TX32M2300 数据手册

泰芯半导体
TaiXin Semiconductor

修订记录

日期	版本	描述
2023-04-11	V2.2	1、修改比较器部分关于正端和负端输入的选择描述；
2023-02-03	V2.1	1、解决部分设备打开阅读时，出现乱码问题；
2022-08-12	V2.0	1、重新整理了文档的格式；

注意：

1、本公司保留对以下所有产品在功能、性能、方案、设计及改进方面的最终解释权。

2、本公司保留对文档复制及更改的权利。

目录

1. 总介

1.1. 概述

本产品使用高性能的 32 位微控制器，最高工作频率可达 72MHz，内置 32KB 高速 Flash 存储器，6KB SRAM，丰富的增强型 I/O 端口和外设连接到外部总线。本产品包含 1 个 12 位的 ADC、一个 8 位精度 DAC、一个多功能比较器、3 个运算放大器、1 个 16 位高级定时器、5 个 16 位通用定时器、1 个 32 位通用定时器、1 个看门狗定时器、1 个系统滴答定时器。还包含标准的通信接口：2 个 SPI/IIC 接口和 2 个 UART 接口，其中 UART0 可以实现从任意一条 pin 里面选择一条进行代码升级，内置一个 32 位除 16 位有无符号除法器。

本产品产品系列工作电压为 2.0V~5.5V，工作温度范围-40◦C ~ 105◦C。多种省电工作模式保证低功耗应用的要求。

本产品提供 4 种封装，包括 LQFP32、SSOP28、SSOP24 和 TSSOP20。根据不同的封装形式，器件中的外设配置不尽相同。

下面给出了该系列产品中所有外设的基本介绍。

这些丰富的外设配置，使得本产品微控制器适合于多种应用场合：

- 风机，风扇等
- 消费类电子
- 智能家居
- 电机驱动和应用控制
- 医疗和手持设备
- 工业控制
- 工业应用：可编程控制器（PLC）、变频器、打印机和扫描仪

1.2. 产品特性

➢ 内核与系统

- 32-Bit RISC 架构的CPU
- 工作最大主频：72MHz
- 单周期 32 位乘法指令
- 32 个中断源，可配置 4 层中断优先级，支持中断入口地址Remap
- 支持位带操作
- 支持双pin调试接口

➢ **存储器**

- 32K Byte的闪存程序存储器(NO EEPROM Flash)，Sector擦写次数 20000 次
- 内部 6K Byte SRAM
- Boot Loader 支持片内 Flash、支持单/双pin UART 在线用户编程（IAP）/ 在线系统编程（ISP）

➢ **时钟、复位和电源管理**

- 2V ~ 5.5V 供电
- 上电/断电复位（POR/PDR）、可编程电压监测器（PVD）
- 外部 1-32MHz晶体振荡器
- 内嵌经出厂调校的 26MHz(+/- 1.5%) 高速振荡器
- 内嵌 128KHz低速振荡器
- PLL输出 72MHz时钟
- 内置时钟安全系统（CSS）
- WDT复位

➢ **DMA 支持**

- 支持的外设：EFLASH、UART、SPI/I2C、CRC、TIMER、ADC

➢ **GPIO**

- 最多能支持 30 个GPIO
- 所有 I/O 口可以触发边沿或电平响应中断，唤醒低功耗模式
- 所有端口均可输入输出 5V 信号
- 支持按键检测

- **通讯接口外设**
  - 2 个SPI高速串行接口，Master下最高支持系统时钟频率一半传输，支持 1/2/4 线主从模式，支持I2C模式
  - 2 个 UART 接口，支持RS232/RS485 协议，UART0 能支持任意IO程序升级
- **定时器**
  - 1 个 16 位高级定时器，支持 4 对互补输出或 8 个独立PWM输出，支持死区插入和事件刹车功能，支持单脉冲模式
  - 5 个 16 位通用定时器，1 个 32 位定时器，每个定时器支持捕获功能
  - 1 个看门狗定时器
  - 1 个系统滴答定时器
- **硬件加速单元**
  - 硬件有符号除法器 (32bit/16bit)
- **高安全性**
  - 支持 5/7/8/16/32 bit CRC效验，保证数据准确性
  - 支持代码加扰以及硬件加解密
- **低功耗**
  - 支持IDLE、STOP、SLEEP低功耗模式
  - 静态功耗<20uA@25℃
  - 低功耗唤醒时间最快 10us
- **1 个 12 位高速模数转换器**
  - 支持最高 1.2Mhz采样率
  - 多达 10 输入通道
- **1 个比较器**
  - 支持 7 个正端输入，3 个负端输入
  - 支持硬件通道轮询
- **3 个运算放大器**
  - 同向放大器

- 内置 4/6/8/10/12 倍增益可配
- 闭环增益带宽可选

- **内置温度传感器**
- **高可靠性**
  - ESD HBM 8KV
  - EFT ±4500V
  - Latch-up ±100mA @105℃
- **96 位的芯片唯一 ID（UID）**
- **封装**
  - Die Form
  - LQFP32/SSOP28/SSOP24/TSSOP20
- **工业级温度范围**
  - -40℃ ~ 105℃
- **应用**
  - 风机等
  - 电机驱动和应用控制
  - 医疗和手持设备
  - PC 游戏外设和 GPS 平台
  - 工业应用：可编程控制器（PLC）、变频器、打印机和扫描仪
  - 警报系统、视频对讲、和暖气通风空调系统等

2. 系统及存储器架构

TX32M2300 系列器件是基于 RISC 处理器的 32 位通用微控制器存储器的组织采用了哈佛结构，预先定义的存储器映射和高达 4 GB 的存储空间，充分保证了系统的灵活性和可扩展性。

2.1. 32 位 RISC 处理器

TX32M2300 系列所用的 32 位 RISC 处理器是一个具有低中断延迟时间和低成本调试特性的 32 位处理器。高集成度和增强的特性使这颗 RISC 处理器适合于那些需要高性能和低功耗微控制器的市场领域。

2.2. 系统架构

TX32M2300 系列器件采用 32 位多层总线结构，该结构可使系统中的多个主机和从机之间的并行通信成为可能。多层总线结构包括一个 AHB 互联矩阵、两个 AHB 总线和两个 APB 总线。AHB 互联矩阵的互联关系接下来将进行说明。在系统主从互联矩阵的互联关系列表中，”1”表示相应的主机可以通过 AHB 互联矩阵访问对应的从机，空白的单元格表示相应的主机不可以通过 AHB 互联矩阵访问对应的从机。

TX32M2300 系列主系统由以下部分构成：

- 二个驱动单元：
  - CPU 内核系统总线（S-bus）
  - DMA controller
- 三个被动单元
  - 内部闪存存储器
  - 内部 SRAM
  - AHB 到 APB 的桥（AHB2APB0/1），它连接所有的 APB 设备

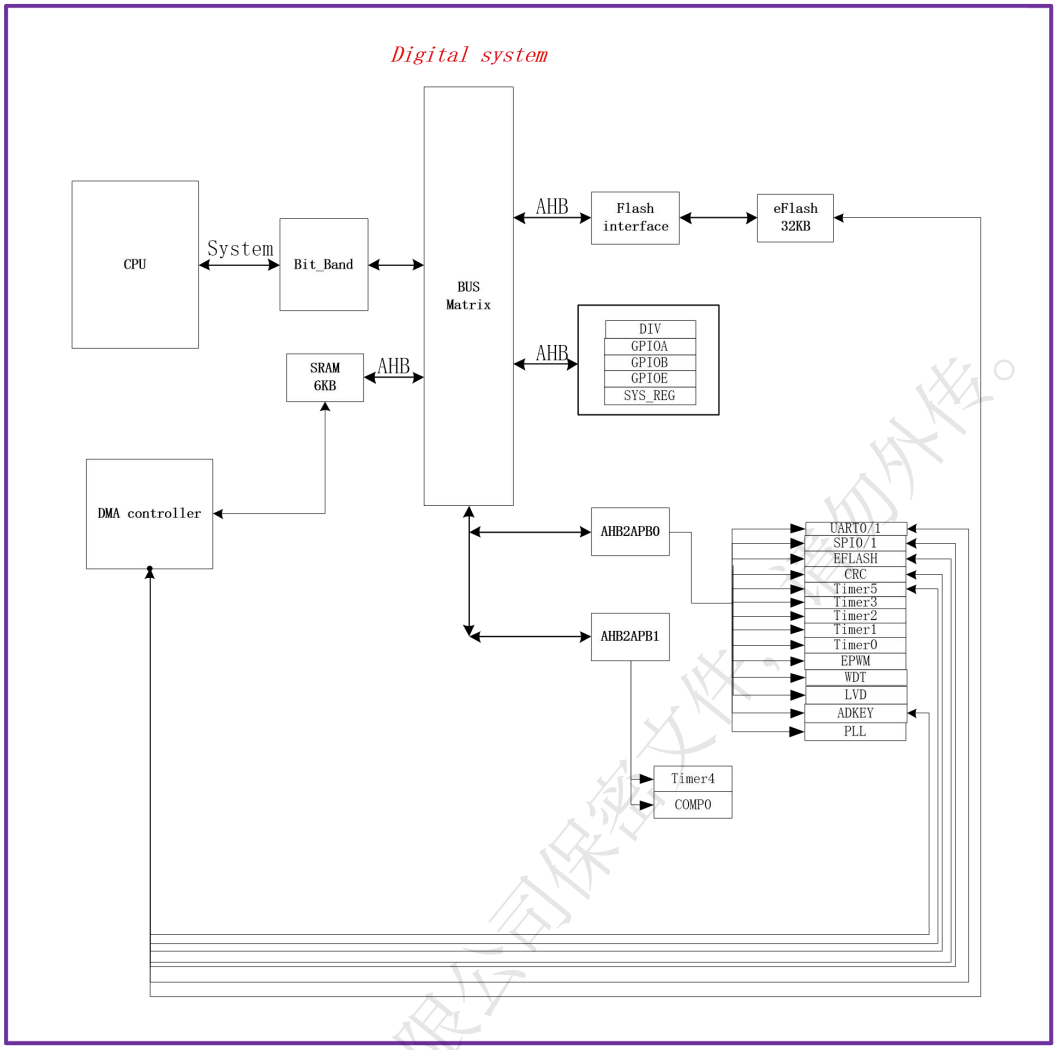


图 2-1 TX32M2300 系列总线系统架构图

系统总线

此总线连接 CPU 内核的系统总线（外设总线）到总线矩阵，总线矩阵协调着内核和各个高速部件间的访问。

DMA controller

此总线将 CPU 与各外设模块访问相联竞争，协调访问优先级，仲裁等。

表 2-1 DMA 模块使用的优先级排布

模块	优先级	说明
CPU	0	最高
SPI0	1	
SPI1	2	
UART0	3	
ADC	4	
UART1	5	
CRC	6	

TIMER5	7	
Eflash	8	最低

总线矩阵（BusMatrix）

总线矩阵管理着内核系统总线与各外设模块的访问仲裁，总线矩阵由主模块总线及从模块总线组成。AHB 外设通过总线矩阵与系统总线相连。

AHB 到 APB 桥（AHB2APB bridges-APB）

AHB 到 APB 桥在 AHB 与 APB 总线间提供同步连接。

注：当对 *APB* 寄存器进行 *8* 位或者 *16* 位访问时，该访问会被自动转换成 *32* 位的访问：桥会自动将 *16* 位或者 *8* 位的数据扩展以配合 *32* 位的宽度。

2.3. 存储器映射

此 32 位 RISC 处理器采用同一套总线来读取指令和加载/存储数据。指令代码和数据都位于相同的存储器地址空间，但在不同的地址范围。程序存储器，数据存储器，寄存器和 I/O 端口都在同一个线性的 4 GB 的地址空间之内。这是 32 位 RISC 的最大地址范围，因为它的地址总线宽度是 32 位。此外，为了降低不同客户在相同应用时的软件复杂度，存储映射是按 32 位 RISC 处理器提供的规则预先定义的。在存储器映射表中，一部分地址空间由 32 位 RISC 的系统外设所占用，且不可更改。此外，其余部分地址空间可由芯片供应商定义使用。

TX32M2300 系列器件的存储器映射表显示了 TX32M2300 系列器件的存储器映射，包括代码、SRAM、外设和其他预先定义的区域。简化了每个外设的地址译码。

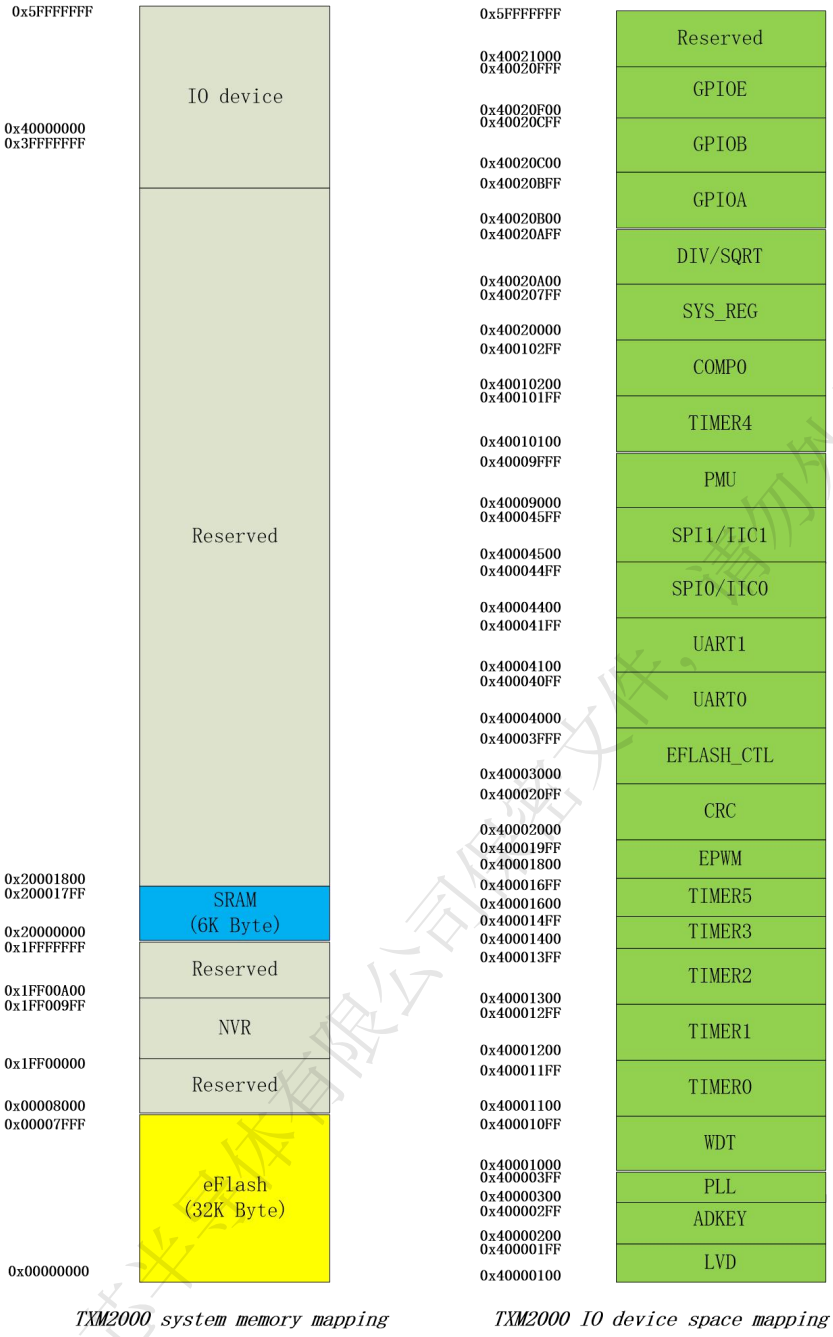


图 2-2 TX32M2300 系列存储器映射表

2.3.1. 位带操作

为了减少“读-改-写”操作的次数，32 位 RISC 处理器提供了一个可以执行单原子比特操作的位带功能。存储器映射包含了两个支持位带操作的区域。其中一个是 SRAM 区的最低 1MB 范围，第二个是片内外设区的最低 1MB 范围。这两个区域中的地址除了普通应用外，还有自己的“位带别名区”。位带别名区把每个比特扩展成一个 32 位的字。当用户访问位带

别名区时，就可以达到访问原始比特的目的。

下面的公式表明了位带别名区中的每个字如何对应位带区的相应比特或目标比特。

bit_word_addr =bit_band_base +(byte_offset×32)+(bit_number×4) (式 3- 1)

其中：

- bit_word_addr 指的是位带区目标比特对应在位带别名区的地址；
- bit_band_base 指的是位带别名区的起始地址；
- byte_offset 指的是位带区目标比特所在的字节的字节地址偏移量；
- bit_number 指的是目标比特在对应字节中的位置(0-7)。

例如，要想访问 0x2000 0200 地址的第 7 位，可访问的位带别名区地址是：

bit_word_addr = 0x2200 0000 + (0x200 * 32)+ (7 * 4)= 0x2200 401C (式 3- 2)

如果对 0x2200 401C 进行写操作，那么 0x2000 0200 的第 7 位将会相应变化；如果对 0x2200 401C 进行读操作，那么视 0x2000 0200 的第 7 位状态而返回 0x01 或 0x00。

2.3.2. 片上 SRAM 存储器

TX32M2300 系列内置最大可到 6K 字节的静态 SRAM。它可以以字节（8 位）、半字（16 位）或字（32 位）进行访问。 SRAM 起始地址为 0x2000 0000。数据总线上最大可到 6K 字节的 SRAM。可以被 CPU 或者 DMA 用最快的系统时钟且不插入任何等待进行访问。DMA 支持访问外设有 timer5、ADC、CRC、SPI0/1、UART0/1、EFLASH。

2.3.3. 片上 FLASH 存储器概述

闪存存储器有两个不同存储区域：

- 主闪存存储块，它包括应用程序和用户数据区（若需要时）
- 信息块，其包含两个部分：

- 选项字节（Option bytes）－内含硬件及存储保护用户配置选项。

- 系统存储器（System memory）－其包含 Boot loader 代码。参见内置闪存存储器章节。

闪存接口基于 AHB 协议执行指令和数据存取。其预取缓冲的功能可加速 CPU 执行代码的速度。

2.3.4. 引导配置

芯片复位后，通过客户自己在主闪存存储块的配置，可选择启动模式 pin 脚为 PC6 还是 PC0，默认工作是上拉还是下拉。在启动延迟之后， CPU 从地址 0x0000 0000 获取堆栈顶的地址，并从启动存储器的 0x0000 0004 指示的地址开始执行代码。

内嵌的自举程序存放在系统存储器，由厂家在生产时写入。该程序可以通过 UART0 对闪存进行重新编程。

2.4. MCLR 功能

默认状态下 TX32M2300 系列支持 MCLR 复位功能的。MCLR 指在某个特定 IO 引脚上输入一个持续 1.7ms 以上的低电平导致系统复位，如同重新上电复位一样。当用户通过 eflash 中用户自定义 bit 来禁止使能 MCLR 功能后。PE2 会变成普通 IO 功能。详细情况请参考闪存存储器中关于用户自定义区域的描述。

3. 嵌入式闪存

3.1. 闪存主要特性

- 32K 字节闪存存储器
- 存储器结构：
  - 主闪存空间：32K 字节
  - 副闪存空间（系统存储器）：2K 字节
- 带预取缓冲器的读接口
- 闪存编程和擦除操作
- 访问和写保护
- 低功耗模式

3.2. 闪存功能描述

3.2.1. 闪存结构

闪存空间由 32 位宽的存储单元组成，既可以存代码又可以存数据。主闪存块按 32 页（每页 1K 字节）分块，以页为单位设置写保护（参见存储保护相关内容）。

模块	名称	地址	大小（字节）
主闪存空间	页 0	0x0000_0000 - 0x0000_03FF	1K
	页 1	0x0000_0400 - 0x0000_07FF	1K
	页 2	0x0000_0800 - 0x0000_0BFF	1K
	页 3	0x0000_0C00 - 0x0000_0FFF	1K
	…	…	1K
	页 30	0x0000_7800 - 0x0000_7BFF	1K
	页 31	0x0000_7C00 - 0x0000_7FFF	1K
副闪存空间	扇区 0	0x1FF0_0000 - 0x1FF0_01FF	512
	扇区 1	0x1FF0_0200 - 0x1FF0_03FF	512
	扇区 2	0x1FF0_0400 - 0x1FF0_057F	384
	用户参数区	0x1FF0_0580 - 0x1FF0_06BF	64
	用户配置区	0x1FF0_05C0 - 0x1FF0_05FF	64
	芯片信息区	0x1FF0_0600 - 0x1FF0_064F	80
	EOTP 芯片区域	0x1FF0_0650 - 0x1FF0_068F	64
	EOTP 用户区域	0x1FF0_0690 - 0x1FF0_06EF	96
闪存寄存器接口	CTRLR0	0x4000_3000 - 0x4000_3003	4
	KST	0x4000_3004 - 0x4000_3007	4
	DONE	0x4000_3008 - 0x4000_300B	4
	PROG_ADDR	0x4000_3010 - 0x4000_3013	4
	PROG_DATA	0x4000_3018 - 0x4000_301B	4
	ERASE_CTRL	0x4000_3020 - 0x4000_3023	4
	TIME_REG0	0x4000_3030 - 0x4000_3033	4
	TIME_REG1	0x4000_3034 - 0x4000_3037	4
	NVR_PASSWORD	0x4000_3050 - 0x4000_3053	4
	MAIN_PASSWORD	0x4000_3054 - 0x4000_3057	4
	CRC_ADDR	0x4000_3058 - 0x4000_305B	4
	CRC_LEN	0x4000_305C - 0x4000_305F	4
	CRC_OUT	0x4000_3060 - 0x4000_3063	4

3.2.2. 闪存读保护

读操作在整个产品工作电压范围内都可以完成，用于存放指令或者数据，Flash运行在36MHz的工作频率上，若工作频率提升到36MHz以上，需要让Flash的读时序执行分频。

芯片带有cache缓冲区和预取缓冲区，提升Flash的访问效率。

若用户配置区打开保护的配置后，当SWD等Debug接口连接上时，会自动对Flash执行保护机制。

读操作有下列2个寄存器完成：

①配置寄存器 (CTRLR0)

②时序0寄存器(TIME_REG0)

3.2.3. 闪存烧写和擦除操作

烧写和擦除操作在整个产品工作电压范围内都可以完成。

烧写和擦除操作有下列 6 个寄存器完成，先根据烧写的时钟配置好烧写时序（TIME_REG1），再配置烧写密码，配置好编程地址，最后配置好编程数据，即可开始执行烧写，然后等烧写结束。

只要 CPU 不去访问 Flash 空间，或者访问 Flash 时已经 cache 命中，进行中的 Flash 写操作不会妨碍 CPU 的运行。也就是说，在对 Flash 进行写/擦除操作的同时，任何对 Flash 的访问都会令总线停顿，直到写/擦除操作完成后才会继续执行，这意味着在写/擦除 Flash 的同时不可以对它取指和访问数据

3.3. 寄存器

3.3.1. 寄存器基地址

Name	Base Address	Description
Eflash	0x40003000	Eflash 的基地址

3.3.2. 寄存器列表

Offset Address	Name	Description
0x0	EFLASH_CTRLR0	控制寄存器
0x04	EFLASH_KST	触发寄存器
0x08	EFLASH_DONE	状态寄存器
0x10	EFLASH_PROGADDR	编程地址寄存器
0x18	EFLASH_PROGDATA	编程数据寄存器
0x20	EFLASH_ERASECTRL	擦除控制寄存器
0x24	EFLASH_INTFERASEENA	中断打断 EFlash 工作状态控制寄存器
0x30	EFLASH_TIMEREG0	时序 0 配置寄存器
0x34	EFLASH_TIMEREG1	时序 1 配置寄存器
0x50	EFLASH_NVRPASSWORD	NVR 区域秘钥寄存器

0x54	EFLASH_MAINPASSWORD	Main 区域秘钥寄存器
0x58	EFLASH_CRCADDR	CRC 地址寄存器
0x5C	EFLASH_CRCLEN	CRC 长度寄存器
0x60	EFLASH_CRCOUT	CRC 输出寄存器

3.3.3. 寄存器详细说明

3.3.3.1. EFLASH_CTRLR0

Bit(s)	Name	Description	R/W	Reset
31:20	Reserved		RO	0
19	Error Interrupt IE	**异常中断的使能位** 使能该中断后，以下异常都会触发： ①非对齐地址编程 ②编程数据错误 ③无写权限 ④擦除失败	RW	0
18	Normal Interrupt IE	**正常功能的中断使能位** 使能该中断，以下情况会触发： ①编程 ②擦除 ③CRC 校验 ④自动编程	RW	0
17	Reserved		RO	0
16	Program Clock select	**Eflash 烧写时钟源选择，推荐使用 RC 时钟** 0x0：高速 RC 时钟 2 分频 0x1：晶振，若 RC 不准时才使用	RW	0
15:12	Reserved		RO	0
11	LVD Program Disable	**LVD 掉电烧录控制位** 在 LVD 断电时，是否允许打断 eflash program 和 erase，一般在断电时，立刻把重要数据保存在 eflash 上，就打开此功能，用于快速让 eflash 处于空闲状态 0x0：关闭 0x1：打开	RW	0
10:6	Reserved		RO	0
5	Block request Mode	**Program/Erase 的请求缓冲模式（测试使用）** 0x0：关闭 0x1：打开	RW	0
4	Cache Write Back Mode	**Program/Sector Erase 自动回写到 cache（测试使用）**	RW	1

		0x0：关闭 0x1：打开		
3	Reserved		RO	1
2	Prefetch Enable	**预取使能位** 0x0：关闭 0x1：打开	RW	0
1	Reserved		RO	0
0	Cache Enable	**Cache 使能位** 0x0：关闭 0x1：打开	RW	0

3.3.3.2. EFLASH_KST

Bit(s)	Name	Description	R/W	Reset
31	EFLASH Error Clear Enable	**异常标志清除使能位** 注意：需要与 BIT[15]同时配置才有效 0x0：关闭 0x1：打开	WO	0
30	EFLASH Pending Clear Enable	**完成状态标志清除使能位** 注意：需要与 BIT[14]同时配置才有效 0x0：关闭 0x1：打开	WO	0
29:27	Reserved		RO	0
26	CRC Kick Enable	**CRC 触发使能位** 注意：需要与 BIT[10]同时配置才有效 0x0：关闭 0x1：打开	WO	0
25	Auto Program EFLASH Enable	**自动编程使能位** 注意：需要与 BIT[9]同时配置才有效 0x0：关闭 0x1：打开	WO	0
24	Auto Program RAM Enable	**SRAM 自动编程使能位** 注意：需要与 BIT[8]同时配置才有效 0x0：关闭 0x1：打开	WO	0
23:21	Reserved		RO	0
20	Cache Clear Kick Enable	**Cache 清除使能位** 注意：需要与 BIT[4]同时配置才有效 0x0：关闭 0x1：打开	WO	0
19:11	Reserved		RO	0
15	EFLASH Error Clear	**异常标志清除位** 注意：需要与 BIT[31]同时配置才有效 写 1 清除	WO	0
14	EFLASH Pending Clear	**完成状态标志清除位**	WO	0

		注意：需要与 BIT[14]同时配置才有效 写 1 清除		
13:11	Reserved		RO	0
10	CRC Kick Start	**CRC 触发位** 注意：需要与 BIT[26]同时配置才有效 0x0：关闭 0x1：打开	WO	0
9	Auto Program EFLASH Kick start	**自动编程触发位** 注意：需要与 BIT[25]同时配置才有效 0x0：关闭 0x1：打开	WO	0
8	Auto Program RAM Kick start	**SRAM 自动编程触发位** 注意：需要与 BIT[24]同时配置才有效 0x0：关闭 0x1：打开	WO	0
7:5	Reserved		RO	0
4	Cache Clear Kick Start	**Cache 清除位** 注意：需要与 BIT[20]同时配置才有效 0x0：关闭 0x1：打开	WO	0
3:0	Reserved		RO	0

3.3.3.3. EFLASH_DONE

Bit(s)	Name	Description	R/W	Reset
31	EFLASH Error	**Error 状态标志位** 0x0：正常 0x1：异常	RO	0
30:20	Reserved		RO	0
19	Chip Erase Error	**全片擦除异常状态** 0x0：工作正常 0x1：擦除异常，error 有效	RO	0
18	Write permission Error	**编程权限异常状态** 0x0：工作正常 0x1：无编程权限，error 有效	RO	0
17	Program Address Error	**编程地址异常状态** 0x0：工作正常 0x1：非对齐地址，error 有效	RO	0
16	Program Data Error	**编程数据异常状态** 已经编程的数据是否和软件配置的数据一致 0x0：工作正常 0x1：数据不一致，error 有效	RO	0
15	EFLASH Busy	Eflash 工作状态标志	RO	0

		0x0：空闲状态 0x1：进行中		
14:11	Reserved		RO	0
10	CRC Done	**CRC 完成标志** 0：进行中 1：空闲状态	RO	1
9	Auto Program EFLASH Done	**自动编程 eflash 完成标志** 0x0：空闲 0x1：结束	RO	0
8	Auto Program RAM Done	**自动编程 RAM 完成标志** 0x0：空闲 0x1：结束	RO	0
7	Reserved		RO	0
6	Program Done	**Program 结束标志** 0x0：进行中 0x1：空闲状态	RO	1
5	Reserved		RO	0
4	Cache Clear Done	**Cache 初始化标志** 0x0：进行中 0x1：空闲状态	RO	1
3:2	Reserved		RO	0
1	Chip Erase Done	**Main 区域全擦除标志** 0x0：正在运行 0x1：空闲状态	RO	1
0	Sector Erase Done	**扇区(512 byte)擦除标志** 0x1：正在运行 0x0：空闲状态	RO	1

3.3.3.4. EFLASH_PROGADDR

Bit(s)	Name	Description	R/W	Reset
31:30	Program Byte	**编程的位宽设置** eflash 通过功能区的 DATA RAM 设置,分为 code 区域和 RAM 区域,其中 RAM 区域可以执行 1/2/4 byte 的编程，而 code 区域只能执行 4 byte 的编程 0x0：1 byte 0x1：2 byte 0x2：4 byte	RW	-
29	Program NVR Select	**编程的地址是否 NVR 区域** 0x0：MAIN 区域 0x1：NVR 区域	RW	-
28:0	Program Address	**eflash 编程的地址** 在自动编程模式下，作为目的地址： 在自动编程 RAM 模式下，此地址作为 RAM 地址；	RW	-

		在自动编程 EFLASH 模式下,此地址作为 EFLASH 地址。 在 code 区域里编程: 仅支持 4 byte 编程 在 DATARAM 区域里编程: 编程地址为 half word 对齐，支持 1/2 byte 编程 编程地址为 word 对齐，支持 1/2/4 byte 编程 编程地址为奇数，仅支持 1 byte 编程		

3.3.3.5. EFLASH_PROGDATA

Bit(s)	Name	Description	R/W	Reset
31:0	Program Data	**eflash 编程的数据，需要配置好地址才可进行**	RW	0

3.3.3.6. EFLASH_ERASECTRL

Bit(s)	Name	Description	R/W	Reset
31	Chip Erase Kick Start	**Chip 全擦除的触发** 写“1”触发，需要先配置密码	RO	0
30	Sector Erase Kick Start	**Sector 擦除的触发** 写“1”触发，需要先配置密码	RO	0
29	NVR Sector Enable	**NVR 的 sector 使能位** 0：MAIN 区域 1：NVR 区域	RW	0
28:7	Reserved		RO	0
6:0	Erase Sector Address	**擦除的 sector 选择，范围 0-127**	RW	0

3.3.3.7. EFLASH_TIMEREG0

Bit(s)	Name	Description	R/W	Reset
31:20	Reserved		RO	0
19:16	PGH	WEb low to PROG2 high hold min time is 15ns	RW	1
15:12	ADS	BYTE/Address/data setup min time is 15ns	RW	1
11:8	ADH	BYTE/Address/data hold min time is 15ns	RW	1
7:4	RW	Latency to next operation after PROG/ERASE low min time is 100ns	RW	8
3:0	RC	Read Cycle Min Time is 25/30ns	RW	0

3.3.3.8. EFLASH_TIME_REG1

Bit(s)	Name	Description	R/W	Reset

31:20	Reserved		RO	0
18:8	1ms unit	1ms 的时间配置值，以 1us 为单位	RW	1000
7:0	1us unit	1us 的时间配置值，系统默认是 26MHz	RW	26

3.3.3.9. EFLASH_NVR_PASSWORD

Bit(s)	Name	Description	R/W	Reset
31:10	NVR password	密码为 0x20150931，只有打开密码后，才能对 NVR 进行擦除和编程	RW	0

3.3.3.10. EFLASH_MAIN_PASSWORD

Bit(s)	Name	Description	R/W	Reset
31:10	Main password	密码为 0x20170230，只有打开密码后，才能对 Main 进行擦除和编程	RW	0

3.3.3.11. EFLASH_CRC_ADDR

Bit(s)	Name	Description	R/W	Reset
31:30	Reserved		RO	0
29	NVR Select	**地址是否 NVR 区域** 0x0：MAIN 区域 0x1：NVR 区域	RW	0
28:2	DMA Address	**CRC DMA 的开始地址，作为源地址** 在自动编程 RAM 模式下，此地址作为 EFLASH 地址； 在自动编程 EFLASH 模式下，此地址作为 RAM 地址。	RW	0
1:0	Reserved	保留，低 2 位地址固定为 0，word 对齐	RO	0

3.3.3.12. EFLASH_CRC_LEN

Bit(s)	Name	Description	R/W	Reset
31:2	DMA length	CRC DMA 的长度	RW	0
1:0	Reserved	保留，低 2 位地址固定为 0，word 对齐	RO	0

3.3.3.13. EFLASH_CRC_OUT

Bit(s)	Name	Description	R/W	Reset
31:00	CRC Result	**CRC 结果输出** 多项式 CRC-32 如下，出来的结果取反就是官方	RO	0

		的结果，写入 EFLASH 的 CRC 需要官方的值取反表示！！！ x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1		

4. 中断和事件

4.1. 嵌套向量中断控制器

基本功能：

- 中断都可屏蔽（除了 NMI）
- 16 个可编程的优先等级（使用了 4 位中断优先级）
- 低延迟的异常和中断处理
- 电源管理控制
- 系统控制寄存器的实现

嵌套向量中断控制器（NVIC）和处理器核的接口紧密相连，可以实现低延迟的中断处理和高效地处理晚到的中断。

嵌套向量中断控制器管理着包括核异常等中断。关于更多的异常和 NVIC 编程的说明请参考 CPU 技术参考手册。

4.2. 系统嘀嗒（SysTick）校准值寄存器

本芯片不支持使用外部时钟计时 1ms。

4.3. 中断功能描述

处理器和嵌套式矢量型中断控制器(NVIC)在处理(Handler)模式下对所有异常进行优先级区分以及处理。当异常发生时，系统自动将当前处理器工作状态压栈，在执行完中断服务子程序 (ISR)后自动将其出栈。

取向量是和当前工作态压栈并行进行的，从而提高了中断入口效率。处理器支持咬尾中断，可实现背靠背中断，大大削减了反复切换工作态所带来的开销。NVIC 异常类型列出了所有的异常类型。

表 4-1 CPU 中的 NVIC 异常类型

异常类型	向量号	优先级	向量地址	描述
-	0	-	0x00000000	保留
复位	1	-3	0x00000004	复位
NMI	2	-2	0x00000008	不可屏蔽中断
HardFault	3	-1	0x0000000C	各种硬件级别故障
MemManage	4	可设置	0x00000010	存储器管理
BusFault	5	可设置	0x00000014	预取指故障，存储器访问故障
UsageFault	6	可设置	0x00000018	未定义的指令或者非法状态
-	7-10	-	0x0000001C- 0x0000002B	保留
SVcall	11	可设置	0x0000002C	通过 SWI 指令实现系统服务调用
DebugMonitor	12	可设置	0x00000030	调试监控器
-	13	-	0x00000034	保留
PendSV	14	可设置	0x00000038	可挂起的系统服务请求
Systick	15	可设置	0x0000003C	系统节拍定时器

SysTick 校准值设为 0x1196E，SysTick 时钟频率配置为 HCLK，此时若 HCLK 时钟被配置为 72MHz，则 SysTick 中断会 1ms 响应一次。

表 4-2 中断向量表

中断序号	向量号	外设中断描述	向量地址
IRQ0	16	lvd_int	0x0000_0040
IRQ1	17	uart0_int	0x0000_0044
IRQ2	18	uart1_int	0x0000_0048
IRQ3	19	spi0_int	0x0000_004C
IRQ4	20	spi1_int	0x0000_0050
IRQ5	21	gpioa_int	0x0000_0054
IRQ6	22	gpiob_int	0x0000_0058
IRQ7	23	gpioe_int	0x0000_005C
IRQ8	24	wkpnd_int	0x0000_0060
IRQ9	25	timer0_int	0x0000_0064
IRQ10	26	timer1_int	0x0000_0068
IRQ11	27	timer2_int	0x0000_006C
IRQ12	28	timer3_int	0x0000_0070
IRQ13	29	timer4_int	0x0000_0074
IRQ14	30	timer5_int	0x0000_0078
IRQ15	31	epwm_tzint	0x0000_007C
IRQ16	32	epwm_etint	0x0000_0080
IRQ17	33	adkey_interrupt	0x0000_0084

IRQ18	34	div_ovf_int	0x0000_0088
IRQ19	35	crc_dma_int	0x0000_008C
IRQ20	36	comp_int	0x0000_0090
IRQ21	37	wdt_interrupt	0x0000_0094
IRQ22	38	eflash_int	0x0000_0098
IRQ23	39	adkey_int1	0x0000_009C
IRQ24	40	-	0x0000_00A0
IRQ25	41	-	0x0000_00A4
IRQ26	42	-	0x0000_00A8
IRQ27	43	-	0x0000_00AC
IRQ28	44	-	0x0000_00B0
IRQ29	45	-	0x0000_00B4
IRQ30	46	-	0x0000_00B8
IRQ31	47	-	0x0000_00BC

4.4. 外部中断/事件控制器（EXTI）

外部中断和时间控制器（EXTI）管理外部和内部异步事件/中断，并生成相应的事件请求到 CPU/中断控制器和到电源管理的唤醒请求。每个输入线可以独立地配置输入类型（脉冲或挂起）和对应的触发事件（上升沿或下降沿或者双边沿都触发）。每个输入线都可以独立地被屏蔽。挂起寄存器保持着状态线的中断请求。

4.4.1. 主要特征

EXTI 控制器的主要特性如下：

- 每个中断/事件都有独立的触发和屏蔽
- 每个中断线都有专用的状态位
- 支持多达 20 个软件的中断/事件请求
- 检测脉冲宽度低于 APB0 时钟宽度的外部信号

参见数据手册中电气特性部分的相关参数

4.4.2. 唤醒事件管理

TX32M2300 系列可以处理外部或内部事件来唤醒内核（WFE）。唤醒事件可以通过下述配置产生：

- 外设的控制寄存器使能一个中断，但不在 NVIC 中使能，同时在 CPU 的系统控制寄存器中使能 SEVONPEND 位。当 CPU 从 WFE 恢复后，需要清除相应外设的中断挂起位和外设 NVIC 中断通道挂起位（在 NVIC 中断清除挂起寄存器中）。
- 配置一个外部或内部 EXTI 线为事件模式，当 CPU 从 WFE 恢复后，因为对应事件线的挂起位没有被置位，不必清除相应外设的中断挂起位或 NVIC 中断通道挂起位。

5. 电源控制

5.1. 电源

芯片的工作电压（VCC） 为 2.0V ~ 5.5V。本芯片采用 CAPLESS 设计，无需在内置 LDO 输出上外挂电容。

5.1.1. 电压调节器

复位后调节器总是使能的。在需要低功耗的场合，可以使能低功耗工作模式，PMUCON0.lpen 做为功耗模式切换控制位。

5.2. 电源管理器

5.2.1. 上电复位（POR） 和掉电复位（PDR）

TX32M2300 系列内部有一个完整的上电复位（POR） 和掉电复位（PDR） 电路，当供电电压达到 2.7V

时系统既能正常工作。当 VDD 低于指定的限位电压 VPOR/VPDR 时，系统保持为复位状态，而无需外部复位电路。

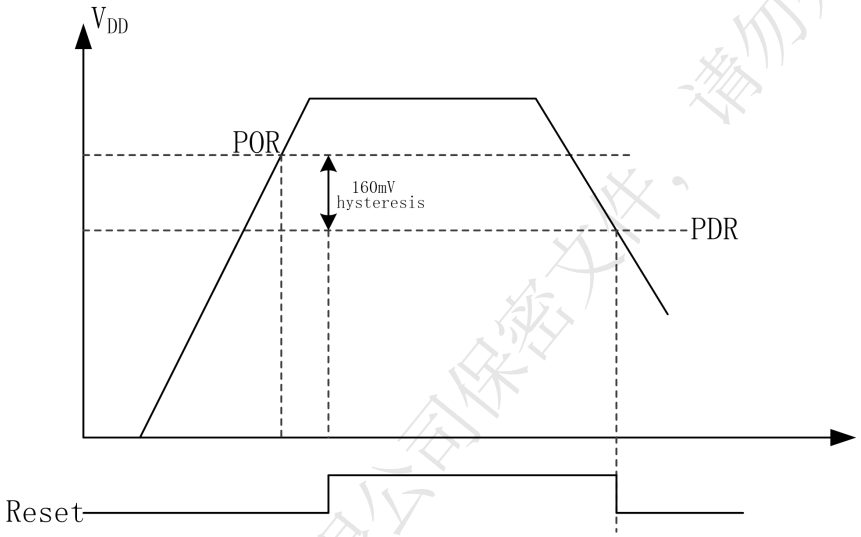


图 6-1 上电复位和掉电复位

5.2.2. 可编程电压监测器（PVD）

TX32M2300 系列内部集成两个电压检测器，一个检测外部供电 VCC,一个检测内部 LDO 输出 VDD，LDO 采样 capless 结构，封装上 VDD 不可见。两种检测电压均阈值可选。当系统监测到 VCC 或 VDD 电压低于配置电压值时，可以选择触发系统复位或通过使能 PVD 中断进入中断子函数。这一特性可用于用于执行紧急关闭任务。检测信号可以选择经过毛刺滤波电路或直接检测，由 LVD_CON.lvdvcc_bps_en 和 LVD_CON.lvdvdd_bps_en 来控制。

