双寻址模式。
4. 配置 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	2
I2C_COMMAND2	RSTART	—	—	—	—
I2C_COMMAND3	WRITE	0	0	1	1
I2C_COMMAND4	READ	0	0	1	N-1
I2C_COMMAND5	READ	1	0	1	1
I2C_COMMAND6	STOP	—	—	—	—

5. 设置 I2C 从机的地址 I2C_SLAVE_ADDR[7:0]。
6. 参考章节 21.4.9，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；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。
7. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
8. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
9. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 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 TX RAM 的地址偏移。
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 9，若地址匹配，继续后面的步骤。
13. I2C 从机发送数据，I2C 主机会根据当前 READ 指令对应的 ack_value 配置的不同回复不同的 ACK 值。
14. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 RX RAM 进行乒乓操作，具体做法参照章节 21.4.9。
15. 当 I2C 主机接收最后一个数据时，将 ack_value 设成 1，I2C 从机接收到 NACK 中断，停止发送。
16. 当整个传输正常结束，I2C 主机执行 STOP 命令，并产生 I2C_TRANS_COMPLETE_INT 中断。

21.5.8 I2C 主机读取从机，7 位寻址，多次命令序列

21.5.8.1 场景介绍

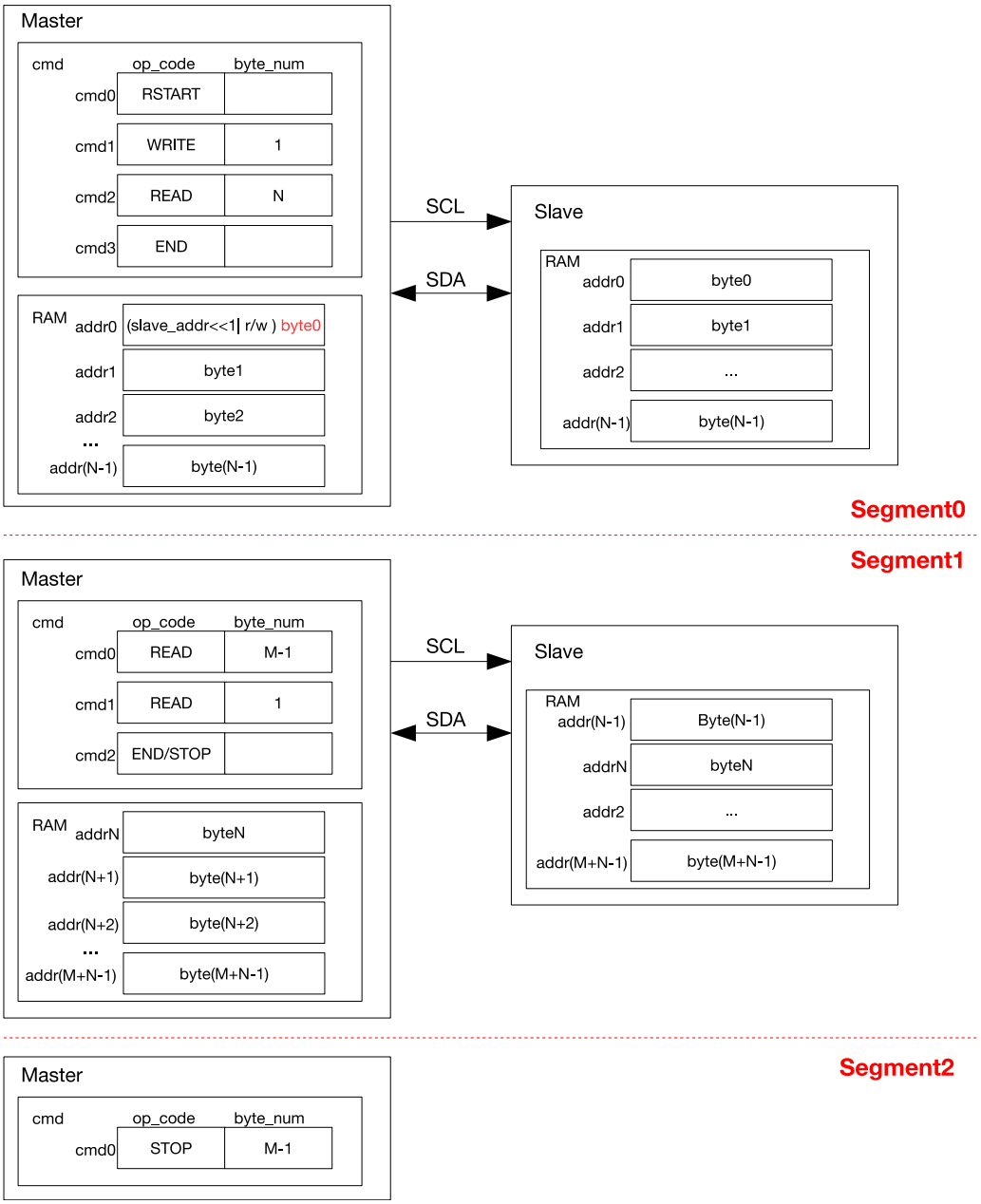


图 21-13. I2C 主机分段读 7 位寻址的从机

图 21-13 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. 第一段流程和图 21-10类似，只是最后一个命令变为 END。
2. 准备好 I2C 从机的发送数据，置位 I2C_TRANS_START (主机)，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(主机)，I2C 主机发送 STOP 位停止传输。

21.5.8.2 配置示例

1. 设置 I2C_MS_MODE 为 1。
2. 向 I2C_CONF_UPGATE 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_er	byte_num
I2C_COMMAND0	RSTART	—	—	—	—
I2C_COMMAND1	WRITE	0	0	1	1
I2C_COMMAND2	READ	0	0	1	N
I2C_COMMAND3	END	—	—	—	—

4. 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。
5. 设置 I2C 从机的地址 I2C_SLAVE_ADDR[7:0]。
6. 向 I2C_CONF_UPGATE 写 1 来同步寄存器。
7. 向 I2C_TRANS_START 位写 1 开始传输。并打开 I2C 从机开始传输。
8. I2C 从机比较 I2C 主机发送的从机地址和自己的地址，当 I2C 主机 WRITE 命令中的 ack_check_en 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 电平一致，传输继续。
   - 不匹配：接收的 ACK 值与 WRITE 命令中的 ack_exp 电平不一致，I2C 主机产生 I2C_NACK_INT 中断，停止发送数据并且产生 STOP。
9. I2C 从机发送数据，，I2C 主机会根据当前 READ 指令对应的 ack_value 配置的不同回复不同的 ACK 值。
10. 若接收数据 N 大于 RX FIFO 深度，在 FIFO 模式下可以对 I2C 主机的 RX RAM 进行乒乓操作，具体做法参照章节 21.4.9。
11. 等到一次 READ 指令完成，I2C 主机执行 END 指令，I2C_END_DETECT_INT 中断产生后，设置 I2C_END_DETECT_INT_CLR 为 1 来清除中断。

12. 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_er	byte_num
I2C_COMMAND0	READ	ack_value	ack_exp	1	M
I2C_COMMAND1	END	—	—	—	—

或者

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND0	READ	0	0	1	M-1
I2C_COMMAND1	READ	1	0	1	1
I2C_COMMAND2	STOP	—	—	—	—

13. 准备 I2C 从机的发送数据。
14. 向 I2C_TRANS_START 位写 1 开始传输，并重复步骤 9 的流程。
15. 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 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 中断。
16. 若最后一个指令为为 END，则重复步骤 11，并在完成后继续下面的步骤。
17. 更新 I2C 主机的指令寄存器。

指令寄存器	op_code	ack_value	ack_exp	ack_check_er	byte_num
I2C_COMMAND1	STOP	—	—	—	—

18. 向 I2C_TRANS_START 位写 1 开始传输。
19. I2C 主机执行 STOP 命令结束传输，并产生 I2C_TRANS_COMPLETE_INT 中断。

21.6 中断

- I2C_DET_START_INT：主机或从机检测到 I2C START 位时，触发此中断。
- I2C_SCL_MAIN_ST_TO_INT: 当 I2C 主状态机 SCL_MAIN_FSM 保持某个状态超过 I2C_SCL_MAIN_ST_TO_I2C[23:0] 个模块时钟周期时，触发此中断。
- I2C_SCL_ST_TO_INT: 当 I2C 状态机 SCL_FSM 保持某个状态超过 I2C_SCL_ST_TO_I2C[23:0] 个模块时钟周期时，触发此中断。
- I2C_RXFIFO_UDF_INT: 当 I2C 通过 APB 总线读取 RX FIFO，但 RX FIFO 为空时，触发该中断。
- I2C_TXFIFO_OVF_INT: 当 I2C 通过 APB 总线写 TX FIFO，但 TX FIFO 为满时，触发该中断。
- I2C_NACK_INT: 当 I2C 配置为主机时，接收到的 ACK 与命令中期望的 ACK 值不一致时，即触发该中断；当 I2C 配置为从机时，接收到的 ACK 值为 1 时即触发该中断。
- I2C_TRANS_START_INT: 当 I2C 发送一个 START 位时，即触发该中断。
- I2C_TIME_OUT_INT: 在传输过程中，当 I2C SCL 保持为高或为低电平的时间超过 $2^{I2C\_TIME\_OUT\_VALUE}$ 个模块时钟后，即触发该中断。
- I2C_TRANS_COMPLETE_INT: 当 I2C 检测到 STOP 位时，即触发该中断。

- I2C_MST_TXFIFO_UDF_INT: 当 I2C 主机的 TX FIFO 下溢时，触发此中断。
- I2C_ARBITRATION_LOST_INT: 当 I2C 主机的 SCL 为高电平，SDA 输出值与输入值不相等时，即触发该中断。
- I2C_BYTE_TRANS_DONE_INT: 当 I2C 发送或接收一个字节，即触发该中断。
- I2C_END_DETECT_INT：当 I2C 主机命令的 op_code 为 END，且检测到 I2C END 状态时，触发此中断。
- I2C_RXFIFO_OVF_INT：当 I2C RX FIFO 上溢时，触发此中断。
- I2C_TXFIFO_WM_INT：I2C TX FIFO 水标中断。当I2C_FIFO_PRT_EN 为 1，且 TX FIFO 指针小于 I2C_TXFIFO_WM_THRHD[4:0] 时，触发此中断。
- I2C_RXFIFO_WM_INT：I2C RX FIFO 水标中断。当I2C_FIFO_PRT_EN 为 1，且 RX FIFO 指针大于 I2C_RXFIFO_WM_THRHD[4:0] 时，触发此中断。

21.7 寄存器列表

本小节的所有地址均为相对于 I2C 主机控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

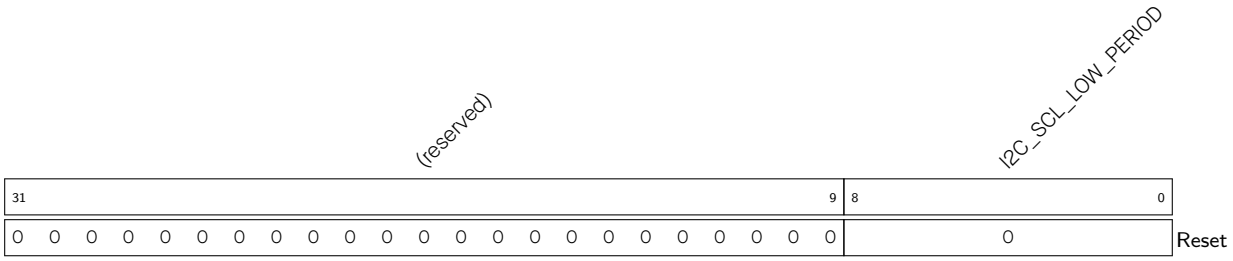
名称	描述	地址	访问
时序寄存器			
I2C_SCL_LOW_PERIOD_REG	配置 SCL 的低电平宽度	0x0000	R/W
I2C_SDA_HOLD_REG	配置 SCL 下降沿后的保持时间	0x0030	R/W
I2C_SDA_SAMPLE_REG	配置 SCL 上升沿后的采样时间	0x0034	R/W
I2C_SCL_HIGH_PERIOD_REG	配置 SCL 时钟的高电平宽度	0x0038	R/W
I2C_SCL_START_HOLD_REG	配置 START 命令产生时 SDA 下降沿和 SCL 下降沿之间的间隔时间	0x0040	R/W
I2C_SCL_RSTART_SETUP_REG	配置 SCL 上升沿和 SDA 下降沿之间的延迟	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	超时控制寄存器	0x000C	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_CLK_CONF_REG	I2C 时钟配置寄存器	0x0054	R/W
I2C_SCL_SP_CONF_REG	电源配置寄存器	0x0080	varies
状态寄存器			
I2C_SR_REG	描述 I2C 的工作状态	0x0008	RO
I2C_FIFO_ST_REG	FIFO 状态寄存器	0x0014	RO
I2C_DATA_REG	读/写 FIFO 寄存器	0x001C	R/W
中断寄存器			
I2C_INT_RAW_REG	原始中断状态	0x0020	R/ SS/ WTC
I2C_INT_CLR_REG	中断清除位	0x0024	WT
I2C_INT_ENA_REG	中断使能位	0x0028	R/W
I2C_INT_STATUS_REG	捕捉 I2C 通信事件的状态	0x002C	RO
命令寄存器			
I2C_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
I2C_COMD5_REG	I2C 命令寄存器 5	0x006C	varies
I2C_COMD6_REG	I2C 命令寄存器 6	0x0070	varies

名称	描述	地址	访问
I2C_COMD7_REG	I2C 命令寄存器 7	0x0074	varies
地址寄存器			
I2C_TXFIFO_START_ADDR_REG	I2C TX FIFO 基地址寄存器	0x0100	HRO
I2C_RXFIFO_START_ADDR_REG	I2C RX FIFO 基地址寄存器	0x0180	HRO
版本寄存器			
I2C_DATE_REG	版本控制寄存器	0x00F8	R/W

21.8 寄存器

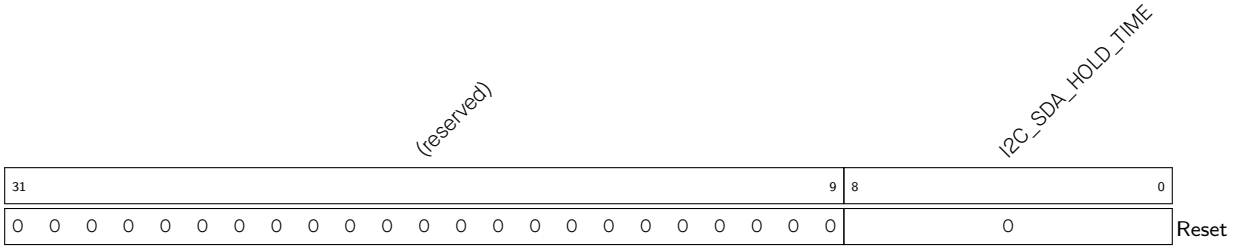
本小节的所有地址均为相对于 I2C 主机控制器 基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 21.1. I2C_SCL_LOW_PERIOD_REG (0x0000)



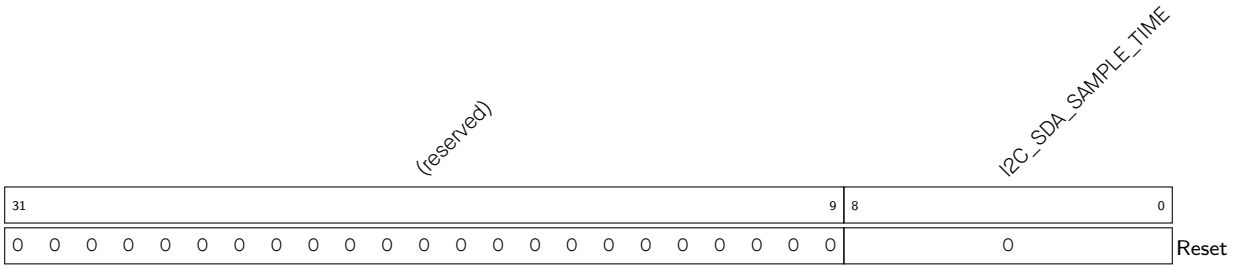
I2C_SCL_LOW_PERIOD 用于配置 SCL 低电平的保持时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

Register 21.2. I2C_SDA_HOLD_REG (0x0030)



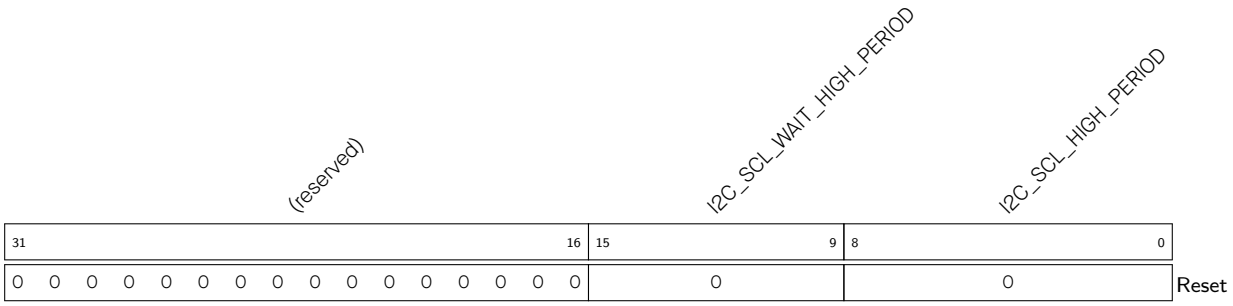
I2C_SDA_HOLD_TIME 用于配置 SCL 下降沿后的数据保持时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

Register 21.3. I2C_SDA_SAMPLE_REG (0x0034)



I2C_SDA_SAMPLE_TIME 用于配置采样 SDA 的时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

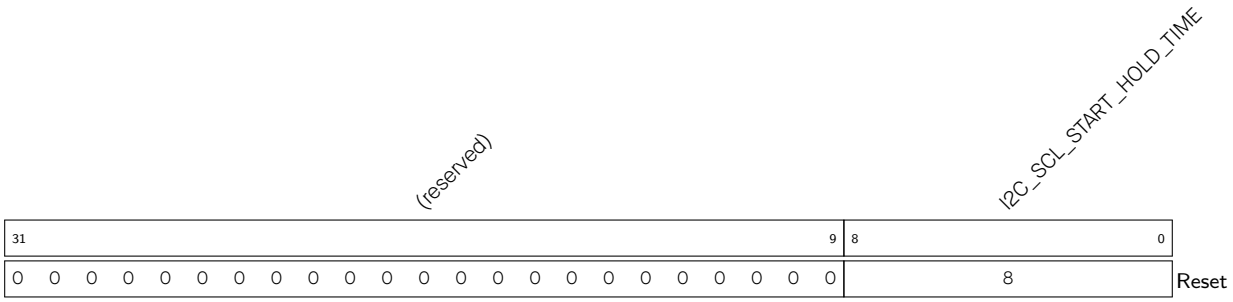
Register 21.4. I2C_SCL_HIGH_PERIOD_REG (0x0038)



I2C_SCL_HIGH_PERIOD 用于配置 SCL 保持高电平的时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

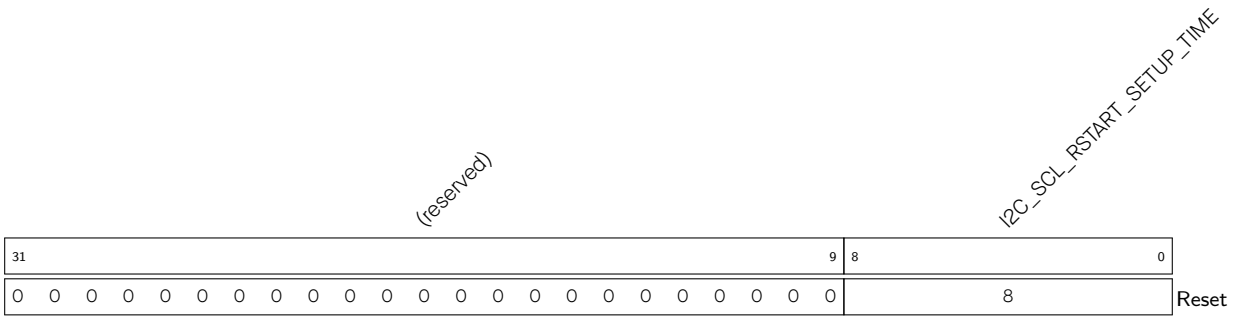
I2C_SCL_WAIT_HIGH_PERIOD 用于配置 SCL_FSM 等待 SCL 翻转至高电平的时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

Register 21.5. I2C_SCL_START_HOLD_REG (0x0040)



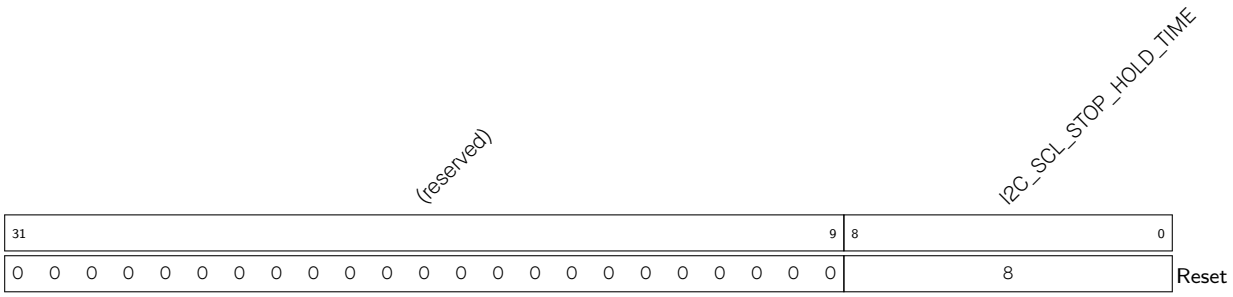
I2C_SCL_START_HOLD_TIME 配置 START 命令产生时 SDA 下降沿和 SCL 下降沿的间隔时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

Register 21.6. I2C_SCL_RSTART_SETUP_REG (0x0044)