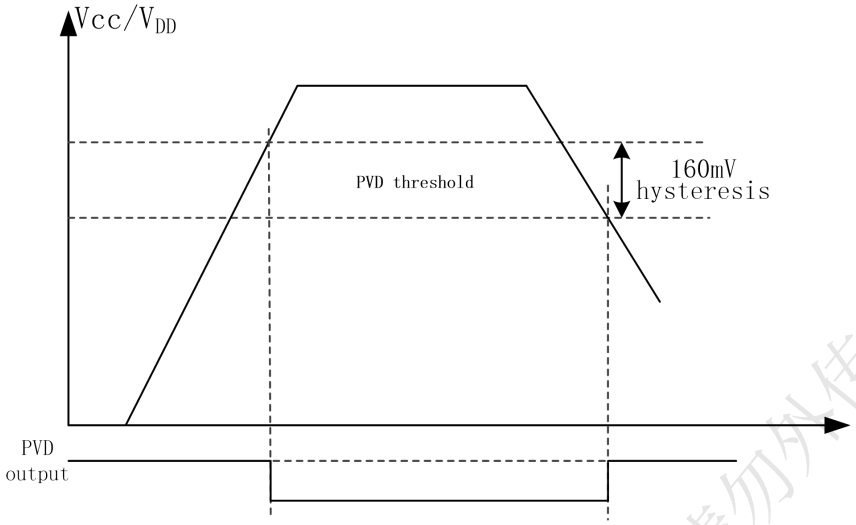


图 6-2 PVD 示意图

5.3. 低功耗模式

在系统或电源复位以后，微控制器处于运行状态，系统所用时钟为 256Khz 内部 RC。当 CPU 不需继续运行时，可以利用多种低功耗模式来节省功耗，例如等待某个外部事件时。用户需要根据最低电源消耗、最快速启动时间和可用的唤醒源等条件，选定一个最佳的低功耗模式。

TX32M2300 系列有三种低功耗模式：

- IDLE 模式（CPU 停止，所有外设包括 CPU 的外设，如 NVIC、系统时钟（SysTick）等仍在运行）
- STOP 模式（CPU,大部分外设停止，可依赖其他时间唤醒此模式后芯片继续运行）
- SLEEP 模式（所有时钟源都停止，依赖外部 IO 唤醒，可以选择复位芯片或继续运行）

此外，在运行模式下，可以通过以下方式中的一种降低功耗：

- 降低系统时钟
- 关闭 APB 和 AHB 总线上未被使用的外设时钟
- 合理配置 APB 与 AHB 的频率关系

5.4. 寄存器

5.4.1. 寄存器基地址

Name	Base Address	Description
LVD	0x40000100	LVD 的基地址

5.4.2. 寄存器列表

Offset Address	Name	Description
0x00	LVD_CON	配置寄存器

5.4.3. 寄存器详细说明

5.4.3.1. LVD_CON

Bit(s)	Name	Description	R/W	Reset
31	lvdvdd_pending	**VDD 电压低于设定阈值触发 pending** write 0 clean this bit	R	0
30	lvdvcc_pending	**VCC 电压低于设定阈值触发 pending** write 0 clean this bit	R	0
29	Reserved		-	-
28:22	dbs_lo_limit	**滤波检测电压信号低毛刺宽度** 单位为 LVDDBS_CLK	RW	0
21:15	dbs_hi_limit	**滤波检测电压信号高毛刺宽度** 单位为 LVDDBS_CLK	RW	0
14	lvdvcc_sync_dis	**检测到 VCC 低电压信号触发 LVDVCC 低电唤醒源是否需要同步** 0x0：需要同步 0x1：不同步	RW	0
13	lvdvdd_bps_en	**使能采样 debounce 后的 VDD 电压状态信号** 0x0：关闭 0x1：打开	RW	0
12	lvdvcc_bps_en	**使能采样 debounce 后的 VCC 电压状态信号** 0x0：关闭 0x1：打开	RW	0
11	lvd_oe	**使能阈值判断触发条件后中断，以及复位功能** 0x0：关闭	RW	1

		0x1：打开		
10	lvdvdd_rst_en	**VDD 阈值判断触发后复位系统使能** 0x0：中断，不复位 0x1：复位，不中断	RW	1
9	lvdvcc_rst_en	**VCC 阈值判断触发后复位系统使能** 0x0：中断，不复位 0x1：复位，不中断	RW	1
8:4	hlvd_s	**VCC LVD 档位设置** 0x00：1.7 0x01：1.8 0x02：1.9 0x03：2 0x04：2.1 0x05：2.2 0x06：2.3 0x07：2.4 0x08：2.55 0x09：2.65 0x0A：2.8 0x0B：2.95 0x0C：3.1 0x0D：3.25 0x0E：3.4 0x0F：3.55 0x10：3.35 0x11：3.55 0x12：3.75 0x13：3.95 0x14：4.15 0x15：4.35 0x16：4.55 0x17：4.75	RW	0x9
3:2	llvd_s	**VDD LVD 档位设置** 0x0：0.99 0x1：1.09 0x2：1.19 0x3：1.29	RW	0x2
1	llvd_en	**VDD LVD 使能位** 0x0：关闭 0x1：打开	RW	1
0	hlvd_en	**VCC LVD 使能位** 0x0：关闭 0x1：打开	RW	1

6. 复位和时钟控制

6.1. 复位

TX32M2300 系列支持电源复位和系统复位和主复位。

6.1.1. 系统复位

系统复位将复位除某些复位状态寄存器和特殊功能寄存器之外的所有寄存器。

当以下事件中的一件发生时，产生一个系统复位：

SLEEP 模式下外部 IO 口/LVDVCC 低电/COMP0 唤醒；

WDT 计数溢出复位

系统复位请求复位

UART0 升级程序复位

系统锁定复位

6.2. 主复位

主复位能将部分系统复位无法复位的寄存器复位。

以下事件可以触发一个主复位：

软件复位

PVD 检测到电压低事件，且控制器处于复位功能模式

当芯片支持 MCLR 复位时，MCLR 脚上一个持续 1ms 以上的低电平，触发复位

6.2.1. 电源复位

上电/掉电复位（POR/PDR 复位）都属于电源复位。电源复位将复位所有的逻辑和模拟模块。

复位入口矢量被固定在地址 0x0000_0004。

6.3. 时钟

6.3.1. XOSC 时钟

高速外部时钟信号（XOSC）由以下两种时钟源产生：

- 外部晶体/陶瓷谐振器
- 用户外部时钟

6.3.2. HIRC 时钟

HIRC 时钟信号由内部 26MHz 的振荡器产生，可直接作为系统时钟或作为 PLL 输入。HIRC 振荡器能够在不需要任何外部器件的条件下提供系统时钟。它的启动时间比 XOSC 晶体振荡器短。然而，即使在校准之后它的时钟频率精度仍较差。需要出厂时校准，校准值写在 flash 系统存储区域。可以程序在使用这个时钟前，可以读取并配置出高精度的 HIRC 时钟。经过出厂效验后，常温状态下 HIRC 精度为 26MHz(+/- 1.5%)。精确的频率在 0x1FF0_0600 – 0x1FF0_06FF 闪存区域有描述，用户可通过读取得到 HIRC 的精确频率。

如果 XOSC 晶体振荡器失效， HIRC 时钟会被作为备用时钟源。参考时钟安全系统（CSS） 。

6.3.3. PLL

内部 PLL 可以用内部时钟源如 XOSC,HIRC 等做为参考，系统配置小数分频比得到想要的任意时钟频率，经过分频电路后可以作为系统时钟。

6.3.4. LIRC 时钟

LIRC 振荡器担当一个低功耗时钟源的角色，它作为系统启动时钟为看门狗和其他单元提供时钟。 LIRC 时钟频率大约 128KHz（在 90KHz 和 166KHz 之间）。进一步信息请参考数

据手册中有关电气特性部分。

6.3.5. 系统时钟（SYSCLK）选择

四种不同的时钟源可被用来驱动系统时钟（SYSCLK）：

- 内部低速 128Khz LIRC
- 内部高速 26Mhz HIRC
- 外部高速晶振 XOSC
- 片内高速 PLL 时钟

6.3.6. DBSCLK 选择

四种不同的时钟源可被用来驱动 GPIO 的 glitch debounce clock（DBSCLK）：

- 外部高速晶振 XOSC
- 内部高速 26Mhz HIRC 的分频时钟
- SYSCLK
- 内部低速 128Khz LIRC

6.3.7. LVDDBS_CLK 时钟选择

四种不同的时钟源可被用来驱动 LVD VCC/VDD 的 debounceclock（LVDDBS_CLK），独立于系统时钟而工作：

- APB0CLK
- 内部高速 26Mhz HIRC 的分频时钟
- 内部低速 128Khz LIRC
- 外部高速晶振 XOSC

6.3.8. CMPCLK 时钟选择

四种不同的时钟源可被用来驱动 compartor 的 clock（CMPCLK）：

- 内部低速 128Khz LIRC
- 内部高速 26Mhz HIRC 的分频时钟
- 系统时钟
- 外部高速晶振 XOSC

当不被使用时，任一个时钟源都可被独立地启动或关闭，由此优化系统功耗。

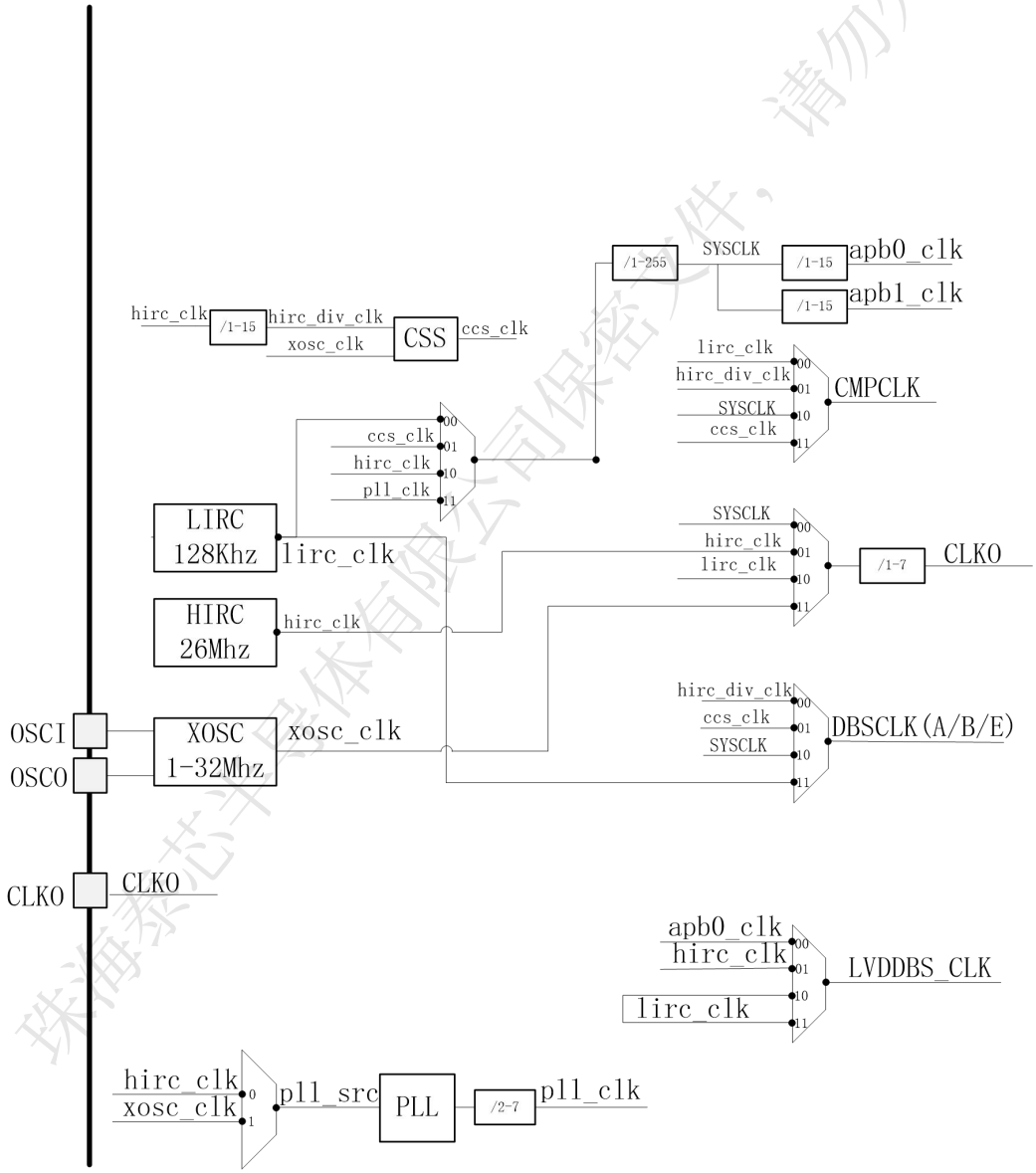


图 7-1 时钟结构

6.3.9. 时钟安全系统（CSS）

时钟安全系统可以通过软件被激活。在使用这个功能时，请确保 XOSC 已经被成功使能。如果 XOSC 时钟发生故障，产生时钟安全中断 CSSI，允许软件完成营救操作。此 CSSI 中断连接到 CPU 的 NMI 中断。注：一旦 *CSS* 被激活，并且 *XOSC*时钟出现故障， *CSS* 中断就产生，并且 *NMI* 也自动产生。 *NMI* 将被不断执行，直到 *CSS* 中断挂起位被清除。因此，在 *NMI* 的处理程序中必须通过设置时钟中断寄存器,（HOSC_MNT）里的 hosc_loss_pending 位写 1 来清除 *CSS* 中断。通过寄存器使能，时钟故障将导致内部 XOSC 时钟自动切换到 hirc_ div_clk。

6.4. 寄存器

6.4.1. 寄存器基地址

Name	Base Address	Description
System	0x40020000	System 的基地址

6.4.2. 寄存器列表

表 7-1 寄存器列表

Offset Address	Name	Description
0x0000	SYS_KEY	系统配置 key 寄存器
0x0004	SYS_CON0	系统控制寄存器 0
0x0008	SYS_CON1	系统控制寄存器 1
0x000c	SYS_CON2	系统控制寄存器 2
0x0010	SYS_CON3	系统控制寄存器 3
0x0014	SYS_CON4	系统控制寄存器 4
0x0018	SYS_CON5	系统控制寄存器 5
0x001c	SYS_CON6	系统控制寄存器 6
0x0020	SYS_CON7	系统控制寄存器 7
0x0024	CLK_CON0	时钟控制寄存器 0
0x0028	CLK_CON1	时钟控制寄存器 1
0x002c	CLK_CON2	时钟控制寄存器 2

0x0030	CLK_CON3	时钟控制寄存器 3
0x0034	CLK_CON4	时钟控制寄存器 4
0x0038	CLK_CON5	时钟控制寄存器 5
0x003c	CLK_CON6	时钟控制寄存器 6
0x0040	CLK_CON7	时钟控制寄存器 7
0x0044	HOSC_MNT	XOSC 控制寄存器
0x0048	SYS_ERR	系统异常错误寄存器
0x004c	WKUP_CON	Wakeup 寄存器
0x0050	LP_CON	低功耗寄存器
0x0054	MBIST_CON	Memory Bist 寄存器
0x0058	MBIST_MISR	Memory Bist Rom result 寄存器
0x005c	RESERVED	
0x0060	MODE	模式配置寄存器
0x0064	PMU_CON	PMU 控制器
0x0068	RPCON	System pending recording
0x006c	AMP_CON0	AMP 控制寄存器 0
0x0070	AMP_CON1	AMP 控制寄存器 1
0x0074	PMUBK	PMU lower power mode backup

6.4.3. 寄存器定义

6.4.3.1. SYS_KEY

Bit(s)	Name	Description	R/W	Reset
31:0	sys_key	**系统寄存器配置 key** 写入 0x3fac87e4 使所有系统寄存器写入使能，Setsys_keyother 值将清除此位 other 值将清除此位。读取返回 sys_key 状态 0x0：锁定所有系统寄存器写入 0x1：解锁所有系统寄存器写	RW	0x1

6.4.3.2. SYS_CON0

Bit(s)	Name	Description	R/W	Reset
31	fast_rst_en	**快速复位使能** 复位时间会在 1ms ~62us	R/W	0x0
30	sleep_goon_en	**触发 wakeup 后，是否继续运行** 0x0：复位 0x1：继续运行	R/W	0x0
29:27	sleep_dly_cnt	**Wakeup 后延迟 goon 运行时间配置** 0x0：two 128K RC cycles 0x1：three 128K RC cycles … 0x7：nine 128K RC cycles	R/W	0x0
26	dbs_soft_rst_	**GPIO debounce 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
25	crc_soft_rst_	**CRC 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
24:21	Reserved		R/W	0xF
20	gpio_soft_rst_	**GPIOA/GPIOB/GPIOE 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
19	adkey_soft_rst_	**ADCKey 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
18	uart1_soft_rst_	**UART1 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
17	uart0_soft_rst_	**UART0 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
16	spi1_soft_rst_	**SPI1 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
15	spi0_soft_rst_	**SPI0 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
14	epwm_soft_rst_	**EPWM 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
13:9	Reserved		R/W	0x1F
8	wdt_soft_rst_	**WDT 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
7	wdt_sys_soft_rst_	**WDT SYS 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
6	timer_soft_rst_	**TIMER 软件复位** 先写 0，后写 1 才能完成复位操作	R/W	0x1
5:0	Reserved	For future usage	R/W	0x3F

6.4.3.3. SYS_CON1

Bit(s)	Name	Description	R/W	Reset

31:14	Reserved		-	-
13	uart_update_dis	**串口单 pin 升级** 0x0：打开 0x1：关闭	R/W	0x0
12	debug_en	**Sleep 模式与 debug 联动配置** Sleep、stopclk 模式将在调试连接时自动唤醒。 0x0：打开 0x1：关闭	R/W	0x0
11	int_remap_en	**中断入口地址重映射使能** 0x0：EFLASH 0x1：SRAM	R/W	0x0
10	nmi_inv_sel	**NMI Pin 检测信号反转配置** 0x0：高电平触发 NMI 0x1：低电平触发 NMI	R/W	0x0
9	cp_mode_sysclk_en	**CP 模式下切换系统时钟** 0x0：切换 0x1：保持 ate-clk	R/W	0x1
8	swd_en	**SWD 使能** 0x0：关闭 0x1：打开	R/W	0x1
7	lvdvcc_wkup_en	**VCC LVD 唤醒使能** 0x0：关闭 0x1：打开	R/W	0x0
6	clk_test_oe	**内部时钟输出到 PA0** 0x0：关闭 0x1：打开	R/W	0x0
5	sys_err_resp_en	**系统总线访问内存时，异常响应反馈使能** 0x0：关闭 0x1：打开	R/W	0x0
4	sys_err_int_en	**系统总线访问出内存异常时，触发 NMI 中断** 0x0：关闭 0x1：打开	R/W	0x0
3	hosc_loss_nmi_en	**监控 XOSC 丢失触发 NMI 中断** 0x0：关闭 0x1：打开	R/W	0x0
2	rxev_enable	**启用 PA1 作为 CPU 的接收事件** 0x0：关闭 0x1：打开	R/W	0x0
1	nmi_int_enable	**开启 PA0 作为 CPU 的外部 NMI 输入** 0x0：关闭 0x1：打开	R/W	0x0
0	lockup_enable	**使能系统锁定触发系统复位** 0x0：关闭 0x1：打开	R/W	0x0

6.4.3.4. SYS_CON2

Bit(s)	Name	Description	R/W	Reset
31:30	Reserved	-	-	-
29:27	pe_deb_en	**GPIOE debounce 使能** 对应控制 GPIOE0 ~GPIOE2	R/W	0x0
26:16	pb_deb_en	**GPIOB debounce 使能** 对应控制 GPIOB0 ~GPIOB10	R/W	0x0
15:0	pa_deb_en	**GPIOA debounce 使能** 对应控制 GPIOA0 ~GPIOA15	R/W	0x0

6.4.3.5. SYS_CON3

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.6. SYS_CON4

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.7. SYS_CON5

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.8. SYS_CON6

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.9. SYS_CON7

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.10. CLK_CON0

Bit(s)	Name	Description	R/W	Reset
31:18	Reserved	-	R/W	-

17:16	lvd_debclk_sel	**Lvd debounce 时钟选择** 0x0: apb0_clk 0x1: HIRC 26M 0x2: LIRC 128K 0x3: LIRC 128K	R/W	0x0
15:14	comp_clk_sel	**比较器时钟选择** 0x0：LIRC 128K 0x1：divider output from HIRC 26M 0x2：sys_clk 0x3：XOSC	R/W	0x0
13:12	Reserved	-	R/W	-
11:10	gpioe_dbs_sel	**GPIOE debounce 时钟选择** 0x0：divider output from HIRC 26M 0x1：XOSC 0x2：sys_clk 0x3：LIRC 128K	R/W	0x0
9:8	gpiob_dbs_sel	**GPIOB debounce 时钟选择** 0x0：divider output from HIRC 26M 0x1：XOSC 0x2：sys_clk 0x3：LIRC 128K	R/W	0x0
7:6	gpioa_dbs_sel	**GPIOA debounce 时钟选择** 0x0：divider output from HIRC 26M 0x1：XOSC 0x2：sys_clk 0x3：LIRC 128K	R/W	0x0
5	Reserved	-	R/W	-
4	pll_refclk_sel	**Pll reference 时钟选择** 0x0：HIRC 26M 1:XOSC	R/W	0x0
3:2	clock_to_io_sel	**Output clock 选择** 0x0：sys_clk 0x1：HIRC 26M 0x2：LIRC 32K 0x3：XOSC	R/W	0x0
1:0	sysclk_sel	**系统时钟选择** 0x0：LIRC 128K 0x1：XOSC 0x2：HIRC 26M 0x3：pll_clk	R/W	0x0

6.4.3.11. CLK_CON1

Bit(s)	Name	Description	R/W	Reset
31	Reserved	-	-	-

30:28	clk_to_io_div	**Clock to IO 分配比** 0x0: divide by 1 0x1: divide by 2 0x2: divide by 3 … 0x6: divide by 7 0x7: close clock output	R/W	0x0
27:24	hirc_clk_div	**HIRC 26M clock 分频比** 0x0: divide by 1 0x1: divide by 2 0x2: divide by 3 … 0xE: divide by 15 0xF: close HIRC divide clock	R/W	0x0
23:19	Reserved	-	-	-
18:16	pll_clk_div	**pll clock 分频比** when n=[1,6], pll_clk divide by n+1 when n=7,stoppll_clk n=0 is forbidden	R/W	0x1
15:12	apb1clk_div	**PB1 clock 分频比** 0x0: divide by 1 0x1: divide by 2 0x2: divide by 3 … 0xE: divide by 15 0xF: close apb1_clk	R/W	0x0
11:8	apb0clk_div	**APB0 clock 分频比** 0x0: divide by 1 0x1: divide by 2 0x2: divide by 3 … 0xE: divide by 15 0xF: close apb0_clk	R/W	0x0
7:0	sysclk_div	**System clock 分频比** 0x0: divide by 1 0x1: divide by 2 0x2: divide by 3 … 0xFE: divide by 255 0xFF: close sys_clk	R/W	0x0

6.4.3.12. CLK_CON2

Bit(s)	Name	Description	R/W	Reset
31	reserved	-	-	-

30	clk_source_en_bps	**pll_clk and xosc clock glitch free? ? ?** 0x0：打开 0x1：关闭	R/W	0x1
29	cp_clk_en	**CP 模式使能 CPU 时钟** 0x0：关闭 0x1：打开	R/W	0x0
28	test_clk_en	**ATE 时钟输出使能** 0x0：关闭 0x1：打开	R/W	0x0
27	comp_clk_en	**比较器时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
26	Reserved	-	-	-
25	crc_clk_en	**CRC 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
24	eflash_mem_clk_en	**eflash erase/program 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
23	Reserved	-	-	-
22	hwdiv_clk_en	**硬件除法器时钟使能** 0x0：关闭 0x1：打开	R/W	0x0
21:19	Reserved	-	-	-
18	uart1_clk_en	**UART1 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
17	uart0_clk_en	**UART0 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
16	spi1_clk_en	**SPI1 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
15	spi0_clk_en	**SPI0 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
14	epwm_clk_en	**EPWM 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
13	timer5_clk_en	**TIMER5 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
12	timer3_clk_en	**TIMER3 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1

11	timer2_clk_en	**TIMER2 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
10	timer1_clk_en	**TIMER1 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
9	timer0_clk_en	**TIMER0 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
8	wdt_clk_en	**WDT 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
7	wdt_sys_clk_en	**WDT sys 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
6	timer4_clk_en	**TIMER4 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
5:3	Reserved	-	-	-
2	sram0_clk_en	**SRAM0 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
1	ahb1_clk_en	**AHB1 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1
0	ahb0_clk_en	**AHB0 时钟使能** 0x0：关闭 0x1：打开	R/W	0x1

6.4.3.13. CLK_CON3

Bit(s)	Name	Description	R/W	Reset
31	hrcosc_en_flag	**HIRC 26M 工作状态** 0x0：not ready 0x1：ready	RO	0x1
30:29	Reserved	-	-	-
28	hrcosc_vtest2	**VDD 测试使能** 输出 VTSOUT 0x0：关闭 0x1：打开	R/W	0x0
27	hrcosc_vtest1	**VDDOSC 测试使能** 输出 VTSOUT 0x0：关闭 0x1：打开	R/W	0x0
26	hrcosc_vsel	**LDO 参考选择**	R/W	0x0

		0x0：Voltage bias 0x1：Current		
25:19	hrcosc_sc	**HRCOSC 频率控制** 0x0：lowest ... 0x7F：highest	R/W	0x0
18	hrcosc_ldos	**RCOSC LDO 电压选择** 0x0：1.5v 0x1：1.6v	R/W	0x0
17	hrcosc_en	**RCOSC 使能** 0x0：disable 0x1：enable	R/W	0x1
16:15	hxosc_fbres		R/W	0x3
14:10	hxosc_dr		R/W	0x10
9:6	hxosc_cto		R/W	0x0
5:2	hxosc_cti		R/W	0x0
1	hxosc_hy		R/W	0x1
0	hxosc_en	**XOSC 使能** 0x0：关闭 0x1：打开	R/W	0x0

6.4.3.14. CLK_CON4

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.15. CLK_CON5

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.16. CLK_CON6

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.17. CLK_CON7

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.18. HOSC_MNT

Bit(s)	Name	Description	R/W	Reset
31:16	high_limit	**XOSC 工作上限限制**	R/W	0x1450
15	hosc_mnt_en	**XOSC 监控功能使能** 0x0：关闭 0x1：打开	R/W	0x0
14	hosc_loss_pending	**读取寄存器** 0x0：正常状态 0x1：挂起，表示外部 xosc 失败 **写寄存器** 0x0：清除 pending 0x1：无效	R/W	0x0
13	hosc_loss_sw_en	**当检测到外部 osc 故障时,硬件自动将 osc 时钟从 xosc 切换到 HIRC** 0x0：关闭 0x1：打开	R/W	0x0
12:0	low_limit	**XOSC 工作下限限制**	R/W	0xA7

6.4.3.19. SYS_ERR

Bit(s)	Name	Description	R/W	Reset
31:1	Reserved	-	-	-
1	clk_err	**时钟异常标志** 0x0：工作正常 0x1：时钟使用错误时，此位可由硬件设置，软件清除	R/W	0
0	sys_err0	**系统访问异常** 如果设置 sys_err_int_en,这个挂起会导致 NMI 中断	R/W	0

clk_err 触发情况			
	lirc disable	hirc disable	xosc disable
sys_clk_sel_lirc	√		
sys_clk_sel_hirc		√	
sys_clk_sel_xosc			√
sys_clk_sel_pll& pll_ref_sel_lirc			
sys_clk_sel_pll& pll_ref_sel_hirc		√	
sys_clk_sel_pll& pll_ref_sel_xosc			√

6.4.3.20. WKUP_CON

Bit(s)	Name	Description	R/W	Reset
31:1	Reserved	-	-	-
29:24	wkup_pnd	**Wakeup IO 的边沿检测触发** 0x0：无触发 0x1：已触发 对应关系： BIT24->port0、BIT25->port1 以此类推	RO	-
23:22	Reserved	-	-	-
21:16	clr_pnd	**Wakeup pending 清除** **写 1 清除**	WO	-
15:14	Reserved	-	-	-
13:8	wkup_edge	**Wakeup 的边沿唤醒选择** 0x0：上升沿触发 0x1：下降沿触发	R/W	0x0
7:6	Reserved	-	-	-
5:0	wkup_en	**Wakeup 使能** 0x0：关闭 0x1：打开 BIT0~BIT3：外部 GPIO 唤醒 BIT4：比较器唤醒 BIT5：LVD 唤醒	R/W	0x0

6.4.3.21. LP_CON

Bit(s)	Name	Description	R/W	Reset
31:9	Reserved	-	-	-
8	lp_mode	**LP 使能** 0x0：关闭 0x1：打开	0x0	R/W
7	lp_mode_auto_en	**Sleep 模式下自动使能 LP 模式** 0x0：关闭 0x1：打开	0x0	R/W
6	hirc_auto_enable	**XOSC 检测出现异常时，自动使能并切换 HIRC26M** 0x0：关闭 0x1：打开	0x0	R/W
5	rc32k_soft_en	**LIRC 使能** 0x0：关闭 0x1：打开	0x1	R/W
4	rc32k_auto_dis	**进入 sleep 模式自动关闭 LIRC** 0x0：关闭 0x1：打开	0x0	R/W
3	hirc_auto_dis	**进入 sleep/stop 模式自动关闭 LIRC** 0x0：关闭	0x0	R/W

		0x1：打开		
2	sram0_auto_dis	**进入 sleep/stop 模式自动关闭 SRAM** 0x0：关闭 0x1：打开	0x0	R/W
1	stopclk	**进入 stop 模式** 0x0：关闭 0x1：打开	0x0	R/W
0	sleep	**进入 sleep 模式** 0x0：关闭 0x1：打开	0x0	R/W

6.4.3.22. MBIST_CON

Bit(s)	Name	Description	R/W	Reset
31:21	Reserved	-	-	-
20	mbist_fail_h	**MBIST 异常标志** 读操作： 0x1：MBIST 异常 写操作： 0x1：清除异常标志	R/W	0x0
19	mbist_tst_done	**MBIST 完成标志** 读操作： 0x1：MBIST 测试完成 写操作： 0x1：清除完成标志	R/W	0x0
18	mbist_clk_en	**MBIST 时钟使能** 0x0：关闭 0x1：打开	R/W	0x0
17:12	Reserved	-	-	-
11	mbist_rf1p_debugz	**MBIST rf1p 调试使能** 0x0：关闭 0x1：打开	R/W	0x0
10	mbist_rf1p_hold_l	**MBIST rf1p 调试使能** 0x0：关闭 0x1：打开	R/W	0x0
9	mbist_rf1p_test_h	**MBIST rf1p 调试使能** 0x0：关闭 0x1：打开	R/W	0x0
8:0	Reserved	-	-	-

6.4.3.23. MBIST_MISR

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.24. MODE

Bit(s)	Name	Description	R/W	Reset
31:0	Reserved	-	-	-

6.4.3.25. PMU_CON

Bit(s)	Name	Description	R/W	Reset
31:23	Reserved		-	-
22:20	dset		R/W	0x4
19:16	pdcore	**VDD 下拉信号选择** 0x0：304pa 0x1：68ua 0x2：263ua 0x3：330ua 0x4：1.31ma 0x5：1.38ma 0x6：1.57ma 0x7：1.64ma 0x8：5.24ma 0x9：5.31ma 0xA：5.51ma 0xB：5.57ma 0xC：6.56ma 0xD：6.62ma 0xE：6.81ma 0xF：6.89ma	R/W	0x4
15:12	lpset	**LP 模式关闭模块** 0x0：关闭 0x1：打开	R/W	0x0
11	lpen	**LP 模式使能** 0x0：关闭 0x1：打开	R/W	0x0
10	vrefaen	**VREF 使能** 0x0：关闭 0x1：打开	R/W	0x1
9	tsensen	**内部温度传感器** 0x0：关闭 0x1：打开	R/W	0x0
8	irefen	**偏置电流使能** 低功率模式时需要设置为 0 0x0：关闭 0x1：打开	R/W	0x1

7	bor5en	**VCC POR 使能** 0x0：关闭 0x1：打开	R/W	0x1
6:3	vddset	**VDD 电压档位选择** 0x0：1.2v 0x1：1.25v 0x2：1.3v 0x3：1.35v 0x4：1.4v 0x5：1.45v 0x6：1.5v 0x7：1.55v 0x8：1.6v 0x9：1.65v 0xA：1.7v 0xB：1.75v 0xC：1.8v 0xD-0xF：1.85v	R/W	0x6
2:0	vbgset	**BGR 电压档位选择** 0x0：1.118v 0x1：1.145v 0x2：1.174v 0x3：1.206v 0x4：1.230v 0x5：1.259v 0x6：1.287v 0x7：1.287v	R/W	0x3

注：vbgset/ vddset 将在上电过程中加载 trim 值，所以他们读回来可能不是默认值参考上表。

6.4.3.26. RPCON

Bit(s)	Name	Description	R/W	Reset
31:20	Reserved		-	-
19	uart0_reset_clr	**Uart0 复位 pending 清除** 写 1 清除	WO	0
18	lockup_reset_clr	**Lockup 复位 pending 清除** 写 1 清除	WO	0
17	soft_reset_clr	**软件复位 pending 清除** 写 1 清除	WO	0
16	sleep_sta_clr	**Sleep pending 清除** 写 1 清除	WO	0
15:4	Reserved		-	-
3	uart0_update_pending	**Uart0 复位 pending**	RO	0
2	lock_reset_pending	**Lockup 复位 pending**	RO	-
1	soft_reset_pending	**软件复位 pending**	RW	-

		写 1 复位		
0	sleep_pending	**Sleep pending**	RO	-

6.4.3.27. AMP_CON0

Bit(s)	Name	Description	R/W	Reset
31:27	qcsel3	**AMP3 filter cap 选择** 0x01：300fp 0x02：300fp 0x04：400fp 0x08：400fp 0x10：600fp	R/W	0x1
26:24	gainsel3	**AMP3 gain 选择** 0x0：x4 0x1：x6 0x2：x8 0x3：x10 0x4：x12 0x5-0x7：disable feedback loop	R/W	0x2
23:19	qcsel2	**AMP2 filter cap 选择** 0x01：300fp 0x02：300fp 0x04：400fp 0x08：400fp 0x10：600fp	R/W	0x1
18:16	gainsel2	**AMP2 gain 选择** 0x0：x4 0x1：x6 0x2：x8 0x3：x10 0x4：x12 0x5-0x7：disable feedback loop	R/W	0x2
15:11	qcsel1	**AMP1 filter cap 选择** 0x01：300fp 0x02：300fp 0x04：400fp 0x08：400fp 0x10：600fp	R/W	0x1
10:8	gainsel1	**AMP1 gain 选择** 0x0：x4 0x1：x6 0x2：x8 0x3：x10 0x4：x12 0x5-0x7：disable feedback loop	R/W	0x2

7	ampldo_en	**AMP LDO 使能** 0x0：关闭 0x1：打开	R/W	0x0
6	bias_en	**BIAS 使能** 0x0：关闭 0x1：打开	R/W	0x0
5	biasadd_en	**BIASADD 使能** 0x0：关闭 0x1：打开	R/W	0x0
4	rcchan_en	**RCCHAN 使能** 0x0：关闭 0x1：打开	R/W	0x0
3	vcmbuff_en	**VCMBUFF 使能** 0x0：关闭 0x1：打开	R/W	0x0
2	amp3_en	**AMP3 使能** 0x0：关闭 0x1：打开	R/W	0x0
1	amp2_en	**AMP2 使能** 0x0：关闭 0x1：打开	R/W	0x0
0	amp1_en	**AMP1 使能** 0x0：关闭 0x1：打开	R/W	0x0

6.4.3.28. AMP_CON1

Bit(s)	Name	Description	R/W	Reset
31:13	Reserved	-	-	-
12	vout2pad_en	**AMP2 输出到 PB2 使能** 0x0：关闭 0x1：打开	R/W	0x0
11	vout1pad_en	**AMP1 输出到 PB1 使能** 0x0：关闭 0x1：打开	R/W	0x0
10:9	rcrsel	**AMP3 RC 滤波电阻选择** 0x0：1K 0x1：10K 0x2：50K 0x3：100K	R/W	0x0
8:6	restrim3	**AMP3 feedback 配置** 0x0：1.17X 0x1：1.05X 0x2：1X 0x3：0.95X	R/W	0x2

		0x4：0.87X 0x5 - 0x7：disable feedback loop		
5:3	restrim2	**AMP2 feedback 配置** 0x0：1.17X 0x1：1.05X 0x2：1X 0x3：0.95X 0x4：0.87X 0x5 - 0x7：disable feedback loop	R/W	0x2
2:0	restrim1	**AMP1 feedback 配置** 0x0：1.17X 0x1：1.05X 0x2：1X 0x3：0.95X 0x4：0.87X 0x5 - 0x7：disable feedback loop	R/W	0x2

6.4.3.29. PMUBK

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	lpen_bk	**backup register of lpen** 0x0：关闭 0x1：打开	R/W	0x1
14:11	lpset_bk	**backup register of lpset** 0x0：关闭 0x1：打开	R/W	0xF
10:7	pdcore_bk	**backup register of pdcore** 0x0：304pa 0x1：68ua 0x2：263ua 0x3：330ua 0x4：1.31ma 0x5：1.38ma 0x6：1.57ma 0x7：1.64ma 0x8：5.24ma 0x9：5.31ma 0xA：5.51ma 0xB：5.57ma 0xC：6.56ma 0xD：6.62ma 0xE：6.81ma 0xF：6.89ma	R/W	0x0
6:3	vddset_bk	**backup register of VDD**		0x6

		0x0：1.2v 0x1：1.25v 0x2：1.3v 0x3：1.35v 0x4：1.4v 0x5：1.45v 0x6：1.5v 0x7：1.55v 0x8：1.6v 0x9：1.65v 0xA：1.7v 0xB：1.75v 0xC：1.8v 0xD - 0xF：1.85v		
2:0	vbgset_bk	**backup register of BGR** 0x0：1.118v 0x1：1.145v 0x2：1.174v 0x3：1.206v 0x4：1.230v 0x5：1.259v 0x6：1.287v 0x7：1.287v	R/W	0x3

6.4.3.30. 芯片信息 0（0x1FF00600）

Bit(s)	Name	Description	R/W	Reset
31:0	UID0	UID 信息 0	RO	0xFFFFFFFF

6.4.3.31. 芯片信息 1（0x1FF00604）

Bit(s)	Name	Description	R/W	Reset
31:0	UID1	UID 信息 1	RO	0xFFFFFFFF

6.4.3.32. 芯片信息 2（0x1FF00608）

Bit(s)	Name	Description	R/W	Reset
31:0	UID2	UID 信息 2	RO	0xFFFFFFFF

7. GPIO

每组 GPIO 端口有四个 32 位配置寄存器(GPIOx_MODER,GPIOx_OTYPER, GPIOx_OSPEEDR and GPIOx_PUPDR)，两个 32 位数据寄存器(GPIOx_IDR and GPIOx_ODR)，一个 32 位置位/复位寄存器(GPIOx_BSRR)和一个 32 位翻转寄存器(GPIOx_TGL)。另外，所有 GPIO 有一个 32 位锁定寄存器(GPIOx_LCKR) 和两个复用功能选择寄存器 (GPIOx_AFRH and GPIOx_AFRL)。

7.1. GPIO 主要特征

- 输出状态：推挽或开漏+上下拉
- 输出寄存器状态值 (GPIOx_ODR) 或者外设功能输出 (alternatefunctionoutput)
- 每个 IO 驱动能力可配置
- 输入状态：浮空、上下拉、模拟
- 输入数据到数据寄存器(GPIOx_IDR) 或外设 (alternate function input)
- 独立置位/复位/翻转 IO 状态 (GPIOx_ BSRR、GPIOx_TGL)
- 通过加锁配置 (GPIOx_LCKR) 锁定 IO 状态
- 模拟功能
- 复用功能

7.2. GPIO 功能描述

GPIO 的每一个端口可以通过软件独立配置成下面状态：

- 输入浮空
- 输入上拉
- 输入下拉
- 模拟功能
- 开漏输出（上拉或下拉）
- 推挽输出（上拉或下拉）
- 复用功能（开漏或推挽、上拉或下拉）

表 8-1 GPIO 的模式配置表

MODE	OTYPE(i)	OSPEED(i)		PUPD(i)		IO configuration	
01	0	SPEED[1:0]		0	0	GP output	PP
	0			0	1	Reserved	
	0			1	0		
	0			1	1		
	1			0	0	GP output	OD
	1			0	1	GP output	OD+PU
	1			1	0	Reserved	
	1			1	1		
10	0	SPEED[1:0]		0	0	AF	PP
	0			0	1	Reserved	
	0			1	0		
	0			1	1		
	1			0	0	AF	OD
	1			0	1	AF	OD+PU
	1			1	0	Reserved	
	1			1	1		
00	x	x	x	0	0	Input	Floating
	x	x	x	0	1	Input	PU
	x	x	x	1	0	Input	PD
	x	x	x	1	1	Reserved	
11	x	x	x	0	0	Input/Output	Analog
	x	x	x	0	1	Reserved	
	x	x	x	1	0		
	x	x	x	1	1		

GP = generate-purpose, PP = push-pull, PU = pull-up, PD = pull-down, OD = open-drain, AF = alternate function.

7.2.1. 通用 IO（GPIO）

复位期间和刚复位后，复用功能未开启，I/O 端口被配置成浮空输入模式。DBG_CLK 这个 IO 可以被配置成上拉或下拉使能，依据客户自己的配置。

当作为输出配置时，写到输出数据寄存器上的值（GPIOx_ODR）输出到相应的 I/O 引脚。可以以推挽或开漏模式使用输出驱动器。

输入数据寄存器（GPIOx_IDR）在每个 AHB 时钟周期捕捉 I/O 引脚上的数据。

所有 GPIO 引脚有一个内部弱上拉和弱下拉，当配置为输入时，它们可以被激活也可以被断开。

7.2.2. 单独的位操作

当对 GPIOx_ODR 的个别位编程时，软件不需要禁止中断：在单次 AHB 写操作里，可以只更改一个或多个位。只需要通过对“置位/复位寄存器”（GPIOx_BSR）或“取反寄存器”（GPIOx_TGL）中想要更改的位写“1”来实现。没被选择的位将不被更改。

7.2.3. 复用功能（AF）

芯片 IO 引脚通过多路选择器连接到片内外设，每个 IO 上同一时刻只能选通一个复用功能。每个 IO 引脚有一个 4 输入的多路选择器连接到复用功能（AF0~AF3），通过配置 GPIOx_AFRH/L 选择输入功能。如果把端口配置成复用输出功能，则引脚和输出寄存器断开，并和片上外设的输出信号连接。如果软件把一个 GPIO 脚配置成复用输出功能，但是外设没有被激活，它的输出将不确定。

7.2.4. GPIO 锁定机制

锁定机制允许在 GPIO 控制寄存器 GPIOx_LCKR 上执行一串锁定程序，然后把 GPIO 的状态锁定，一旦 GPIO 状态被锁定，将不可改变，直到 CPU 复位。被锁定的寄存器有（GPIOx_MODER、GPIOx_OTYPER, GPIOx_OSPEEDR、GPIOx_PUPDR、GPIOx_AFRL、GPIOx_AFRH）。

锁定序列参考 GPIOx_LCKR 寄存器描述。

7.2.5. 输入配置

当 I/O 端口配置为输入时：

- 输出缓冲器被禁止
- 施密特触发输入被激活

- 根据输入配置（上拉、下拉或浮空）的不同，弱上拉和下拉电阻被连接
- 出现在 I/O 脚上的数据在每个 AHB 时钟被采样到输入数据寄存器
- 对输入数据寄存器的读访问可得到 I/O 状态

7.2.6. 输出配置

当 I/O 端口被配置为输出时：

- 输出缓冲器被激活
  - ✓ 开漏模式：输出寄存器上的“0”激活 N-MOS，而输出寄存器上的“1”将端口置于高阻态（P-MOS 从不被激活）。
  - ✓ 推挽模式：输出寄存器上的“0”激活 N-MOS，而输出寄存器上的“1”将激活 P-MOS。
- 施密特触发输入被激活
- 弱上拉和下拉电阻根据配置被激活
- 出现在 I/O 脚上的数据在每个 AHB 时钟被采样到输入数据寄存器
- 在开漏模式时，对输入数据寄存器的读访问可得到 I/O 状态
- 在推挽模式时，对输出数据寄存器的读访问得到最后一次写的值。

7.2.7. 模拟输入配置

当 I/O 端口被配置为模拟输入配置时：

- 输出缓冲器被禁止；
- 禁止施密特触发输入，实现了每个模拟 I/O 引脚上的零消耗。施密特触发输出值被强制为“0”；
- 弱上拉和下拉电阻被禁止；
- 读取输入数据寄存器时数值为“0”。

7.3. 寄存器

7.3.1. 寄存器基地址

Name	Base Address	Description
GPIOA	0x40020B00	GPIOA 基地址
GPIOB	0x40020C00	GPIOB 基地址
GPIOC	0x40020F00	GPIOE 基地址

7.3.2. 寄存器列表

Offset Address	Name	Description
0x0000	GPIOA_MODE	GPIOA 模式选择寄存器
0x0004	GPIOA_OTYPE	GPIOA 输出方式寄存器
0x0008	GPIOA_OSPEEDL	GPIOA 驱动能力配置寄存器 L
0x000c	GPIOA_OSPEEDH	GPIOA 驱动能力配置寄存器 H
0x0010	GPIOA_PUPD	GPIOA 上下拉配置寄存器
0x0014	GPIOA_IDR	GPIOA 输入寄存器
0x0018	GPIOA_ODR	GPIOA 输出寄存器
0x001c	GPIOA_BSR	GPIOA BIT Set 和 BIT Reset 寄存器
0x0020	GPIOA_LCK	GPIOA 加锁寄存器
0x0024	GPIOA_AFRL	GPIOA 多功能配置寄存器 L
0x0028	GPIOA_AFRH	GPIOA 多功能配置寄存器 H
0x002c	GPIOA_TGL	GPIOA 翻转寄存器
0x0030	GPIOA_IMK	GPIOA 中断寄存器

Offset Address	Name	Description
0x0000	GPIOB_MODE	GPIOB 模式选择寄存器
0x0004	GPIOB_OTYPE	GPIOB 输出方式寄存器
0x0008	GPIOB_OSPEEDL	GPIOB 驱动能力配置寄存器 L

0x000c	GPIOB_OSPEEDH	GPIOB 驱动能力配置寄存器 H
0x0010	GPIOB_PUPD	GPIOB 上下拉配置寄存器
0x0014	GPIOB_IDR	GPIOB 输入寄存器
0x0018	GPIOB_ODR	GPIOB 输出寄存器
0x001c	GPIOB_BSR	GPIOB BIT Set 和 BIT Reset 寄存器
0x0020	GPIOB_LCK	GPIOB 加锁寄存器
0x0024	GPIOB_AFRL	GPIOB 多功能配置寄存器 L
0x0028	GPIOB_AFRH	GPIOB 多功能配置寄存器 H
0x002c	GPIOB_TGL	GPIOB 翻转寄存器
0x0030	GPIOB_IMK	GPIOB 中断寄存器

Offset Address	Name	Description
0x0000	GPIOE_MODE	GPIOE 模式选择寄存器
0x0004	GPIOE_OTYPE	GPIOE 输出方式寄存器
0x0008	GPIOE_OSPEEDL	GPIOE 驱动能力配置寄存器 L
0x000c	GPIOE_OSPEEDH	GPIOE 驱动能力配置寄存器 H
0x0010	GPIOE_PUPD	GPIOE 上下拉配置寄存器
0x0014	GPIOE_IDR	GPIOE 输入寄存器
0x0018	GPIOE_ODR	GPIOE 输出寄存器
0x001c	GPIOE_BSR	GPIOE BIT Set 和 BIT Reset 寄存器
0x0020	GPIOE_LCK	GPIOE 加锁寄存器
0x0024	GPIOE_AFRL	GPIOE 多功能配置寄存器 L
0x0028	GPIOE_AFRH	GPIOE 多功能配置寄存器 H
0x002c	GPIOE_TGL	GPIOE 翻转寄存器
0x0030	GPIOE_IMK	GPIOE 中断寄存器

7.3.3. 寄存器详细定义

7.3.3.1. GPIOx_MODE

Bit(s)	Name	Description	R/W	Reset

31:30	MODER15	**GPIOx15 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
29:28	MODER14	**GPIOx14 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
27:26	MODER13	**GPIOx13 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
25:24	MODER12	**GPIOx12 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
23:22	MODER11	**GPIOx11 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
21:20	MODER10	**GPIOx10 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
19:18	MODER9	**GPIOx9 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
17:16	MODER8	**GPIOx8 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
15:14	MODER7	**GPIOx7 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
13:12	MODER6	**GPIOx6 模式选择**	RW	0x0

		0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入		
11:10	MODER5	**GPIOx5 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
9:8	MODER4	**GPIOx4 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
7:6	MODER3	**GPIOx3 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
5:4	MODER2	**GPIOx2 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
3:2	MODER1	**GPIOx1 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0
1:0	MODER0	**GPIOx0 模式选择** 0x0：输入模式 0x1：输出模式 0x2：AF 多功能模式 0x3：模拟输入	RW	0x0

7.3.3.2. GPIOx_OTYPE

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	OT15	**GPIOx15 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
14	OT14	**GPIOx14 输出的类型** 0x0：推挽输出	RW	0x0

		0x1：开漏输出		
13	OT13	**GPIOx13 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
12	OT12	**GPIOx12 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
11	OT11	**GPIOx11 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
10	OT10	**GPIOx10 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
9	OT9	**GPIOx9 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
8	OT8	**GPIOx8 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
7	OT7	**GPIOx7 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
6	OT6	**GPIOx6 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
5	OT5	**GPIOx5 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
4	OT4	**GPIOx4 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
3	OT3	**GPIOx3 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
2	OT2	**GPIOx2 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
1	OT1	**GPIOx1 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0
0	OT0	**GPIOx0 输出的类型** 0x0：推挽输出 0x1：开漏输出	RW	0x0

7.3.3.3. GPIOx_OSPEEDL

Bit(s)	Name	Description	R/W	Reset
31:28	OSPEED7	**GPIOx7 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
27:24	OSPEED6	**GPIOx6 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
23:20	OSPEED5	**GPIOx5 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
19:16	OSPEED4	**GPIOx4 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
15:12	OSPEED3	**GPIOx3 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
11:8	OSPEED2	**GPIOx2 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
7:4	OSPEED1	**GPIOx1 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
3:0	OSPEED0	**GPIOx0 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0

7.3.3.4. GPIOx_OSPEEDH

Bit(s)	Name	Description	R/W	Reset
31:28	OSPEED15	**GPIOx15 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
27:24	OSPEED14	**GPIOx14 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
23:20	OSPEED13	**GPIOx13 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
19:16	OSPEED12	**GPIOx12 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
15:12	OSPEED11	**GPIOx11 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
11:8	OSPEED10	**GPIOx10 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
7:4	OSPEED9	**GPIOx9 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0
3:0	OSPEED8	**GPIOx8 驱动能力配置** 0x0：low speed 0x1：speed1 0x2：speed2 0x3：speed3	RW	0x0

7.3.3.5. GOIOx_PUPD

Bit(s)	Name	Description	R/W	Reset
31	PD15	**GPIOx15 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
30	PD14	**GPIOx14 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
29	PD13	**GPIOx13 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
28	PD12	**GPIOx12 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
27	PD11	**GPIOx11 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
26	PD10	**GPIOx10 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
25	PD9	**GPIOx9 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
24	PD8	**GPIOx8 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
23	PD7	**GPIOx7 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
22	PD6	**GPIOx6 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
21	PD5	**GPIOx5 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
20	PD4	**GPIOx4 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
19	PD3	**GPIOx3 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
18	PD2	**GPIOx2 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
17	PD1	**GPIOx1 下拉电阻（10K）选择**	RW	0x0

		0x0：关闭 0x1：打开		
16	PD0	**GPIOx0 下拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
15	PU15	**GPIOx15 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
14	PU14	**GPIOx14 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
13	PU13	**GPIOx13 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
12	PU12	**GPIOx12 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
11	PU11	**GPIOx11 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
10	PU10	**GPIOx10 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
9	PU9	**GPIOx9 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
8	PU8	**GPIOx8 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
7	PU7	**GPIOx7 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
6	PU6	**GPIOx6 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
5	PU5	**GPIOx5 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
4	PU4	**GPIOx4 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
3	PU3	**GPIOx3 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
2	PU2	**GPIOx2 上拉电阻（10K）选择**	RW	0x0

		0x0：关闭 0x1：打开		
1	PU1	**GPIOx1 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0
0	PU0	**GPIOx0 上拉电阻（10K）选择** 0x0：关闭 0x1：打开	RW	0x0

7.3.3.6. GPIOx_IDR

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	IDR15	**GPIOx15 输入数据读取**	RO	0x0
14	IDR14	**GPIOx14 输入数据读取**	RO	0x0
13	IDR13	**GPIOx13 输入数据读取**	RO	0x0
12	IDR12	**GPIOx12 输入数据读取**	RO	0x0
11	IDR11	**GPIOx11 输入数据读取**	RO	0x0
10	IDR10	**GPIOx10 输入数据读取**	RO	0x0
9	IDR9	**GPIOx9 输入数据读取**	RO	0x0
8	IDR8	**GPIOx8 输入数据读取**	RO	0x0
7	IDR7	**GPIOx7 输入数据读取**	RO	0x0
6	IDR6	**GPIOx6 输入数据读取**	RO	0x0
5	IDR5	**GPIOx5 输入数据读取**	RO	0x0
4	IDR4	**GPIOx4 输入数据读取**	RO	0x0
3	IDR3	**GPIOx3 输入数据读取**	RO	0x0
2	IDR2	**GPIOx2 输入数据读取**	RO	0x0
1	IDR1	**GPIOx1 输入数据读取**	RO	0x0
0	IDR0	**GPIOx0 输入数据读取**	RO	0x0

7.3.3.7. GPIOx_ODR

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	ODR15	**GPIOx15 输出寄存器**	RW	0x0
14	ODR14	**GPIOx14 输出寄存器**	RW	0x0
13	ODR13	**GPIOx13 输出寄存器**	RW	0x0
12	ODR12	**GPIOx12 输出寄存器**	RW	0x0
11	ODR11	**GPIOx11 输出寄存器**	RW	0x0
10	ODR10	**GPIOx10 输出寄存器**	RW	0x0
9	ODR9	**GPIOx9 输出寄存器**	RW	0x0
8	ODR8	**GPIOx8 输出寄存器**	RW	0x0

7	ODR7	**GPIOx7 输出寄存器**	RW	0x0
6	ODR6	**GPIOx6 输出寄存器**	RW	0x0
5	ODR5	**GPIOx5 输出寄存器**	RW	0x0
4	ODR4	**GPIOx4 输出寄存器**	RW	0x0
3	ODR3	**GPIOx3 输出寄存器**	RW	0x0
2	ODR2	**GPIOx2 输出寄存器**	RW	0x0
1	ODR1	**GPIOx1 输出寄存器**	RW	0x0
0	ODR0	**GPIOx0 输出寄存器**	RW	0x0

7.3.3.8. GPIOx_BSR

Bit(s)	Name	Description	R/W	Reset
31	BR15	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
30	BR14	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
29	BR13	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
28	BR12	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
27	BR11	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
26	BR10	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
25	BR9	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
24	BR8	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
23	BR7	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
22	BR6	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
21	BR5	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
20	BR4	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
19	BR3	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
18	BR2	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
17	BR1	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0
16	BR0	**Bit Reset 配置** 写 1 有效，有效时，GPIO 输出低电平	WO	0x0

15	BS15	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
14	BS14	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
13	BS13	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
12	BS12	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
11	BS11	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
10	BS10	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
9	BS9	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
8	BS8	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
7	BS7	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
6	BS6	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
5	BS5	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
4	BS4	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
3	BS3	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
2	BS2	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
1	BS1	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0
0	BS0	**Bit Set 配置** 写 1 有效，有效时，GPIO 输出高电平	WO	0x0

7.3.3.9. GPIOx_LCK

Bit(s)	Name	Description	R/W	Reset
31:17	Reserved	-	-	-
16	LCKEN	**GPIO 加锁有效使能** 0x0：关闭 0x1：打开	RW	0x0
15	LCK15	**GPIOx15 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
14	LCK14	**GPIOx14 加锁使能**	RW	0x0

		0x0：关闭 0x1：打开		
13	LCK13	**GPIOx13 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
12	LCK12	**GPIOx12 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
11	LCK11	**GPIOx11 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
10	LCK10	**GPIOx10 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
9	LCK9	**GPIOx9 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
8	LCK8	**GPIOx8 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
7	LCK7	**GPIOx7 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
6	LCK6	**GPIOx6 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
5	LCK5	**GPIOx5 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
4	LCK4	**GPIOx4 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
3	LCK3	**GPIOx3 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
2	LCK2	**GPIOx2 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
1	LCK1	**GPIOx1 加锁使能** 0x0：关闭 0x1：打开	RW	0x0
0	LCK0	**GPIOx0 加锁使能** 0x0：关闭 0x1：打开	RW	0x0

对 GPIO 进行加锁，需要进行以下操作（以加锁 GPIOA15 为示例）：

①GPIOA_LCK = (1<<16) | (1<<15);

②GPIOA_LCK = (0<<16) | (1<<15);

③GPIOA_LCK = (1<<16) | (1<<15);

④读取寄存器 GPIOA_LCK;

⑤读取寄存器 GPIOA_LCK，确认 BIT16 是否为 1。

7.3.3.10. GPIOx_AFRL

Bit(s)	Name	Description	R/W	Reset
31:28	AFR7	**GPIOx7 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
27:24	AFR6	**GPIOx6 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
23:20	AFR5	**GPIOx5 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
19:16	AFR4	**GPIOx4 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
15:12	AFR3	**GPIOx3 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
11:8	AFR2	**GPIOx2 的 AF 多功能配置** 0x0：AF0 0x1：AF1	RW	0x0

		0x2：AF2 0x3：AF3 Other：保留		
7:4	AFR1	**GPIOx1 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
3:0	AFR0	**GPIOx0 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0

7.3.3.11. GPIOx_AFRH

Bit(s)	Name	Description	R/W	Reset
31:28	AFR15	**GPIOx15 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
27:24	AFR14	**GPIOx14 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
23:20	AFR13	**GPIOx13 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
19:16	AFR12	**GPIOx12 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
15:12	AFR11	**GPIOx11 的 AF 多功能配置**	RW	0x0

		0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留		
11:8	AFR10	**GPIOx10 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
7:4	AFR9	**GPIOx9 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0
3:0	AFR8	**GPIOx8 的 AF 多功能配置** 0x0：AF0 0x1：AF1 0x2：AF2 0x3：AF3 Other：保留	RW	0x0

7.3.3.12. GPIOx_TGL

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	TG15	**GPIOx15 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
14	TG14	**GPIOx14 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
13	TG13	**GPIOx13 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
12	TG12	**GPIOx12 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
11	TG11	**GPIOx11 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
10	TG10	**GPIOx10 翻转配置**	WO	0x0

		0x0：无效 0x1：输出翻转		
9	TG9	**GPIOx9 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
8	TG8	**GPIOx8 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
7	TG7	**GPIOx7 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
6	TG6	**GPIOx6 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
5	TG5	**GPIOx5 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
4	TG4	**GPIOx4 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
3	TG3	**GPIOx3 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
2	TG2	**GPIOx2 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
1	TG1	**GPIOx1 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0
0	TG0	**GPIOx0 翻转配置** 0x0：无效 0x1：输出翻转	WO	0x0

7.3.3.13. GPIOx_IMK

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	IMK15	**GPIOx15 中断使能** 0x0：关闭 0x1：打开	RW	0x0
14	IMK14	**GPIOx14 中断使能** 0x0：关闭 0x1：打开	RW	0x0
13	IMK13	**GPIOx13 中断使能**	RW	0x0

		0x0：关闭 0x1：打开		
12	IMK12	**GPIOx12 中断使能** 0x0：关闭 0x1：打开	RW	0x0
11	IMK11	**GPIOx11 翻转配置** 0x0：无效 0x1：输出翻转	RW	0x0
10	IMK10	**GPIOx10 中断使能** 0x0：关闭 0x1：打开	RW	0x0
9	IMK9	**GPIOx9 中断使能** 0x0：关闭 0x1：打开	RW	0x0
8	IMK8	**GPIOx8 中断使能** 0x0：关闭 0x1：打开	RW	0x0
7	IMK7	**GPIOx7 中断使能** 0x0：关闭 0x1：打开	RW	0x0
6	IMK6	**GPIOx6 中断使能** 0x0：关闭 0x1：打开	RW	0x0
5	IMK5	**GPIOx5 中断使能** 0x0：关闭 0x1：打开	RW	0x0
4	IMK4	**GPIOx4 中断使能** 0x0：关闭 0x1：打开	RW	0x0
3	IMK3	**GPIOx3 中断使能** 0x0：关闭 0x1：打开	RW	0x0
2	IMK2	**GPIOx2 中断使能** 0x0：关闭 0x1：打开	RW	0x0
1	IMK1	**GPIOx1 中断使能** 0x0：关闭 0x1：打开	RW	0x0
0	IMK0	**GPIOx0 中断使能** 0x0：关闭 0x1：打开	RW	0x0

8. 通信接口外设 CSI

8.1. SPI_I2C

该模块实现了 SPI/IIC 共用填一个控制器的功能，当使用 SPI 时，IIC 功能不可使用，同一个时刻，只能使用一种协议。

8.1.1. SPI 功能描述

1、支持主机模式和从机模式

2、支持一下四种采样方式：

mode 0：时钟 idel 为 0，上升沿采样，下降沿出数据

mode 1：时钟 idel 为 0，下降沿采样，上升沿出数据

mode 2：时钟 idle 为 1，下降沿采样，上升沿出数据

mode 3：时钟 idel 为 1，上升沿采样，下降沿出数据

3、支持正常模式、3 线模式、双线模式和四线模式。

正常模式：CLK、CS、IO0(MOSI)、IO1(MISO)

3 线模式：CLK、CS、IO0(接收和发送都通过 IO0)

双线模式：CLK、CS、O0、IO1(接收和发送都通过 IO0、IO1)

四线模式：CLK、CS、IO0、IO1、IO2、IO3(接收和发送都通过 IO0、IO1、IO2、IO3)

4、数据大小支持 1bit 到 32bit，可配置，但是模式不同时有限制。在正常模式和 3 线模式下，数据大小可以是 1 到 32 位可配置；在双线模式下，数据大小必须能被 2 整除；在四线模式下，数据大小必须能被 4 整除。

1、可以选择在一个数据帧中先发送低位数据或高位数据。当高位在前时，IO3 传输最高位，IO2 传输第二高位，IO1 传输第三高位，IO0 传输第四个高位。当低位优先时，IO3 传输最低位，IO2 传输次低位，IO1 传输第三个低位，IO0 传输第四个低位。

6、支持 DMA 功能。

8.1.2. I2C 功能描述

1、支持主机模式和从机模式
2、master 支持时钟同步和仲裁
3、slave 支持在发送数据没有准备好或者接收 buffer 满的时候拉低 SCL
4、slave 支持 7bit 地址或者 10bit 地址
5、支持 DMA

8.1.3. SPI 时序图

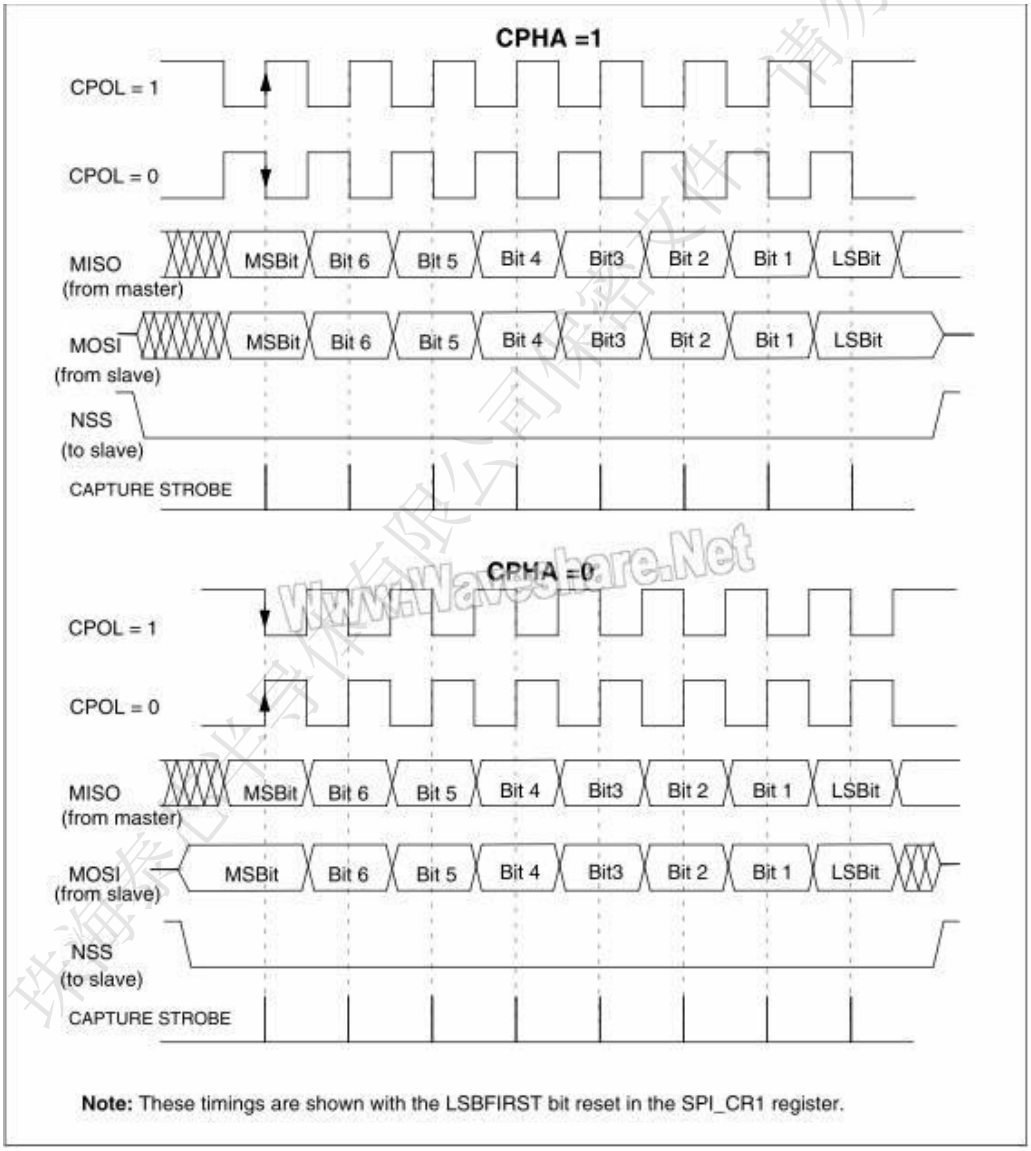


图 9-1 正常模式下时序图

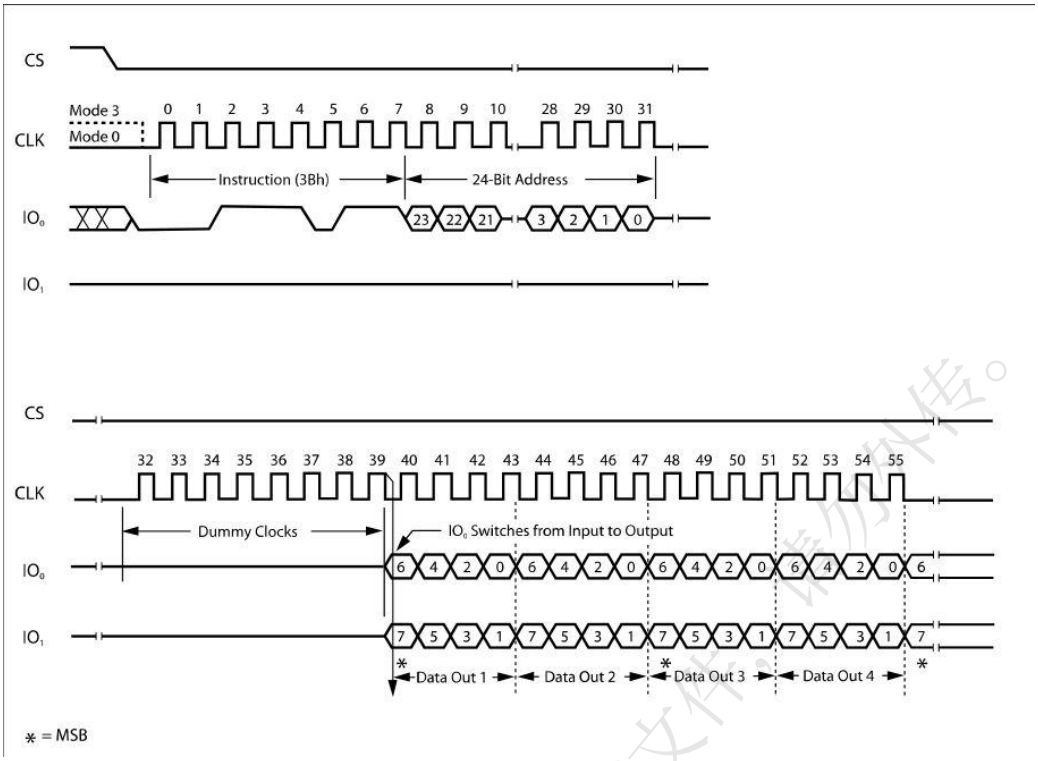


图 9-2 双线模式下时序图

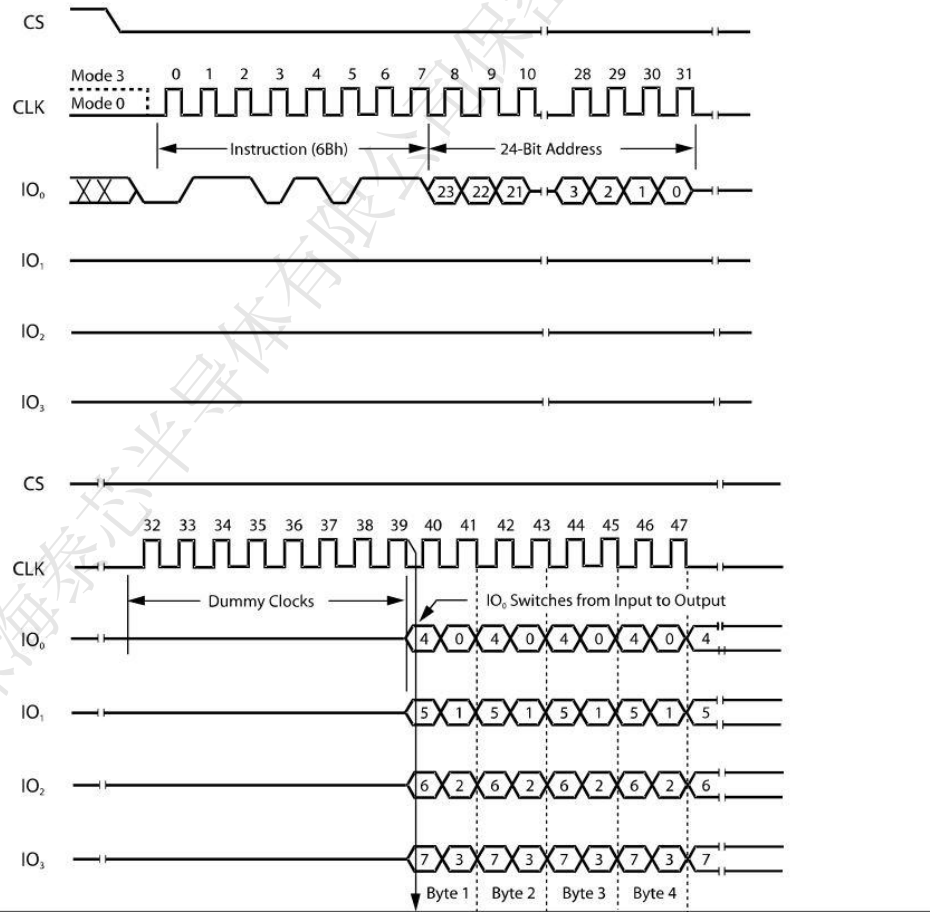


图 9-3 四线模式下时序图

8.1.4. IO MAPPING

SPI_I2C0:

SPI0_NSS：PA0(AF2)

SPI0_SCK/I2C0_SCL/MOSI：PA1(AF2)、PA10(AF1)、PB3(AF3)

SPI0_IO0/I2C0_SDA/MISO ：PA2(AF2)、PA11(AF1)、PE2(AF0)

SPI0_IO1：PA3(AF2)、PA12(AF1)、PB4(AF3)

SPI0_IO2：PA4(AF2)

SPI0_IO3：PA5(AF2)

SPI_I2C1:

SPI1_NSS：PB5(AF1)、PA6(AF2)

SPI1_SCK/I2C1_SCL/MOSI：PB6(AF1)、PA7(AF2)

SPI1_IO0/I2C1_SDA/MISO：PB7(AF1)、PA8(AF2)

SPI1_IO1：PB8(AF1)、PA9(AF2)

SPI1_IO2：PB9(AF1)、PA10(AF2)

SPI1_IO3：PB10(AF1)、PA11(AF2)

8.2. 寄存器

8.2.1. 寄存器基地址

Name	Base Address	Description
SPI_I2C0	0x40004400	SPI0/I2C0 基地址
SPI_I2C1	0x40004500	SPI1/I2C1 基地址

8.2.2. 寄存器列表

Offset Address	Name	Description
0x0000	SPIx_CON0/I2Cx_CON0	SPI/I2C 控制寄存器 0
0x0004	SPIx_CON1/I2Cx_CON1	SPI/I2C 控制寄存器 1
0x0008	SPIx_DATA/I2Cx_DATA	SPI/I2C 数据寄存器

0x000c	SPIx_BAUD/I2Cx_BAUD	SPI/I2C 波特率寄存器
0x0010	SPIx_DAMLEN/I2Cx_DAMLEN	SPI/I2C DMA 长度寄存器
0x0014	SPIx_DMACNT/I2Cx_DMACNT	SPI/I2C DMA 计数寄存器
0x0018	SPIx_DMASTART/I2Cx_DMASTART	SPI/I2C DMA 触发寄存器
0x001c	SPIx_STA/I2Cx_STA	SPI/I2C 状态寄存器

8.2.3. 寄存器详细说明

8.2.3.1. SPIx_CON0

Bit(s)	Name	Description	R/W	Reset
31:20	Reserved	-	-	-
19:14	SPI_FRAME_SIZE	**SPI 帧大小配置** 0x00：无效 0x01：每一帧发送 1bit 0x02：每一帧发送 2bit ... 0x20：每一帧发送 32bit Other：无效	RW	0x0
13	NSS_POS_IE	**SPI 从机模式，SPI_NSS 引脚中断使能** 0x0：关闭 0x1：打开	RW	0x0
12	SPI_NSS_EN	**SPI 从机模式，SPI_NSS 引脚使能** 0x0：关闭 0x1：打开	RW	0x0
11	SPI_NSS	**SPI NSS 引脚控制输出** 此位仅在 SPI 主模式下有效 0x0：输出低电平 0x1：输出高电平	RW	0x0
10:8	RXSEL	**SPI 主机采样延迟的配置** 0x0：no delay 0x1：delay 1 cycle 0x2：delay 2 cycle ... 0x7：delay 7 cycle	RW	0x0
7	SLAVE_SYNC_EN	**SPI SLAVE 模式下，输入数据是否需要同步的选择位** 0x0：关闭 0x1：打开	RW	0x0
6	MASTER_SYNC_EN	**SPI MASTER 模式下，输入数据是否需要同步的选择位**	RW	0x0

		0x0：关闭 0x1：打开		
5	Reserved	-	-	-
4	LSBFE	**SPI 传输从低位使能** 0x0：高位优先 0x1：低位优先	RW	0x0
3:2	WIREMODE	**SPI 通信线模式选择** 0x0：正常模式 0x1：3 线模式 0x2：双线模式 0x3：四线模式	RW	0x0
1:0	SPIMODE	**SPI 模式选择** 0x0：时钟 idel 为 0，上升沿采样，下降沿出数据 0x1：时钟 idel 为 0，下降沿采样，上升沿出数据 0x2：时钟 idle 为 1，下降沿采样，上升沿出数据 0x3：时钟 idel 为 1，上升沿采样，下降沿出数据	RW	0x0

8.2.3.2. IICx_CON0

Bit(s)	Name	Description	R/W	Reset
31:23	Reserved	-	-	-
22	RX_NACK_IE	**接收到 NACK 中断使能** 0x0：关闭 0x1：打开	RW	0x0
21	I2C_AL_IE	**主机仲裁丢失中断使能** 0x0：关闭 0x1：打开	RW	0x0
20	STOP_IE	**检测到线上有 STOP 信号中断使能，不管是从机主机都会起来** 0x0：关闭 0x1：打开	RW	0x0
19	ADR_MATCH_IE	**SLAVE 地址匹配中断使能** 0x0：关闭 0x1：打开	RW	0x0
18:14	I2C_FILTER_CNT	I2C 每（I2C_FILTER_CNT +1）个 I2C 模块时钟，采样一次 SCL 和 SDA，用于滤除一定宽度的线路毛刺。I2C_FILTER_CNT 配置越大，虑掉的毛刺宽度越大。	RW	0x0
13	BROADCAST_IE	**广播中断使能** 0x0：关闭 0x1：打开	RW	0x0
12	BROADCAST_EN	**I2C 作为 slave 时候，是否使能接收广播地址** 0x0：忽略广播地址 0x1：接收到广播地址时，回应 ACK，并且将 BROADCAST_PEND 置起来，可以产生中断	RW	0x0

11:2	SLAVE_ADR	**I2C 作为 SLAVE 时候，SLAVE 地址**	RW	0x0
1	TX_NACK	**I2C 在接收数据的时候，回应 NACK 还是 ACK 选择位** 0x0：ACK 0x1：NACK	RW	0x0
0	SLAVE_ADR_WIDTH	**I2C 作为 SLAVE 时候，SLAVE 地址宽度** 0x0：7bit 0x1：10bit	RW	0x0

8.2.3.3. SPIx_CON1 / IICx_CON1

Bit(s)	Name	Description	R/W	Reset
31:10	Reserved		-	-
9	DMA_IE	**DMA 完成中断使能** 0x0：关闭 0x1：打开	RW	0x0
8	BUF_OV_IE	**buffer 溢出，有数据丢失了中断使能** 0x0：关闭 0x1：打开	RW	0x0
7	RBUF_NEMPTY_IE	**接收 buffer 不空中断使能** 0x0：关闭 0x1：打开	RW	0x0
6	TBUF_NFULL_IE	**发送 buffer 不满中断使能** 0x0：关闭 0x1：打开	RW	0x0
5	SSP_IE	**I2C 接口：** I2Cmaster 接收或者发送完成一帧数据（START + WRITE/READ(8bit+ack) + STOP）中断使能 I2C slave 接收或者发送完成一帧数据(8bit+ack)中断使能； 0x0：关闭 0x1：打开	RW	0x0
4	DMA_EN	**发送 DMA 使能** 0x0：关闭 0x1：打开 注：如果 SSP_TX_EX_EN==0，表示 DMA 发送；如果 SSP_TX_EX_EN==1，表示 DMA 接收；	RW	0x0
3	SSP_TX_RX_EN	**接口发送使能** 0x0：RX_EN 0x1：TX_EN	RW	0x0
2	SLAVE	**SLAVE 模式使能** 0x0：master mode 0x1：slave mode	RW	0x0
1	SPI_I2C_SEL	**SPI 和 I2C 选择位**	RW	0x0

		0x0：SPI 接口 0x1：I2C 接口		
0	SSP_EN	**模块使能** 0x0：关闭 0x1：打开	RW	0x0

8.2.3.4. SPIx_DATA / IICx_DATA

Bit(s)	Name	Description	R/W	Reset
31:0	SSP_CMD_DATA	**SPI 接口：** Write：将想要发送的数据写入这个寄存器，触发 SPI 发送； Read：读这个寄存器，获取收到的数据。 **I2C 接口：** Write： [7:0] 将想要发送的数据写入这 8bit； [8] START 使能位，在将这 byte 数据发出去前，先插入一个 start bit（只在 I2C_MASTER 模式有效）； [9] STOP 使能位，在将这 byte 数据发出去后，紧跟一个 stop bit（只在 I2C_MASTER 模式有效）。 Read： 读这个寄存器，获取收到的数据（bit7~bit0） [31:10] 没有使用	RW	0x0

8.2.3.5. SPIx_BAUD / IICx_BAUD

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15:0	BAUD	**MASTER 模式** ①SPI 波特率=apb0_clock/(2*(BAUD+1)) ②I2C 波特率= apb0_clock/(4*(BAUD+1) **SLAVE 模式** ①SPI 没有用 ②I2C 用于定义从机准备好发送数据，再 delay(BAUD+1)个 apb0_clock 周期后才释放 SCL。	RW	0x0

8.2.3.6. SPIx_DMALEN / IICx_DMALEN

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	SPI_DMA_LEN	**配置接收和发送 DMA 数据的长度** 接收单位 byte	RW	0x0

8.2.3.7. SPIx_DMACNT / IICx_DMACNT

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	SPI_DMA_CNT	接收和发送 DMA 模式:实际接收并且保存到 SRAM 的数据 byte 长度 在 master 模式，RX_DMA_EN 为 0 时候，接收数据：表示实际接收并且已经被 CPU 读走的数据 byte 长度； 注：在 RX_DMA_EN 上升沿，TX_DMA_EN 上升沿，以及 MSTER 读上升沿的时候，被清零。	RO	0x0

8.2.3.8. SPIx_DMASTADR / IICx_DMASTADR

Bit(s)	Name	Description	R/W	Reset
31:13	Reserved	-	-	-
12:0	SPI_DMA_STADR	DMA 地址，一帧数据是 32bit 时候，需要 32bit 对齐，其他时候可以任意配置	RW	0x0

8.2.3.9. SPIx_STA / IICx_STA

Bit(s)	Name	Description	R/W	Reset
31:28	Reserved	-	-	-
27	SLV_ADDRED	**从机模式，从机被寻址标志位** 0x0：从机没有被寻址 0x1：从机已经被寻址	RO	0x0
26:24	STATE	**I2C 状态** 0x0：IDLE 空闲空闲 0x1：START 发送 start 已经接收到 start，等待 SCL 变 0 0x2：TX 发送 1byte 数据发送 1byte 数据 0x3：RX 接收 1byte 数据接收 1byte 数据 0x4：STOP 发送 stop no use 0x5：ADR0 no use 等待接收 1byte 地址 0x6：ADR1 no use 等待接收 2byte 地址	RO	0x0