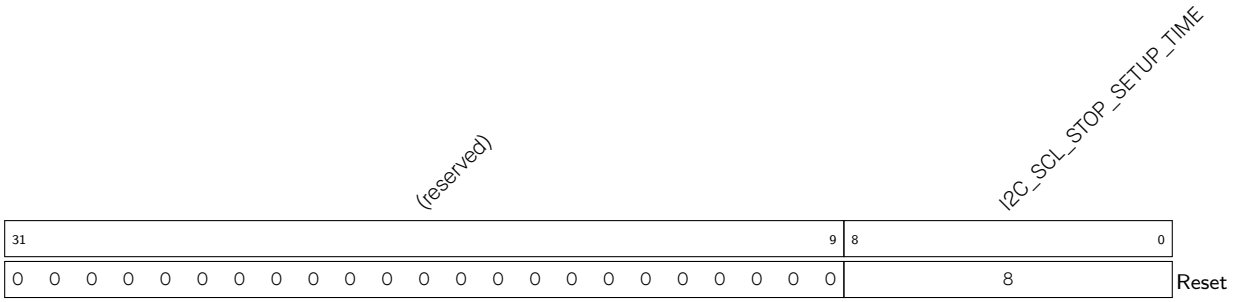
I2C_SCL_RSTART_SETUP_TIME 配置 RSTART 命令产生时 SCL 上升沿和 SDA 下降沿的间隔时间，以 I2C 主机控制器的时钟周期数为单位。(R/W)

Register 21.7. I2C_SCL_STOP_HOLD_REG (0x0048)



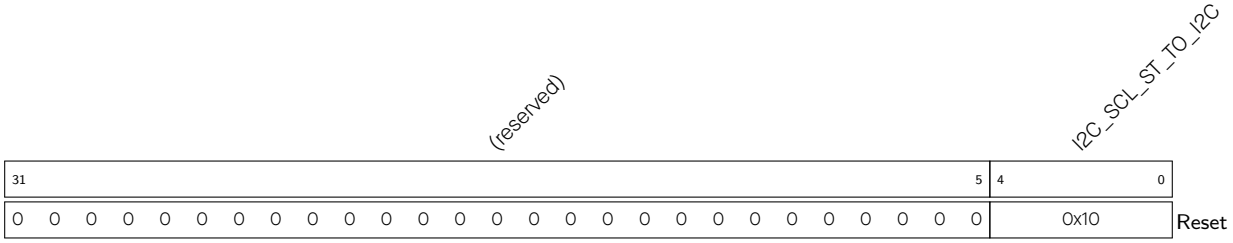
I2C_SCL_STOP_HOLD_TIME 配置 STOP 命令后的延迟，以 I2C 主机控制器时钟周期数为单位。(R/W)

Register 21.8. I2C_SCL_STOP_SETUP_REG (0x004C)



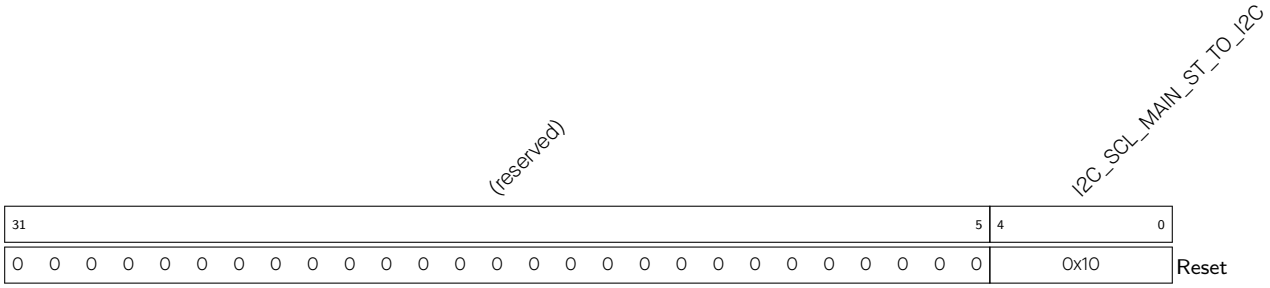
I2C_SCL_STOP_SETUP_TIME 配置 SCL 上升沿和 SDA 上升沿的间隔时间，以 I2C 主机控制器时钟周期数为单位。(R/W)

Register 21.9. I2C_SCL_ST_TIME_OUT_REG (0x0078)



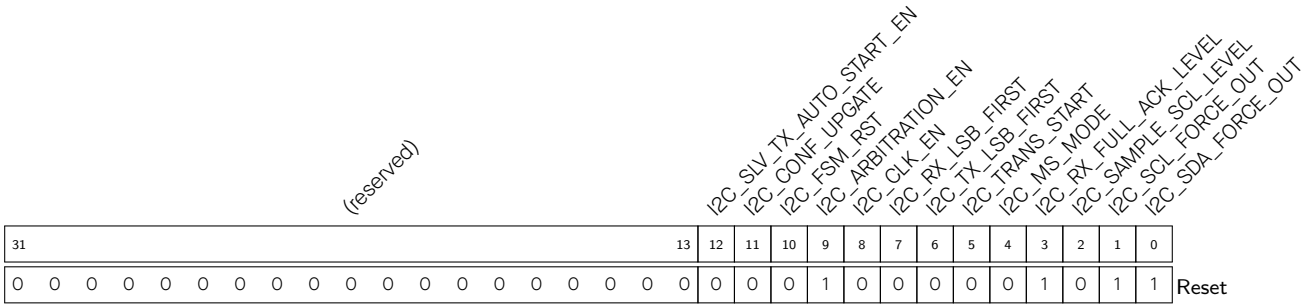
I2C_SCL_ST_TO_I2C SCL_FSM 状态不变的最大时间，不能大于 23。(R/W)

Register 21.10. I2C_SCL_MAIN_ST_TIME_OUT_REG (0x007C)



I2C_SCL_MAIN_ST_TO_I2C SCL_MAIN_FSM 状态不变的最大时间，不能大于 23。(R/W)

Register 21.11. 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 用于选择采样模式。0：SCL 为高电平时采样 SDA 数据；1：SCL 为低电平时采样 SDA 数据。(R/W)

I2C_RX_FULL_ACK_LEVEL 用于配置主机在 I2C_RXFIFO_CNT 达到阈值时需发送的 ACK 电平值。(R/W)

I2C_MS_MODE 置位此位，将 I2C 主机控制器配置为主机。清零此位，则 I2C 主机控制器不可运行。(R/W)

I2C_TRANS_START 置位此位，开始发送 TX FIFO 中的数据。(WT)

I2C_TX_LSB_FIRST 用于控制待发送数据的发送顺序。0：从最高有效位开始发送数据；1：从最低有效位开始发送数据。(R/W)

I2C_RX_LSB_FIRST 用于控制接收数据的存储顺序。0：从最高有效位开始接收数据；1：从最低有效位开始接收数据。(R/W)

I2C_CLK_EN 保留。(R/W)

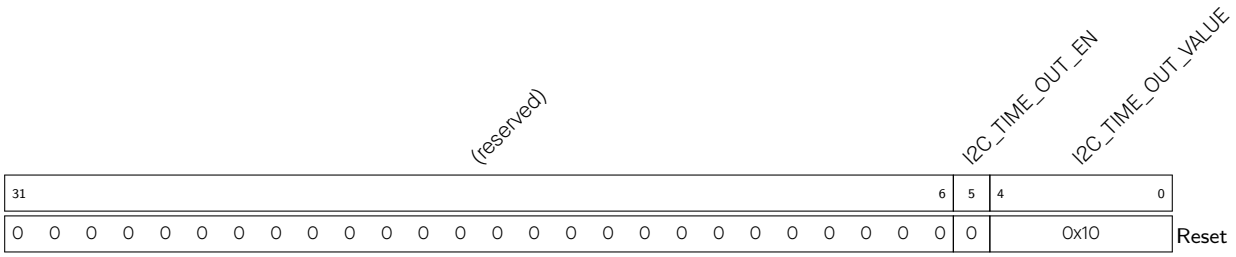
I2C_ARBITRATION_EN I2C 总线仲裁的使能位。(R/W)

I2C_FSM_RST 用于复位 SCL_FSM。(WT)

I2C_CONF_UPGATE 同步位。(WT)

I2C_SLV_TX_AUTO_START_EN 从机自动发送数据的使能位。(R/W)

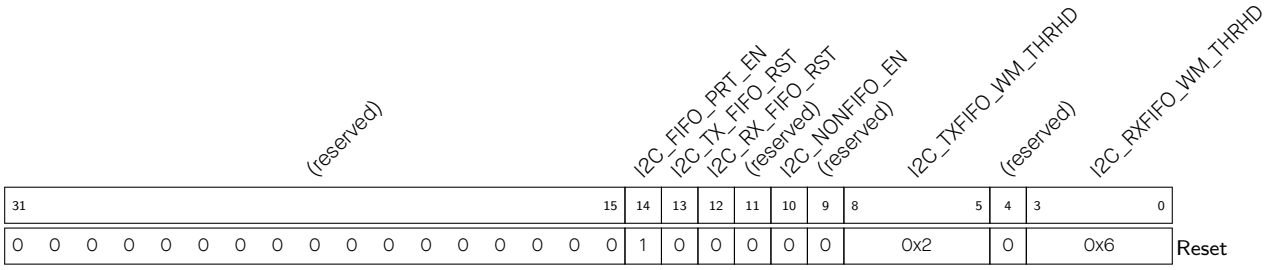
Register 21.12. I2C_TO_REG (0x000C)



I2C_TIME_OUT_VALUE 用于配置接收一位数据的超时时间，以 I2C_SCLK 时钟周期为单位。配置的超时时间为 $2^{I2C\_TIME\_OUT\_VALUE}$ 个时钟周期。(R/W)

I2C_TIME_OUT_EN 超时控制使能位。(R/W)

Register 21.13. I2C_FIFO_CONF_REG (0x0018)



I2C_RXFIFO_WM_THRHD 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_TXFIFO_WM_THRHD[3:0] 时，I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

I2C_TXFIFO_WM_THRHD 直接访问模式下，TX FIFO 的水标阈值。I2C_FIFO_PRT_EN 为 1 且 TX FIFO 计数值小于 I2C_TXFIFO_WM_THRHD[3:0] 时，I2C_TXFIFO_WM_INT_RAW 位有效。(R/W)

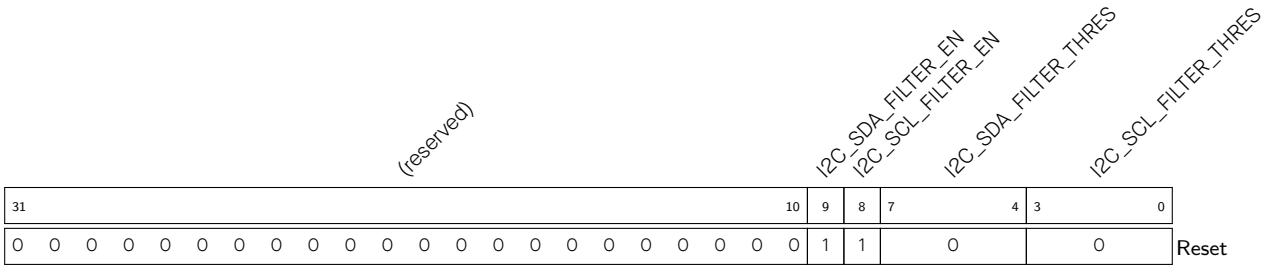
I2C_NONFIFO_EN 置位此位，使能 APB 直接访问。(R/W)

I2C_RX_FIFO_RST 置位此位，复位 RX FIFO 。(R/W)

I2C_TX_FIFO_RST 置位此位，复位 TX FIFO 。(R/W)

I2C_FIFO_PRT_EN 直接访问模式下 FIFO 指针的控制使能位。该位控制 TX FIFO 和 RX FIFO 溢出、下溢、为满、为空时的有效位和中断。(R/W)

Register 21.14. I2C_FILTER_CFG_REG (0x0050)



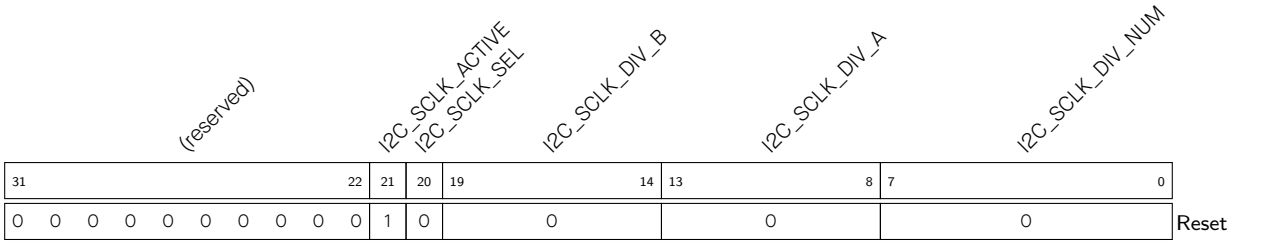
I2C_SCL_FILTER_THRES SCL 输入信号的脉冲宽度小于该字段的值时，I2C 主机控制器忽略此脉冲。该寄存器的值以 I2C 主机控制器时钟周期数为单位。(R/W)

I2C_SDA_FILTER_THRES SDA 输入信号的脉冲宽度小于该字段的值时，I2C 主机控制器忽略此脉冲。该寄存器的值以 I2C 主机控制器时钟周期数为单位。(R/W)

I2C_SCL_FILTER_EN SCL 的滤波使能位。(R/W)

I2C_SDA_FILTER_EN SDA 的滤波使能位。(R/W)

Register 21.15. I2C_CLK_CONF_REG (0x0054)



I2C_SCLK_DIV_NUM 分频系数的整数部分。(R/W)

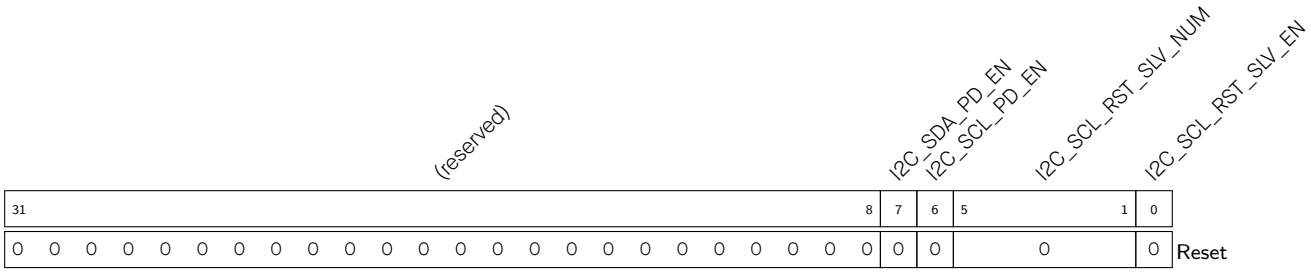
I2C_SCLK_DIV_A 分频系数小数部分的分子。(R/W)

I2C_SCLK_DIV_B 分频系数小数部分的分母。(R/W)

I2C_SCLK_SEL 选择 I2C 主机控制器的时钟源。0：XTAL_CLK；1：RC_FAST_CLK。(R/W)

I2C_SCLK_ACTIVE I2C 主机控制器的时钟开关。(R/W)

Register 21.16. I2C_SCL_SP_CONF_REG (0x0080)



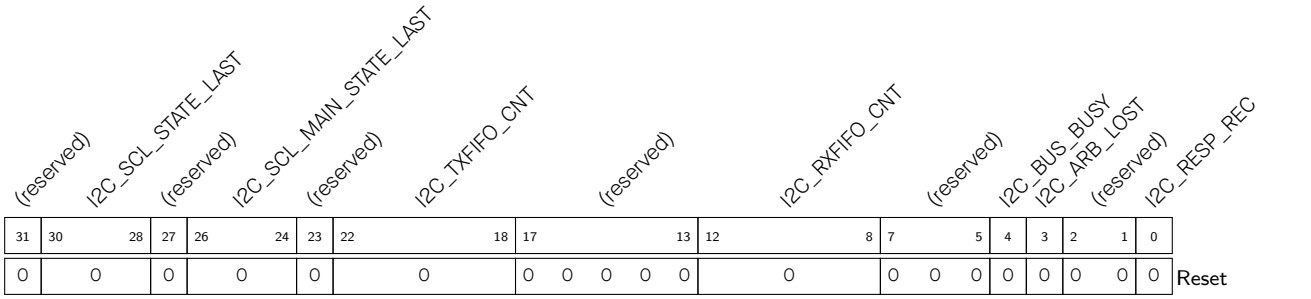
I2C_SCL_RST_SLV_EN I2C 主机处于空闲状态时，置位此位发送 SCL 脉冲。脉冲数量为 I2C_SCL_RST_SLV_NUM[4:0]。(R/W/SC)

I2C_SCL_RST_SLV_NUM 配置生成的 SCL 脉冲。I2C_SCL_RST_SLV_EN 为 1 时有效。(R/W)

I2C_SCL_PD_EN 降低 I2C SCL 输出功耗的使能位。0：正常工作；1：不工作，降低功耗。将 I2C_SCL_FORCE_OUT 和 I2C_SCL_PD_EN 置 1 拉伸 SCL。(R/W)

I2C_SDA_PD_EN 降低 I2C SDA 输出功耗的使能位。0：正常工作；1：不工作，降低功耗。将 I2C_SDA_FORCE_OUT 和 I2C_SDA_PD_EN 置 1 拉伸 SDA 。(R/W)

Register 21.17. I2C_SR_REG (0x0008)



I2C_RESP_REC 接收的 ACK 电平值。0：ACK；1：NACK。(RO)

I2C_ARB_LOST I2C 主机控制器不控制 SCL 线时，该寄存器变为 1。(RO)

I2C_BUS_BUSY 0：I2C 总线处于空闲状态；1：I2C 总线正在传输数据。(RO)

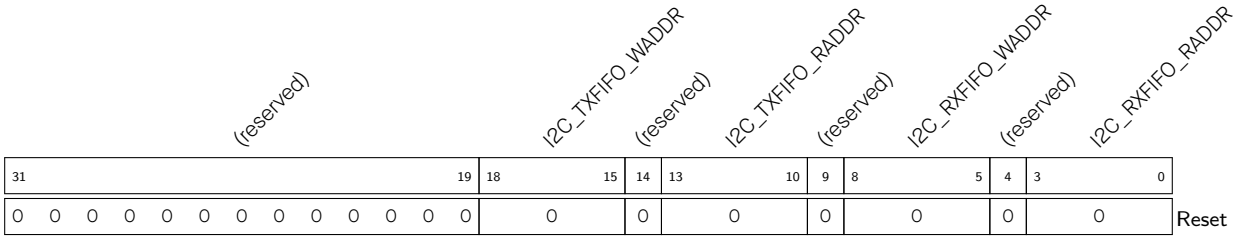
I2C_RXFIFO_CNT 该字段为需发送数据的字节数。(RO)

I2C_TXFIFO_CNT 该字段存储 RAM 接收数据的字节数。(RO)

I2C_SCL_MAIN_STATE_LAST 该字段为 I2C 主机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 该字段为生成 SCL 的状态机状态。0：空闲状态；1：开始；2：下降沿；3：低电平；4：上升沿；5：高电平；6：停止 (RO)

Register 21.18. I2C_FIFO_ST_REG (0x0014)



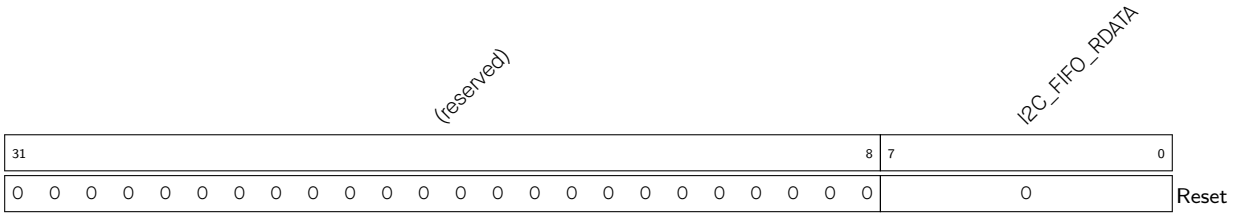
I2C_RXFIFO_RADDR APB 总线读 RX FIFO 的偏移地址。(RO)

I2C_RXFIFO_WADDR I2C 主机控制器接收数据和写 RX FIFO 的偏移地址。(RO)

I2C_TXFIFO_RADDR I2C 主机控制器读 TX FIFO 的偏移地址。(RO)

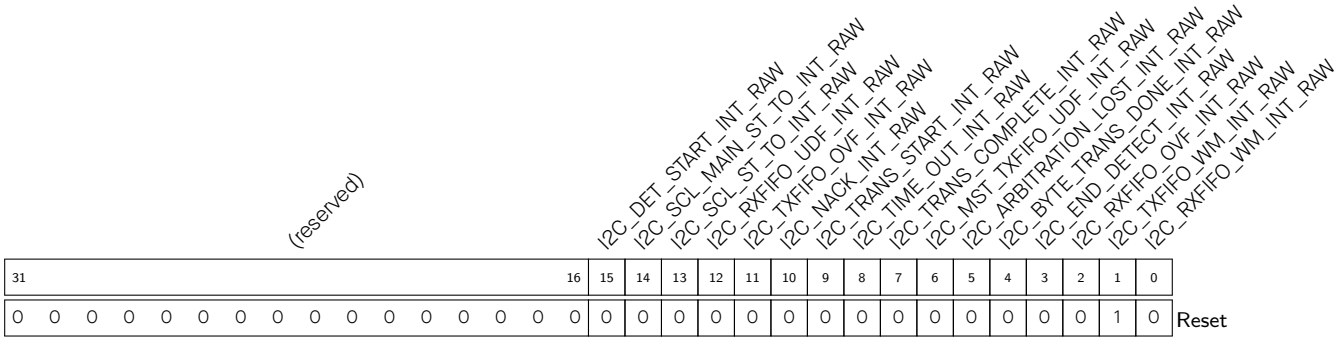
I2C_TXFIFO_WADDR APB 总线写 TX FIFO 的偏移地址。(RO)

Register 21.19. I2C_DATA_REG (0x001C)