		0x7：no use		
23:20	Reserved	-	-	-
19	CLR_BUF_CNT	Write 1 清零 BUF_CNT Write 0 没有任何作用 Read：总是返回 0	WO	0x0
18:16	BUF_CNT	buffer 里面有多少 byte 有效数据，CPU 每读(RX_EN)/写(TX_EN)一次 CMD_DATA 寄存器，减/加一个帧数据宽度，8bit 数据减 1，16bit 数据减 2，24bit 和 32bit 数据减 4	RO	0x0
15	MASTER_RX_BUSY	**在 MASTER 读从机模式，指示软件配置的连续读取 DMA_LEN byte 数据是否完成** 0x0：已经读完 0x1：还没有读完	RO	0x0
14	I2C_RX_NACK	**I2C 主机或者从机，在发送数据的时候，在第 9bit 握手阶段，检测到 ACK 还是 NACK** 0x0：ACK 0x1：NACK	RO	0x0
13	I2C_SLAVE_RW	**IIC 从机，在地址阶段，接收到的主机发过来的读写标志** 0x1：主机将读从机 0x0：主机将写从机	RO	0x0
12	I2C_BUS_BUSY	**IIC 线路是否 busy 指示位** 0x0：线路上一直没有出现 START，或者出现 START 后又出现了 STOP，线路空闲 0x1：检测到线路上出现了 START,并且一直没有出现 STOP，线路忙 注：只能硬件清	RO	0x0
11	SPI_SLAVE_CS	**SPI SLAVE，CS 的状态**	RO	0x01
10	SSP_BUSY	**SPI 或者 IIC 正忙** 0x0：MASTER 空闲 SLAVE 没有在发送数据，接收的时候不使用 0x1：MASTER 正在接收或者发送一帧数据 SLAVE 正在发送一帧数据，接收的时候不使用 注：硬件清，或者关掉 SPI 和 IIC(CON1[0]==0)，或者在 slave 时候，CLR_BUF_CNT 会同时清掉 SSP_BUSY	RO	0x0
9	AL_PEND	**IIC 主机，检测到仲裁丢失** 0x0：没有仲裁丢失 0x1：仲裁丢失 注：只能软件清，必须清掉 IIC 才能正常工作	RC	0x0
8	STOP_PEND	**IIC 接口，检测到线路上产生了 STOP 位** 0x0：没有检测到 STOP 位。 0x1：检测到 STOP 位。 注：只能软件清	RC	0x0
7	ADR_MTCH_PEND	**IIC 从机，接收到主机发送过来的对的从机地址** 0x1：从机地址匹配	RC	0x0

		0x0：从机地址不匹配 注：只能软件清		
6	I2C_BROADCAST_PEND	**IIC 作为 SLAVE 时候，检测到广播地址标志位** 0x0：没有检测到广播地址 0x1：检测到广播地址 注：只能软件清	RC	0x0
5	SPI_NSS_POS	**检测到 SPI NSS 引脚上升沿** 0x0：没有检测到上升沿 0x1：检测到上升沿 注：只能软件清	RC	0x0
4	DMA_PEND	**发送 DMA 或者接收 DMA 完成标志** 0x0：DMA 没有完成 0x1：DMA 已经完成 注：只能软件清	RC	0x0
3	BUF_OV	**buffer 溢出，有数据丢失了** 0x0：buffer 没有溢出 0x1：在 buffer 已经是满状态（buffer 容量是 5byte，满并不一定是 5byte，而是空间容不下一帧数据了），又有数据接收到。丢弃后来的数据. 注：只能软件清	RC	0x0
2	BUF_EMPTY	**buffer 空标志** 0x0：不空 0x1：空	RO	0x01
1	BUF_FULL	**buffer 满标志位** 0x0：不满 0x1：满	RO	0x0
0x0	SSP_DONE	**完成标志** 0x0：没有完成 0x1：SPI 接口 SPI 已经完成一帧（8bit/16bit/24bit/32bit）数据的接收或者发送 I2C 接口 I2C MASTER 已经完成了一帧数据的接收或者发送（START（可选） + WRITE/READ（8bit+ack） + STOP（可选））I2C SLAVE 已经完成了一帧数据的接收或者发送（8bit+ack） 注：只能软件清	RC	0x0

8.2.4. 使用说明

8.2.4.1. SPI 模块使用

SPI MASTER 初始化:

1.IO mapping 配置，将数据通路打通

2.配置 BAUD

3.配置 CON0

4.配置配置 CON1

5.SPI MASTER 使能：SPIx->CON1 | = 0x01；

SPI MASTER 非 DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2. 若初始化时 SPI_NSS_EN, 需要将 NSS 拉低： SPIx->CON0 &= ~(1<<11);//nssnegedge

2.判断发送 buffer 是否 full，没有 full 就可以往 CMD_DATA 填数据

3.将所有想发的数据都填入 CMD_DATA 后，等待发送 buffer 空，以及 ssp_busy 为 0。

4.若初始化时 SPI_NSS_EN,需要将 NSS 拉高:SPIx->CON0 |= (1<<11);//nssposedge

5.结束。

SPI MASTER DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2. 若初始化时 SPI_NSS_EN, 需要将 NSS 拉低： SPIx->CON0 &= ~(1<<11);//nssnegedge

2.配置 DMA_STADR,DMA_LEN

3.使能 DMA:SPIx->CON1 |= (1<<4);// DMA EN\

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.若初始化时 SPI_NSS_EN,需要将 NSS 拉高:SPIx->CON0 |= (1<<11);//nssposedge

6.结束。

SPI MASTER 非 DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2. 若初始化时 SPI_NSS_EN, 需要将 NSS 拉低： SPIx->CON0 &= ~(1<<11);//nssnegedge

2.配置 DMA_LEN 为想接收的数据的 byte 数

3.写 CMD_DATA 触发接收开始。

4.查询接收 buffer 为非空。通过读 DMA_DATA 取走数据。

5.重复 4，直到读走 DMA_LEN 的数据

6.若初始化时 SPI_NSS_EN,需要将 NSS 拉高:SPIx->CON0 |= (1<<11);//nssposedge

7.结束。

SPI MASTER DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2. 若初始化时 SPI_NSS_EN, 需要将 NSS 拉低： SPIx->CON0 &= ~(1<<11);//nssnegedge

2.配置 DMA_LEN 为想接收的数据的 byte 数

3.使能 DMA:SPIx->CON1 |= (1<<4);//

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.若初始化时 SPI_NSS_EN,需要将 NSS 拉高:SPIx->CON0 |= (1<<11);//nssposedge

6.结束。

SPI SLAVE 初始化:

1.IO mapping 配置，将数据通路打通

2.配置 CON0

3.配置 CON1

4.SPI SLAVE 使能：SPIx->CON1 | = 0x05;

SPI SLAVE 非 DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2.判断发送 buffer 是否 full，没有 full 就可以往 CMD_DATA 填数据

3.将所有想发的数据都填入 CMD_DATA 后，等待发送 buffer 空，以及 ssp_busy 为 0。

4.结束。

SPI SLAVE DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2.配置 DMA_STADR,DMA_LEN

3.使能 DMA:SPIx->CON1 |= (1<<4);// DMA EN\

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.结束。

SPI SLAVE 非 DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2.查询接收 buffer 为非空。通过读 DMA_DATA 取走数据。

3.重复 2，直到读走所有需要的数据

4.结束。

SPI SLAVE DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2.配置 DMA_LEN 为想接收的数据的 byte 数

3.使能 DMA:SPIx->CON1 |= (1<<4);

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.结束。

8.2.4.2. I2C 模块使用说明

I2C MASTER 初始化:

1.IO mapping 配置，将数据通路打通

2.配置 BAUD

3.配置 CON0

4.配置配置 CON1

5.SPI MASTER 使能：SPIx->CON1 | = 0x03;

I2C MASTER 非 DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2.判断发送 buffer 是否 full，没有 full 就可以往 CMD_DATA 填数据

3.将所有想发的数据都填入 CMD_DATA 后，等待发送 buffer 空，以及 ssp_busy 为 0。

4.结束。

I2C MASTER DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2.配置 DMA_STADR,DMA_LEN

3.使能 DMA:SPIx->CON1 |= (1<<4);

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.结束。

I2C MASTER 非 DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2.配置 DMA_LEN 为想接收的数据的 byte 数

3.写 CMD_DATA 触发接收开始。

4.查询接收 buffer 为非空。通过读 DMA_DATA 取走数据。

5.重复 4，直到读走 DMA_LEN 的数据

6.结束。

I2C MASTER DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2.配置 DMA_LEN 为想接收的数据的 byte 数

3.使能 DMA:SPIx->CON1 |= (1<<4);

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.结束。

I2C SLAVE 初始化:

1.IO mapping 配置，将数据通路打通

2.配置 CON0

3.配置 CON1

4.SPI SLAVE 使能：SPIx->CON1 | = 0x07；

I2C SLAVE 非 DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2.判断发送 buffer 是否 full，没有 full 就可以往 CMD_DATA 填数据

3.将所有想发的数据都填入 CMD_DATA 后，等待发送 buffer 空，以及 ssp_busy 为 0。

4.结束。

I2C SLAVE DMA 发送:

1.SSP_TX_EN 使能：SPIx->CON1 |= (1<<3); //tx enable

2.配置 DMA_STADR,DMA_LEN

3.使能 DMA:SPIx->CON1 |= (1<<4);// DMA EN\

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.结束。

I2C SLAVE 非 DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2.查询接收 buffer 为非空。通过读 DMA_DATA 取走数据。

3.重复 2，直到读走所有需要的数据

4.结束。

I2C SLAVE DMA 接收:

1.SSP_RX_EN 使能：SPIx->CON1 &= ~(1<<3); //rx enable

2.配置 DMA_LEN 为想接收的数据的 byte 数

3.使能 DMA:SPIx->CON1 |= (1<<4);// DMA EN\

4.等待 DMA 结束：while((SPIx->STA & (1<<4))==0);//WAIT DMA PEND

5.结束。

9. UART

9.1. 概述

UART0：

- 支持 8bit 数据和 9bit 数据模式
- 支持奇偶校验，奇校验/偶校验可选
- 支持检测系统更新数据串（0x70->0x61->0x73->0x53->0xf5->0x1e->0xf4->0xec）
- 具有 4 帧数据的接收缓存，一帧数据的发送缓存
- 硬件检测接收 time out，time out 长度可以配置，配置范围：1~65536 比特率时间。
- GPIO 映射：

  UART0 RX：PB8(fun2)、PE1(fun3)、PA7(fun3)

  UART0 TX：PB7(fun2)、PE0(fun3)、PA6(fun3)

UART1：

- 支持 8bit 数据和 9bit 数据模式
- 支持奇偶校验，奇校验/偶校验可选

- 具有 4 帧数据的接收缓存,一帧数据的发送缓存
- 支持接收和发送 DMA
- 支持 RS485 模式
- 硬件检测接收 time out，time out 长度可以配置，配置范围：1~65536 比特率时间。
- GPIO 映射：

  UART1 RX：PB3(fun2)、PE1(fun2)、PA12(fun3)

  UART1 TX：PB4(fun2)、PE0(fun2)、PA11(fun3)、PE2(fun1)

  UART1 DE：PB5(fun2)、PB10(fun2)

  UART1 RE：PB6(fun2)、PE2(fun2)

9.2. 寄存器

9.2.1. 寄存器基地址

Name	Base Address	Description
UART0	0x40004000	UART0 基地址
UART1	0x40004100	UART1 基地址

9.2.2. 寄存器列表

Offset Address	Name	Description
0x0000	UART0_CON	UART0 控制寄存器
0x0004	UART0_BUAD	UART0 波特率寄存器
0x0008	UART0_DATA	UART0 数据寄存器
0x000c	UART0_STA	UART0 状态寄存器

Offset Address	Name	Description
0x0000	UART1_CON	UART1 控制寄存器
0x0004	UART1_BUAD	UART1 波特率寄存器

0x0008	UART1_DATA	UART1 数据寄存器
0x000c	UART1_STA	UART1 状态寄存器
0x0010	UART1_TSTADR	UART1 发送 DMA 起始地址
0x0014	UART1_RSTADR	UART1 接收 DMA 起始地址
0x0018	UART1_TDMALEN	UART1 发送 DMA 的长度
0x001c	UART1_RDMALEN	UART1 接收 DMA 的长度
0x0020	UART1_TDMACNT	UART1 已发送 DMA 长度
0x0024	UART1_RDMACNT	UART1 已接收 DMA 长度
0x0028	UART1_DMACON	UART1 DMA 控制寄存器
0x002c	UART1_DMASTA	UART1 DMA 状态寄存器
0x0030	UART1_RS485CON	UART1 RS485 控制寄存器
0x0034	UART1_RS485DET	UART1 RS485 DET 寄存器
0x0038	UART1_RS485TAT	UART1 RS485 TAT 寄存器

9.2.3. 寄存器详细说明

9.2.3.1. UARTx_CON

Bit(s)	Name	Description	R/W	Reset
31:16	TO_BIT_LEN	**配置超时时间** 单位为比特率时间 Time_out_time= (TO_BIT_LEN+1) * bit_rate_time	RW	54
15	TCIE	**传输完全中断使能** 0x0：关闭 0x1：打开，当 UART_STA 寄存器中的 TC=1 时产生 UART 中断	RW	0x0
14	TMR_PWM_EN	UART 的输出和 TIMER 的 PWM 一起运算后才输出使能位（UART0 输出 timer0 的 PWM、UART1 输出 timer1 的 PWM） 0x0：关闭 0x1：打开	RW	0x0
13	TO_IE	**Time out 中断使能** 0x0：关闭 0x1：打开	RW	0x0
12	TO_EN	**Time out 检测使能位** 检测是否在经历了 5 *（1 start_bit + 8 data	RW	0x0

		bit + 2 stop bit）时间，没有接收到数据。每次 TO_EN 后，需要等到接收到一 byte 数据才会开始检测。接收到 1byte 数据这个标志，会在每次清 TO_PEND 的时候被清掉，或者在 TO_EN 等于 0 的时候被清掉。 0x0：关闭 0x1：打开		
11	FERR_IE	**帧出错中断使能位** 0x0：关闭 0x1：打开	RW	0x0
10	TXBUF_EMPTY_IE	**发送 buffer 空中断使能位** 0x0：关闭 0x1：打开	RW	0x0
9	RXBUF_NEMPTY_IE	**接收 buffer 空中断使能位** 0x0：关闭 0x1：打开	RW	0x0
8	TX_INV	**发送输出信号取反选择位** 0x0：关闭 0x1：打开	RW	0x0
7	RX_INV	**接收输入信号取反选择位** 0x0：不取反 0x1：取反	RW	0x0
6	ODD_EN	**奇偶校验选择** 0x0：偶校验 0x1：奇校验	RW	0x0
5	PARITY_EN	**奇偶校验使能** 0x0：关闭 0x1：打开 注：奇偶检验和 BIT9_EN 只能二选一	RW	0x0
4	BIT9_EN	**UART 传输 9bit 数据选择位** 0x0：UART 传输 8bit 数据 0x1：UART 传输 9bit 数据 注：奇偶检验和 BIT9_EN 只能二选一	RW	0x0
3	STOP_BIT	**结束位使能位** 0x0：一个结束位 0x1：两个结束位	RW	0x0
2:1	WORK_MODE	**工作模式** 0x0：全双工，可以同时收发 0x1：单工发送 0x2：单工接收 0x3：一根数据线，硬件自动切换该 IO 方向；在不是 rs485M 模式，软件触发发送 IO 方向就是输出，没有数据发送 IO 方向是输入。在 RS485 模式，DE 为有效时候 IO 是输出，在 DE 为无效的时候 IO 是输入	RW	0x0
0	UART_EN	**UART 模块使能位**	RW	0x0

		0x0：关闭 0x1：打开		

9.2.3.2. UARTx_BAUD

Bit(s)	Name	Description	R/W	Reset
31:18	Reserved	-	-	-
17:0	UARTx_BAUD	**UART 波特率设置** 波特率=SYS_CLK /(UARTx_BAUD+1) 注：UARTx_BAUD 一定要配置为大于等于 6，否则输入信号会被内部滤波器滤掉。	RW	0xa93

9.2.3.3. UARTx_DATA

Bit(s)	Name	Description	R/W	Reset
31:9	Reserved	-	-	-
8:0	UARTx_DATA	**UART 传输数据** 写 8/9 位数据到 UARTx_DATA：触发 UART 模块开始发送数据。 读 UARTx_DATA：获取接收到的低 8/9bit 数据，读之前，先读取 PERR[0]获取奇偶校验信息	RW	0x0

9.2.3.4. UARTx_STA

Bit(s)	Name	Description	R/W	Reset
31:16	-	-	-	-
15	TC	**传输完成** 如果一个帧的传输完成并且 TX_BUF_EMPTY 被设置，那么这个位由硬件设置。如果 UARTx_CON 寄存器中的 TCIE=1，则产生中断。 它由一个软件序列(写入 usart_dr 寄存器)清除。TC 位也可以通过写入“1”来清除。 0x0：表示传输未完成 0x1：传输完成	RO	0x0
14	UPDATE_DETECT_EN	**系统升级检测使能位** 检测是否接收到了下面的一串连续数据 0x70->0x61->0x73->0x53->0xf5->0x1e->0xf4->0xec 0x0：关闭 0x1：打开 注：当 UPDATE_DETECT_EN 为 1，并且 UPDATE_DETECT_PEND 为 0 时候	RO	0x0

		UARTx_CON:将固定为 0x1005(单工接收,1stop,8bit 模式，没有奇偶校验，没有取反，所有中断不使能）；UARTx_BAUD : 将由 EFLASH 模块配置，默认值会是 0xa93; UART_DMACON：将固定为 0x00（不使能 DMA）； UART_RS485_CON：将固定为 0x00（不使能 rs485）； 软件不可以改变。		
13	UPDATE_DETECT_PEND_G	这一位和 UPDATE_DETECT_PEND 一样，只是这位一旦为 1，除非 reset UART0，否则一直不会变为 0。	RO	0x0
12	UPDATE_DETECT_PEND	**检测到系统升级标志位** 表示接收到了下面的一串数据 0x70->0x61->0x73->0x53->0xf5->0x1e->0xf4->0xec。写 1 清零。 0x0：没有检测到系统升级 0x1：检测到系统升级 注：只有 UART0 有该位。只能软件清。	WC/R	0x0
11	TO_PEND	**Time out 标志位** 表示经历了 5 *（1 start_bit + 8 data bit + 2 stop bit）时间，没有接收到数据。写 1 清零。 0x0：time out 没有出现 0x1：time out 出现	WC/R	0x0
10:7	PERR	**接收数据奇偶校验出错** 4bit，分别对应 BUF 里面的 4 个数据。在读 UARTx_DATA 之前，需要先读这个寄存器，硬件会自动将 PERR[0]对应你读 UARTx_DATA 获取的 data。	RO	0x0
6:4	RX_CNT	**RX BUF 有多少个数据** 0x0：0 个数据 0x1：1 个数据 0x2：2 个数据 0x3：3 个数据 0x4：4 个数据 0ther：无效	RO	0x0
3	FERR	**帧错误** 表示在 STOPbit 期间检测到 RX_IN 出现了低电平。写 1 清零 0x0：没有帧错误 0x1：出现了帧错误	WC/R	0x0
2	RX_BUF_OV	接收 BUF 最多可以容纳 4 个 8/9bit 的数据，接收到 4 个数据后，如果软件还没有来得及读走，又收到一个数据,这个标志为会起来。写 1 清零。 0x0：接收了(0~4byte)数据 0x1：接收了(>4byte)的数据，BUF 里面只保存最开始的 4byte，其他丢弃	WC/R	0x0

1	RX_BUF_NOT EMPTY	**接收 BUF 不空标志位** 写 1 清零 0x0：接收 BUF 没有数据 0x1：接收 BUF 有数据	RO	0x0
0	TX_BUF_EMPTY	**发送完成标志位** 写 1 清零，或者写 UARTx_DATA 清 0 0x0：发送没有完成 0x1：发送完	RO	0x0

9.2.3.5. UART1_TSTADR

Bit(s)	Name	Description	R/W	Reset
31:13	Reserved	-	-	-
12:0	TSTADRT	**发送 DMA 起始地址** 如果是 9bit 数据，需要 16bit 对齐； 如果是 8bit 数据，可以任意配置。	RW	0x0

9.2.3.6. UART1_RSTADR

Bit(s)	Name	Description	R/W	Reset
31:13	Reserved	-	-	-
12:0	RSTADRT	**接收 DMA 起始地址** 如果是 9bit 数据，需要 16bit 对齐； 如果是 8bit 数据，可以任意配置。	RW	0x0

9.2.3.7. UART1_TDMALEN

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	TDMALEN	**计划发送 DMA byte 长度** 8bit 数据模式，uart 一帧数据等于 1byte； 9bit 数据，uart 一帧数据等于 2byte，TDMALEN 需要配置成 2 的倍数。	RW	0x0

9.2.3.8. UART1_RDMALEN

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	RDMALEN	**计划接收 DMA byte 长度** 8bit 数据模式，uart 一帧数据等于 1byte；	RW	0x0

		9bit 数据，uart 一帧数据等于 2byte，RDMALEN 需要配置成 2 的倍数。		

9.2.3.9. UART1_TDMACNT

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	TDMACNT	**已经发送 DMA byte 长度** 8bit 数据模式，uart 一帧数据等于 1byte； 9bit 数据，uart 一帧数据等于 2byte。	RO	0x0

9.2.3.10. UART1_RDMACNT

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	RDMACNT	**已经接收 DMA byte 长度** 8bit 数据模式，uart 一帧数据等于 1byte； 9bit 数据，uart 一帧数据等于 2byte。	RO	0x0

9.2.3.11. UART1_DMACON

Bit(s)	Name	Description	R/W	Reset
31:10	Reserved	-	-	-
9	RX_DMA_EN_CFG_EN	**RX_DMA_EN 配置使能** 配置 RX_DMA_EN 时候，这位同时写 1 才可以。自动清 0。 0x0：关闭 0x1：打开	WO	0x0
8	TX_DMA_EN_CFG_EN	**TX_DMA_EN 配置使能** 配置 TX_DMA_EN 时候，这位同时写 1 才可以。自动清 0。 0x0：关闭 0x1：打开	WO	0x0
7：5	Reserved	-	-	-*
4	RX_DMA_PERR_IE	接收 DMA 一包数据出现至少一个数据奇偶校验错误，中断使能 0x0：关闭 0x1：打开	RW	0x0
3	RX_DMA_IE	**接收 DMA 中断使能** 0x0：关闭 0x1：打开	RW	0x0
2	TX_DMA_IE	**发送 DMA 中断使能**	RW	0x0

		0x0：关闭 0x1：打开		
1	RX_DMA_EN	**接收 DMA 使能** 配置该位同时要往 RX_DMA_EN_CFG_EN 写 1 0x0：关闭 0x1：打开	RW	0x0
0	TX_DMA_EN	**发送 DMA 使能** 配置该位同时要往 TX_DMA_EN_CFG_EN 写 1 0x0：关闭 0x1：打开	RW	0x0

9.2.3.12. UART1_DMASTA

Bit(s)	Name	Description	R/W	Reset
31:3	Reserved	-	-	-
2	RX_DMA_PERR	**接收 DMA 一包数据出现至少一个数据奇偶校验错误标志位** 0x0：没有数据出现奇偶校验错误 0x1：出现了至少一个数据奇偶校验错误	RC	0x0
1	RX_DMA_PEND	**接收 DMA 完成标志位** 指数据接收完成并保存到 SRAM 0x0：DMA 没有完成 0x1：DMA 已经完成	RC	0x0
0	TX_DMA_PEND	**发送 DMA 完成标志位** 指数据从 SRAM 取出并且通过 uart 完整发送出去。 0x0：DMA 没有完成 0x1：DMA 已经完成	RC	0x0

9.2.3.13. UART1_RS485CON

Bit(s)	Name	Description	R/W	Reset
31:10	Reserved	-		-
9	RE_EN	RE 使能 0x0：关闭 0x1：打开	RW	0x0
8	DE_EN	**DE 使能** 0x0：关闭 0x1：打开	RW	0x0
7:4	Reserved	-	-	-
3	RS485_MODE	**RS485 工作模式** 0x0：软件配置 DE_EN，DE 会一直有效，直到配置 DE_EN 为 0；软件配置 RE_EN，RE 会一直有效，	RC	0x0

		直到配置 RE_EN 为 0；硬件会自动插入 DE_DAT，DE_AT，RE2DE_T，DE2RE_T 时间。这一位等于 0 的时候，DE_EN 和 RE_EN 不能同时置 1 0x1：硬件自动切换 DE 和 RE 模式，有数据需要发送切换到 DE，没有数据发送保持 RE		
2	RE_POL	**RE 极性** 0x0：高电平有效 0x1：低电平有效	RC	0x0
1	DE_POL	**DE 极性** 0x0：高电平有效 0x1：低电平有效	RW	0x0
0	RS485_EN	**RS485 使能** 0x0：关闭 0x1：打开	RW	0x0

9.2.3.14. UART1_RS485DET

Bit(s)	Name	Description	R/W	Reset
31:25	Reserved	-	-	-
24:16	DE_DAT	STOP BIT 结束到 DE 无效之间的时间间隔，单位是 uart 模块时钟	RW	0x0
15:9	-	-	-	-
8:0	DE_AT	DE 有效到发送 START BIT 之间的时间间隔，单位是 uart 模块时钟	RW	0x0

9.2.3.15. UART1_RS485TAT

Bit(s)	Name	Description	R/W	Reset
31:16	RE2DE_T	RE 有效到 DE 有效之间的时间间隔，单位是 uart 模块时钟	RW	0x0
15:0	DE2RE_T	DE 有效到 RE 有效之间的时间间隔，单位是 uart 模块时钟	RW	0x0

9.2.4. 使用说明

1）DMA 模式不使能的 UART 发送：

1.配置 UARTx_BAUD；

2.配置 UARTx_CON；

TX_IE ： 如果需要中断，这位配 1

TX_INV ： 如果需要将输出取反，这位配 1

UART_EN ： 配 1

BIT9_EN ： 如果需要传输 9bit 数据，这位配 1

BIT9 ： 如果 BIT9_EN，这位写入需要传输的第 9bit 数据

3.配置 UART_RS485_CON,UART_RS485_DAT,UART_RS485_TAT;

4.等待 UARTx_STA[0]（TX_BUF_EMPTY）位为 1，然后往 UARTx_DATA 中写入需要发送的 8bit/9bit 数据

5.如果还有需要发送的数据，重复 4.

6.如果已经没有数据需要发送，等待所有数据发送完成：等待 UARTx_STA[15] ：TC 置 1；

2）DMA 模式使能的 UART 发送：

1.配置 UARTx_BAUD.

2.配置 UARTx_CON

TX_IE ： 如果需要中断，这位配 1

TX_INV ： 如果需要将输出取反，这位配 1

UART_EN ： 配 1

BIT9_EN ： 如果需要传输 9bit 数据，这位配 1

BIT9 ： 如果 BIT9_EN，这位写入需要传输的第 9bit 数据

3.配置 UART_RS485_CON,UART_RS485_DAT,UART_RS485_TAT;

4.配置 DMACON,DMA_TSTADR,DMA_TDMALEN;

5.等待 DMA 完成:UART_DMA_CON[0]清零，或者 UART_DMASTA[0]为 1；

3）DMA 模式不使能的 UART 接收：

1.配置 UARTx_BAUD.

2.配置 UARTx_CON

RX_IE ： 如果需要中断，这位配 1

RX_INV ： 如果需要将输入取反，这位配 1

UART_EN ： 配 1

BIT9_EN ： 如果需要传输 9bit 数据，这位配 1

3.配置 UART_RS485_CON,UART_RS485_DAT,UART_RS485_TAT;

4.等待接收到数据:接收到数据后，UARTx_STA[1] ：RX_BUF_NOTEMPTY 会置 1，同时可以通过查看 UARTx_STA[6：4] ：RX_CNT 了解收到了多少 byte 数据。

5.读取接收到的数据，通过读 UARTx_DATA 获取接收到的数据。

6. 如果还有数据需要接收，重复 4 和 5.

4）DMA 模式使能的 UART 接收：

1.配置 UARTx_BAUD.

2.配置 UARTx_CON

RX_IE ： 如果需要中断，这位配 1

RX_INV ： 如果需要将输入取反，这位配 1

UART_EN ： 配 1

BIT9_EN ： 如果需要传输 9bit 数据，这位配 1

3.配置 UART_RS485_CON,UART_RS485_DAT,UART_RS485_TAT;

4.配置 DMACON,DMA_RSTADR,DMA_RDMALEN;

5.等待 DMA 完成:UART_DMA_CON[1]清零，或者 UART_DMASTA[1]为 1；

10. CRC 计算单元

10.1. 主要特性

- 支持 5/7/8/16/32 等不同长度的多项式
- 支持自定义的多项式

10.2. 寄存器

10.2.1. 寄存器基地址

Name	Base Address	Description
CRC	0x40002000	CRC 的基地址

10.2.2. 寄存器列表

Offset Address	Name	Description
0x00	CRC_CFG	配置寄存器
0x04	CRC_INIT	初始化配置寄存器
0x08	CRC_INV	取反寄存器
0x0C	CRC_POLY	多项式配置寄存器
0x10	CRC_KST	触发寄存器
0x14	CRC_STA	状态寄存器
0x18	CRC_ADDR	地址寄存器
0x1C	CRC_LEN	长度寄存器
0x20	CRC_OUT	结果输出寄存器

10.2.3. 寄存器详细说明

10.2.3.1. CRC_CFG

Bit(s)	Name	Description	R/W	Reset
31:14	Reserved		RO	0
13:8	POLY_WIDTH	**Poly 的位宽配置项** 支持 5/7/8/16/32 的配置，默认是 32 Bit	RW	32
7:2	Reserved		RO	0
1	BIT_ORDER_EN	**输出的数据是否取反** 0x0：关闭 0x1：打开，DATA[7:0] toDATA[0:7]	RW	0
0	INT_EN	**CRC 中断使能** 0x0：关闭	RW	0

		0x1：打开		

10.2.3.2. CRC_INIT

Bit(s)	Name	Description	R/W	Reset
31:0	INIT_VALUE	**CRC 初始值**	RW	0xFFFFFFFF

10.2.3.3. CRC_INV

Bit(s)	Name	Description	R/W	Reset
31:0	INV_VALUE	**CRC 输出结果反转**	RW	0xFFFFFFFF

10.2.3.4. CRC_POLY

Bit(s)	Name	Description	R/W	Reset
31:0	POLY_VALUE	**配置 CRC ploy 值** 默认值为 0xhedb88320	RW	0xhedb88320

10.2.3.5. CRC_KST

Bit(s)	Name	Description	R/W	Reset
31:1	Reserved		RW	0
1	Pending Clear	**清除 CRC 的标志位** 写 1 清除	WO	0

10.2.3.6. CRC_STA

Bit(s)	Name	Description	R/W	Reset
31:1	Reserved		RO	0
1	Pending	**CRC 完成标志位**	RO	0

10.2.3.7. CRC_POLY

Bit(s)	Name	Description	R/W	Reset
31:0	POLY_VALUE	**配置 CRC ploy 值** 默认值为 0xhedb88320	RW	0xhedb88320

10.2.3.8. CRC_ADDR

Bit(s)	Name	Description	R/W	Reset
31:2	DMA Start Address	**CRC DMA 开始地址** 注意：物理地址，比如地址为 0x0 重定向 0x2000_0000	RO	0
1:0	Reserved	保留，用于 4 byte 对齐	RO	0

10.2.3.9. CRC_LEN

Bit(s)	Name	Description	R/W	Reset
31:0	DMA Length	**CRC DMA 长度配置** 4 byte 对齐	RW	0x0

10.2.3.10. CRC_OUT

Bit(s)	Name	Description	R/W	Reset
31:0	CRC Result	**CRC 计算结果输出**	RO	0x0

10.3. 操作流程

2、配置 CRC 初始值(CRC_INIT)和取反(CRC_INV)；

3、配置 CRC 多项式(CRC_POLY)，CRC 多项式的位宽(CRC_CFG)；

4、配置 DMA 的开始地址（物理地址），最后配置 DMA 的长度，一旦长度不为 0，即开始执行 CRC 校验；

5、等待 CRC 的 pending(CRC_STA)，得到 CRC 的结果(CRC_OUT)，最后清除 pending(CRC_KST)。

11. 硬件加速单元

11.1. 加速单元简介

本加速单元包含一个硬件除法器。硬件除法在一些高性能的应用中非常有用，能自动执行有符号或者无符号的 32 位/16 位整数除法运算。

11.2. 硬件除法主要特征

- 有符号或者无符号整数除法运算
- 32 位除数和 16 位被除数，输出 32 位的商和 16 位余数
- 8 个 HCLK 周期完成
- 如果除数为零，会产生溢出中断标志位
- 写除数自动执行除法运算
- 读商和余数寄存器时自动等待运算结束，不需要检查状态位

11.3. 硬件除法功能介绍

硬件除法单元包括 4 个 32 位数据寄存器，分别为被除数，除数，商和余数，可以做有符号或者无符号的 16 位除法运算。通过硬件除法控制寄存器 SIGN 可以选择是有符号除法还是无符号除法。每一次写入除数寄存器，会自动触发除法运算，在运算结束后，结果会写入到商和余数寄存器里。如果在结束前读商寄存器、余数寄存器或者状态寄存器，读操作会被暂停，直到结束才返回运算结果。如果除数为零，会产生溢出中断标志位

11.4. 寄存器

11.4.1. 寄存器基地址

Name	Base Address	Description

Name	Base Address	Description
DIV	0x40020A00	硬件除法器基地址

11.4.2. 寄存器列表

Offset Address	Name	Description
0x0000	DIV_DVD	被除数寄存器
0x0004	DIV_DVS	除数寄存器
0x0008	DIV_QUO	商寄存器
0x000c	DIV_REM	余数寄存器
0x0010	DIV_SR	状态寄存器
0x0014	DIV_CON	控制寄存器

11.4.3. 寄存器详细说明

11.4.3.1. DIV_DVD

Bit(s)	Name	Description	R/W	Reset
31:0	Dividend	被除数寄存器位（Dividend data）	RW	0x0

11.4.3.2. DIV_DVS

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15：0	DIVISOR	除数寄存器位（Divisor data） 写完该寄存器后，自动触发除法运算。	RW	0x0

11.4.3.3. DIV_QUO

Bit(s)	Name	Description	R/W	Reset
31：0	QUOTIENT	商寄存器位（Quotient data）	RW	0x0

11.4.3.4. DIV_REM

Bit(s)	Name	Description	R/W	Reset

31:16	Reserved	-	-	-
15：0	REMAINDER	余数寄存器位（Remainder data）	RW	0x0

11.4.3.5. DIV_SR

Bit(s)	Name	Description	R/W	Reset
31:1	Reserved	-	-	-
0	DIV0_FLAG	read 1:当前除法操作除数为 0 read 0:当前除数不为 0 写 1 与写 DVSR 均能清除这个 bit 写 0：无操作	RW	0x0

11.4.3.6. DIV_CON

Bit(s)	Name	Description	R/W	Reset
31:2	Reserved	-	-	-
1	INT_EN	0x0：除数为 0 不产生中断信号 0x1：除数为 0 产生中断信号	RW	0x1
0	SIGN	0x0：无符号除法 0x1：有符号除法	RW	0x0

12. 比较器（COMP）

12.1. 简介

芯片内嵌一个通用比较器 COMP，可独立使用，也可与定时器和 EPWM 结合使用。

12.2. 主要特性

- 轨对轨比较器
- 可复用 I/O，内部一端连接到 DAC 上
- 可编程迟滞电压
- 支持比较结果的滤波功能

- 输出端可以重定向到一个 I/O 端口
- 输出端连接到多个定时器输入端，可以触发定时器的捕获事件
- 输出端连接到多个 EPWM 输入端，可以触发 EPWM 刹车
- COMP 有 7 个正相输入和 4 个反向输入
- 比较器可产生中断，并支持把 CPU 从睡眠模式和停机模式唤醒

12.3. 功能描述

12.3.1. 比较器功能框图

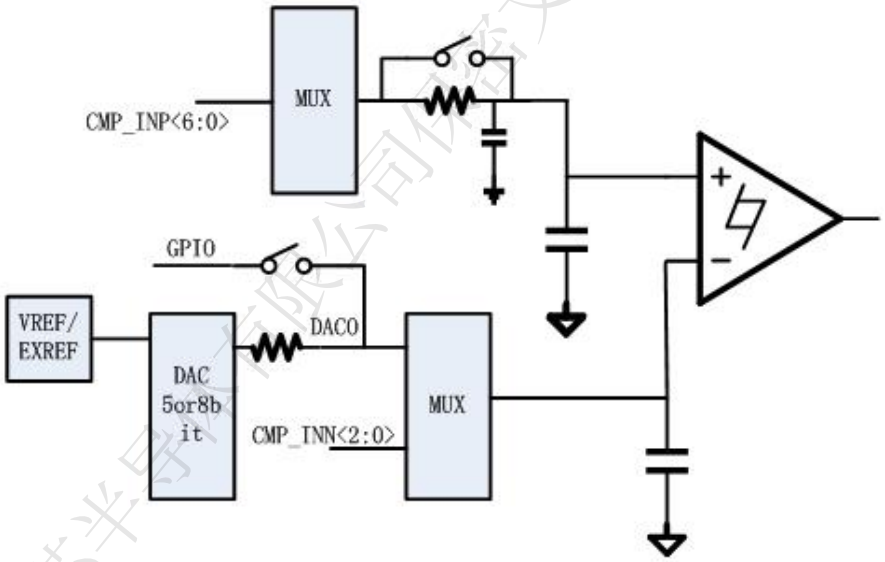


图 12-1 通道结构图

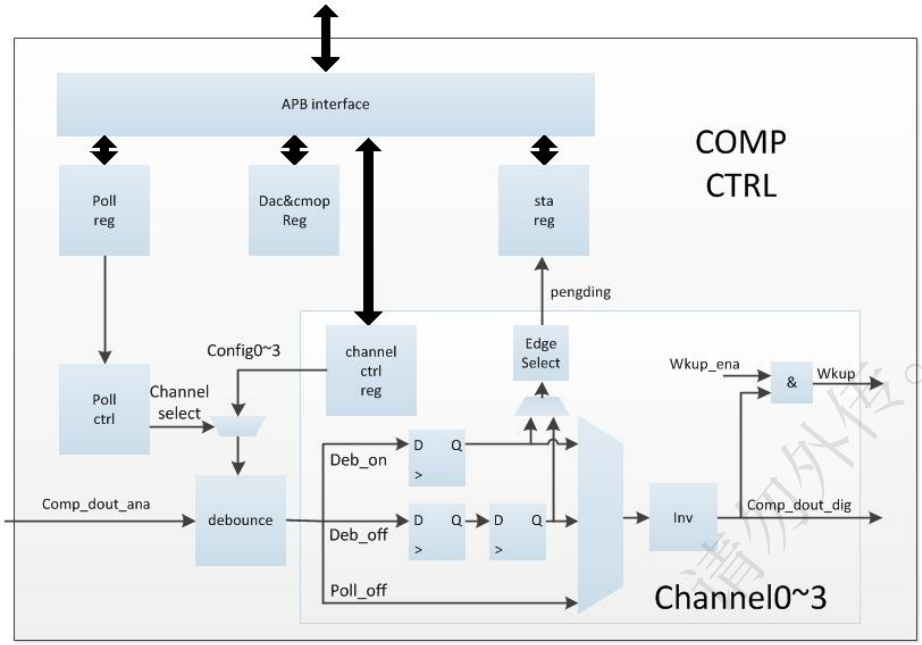


图 12-2 控制结构图

12.3.2. 比较器输入和输出

作为比较器输入的 I/O 引脚必须在 GPIO 寄存器中设置为模拟模式。

比较器的输出可以重定向到多个 I/O 端口。

比较器的输出内部连接到多个定时器的输入：作为输入捕获，测量时序。

比较器的输出内部连接到多个 EPWM 的输入：作为刹车。

12.3.3. 比较器工作模式

比较器分为普通工作模式和轮询工作模式。

在普通工作模式下，比较器只能比较软件设定的通道，通过配置 CMP_CHAN0 完成操作。软件操作流程如下：

1. 配置 CMP_CON 寄存器，设置 DAC 和 CMP 的工作条件；
2. 配置 CMP_CHAN0 寄存器的 POLL_INPSEL 和 POLL_INNSEL 位选择比较的信号；
3. 配置 CMP_CHAN0 寄存器中断使能，反相使能，滤波周期，唤醒边沿以及唤醒使能；
4. 配置 CMP_CON 寄存器的 CMPEN 位使能比较器。

在轮询工作模式下，比较器会每隔一段时间比较不同的通道，最多轮询四个通道，每个通道的输入可选择。可配置正相和反相端同时轮询，也可以选择一端轮询，另一端固定通道，通过配置 CMP_POLL 以及 CMP_CHAN0~3 完成操作。软件操作流程如下：

1. 配置 CMP_CON 寄存器，设置 DAC 和 CMP 的工作条件；
2. 配置 CMP_CHAN0~3 寄存器的 POLL_INPSEL 和 POLL_INNSEL 位选择比较的信号；
3. 配置 CMP_CHAN0~3 寄存器中断使能，反相使能，滤波周期，唤醒边沿以及唤醒使能；
4. 配置 CMP_POLL 寄存器轮询通道数，轮询周期，正相和反相轮询使能以及轮询 MASK；
5. 配置 CMP_CON 寄存器的 CMPEN 位使能比较器。

注：poll_num 为 0 时，默认为普通工作模式。

12.3.4. 比较器滤波控制

因比较器输入端发生变化导致输出端的不稳定状态，比较器提供了两种方法对比较器输出进行滤波，数字滤波与硬件 MARK。

1.数字滤波即直接对比较器的输出结果进行滤波 filt_num（COMP_CHANx 寄存器）个时钟周期。

2.硬件 MARK 即在 CHANNEL 的轮询周期的前 poll_mark（COMP_POLL 寄存器）个时钟周期将比较器的输出屏蔽。如比较器的输出稳定需要 3 个时钟周期，可以把 poll_mark 配成 4，硬件就取 4 个时钟周期后的比较器结果进行寄存，直接屏蔽掉不稳定的结果。

在普通工作模式下，只能使用数字滤波的方法。在轮询工作模式下，两种方法都可以使用。

12.3.5. 比较器轮询周期

在轮询工作模式下，轮询周期需满足以下公式：

1.使用数字滤波：Poll_period>= 2 * filt_num + 6

2.使用硬件 MARK：Poll_period>=poll_mark + 2

3 同时使用需满足：①Poll_period>=poll_mark + 2 *filt_num+ 6②Filt_num + 5 <=

poll_mark

在轮询工作模式下，比较器输入端的信号保持时间需满足以下公式：

Hold_time>poll_num * poll_period

注：轮询且通道反相输出时，必须使用硬件 MARK 方式。

12.3.6. 比较器中断和唤醒

比较器的输出可以内部连接到外部中断和时间控制器。比较器输出唤醒信号，能产生中断，事件或用来退出低功耗模式。

12.3.7. 比较器锁定机制

比较器能用于安全的用途，比如过流保护或者过热保护。在某些特定的安全需求的应用中，有必要保证比较器设置不能被无效寄存器访问或者程序计数器破坏所改变。

为了这个目的，比较器控制和状态寄存器可以设为写保护。

一旦设置完成，LOCK 位必须设为 1，这导致整个寄存器变为只读，包括 LOCK 位在内。

写保护只能被 MCU 复位清除。

12.3.8. 比较器迟滞现象

比较器的可配置迟滞电压能防止无效的输出变化产生的噪声信号。在不需要强制迟滞电压的情况下迟滞现象可以被禁止。

12.4. 寄存器

12.4.1. 寄存器基地址

Name	Base Address	Description
COMP	0x40010200	比较器基地址

12.4.2. 寄存器列表

Offset Address	Name	Description
0x0000	COMP_CON	控制寄存器
0x0004	COMP_POLL	轮询寄存器
0x0008	COMP_CHAN0	通道 0 寄存器
0x000c	COMP_CHAN1	通道 1 寄存器
0x0010	COMP_CHAN2	通道 2 寄存器
0x0014	COMP_CHAN3	通道 3 寄存器
0x0018	COMP_CLR	清除寄存器
0x001C	COMP_STA	状态寄存器

12.4.3. 寄存器详细说明

12.4.3.1. COMP_CON

Bit(s)	Name	Description	R/W	Reset
31	Lock	**比较器锁** 该位只能写一次，由软件置“1”，由系统复位清零。 它令比较器的所有控制位为只读。 0x0：COMP_CON 可读可写 0x1：COMP_CON 只读	RW	0x0
30:29	reserve	-	-	-
28:21	Dak	DAC 数据输入	RW	0x0
20:19	Cmphyst	**CMP 迟滞功能选择** 0x0：没有迟滞 0x1：20mV 0x2：40mV 0x3：70mV	RW	0x0
18:17	Cmpimir	**CMP 偏置电流比例** 0x0：2:1 0x1：3:1 0x2：4:1 0x3：5:1	RW	0x2
16:15	Cmpmode	**比较器工作模式选择** 0x0：最低功耗模式 0x1：此低功耗模式	RW	0x2

		0x2：常规模式 0x3：最高速模式		
14:13	Daflres	**DAC Filter res select** 0x0：0 0x1：10K 0x2：20K 0x3：30K	RW	0x0
12:11	Dafssel	**8 bit DAC 量程选择** 0x0：1.6V 0x1：2.4V 0x2：3.2V 0x3：4.8V	RW	0x0
10:8	Isumres	**比较器正相输入端 Fitter RES select** 0x0：50K 0x1：100K 0x2：150K 0x3：200K Other：bypass RES 滤波电容约 3.5 pf	RW	0x6
7	Cmpen	**比较器使能** 0x0：关闭 0x1：打开	RW	0x0
6	Cmpnen	**CMP 反相输入** 0x0：关闭 0x1：打开	RW	0x1
5	Cmppen	**CMP 正相输入** 0x0：关闭 0x1：打开	RW	0x1
4	Dabitsel	**DAC 模式选择** 0x0：8 bit 0x1：5 bit	RW	0x1
3	Dacoutsw	**DAC 输出到 GPIO** 0x0：关闭 0x1：打开	RW	0x0
2	Daen	**DAC 使能** 0x0：关闭 0x1：打开	RW	0x0
1	Dafpen	**8 bit DAC fast path enable** 0x0：关闭 0x1：打开	RW	0x0
0	Darefsel	**5 bit DAC 参考源（量程）选择** 0x0：inside ref,1.2V 0x1：outside ref	RW	0x1

12.4.3.2. COMP_POLL

Bit(s)	Name	Description	R/W	Reset
31	lock	**比较器锁** 该位只能写一次，由软件置“1”，由系统复位清零。 它令比较器的所有控制位为只读。 0x0：COMP_POLL 可读可写 0x1：COMP_POLL 只读	RW	0x0
30:12	reserved	-	-	-
11:7	Poll_mark	**通道 mark 周期** 0x00：禁止 mark 0x01：1 CMP CLK 0x02：2 CMP CLK … 0x1F：31 CMP CLK	RW	0x0
6:5	Poll_num	**每个轮询通的通道数** 0x0：1 个通道 0x1：2 个通道 0x2：3 个通道 0x3：4 个通道	RW	0x0
4:2	Poll_period	**轮询周期** 0x0：1 CMP CLK 0x1：2 CMP CLK 0x2：4 CMP CLK .. 0x7：128 CMP CLK	RW	0x0
1	npoll_en	**反相端轮询使能** 0x0：关闭 0x1：打开	RW	0x0
0	ppoll_en	**正相端轮询使能** 0x0：关闭 0x1：打开	RW	0x0

12.4.3.3. COMP_CHANx

Bit(s)	Name	Description	R/W	Reset
31	Lock	**比较器锁** 该位只能写一次，由软件置“1”，由系统复位清零。 它令比较器的所有控制位为只读。 0x0：COMP_CHAN0 可读可写 0x1：COMP_CHAN0 只读	RW	0x0
30:16	Reserve	-	-	-
15:13	poll_innsel	**< 2 >通道 x 使能位**	RW	0x0

		0x0：关闭 0x1：打开 **<1:0>反相端通道 x 的输入选择** 0x0：PA9 0x1：PA10 0x2：PA11 0x3：DAC		
12:10	poll_inpsel	**正相端通道 x 的输入选择** 0x0：PA0 0x1：PA3 0x2：PA5 0x3：PA6 0x4：AMP_VOUT1(运放 1 输出) 0x5：AMP_VOUT2(运放 2 输出) 0x6：AMP_VOUT3(运放 3 输出) 0x7：断开与 GPIO 连接	RW	0x0
9:8	Cpu_wkup_sel	**唤醒边沿选择** 0x0：比较器 0 上升沿唤醒 0x1：比较器 0 下降沿唤醒 0x2：比较器上升沿和下降沿都能唤醒 others：保留	RW	0x0
7:3	filt_num	**滤波周期** 0x00：0CMP_CLK 0x01：1 CMP_CLK 0x02：2 CMP_CLK .. 0x1F：31 CMP_CLK	RW	0x0
2	Inv_en	**反相使能** 0x0：关闭 0x1：打开	RW	0x0
1	wkup_en	**唤醒使能** 0x0：关闭 0x1：打开	RW	0x0
0	Int_en	**通道 x 中断使能** 0x0：关闭 0x1：打开	RW	0x0

12.4.3.4. COMP_CLR

Bit(s)	Name	Description	R/W	Reset
31:4	reserved	-	-	-
3	Ch3_pend_clr	写 1 清除比较器通道 3 输出标志信号	WO	0x0
2	Ch2_pend_clr	写 1 清除比较器通道 2 输出标志信号	WO	0x0
1	Ch1_pend_clr	写 1 清除比较器通道 1 输出标志信号	WO	0x0
0	Ch0_pend_clr	写 1 清除比较器通道 0 输出标志信号	WO	0x0

12.4.3.5. COMP_STA

Bit(s)	Name	Description	R/W	Reset
31:4	reserved	-	-	-
3	Ch3_pend	比较器通道 3 输出标志信号	RO	0x0
2	Ch2_pend	比较器通道 2 输出标志信号	RO	0x0
1	Ch1_pend	比较器通道 1 输出标志信号	RO	0x0
0	Ch0_pend	比较器通道 0 输出标志信号	RO	0x0

13. 运算放大器（OPA）

13.1. 简介

芯片内嵌三个运算放大器，运算放大器的输入和输出都连接到 I/O，通过共享 I/O 可以与 ADC、比较器相连。

13.2. 主要特征

- 轨对轨输入/输出
- 输出连接到 I/O 上

13.3. 寄存器描

13.3.1. 寄存器基地址

Name	Base Address	Description
AMP	0x4002006C	运放基地址

13.3.2. 寄存器列表

Offset Address	Name	Description

0x0000	AMP_CON0	控制寄存器 0
0x0004	AMP_CON1	控制寄存器 1

13.3.3. 寄存器详细说明

13.3.3.1. AMP_CON0

Bit(s)	Name	Description	R/W	Reset
31:27	qcsel3	**AMP3 filter cap 选择** 0x01：300fp 0x02：300fp 0x04：400fp 0x08：400fp 0x10：600fp	R/W	0x1
26:24	gainsel3	**AMP3 gain 选择** 0x0：x4 0x1：x6 0x2：x8 0x3：x10 0x4：x12 0x5-0x7：disable feedback loop	R/W	0x2
23:19	qcsel2	**AMP2 filter cap 选择** 0x01：300fp 0x02：300fp 0x04：400fp 0x08：400fp 0x10：600fp	R/W	0x1
18:16	gainsel2	**AMP2 gain 选择** 0x0：x4 0x1：x6 0x2：x8 0x3：x10 0x4：x12 0x5-0x7：disable feedback loop	R/W	0x2
15:11	qcsel1	**AMP1 filter cap 选择** 0x01：300fp 0x02：300fp 0x04：400fp 0x08：400fp 0x10：600fp	R/W	0x1
10:8	gainsel1	**AMP1 gain 选择** 0x0：x4	R/W	0x2

		0x1：x6 0x2：x8 0x3：x10 0x4：x12 0x5-0x7：disable feedback loop		
7	ampldo_en	**AMP LDO 使能** 0x0：关闭 0x1：打开	R/W	0x0
6	bias_en	**BIAS 使能** 0x0：关闭 0x1：打开	R/W	0x0
5	biasadd_en	**BIASADD 使能** 0x0：关闭 0x1：打开	R/W	0x0
4	rcchan_en	**RCCHAN 使能** 0x0：关闭 0x1：打开	R/W	0x0
3	vcmbuff_en	**VCMBUFF 使能** 0x0：关闭 0x1：打开	R/W	0x0
2	amp3_en	**AMP3 使能** 0x0：关闭 0x1：打开	R/W	0x0
1	amp2_en	**AMP2 使能** 0x0：关闭 0x1：打开	R/W	0x0
0	amp1_en	**AMP1 使能** 0x0：关闭 0x1：打开	R/W	0x0

13.3.3.2. AMP_CON1

Bit(s)	Name	Description	R/W	Reset
31:13	Reserved	-	-	-
12	vout2pad_en	**AMP2 输出到 PB2 使能** 0x0：关闭 0x1：打开	R/W	0x0
11	vout1pad_en	**AMP1 输出到 PB1 使能** 0x0：关闭 0x1：打开	R/W	0x0
10:9	rcrsel	**AMP3 RC 滤波电阻选择** 0x0：1K 0x1：10K 0x2：50K 0x3：100K	R/W	0x0

8:6	restrim3	**AMP3 feedback 配置** 0x0：1.17X 0x1：1.05X 0x2：1X 0x3：0.95X 0x4：0.87X 0x5 - 0x7：disable feedback loop	R/W	0x2
5:3	restrim2	**AMP2 feedback 配置** 0x0：1.17X 0x1：1.05X 0x2：1X 0x3：0.95X 0x4：0.87X 0x5 - 0x7：disable feedback loop	R/W	0x2
2:0	restrim1	**AMP1 feedback 配置** 0x0：1.17X 0x1：1.05X 0x2：1X 0x3：0.95X 0x4：0.87X 0x5 - 0x7：disable feedback loop	R/W	0x2

14. ADC

14.1. 功能简介

该模块是一个12bit的逐次逼近式的ADC控制器。ADC支持多种工作模式：单次转换和连续转换，可选择通道自动扫描，以及多通道触发扫描。ADC的启动包括软件启动、外部引脚触发以及其他片内外设启动（timer触发启动、epwm）。

14.2. 主要特征

最高12位可编程分辨率的SAR ADC，多达13路外部输入通道和5路内部通道

高达1.2Msps转换速率

支持多种工作模式

- ✓ 单次转换模式：A/D转换在指定通道完成一次转换
- ✓ 单周期扫描模式：A/D转换在所有指定通道完成一个周期（从低序号通道到高序号通道或者从高序号通道到低序号通道）转换

✓ 连续扫描模式：A/D转换连续执行单周期扫描模式直到软件停止A/D转换

✓ 多通道独立触发模式：可配置6个通道，每个通道独立触发源，触发启动A/D采样通道采样时间，分辨率可软件配置

支持DMA传输。

A/D转换开始条件

✓ 软件启动

✓ 外部IO触发启动

✓ 内部timer触发启动

✓ 内部epwm触发启动

14.3. 结构框图

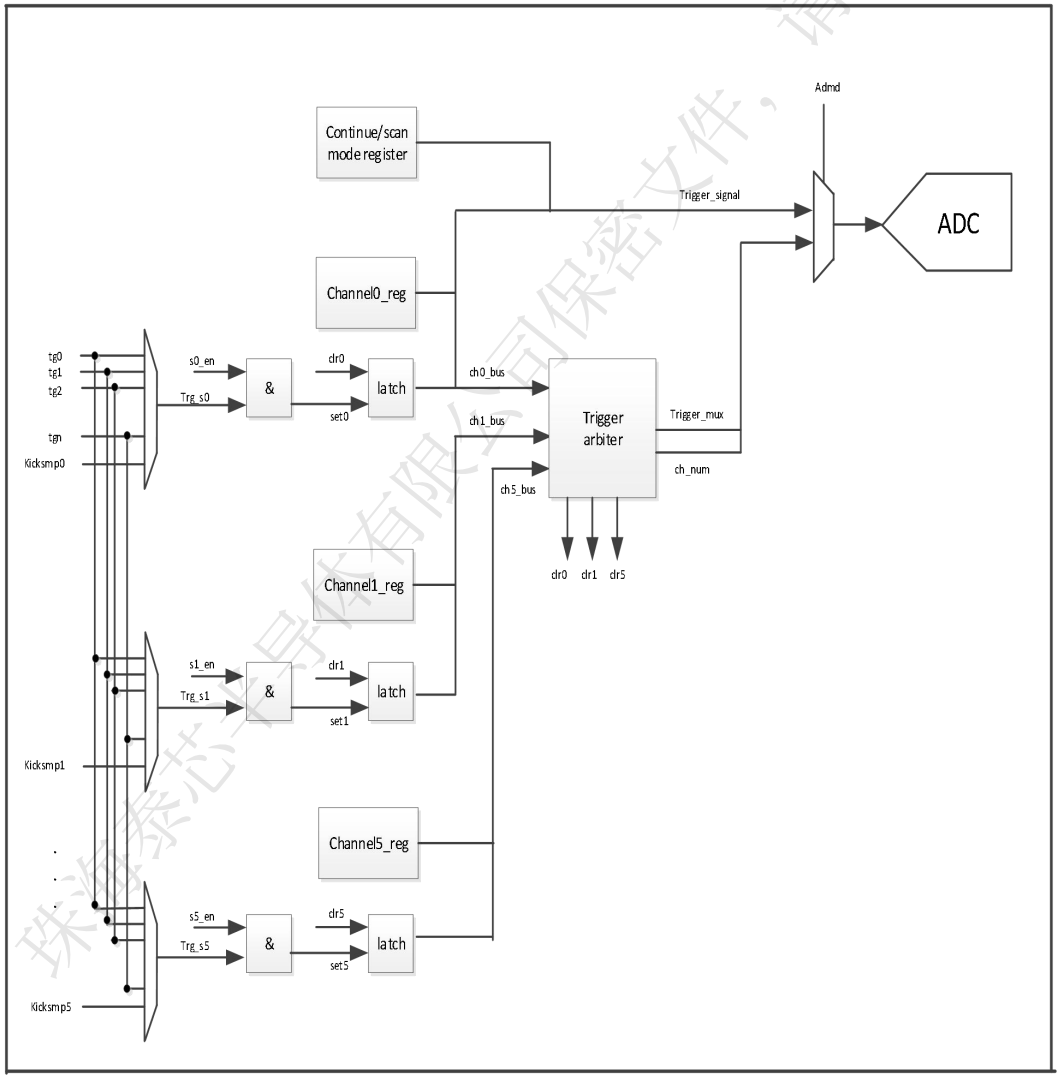


图 14-1 ADC 功能结构图

14.4. 功能描述

14.4.1. ADC 开关控制

通过设置 ADCFG 寄存器的 ADEN 位可给 ADC 上电。当第一次设置 ADEN 位时，它将 ADC 从断电状态下唤醒。ADC 上电延迟一段时间后（tSTAB），设置采样模式、通道之后，设置 ADST 位开始进行转换。通过清除 ADST 位可以停止转换，设置 ADEN 位可置于断电模式。

14.4.2. 通道选择

包含 15 路外部输入通道、内部温度传感器通道和内部 1.2V 参考电压通道。每个外部输入通道都有独立的使能位，可通过设置 ADCHS 寄存器的对应位来设置。

14.4.3. ADC 工作模式

14.4.3.1. 单次转换模式

在单次转换模式下，A/D 转换相应通道上只执行一次，具体流程如下：

- 软件配置转换模式 ADMD 为 0，使能 ADEN。
- 配置 TRGCON0 寄存器 PADSEL0 和 INSSEL0 选择采样通道。
- 通过软件将 ADST 置 1 启动 ADC。
- 当 A/D 转换完成，A/D 转换的数据值将存储于 A/D 的数据寄存器 ADC_DATA0 中。
- A/D 转换完成，状态寄存器 ADC_STA 的 DONE0 位置‘1’。若此时控制寄存器 ADC_IE 的 TRGIE0 位置‘1’，将产生 AD 转换结束中断请求。
- A/D 转换期间，ADST 位保持为 1。A/D 转换结束，ADST 位自动清 0，A/D 转换器进入空闲模式。

14.4.3.2. 单周期扫描模式

在单周期扫描模式下，将进行一次从被使能的最小序号通道向最大序号通道的 A/D 转换，可通过配置寄存器位 SCAN_DIR 选择扫描通道方向，操作步骤如下：

- 软件配置转换模式 ADMD 为 1，使能 ADEN。
- 配置 DMA 基地址、长度，使能 DMAEN。
- 配置 ADC_CHS 选择采样通道。
- 软件将 ADST 置 1、或外部触发置位 ADST 启动 ADC，外部触发只能通过 TRGCON0 来配置。可软件配置启动延时。
- 每路 A/D 转换完成后，A/D 转换数值将有序装载到相应的 SRAM 中，ADC_STA 的 DONE0 转换结束标志被设置，如果设置了转换结束中断，则在所有通道转换都完成后产生中断请求。
- 转换结束后，ADST 位自动清 0，A/D 转换器进入空闲状态。

14.4.3.3. 连续扫描模式

在连续扫描模式下，A/D 转换在 ADCHS 寄存器中的 CHENn 位被使能的通道上顺序进行（可通过配置寄存器位 SCAN_DIR 选择扫描通道方向），操作步骤如下：

- 软件配置转换模式 ADMD 为 2，使能 ADEN。
- 配置 DMA 基地址、长度，使能 DMAEN。
- 配置 ADC_CHS 选择采样通道。
- 软件将 ADST 置 1 启动 ADC。
- 每路 A/D 转换完成后，A/D 转换数值将有序装载到相应的 SRAM 中。
- 当所有通道的 A/D 转换完成一轮后，ADC_STA 的 DONE0 转换结束标志被置 1；当 ADST 位被清 0，该 DONE0 标志位也会被置 1。
- 只要 ADST 位保持为 1，ADC 按照 SCANDIR 重复扫描采样通道。当 ADST 位被清 0 时，硬件等待当前 channel A/D 转换完成后停止。

14.4.3.4. 多通道独立触发模式

该模式可同时选择启动 1~3 个独立通道，通过 TRGCONx 的 PADSELx 和 INSSELx 来选择对应的 ADC 采样通道，并且每个通道的采样触发源可通过 TRGCONx 的 SRCSELx 寄存器选择。操作步骤如下：

- 软件配置转换模式 ADMD 为 3，使能 ADEN。
- 配置 TRGCONx 的 PADSELx 和 INSSELx 选择对应采样通道。
- 配置 TRGCONx 的 SRCSELx 选择采样触发源。
- 配置 TRGCONx 的 KICKSMPx 启动采样，或者通过触发源启动采样。
- 配置 TRGCONx 的 TRGENx 选择触发使能。
- 读取 ADC_STA 对应的 DONEx 位和 ADC_DATAx 得到对应的 ADC 采样数据。

14.4.4. DMA 请求

单周期扫描和连续扫描时最近一次转换的结果会保存在 ADDATA 寄存器中。DMA 传输时传输所有扫描通道的结果顺序存放到对应 SRAM 中，通过 ADC_DMAADR 和 ADC_DMALEN 配置 dma 起始地址和长度，并且由 DATCNT 寄存器指示存放的采样个数。

14.4.5. 采样频率设置

ADC 的时钟 ADCLK 由 PCLK 分频得到，分频系数可通过设置 ADCFG 寄存器的 ADCPRE 位来确定，即 PCLK/(N+1)分频后作为 ADC 时钟。

ADC 最快采样率= F_{pclk} /(ADCPRE+1)/20

14.4.6. 连续采样间隔时间可配置

对于连续扫描模式，连续两个通道采样之间可以配置采样延时（即在一个通道扫描完成之后，等待一

段时间再进行下一通道扫描）。

14.4.7. 外部触发转换

ADC 转换可以由外部事件触发（例如定时器、IO）。如果设置了 ADC_TRGCONx 寄存器的 TRGENx 位，就可以使用外部事件触发转换。通过设置 SRCSELx 位可以选择外部触发源。

具体的外部触发源选择情况，可以参考 ADC_TRGCONx 控制寄存器的描述。

外部触发可设置延时控制，具体参考 ADC_CR 的 TRGSHIFT 的描述。

在触发信号产生后，延时 N 个 PCLK 的时钟周期再开始采样。如果是触发扫描模式，则只有第一个通道采样被延时，其他通道是在上一个采样结束后立即开始。

14.5. ADC 接口时序

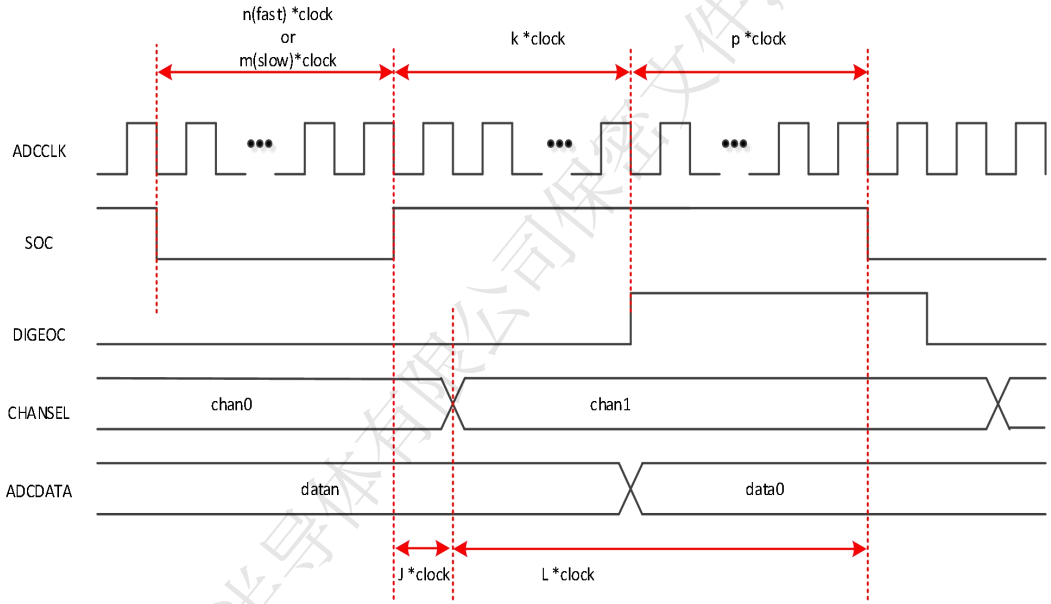


图 14-2 ADC 时序图

N：采样周期（高速），可配置 FSMPCYC“4~31”，至少 5 个 ADCCLK
M：采样周期（低速），可配置 SSMPCYC“4~31”,至少 5 个 ADCCLK
k：转换时间，可配置 RSLTCYC“9~13”
J：通道保持时间，可配置 CHHOLDCYC“3~RSLTCYC”，至少 4 个 ADCCLK
L：通道建立时间，可配置 CHSETUPCYC“3~(RSLTCYC-CHHOLDCYC)”
P：连续采样间隔时间，可配置 D2DCYC“0~15”至少 1 个 ADCCLK

14.5.1. ADC 上电时序

延时时间	使能信号

100us	BIASEN_VDD
300us	VCMEN_VDD
100us	CMPEN_VDD
100us	ADCEN_VDD
500us	SOC_VDD/CK_VDD

14.6. 寄存器

14.6.1. 寄存器基地址

Name	Base Address	Description
ADC	0x40000200	ADC 基地址

14.6.2. 寄存器列表

Offset Address	Name	Description
0x0000	ADC_CFG0	配置寄存器 0
0x0004	ADC_CFG1	配置寄存器 1
0x0008	ADC_TRGCON0	触发寄存器 0
0x000c	ADC_TRGCON1	触发寄存器 1
0x0010	ADC_TRGCON2	触发寄存器 2
0x0014	ADC_TRGCON3	触发寄存器 3
0x0018	ADC_TRGCON4	触发寄存器 4
0x001c	ADC_TRGCON5	触发寄存器 5
0x0020	ADC_CR	控制寄存器
0x0024	ADC_CHS	通道寄存器
0x0028	ADC_IE	中断寄存器
0x002c	ADC_STA	状态寄存器
0x0030	ADC_DATA0	数据寄存器 0
0x0034	ADC_DATA1	数据寄存器 1
0x0038	ADC_DATA2	数据寄存器 2
0x003c	ADC_DATA3	数据寄存器 3

0x0040	ADC_DATA4	数据寄存器 4
0x0044	ADC_DATA5	数据寄存器 5
0x0048	ADC_DMAADR	DMA 地址寄存器
0x004c	ADC_DMACNT	DMA 计数寄存器
0x0050	ADC_DMALEN	DMA 长度寄存器
0x0054	ADC_ANACON	模拟控制寄存器
0x0058	ADC_ANAPAR0	模拟配置寄存器 0
0x005c	ADC_ANAPAR1	模拟配置寄存器 1
0x0060	ADC_ANAPAR2	模拟配置寄存器 2

14.6.3. 寄存器详细说明

14.6.3.1. ADC_CFG0

Bit(s)	Name	Description	R/W	Reset
31:28	CHHOLDCYC	**通道保持时间** N=n+1 个 ADC clock 周期宽度	RW	0x3
27:23	CHSETCYC	**通道建立时间** N=n+1 个 ADC clock 周期宽度	RW	0xA
22:18	Reserve	-	-	-
17:13	FSMPCYC	**高速采样时间配置** N=n+1 个 ADC clock 周期宽度	RW	0x4
12:8	RSLTCYC	**转换周期配置（影响数据有效精度）** N=n+1 个时钟周期宽度，n 为 ADC 转换数据分辨率 5’h9: 8 位有效 5’ha：9 位有效 5’hb: 10 位有效 5’hc：11 位有效 5’hd：12 位有效	RW	0xD
7:0	ADCPRE	**ADC 时钟预分频（ADC prescaler）** n =（n+1）分频	RW	0x3

14.6.3.2. ADC_CFG1

Bit(s)	Name	Description	R/W	Reset
31:27	Reserved	-	-	-
26:18	SSMPCYC	**低速采样时间配置** N=n+1 个 ADC clock 周期宽度	RW	0x4

17	SMPCSEL17		RW	0x0
16	SMPCSEL16		RW	0x0
15	SMPCSEL15		RW	0x0
14	SMPCSEL14		RW	0x0
13	SMPCSEL13		RW	0x0
12	SMPCSEL12		RW	0x1
11	SMPCSEL11		RW	0x1
10	SMPCSEL10		RW	0x1
9	SMPCSEL9		RW	0x1
8	SMPCSEL8		RW	0x1
7	SMPCSEL7		RW	0x1
6	SMPCSEL6		RW	0x1
5	SMPCSEL5		RW	0x1
4	SMPCSEL4		RW	0x1
3	SMPCSEL3		RW	0x1
2	SMPCSEL2		RW	0x1
1	SMPCSEL1		RW	0x1
0	SMPCSEL0	**通道采样速度选择** 0x0：低速 0x1：高速 [12:0]对应外部通道 AINPAD[12:0] [17:13]对应内部通道 AIN[4:0] 0=低速，对应通道的采样周期为 SSMPCYC 1=高速，对应通道的采样周期为 FSMPCYC	RW	0x1

14.6.3.3. ADC_TRGCON0/1/2/3/4/5

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15	OUTSEL	ADC 结果输出范围选择 0x0：[0,2^12-1] 0x1：[-2^11,2^11-1]	RW	0x0
14:13	Reserved	-	-	-
12:8	CHANSEL	采样通道选择： 0x0：AINPAD[0] = AMP_VOUT1 0x1：AINPAD[1] = AMP_VOUT2 0x2：AINPAD[2] = AMP_VOUT3 0x3：AINPAD[3] = PA2 0x4：AINPAD[4] = PA5 0x5：AINPAD[5] = PA8 0x6：AINPAD[6] = PA9 0x7：AINPAD[7] = PA10 0x8：AINPAD[8] = PA11	RW	0x0

		0x9：AINPAD[9] = PA12 0xA：AINPAD[10] = PA13 0xB：AINPAD[11] = PA14 0xC：AINPAD[12] = PB3 0xD：AIN[0] = 选择内部 PLL_CPOUT_ADC 0xE：AIN[1] = 选择内部 PMU_VPTAT 0xF：AIN[2] = 选择内部 PMU_VREF_BGBUF 0x10：AIN[3] = 选择内部 vdd 电压 0x11：AIN[4] = 选择内部 0.5*VCCA 电压 0x12~0x1F：不选择任何通道		
7:4	SRCSELx	通道触发源选择 0x0：epwn0_soca 0x1：epwn0_socb 0x2：epwn1_soca 0x3：epwn1_socb 0x4：epwn2_soca 0x5：epwn2_socb 0x6：epwn3_soca 0x7：epwn3_socb 0x8：timer1 0x9：timer2 0xA：timer3 0xB：timer4 0xC：timer5 0xD：adkey_trgio0 = TRG13_SEL ?timer0 :PA11; 0xE：adkey_trgio1 = PA14 0xF：software KICKSMPx 触发条件为上升沿	RW	0xC
3	TRGENx	触发通道使能 高电平有效 0x0：关闭 0x1：打开	RW	0x0
2:1	Reserved	-	-	-
0	KICKSMPx	软件使能采样 高电平有效 0x0：关闭 0x1：打开	WO	0x0

Note:当前有 3 个触发采样通道，每个寄存器配置对应的通道。优先级：TRG0 > TRG1 > TRG2

14.6.3.4. ADC_CR

Bit(s)	Name	Description	R/W	Reset
31:22	Reserved	-	-	-
21:20	CAL_CTRL1	ADC 通路 PAD[3]~PAD[12],AIN[0]~AIN[4]校准	RW	0x1

		控制位 0x0：ADC 结果 Bypass 0x1：ADC 结果做 ADC 校准，且校准参数值选择 eflash 中的初始化数据 0x3：ADC 结果做 ADC 校准，且校准参数值选择软件配置值 ADC_ANAPAR0		
19:17	CAL_CTRL0	ADC 通路 PAD[0],PAD[1],PAD[2]校准控制位 0x0：ADC 结果 Bypass 0x1：ADC 结果只做 ADC 校准，且校准参数值选择 eflash 中的初始化数据 0x3：ADC 结果做 ADC+AMP 校准，且 AMP Gain 值选择 eflash 中的初始化数据 0x7：ADC 结果做 ADC+AMP 校准，且 AMP Gain 值选择软件配置值 ADC_ANAPAR1/2	RW	0x3
16	SW_RST	软件复位模块内部状态机，高电平有效	WO	0x0
15:12	D2DCYC	两次连续采样之间的间隔时间周期，N=n+1 个 ADC clock 周期宽度	RW	0x0
11	Reserved		-	-
10:8	TRGSHIFT	外部触发延时采样（External trigger shift sample） 在触发信号产生后，延时 N 个 PCLK 的时钟周期再开始采样。 0x0：不延时 1：4 个周期 0x2：8 个周期 3：16 个周期 0x4：32 个周期 5：64 个周期 0x6：128 个周期 7：256 个周期	RW	0x0
7	SCANDIR	ADC 扫描通道顺序（ADC scan direction） 在周期扫描或者连续扫描方式时，设置扫描通道的顺序 0x0：ADC 通道选择寄存器按从低到高的顺序扫描 即： PAD[0],PAD[1]...PAD[12],IN[0],IN[2]...IN[4] 0x1:ADC 通道选择寄存器按从高到低的顺序扫描 即： IN[4],IN[3]...IN[0],PAD[12],PAD[11]...PAD[0]	RW	0x0
6	TRG13_SEL	触发源 13 选择位： 0x0：选择 IOPA11 触发 0x1：选择 time1 触发	RW	0x0
5	TESTMD	将内部 AIN[0]~AIN[4]的电压导通到 PA2，仅供测试使用，AIN[0]~AIN[4]选择由 ADC_TRGCON0[12:8]配置 0x0：test 无效，ADC 正常工作	RW	0x0

		0x1：test 有效，ADC 不工作		
4:3	ADMD	A/D 转换模式（ADC mode） 0x0：单次转换 0x1：周期扫描 0x2：连续扫描 0x3：触发模式 当改变转换模式时，软件要先禁用 ADST 位。	RW	0x0
2	ADST	ADST：A/D 转换开始（ADC start） 0x0：转换结束或进入空闲状态 0x1：转换开始 ADST 置位有下列两种方式： 在单次模式或者单周期模式下，转换完成后，ADST 将被硬件自动清除； 在连续扫描模式下，A/D 转换将一直进行，直到软件写 0 到该位或系统复位。 单次转换、周期扫描、连续扫描使用该位启动 ADC 采样	RW	0x0
1	DMAEN	DMA 使能（Direct memory access enable） 0x0：DMA 禁止 0x1：DMA 请求使能	RW	0x0
0	ADEN	A/D 转换使能（ADC enable） 0x0：禁用 0x1：使能	RW	0x0

14.6.3.5. ADC_CHS

Bit(s)	Name	Description	R/W	Reset
31:18	Reserved	-	-	-
17:0	CHxEN	**单周期/连续扫描模式下 ADC 输入通道使能** 0x0：禁用 0x1：使能 对应的通道映射： CHxEN[0]：AINPAD[0] = AMP_VOUT1 CHxEN[1]：AINPAD[1] = AMP_VOUT2 CHxEN[2]：AINPAD[2] = AMP_VOUT3 CHxEN[3]：AINPAD[3] = PA2 CHxEN[4]：AINPAD[4] = PA5 CHxEN[5]：AINPAD[5] = PA8 CHxEN[6]：AINPAD[6] = PA9 CHxEN[7]：AINPAD[7] = PA10 CHxEN[8]：AINPAD[8] = PA11 CHxEN[9]：AINPAD[9] = PA12 CHxEN[10]：AINPAD[10] = PA13 CHxEN[11]：AINPAD[11] = PA14 CHxEN[12]：AINPAD[12] = PB3	RW	0x0

		CHxEN[13]：AIN[0] = 选择内部 PLL_CPOUT_ADC CHxEN[14]：AIN[1] = 选择内部 PMU_VPTAT CHxEN[15]：AIN[2] = 选择内部 PMU_VREF_BGBUF CHxEN[16]：AIN[3] = 选择内部 vdd 电压 CHxEN[17] ： AIN[4] = 选择内部 0.5*VCCA 电压；		

14.6.3.6. ADC_IE

Bit(s)	Name	Description	R/W	Reset
31:22	Reserved	-	-	-
21:16	INTADR_SEL	**A/D 转换结束 DONE 中断入口地址选择位** 0x0：中断入口地址为 17 号 0x1：中断入口地址为 23 号 Bit16：通道 0 Bit17：通道 1 Bit18：通道 2 Bit19：通道 3 Bit20：通道 4 Bit21：通道 5	RW	0x0
15:14	Reserved	-	-	-
13	DMAFIE	**DMA 全满中断使能**	RW	0x0
12	DMAHIE	**DMA 半满中断使能**	RW	0x0
11:7	Reserved	-	-	-
6	OVRIE	overrun 中断使能	RW	0x0
5:0	TRGIE0/1/2/3/4/5	**A/D 中断使能位** 0x0：禁用 A/D 中断 0x1：使能 A/D 中断 Bit0：通道 0 Bit1：通道 1 Bit2：通道 2 Bit3：通道 3 Bit4:通道 4 Bit5:通道 5	RW	0x0

14.6.3.7. ADC_STA

Bit(s)	Name	Description	R/W	Reset
31:22	Reserved	-	-	-
21:15	CHANNELSEL	**该 7 位显示当前数据所对应的通道** n = 通道 n 的转换数据	0x0	RO
14	BUSY	**忙/空闲（Busy）** 0x0：A/D 转换器空闲 0x1：A/D 转换器忙碌	0x0	RW

13	DMAFF	**DMA 全满标志** 该标志位写‘1’清零	0x0	RW
12	DMAHF	**DMA 半满标志** 该标志位写‘1’清零	0x0	RW
11:6	OVRUN0/1/2/3/4/5	**通道数据覆盖标志位** 0x0：ADC_DATA 数据最近一次转换结果 0x1：ADC_DATA 数据被覆盖 新的转换结果装载至寄存器之前，若 DATA[15:0]的数据没有被读取，OVERRUN 将置‘1’ 该标志位写‘1’清零 Bit6：通道 0 Bit7：通道 1 Bit8：通道 2 Bit9：通道 3 Bit10：通道 4 Bit11：通道 5	0x0	RW
5:0	DONE0/1/2/3/4/5	**A/D 转换结束标志位** 该位由硬件在通道组转换结束时设置，由软件清除。 0x0：A/D 转换未完成 0x1：A/D 转换完成 该标志位写‘1’清零 Bit0：通道 0 Bit1：通道 1 Bit2：通道 2 Bit3：通道 3 Bit4：通道 4 Bit5：通道 5	0x0	RW

Note:当 DONE 信号清掉时，硬件认为软件已经读走 ADC_DATA，OVRUN 不会置 1。

14.6.3.8. ADC_DATA0

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15:0	DATA0	A/D 转换结果（Transfer data） 单次采样、周期扫描、连续扫描以及 触发通道 0 的数据存放在该寄存器	RW	0x0

14.6.3.9. ADC_DATA1/2/3/4/5

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15:0	DATA	A/D 转换结果（Transfer data） 触发通道 1/2/3/4/5 的数据存放在该寄存器	RW	0x0

14.6.3.10. ADC_DMAADR

Bit(s)	Name	Description	R/W	Reset
31:0	DMAADR	DMA 起始地址，byte 地址	RW	0x0

14.6.3.11. ADC_DMACNT

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	dmadatcnt	已经 DMA 完成的 ADC 数据个数	RO	0x0

14.6.3.12. ADC_DMALEN

Bit(s)	Name	Description	R/W	Reset
31:12	Reserved	-	-	-
11:0	dmalen	配置 dma buffer 长度：N=n+1 个 ADC 数据	RW	0x0

14.6.3.13. ADC_ANACON

Bit(s)	Name	Description	R/W	Reset
31:13	Reserved	-	-	-
12	VOLTAGE_SEL	**ADC 校准电源电压选择** 0x0：3.3V 0x1：5V	RW	0x1
11:10	TENSEL_VDD<1:0>	**测试通路选择信号** TENSEL=2’bxx 0x0：选择 VCM 到 TOT 0x1：选择 VREFP 到 TOT 0x2：选择输入信号通路到 TOT 0x3：均不选择	RW	0x3
9	CMPBSSEL_VDD	**比较器偏置电流选择信号** 0x0：1X 0x1：1.5X	RW	0x0
8:7	BIASSEL_VDD<1:0>	**内部偏置电流选择信号** BIASSEL=2’bxx 0x0：1.33X 0x1：1X 0x2：1X 0x3：0.8X	RW	0x2
6:4	VREFSEL_VDD<2:0>	**内部 LDO 档位选择信号** VREFSEL_VDD<2:0>=3’bxxx	RW	0x0

		3’b000~3’b111 分别对应 1.5~4.3V，步长为 0.4V		
3	SELINREF_VDD	**选择内部 VREFP 使能信号** 0x0：选择内部 REF 0x1：选择外部 REF	RW	0x0
2	VCMEN_VDD	**内部 VCMBUFFER 模块使能信号** 0x0：关闭 0x1：打开	RW	0x0
1	CMPEN_VDD	**比较器模块使能信号** 0x0：关闭 0x1：打开	RW	0x0
0	BIASEN_VDD	**偏置电流模块使能信号** 0x0：关闭 0x1：打开	RW	0x0