I2C_FIFO_RDATA 用于读取 RX FIFO 的数据，或向 TX FIFO 写数据。(R/W)

Register 21.20. I2C_INT_RAW_REG (0x0020)



I2C_RXFIFO_WM_INT_RAW I2C_RXFIFO_WM_INT 的原始中断位。(R/SS/WTC)

I2C_TXFIFO_WM_INT_RAW I2C_TXFIFO_WM_INT 的原始中断位。(R/SS/WTC)

I2C_RXFIFO_OVF_INT_RAW I2C_RXFIFO_OVF_INT 的原始中断位。(R/SS/WTC)

I2C_END_DETECT_INT_RAW I2C_END_DETECT_INT 的原始中断位。(R/SS/WTC)

I2C_BYTE_TRANS_DONE_INT_RAW I2C_BYTE_TRANS_DONE_INT 的原始中断位。(R/SS/WTC)

I2C_ARBITRATION_LOST_INT_RAW I2C_ARBITRATION_LOST_INT 的原始中断位。(R/SS/WTC)

I2C_MST_TXFIFO_UDF_INT_RAW I2C_MST_TXFIFO_UDF_INT 的原始中断位。(R/SS/WTC)

I2C_TRANS_COMPLETE_INT_RAW I2C_TRANS_COMPLETE_INT 的原始中断位。(R/SS/WTC)

I2C_TIME_OUT_INT_RAW I2C_TIME_OUT_INT 的原始中断位。(R/SS/WTC)

I2C_TRANS_START_INT_RAW I2C_TRANS_START_INT 的原始中断位。(R/SS/WTC)

I2C_NACK_INT_RAW I2C_NACK_INT 的原始中断位。(R/SS/WTC)

I2C_TXFIFO_OVF_INT_RAW I2C_TXFIFO_OVF_INT 的原始中断位。(R/SS/WTC)

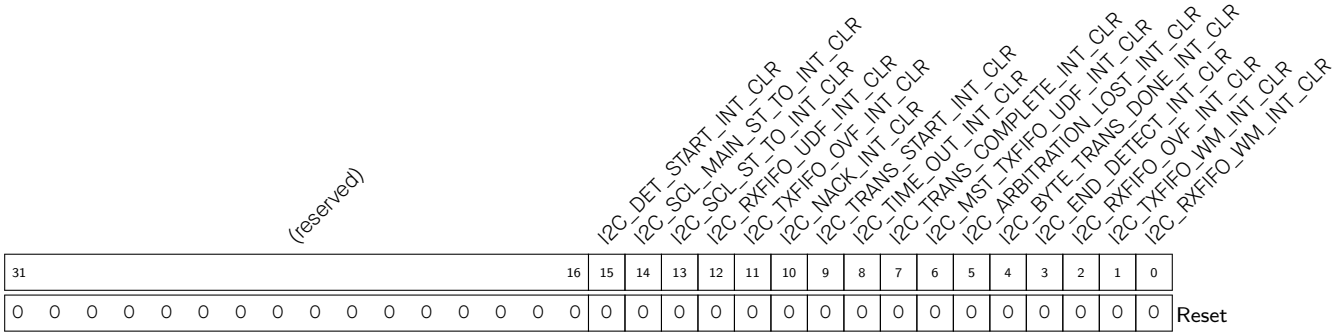
I2C_RXFIFO_UDF_INT_RAW I2C_RXFIFO_UDF_INT 的原始中断位。(R/SS/WTC)

I2C_SCL_ST_TO_INT_RAW I2C_SCL_ST_TO_INT 的原始中断位。(R/SS/WTC)

I2C_SCL_MAIN_ST_TO_INT_RAW I2C_SCL_MAIN_ST_TO_INT 的原始中断位。(R/SS/WTC)

I2C_DET_START_INT_RAW I2C_DET_START_INT 的原始中断位。(R/SS/WTC)

Register 21.21. I2C_INT_CLR_REG (0x0024)



I2C_RXFIFO_WM_INT_CLR 置位此位，清除 I2C_RXFIFO_WM_INT 中断。(WT)

I2C_TXFIFO_WM_INT_CLR 置位此位，清除 I2C_TXFIFO_WM_INT 中断。(WT)

I2C_RXFIFO_OVF_INT_CLR 置位此位，清除 I2C_RXFIFO_OVF_INT 中断。(WT)

I2C_END_DETECT_INT_CLR 置位此位，清除 I2C_END_DETECT_INT 中断。(WT)

I2C_BYTE_TRANS_DONE_INT_CLR 置位此位，清除 I2C_BYTE_TRANS_DONE_INT 中断。(WT)

I2C_ARBITRATION_LOST_INT_CLR 置位此位，清除 I2C_ARBITRATION_LOST_INT 中断。(WT)

I2C_MST_TXFIFO_UDF_INT_CLR 置位此位，清除 I2C_MST_TXFIFO_UDF_INT 中断。(WT)

I2C_TRANS_COMPLETE_INT_CLR 置位此位，清除 I2C_TRANS_COMPLETE_INT 中断。(WT)

I2C_TIME_OUT_INT_CLR 置位此位，清除 I2C_TIME_OUT_INT 中断。(WT)

I2C_TRANS_START_INT_CLR 置位此位，清除 I2C_TRANS_START_INT 中断。(WT)

I2C_NACK_INT_CLR 置位此位，清除 I2C_NACK_INT 中断。(WT)

I2C_TXFIFO_OVF_INT_CLR 置位此位，清除 I2C_TXFIFO_OVF_INT 中断。(WT)

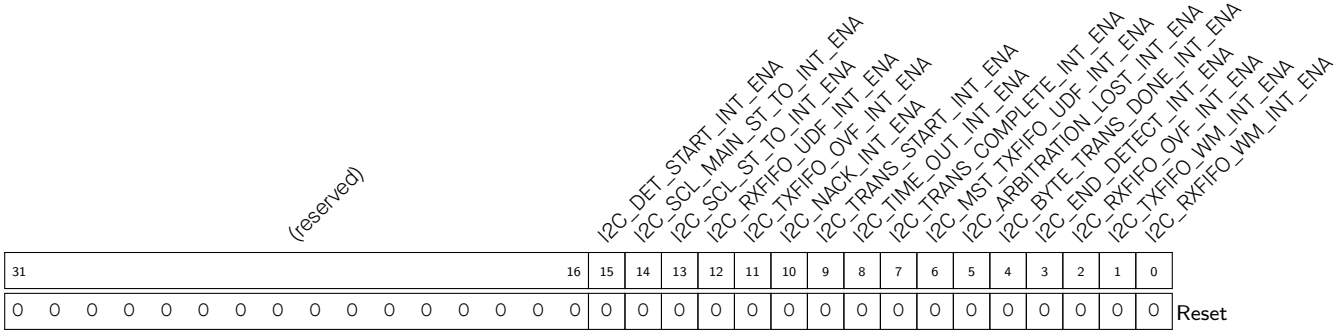
I2C_RXFIFO_UDF_INT_CLR 置位此位，清除 I2C_RXFIFO_UDF_INT 中断。(WT)

I2C_SCL_ST_TO_INT_CLR 置位此位，清除 I2C_SCL_ST_TO_INT 中断。(WT)

I2C_SCL_MAIN_ST_TO_INT_CLR 置位此位，清除 I2C_SCL_MAIN_ST_TO_INT 中断。(WT)

I2C_DET_START_INT_CLR 置位此位，清除 I2C_DET_START_INT 中断。(WT)

Register 21.22. I2C_INT_ENA_REG (0x0028)



I2C_RXFIFO_WM_INT_ENA I2C_RXFIFO_WM_INT 的使能位。(R/W)

I2C_TXFIFO_WM_INT_ENA I2C_TXFIFO_WM_INT 的使能位。(R/W)

I2C_RXFIFO_OVF_INT_ENA I2C_RXFIFO_OVF_INT 的使能位。(R/W)

I2C_END_DETECT_INT_ENA I2C_END_DETECT_INT 的使能位。(R/W)

I2C_BYTE_TRANS_DONE_INT_ENA I2C_BYTE_TRANS_DONE_INT 的使能位。(R/W)

I2C_ARBITRATION_LOST_INT_ENA I2C_ARBITRATION_LOST_INT 的使能位。(R/W)

I2C_MST_TXFIFO_UDF_INT_ENA I2C_MST_TXFIFO_UDF_INT 的使能位。(R/W)

I2C_TRANS_COMPLETE_INT_ENA I2C_TRANS_COMPLETE_INT 的使能位。(R/W)

I2C_TIME_OUT_INT_ENA I2C_TIME_OUT_INT 的使能位。(R/W)

I2C_TRANS_START_INT_ENA I2C_TRANS_START_INT 的使能位。(R/W)

I2C_NACK_INT_ENA I2C_NACK_INT 的使能位。(R/W)

I2C_TXFIFO_OVF_INT_ENA I2C_TXFIFO_OVF_INT 的使能位。(R/W)

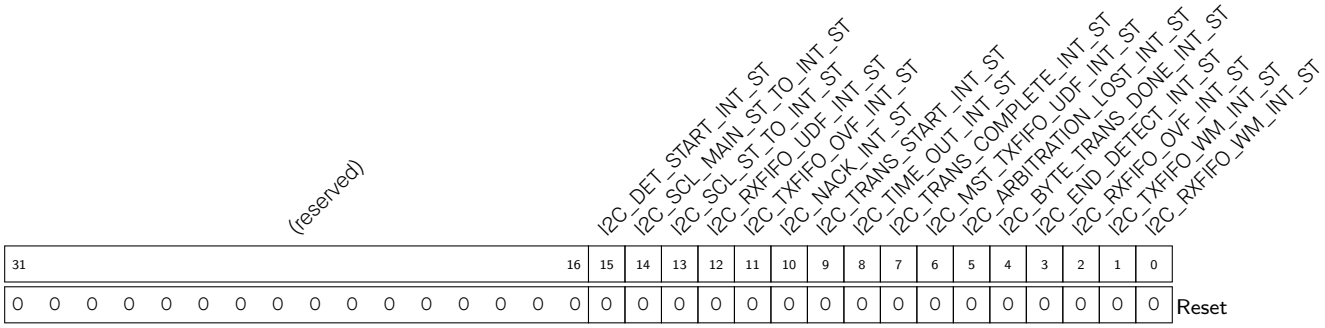
I2C_RXFIFO_UDF_INT_ENA I2C_RXFIFO_UDF_INT 的使能位。(R/W)

I2C_SCL_ST_TO_INT_ENA I2C_SCL_ST_TO_INT 的使能位。(R/W)

I2C_SCL_MAIN_ST_TO_INT_ENA I2C_SCL_MAIN_ST_TO_INT 的使能位。(R/W)

I2C_DET_START_INT_ENA I2C_DET_START_INT 的使能位。(R/W)

Register 21.23. I2C_INT_STATUS_REG (0x002C)



I2C_RXFIFO_WM_INT_ST I2C_RXFIFO_WM_INT 的屏蔽状态位。(RO)

I2C_TXFIFO_WM_INT_ST I2C_TXFIFO_WM_INT 的屏蔽状态位。(RO)

I2C_RXFIFO_OVF_INT_ST I2C_RXFIFO_OVF_INT 的屏蔽状态位。(RO)

I2C_END_DETECT_INT_ST I2C_END_DETECT_INT 的屏蔽状态位。(RO)

I2C_BYTE_TRANS_DONE_INT_ST I2C_BYTE_TRANS_DONE_INT 的屏蔽状态位。(RO)

I2C_ARBITRATION_LOST_INT_ST I2C_ARBITRATION_LOST_INT 的屏蔽状态位。(RO)

I2C_MST_TXFIFO_UDF_INT_ST I2C_MST_TXFIFO_UDF_INT 的屏蔽状态位。(RO)

I2C_TRANS_COMPLETE_INT_ST I2C_TRANS_COMPLETE_INT 的屏蔽状态位。(RO)

I2C_TIME_OUT_INT_ST I2C_TIME_OUT_INT 的屏蔽状态位。(RO)

I2C_TRANS_START_INT_ST I2C_TRANS_START_INT 的屏蔽状态位。(RO)

I2C_NACK_INT_ST I2C_NACK_INT 的屏蔽状态位。(RO)

I2C_TXFIFO_OVF_INT_ST I2C_TXFIFO_OVF_INT 的屏蔽状态位。(RO)

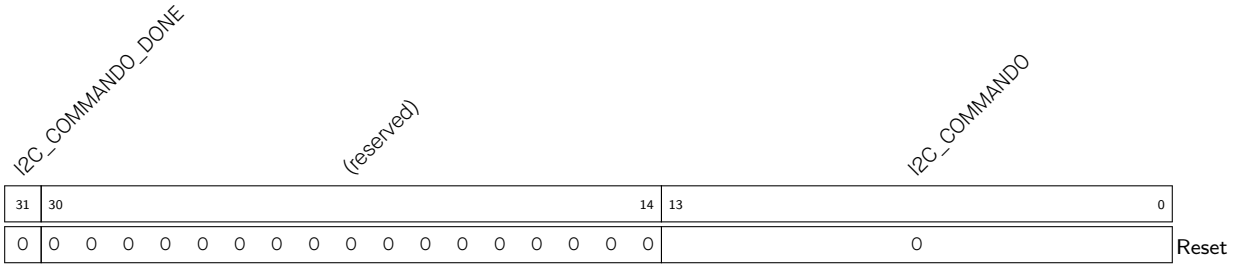
I2C_RXFIFO_UDF_INT_ST I2C_RXFIFO_UDF_INT 的屏蔽状态位。(RO)

I2C_SCL_ST_TO_INT_ST I2C_SCL_ST_TO_INT 的屏蔽状态位。(RO)

I2C_SCL_MAIN_ST_TO_INT_ST I2C_SCL_MAIN_ST_TO_INT 的屏蔽状态位。(RO)

I2C_DET_START_INT_ST I2C_DET_START_INT 的屏蔽状态位。(RO)

Register 21.24. I2C_COMD0_REG (0x0058)



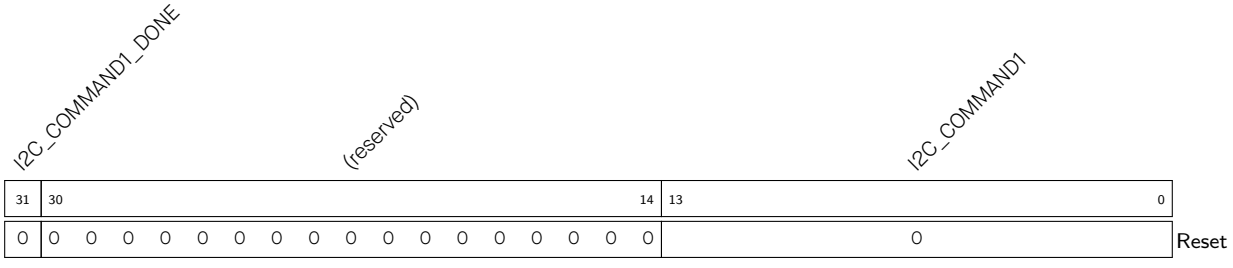
I2C_COMMAND0 命令寄存器 0 的内容。该命令包括三个部分：

- op_code 为命令，1：WRITE；2：STOP；3：READ；4：END；6：RSTART。
- byte_num 表示需发送或接收的字节数。
- ack_check_en、ack_exp 和 ack 用于控制 ACK 位。更多信息详见章节 21.4.8。

(R/W)

I2C_COMMAND0_DONE 完成命令 0 时，该位翻转为高电平。(R/W/SS)

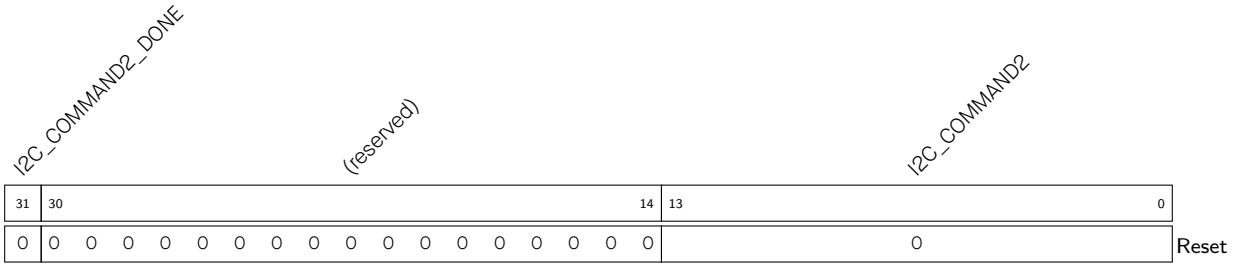
Register 21.25. I2C_COMD1_REG (0x005C)



I2C_COMMAND1 命令寄存器 1 的内容，同 I2C_COMMAND0。(R/W)

I2C_COMMAND1_DONE 完成命令 1 时，该位翻转为高电平。(R/W/SS)

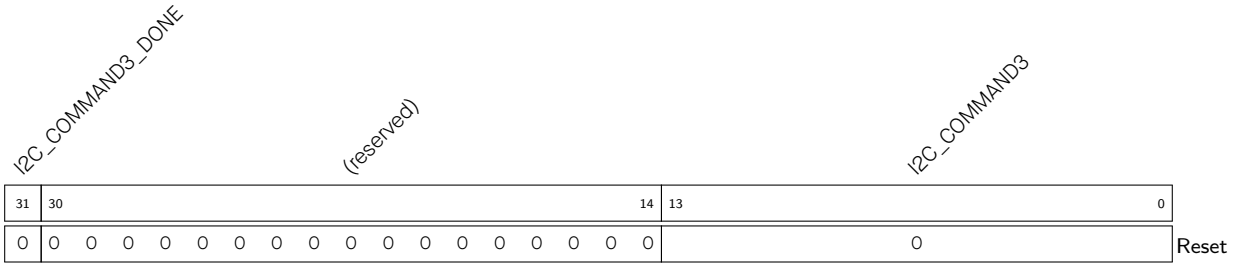
Register 21.26. I2C_COMD2_REG (0x0060)



I2C_COMMAND2 命令寄存器 2 的内容，同 I2C_COMMAND0。(R/W)

I2C_COMMAND2_DONE 完成命令 2 时，该位翻转为高电平。(R/W/SS)

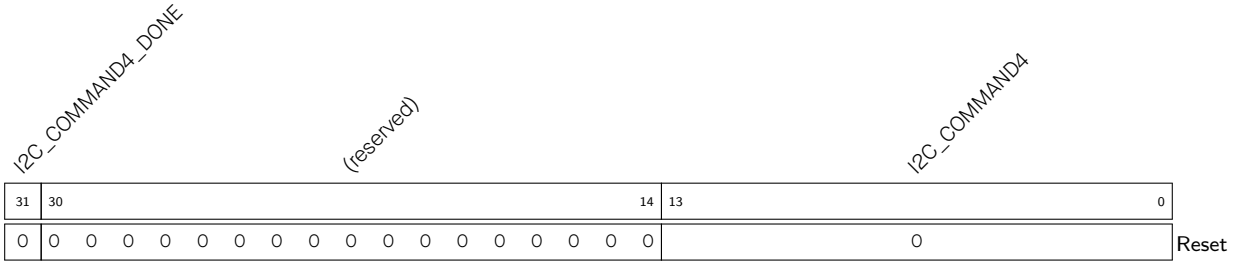
Register 21.27. I2C_COMD3_REG (0x0064)



I2C_COMMAND3 命令寄存器 3 的内容，同 I2C_COMMAND0。(R/W)

I2C_COMMAND3_DONE 完成命令 3 时，该位翻转为高电平。(R/W/SS)

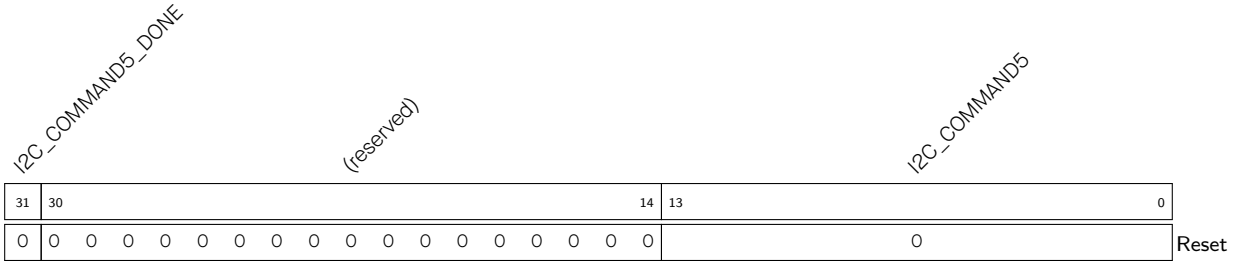
Register 21.28. I2C_COMD4_REG (0x0068)



I2C_COMMAND4 命令寄存器 4 的内容，同 I2C_COMMAND0。(R/W)

I2C_COMMAND4_DONE 完成命令 4 时，该位翻转为高电平。(R/W/SS)

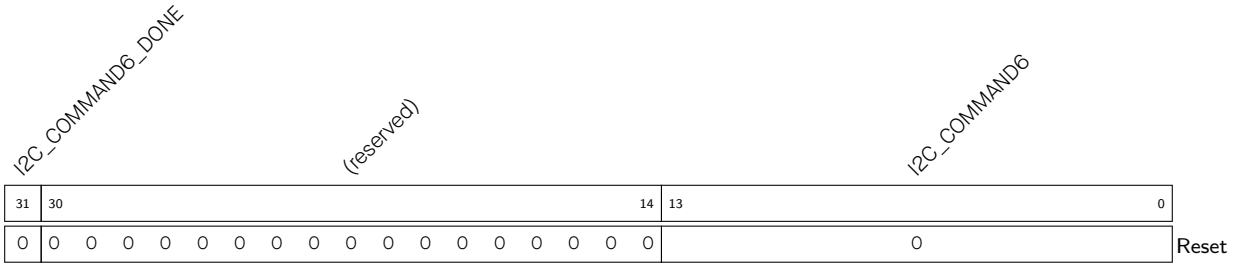
Register 21.29. I2C_COMD5_REG (0x006C)



I2C_COMMAND5 命令寄存器 5 的内容，同 I2C_COMMAND0。(R/W)