14.6.3.14. ADC_ANAPAR0

Bit(s)	Name	Description	R/W	Reset
31:28	Reserved	-	-	-
27:16	OFFSET	ADC 校准中的参数 Offset（有符号） -(c*b)/a	RW	0x0
15:0	GAIN	ADC 校准中的参数 Gain（无符号） (a/c)*2^15	RW	0x0

Note: 校准公式 ADCDATA/GAIN + OFFSET，GAIN 范围[2^14,2^16-1]

14.6.3.15. ADC_ANAPAR1

Bit(s)	Name	Description	R/W	Reset
31:29	Reserved	-	-	-
28:16	OFFSET0	AINPAD[0]（AMP+ADC）校准中的参数 Offset（有符号） -(D(Vcmm)*(Gaini/Gainm - D(Vcmi) + D(Vos)*(Gaini/Gainm))- D(Vos)*Gaini	RW	0x0
15:0	GAIN	AMP+ADC 校准中的参数 Gain（无符号） （Gainm/Gaini)*2^15	RW	0x0

Note: 校准公式 ADCDATA/GAIN + OFFSET，GAIN 范围[2^14,2^16-1]

14.6.3.16. ADC_ANAPAR2

Bit(s)	Name	Description	R/W	Reset
31:26	Reserved			
25:13	OFFSET2	AINPAD[2]（AMP+ADC）校准中的参数 Offset（有	RW	0x0

		符号） -(D(Vcmm)*(Gaini/Gainm - D(Vcmi) + D(Vos)*(Gaini/Gainm))- D(Vos)*Gaini		
12:0	OFFSET1	AINPAD[1]（AMP+ADC）校准中的参数 Offset（有符号） -(D(Vcmm)*(Gaini/Gainm - D(Vcmi) + D(Vos)*(Gaini/Gainm))- D(Vos)*Gaini	RW	0x0

15. TIMER

15.1. 简介

TX32M2300 系列一共包含 6 个普通 timer 和一个 watchdog。其中 timer0/1/2/3/5 为 16 位 timer，timer4 为 32 位 timer。

定时器 timer0/1/2/3/5 由一个 16 位的自动装载计数器组成，它由一个可编程的预分频器驱动。

它使用于多用途，包含测量输入信号的脉冲宽度（输入捕获），或者产生输出波形（输出比较、PWM 等）。

15.2. 主要特性

- 16 位递增计数器
- 可编程（可以实时修改）预分频器
- 支持选择 GPIO 作为计数时钟源
- 支持不同计数时钟源
- 允许在每次计数器周期之后更新定时周期寄存器
- 支持输入捕获功能，
  - 最多支持同时保存 4 个捕获事件指针

- 每个捕获事件极性独立可配置
- 每次捕获事件发生可配置是否复位计数值

- PWM 输出
- 支持周期和占空比 auto-reload
- 使用外部信号控制定时器和定时器互联的同步电路

15.3. 功能描述

15.3.1. 时基单元

可编程高级控制定时器的主要部分是一个 16 位计数器和与其相关的自动装载寄存器。这个计数器可以向上计数。此计数器时钟由预分频器分频得到。

计数器、自动装载寄存器和预分频器寄存器可以由软件读写，即使计数器还在运行读写仍然有效。

时基单元包括：

- 计数器寄存器（TIMERx_CNT）
- 周期寄存器（TIMERx_RD）
- 分频寄存器（TIMERx_CON[10:8]）

自动装载寄存器是预先装载的，写或读自动重装载寄存器将访问预装载寄存器。预装载寄存器的内容在每次的更新事件（计数值等于周期值）时传送到影子寄存器，并且产生溢出事件。

在定时器模式下，计数器从 0 计数到周期值（TIMERx_PRD），然后重新从 0 开始计数，并且产生一个计数器溢出中断。

分频寄存器可以将计数器的时钟频率按 2 的 n 次方分频，从而提高计数的最大周期。

15.3.2. 计数源选择

计数器时钟可由下列时钟源提供：

- 系统时钟
- 内部高速 RC
- 外部低速 RC
- 外部晶振
- 外部 GPIO

通过配置 TIMERx_CTL 寄存器的 inc_src_sel 选择不同的计数源，配置 TIMERx_CTL 的 psc 配置不同的分频系数。

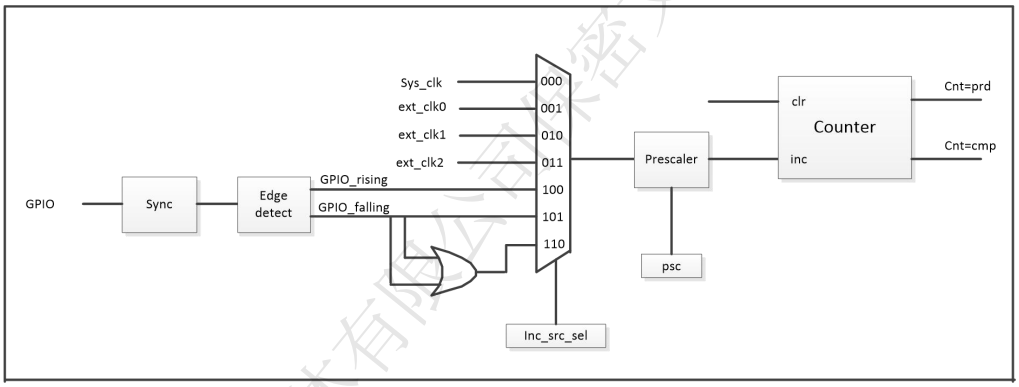


图 15-1 TImer 的结构图

15.3.3. 输入捕获源

输入捕获支持外部 GPIO 触发，内部比较器触发以及外部 3 个 GPIO 异或一起作为捕获源。

输入捕获通道：

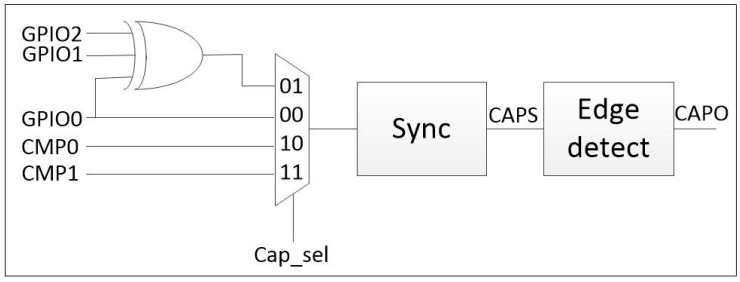


图 15-2 捕获结构图

上图中 pin 对应 GPIO 输入：

	CAP （Cap_sel==0x0）	XOR CAP PIN(Cap_sel==0x1)
TIMER0	PA6/A10	T0 ^ T1 ^ T2
TIMER1	PA3	T0 ^ T1 ^ T2
TIMER2	PE0/PA9	T0 ^ T1 ^ T2
TIMER3	PA13	T0 ^ T1 ^ T3
TIMER4	PA0	PA0 ^ PA1 ^ PA2
TIMER5	PA15/PA10	T0 ^ T1 ^ T5
备注：T0/T1/T2/T3/T5 分别指 TIMER0/1/2/3/5 的一根 CAP PIN		

cmp0 和 cmp1 分别是比较器 0/1 的比较输出信号。首先通过一个选择器选择输入触发通路后，经过同步器同步得到 CAPS，然后边沿检测器产生边沿触发信号，最后通过一个选择器选出最终捕获信号 CAP0。当捕获信号有效时，自动把当前计数器的值存入 TIMx_CAPx 寄存器，并且产生捕获中断信号。

15.3.4. 输入捕获模式

当捕获事件发生时，把当前计数值保存到对应捕获寄存器寄存器（TNRx_CAP1/2/3/4）。

通过配置寄存器 TIMERx_CTL 的 cap_num 可以配置使用 1~4 个捕获寄存器，例如使用 3 个寄存器，则第 1 次捕获事件产生，捕获值保存在 TIMERx_CAP1，第 2 次捕获事件产生，捕获值保存在 TIMERx_CAP2，第 3 次捕获事件产生，捕获值保存在 TIMERx_CAP3，第 4 次捕获事件产生，捕获值保存在 TIMERx_CAP1，依次循环。

通过配置寄存器 TIMERx_CTL 的 capxpol 可以独立配置每个捕获事件极性，选择上升沿或者下降沿捕获。

通过配置寄存器 TIMERx_CTL 的 ctrrst1/2/3/4 可以独立配置对应的捕获事件发生时，是否复位计数器的值。

每次捕获事件发生，都会产生对应的捕获标志（TIMERx_FLAG）。通过配置寄存器 TIMERx_IE 可以独立配置每次捕获事件发生是否产生中断。

捕获模式下，支持计数值溢出中断。该模式下，计数周期固定为 16’hffff。当计数值达到 16’hffff，会产生溢出标志 ovf_flag，通过配置 TIMERx_IE 寄存器可以产生溢出中断。

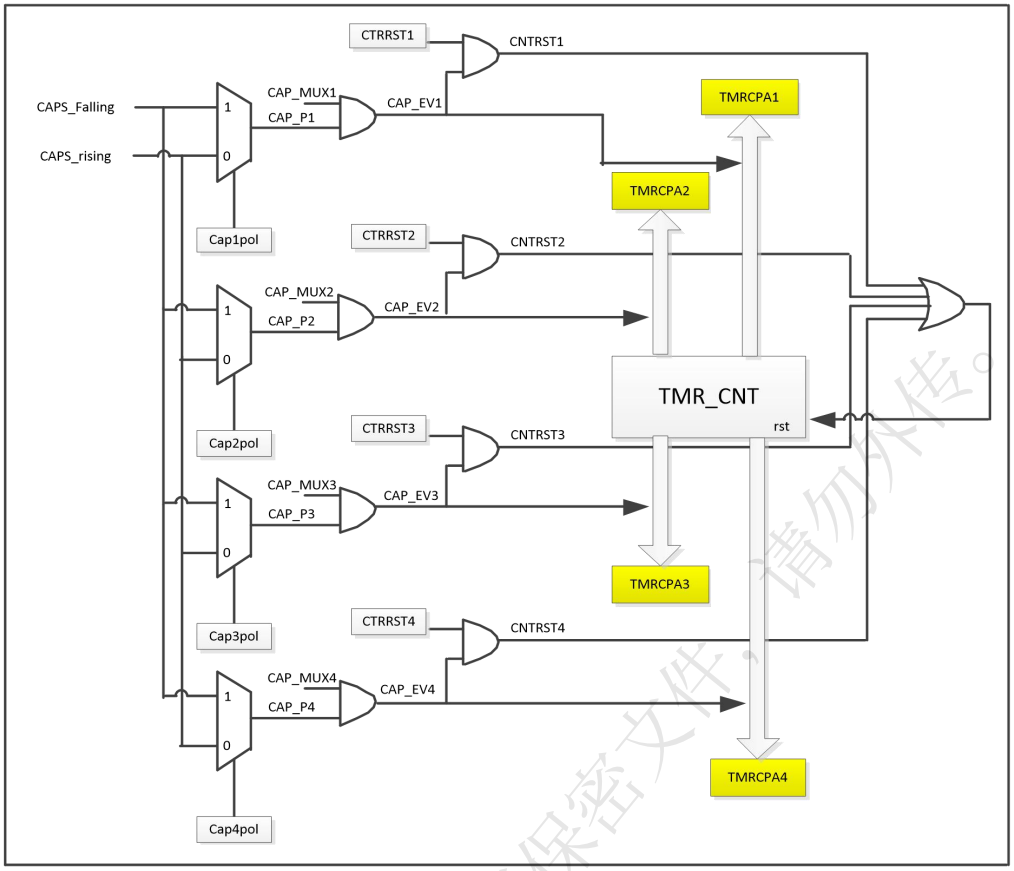


图 15-3 结构框图 1

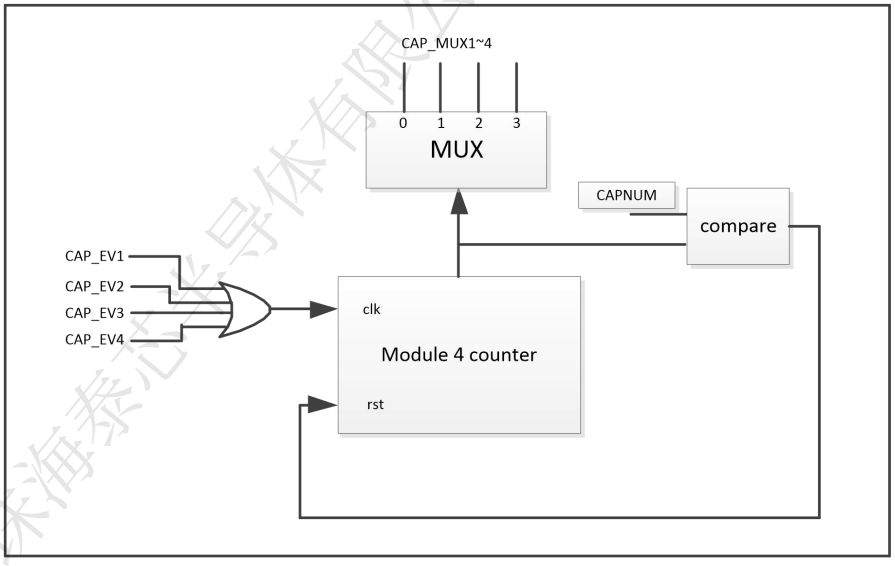


图 15-4 结构框图 2

15.3.5. PWM 模式

PWM 工作模式可以产生一个又 TIMERx_PRD（TIMERx_CAP1）寄存器确定周期，TIMERx_CMP

（TIMERx_CAP2）寄存器确定占空比的信号。

当使用 PWM 模式时，TIMERx_CAP3 为 TIMERx_CAP1 的影子寄存器，TIMERx_CAP4 为 TIMERx_CAP2 的影子寄存器。每次计数值 TIMER_CNT 达到 TIMERx_RPD 时，自动把 TIMERx_CAP3 的值赋值到 TIMERx_CAP1，把 TIMERx_CAP4 的值赋值到 TIMERx_CAP2。

写 TIMERx_CAP1 寄存器，会自动写相同值到 TIMERx_CAP3 寄存器。写 TIMERx_CAP2 寄存器，会自动写相同值到 TIMERx_CAP4 寄存器。

当计数值达到 TIMERx_CMP 值或计数值达到 TIMERx_PRD 值时，会产生对应标志位，如果设置了对应的中断使能位，会产生中断。可以在中断中修改 preload 寄存器（TIMERx_CAP3/4）的值，这样下一次计数达到周期值时，就会自动加载预配置的值到周期寄存器和比较器寄存器。

提前配置好对应 PWM 输出 GPIO 对应的复用功能选择，在 TIMERx_PRD 和 TIMERx_CMP 寄存器中配置所需的 PWM 周期和占空比，配置 TIMx_CTL 寄存器上的 tmr_mode=0x2，然后启动 PWM 输出。这样，对应 IO 上就会产生期望的 PWM 波形信号，直到软件配置关闭 PWM 输出。

15.3.6. 触发 ADC 采样

Timer 还有一个增强型的功能，内部把 synco 连接到 ADC 触发源，触发 ADC 采样。具体参考 ADC 章节。

15.3.7. timer 同步输出

输出同步信号可选择：

- 同步输入 SYNCI
- 计数值等于 TIMERx_PRD
- 计数值等于 TIMERx_CMP

15.3.8. 从模式

TIMER 定时器能够在外部模式下和一个外部的触发 SYNCI 同步：复位模式、触发模式和门控模式。

15.3.9. 复位模式

在发生一个触发输入事件时，计数器和它的预分频器能够重新被初始化；例如，计数器在正常运转，SYNCI 出现一个上升沿，此时计数器被清零然后从 0 重新开始计数。同时触发标志（TIMERx_FLAG 中的）被设置，根据 TIMERx_IE 寄存器中的 slave_ie 的设置，产生中断。

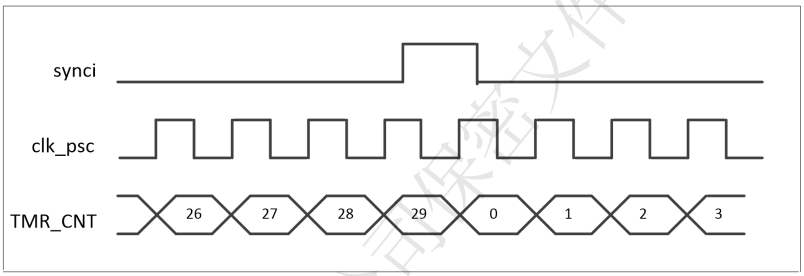


图 15-5 复位时序

15.3.10. 触发模式

计数器的使能依赖于同步输入端 SYNCI 的事件。

如下例子，当 SYNCI 出现一个上升沿时，计数器开始在内部时钟驱动下计数，同时设置 slave_flag 位，并且根据 TIMERx_IE 寄存器中的 slave_ie 的设置，产生中断。

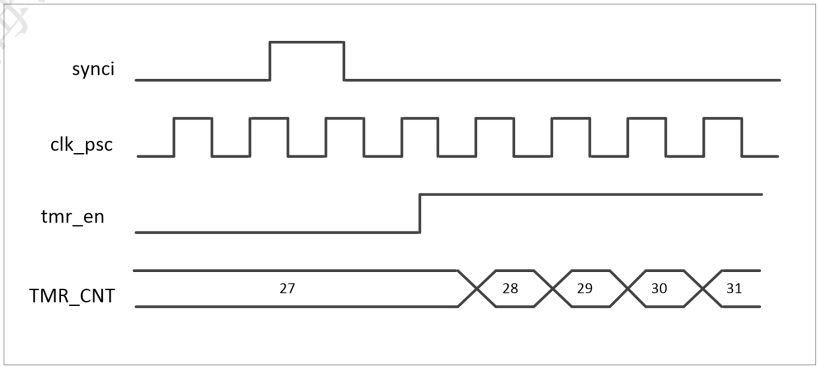


图 15-6 触发时序图

15.3.11. 门控模式

计数器的使能依赖于输入同步信号 SYNCI 的电平。在 SYNCI 有效电平时，TIMERx_CNT 计数，无效电平，停止计数。

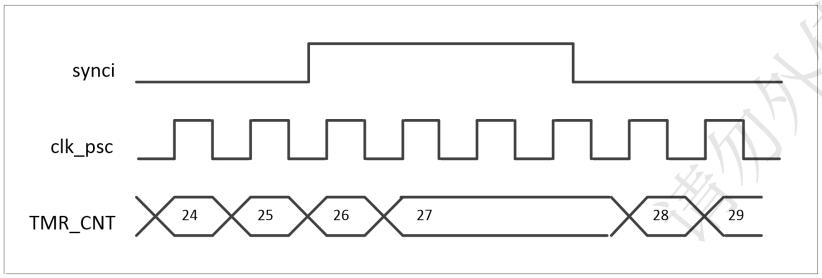


图 15-7 门控时序图

15.3.12. DMA 传输模式

15.3.12.1. DMA 产生 PWM

TIMER 支持 DMA 传输，可以用来先把 PWM 的周期和占空比提前存放到 SRAM 指定位置，然后每个周期硬件自动从 SRAM 中 load 会数据，修改 PWM 的周期和占空比，产生期望的 PWM 信号。

具体流程如下：

1) 提前把期望产生的 PWM 周期和占空比写到 SRAM。
2) 配置 TIMERx_DADR 和 TIMERx_DLEN 寄存器，配置 DMA 的起始地址和长度。
3) 配置 TIMERx_DCTL 寄存器的 dma_lpbk，配置采用循环或者单次模式。如果 dma_lpbk=0，在执行完指定 dma 长度之后，PWM 自动停止输出。如果 dma_lpbk=1，在执行完指定 dma 长度之后，自动从起始地址从新开始取数据，直到软件把 dma_en 关闭，才停止。
4) 通过配置 TIMERx_DCTL 寄存器的 dma_en，使能 dma 传输模式。

15.3.12.2. 捕获数据通过 DMA 写到 SRAM

支持把捕获数据自动保存到 SRAM 中。

1) 将 TIMERX_DCTL 清零。

2) 配置 TIMERx_DADR 和 TIMERx_DLEN 寄存器，配置 DMA 的起始地址和长度。

3) 配置 TIMERx_IE 寄存器的 dma_fl_ie，使能 DMA 完成中断。

4) 配置并启动捕获功能。

5) 中断之后，在 SRAM 中获取之前的捕获数据。

15.3.12.3. DMA 中断标志

支持 DMA buffer 半满中断和全满中断，通过 TIMERx_IE 寄存器的 dma_hf_ie 和 dma_fl_ie 使能中断。

由于面积限制，dma 功能暂时没有添加，所以上面所说的 dma 功能没有，后续可能考虑添加到某个 timer。

15.3.13. 多个 timer 之间级联

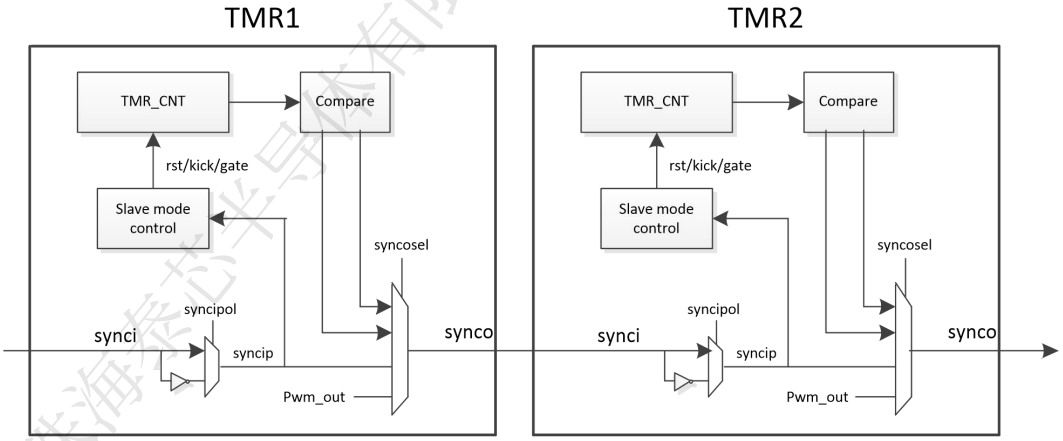


图 15-8 timer 多个级联

15.3.14. 多个 timer 计数值同时清零

支持把所有 timer 的计数值同时清零。下面例子，把两个 TIMER 的计数值同时清零，同

步配置如下：

1) 配置 TIMER0_CTL 寄存器的 syncipol = 0，
2) 配置 TIMER0_CTL 寄存器的 syncosel = 2，直接把 TIMER0 的 synci 连接到 TIMER0 的 synco。
3) 配置 TIMER0_CTL 寄存器的 slave_mode = 2，复位模式。
4) 配置 TIMER1_CTL 寄存器的 syncipol = 0。
5) 配置 TIMER1_CTL 寄存器的 slave_mode = 2，复位模式。
6) 当在 synci 上产生一个有效边沿跳变，就会同时复位 TIMER0 和 TIMER1 的计数值。

15.3.15. 产生带载波的 PWM 信号

一个 timer 作为后面 timer 的载波；具体例子如下：

1) 把 TIMER1 配置为 PWM 输出模式（低频）；
2) 配置 TIMER1_CTL 寄存器的 syncosel，选择 PWM_OUT 输出到 synco。
3) 配置 TIMER2_CTL 寄存器的 slave_mode = 3，门控模式。
4) 配置 TIMER2 位 PWM 输出模式（高频）。
5) 配置 TIMER1 和 TIMER2 使能。
6) 这样，就能产生带载波的 PWM 信号。

15.4. 寄存器

15.4.1. 寄存器基地址

Name	Base Address	Description
TIMER0	0x40001100	TIMER0 基地址
TIMER1	0x40001200	TIMER1 基地址
TIMER2	0x40001300	TIMER2 基地址
TIMER3	0x40001400	TIMER3 基地址
TIMER5	0x40001600	TIMER5 基地址

15.4.2. 寄存器列表

Offset Address	Name	Description
0x0000	TIMERx_CON	控制寄存器
0x0004	TIMERx_EN	使能寄存器
0x0008	TIMERx_IE	中断寄存器
0x000c	TIMERx_FLG	状态寄存器
0x0010	TIMERx_CLR	清除寄存器
0x0014	TIMERx_CNT	计数寄存器
0x0018	TIMERx_CAP1	数据寄存器 1
0x001c	TIMERx_CAP2	数据寄存器 2
0x0020	TIMERx_CAP3	数据寄存器 3
0x0024	TIMERx_CAP4	数据寄存器 4
0x0028	TIMER5_DCTL	TIMER5 DMA 控制寄存器
0x002c	TIMER5_DADR	TIMER5 DMA 地址寄存器
0x0030	TIMER5_DLEN	TIMER5 DMA 长度寄存器
0x0034	TIMER5_DCNT	TIMER5 DMA 计数寄存器
0x0038	TIMER_ALLCON	TIMER 总寄存器

15.4.3. 寄存器详细说明

15.4.3.1. TIMERx_CON

Bit(s)	Name	Description	R/W	Reset
31:26	reserved	-	-	-
25	pwmpol	**PWM 极性选择**	RW	0
24	cap4pol	**capture 事件 4 的极性选择** 0x0：上升沿 0x1：下降沿 GPIO XOR 时，该位无效，capture 事件 4 默认边沿检测	RW	0
23	cap3pol	**capture 事件 3 的极性选择** 0x0：上升沿 0x1：下降沿 GPIO XOR 时，该位无效，capture 事件 3 默认边	RW	0

		沿检测		
22	cap2pol	**capture 事件 2 的极性选择** 0x0：上升沿 0x1：下降沿 GPIO XOR 时，该位无效，capture 事件 2 默认边沿检测	RW	0
21	cap1pol	**capture 事件 1 的极性选择** 0x0：上升沿 0x1：下降沿 GPIO XOR 时，该位无效，capture 事件 1 默认边沿检测	RW	0
20	ctrrst4	**capture 事件 4 发生时，CNT 的值自动清零**	RW	0
19	ctrrst3	**capture 事件 3 发生时，CNT 的值自动清零**	RW	0
18	ctrrst2	**capture 事件 2 发生时，CNT 的值自动清零**	RW	0
17	ctrrst1	**capture 事件 1 发生时，CNT 的值自动清零**	RW	0
16:15	cap_cnt	**捕获功能模式选择** 0x0：data store in CAP1 0x1：data store in CAP1 CAP2 0x2：data store in CAP1 CAP2 CAP3 0x3：data store in CAP1 CAP2 CAP3 CAP4	RW	0
14:13	cap_sel	**捕获方式** 0x0：GPIO 0x1：GPIO XOR 0x2：compare0 output 0x3：compare1 output	RW	0
12:11	syncosel	**输出同步信号选择** 0x0：CNT 值=PRD 值 0x1：CNT 值=CMP 值 0x2：把 SYNCI 值输出到 SYNCO 0x3：PWM 输出赋值到 SYNCO	RW	
10	syncipol	**synci 极性取反** 0x0：不取反 0x1：取反	RW	0
9:8	slave_mode	**synci 功能选择** 0x0：disable 0x1：kick start 0x2：reset 0x3：gating	RW	0
7:5	psc	**Timer 预分频设置** 0x0：0 分频 0x1：2 分频 0x2：4 分频 0x3：8 分频 0x4：16 分频 0x5：32 分频 0x6：64 分频	RW	0

		0x7：128 分频		
4:2	Inc_src_sel	**Timer 计数源选择** 0x1：内部低速 RC 2 分频（32K/2） 0x2：内部高速 RC 2 分频（26M/2） 0x3：外部晶振 2 分频时钟 0x4：timer inc pin rising 0x5：timer inc pin falling 0x6：timer inc pin rising 和 falling Others：系统时钟	RW	0
1:0	Mode_sel	**Timer 模式选择** 0x0：timer counter mode 0x1：timer pwm mode 0x2：timer capture mode Others：保留	RW	0

15.4.3.2. TIMERx_EN

Bit(s)	Name	Description	R/W	Reset
31:1	reserved	-	-	-
0	tmren	TIMER 使能信号，高电平有效	RW	0x0

15.4.3.3. TIMERx_IE

Bit(s)	Name	Description	R/W	Reset
31:10	reserved	-	-	-
9	dma_fl_ie	dma buffer 全满中断使能	RW	0x0
8	dma_hf_ie	dma buffer 半满中断使能	RW	0x0
7	slave_ie	slave mode 的触发模式或者复位模式中断使能	RW	0x0
6	cmp_ie	CNT 值等于 CMP 值时，中断使能， 仅在 pwm mode 时有效	RW	0x0
5	prd_ie	CNT 值等于 PRD 值时，中断使能， 仅在 counter mode/PWM mode 时有效	RW	0x0
4	ovf_ie	CNT 值溢出（16’hffff）时，中断使能	RW	0x0
3	cap4_ie	capture 事件 4 发生时，中断使能	RW	0x0
2	cap3_ie	capture 事件 3 发生时，中断使能	RW	0x0
1	cap2_ie	capture 事件 2 发生时，中断使能	RW	0x0
0	cap1_ie	capture 事件 1 发生时，中断使能	RW	0x0

15.4.3.4. TIMERx_FLG

Bit(s)	Name	Description	R/W	Reset
31:10	reserved		-	-
9	dma_fl_flg	dma buffer 全满标志	RO	0x0

8	dma_hf_flg	dma buffer 半满标志	RO	0x0
7	slave_flg	slave mode 发生标志（仅复位或触发）	RO	0x0
6	cmp_flg	CNT 值等于 CMP 值标志， 仅在 pwm mode 时有效	RO	0x0
5	prd_flg	CNT 值等于 PRD 值标志， 仅在 counter mode/PWM mode 时有效	RO	0x0
4	ovf_flg	CNT 值溢出（16’hffff）标志	RO	0x0
3	cap4_flg	capture 事件 4 发生标志	RO	0x0
2	cap3_flg	capture 事件 3 发生标志	RO	0x0
1	cap2_flg	capture 事件 2 发生标志	RO	0x0
0	cap1_flg	capture 事件 1 发生标志	RO	0x0

15.4.3.5. TIMERx_CLR

Bit(s)	Name	Description	R/W	Reset
31:10	reserved	-	-	-
9	dma_fl_clr	dma buffer 全满标志清除	WO	0x0
8	dma_hf_clr	dma buffer 半满标志清除	WO	0x0
7	slave_clr	slave mode 发生标志清除	WO	0x0
6	cmp_clr	CNT 值等于 CMP 值标志清除	WO	0x0
5	prd_clr	CNT 值等于 PRD 值标志清除	WO	0x0
4	ovf_clr	CNT 值溢出（16’hffff）标志清除	WO	0x0
3	cap4_clr	capture 事件 4 发生标志清除	WO	0x0
2	cap3_clr	capture 事件 3 发生标志清除	WO	0x0
1	cap2_clr	capture 事件 2 发生标志清除	WO	0x0
0	cap1_clr	capture 事件 1 发生标志清除	WO	0x0

15.4.3.6. TIMERx_CNT

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	-	-	-
15:0	CNT	计数寄存器	RW	0x0

15.4.3.7. TIMERx_CAP1

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15:0	CAP1/PR	捕获模式：捕获寄存器 1 定时模式、PWM 模式：计数周期寄存器	RW	0xFFFF

15.4.3.8. TIMERx_CAP2

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved		-	-
15: 0	CAP2/CMP	捕获模式：捕获寄存器 2 定时模式、PWM 模式：比较寄存器	RW	0x0

15.4.3.9. TIMERx_CAP3

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15: 0	CAP3/PR_SD	捕获模式：捕获寄存器 3 定时模式、PWM 模式：计数周期影子寄存器	RW	0xFFFF

15.4.3.10. TIMERx_CAP4

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15: 0	CAP4/CMP_SD	捕获模式：捕获寄存器 4 定时模式、PWM 模式：比较影子寄存器	RW	0x0

15.4.3.11. TIMER5_DCTL

Bit(s)	Name	Description	R/W	Reset
31:2	Reserveds			
1	dma_lpbk	dma 模式选择 0：单次模式，指定 dma 长度完成之后，disable TIMER 1：循环模式，指定 dma 长度完成之后，重新从起始地址开始	RW	0x0
0	dma_en	dma 使能	RW	0x0

15.4.3.12. TIMER5_DADR

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15: 0	dma_stadr	dma 起始地址（word 对齐）	RW	0x0

15.4.3.13. TIMER5_DLEN

Bit(s)	Name	Description	R/W	Reset

31:16	Reserved			
15: 0	dma_len	Dma buffer 长度（32bit），如果 buffer 为 n，配置为 n-1；	RW	0x0

15.4.3.14. TIMER5_DCNT

Bit(s)	Name	Description	R/W	Reset
31:16	Reserved	-	-	-
15: 0	dma_cnt	dma 数据有效个数	RW	0x0

15.4.3.15. TIMER_ALLCON

Bit(s)	Name	Description	R/W	Reset
31:14	Reserved	-	-	-
13	tmr5_sync	timer5 计数值清零	RO	0x0
12	Reserved	-	-	-
11	tmr3_sync	timer3 计数值清零	RO	0x0
10	tmr2_sync	timer2 计数值清零	RO	0x0
9	tmr1_sync	timer1 计数值清零	RO	0x0
8	tmr0_sync	timer0 计数值清零	RO	0x0
7:6	reserved	-	-	-
5	tmr5_kick	timer5 启动计数	RO	0x0
4	reserved	-	-	-
3	tmr3_kick	timer3 启动计数	RO	0x0
2	tmr2_kick	timer2 启动计数	RO	0x0
1	tmr1_kick	timer1 启动计数	RO	0x0
0	tmr0_kick	timer0 启动计数	RO	0x0

16. EPWM

16.1. 简介

EPWM 外设在许多商业及工业级电力电子系统控制中是一个关键模块。这些系统包括数字电机控制、电源开关模式控制、不间断电源 UPS 和其他形式的电源转换模块。EPWM 有时可作为 DAC 使用，占空比即为 DAC 输出的模拟量。

在 TX32M2300 中一共有 4 个 EPWM 模块，每个 EPWM 模块都支持以下功能：

1. 专用的 16 位时间控制器，可用于周期和频率的控制。
2. 两个 PWM 输出（EPWMxA 和 EPWMxB）可进行以下配置：
   - 两个独立的 PWM 输出进行单边控制；
   - 两个独立的 PWM 输出进行双边对称控制；
   - 一个独立的 PWM 输出进行双边不对称控制；
3. 通过软件对 PWM 信号进行异步覆盖控制。
4. 每个 EPWM 带有 CMPA,CMPB,CMPC,CMPD 共 4 个事件触发寄存器，可用于触发 PWM 翻转和 ADC 采样。
5. 各 EPWM 可与其他 EPWM 模块进行可编程的超前或滞后相位控制。
6. 可在每个周期对各个 EPWM 进行硬件锁定（同步）相位关系。
7. 具有独立的上升沿和下降沿死区延迟控制。
8. 可编程故障区（trip zone）用于故障时的周期循环（cycle-by-cycle trip）控制和单次（one-shot trip）控制。
9. 可通过 CPU，IO 或者比较器使 PWM 输出为强制高、低或高阻抗逻辑电平。
10. EPWM 模块中的所有事件都可以触发 CPU 中断和启动 ADC 开始转换（ADC SOC）。
11. 可编程事件有效降低了中断时 CPU 的负担。

。

16.2. 子模块介绍

16.2.1. 时基计数器子模块（Time-base）

时基计数器子模块有如下配置选择：

1. 标定与系统时钟有关的时基计数器时钟；
2. 配置 PWM 时基计数器的频率和周期；
3. 设置时基计数器的模式：
   - 递增计数
   - 递减计数
   - 增减计数

4. 配置与另一 EPWM 模块的时基相位关系；
5. 通过软件或硬件同步不同模块之间的时基计数器；
6. 在同步事件发生后配置时基计数器的方向（递增、递减或增减计数）；
7. 指定 EPWM 模块的同步输出信号源；
   - 同步输入信号
   - 时基计数器等于 0
   - 时基计数器等于 CMPA
   - 时基计数器等于 CMPB
   - 没有输出同步信号产生

16.2.2. 时基计数比较器子模块（Counter-comparator）

1. 指定 EPWMxA 和 EPWMxB 上输出的 PWM 波的占空比；
2. 指定 EPWMxA 和 EPWMxB 上开关转换的时间；
3. 产生事件触发信号，包括：
   - 时基计数器等于 0
   - 时基计数器等于周期值
   - 时基计数器等于 CMPA
   - 时基计数器等于 CMPB
   - 时基计数器等于 CMPC

16.2.3. 动作产生子模块（Action-qualifier）

动作产生子模块可以将事件设置转换为各种动作类型，从而在 EPWMxA 和 EPWMxB 输出需要的波形。

1. 动作产生子模块具有以下作用：产生动作（置 1、清 0、反转）；
2. 通过软件强制控制 PWM 输出状态；

16.2.4. 死区产生子模块（Dead-band）

1. 控制开关的传统互补死区关系；
2. 指定输出上升沿延时值；
3. 指定输出下降沿延时值；
4. 忽略死区模块，这种情况下 PWM 波形无任何变化地通过；

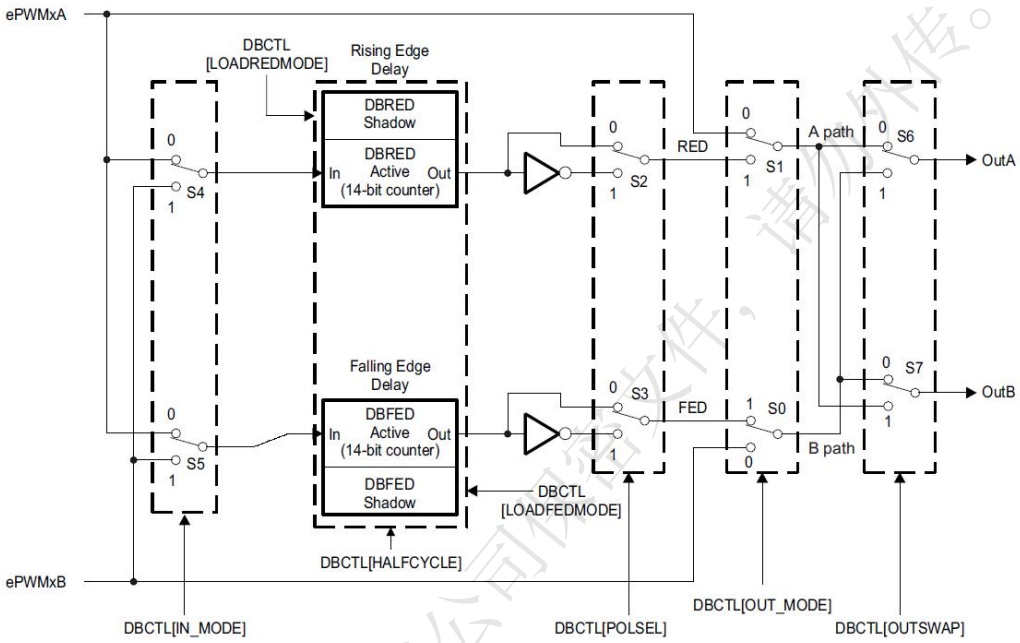


图 16-1 死区结构框图

16.2.5. 事件触发子模块（Event-trigger）

事件触发子模块主要功能如下：

1. 接收由时基计数器或时基计数比较器子模块产生的事件输入；
2. 通过时基计数器方向限定事件 up/down；
3. 在下列情况下使用计数逻辑产生中断请求和启动 adc 转换：
   - 每次事件
   - 每两次事件
   - 每三次事件
4. 通过事件计数器和标志提供事件产生的可视化；
5. 允许软件强制产生中断和启动 ADC 转换；
6. 当事件产生时事件触发子模块管理时基子模块的事件的产生和计数比较子模块产

生 CPU 中断，产生 ADC 开始转换的脉冲；

16.2.6. 故障区子模块（Trip-zone）

Trip-Zone 子模块的主要功能如下：

1. 故障区输入信号 TZ1 到 TZ6 可以灵活地映射到任何一个 EPWM 模块；
2. 根据故障情况，EPWMxA 和 EPWMxB 输出可以被强制输出为下列信号：
   - 强制输出高电平
   - 强制输出低电平
   - 强制输出高阻抗
   - 没有动作发生
3. 支持在短路或过流情况下的单次 trip；
4. 支持周期循环（CBC）限流操作；
5. 每个 Trip-zone 输入引脚都可以被分派为单次或周期循环的操作；
6. 在任何一个 Trip-zone 引脚上都有可能发生中断；
7. 支持软件强制 trip；
8. 如果不需要，可以忽略 Trip-zone 子模块；

16.3. 功能描述及常用拓扑

EPWM 间的同步功能和主要配置选项如下：

1. 选择同步（Syncin）信号源；
2. 使能同步选通脉冲开关，出现同步脉冲时会将相位寄存器中的数值加载进计数器；
3. 也可不做任何操作，关掉同步选通脉冲开关；
4. SyncOut 的连接选择：
   - Sync flow-through ——将 SyncOut 直接与 SyncIn 相连；
   - Master 模式下，在 PWM 边界提供一个 sync 信号——SyncOut 与 CTR == 0 连接；
   - Master 模式下，在任意可编程时间点下提供一个 sync 信号——SyncOut 与 CTR == CMPA 相连或者 CTR == CMPB 相连；
   - 模块是单独一个模式并且和其他模块不同步时——SyncOut 与 X 相连（不使能）；