I2C_COMMAND5_DONE 完成命令 5 时，该位翻转为高电平。(R/W/SS)

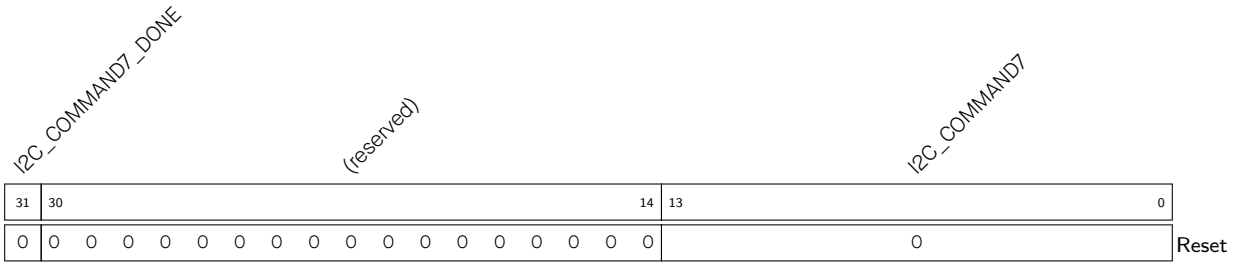
Register 21.30. I2C_COMD6_REG (0x0070)



I2C_COMMAND6 命令寄存器 6 的内容，同 I2C_COMMAND0。(R/W)

I2C_COMMAND6_DONE 完成命令 6 时，该位翻转为高电平。(R/W/SS)

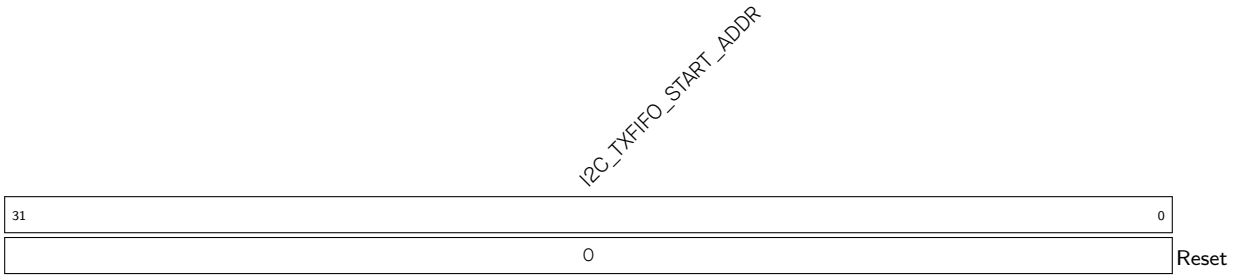
Register 21.31. I2C_COMD7_REG (0x0074)



I2C_COMMAND7 命令寄存器 7 的内容，同 I2C_COMMAND0。(R/W)

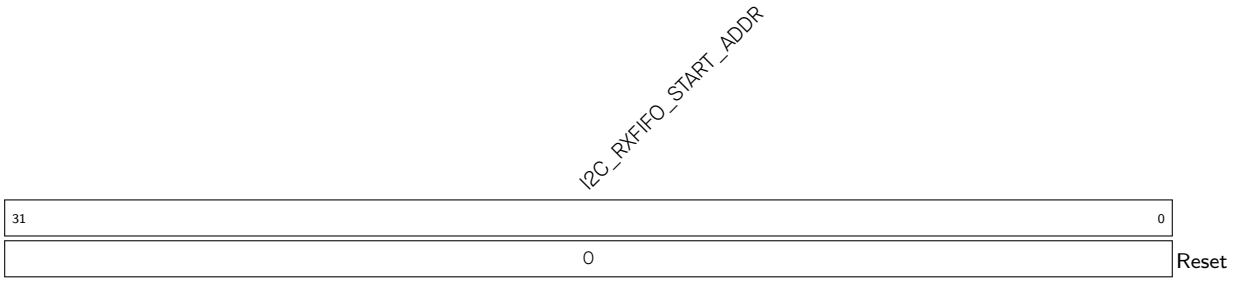
I2C_COMMAND7_DONE 完成命令 7 时，该位翻转为高电平。(R/W/SS)

Register 21.32. I2C_TXFIFO_START_ADDR_REG (0x0100)



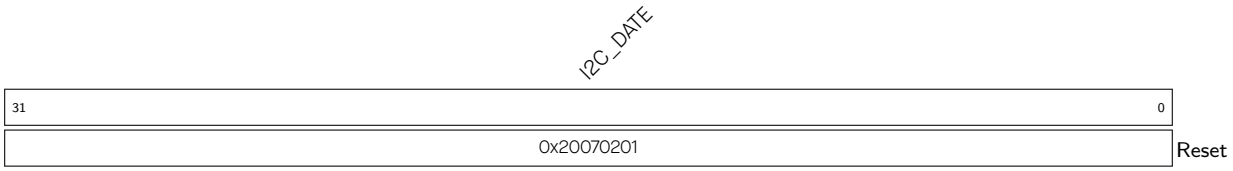
I2C_TXFIFO_START_ADDR I2C TX FIFO 的起始地址。(HRO)

Register 21.33. I2C_RXFIFO_START_ADDR_REG (0x0180)



I2C_RXFIFO_START_ADDR I2C RX FIFO 的起始地址。(HRO)

Register 21.34. I2C_DATE_REG (0x00F8)



I2C_DATE 版本控制寄存器。(R/W)

22 LED PWM 控制器 (LEDC)

22.1 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。该外设也可成生成 PWM 信号用作其他用途。

22.2 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 14 位
- PWM 输出信号的相位和占空比可调节
- 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
- 低功耗模式 (Light-sleep mode) 下可输出 PWM 信号
- 三个可分频的时钟源:
  - PLL_F60M_CLK
  - RC_FAST_CLK
  - XTAL_CLK
- 四个独立定时器，可实现小数分频

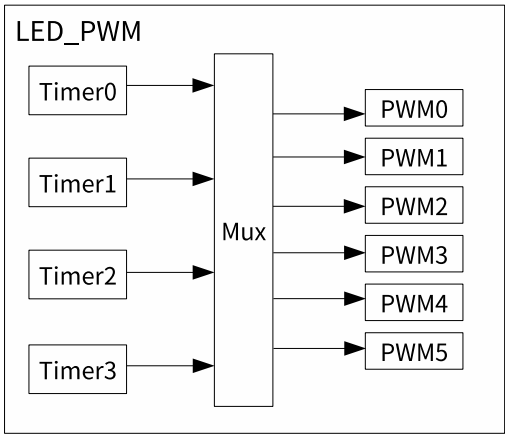


图 22-1. LED PWM 控制器架构

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）。六个 PWM 生成器的功能和运行方式也相同，下文将统称为 PWM*n*（n 的范围是 0 到 5）。

22.3 功能描述

22.3.1 架构

图 22-1为 LED PWM 控制器的架构。

四个定时器可独立配置（即每个定时器可配置自己的时钟分频器和计数器最大值），每个定时器内部有一个时基计数器（即基于基准时钟周期计数的计数器）。每个 PWM 生成器在四个定时器中择一，以该定时器的计数值为基准生成 PWM 信号。

图 22-2 为定时器和 PWM 生成器的主要功能块。

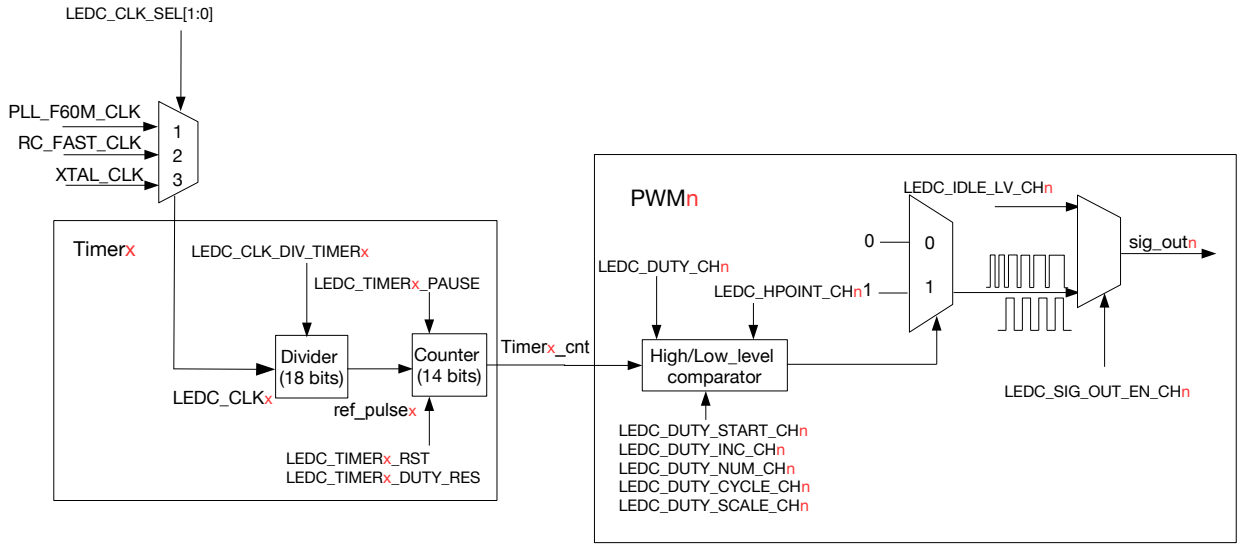


图 22-2. 定时器和 PWM 生成器功能块

22.3.2 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 22-2 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。

22.3.2.1 时钟源

软件配置的 LED PWM 寄存器使用 APB_CLK 时钟。更多关于 APB_CLK 的信息，详见章节 6 *复位和时钟*。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 SYSTEM_PERIP_CLK_EN0_REG 寄存器的 SYSTEM_LEDC_CLK_EN 使能，通过软件置位 SYSTEM_PERIP_RST_EN0_REG 寄存器的 SYSTEM_LEDC_RST 位复位。更多信息，请参阅章节 13 *系统寄存器 (SYSTEM)* 的表 13-1。

LED PWM 控制器的定时器有三个时钟源信号可以选择：PLL_F60M_CLK、RC_FAST_CLK 和 XTAL_CLK（更多有关时钟源的信息详见章节 6 *复位和时钟*）。为 LEDC_CLK*x* 选择时钟源信号的配置如下：

- PLL_F60M_CLK：将 LEDC_CLK_SEL[1:0] 置 1
- RC_FAST_CLK：将 LEDC_CLK_SEL[1:0] 置 2
- XTAL_CLK：将 LEDC_CLK_SEL[1:0] 置 3

之后，LEDC_CLKx 信号会进入时钟分频器。

22.3.2.2 时钟分频器配置

LEDC_CLKx 信号传输到时钟分频器，产生 ref_pulsex 信号供计数器使用。ref_pulsex 的频率等于 LEDC_CLKx 的频率经分频系数 LEDC_CLK_DIV 分频后的结果（见图 22-2）。

分频系数 LEDC_CLK_DIV 为小数，因此其值可为非整数，使频率更加精确。分频系数 LEDC_CLK_DIV 可根据下列等式配置：

$$LEDC\_CLK\_DIV = A + \frac{B}{256}$$

- 整数部分 *A* 为 LEDC_CLK_DIV_TIMERx 字段的高 10 位（即 LEDC_TIMERx_CONF_REG[21:12]）
- 小数部分 *B* 为 LEDC_CLK_DIV_TIMERx 字段的低 8 位（即 LEDC_TIMERx_CONF_REG[11:4]）

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_CLKx 时钟周期产生一个 ref_pulsex 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_CLKx 时钟周期和 (*A*+1) 个 LEDC_CLKx 时钟周期轮流进行非整数分频。这样一来，ref_pulsex 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_pulsex 时钟脉冲中：

- 有 *B* 个以 (*A*+1) 个 LEDC_CLKx 时钟周期分频
- 有 (256-*B*) 个以 *A* 个 LEDC_CLKx 时钟周期分频
- 以 (*A*+1) 个 LEDC_CLKx 时钟周期分频的时钟脉冲均匀分布在以 *A* 分频的时钟脉冲中

图 22-3 展示了 LEDC_CLK_DIV 分频系数非整数时，LEDC_CLKx 时钟周期和 ref_pulsex 时钟脉冲的关系。

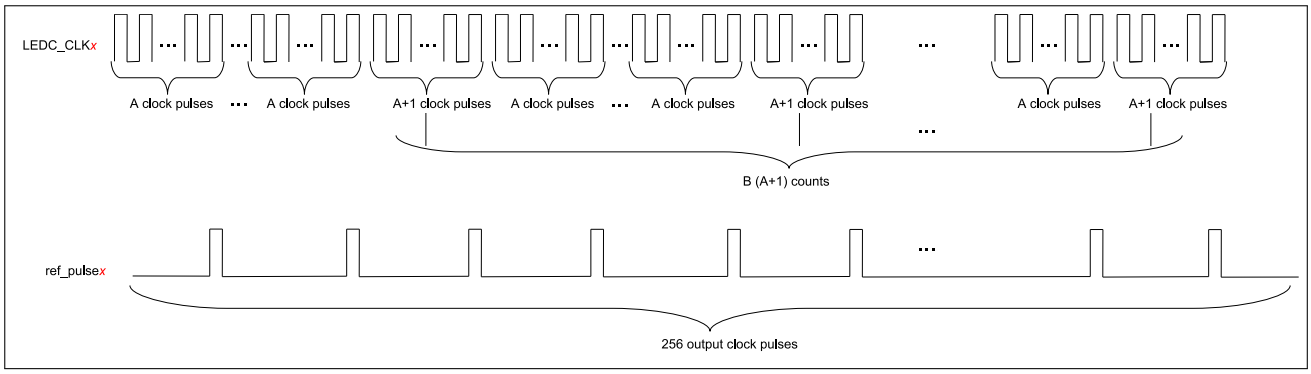


图 22-3. LEDC_CLK_DIV 非整数时的分频

在运行时改变定时器时钟的分频系数，需先配置 LEDC_CLK_DIV_TIMERx 字段，然后置位 LEDC_TIMERx_PARA_UP 字段应用新配置。新配置会在计数器下次溢出时生效。LEDC_TIMERx_PARA_UP 字段由硬件自动清除。

22.3.2.3 14 位计数器

每个定时器有一个以 ref_pulsex 为基准时钟的 14 位时基计数器（见图 22-2）。LEDC_TIMERx_DUTY_RES 字段用于配置 14 位计数器的最大值。因此，PWM 信号的最大精确度为 14 位。计数器最大可计数至 $(2^{LEDC\_TIMERx\_DUTY\_RES} - 1)$，然后溢出重新从 0 开始计数。软件可以读取、复位、暂停计数器。

计数器可在每次溢出时触发 (LEDC_TIMERx_OVF_INT) 中断，这个中断为硬件自动产生，不需要配置。计数器也可配置为在溢出 ($LEDC\_OVF\_NUM\_CHn + 1$) 次时触发 LEDC_OVF_CNT_CHn_INT 中断，该中断配置步骤如下：

1. 配置 LEDC_TIMER_SEL_CHn 为 PWM 生成器选择该计数器
2. 置位 LEDC_OVF_CNT_EN_CHn 使能计数器
3. 把 LEDC_OVF_NUM_CHn 的值设为计数器触发中断的溢出次数减 1
4. 置位 LEDC_OVF_CNT_CHn_INT_ENA 使能溢出中断
5. 置位 LEDC_TIMERx_DUTY_RES 使能定时器，等待 LEDC_OVF_CNT_CHn_INT 中断产生

如图 22-2 所示，PWM 生成器输出信号 sig_outn 的频率取决于定时器时钟源 LEDC_CLKx 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_TIMERx_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 22-1 列出了常用配置频率及其对应精度。

表 22-1. 常用配置频率及精度

LEDC_CLKx	PWM 频率	最高精度（位）[1]	最低精度（位）[2]
PLL_F60M_CLK (60 MHz)	1 kHz	14	5
PLL_F60M_CLK (60 MHz)	5 kHz	13	3
PLL_F60M_CLK (60 MHz)	10 kHz	12	2
XTAL_CLK (40 MHz)	1 kHz	14	5
XTAL_CLK (40 MHz)	4 kHz	13	3
RC_FAST_CLK (17.5 MHz)	1 kHz	14	4
RC_FAST_CLK (17.5 MHz)	1.75 kHz	13	3

[1] 最高精度指时钟分频系数 LEDC_CLK_DIV 为 1 时的精度。如果经公式计算出的最高精度超过了计数器位宽 14 位，则最高精度为 14。

[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。

在运行时改变计数器的最大值，需先置位 LEDC_TIMERx_DUTY_RES 字段，然后置位 LEDC_TIMERx_PARA_UP 字段。新的配置在计数器下一次溢出时生效。如果重新配置 LEDC_OVF_CNT_EN_CHn 字段，需置位 LEDC_PARA_UP_CHn 应用新配置。总之，更改配置时需置位 LEDC_TIMERx_PARA_UP 或 LEDC_PARA_UP_CHn 应用新配置。
LEDC_TIMERx_PARA_UP 和 LEDC_PARA_UP_CHn 字段由硬件自动清除。

22.3.3 PWM 生成器

要生成 PWM 信号，PWM 生成器 (PWMn) 需选择一个定时器 (Timerx)。每个 PWM 生成器均可通过置位 LEDC_TIMER_SEL_CHn 单独配置，在四个定时器中选择一个输出 PWM 信号。

如图 22-2 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 14 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 22-4 展示了如何使用 Hpoint*n* 和 Lpoint*n* 生成占空比固定的 PWM 信号。

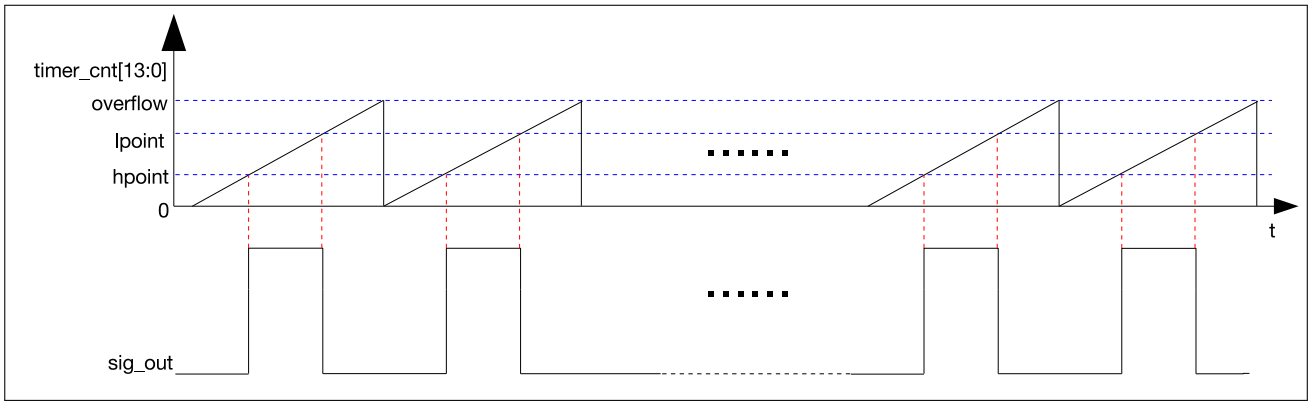


图 22-4. LED PWM 输出信号图

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*[18:4] 和 LEDC_HPOINT_CH*n* 的和。通过配置 LEDC_DUTY_CH*n*[18:4] 和 LEDC_HPOINT_CH*n* 两个字段，可设置 PWM 输出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*[18:4] 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*[18:4] 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。

22.3.4 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。如果开启占空比渐变功能，Lpoint*n* 的值会在计数器溢出固定次数后递增或递减。图 22-5 展示了占空比渐变功能。

占空比渐变功能可通过以下寄存器字段配置：

- LEDC_DUTY_CH*n* 用于设置 Lpoint*n* 的初始值。
- LEDC_DUTY_START_CH*n* 置 1 或清零，使能或关闭占空比渐变功能。
- LEDC_DUTY_CYCLE_CH*n* 用于设置 Lpoint*n* 在计数器溢出多少次时递增或递减。也就是说，Lpoint*n* 会在计数器溢出 LEDC_DUTY_CYCLE_CH*n* 次时递增或递减。
- LEDC_DUTY_INC_CH*n* 置 1 或清零，Lpoint*n* 递增或递减。

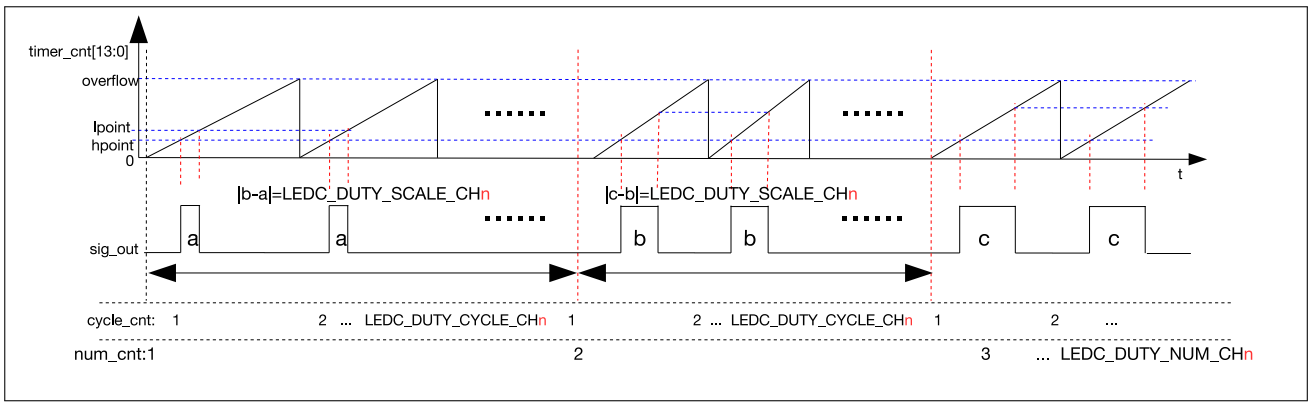


图 22-5. 输出信号占空比渐变图

- LEDC_DUTY_SCALE_CH*n* 用于设置 Lpoint*n* 递增或递减的值。
- LEDC_DUTY_NUM_CH*n* 用于设置占空比渐变停止前，Lpoint*n* 递增或递减的最大次数。

如果重新配置 LEDC_DUTY_CH*n*、LEDC_DUTY_START_CH*n*、LEDC_DUTY_CYCLE_CH*n*、LEDC_DUTY_INC_CH*n*、LEDC_DUTY_SCALE_CH*n* 和 LEDC_DUTY_NUM_CH*n*字段，需置位 LEDC_PARA_UP_CH*n* 应用新配置。
LEDC_PARA_UP_CH*n* 置位后，新配置立即生效。LEDC_TIMER*x*_PARA_UP 字段由硬件自动清除。

22.3.5 中断

- LEDC_OVF_CNT_CH*n*_INT：定时器计数器溢出 (LEDC_OVF_NUM_CH*n* + 1) 次且寄存器 LEDC_OVF_CNT_EN_CH*n* 置 1 时触发中断。
- LEDC_DUTY_CHNG_END_CH*n*_INT：PWM 生成器渐变完成后触发中断。
- LEDC_TIMER*x*_OVF_INT：定时器达到最大计数值时触发中断。

22.4 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 3 系统和存储器 中的表 3-3。

请查看章节 寄存器的访问类型，了解“访问”列缩写的含义。

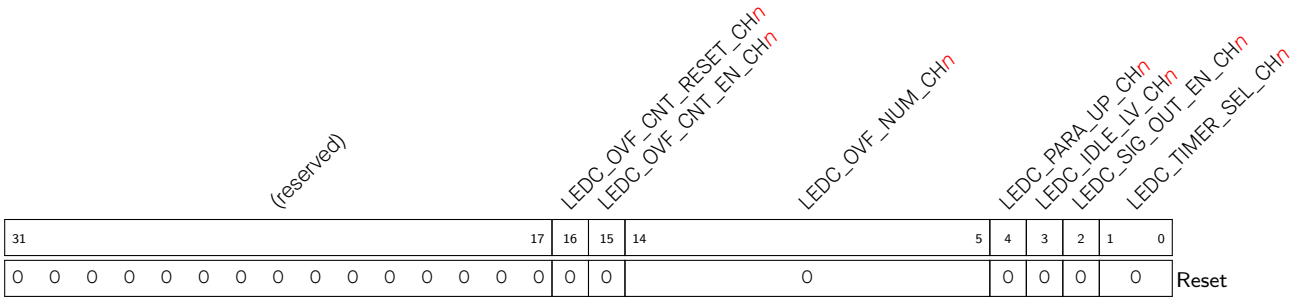
名称	描述	地址	访问
配置寄存器			
LEDC_CH0_CONF0_REG	通道 0 的配置寄存器 0	0x0000	varies
LEDC_CH0_CONF1_REG	通道 0 的配置寄存器 1	0x000C	varies
LEDC_CH1_CONF0_REG	通道 1 的配置寄存器 0	0x0014	varies
LEDC_CH1_CONF1_REG	通道 1 的配置寄存器 1	0x0020	varies
LEDC_CH2_CONF0_REG	通道 2 的配置寄存器 0	0x0028	varies
LEDC_CH2_CONF1_REG	通道 2 的配置寄存器 1	0x0034	varies
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	varies
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	varies
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	varies
LEDC_CONF_REG	LEDC 全局配置寄存器	0x00D0	R/W
高位点寄存器			
LEDC_CH0_HPOINT_REG	通道 0 的高位点寄存器	0x0004	R/W
LEDC_CH1_HPOINT_REG	通道 1 的高位点寄存器	0x0018	R/W
LEDC_CH2_HPOINT_REG	通道 2 的高位点寄存器	0x002C	R/W
LEDC_CH3_HPOINT_REG	通道 3 的高位点寄存器	0x0040	R/W
LEDC_CH4_HPOINT_REG	通道 4 的高位点寄存器	0x0054	R/W
LEDC_CH5_HPOINT_REG	通道 5 的高位点寄存器	0x0068	R/W
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	定时器 0 配置	0x00A0	varies
LEDC_TIMER0_VALUE_REG	定时器 0 的当前计数器值	0x00A4	RO
LEDC_TIMER1_CONF_REG	定时器 1 配置	0x00A8	varies

名称	描述	地址	访问
LEDC_TIMER1_VALUE_REG	定时器 1 的当前计数器值	0x00AC	RO
LEDC_TIMER2_CONF_REG	定时器 2 配置	0x00B0	varies
LEDC_TIMER2_VALUE_REG	定时器 2 的当前计数器值	0x00B4	RO
LEDC_TIMER3_CONF_REG	定时器 3 配置	0x00B8	varies
LEDC_TIMER3_VALUE_REG	定时器 3 的当前计数器值	0x00BC	RO
中断寄存器			
LEDC_INT_RAW_REG	原始中断状态	0x00C0	R/WTC/SS
LEDC_INT_ST_REG	屏蔽中断状态	0x00C4	RO
LEDC_INT_ENA_REG	中断使能位	0x00C8	R/W
LEDC_INT_CLR_REG	中断清除位	0x00CC	WT
版本寄存器			
LEDC_DATE_REG	版本控制寄存器	0x00FC	R/W

22.5 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 3 系统和存储器 中的表 3-3。

Register 22.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14**n*)



LEDC_TIMER_SEL_CH*n* 用于选择通道 *n* 的定时器。

0：选择定时器 0

1：选择定时器 1

2：选择定时器 2

3：选择定时器 3 (R/W)

LEDC_SIG_OUT_EN_CH*n* 置位此位，使能通道 *n* 的信号输出。(R/W)

LEDC_IDLE_LV_CH*n* 控制通道 *n* 不工作时（LEDC_SIG_OUT_EN_CH*n* 为 0 时）的输出电平。(R/W)

LEDC_PARA_UP_CH*n* 用于更新通道 *n* 的下列字段，由硬件自动清除。(WT)

- LEDC_HPOINT_CH*n*
- LEDC_DUTY_START_CH*n*
- LEDC_SIG_OUT_EN_CH*n*
- LEDC_TIMER_SEL_CH*n*
- LEDC_DUTY_NUM_CH*n*
- LEDC_DUTY_CYCLE_CH*n*
- LEDC_DUTY_SCALE_CH*n*
- LEDC_DUTY_INC_CH*n*
- LEDC_OVF_CNT_EN_CH*n*

见下页...

Register 22.1. LEDC_CH*n*_CONF0_REG (*n*: 0-5) (0x0000+0x14*n*)

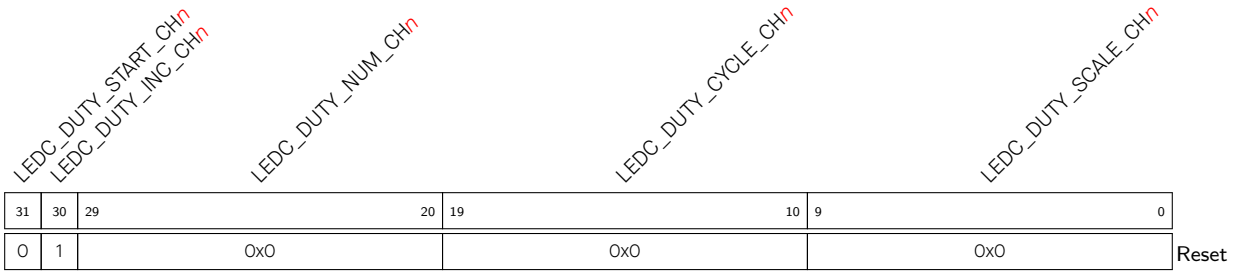
接上页...

LEDC_OVF_NUM_CH*n* 用于配置定时器溢出次数的最大值减 1。通道 *n* 的定时器溢出次数达到 (LEDC_OVF_NUM_CH*n* +1) 次时，触发 LEDC_OVF_CNT_CH*n*_INT 中断。(R/W)

LEDC_OVF_CNT_EN_CH*n* 用于使能计数器，计算通道 *n* 选择的定时器溢出的次数。(R/W)

LEDC_OVF_CNT_RESET_CH*n* 置位此位，复位通道 *n* 的定时器溢出计数器。(WT)

Register 22.2. LEDC_CH*n*_CONF1_REG (*n*: 0-5) (0x000C+0x14*n*)



LEDC_DUTY_SCALE_CH*n* 用于配置渐变时占空比的步长变化。(R/W)

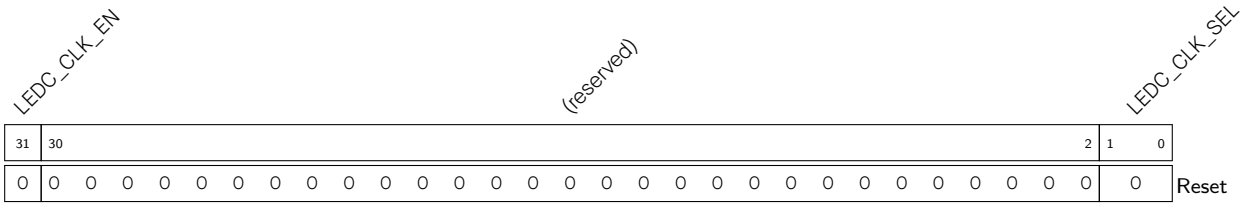
LEDC_DUTY_CYCLE_CH*n* 通道 *n* 占空比每隔 LEDC_DUTY_CYCLE_CH*n* 周期变化一次。(R/W)

LEDC_DUTY_NUM_CH*n* 用于设置占空比渐变停止前递增或递减的最大次数。(R/W)

LEDC_DUTY_INC_CH*n* 决定了通道 *n* 输出信号的占空比是递增还是递减。1：递增；0：递减。(R/W)

LEDC_DUTY_START_CH*n* 此位置 1 时，LEDC_CH*n*_CONF1_REG 中的其他字段在定时器下次溢出时生效。(R/W/SC)

Register 22.3. LEDC_CONF_REG (0x00D0)

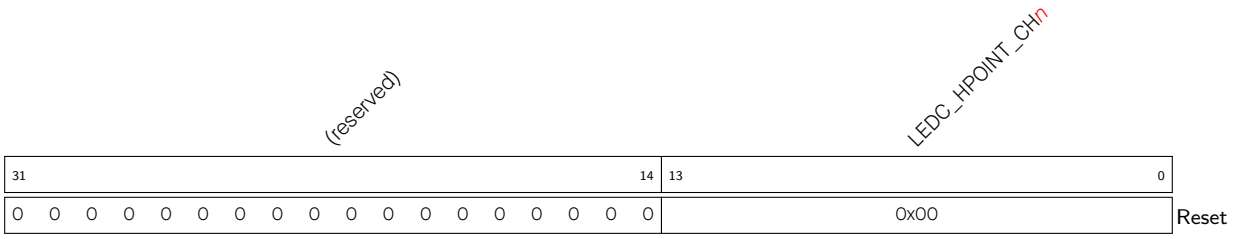


LEDC_CLK_SEL 用于设置 4 个定时器共同的时钟源。1：PLL_F60M_CLK；2：RC_FAST_CLK；3：XTAL_CLK。(R/W)

LEDC_CLK_EN 用于控制时钟。

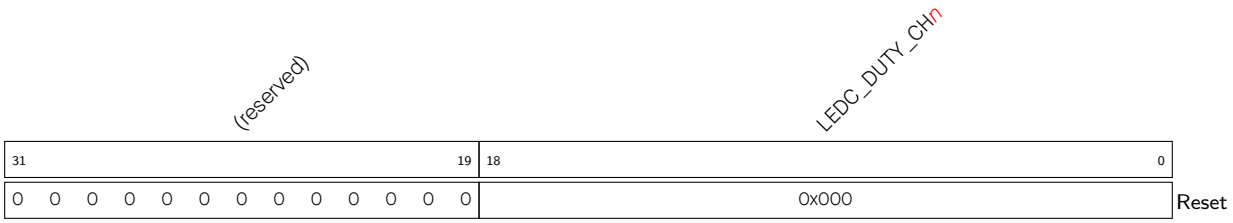
1：强制开启寄存器时钟。0：仅在应用写寄存器时支持时钟。(R/W)

Register 22.4. LEDC_CH*n*_HPOINT_REG (*n*: 0-5) (0x0004+0x14*n*)



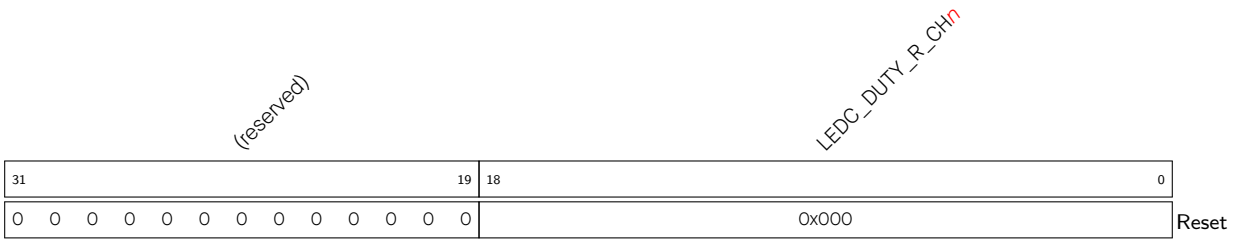
LEDC_HPOINT_CH*n* 该通道所选定时器计数值达到该字段的值时，输出信号翻转为高电平。(R/W)

Register 22.5. LEDC_CH*n*_DUTY_REG (*n*: 0-5) (0x0008+0x14*n*)



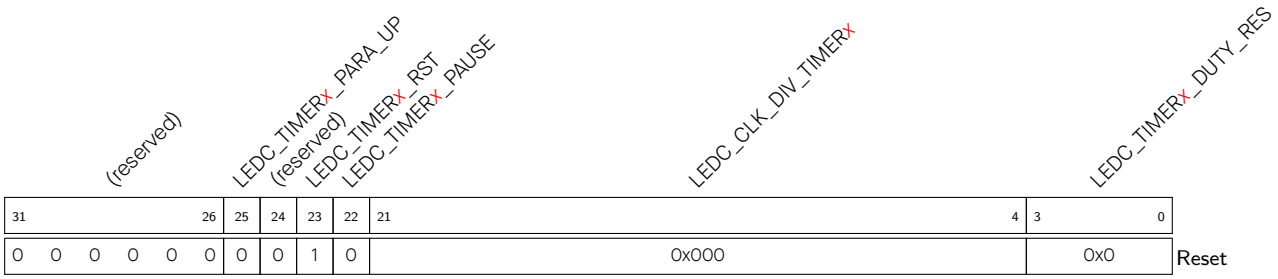
LEDC_DUTY_CH*n* 通过控制低位点改变输出信号占空比。该通道所选定时器达到低位点时，输出信号翻转为低电平。(R/W)

Register 22.6. LEDC_CH*n*_DUTY_R_REG (*n*: 0-5) (0x0010+0x14*n*)



LEDC_DUTY_R_CH*n* 存储通道 *n* 输出信号的当前占空比。(RO)

Register 22.7. LEDC_TIMERx_CONF_REG (x: 0-3) (0x00A0+0x8*x)



LEDC_TIMERx_DUTY_RES 用于控制定时器 x 计数器的计数范围。(R/W)

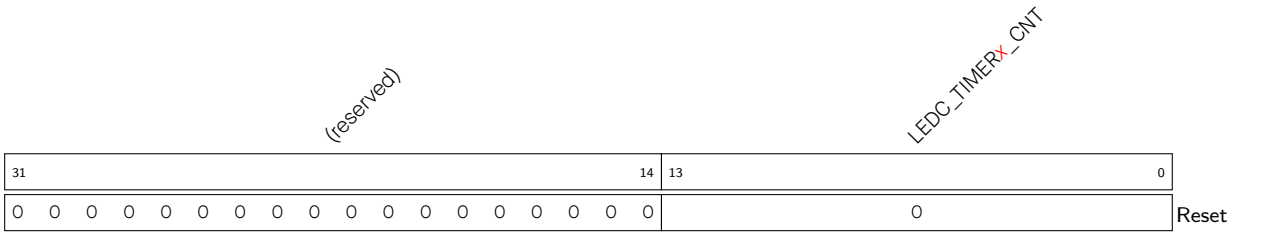
LEDC_CLK_DIV_TIMERx 用于配置定时器 x 分频器的分频系数。低 8 位为小数部分。(R/W)

LEDC_TIMERx_PAUSE 用于暂停定时器 x 的计数器。(R/W)

LEDC_TIMERx_RST 用于复位定时器 x。复位后计数器为 0。(R/W)

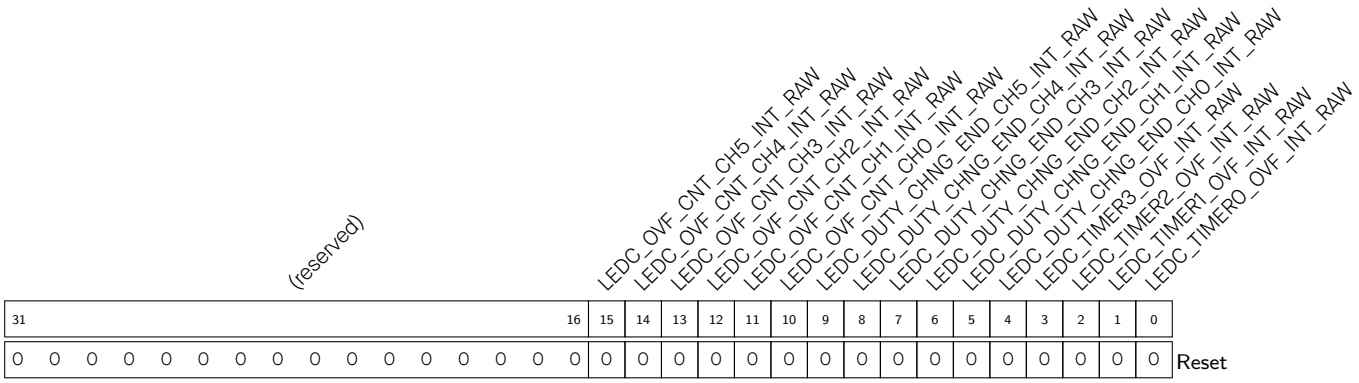
LEDC_TIMERx_PARA_UP 置位此位，更新 LEDC_CLK_DIV_TIMERx 和 LEDC_TIMERx_DUTY_RES。(WT)

Register 22.8. LEDC_TIMERx_VALUE_REG (x: 0-3) (0x00A4+0x8*x)



LEDC_TIMERx_CNT 存储定时器 x 的当前计数器值。(RO)

Register 22.9. LEDC_INT_RAW_REG (0x00C0)

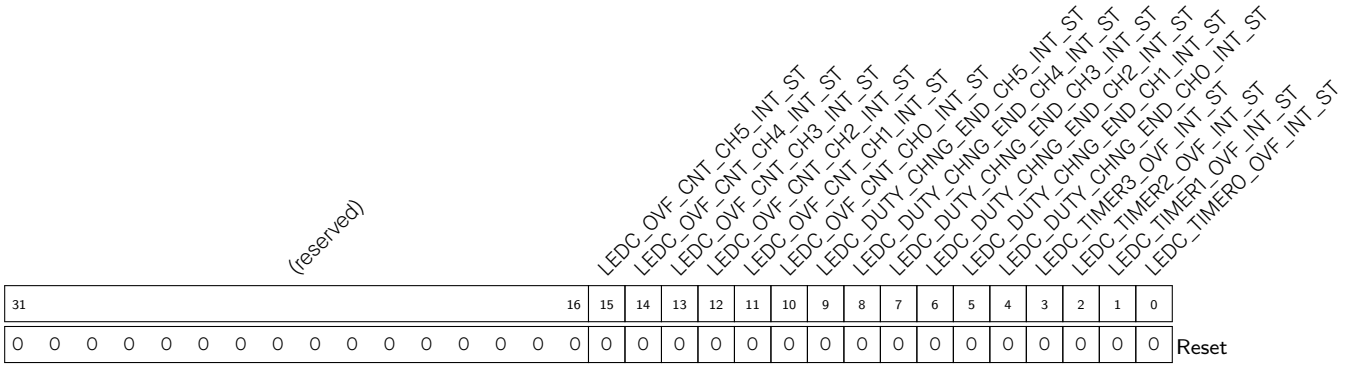


LEDC_TIMER*x*_OVF_INT_RAW LEDC_TIMER*x*_OVF_INT 的原始中断状态。(R/WTC/SS)

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 22.10. LEDC_INT_ST_REG (0x00C4)

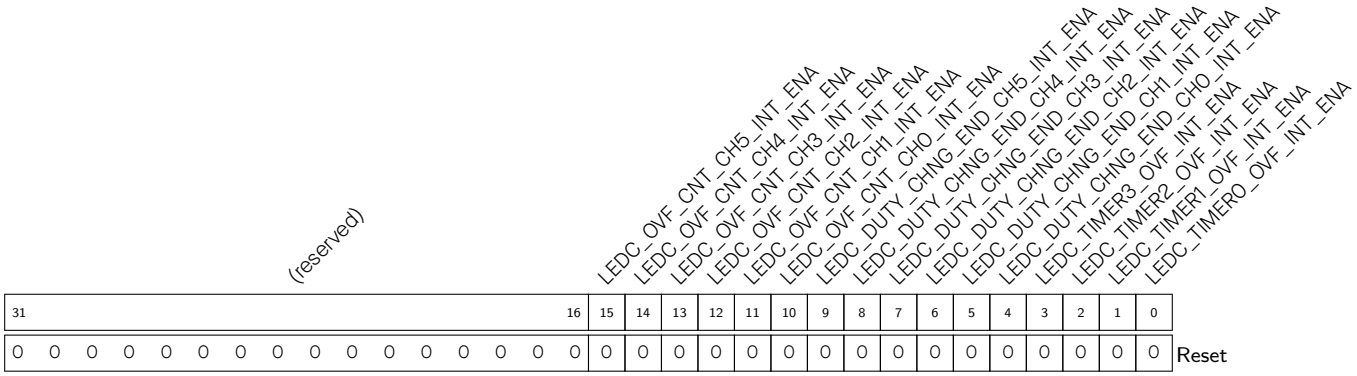


LEDC_TIMER*x*_OVF_INT_ST LEDC_TIMER*x*_OVF_INT_ENA 置 1 时，LEDC_TIMER*x*_OVF_INT 中断的屏蔽中断状态位。(RO)