最常见的两种——Master 模式和 Slave 模式——如下图所示：

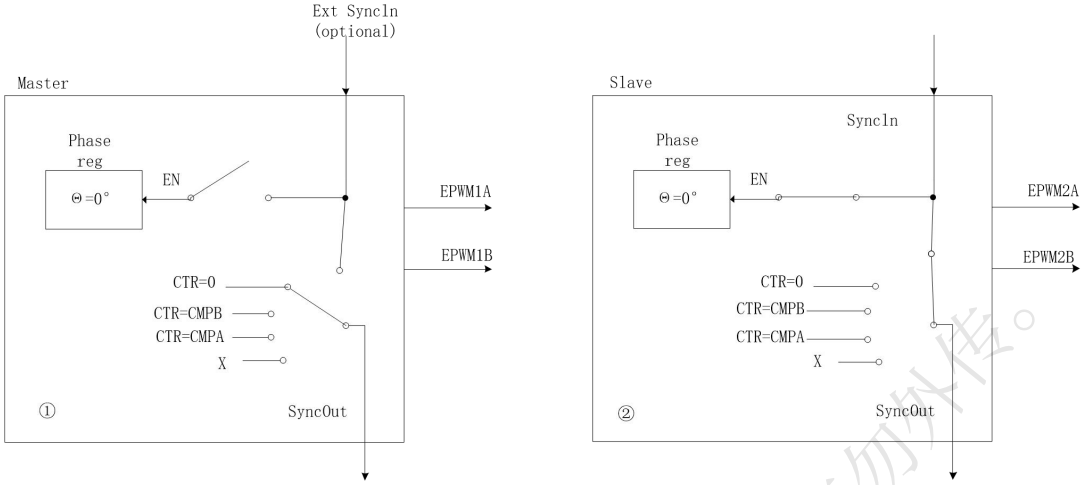
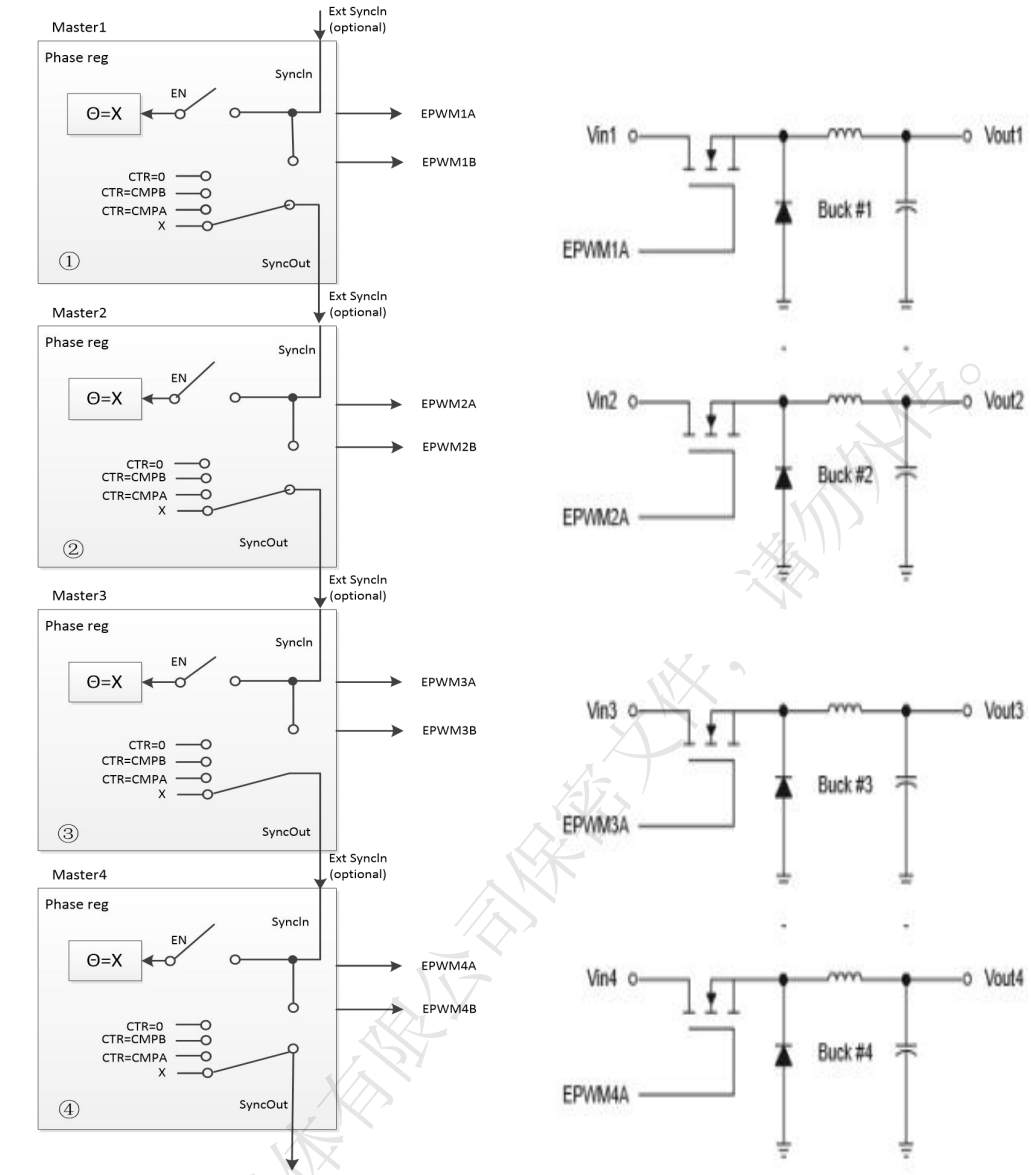


图 16-2 同步示意图

16.3.1. 独立频率控制多个 Buck 转换电路

最简单的功率变换电路拓扑之一是 buck。将一个 EPWM 模块配置为 master 时，可以用一个频率控制两个 buck 级。如果要求每个 buck 电路用独立频率控制，那么每个 converter 级都需要分配一个 EPWM 模块。下图显示了 4 个 buck 级及波形，每个都工作在独立频率上。在这种情况下，所有的四个 EPWM 模块则都被配置为 Master 并且均不同步。注意即使有四个 buck 级，但是一共只有三种波形。



Note:Θ = X indicates value in phase register is a “don’t care”

图 16-3 控制示意图

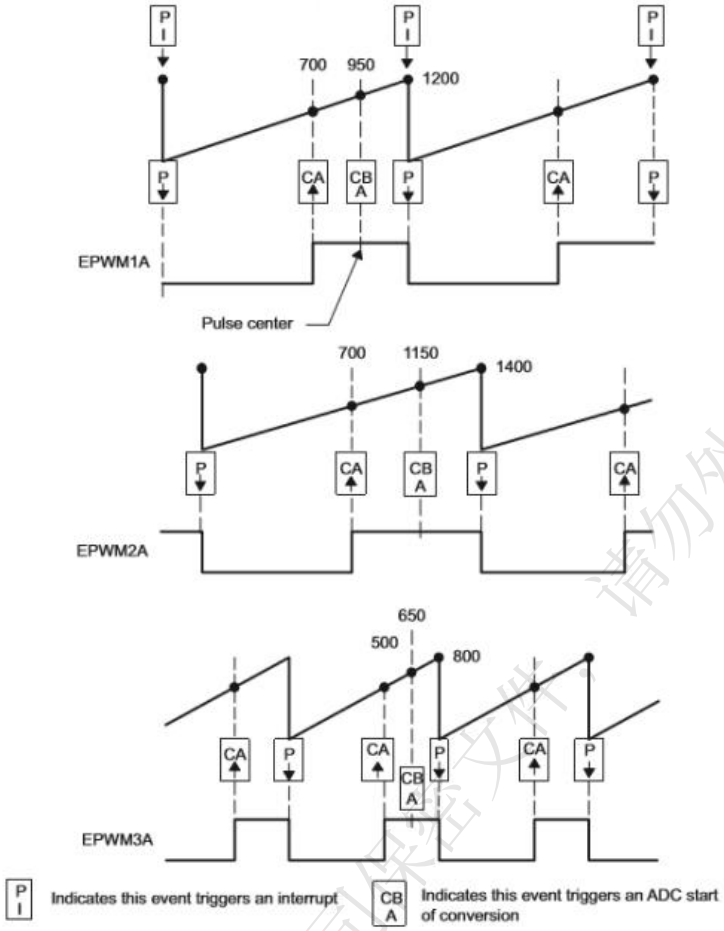


图 16-4 时序示意图

16.3.2. 相同频率控制多个 Buck 转换电路

如果要求两个 Buck 电路同步，那么 EPWM2 可以被配置为 slave 并且工作在 EPWM1 的整数倍频率下。从 master 到 slave 的 sync 信号可以确保这些模块保持锁定。下图显示的就是这种设置下的模型及波形。

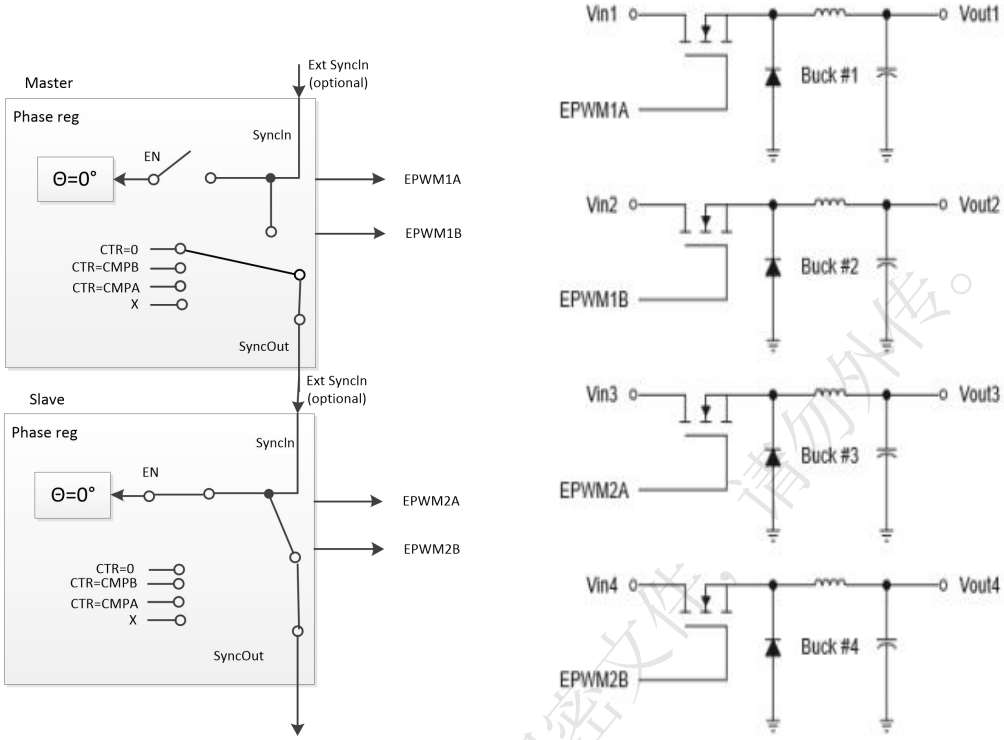


图 16-5 控制示意图 2

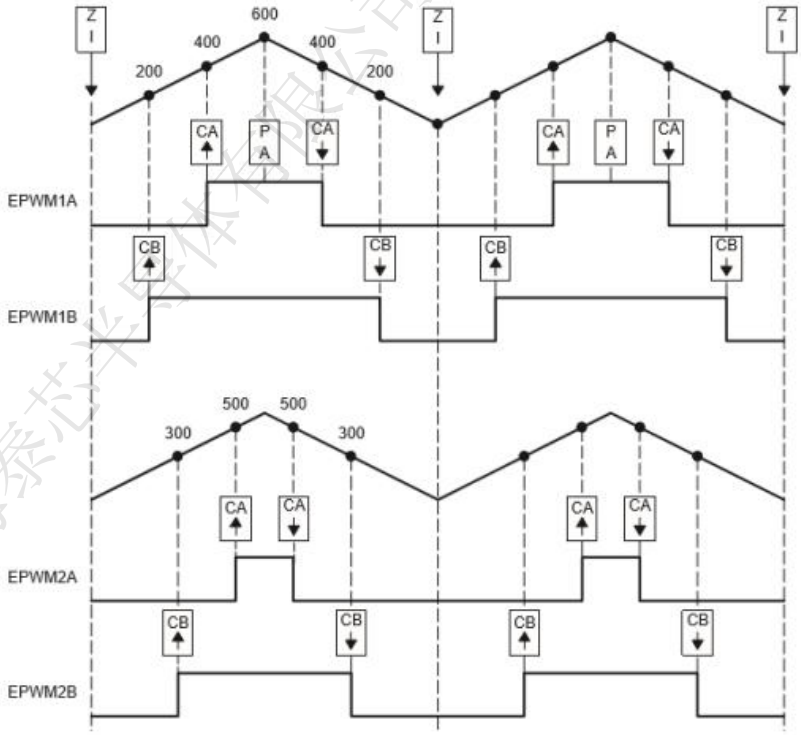


图 16-6 时序示意图 2

16.3.3. 控制多个 H 半桥电路（HHB）转换电路

控制多个开关元件的拓扑也可以用相同的 EPWM 模块处理。用一个 EPWM 模块控制一个 H 半桥级电路是有可能的。这种控制的方法还可以被扩充至多级 H 半桥级电路。下图显示在控制两个同步的 H 半桥级电路时，可以使第二级电路工作在第一级电路的工作频率的整数倍下。及这种设置下产生的波形。

Slave 被配置为 Sync flow-through。如果需要，可以增加第三个 H 半桥由第三个 EPWM 模块控制，并且必须与 master 同步。

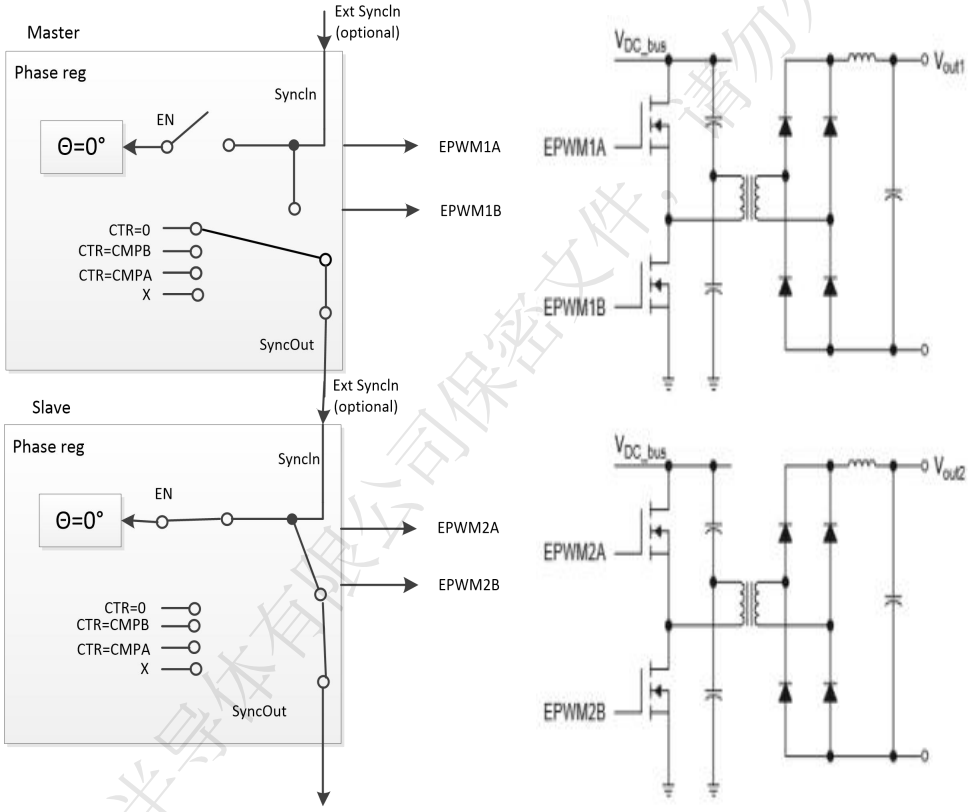


图 16-7 控制示意图 3

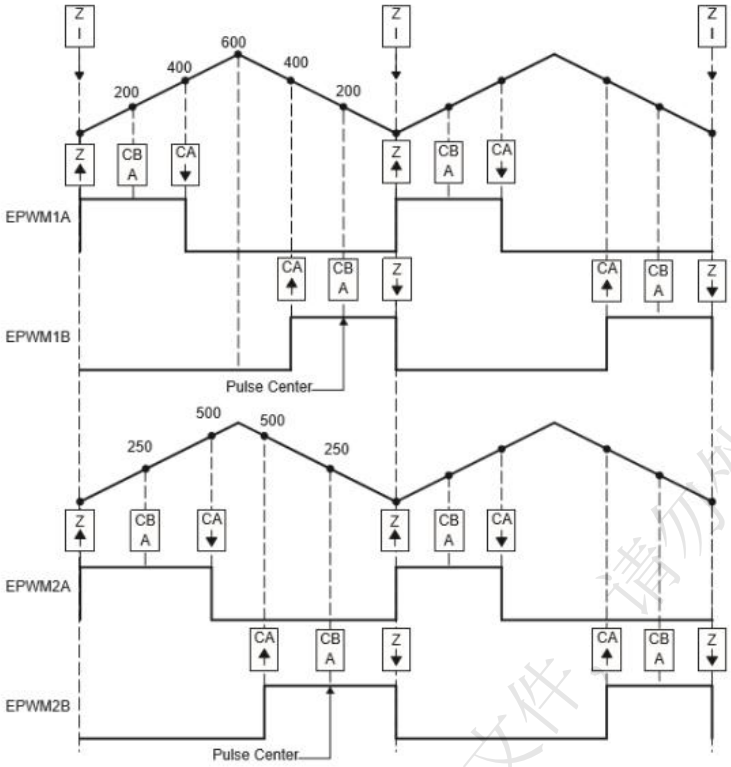


图 16-8 时序示意图 3

16.3.4. 控制电机的双三相逆变器（ACI 和 PMSM）

多模块控制单个功率级的思想可以扩充至三相逆变器中。在这种情况下，6 个开关元件可以由三个 PWM 模块控制，每支路一个。每条支路都必须在相同的频率下开关并且所有的支路都要同步。所以，1 个 master+2 个 slave 的组合可以轻松地解决这样的需求。下图显示了 6 个 PWM 模块是如何控制 2 个独立的三相逆变器的，每一个模块都运行着一个电机。接着的图是其运行波形。

如前几节所示，我们可以让每个逆变器都工作在不同的频率下（例如图 3-9 的例子中模块 1 和 4 都作为 master）；或者设置 1 个为 master、5 个为 slave，使两个逆变器同步。在这种情况下，模块 4、5、6（三个模块的频率相等）的频率是模块 1、2、3（三个模块的频率相等）的频率的整数倍。

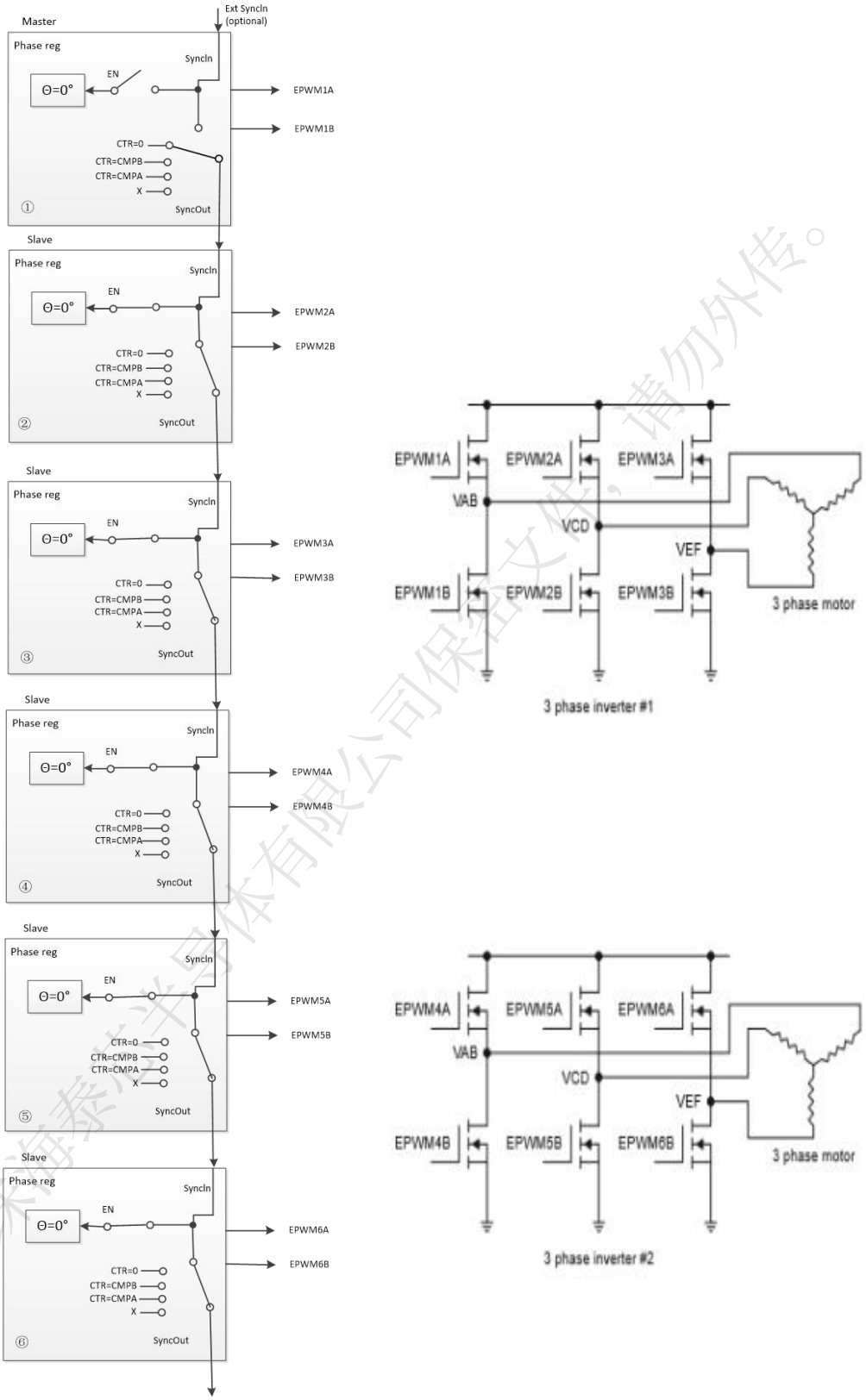


图 16-9 控制示意图 4

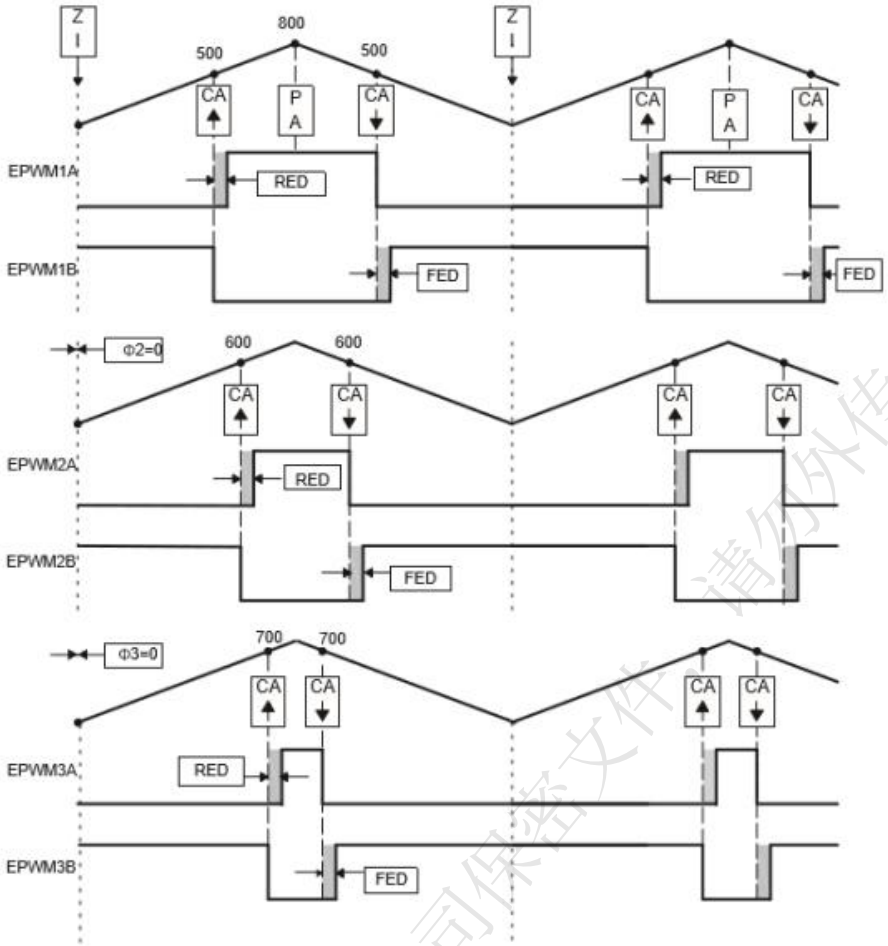


图 16-10 时序示意图 4

16.3.5. PWM 模块间相位控制的实际应用

直到目前为止，没有一种例子使用了相位寄存器（TBPHS）。相位寄存器不是被设置为 0 就是被完全忽略该值。然而，通过给 TBPHS 寄存器配置适当的值，多个 PWM 模块可以处理其他的功率拓扑级，这些拓扑级依靠各支路间的相位关系可以进行正确的操作。一个 PWM 模块可以配置 SyncIn 脉冲使得 TBPHS 寄存器可以加载进 TBCTR 寄存器。如上述观点所示，下图显示了一个相位关系为 120° 的 master 和 slave。

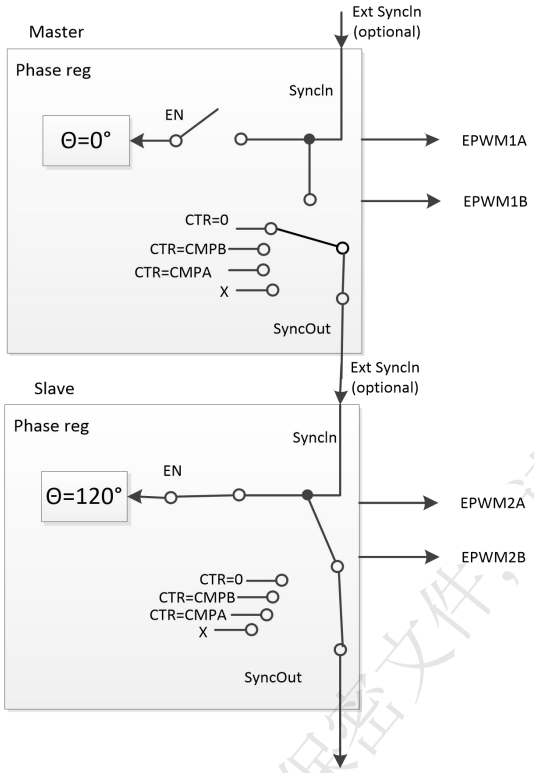


图 16-11 控制示意图 5

下图显示了这个配置下产生的时序波形图。Master 和 slave 的 TBPRD 都等于 600。Slave 的 TBPHS = 200（200/600*360° = 120°）。当 master 产生 SyncIn 脉冲时，TBPHS = 200 的值就会被加载进 slave 的 TBCTR 寄存器。所以 slave 的时基总是超前于 master 的时基 120°。

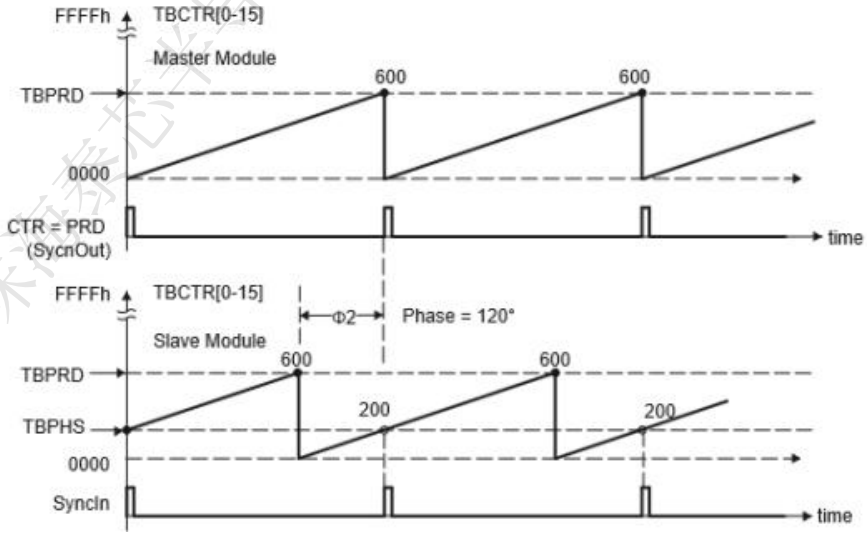


图 16-12 时序示意图 5

16.3.6. 控制三相交错 DC/DC 交换电路

如下图，通用的功率拓扑是利用了模块之间的相位偏移。该系统使用了 3 个 PWM 模块，其中模块 1 被配置为 master。工作时，相邻的模块之间的相位关系为 F = 120° 。这是通过分别设置 slave 的 TBPHS 寄存器 2 和 TBPHS 寄存器 3 的值为周期的 1/3 和 2/3 来实现的。例如，如果周期寄存器被载入一个 900 计数的值，那么 TBPHS（slave2） = 300 并且 TBPHS（slave3） = 600.所有的 slave 模块都要与 master1 模块同步。

这个思想通过适当设置 TBPHS 寄存器的值可以扩充 4 个甚至更多的相位。下列公式提供了 TBPHS 寄存器 N 个相位值：

TBPHS（N,M） = （TBPRD/N）*（-1）

其中，N 为相位值的个数

M 为 PWM 模块的个数

例如，在 3 个相位值的情况下，TBPRD = 900，

TBPHS（3,2） = （900/3）*（2-1） = 300（这是 slave 模块 2 的相位值）；

TBPHS（3,3） = （600）（这是 slave 模块 3 的相位值）。

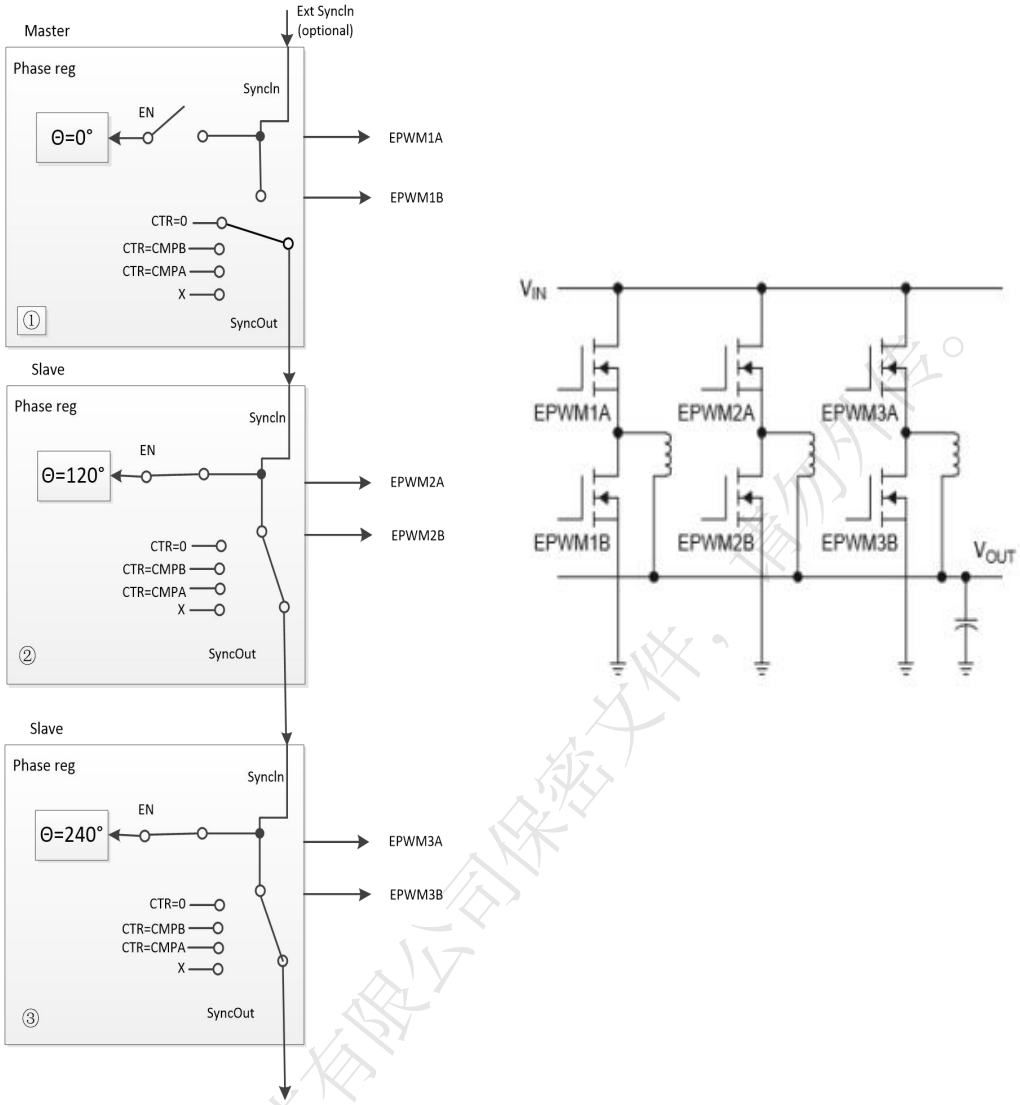


图 16-13 控制示意图 6

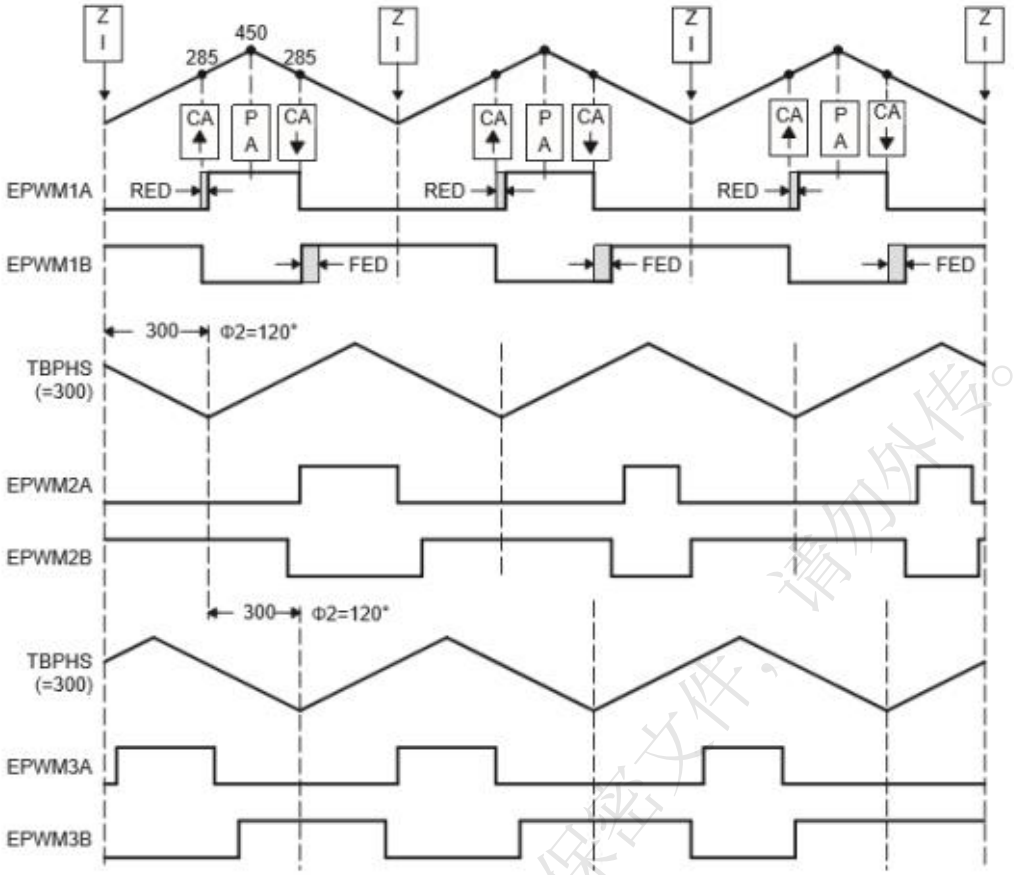


图 16-14 时序示意图 6

16.3.7. 控制 0 电压转换的全桥转换电路（ZVSFB）

下图假设了一种在各模块之间静态的或恒定不变的相位关系。在这种情况下，可以通过调制占空比在实现控制。也有可能在循环的基础中动态地改变相位值。这个特性使得模块本身可以像相移全桥或 0 电压转换全桥一样控制功率级拓扑。在这里控制的参数不是占空比（占空比保持不变或保持大约为 50%），而是个支路之间的相位关系。通过两个 PWM 模块的资源配置控制单功率级，就可以实现这样的一个系统，这就要求控制 4 个开关元件。下图显示一个同步的 master/slave 组合共同控制一个 H 全桥。在这种情况下，master 和 slave 都被要求在相同的 PWM 频率下进行转换。可以通过 slave 的相位寄存器（TBPHS）来控制相位。不使用 master 的相位寄存器，因此 master 的相位寄存器应初始化为 0。

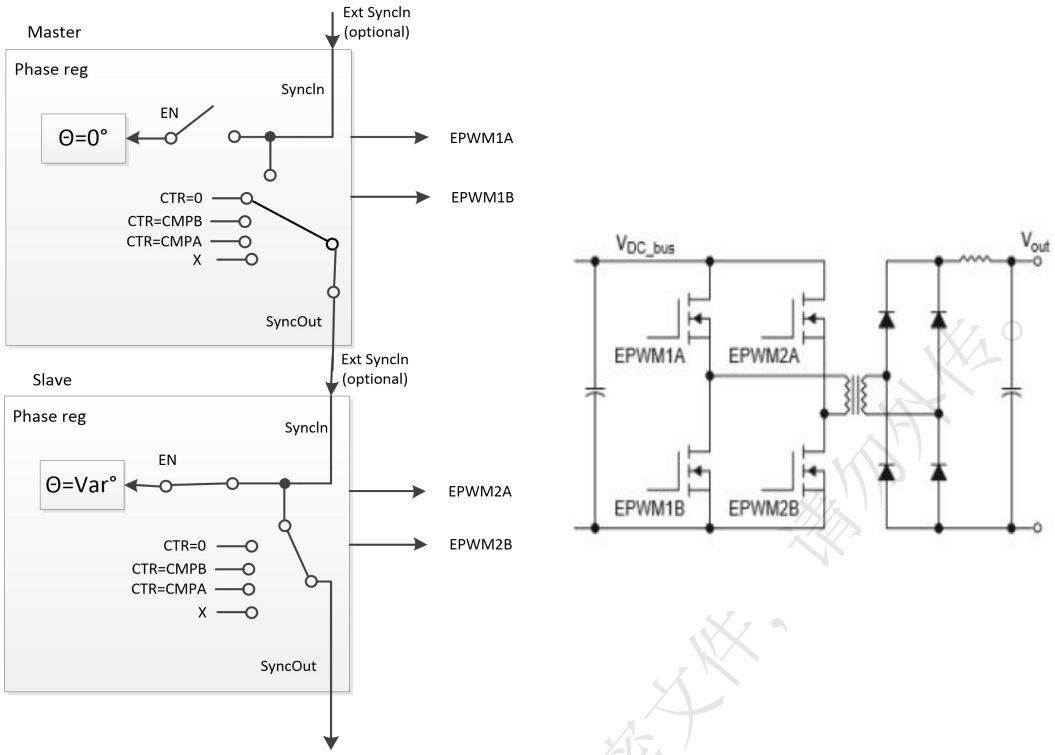


图 16-15 控制示意图 7

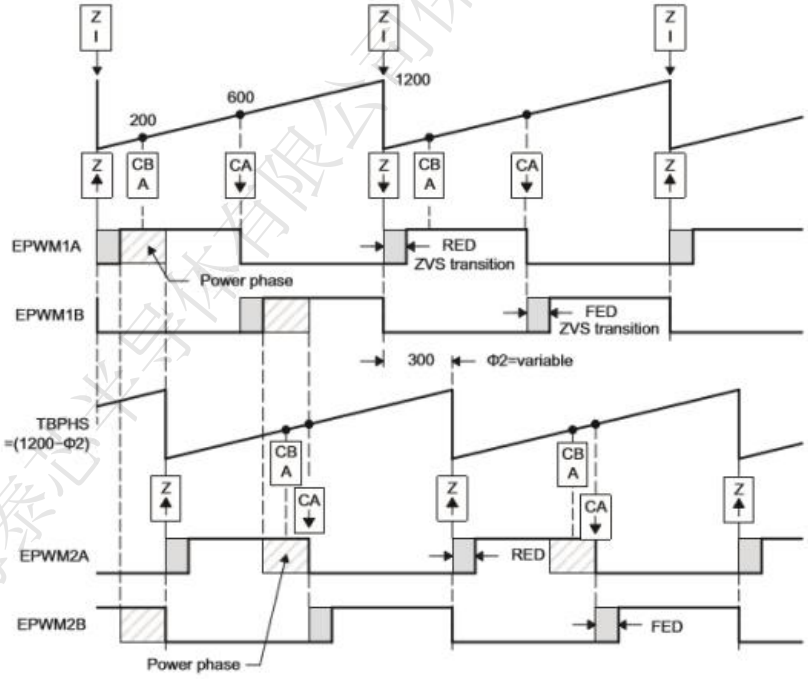


图 16-16 时序示意图 7

16.3.8. 控制峰值电流模式（Peak Current Mode）控制 Buck 模块

峰值电流模式控制技术提供了自动过流限流、输入电压变化的快速校正以及还原磁饱和

等优点。下图显示了 EPWM1A 和片上比较器在 Buck 变换器拓扑上的应用。输出电流可以通过电流感应电阻感应并转到片上比较器的正极。内部可编程 8 位 DAC 提供一个参考峰值电流给比较器的正极。作为选择，一个外部参考电流也连到该输入端。比较器的输出是数字比较器子模块的输入。EPWM 模块这么配置，是为了当感应电流达到峰值参考电流时立即触发 EPWM 的输出。采用了一种周期循环的机制。并显示了这种配置下产生的波形。