LEDC_DUTY_CHNG_END_CH*n*_INT_ST LEDC_DUTY_CHNG_END_CH*n*_INT_ENA 置 1 时，LEDC_DUTY_CHNG_END_CH*n*_INT 中断的屏蔽中断状态位。(RO)

LEDC_OVF_CNT_CH*n*_INT_ST LEDC_OVF_CNT_CH*n*_INT_ENA 置 1 时，LEDC_OVF_CNT_CH*n*_INT 中断的屏蔽中断状态位。(RO)

Register 22.11. LEDC_INT_ENA_REG (0x00C8)

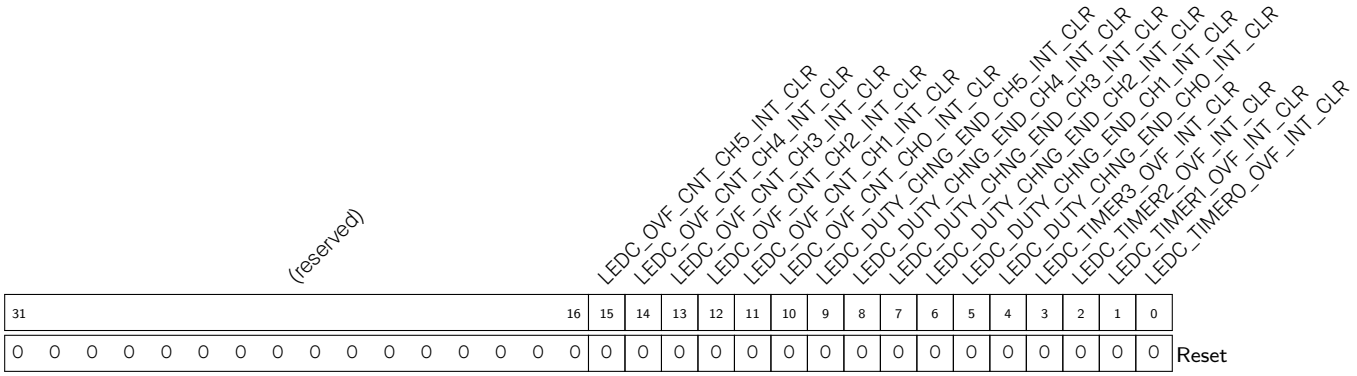


LEDC_TIMER*x*_OVF_INT_ENA LEDC_TIMER*x*_OVF_INT 中断的使能位。(R/W)

LEDC_DUTY_CHNG_END_CH*n*_INT_ENA LEDC_DUTY_CHNG_END_CH*n*_INT 中断的使能位。(R/W)

LEDC_OVF_CNT_CH*n*_INT_ENA LEDC_OVF_CNT_CH*n*_INT 中断的使能位。(R/W)

Register 22.12. LEDC_INT_CLR_REG (0x00CC)

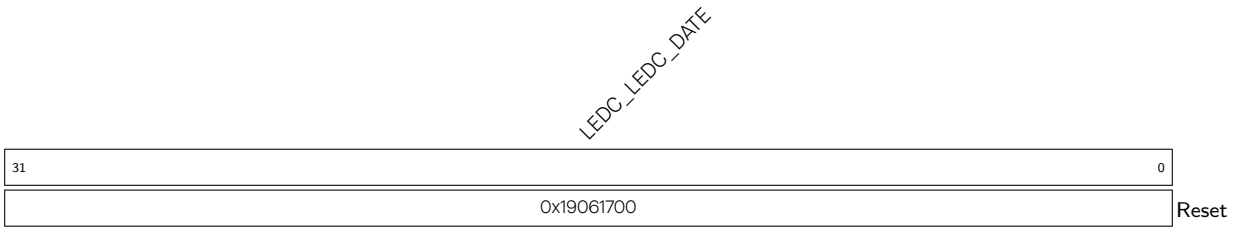


LEDC_TIMER*x*_OVF_INT_CLR 置位此位，清除 LEDC_TIMER*x*_OVF_INT 中断。(WT)

LEDC_DUTY_CHNG_END_CH*n*_INT_CLR 置位此位，清除 LEDC_DUTY_CHNG_END_CH*n*_INT 中断。(WT)

LEDC_OVF_CNT_CH*n*_INT_CLR 置位此位，清除 LEDC_OVF_CNT_CH*n*_INT 中断。(WT)

Register 22.13. LEDC_DATE_REG (0x00FC)



LEDC_LEDC_DATE 版本控制寄存器。(R/W)

23 片上传感器与模拟信号处理

23.1 概述

ESP8684 搭载了以下模拟信号处理设备和片上传感器：

- 一个 12 位逐次逼近型模拟数字转换器 (SAR ADC)，支持五个通道的模拟信号检测；
- 一个温度传感器：用于测量 ESP8684 芯片内部温度。

23.2 SAR ADC

23.2.1 概述

ESP8684 内置了一个 12 位的 SAR ADC，可测量最多来自五个管脚的模拟信号。SAR ADC 由 DIG ADC 控制器控制。DIG ADC 控制器可驱动 Digital_Reader 分别对 SAR ADC 的通道电压进行采样，支持多通道扫描和阈值监控。

23.2.2 特性

SAR ADC 具有如下特性：

- SAR ADC 有专用的 ADC Reader 模块 Digital_Reader 获取采样结果
- 支持 12 位采样分辨率
- 支持采集最多 5 个管脚上的模拟电压
- DIG ADC 控制器：
  - 配有单次采样和多通道扫描控制模块，分别支持单次采样模式和多通道扫描模式
  - 支持单次采样模式和多通道扫描模式同时工作
  - 在多通道扫描模式下，支持自定义扫描通道顺序
  - 提供两个滤波器，滤波系数可配
  - 支持阈值监控，采样值大于设置的高阈值或小于设置的低阈值将产生中断

23.2.3 功能描述

SAR ADC 的主要元件与连接情况见图 23-1。

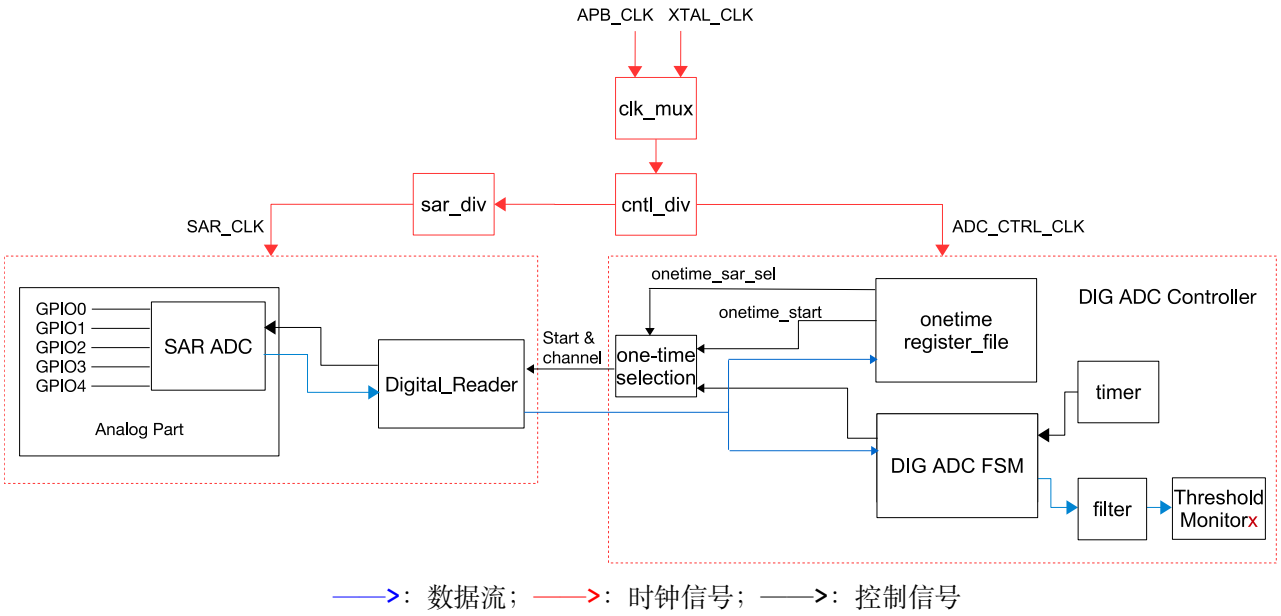


图 23-1. SAR ADC 的功能概况

如图 23-1 所示，SAR ADC 模块主要包括以下元件：

- SAR ADC：可对五个通道进行电压检测；
- 时钟管理：对时钟源进行选择和分频：
  - 时钟源：可选择 APB_CLK 或 XTAL_CLK；
  - 分频时钟：
    * SAR_CLK：SAR ADC 和 Digital_Reader 的工作时钟；其中控制 SAR_CLK 分频的 sar_div 的分频系数至少是 2 分频；
    * ADC_CTRL_CLK：DIG ADC FSM 的工作时钟。
- Digital_Reader：由 DIG ADC FSM 驱动，读取 SAR ADC 的数值；
- DIG ADC FSM：生成整个 ADC 采样过程中所需的各种信号，下文简称 FSM。
- 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。

以下小节将详细介绍各个元件。

23.2.3.1 输入信号

SAR 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 23-1 列出了所有可能需要经过 SAR ADC 处理的模拟信号。

表 23-1. SAR ADC 的信号输入

信号名称	通道编号
GPIO0	0
GPIO1	1
GPIO2	2
GPIO3	3
GPIO4	4

23.2.3.2 ADC 转换和衰减

SAR ADC 转换模拟信号时，转换分辨率（12 位）电压范围为 0 mV ~ V_{ref}。其中，V_{ref} 为 SAR ADC 内部参考电压。因此，转换结果 (data) 可以使用以下公式转换成模拟电压输出 V_{data}：

$$V_{data} = \frac{V_{ref}}{4095} \times data$$

如需转换大于 V_{ref} 的电压，信号输入 SAR ADC 前可进行衰减。衰减可配置为 0 dB、2.5 dB、6 dB 和 10 dB。

23.2.3.3 DIG ADC 控制器

DIG ADC 控制器使用快速时钟，实现了采样速率大幅提升。该控制器最高支持 12 位采样分辨率，同时支持软件触发的单次采样和专用定时器触发的多通道扫描。更多参数和性能信息见《ESP8684 系列芯片技术规格书》中的 ADC 特性章节。

软件驱动的单次采样的具体配置如下：

- 置位 APB_SARADC1_ONETIME_SAMPLE 选择对 SAR ADC 进行单次采样；
- 配置 APB_SARADC_ONETIME_CHANNEL 选择采样通道；
- 配置 APB_SARADC_ONETIME_ATTEN 选择衰减；
- 配置 APB_SARADC_ONETIME_START 启动单次采样；
- 采样结束即触发 APB_SARADC_ADC1_DONE_INT_RAW 中断。软件检测该中断后，可在 APB_SARADC_ADC1_DATA 寄存器内读到采样值。

如果选择专用定时器驱动的多通道扫描，可采用如下配置。注，在多通道扫描模式下，扫描序列可根据样式表的描述进行，样式表可配置。

- 配置 APB_SARADC_TIMER_TARGET 设置 DIG ADC 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。定时器的工作时钟见章节 23.2.3.4；
- 配置 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；
- **每次采样完成均会有中断产生，需要软件从对应寄存器中获取，否则采样数据经过阈值监控器之后将被直接丢弃。**

23.2.3.4 DIG ADC 时钟

用户可配置 APB_SARADC_CLK_SEL 选择 DIG ADC 控制器的工作时钟：

- 1：选择 XTAL_CLK 的分频时钟 ADC_CTRL_CLK；
- 0：选择 APB_CLK。

如果选择使用 ADC_CTRL_CLK，用户可配置 APB_SARADC_CLKM_DIV_NUM 选择分频系数。

注意，由于 SAR ADC 有速度限制，所以 Digital_Reader 和 SAR ADC 的工作时钟是 SAR_CLK。SAR_CLK 频率会影响采样精度，频率越低采样精度越高。SAR_CLK 由 ADC_CTRL_CLK 经过专用分频器分频所得。分频系数通过 APB_SARADC_SAR_CLK_DIV 配置。

ADC 每采样一个数据需要 25 个 SAR_CLK 时钟周期数，所以最大采样速率受到 SAR_CLK 的频率限制。更多时钟信息，见章节 6 复位和时钟。

23.2.3.5 DIG ADC FSM

概述

图 23-2 展示了 DIG ADC FSM 的工作原理。

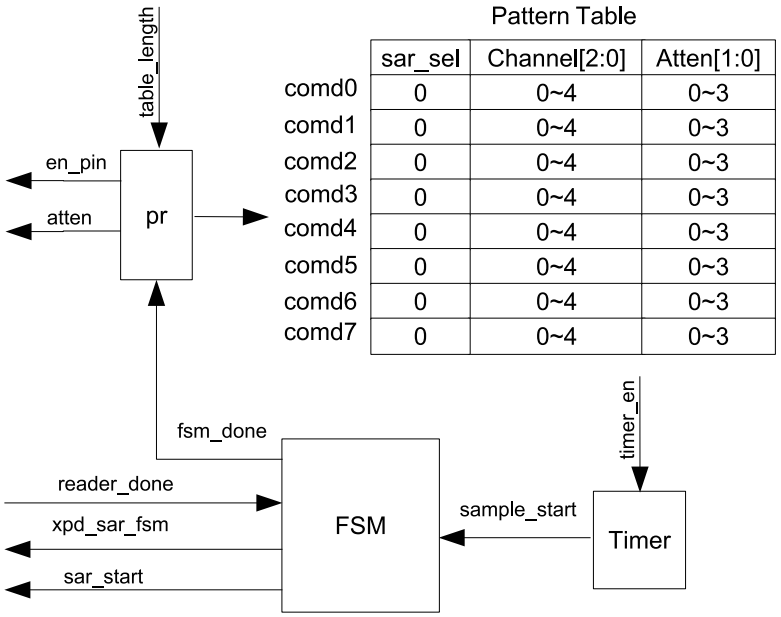


图 23-2. DIG ADC FSM 概况

其中，

- Timer：表示 DIG ADC 的专用定时器，可产生 sample_start 信号；
- 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。

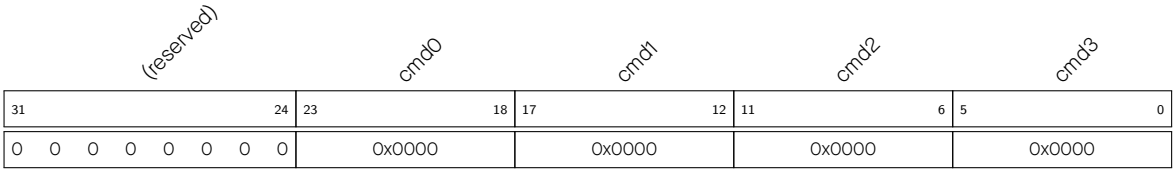
相关执行过程如下：

- 置位 APB_SARADC_TIMER_EN 使能 DIG ADC 的专用定时器。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，选择 SAR ADC 用作工作 ADC，同时配置 ADC 通道以及衰减；
  - 根据配置信息，输出相应的 en_pad（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_Reade 返回的 reader_done 信号后，
  - 结束采样；
  - 数据传输给滤波器 (filter) 和阈值监控器 (threshold monitor) 后丢失（见图 23-1）；
  - 更新样式表指针 pr，等待下一次采样。注意，如果指针 pr 小于 APB_SARADC_SAR_PATT_LEN (table_length)，则 pr = pr + 1；否则 pr 将被清零。

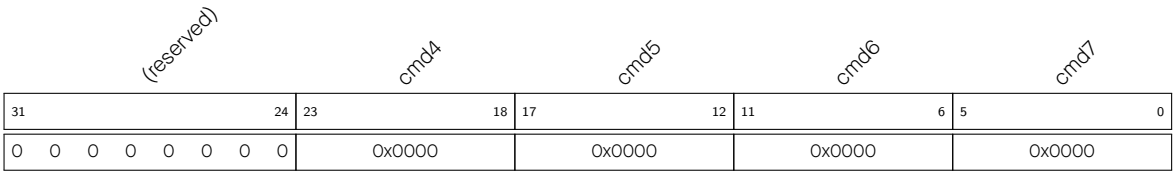
样式表结构

DIG ADC 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，如图 23-3 和图 23-4 所示：



cmd *x* 表示样式表中的样式，即样式 0 ~ 样式 3。

图 23-3. APB_SARADC_SAR_PATT_TAB1_REG 与样式 0 - 3



cmd *x* 表示样式表中的样式，即样式 4 ~ 样式 7。

图 23-4. APB_SARADC_SAR_PATT_TAB2_REG 与样式 4 - 7

每个寄存器包含四个样式，每个样式长度为六位，共包括三个字段，分别存储了选择的工作 ADC、通道和衰减信息，具体见表 23-5。

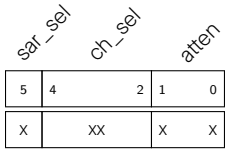


图 23-5. 样式表中的样式结构

atten 衰减配置信息。0：0 dB；1：2.5 dB；2：6 dB；3：10 dB。

ch_sel 扫描通道选择信息，更多信息见表 23-1。

sar_sel ADC 选择信息。ESP8684 只有一个 ADC，因此该位只能配置为 0。

多通道扫描配置示例

例如，希望实现如下所示的多通道扫描方式：

- 扫描通道 2，且衰减配置为 10 dB；
- 扫描通道 0，且衰减配置为 2.5 dB。

则具体的配置如下：

- 配置第一个样式 cmd0，如下图所示：

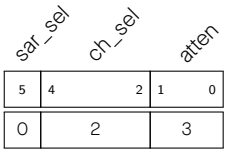


图 23-6. cmd0 配置示例

atten 配置该字段的值为 3，即衰减配置为 10 dB。

ch_sel 配置该字段的值为 2，即选择通道 2（见表 23-1）。

sar_sel 配置该位为 0。

- 配置第二个样式 cmd1，如下图所示：

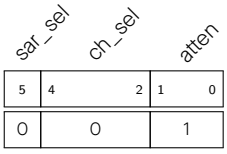


图 23-7. cmd1 配置示例

atten 配置该字段的值为 1，即衰减配置为 2.5 dB。

ch_sel 配置该字段的值为 0，即选择通道 0（见表 23-1）。

sar_sel 配置该位为 0。

- 配置 APB_SARADC_SAR_PATT_LEN 为 1，即选择使用上述配置好的样式表 0 和样式表 1；
- 使能定时器，则 DIG ADC 控制器将根据上述样式配置，周期性采样通道 2 和通道 0。

23.2.3.6 ADC 滤波器

DIG ADC 控制器支持滤波功能，提供两个滤波器。两个滤波器均可配置 SAR ADC 的任一通道，然后对目标通道的采样数据进行滤波。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 采样值
- $data_{prev}$：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* 作用的 ADC 通道
- 配置 APB_SARADC_FILTER_FACTOR*x* 设置滤波器 *x* 的滤波系数

注意，这里的 *x* 为滤波器编号：*x* 为 0 表示滤波器 *0*；为 1 表示滤波器 *1*。

23.2.3.7 阈值监控

DIG ADC 控制器包含两个阈值监控器，可配置到 SAR ADC 的任意通道上。当 ADC 采样值大于设定的高阈值，则触发高阈值中断；若采样值小于设定的低阈值，则触发低阈值中断。

阈值监控配置如下：

- 配置 APB_SARADC_THRES*x*_EN 使能阈值监控 *x* 的功能；
- 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。

注意，这里的 *x* 为阈值监控器编号：*x* 为 0 表示阈值监控器 *0*；为 1 表示阈值监控器 *1*。

23.3 温度传感器

23.3.1 概述

ESP8684 搭载了温度传感器可以实时监测芯片内部温度。

23.3.2 特性

温度传感器的主要特性包括：

- 支持软件触发，且一旦触发后，可持续读取数据
- 可根据使用环境配置温度偏移，提高测试精度
- 测量范围可调节

23.3.3 功能描述

温度传感器可由软件启动，具体配置如下：

- 置位 APB_SARADC_TSENS_PU，温度传感器上电；
- 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；
- 首次开启温度传感器，需要等待一定的启动时间（大致为 100 μs），之后可以从 APB_SARADC_TSENS_OUT 中即可持续获取温度值。

温度传感器的输出值需要使用转换公式转换成实际的温度值 (°C)。转换公式如下：

$$T(°C) = 0.4386 * VALUE - 27.88 * offset - 20.52$$

其中 VALUE 即温度传感器的输出值，offset 由温度偏移决定。温度传感器在不同的实际使用环境（测量温度范围）下，温度偏移不同，见表 23-2。

表 23-2. 温度传感器的温度偏移

测量范围 (°C)	温度偏移 (°C)
50 ~ 125	-2
20 ~ 100	-1
-10 ~ 80	0
-30 ~ 50	1
-40 ~ 20	2

23.4 中断

- APB_SARADC_ADC1_DONE_INT：SAR ADC 完成一次转换，即触发此中断；
- APB_SARADC_THRES*x*_HIGH_INT：超过阈值监控器 *x* 的高阈值，即触发此中断；
- APB_SARADC_THRES*x*_LOW_INT：低于阈值监控器 *x* 的低阈值，即触发此中断。

23.5 寄存器列表

本小节的所有地址均为相对于 ADC 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 3 *系统和存储器* 中的表 3-3。

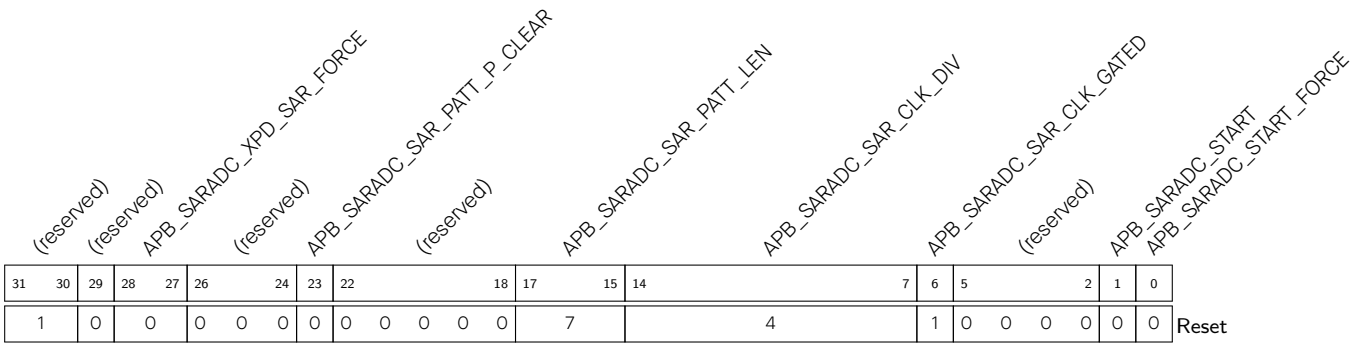
请查看章节 *寄存器的访问类型*，了解“访问”列缩写的含义。

名称	描述	地址	访问
配置寄存器			
APB_SARADC_CTRL_REG	SAR ADC FSM 的配置寄存器	0x0000	R/W
APB_SARADC_CTRL2_REG	SAR ADC FSM 的采样配置寄存器	0x0004	R/W
APB_SARADC_FILTER_CTRL1_REG	滤波器配置寄存器 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	滤波器配置寄存器 0	0x0028	R/W
APB_SARADC_1_DATA_STATUS_REG	SAR ADC 采样数据寄存器	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	采样阈值使能寄存器	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_ADC_CLKM_CONF_REG	SAR ADC 时钟控制寄存器	0x0054	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
版本寄存器			
APB_SARADC_APB_CTRL_DATE_REG	版本控制寄存器	0x03FC	R/W

23.6 寄存器

本小节的所有地址均为相对于 ADC 控制器基地址的地址偏移量（相对地址），具体基地址请见章节 3 系统和存储器 中的表 3-3。

Register 23.1. APB_SARADC_CTRL_REG (0x0000)



APB_SARADC_START_FORCE 0：选择使用 FSM 启动 SAR ADC；1：选择使用软件启动 SAR ADC。(R/W)

APB_SARADC_START 写 1 选择使用软件启动 SAR ADC。仅当 APB_SARADC_START_FORCE = 1 时有效。(R/W)

APB_SARADC_SAR_CLK_GATED 0：SAR ADC 时钟一直处于打开状态；1：SAR ADC 处于空闲状态时，SAR ADC 时钟关闭。(R/W)

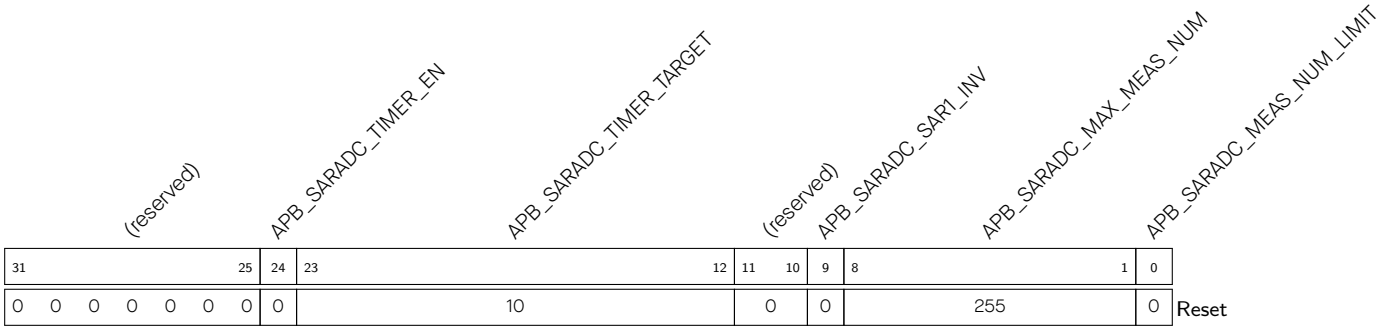
APB_SARADC_SAR_CLK_DIV SAR ADC 的时钟分频系数。该系数不可小于 2。(R/W)

APB_SARADC_SAR_PATT_LEN 配置 SAR ADC 需要使用的样式数量。如果此字段设置为 1，则将使用样式表中的样式 0 (cmd0) 和样式 1 (cmd1)。(R/W)

APB_SARADC_SAR_PATT_P_CLEAR 清除 DIG ADC 控制器的样式表指针。(R/W)

APB_SARADC_XPD_SAR_FORCE 强制选择 XPD SAR。(R/W)

Register 23.2. APB_SARADC_CTRL2_REG (0x0004)



APB_SARADC_MEAS_NUM_LIMIT 使能 SAR ADC 最大转换次数限制。仅当使用定时器控制 SAR ADC 时有效。(R/W)

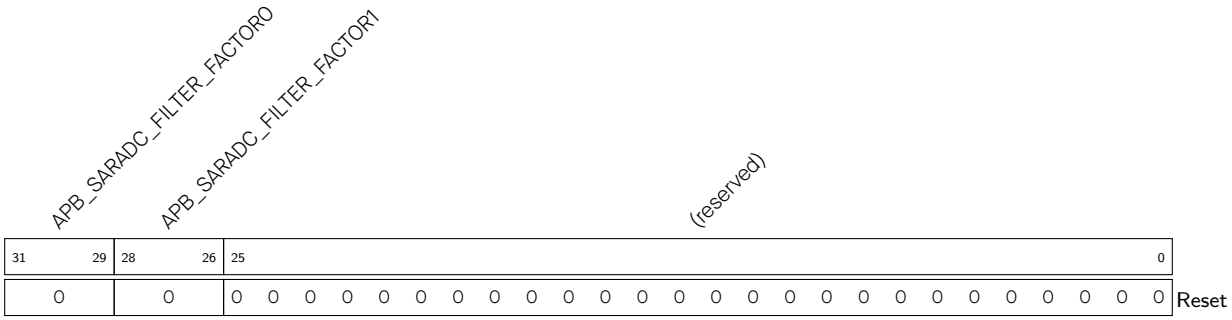
APB_SARADC_MAX_MEAS_NUM 设置 SAR ADC 最大转换次数。(R/W)

APB_SARADC_SAR1_INV 写 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 使能 SAR ADC 定时器触发。(R/W)

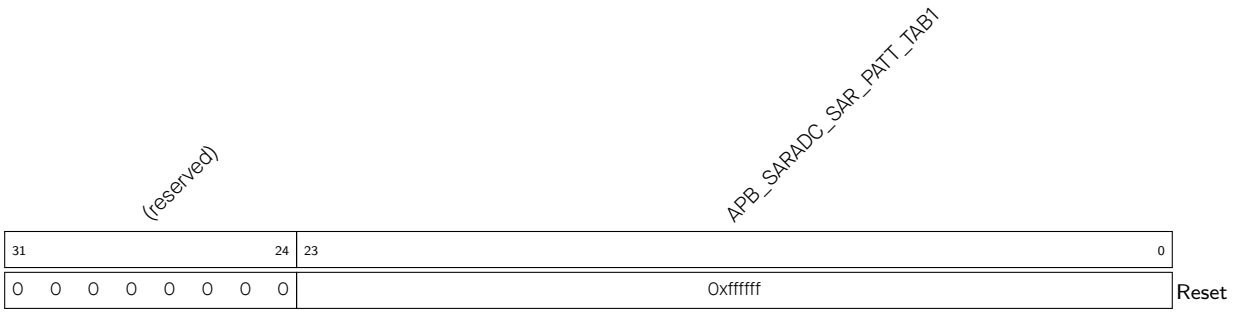
Register 23.3. APB_SARADC_FILTER_CTRL1_REG (0x0008)



APB_SARADC_FILTER_FACTOR1 配置 SAR ADC 滤波器 1 的滤波系数。(R/W)

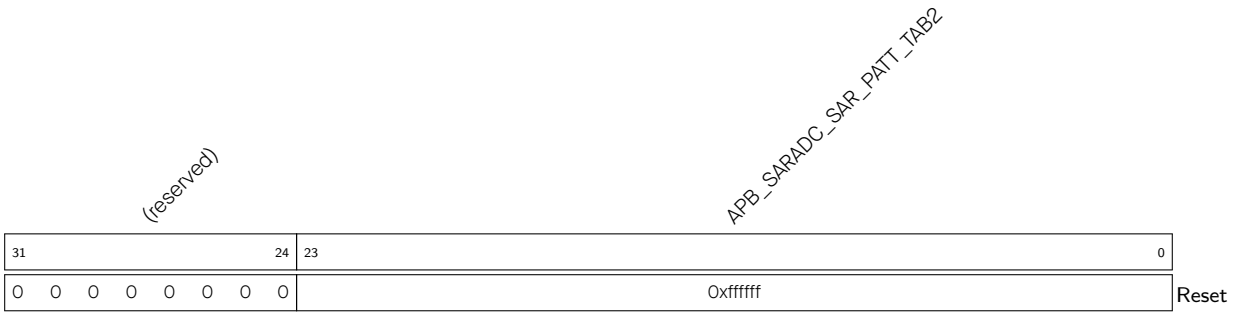
APB_SARADC_FILTER_FACTOR0 配置 SAR ADC 滤波器 0 的滤波系数。(R/W)

Register 23.4. APB_SARADC_SAR_PATT_TAB1_REG (0x0018)



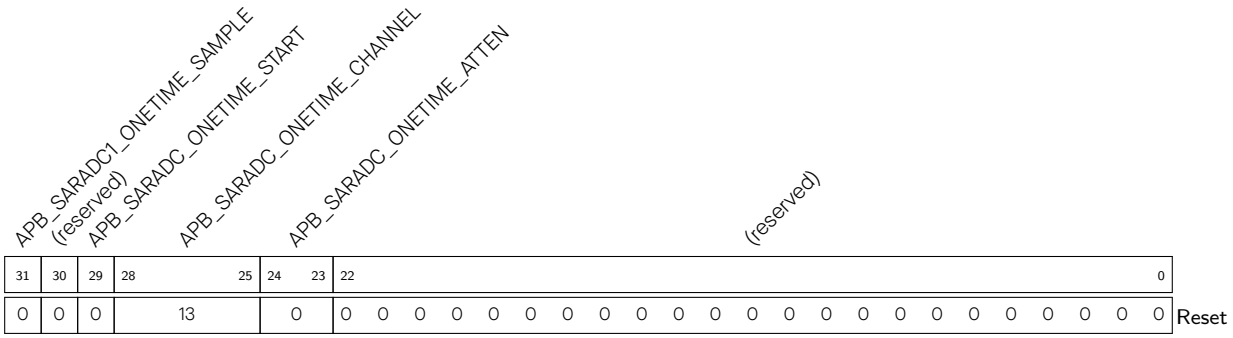
APB_SARADC_SAR_PATT_TAB1 样式表的样式 0 ~ 3。每个样式占 6 位。(R/W)

Register 23.5. APB_SARADC_SAR_PATT_TAB2_REG (0x001C)



APB_SARADC_SAR_PATT_TAB2 样式表的样式 4 ~ 7。每个样式占 6 位。(R/W)

Register 23.6. APB_SARADC_ONETIME_SAMPLE_REG (0x0020)



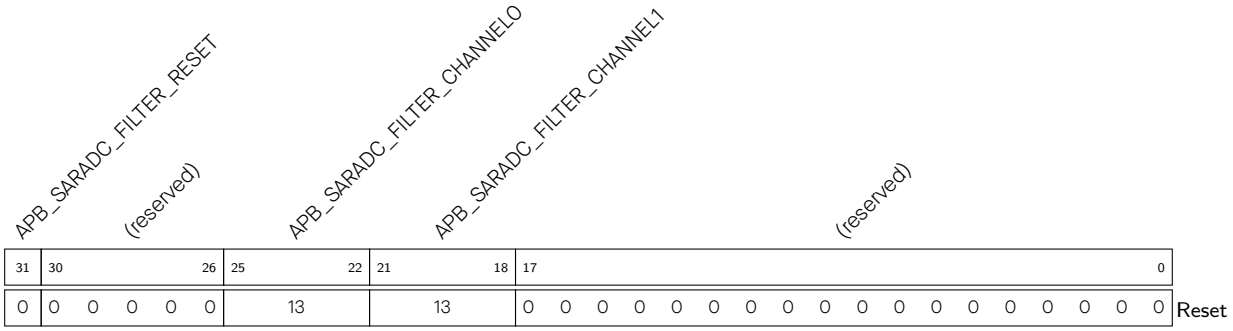
APB_SARADC_ONETIME_ATTEN 配置单次采样的衰减。(R/W)

APB_SARADC_ONETIME_CHANNEL 配置单次采样的通道。(R/W)

APB_SARADC_ONETIME_START 启动 SAR ADC 单次采样。(R/W)

APB_SARADC1_ONETIME_SAMPLE 使能 SAR ADC 单次采样。(R/W)

Register 23.7. APB_SARADC_FILTER_CTRL0_REG (0x0028)

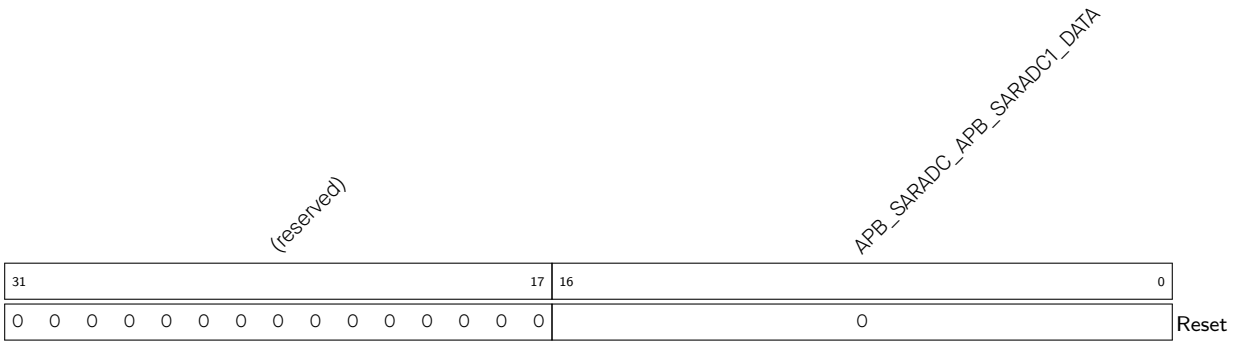


APB_SARADC_FILTER_CHANNEL1 配置 SAR ADC 滤波通道 1。(R/W)

APB_SARADC_FILTER_CHANNEL0 配置 SAR ADC 滤波通道 0。(R/W)

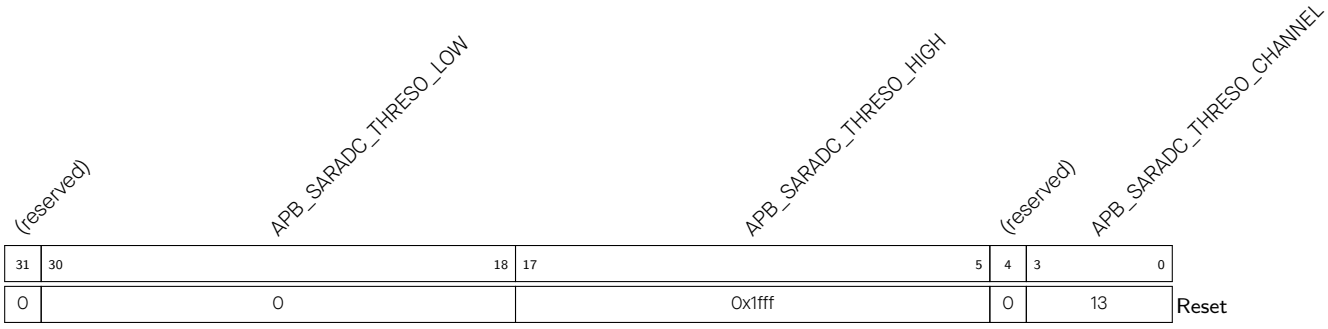
APB_SARADC_FILTER_RESET 复位 SAR ADC 滤波器。(R/W)

Register 23.8. APB_SARADC_1_DATA_STATUS_REG (0x002C)



APB_SARADC_ADC1_DATA SAR ADC 的转换数据。(RO)

Register 23.9. APB_SARADC_THRES0_CTRL_REG (0x0034)

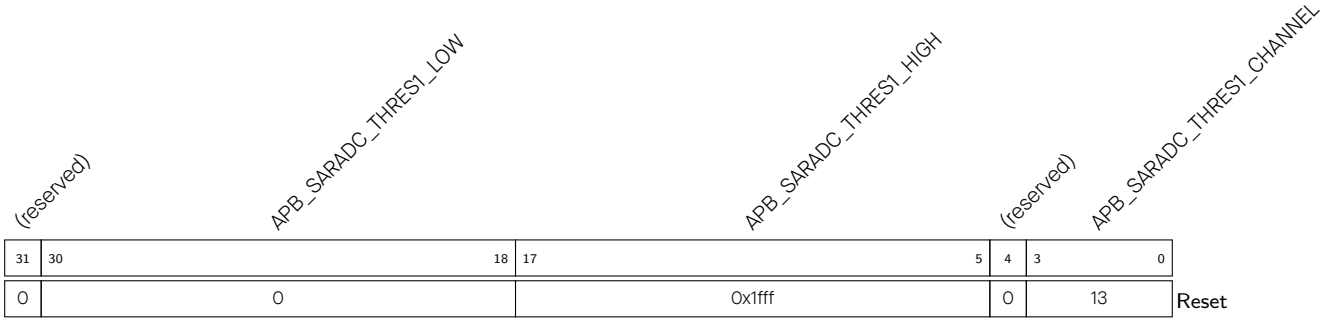


APB_SARADC_THRES0_CHANNEL 配置 SAR ADC 阈值监控器 0 需要监控的通道。(R/W)

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 23.10. APB_SARADC_THRES1_CTRL_REG (0x0038)

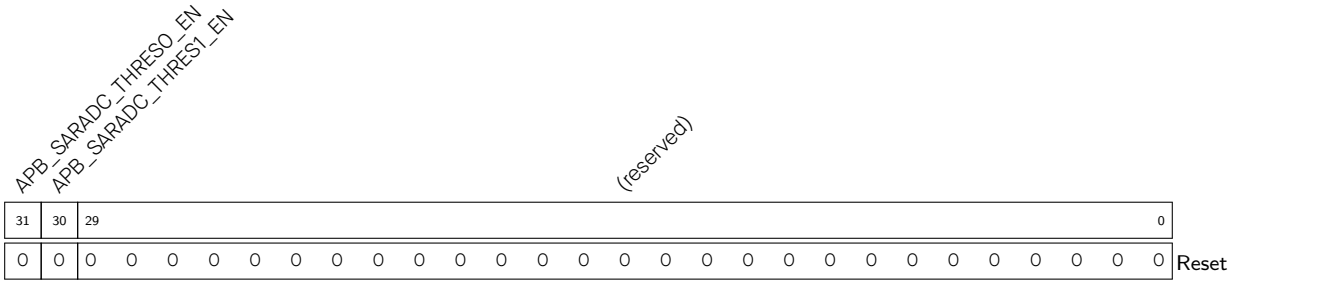


APB_SARADC_THRES1_CHANNEL 配置 SAR ADC 阈值监控器 1 需要监控的通道。(R/W)

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)

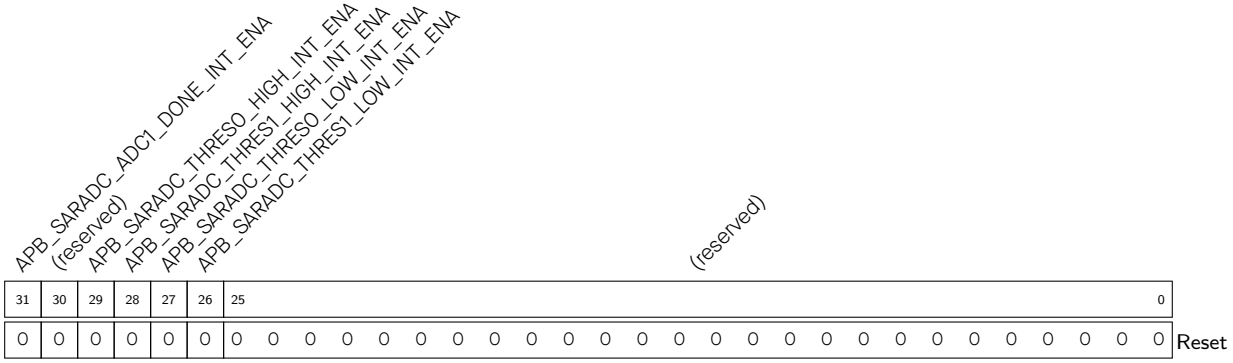
Register 23.11. APB_SARADC_THRES_CTRL_REG (0x003C)



APB_SARADC_THRES1_EN 使能阈值监控器 1。(R/W)

APB_SARADC_THRES0_EN 使能阈值监控器 0。(R/W)

Register 23.12. APB_SARADC_INT_ENA_REG (0x0040)



APB_SARADC_THRES1_LOW_INT_ENA APB_SARADC_THRES1_LOW_INT 的中断使能位。(R/W)

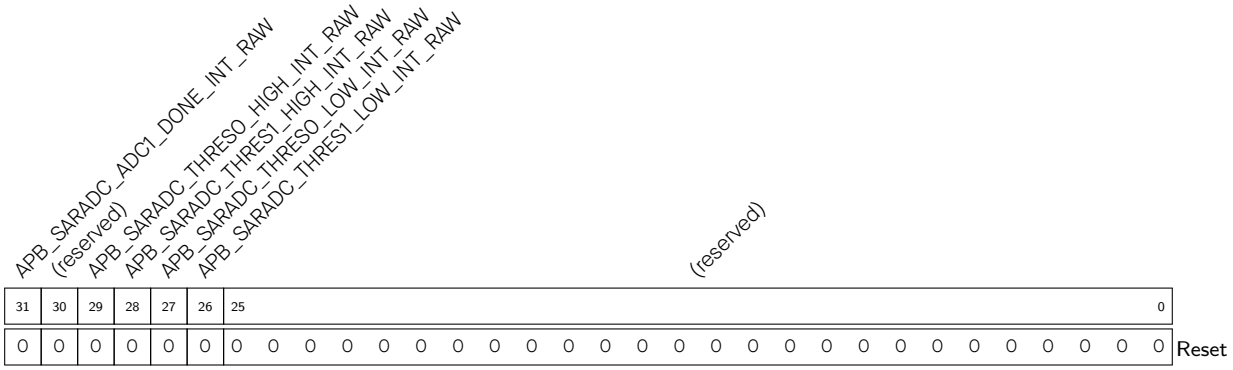
APB_SARADC_THRES0_LOW_INT_ENA APB_SARADC_THRES0_LOW_INT 的中断使能位。(R/W)

APB_SARADC_THRES1_HIGH_INT_ENA APB_SARADC_THRES1_HIGH_INT 的中断使能位。(R/W)

APB_SARADC_THRES0_HIGH_INT_ENA APB_SARADC_THRES0_HIGH_INT 的中断使能位。(R/W)

APB_SARADC_ADC1_DONE_INT_ENA APB_SARADC_ADC1_DONE_INT 的中断使能位。(R/W)

Register 23.13. APB_SARADC_INT_RAW_REG (0x0044)



APB_SARADC_THRES1_LOW_INT_RAW APB_SARADC_THRES1_LOW_INT 的原始中断位。(RO)

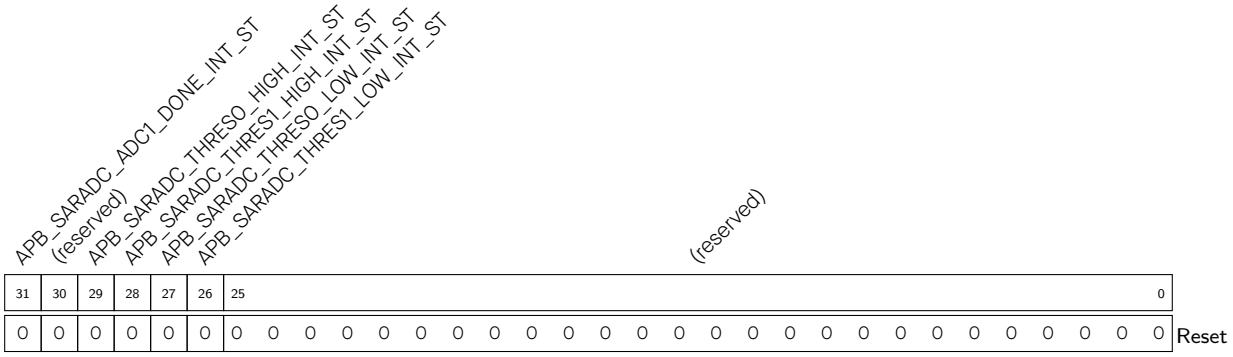
APB_SARADC_THRES0_LOW_INT_RAW APB_SARADC_THRES0_LOW_INT 的原始中断位。(RO)

APB_SARADC_THRES1_HIGH_INT_RAW APB_SARADC_THRES1_HIGH_INT 的原始中断位。(RO)

APB_SARADC_THRES0_HIGH_INT_RAW APB_SARADC_THRES0_HIGH_INT 的原始中断位。(RO)

APB_SARADC_ADC1_DONE_INT_RAW APB_SARADC_ADC1_DONE_INT 的原始中断位。(RO)

Register 23.14. APB_SARADC_INT_ST_REG (0x0048)



APB_SARADC_THRES1_LOW_INT_ST APB_SARADC_THRES1_LOW_INT 的中断状态位。

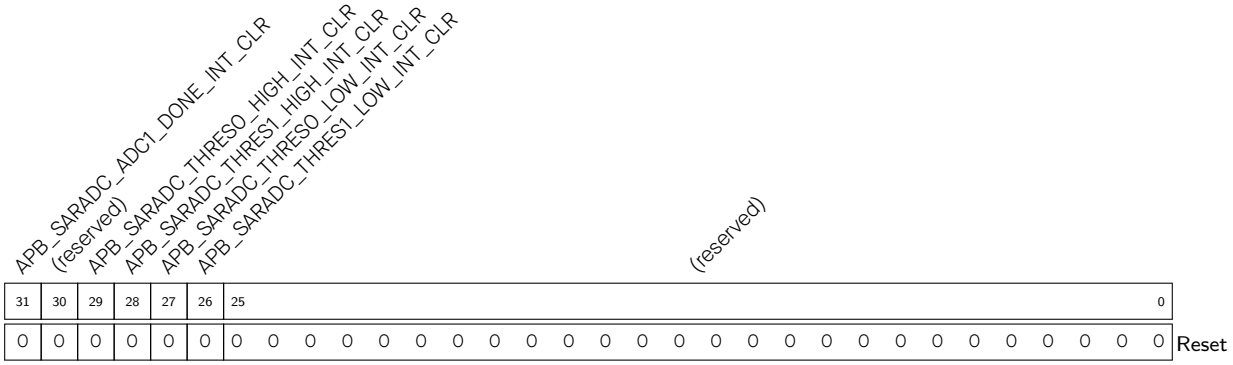
APB_SARADC_THRES0_LOW_INT_ST APB_SARADC_THRES0_LOW_INT 的中断状态位。

APB_SARADC_THRES1_HIGH_INT_ST APB_SARADC_THRES1_HIGH_INT 的中断状态位。

APB_SARADC_THRES0_HIGH_INT_ST APB_SARADC_THRES0_HIGH_INT 的中断状态位。

APB_SARADC_ADC1_DONE_INT_ST APB_SARADC_ADC1_DONE_INT 的中断状态位。

Register 23.15. APB_SARADC_INT_CLR_REG (0x004C)



APB_SARADC_THRES1_LOW_INT_CLR APB_SARADC_THRES1_LOW_INT 的中断清除位。(WO)

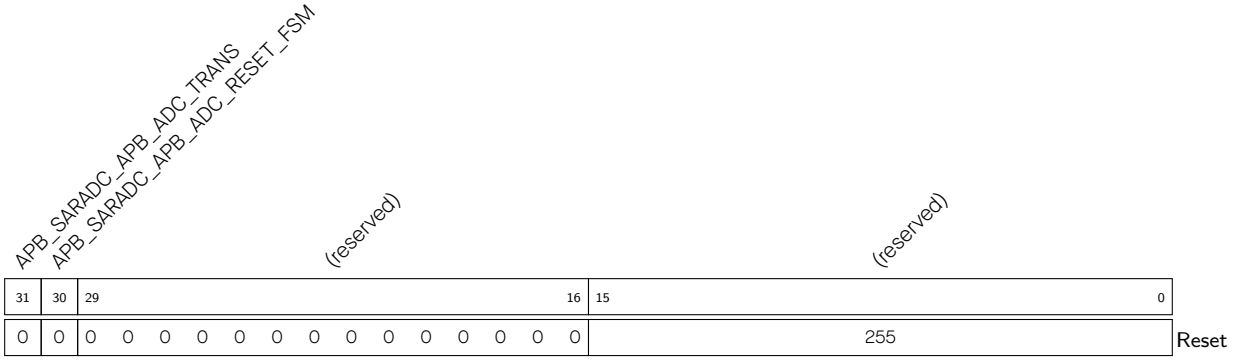
APB_SARADC_THRES0_LOW_INT_CLR APB_SARADC_THRES0_LOW_INT 的中断清除位。(WO)

APB_SARADC_THRES1_HIGH_INT_CLR APB_SARADC_THRES1_HIGH_INT 的中断清除位。(WO)

APB_SARADC_THRES0_HIGH_INT_CLR APB_SARADC_THRES0_HIGH_INT 的中断清除位。(WO)

APB_SARADC_ADC1_DONE_INT_CLR APB_SARADC_ADC1_DONE_INT 的中断清除位。(WO)

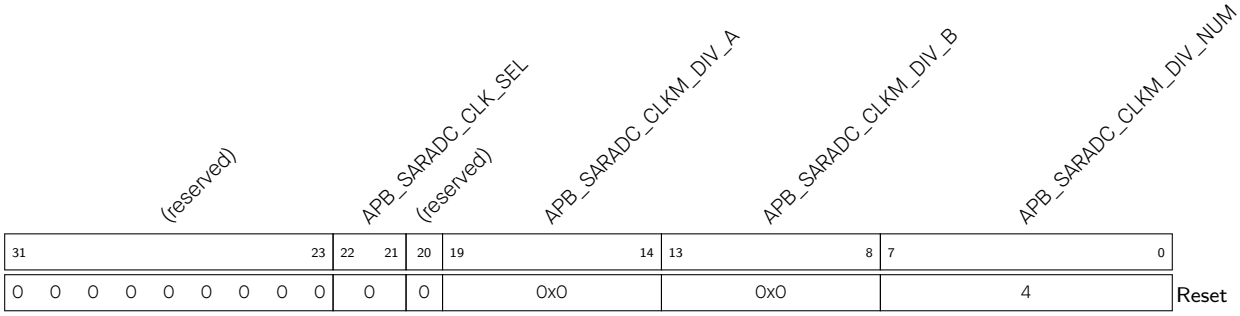
Register 23.16. APB_SARADC_DMA_CONF_REG (0x0050)



APB_SARADC_APB_ADC_RESET_FSM 复位 DIG ADC 控制器状态。(R/W)

APB_SARADC_APB_ADC_TRANS ESP8684 未给 SAR ADC 配备 DMA，因此该位必须配置为 0，经过滤波器和阈值监控模块之后，数据丢失。(R/W)

Register 23.17. APB_SARADC_APB_ADC_CLKM_CONF_REG (0x0054)



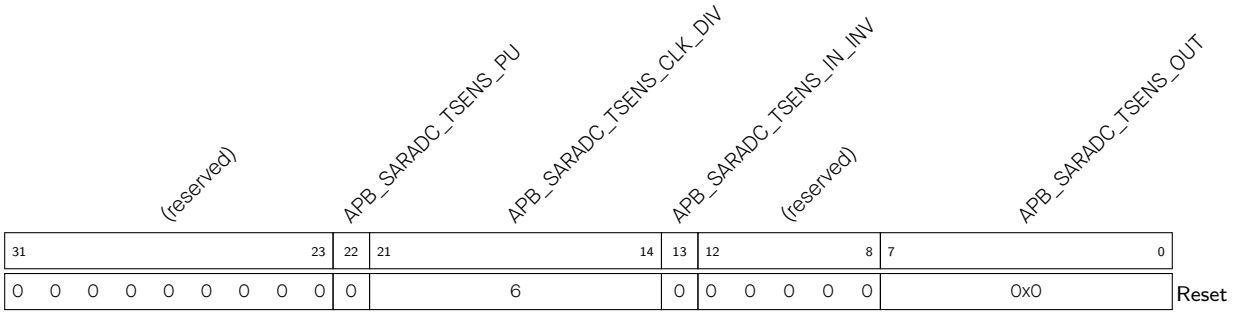
APB_SARADC_CLKM_DIV_NUM ADC 时钟的整数分频系数。分频系数 = APB_SARADC_CLKM_DIV_NUM + APB_SARADC_CLKM_DIV_B/ APB_SARADC_CLKM_DIV_A。(R/W)

APB_SARADC_CLKM_DIV_B 小数分频系数中的分子。(R/W)

APB_SARADC_CLKM_DIV_A 小数分频系数中的分母。(R/W)

APB_SARADC_CLK_SEL 选择时钟源。0：APB_CLK；1：XTAL_CLK 的分频时钟。(R/W)

Register 23.18. APB_SARADC_APB_TSENS_CTRL_REG (0x0058)



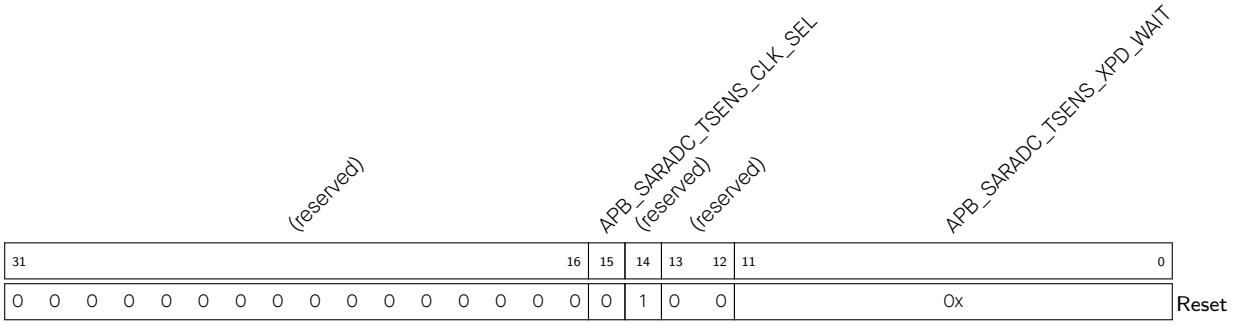
APB_SARADC_TSENS_OUT 温度传感器的输出值。(RO)

APB_SARADC_TSENS_IN_INV 反转温度传感器的输入值。(R/W)

APB_SARADC_TSENS_CLK_DIV 温度传感器的时钟分频系数。(R/W)

APB_SARADC_TSENS_PU 温度传感器上电。(R/W)

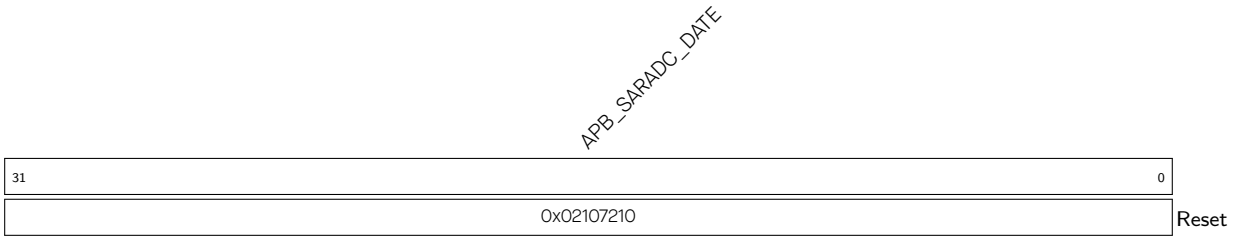
Register 23.19. APB_SARADC_APB_TSENS_CTRL2_REG (0x005C)



APB_SARADC_TSENS_XPD_WAIT 温度传感器的复位释放前，需要等待的时间。(R/W)

APB_SARADC_TSENS_CLK_SEL 选择温度传感器的工作时钟。0：RC_FAST_CLK。1：XTAL_CLK。(R/W)

Register 23.20. APB_SARADC_APB_CTRL_DATE_REG (0x03FC)



APB_SARADC_DATE 版本控制寄存器 (R/W)

相关文档和资源

相关文档

- 《ESP8684 技术规格书》 – 提供 ESP8684 芯片的硬件技术规格。
- 《ESP8684 硬件设计指南》 – 提供基于 ESP8684 芯片的产品设计规范。
- 证书
  https://espressif.com/zh-hans/support/documents/certificates
- 文档更新和订阅通知
  https://espressif.com/zh-hans/support/download/documents

开发者社区

- 《ESP8684 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

- ESP8684 系列芯片 – ESP8684 全系列芯片。
  https://espressif.com/zh-hans/products/socs?id=ESP8684
- ESP8684 系列模组 – ESP8684 全系列模组。
  https://espressif.com/zh-hans/products/modules?id=ESP8684
- ESP8684 系列开发板 – ESP8684 全系列开发板。
  https://espressif.com/zh-hans/products/devkits?id=ESP8684
- 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
BOOTCTRL	芯片 Boot 控制
DS	数字签名
DMA	DMA 控制器
eFuse	eFuse 控制器
HMAC	HMAC 加速器
I2C	I2C 控制器
I2S	I2S 控制器
LEDC	LED 控制 PWM
MCPWM	电机控制 PWM
PCNT	脉冲计数器控制器
RNG	随机数生成器
RSA	RSA 加速器
SDHOST	SD/MMC 主机控制器
SHA	SHA 加速器
SPI	SPI 控制器
SYSTIMER	系统定时器
TIMG	定时器组
TWAI	双线汽车接口
UART	UART 控制器
ULP 协处理器	超低功耗协处理器
USB OTG	USB On-The-Go
WDT	看门狗定时器

寄存器相关缩写

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
- 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

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。
WOS	**软件写 0 置位**。如果用户软件将 0 写入此寄存器/字段，硬件会自动置位此寄存器/字段。
WC	**软件写清零**。如果用户软件写入此寄存器/字段，硬件会自动清零此寄存器/字段。
W1C	**软件写 1 清零**。如果用户软件将 1 写入此寄存器/字段，硬件会自动清零此寄存器/字段。
WOC	**软件写 0 清零**。如果用户软件将 0 写入此寄存器/字段，硬件会自动清零此寄存器/字段。
WT	**软件写产生边沿触发信号**。如果用户软件将 1 写入此字段，将会产生边沿触发信号（APB 总线中的脉冲）或清除相应的 WTC 字段（详见 WTC）。
WTC	**软件写其他寄存器位清零本寄存器位**。如果用户软件将 1 写入相应的 WT 字段，硬件会自动清除此字段（详见 WT）。
W1T	**软件写 1 取反**。如果用户软件将 1 写入此字段，硬件会自动取反相应字段，否则不会取反。

WOT **软件写 0 取反**。如果用户软件将 0 写入此字段，硬件会自动取反相应字段，否则不会取反。

WL **软件仅在锁禁用时写**。如果锁被禁用，用户软件可以写入此寄存器/字段。

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

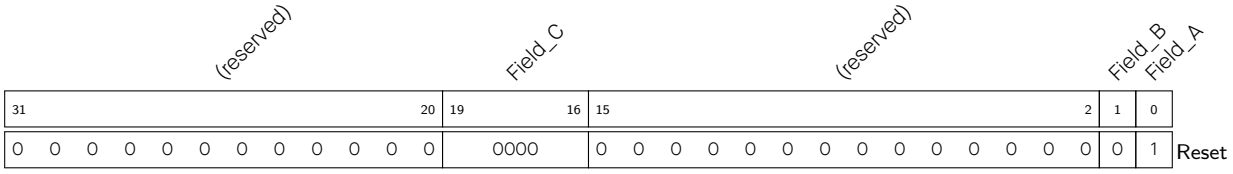
保留域的值不能修改。由于写寄存器时必须整体写，不能只写部分域，因此，在写带有保留域的寄存器时需要特别注意，只能采用以下两种方式：

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 23.21. 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 23-7。
- **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 23-7. 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
2024-02-19	v1.2	更新字体为 Maison Neue 更新以下章节： • 章节 2 通用 DMA 控制器 (GDMA)：更新 suc_eof 和 EOF 标志的相关描述 • 章节 19 UART 控制器 (UART)：更新产生 wake_up 所需的上升沿个数 • 章节 21 I2C 主机控制器 (I2C)：更新 I2C 超时值配置及字段 I2C_TIME_OUT_VALUE 的描述 • 章节 9 低功耗管理 (RTC_CNTL)：更新寄存器 RTC_CNTL_WDT_WKEY 的描述
2023-10-27	v1.1	新增章节 如何配置寄存器的保留域 和章节 中断配置寄存器 更新以下章节： • 章节 8 中断矩阵 (INTMTRX)：更新寄存器前缀 APB_CNTL 为 SYSCON • 章节 11 定时器组 (TIMG)：更新 TIMG_WDT_CLK_PRESCALE 的描述 • 章节 20 SPI 控制器 (SPI)：更新时钟相关信息 • 章节 5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)：更新 5.9 中的描述 • 章节 8 中断矩阵 (INTMTRX)：删除 INTERRUPT_COREO_GPIO_INTERRUPT_PRO_NMI_MAP_REG 寄存器及相关信息 • 章节 5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX)：删除 GPIO_PCPU_NMI_INT_REG 寄存器及相关信息 • 章节 7 芯片 Boot 控制：在小节 7.3.2 中添加 SPI Download Boot 相关信息，并将 Download Boot 重命名为 Joint Download Boot • 章节 7 芯片 Boot 控制：新增 eFuse 对芯片 Boot 控制的细节描述 • 章节 21 I2C 主机控制器 (I2C)：更新 I2C_COMD0_REG、I2C_SDA_FORCE_OUT 和 I2C_SCL_FORCE_OUT 的描述 • 章节 13 系统寄存器 (SYSTEM)：更新寄存器 SYSTEM_SYSCLK_CONF_REG 的描述
2023-05-20	v1.0	更新以下章节： • 章节 2 通用 DMA 控制器 (GDMA)：更新 GDMA_IN_SUC_EOF_CH*n*_INT 中断和 GDMA_INLINK_DSCR_ADDR_CH*n* 字段的描述 • 在章节 11 定时器组 (TIMG) 更新读取定时器值的步骤 • 在章节 12 看门狗定时器 (WDT) 删掉 ULP-RISC-V • 在章节 19 UART 控制器 (UART) 新增终止状态 (break condition) 的相关描述，更新停止位最大位数和相关描述 • 在章节 22 LED PWM 控制器 (LEDC) 新增占空比精度计算公式和表 14 位计数器 • 章节 23 片上传感器与模拟信号处理：更新图 23-3 和图 23-4 的样式表编号 • 章节 3 系统和存储器：更新 cache 相关描述

见下页...

接上页...

日期	版本	发布说明
2022-10-27	v0.3	新增以下章节： • 2 通用 DMA 控制器 (GDMA) • 9 低功耗管理 (RTC_CNTL) • 20 SPI 控制器 (SPI) • 23 片上传感器与模拟信号处理 更新以下章节： • 18 随机数发生器 (RNG)
2022-07-14	v0.2	新增以下章节： • 4 eFuse 控制器 (eFuse) • 15 ECC 硬件加速器 (ECC) 更新以下章节： • 1 ESP-RISC-V CPU • 5 IO MUX 和 GPIO 交换矩阵 (GPIO, IO MUX) • 6 复位和时钟
2022-05-18	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。