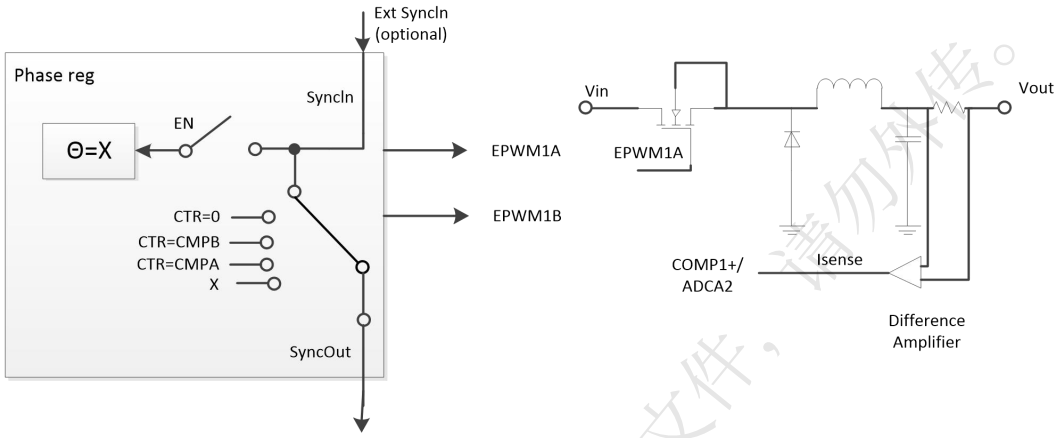


图 16-17 控制示意图 8

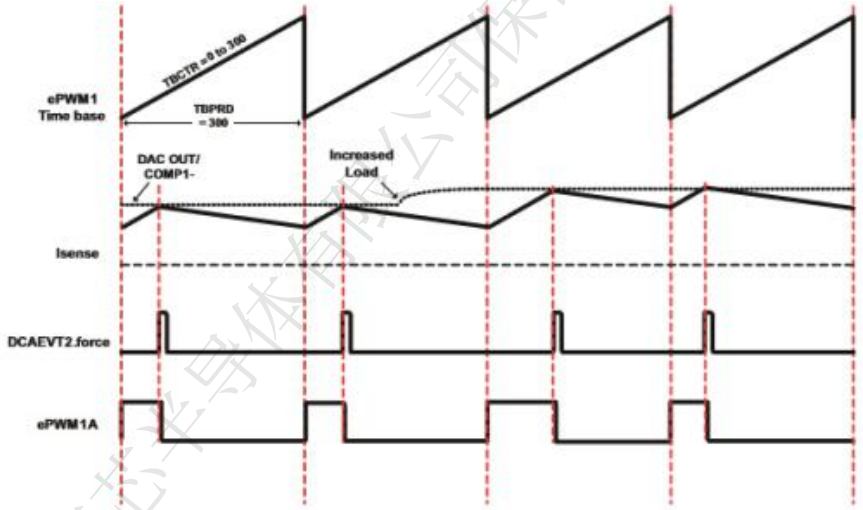


图 16-18 时序示意图 8

16.3.9. 控制 H 桥 LLC 谐振变换器

多年来，多种谐振变换器的拓扑在电力电子领域都很有名。另外，近来在要求高效和高功率密度的消费级电子应用方面，H 桥 LLC 谐振变换器拓扑十分受欢迎。例如，EPWM1 的单通道配置十分详细，但其配置可以简单扩充为多通道。控制参数是频率而不是占空比（占空

比应保持不变或保持在大约 50%）。用户需要实时更新死区时间，通过调整足够的软交换时延来提高效率。

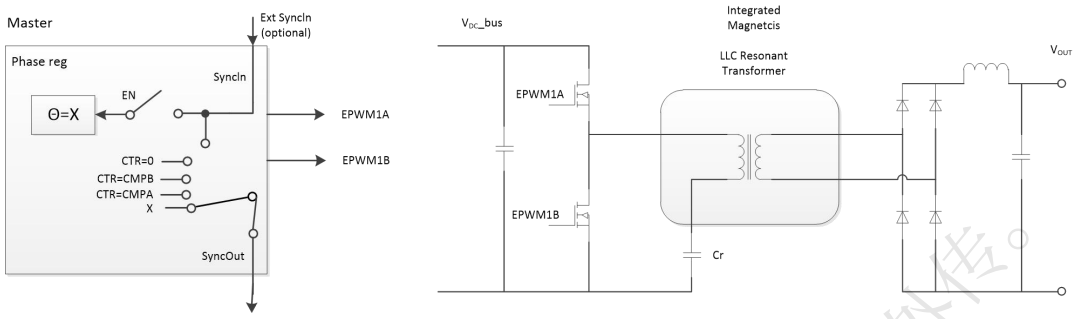


图 16-19 控制示意图 9

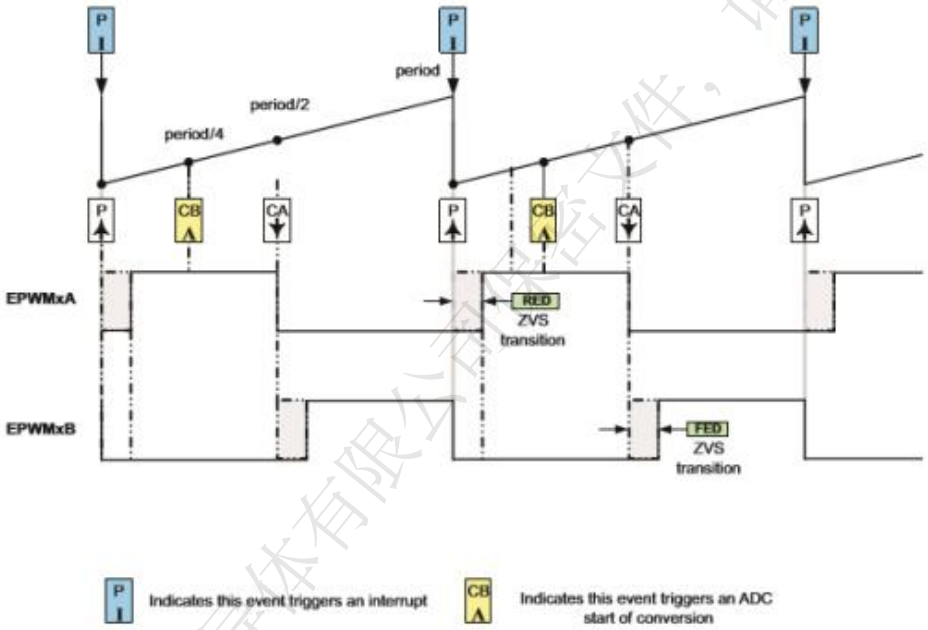


图 16-20 控制示意图 9

16.4. 寄存器

16.4.1. 寄存器基地址

Name	Base Address	Description
EPWM	0x40001800	EPWM 基地址

16.4.2. 寄存器列表

Offset Address	Name	Description
0x0000	EPWMx_TBCTL	TB 控制寄存器
0x0004	EPWMx_TBPRD	TB 周期寄存器
0x0008	EPWMx_TBPHASE	TB 相位寄存器
0x000c	EPWMx_CMPCTL	CMP 控制寄存器
0x0010	EPWMx_CMPA	CMPA 寄存器
0x0014	EPWMx_CMPB	CMPB 寄存器
0x0018	EPWMx_CMPC	CMPC 寄存器
0x001c	EPWMx_CMPD	CMPD 寄存器
0x0020	EPWMx_AQCTLAB	AQ 控制寄存器
0x0024	EPWMx_AQSFRC	AQ 行为寄存器 0
0x0028	EPWMx_AQCSFRC	AQ 行为寄存器 1
0x002c	EPWMx_DBCTL	DB 控制寄存器
0x0030	EPWMx_DEDELAY	DB 延迟寄存器
0x0034	EPWMx_ETCTL	ET 控制寄存器
0x0038	EPWMx_ETCTL2	ET 控制寄存器 2
0x003C	EPWMx_ETFLAG	ET 状态寄存器
0x0040	EPWMx_DCCTL	DC 控制寄存器
0x0044	EPWMx_DCTRIPSEL	DC 触发寄存器
0x0048	EPWMx_BLANKOFFSET	DC 窗口寄存器
0x004C	EPWMx_WINDIDTH	DC 窗口长度寄存器
0x0050	EPWMx_TZCTL	TZ 控制寄存器
0x0054	EPWMx_TZFLAG	TZ 状态寄存器
0x0058	EPWMx_DCCAP	DC 捕获寄存器
0x005C	EPWM_TTCTL	总控制寄存器

16.4.3. 寄存器详细说明

16.4.3.1. EPWMx_TBCTL

Bit(s)	Name	Description	R/W	Reset
31:16	TBCTR	**读取该寄存器可以知道此时计数器的计数值**	R	0x0
15:13	CLK_DIV	**模块时钟分频** 0x0：epwm clk 0x1：epwm clk div 2 0x2：epwm clk div 4 0x3：epwm clk div 8 0x4：epwm clk div 16	RW	0x0
12:10	Reserved	-	-	-
9	TBCTR_DIR	**TB 计数方向状态位** 0x0：TB 处于递减 0x1：TB 处于递增	R	0x0
8	SYNCI	**输入同步锁存状态位** **读：** 0x0：没有发生外部同步事件。 0x1：发生了外部同步事件(EPWMxSYNCI) **写：** 0x0：无效: 0x1：在此位上写入 1 将清除闩锁事件。	RW	0x0
7:5	SYNCO_SEL	**同步信号输出选择** 0x0：sync in 0x1：tbcnt equal zero 0x2：tbcnt equal cmpa 0x3：tbcnt equal cmpb 0x4：tbcnt equal cmpc 0x5：tbcnt equal cmpd 0x6 ~ 0x7：无效	RW	0x0
4	SWF_SYNC	**软件同步使能** 0x0：关闭 0x1：打开	W	0x0
3	PHSEN	**相位同步使能** 0x0：关闭 0x1：打开	RW	0x0
2:1	CTRMODE	**计数模式选择** 0x0：Up mode 0x1：Down mode 0x2：Up Dowm mode 0x3：constant	RW	0x0
0	PRDLD	**周期寄存器的影子寄存器使能** 0x0：当计数器 TBCTR 等于零时，周期寄存器(TBPRD)从阴影寄存器加载	RW	0x0

		0x1：立即加载 TBPRD 寄存器，而不使用阴影寄存器		

16.4.3.2. EPWMx_TBPRD

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	-	-	-
15:0	TBPRD	**TB 计数器的周期配置** 该寄存器的阴影由 TBCTL[PRDLD]位启用和禁用。默认情况下，这个寄存器是隐藏的。 ● 如果 TBCTL[PRDLD] = 0，则启用影子寄存器，任何写入或读取将自动到影子寄存器。在这种情况下，当 TB 计数器等于零时，寄存器将从影子寄存器加载； ● 如果 TBCTL[PRDLD] = 1，那么阴影被禁用，任何写或读都将直接到计数寄存器，即主动控制硬件的寄存器；	RW	0x0

16.4.3.3. EPWMx_TBPHASE

Bit(s)	Name	Description	R/W	Reset
31:17	reserved	-	-	-
16	TBDIR	**设置计数器的计数方向** 设置所选 ePWM 相对于提供同步输入信号的时基的时基计数器方向 ● 如果 TBCTL[PHSEN] = 0，则同步事件将被忽略，并且没有加载该方向的时基方向； ● 如果 TBCTL[PHSEN] = 1，则当同步事件发生时，时基方向将被加载方向(TBDIR)； 同步事件可以由输入同步信号(EPWMxSYNCI)发起，也可以由软件强制同步。 注意：此位仅用于上下模式	RW	0x0
15:0	TBPHS	**同步信号触发时，进行同步的相位值** 这些位设置所选 ePWM 相对于提供同步输入信号的时基的时基计数器相位。 ● 如果 TBCTL[PHSEN] = 0，则同步事件将被忽略，并且不加载时序计数器； ● 如果 TBCTL[PHSEN] = 1，则当同步事件发生时，时基计数器(TBCTR)将加载相位(TBPHS)； 同步事件可以由输入同步信号(EPWMxSYNCI)发起，也可以由软件强制同步。	RW	0x0

16.4.3.4. EPWMx_CMPCTL

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	-	-	-
15	SHDWDFULL	**CMPD 影子寄存器状态** 0x0：影子寄存器未满 0x1：影子寄存器处于满的状态，如果 CPU 再写会覆盖到	RO	0x0
14	SHDWDMODE	**CMPD 加载模式选择** 0x0：阴影模式，作为双缓冲。所有通过 CPU 的写操作都要访问影子寄存器 0x1：立即模式，只使用活动比较寄存器。所有写和读操作都直接访问活动寄存器，以便立即进行比较操作	RW	0x0
13:12	LOADDMODE	**CMPD 加载数据时刻点选择** 该功能只在影子模式下有效 0x0：在 CTR=0 时，加载比较值数据 0x1：在 CTR=PRD 时，加载比较值数据 0x2：加载 CTR=0 和 CTR=PRD 时，都加载比较值数据 0x3：无效	RW	0x0
11	SHDWCFULL	**CMPC 影子寄存器状态** 0x0：影子寄存器未满 0x1：影子寄存器处于满的状态，如果 CPU 再写会覆盖到	RO	0x0
10	SHDWCMODE	**CMPC 加载模式选择** 0x0：阴影模式，作为双缓冲。所有通过 CPU 的写操作都要访问影子寄存器 0x1：立即模式，只使用活动比较寄存器。所有写和读操作都直接访问活动寄存器，以便立即进行比较操作	RW	0x0
9:8	LOADCMODE	**CMPC 加载数据时刻点选择** 该功能只在影子模式下有效 0x0：在 CTR=0 时，加载比较值数据 0x1：在 CTR=PRD 时，加载比较值数据 0x2：加载 CTR=0 和 CTR=PRD 时，都加载比较值数据 0x3：无效	RW	0x0
7	SHDWBFULL	**CMPB 影子寄存器状态** 0x0：影子寄存器未满 0x1：影子寄存器处于满的状态，如果 CPU 再写会覆盖到	RO	0x0
6	SHDWAFULL	**CMPA 影子寄存器状态** 0x0：影子寄存器未满	RO	0x0

		0x1：影子寄存器处于满的状态，如果 CPU 再写会覆盖到		
5	SHDWBMODE	**CMPB 加载模式选择** 0x0：阴影模式，作为双缓冲。所有通过 CPU 的写操作都要访问影子寄存器 0x1：立即模式，只使用活动比较寄存器。所有写和读操作都直接访问活动寄存器，以便立即进行比较操作	RW	0x0
4	SHDWAMODE	**CMPA 加载模式选择** 0x0：阴影模式，作为双缓冲。所有通过 CPU 的写操作都要访问影子寄存器 0x1：立即模式，只使用活动比较寄存器。所有写和读操作都直接访问活动寄存器，以便立即进行比较操作	RW	0x0
3:2	LOADBMODE	**CMPB 加载数据时刻点选择** 该功能只在影子模式下有效 0x0：在 CTR=0 时，加载比较值数据 0x1：在 CTR=PRD 时，加载比较值数据 0x2：加载 CTR=0 和 CTR=PRD 时，都加载比较值数据 0x3：无效	RW	0x0
1:0	LOADAMODE	**CMPA 加载数据时刻点选择** 该功能只在影子模式下有效 0x0：在 CTR=0 时，加载比较值数据 0x1：在 CTR=PRD 时，加载比较值数据 0x2：加载 CTR=0 和 CTR=PRD 时，都加载比较值数据 0x3：无效	RW	0x0

16.4.3.5. EPWMx_CMPA

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	–	–	–
15:0	CMPA	**CMPA 寄存器值** 默认情况下，对该寄存器的写入是隐藏的。通过 CMPCTL[SHDWAMODE]位启用和禁用阴影。 ● 如果 CMPCTL[SHDWAMODE]=0，则启用阴影，任何写入都将自动到阴影寄存器。CMPCTL[LOADAMODE]位字段决定哪个事件将从影子寄存器加载活动寄存器。 ● 在写入之前，可以读取 CMPCTL[shdwfulll]位，以确定阴影寄存器当前是否已满。 ● 如果 CMPCTL[SHDWAMODE]=1，那么影子寄存器被禁用，任何写入都将直接到活动寄存器，即主动控制硬件的寄存器。	RW	0x0

16.4.3.6. EPWMx_CMPB

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	–	–	–
15:0	CMPB	**CMPB 寄存器值** 默认情况下，对该寄存器的写入是隐藏的。通过 CMPCTL[SHDWAMODE]位启用和禁用阴影。 ● 如果 CMPCTL[SHDWAMODE]=0，则启用阴影，任何写入都将自动到阴影寄存器。CMPCTL[LOADAMODE]位字段决定哪个事件将从影子寄存器加载活动寄存器。 ● 在写入之前，可以读取 CMPCTL[shdwfulll]位，以确定阴影寄存器当前是否已满。 ● 如果 CMPCTL[SHDWAMODE]=1，那么影子寄存器被禁用，任何写入都将直接到活动寄存器，即主动控制硬件的寄存器。	RW	0x0

16.4.3.7. EPWMx_CMPC

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	–	–	–
15:0	CMPC	**CMPC 寄存器值** 默认情况下，对该寄存器的写入是隐藏的。通过 CMPCTL[SHDWAMODE]位启用和禁用阴影。 ● 如果 CMPCTL[SHDWAMODE]=0，则启用阴影，任何写入都将自动到阴影寄存器。CMPCTL[LOADAMODE]位字段决定哪个事件将从影子寄存器加载活动寄存器。 ● 在写入之前，可以读取 CMPCTL[shdwfulll]位，以确定阴影寄存器当前是否已满。 ● 如果 CMPCTL[SHDWAMODE]=1，那么影子寄存器被禁用，任何写入都将直接到活动寄存器，即主动控制硬件的寄存器。	RW	0x0

16.4.3.8. EPWMx_CMPD

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	–	–	–
15:0	CMPD	**CMPD 寄存器值** 默认情况下，对该寄存器的写入是隐藏的。通过 CMPCTL[SHDWAMODE]位启用和禁用阴影。 ● 如果 CMPCTL[SHDWAMODE]=0，则启用阴影，	RW	0x0

		任何写入都将自动到阴影寄存器。CMPCTL[LOADAMODE]位字段决定哪个事件将从影子寄存器加载活动寄存器。 ● 在写入之前,可以读取 CMPCTL[shdwfulll]位，以确定阴影寄存器当前是否已满。 ● 如果 CMPCTL[SHDWAMODE]=1,那么影子寄存器被禁用，任何写入都将直接到活动寄存器，即主动控制硬件的寄存器。		

16.4.3.9. EPWMx_AQCTLAB

Bit(s)	Name	Description	R/W	Reset
31:28	reserved	-	-	-
27:26	CBDB	**计数器 CNT 处于递减趋势下，当 CNT=CMPB 时 EPWMxB 的行为控制** 0x0：保持原来 0x1：EPWMxB 输出低电平 0x2：EPWMxB 输出高电平 0x3：翻转	RW	0x0
25:24	CBUB	**计数器 CNT 处于递增趋势下，当 CNT=CMPB 时 EPWMxB 的行为控制** 0x0：保持原来 0x1：EPWMxB 输出低电平 0x2：EPWMxB 输出高电平 0x3：翻转	RW	0x0
23:22	CADB	**计数器 CNT 处于递减趋势下，当 CNT=CMPA 时 EPWMxB 的行为控制** 0x0：保持原来 0x1：EPWMxB 输出低电平 0x2：EPWMxB 输出高电平 0x3：翻转	RW	0x0
21:20	CAUB	**计数器 CNT 处于递增趋势下，当 CNT=CMPA 时 EPWMxB 的行为控制** 0x0：保持原来 0x1：EPWMxB 输出低电平 0x2：EPWMxB 输出高电平 0x3：翻转	RW	0x0
19:18	PRDB	**计数器 CNT 处于 up and down 模式下,当 CNT=PRD 时 EPWMxB 的行为控制** 0x0：保持原来 0x1：EPWMxB 输出低电平 0x2：EPWMxB 输出高电平 0x3：翻转	RW	0x0
17:16	ZROB	**计数器 CNT 处于 up and down 模式下，当 CNT=0 时 EPWMxB 的行为控制**	RW	0x0

		0x0：保持原来 0x1：EPWMxB 输出低电平 0x2：EPWMxB 输出高电平 0x3：翻转		
15:12	reserved	-	-	-
11:10	CBDA	**计数器 CNT 处于递减趋势下，当 CNT=CMPB 时 EPWMxA 的行为控制** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0
9:8	CBUA	**计数器 CNT 处于递增趋势下，当 CNT=CMPB 时 EPWMxA 的行为控制** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0
7:6	CADA	**计数器 CNT 处于递减趋势下，当 CNT=CMPA 时 EPWMxA 的行为控制** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0
5:4	CAUA	**计数器 CNT 处于递增趋势下，当 CNT=CMPA 时 EPWMxA 的行为控制** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0
3:2	PRDA	**计数器 CNT 处于 up and down 模式下，当 CNT=PRD 时 EPWMxA 的行为控制** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0
1:0	ZROA	**计数器 CNT 处于 up and down 模式下，当 CNT=0 时 EPWMxA 的行为控制** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0

16.4.3.10. EPWMX_AQSFRC

Bit(s)	Name	Description	R/W	Reset

31:8	reserved	-	-	-
7:6	RLDCSF	**AQCSFRC 影子寄存器加载状态** 0x0：CNT=0 0x1：CNT=PRD 0x2：CNT=0 或者 CNT=PRD 0x3：立即加载	R	0x0
5	OTSFB	**软件触发 One-Time，让 EPWMxB 响应行为** 0x0：无效 0x1：立即触发一次 One-Time，EPWMxB 做出相对应的行为	RW	0x0
4:3	ACTSFB	触发 **One-Time 信号后，EWPMxB 行为选择** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0
2	OTSFA	**软件触发 One-Time，让 EPWMxA 响应行为** 0x0：无效 0x1：立即触发一次 One-Time，EPWMxA 做出相对应的行为	RW	0x0
1:0	ACTSFA	触发 **One-Time 信号后，EWPMxA 行为选择** 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：翻转	RW	0x0

16.4.3.11. EPWMX_AQCSFRC

Bit(s)	Name	Description	R/W	Reset
31:4	reserved	-	-	-
3:2	CSFB	**软件循环触发配置（EPWMxB）** 立即模式下，连续触发会在下一个 TBCLK 边沿生效； 影子模式下，要在影子寄存器加载后的下一个 TBCLK 边沿才会生效。 0x0：保持原来 0x1：EPWMxA 输出低电平 0x2：EPWMxA 输出高电平 0x3：保持原来	RW	0x0
1:0	CSFA	**软件循环触发配置（EPWMxA）** 立即模式下，连续触发会在下一个 TBCLK 边沿生效； 影子模式下，要在影子寄存器加载后的下一个 TBCLK 边沿才会生效。 0x0：保持原来 0x1：EPWMxA 输出低电平	RW	0x0

		0x2：EPWMxA 输出高电平 0x3：保持原来		

16.4.3.12. EPWMX_DBCTL

Bit(s)	Name	Description	R/W	Reset
31:19	reserved	-	-	-
18	SHDWDBCTLMODE	**DBCTL加载模式** 0x0：立即模式 0x1：影子模式	RW	0x0
17:16	LOADDBCTLMODE	**DBCTL加载点选择** 0x0：CNT = 0 0x1：CNT = PRD 0x2：CNT = 0，或者CNT= PRD 0x3：无效 注意：只在影子模式有效	RW	0x0
15	HALFCYCLE	**死区时钟分频选择** 0x0：不分频 0x1：1/2 EPWM 时钟	RW	0x0
14	reserved	-	-	-
13:12	OUTSWAP	**死区输出交换设置** 在图16-1中，BIT13对应S7，BIT12对应这个S6 0x0：OutA = A-path，OutB = B-path 0x1：OutA = A-path，OutB = A-path 0x2：OutA = B-path，OutB = B-path 0x3：OutA = B-path，OutB = A-path	R/W	0x0
11	SHDWDBFEDMODE	**FED死区加载模式** 0x0：立即模式 0x1：影子模式	R/W	0x0
10	SHDWDBREDMODE	**RED死区加载模式** 0x0：立即模式 0x1：影子模式	R/W	0x0
9-8	LOADFEDMODE	**DBFED加载时刻选择** 0x0：CNT = 0 0x1：CNT = PRD 0x2：CNT = 0，或者CNT= PRD 0x3：无效 注意：只在影子模式有效	R/W	0x0
7-6	LOADREDMODE	**DBRED加载时刻选择** 0x0：CNT = 0 0x1：CNT = PRD 0x2：CNT = 0，或者CNT= PRD 0x3：无效 注意：只在影子模式有效	R/W	0x0
5:4	IN_MODE	**死区输入控制**	RW	0x0

		死区的输入的行为来至于 AQ 模块 0x0：死区上升沿和下降沿的输入源都选择 EPWMxA 0x1：死区上升沿的输入源选择 EPWMxB，死区下降沿的输入源选择 EPWMxA 0x2：死区上升沿的输入源选择 EPWMxA，死区下降沿的输入源选择 EPWMxB 0x3：死区上升沿和下降沿的输入源都选择 EPWMxA		
3:2	POLSEL	**死区输出极性选择** 0x0：EPWMxA 和 EPWMxB 保持原来的极性输出 0x1：EPWMxA 输出极性翻转，EPWMxB 保持原来极性输出 0x2：EPWMxA 保持原来极性输出，EPWMxB 输出极性翻转 0x3：EPWMxA 和 EPWMxB 输出极性都翻转	RW	0x0
1:0	OUT_MODE	**死区输出模式选择** 0x0：关闭死区功能 0x1：关闭上升沿死区，打开下降沿死区 0x2：打开上升沿死区，关闭下降沿死区 0x3：同时打开上升沿和下降沿的死区	RW	0x0

16.4.3.13. EPWMX_DBDELAY

Bit(s)	Name	Description	R/W	Reset
31:30	reserved	--	-	-
29:16	FED	**下降沿的死区时间配置** 14bit 位宽	RW	0x0
15:14	reserved	-	-	-
13:0	RED	**上降沿的死区时间配置** 14bit 位宽	RW	0x0

16.4.3.14. EPWMx_ETCTL

Bit(s)	Name	Description	R/W	Reset
31:24	reserved	-	-	-
23:20	SOCBCNT	**SOCB 触发的次数** 0x0：没有事件触发 0x1：已经触发 1 次 0x2：已经触发 2 次 ... 0xF：已经触发 15 次	RO	0x0
19:16	SOCBPRD	**SOCB 触发的频次选择** 0x0：关闭	RW	0x0

		0x1：ETCTL[SOCBCNT] = 0x1 时触发中断信号 0x2：ETCTL[SOCBCNT] = 0x2 时触发中断信号 ... 0xF：ETCTL[SOCBCNT] = 0xF 时触发中断信号		
15:12	SOCACNT	**SOCA 触发的次数** 0x0：没有事件触发 0x1：已经触发 1 次 0x2：已经触发 2 次 ... 0xF：已经触发 15 次	RO	0x0
11:8	SOCAPRD	**SOCA 触发的频次选择** 0x0：关闭 0x1：ETCTL[SOCACNT] = 0x1 时触发中断信号 0x2：ETCTL[SOCACNT] = 0x2 时触发中断信号 ... 0xF：ETCTL[SOCACNT] = 0xF 时触发中断信号	RW	0x0
7:4	INTCNT	**中断触发的次数** 0x0：没有事件触发 0x1：已经触发 1 次 0x2：已经触发 2 次 ... 0xF：已经触发 15 次	RO	0x0
3:0	INTPRD	**中断触发的频次选择** 0x0：关闭 0x1：ETCTL[INTCNT] = 0x1 时触发中断信号 0x2：ETCTL[INTCNT] = 0x2 时触发中断信号 ... 0xF：ETCTL[INTCNT] = 0xF 时触发中断信号	RW	0x0

16.4.3.15. EPWMx_ETCTL2

Bit(s)	Name	Description	R/W	Reset
31:26	Reserved	-	-	-
25:16	MULTSCSEL	多点触发信号选择 每 1 bit 表示一个信号的使能位，可以多通道使能 BIT25：CNT = 0 BIT24：CNT = PRD BIT23：CNT 处于递增模式，且 CNT = CMPA BIT22：CNT 处于递减模式，且 CNT = CMPA BIT21：CNT 处于递增模式，且 CNT = CMPB BIT20：CNT 处于递减模式，且 CNT = CMPB BIT19：CNT 处于递增模式，且 CNT = CMPC BIT18：CNT 处于递减模式，且 CNT = CMPC BIT17：CNT 处于递增模式，且 CNT = CMPD	RW	0x0

		BIT16：CNT 处于递减模式，且 CNT = CMPD		
15	Reserved	-	-	-
14	SOCBEN	**SOCB 触发使能** 0x0：关闭 0x1：打开	RW	0x0
13:10	SOCBSEL	**SOCB 触发源选择** 0x0：无效 0x1：CNT = 0 0x2：CNT = PRD 0x3：在 up-down 模式下，CNT = 0 或者 CNT = PRD 0x4：在 CNT 处于递增趋势，且 CNT = CMPA 0x5：在 CNT 处于递减趋势，且 CNT = CMPA 0x6：在 CNT 处于递增趋势，且 CNT = CMPB 0x7：在 CNT 处于递减趋势，且 CNT = CMPB 0x8：在 CNT 处于递增趋势，且 CNT = CMPC 0x9：在 CNT 处于递减趋势，且 CNT = CMPC 0xA：在 CNT 处于递增趋势，且 CNT = CMPD 0xB：在 CNT 处于递减趋势，且 CNT = CMPD 0xC：多点触发信号 ETCTL2[MULTSCSEL] Other：无效	RW	0x0
9	SOCAEN	**SOCA 触发使能** 0x0：关闭 0x1：打开	RW	0x0
8:5	SOCASEL	**SOCA 触发源选择** 0x0：无效 0x1：CNT = 0 0x2：CNT = PRD 0x3：在 up-down 模式下，CNT = 0 或者 CNT = PRD 0x4：在 CNT 处于递增趋势，且 CNT = CMPA 0x5：在 CNT 处于递减趋势，且 CNT = CMPA 0x6：在 CNT 处于递增趋势，且 CNT = CMPB 0x7：在 CNT 处于递减趋势，且 CNT = CMPB 0x8：在 CNT 处于递增趋势，且 CNT = CMPC 0x9：在 CNT 处于递减趋势，且 CNT = CMPC 0xA：在 CNT 处于递增趋势，且 CNT = CMPD 0xB：在 CNT 处于递减趋势，且 CNT = CMPD 0xC：多点触发信号 ETCTL2[MULTSCSEL] Other：无效	RW	0x0
4	INTEN	**中断触发使能** 0x0：关闭 0x1：打开	RW	0x0
3:0	INTSEL	**中断触发源选择** 0x0：无效 0x1：CNT = 0 0x2：CNT = PRD 0x3：在 up-down 模式下，CNT = 0 或者 CNT = PRD	RW	0x0

		0x4：在 CNT 处于递增趋势，且 CNT = CMPA 0x5：在 CNT 处于递减趋势，且 CNT = CMPA 0x6：在 CNT 处于递增趋势，且 CNT = CMPB 0x7：在 CNT 处于递减趋势，且 CNT = CMPB 0x8：在 CNT 处于递增趋势，且 CNT = CMPC 0x9：在 CNT 处于递减趋势，且 CNT = CMPC 0xA：在 CNT 处于递增趋势，且 CNT = CMPD 0xB：在 CNT 处于递减趋势，且 CNT = CMPD 0xC：多点触发信号 ETCTL2[MULTSCSEL] 0ther：无效		

16.4.3.16. EPWMx_ETFLAG

Bit(s)	Name	Description	R/W	Reset
31:12	reserved	-	-	-
11	FRCSOCB	**软件强制 SOCB 触发使能** 0x0：无效 0x1：产生 SOCB 信号	RW	0x0
10	FRCSOCA	**软件强制 SOCA 触发使能** 0x0：无效 0x1：产生 SOCA 信号	RW	0x0
9	Reserved	-	-	-
8	FRCINT	**软件强制中断触发使能** 0x0：无效 0x1：产生中断	RW	0x0
7	CLRSOCB	**SOCB 触发清除标志** 写 1 清除	RW	0x0
6	CLRSOCA	**SOCA 触发清除标志** 写 1 清除	RW	0x0
5	Reserved	-	-	-
4	CLRINT	**中断触发清除标志** 写 1 清除	RW	0x0
3	SOCB	**SOCB 触发标志位** 0x0：未触发 0x1：已经触发	R	0x0
2	SOCA	**SOCA 触发标志位** 0x0：未触发 0x1：已经触发	RW	0x0
1	reserved		-	-
0	INT	**中断触发标志位** 0x0：未触发 0x1：已经触发	RW	0x0

16.4.3.17. EPWMx_DCCTL

Bit(s)	Name	Description	R/W	Reset
31:20	reserved	-	-	-
19	AEVT2FRCSYNCSEL	**DCAEVT2 强制同步信号选择** 0x0：同步信号 0x1：异步信号	RW	0x0
18	AEVT2SRCSEL	**DCAEVT2 信号源选择** 0x0：DCAEVT2 Signal 0x1：DCEVTFILT Signal	RW	0x0
17	AEVT1SYNCE	**DCAEVT1 同步信号使能** 0x0：关闭 0x1：打开	RW	0x0
16	AEVT1SOCE	**DCAEVT1 SOC 使能** 0x0：关闭 0x1：打开	RW	0x0
15	AEVT1FRCSYNCSEL	**DCAEVT1 强制同步信号选择** 0x0：同步信号 0x1：异步信号	RW	0x0
14	AEVT1SRCSEL	**DCAEVT1 信号源选择** 0x0：DCAEVT1 Signal 0x1：DCEVTFILT Signal	RW	0x0
13	BEVT2FRCSYNCSEL	**DCBEVT2 强制同步信号选择** 0x0：同步信号 0x1：异步信号	RW	0x0
12	BEVT2SRCSEL	**DCBEVT2 信号源选择** 0x0：DCBEVT2 Signal 0x1：DCEVTFILT Signal	RW	0x0
11	BEVT1SYNCE	**DCBEVT1 同步信号使能** 0x0：关闭 0x1：打开	RW	0x0
10	BEVT1SOCE	**DCBEVT1 SOC 使能** 0x0：关闭 0x1：打开	RW	0x0
9	BEVT1FRCSYNCSEL	**DCBEVT1 强制同步信号选择** 0x0：同步信号 0x1：异步信号	RW	0x0
8	BEVT1SRCSEL	**DCBEVT1 信号源选择** 0x0：DCBEVT1 Signal 0x1：DCEVTFILT Signal	RW	0x0
7:6	PULSESEL	**Blanking 脉冲选择和捕获起始对齐点** 0x0：CNT = PRD 0x1：CNT = 0 Other：无效	RO	0x0
5	BLANKINV	**Blanking 窗口取反使能** 0x0：关闭	RW	0x0

		0x1：打开		
4	BLANKE	**Blanking 窗口使能** 0x0：关闭 0x1：打开	RW	0x0
3:2	FILTSRCSEL	**滤波信号源选择** 0x0：选择 DCAEVT1 Signal 0x1：选择 DCAEVT2 Signal 0x2：选择 DCBEVT1 Signal 0x3：选择 DCBEVT2 Signal	RW	0x0
1	SHDWMODE	**Capture 影子功能** 0x0：影子模式 0x1：立即模式	RW	0x0
0	CAPE	**Capture 功能使能** 0x0：关闭 0x1：打开	RW	0x0

16.4.3.18. EPWMx_DCTRIPSEL

Bit(s)	Name	Description	R/W	Reset
31:28	reserved	-	-	-
27:25	DCBEVT2	**DCB EVT2 输入触发条件选择** 0x0：关闭 0x1：DCBH = low，DCBL = don't care 0x2：DCBH = high，DCBL = don't care 0x3：DCBL = low，DCBH = don't care 0x4：DCBL = high，DCBH = don't care 0x5：DCBL = high，DCBH = low 0x6：DCBL = high，DCBH = high 0x7：DCBL = low，DCBH = low	RW	0x0
24:22	DCBEVT1	**DCB EVT1 输入触发条件选择** 0x0：关闭 0x1：DCBH = low，DCBL = don't care 0x2：DCBH = high，DCBL = don't care 0x3：DCBL = low，DCBH = don't care 0x4：DCBL = high，DCBH = don't care 0x5：DCBL = high，DCBH = low 0x6：DCBL = high，DCBH = high 0x7：DCBL = low，DCBH = low	RW	0x0
21:19	DCAEVT2	**DCA EVT2 输入触发条件选择** 0x0：关闭 0x1：DCAH = low，DCAL = don't care 0x2：DCAH = high，DCAL = don't care 0x3：DCAL = low，DCAH = don't care 0x4：DCAL = high，DCAH = don't care 0x5：DCAL = high，DCAH = low	RW	0x0

		0x6：DCAL = high，DCAH = high 0x7：DCAL = low，DCAH = low		
18:16	DCAEVT1	**DCA EVT1 输入触发条件选择** 0x0：关闭 0x1：DCAH = low，DCAL = don't care 0x2：DCAH = high，DCAL = don't care 0x3：DCAL = low，DCAH = don't care 0x4：DCAL = high，DCAH = don't care 0x5：DCAL = high，DCAH = low 0x6：DCAL = high，DCAH = high 0x7：DCAL = low，DCAH = low	RW	0x0
15:12	DCBLCOMPSEL	**DCBL 输入源选择** DCBL（Digital Compare B Low Input）是 DC 输入源的 L 端 0x0：TZ1 input 0x1：TZ2 input 0x2：TZ3 input 0x3：TZ4 input 0x8：COMP1OUT input 0x9：COMP2OUT input 0xA：COMP3OUT input 0xB：COMP4OUT input 0xC：COMP5OUT input	RW	0x0
11:8	DCBHCOMPSEL	**DCBH 输入源选择** DCBH（Digital Compare B High Input）是 DC 输入源的 H 端 0x0：TZ1 input 0x1：TZ2 input 0x2：TZ3 input 0x3：TZ4 input 0x8：COMP1OUT input 0x9：COMP2OUT input 0xA：COMP3OUT input 0xB：COMP4OUT input 0xC：COMP5OUT input	RW	0x0
7:4	DCALCOMPSEL	**DCAL 输入源选择** DCAL（Digital Compare A Low Input）是 DC 输入源的 L 端 0x0：TZ1 input 0x1：TZ2 input 0x2：TZ3 input 0x3：TZ4 input 0x8：COMP1OUT input 0x9：COMP2OUT input 0xA：COMP3OUT input 0xB：COMP4OUT input	RW	0x0

		0xC：COMP5OUT input		
3:0	DCAHCOMPSEL	**DCAH 输入源选择** DCAH（Digital Compare A High Input）是 DC 输入源的 H 端 0x0：TZ1 input 0x1：TZ2 input 0x2：TZ3 input 0x3：TZ4 input 0x8：COMP1OUT input 0x9：COMP2OUT input 0xA：COMP3OUT input 0xB：COMP4OUT input 0xC：COMP5OUT input	RW	0x0

16.4.3.19. EPWMx_BLANKOFFSET

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	-	-	-
15:0	OFFSET	**Blanking 窗口偏移量** 配置 Blanking 窗口从起始到有效作用的偏移时间，起始点由 DCTRL[PULSESEL]决定。	RW	0x0

16.4.3.20. EPWMx_WINWIDTH

Bit(s)	Name	Description	R/W	Reset
31:12	reserved	-	-	-
11:0	WINDOW	**Blanking 窗口的长度** 0x0：Blanking 窗口不起作用 Other：Blanking 生效，单位为 TBCLK 的时间	RW	0x0

16.4.3.21. EPWMx_TZCTL

Bit(s)	Name	Description	R/W	Reset
31	DCBEVT1_OSTEN	**DCB EVT1 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
30	DCAEVT1_OSTEN	**DCA EVT1 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
29	OSHT6	**Trip-zone 6 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
28	OSHT5	**Trip-zone 5 OSHT（one-shot-trip）使能**	0x0	RW

		0x0：关闭 0x1：打开		
27	OSHT4	**Trip-zone 4 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
26	OSHT3	**Trip-zone 3 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
25	OSHT2	**Trip-zone 2 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
24	OSHT1	**Trip-zone 1 OSHT（one-shot-trip）使能** 0x0：关闭 0x1：打开	0x0	RW
23	DCBEVT2_CBCEN	**DCB EVT2 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
22	DCAEVT2_CBCEN	**DCA EVT2 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
21	CBC6	**Trip-zone 6 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
20	CBC5	**Trip-zone 5 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
19	CBC4	**Trip-zone 4 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
18	CBC3	**Trip-zone 3 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
17	CBC2	**Trip-zone 2 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
16	CBC1	**Trip-zone 1 CBC（cycle by cycle）使能** 0x0：关闭 0x1：打开	0x0	RW
15:12	reserved	-	-	-
11:10	DCBEVT2_FRCEN	**DCB EVT2 软件触发行为** 0x0：EPWMxB 输出高阻态 0x1：EPWMxB 输出高电平 0x2：EPWMxB 输出低电平 0x3：EPWMxB 保持原来的状态	0x0	RW
9:8	DCAEVT2_FRCEN	**DCA EVT2 软件触发行为**	0x0	RW

		0x0：EPWMxA 输出高阻态 0x1：EPWMxA 输出高电平 0x2：EPWMxA 输出低电平 0x3：EPWMxA 保持原来的状态		
7:6	DCBEVT1_FRCEN	**DCB EVT1 软件触发行为** 0x0：EPWMxB 输出高阻态 0x1：EPWMxB 输出高电平 0x2：EPWMxB 输出低电平 0x3：EPWMxB 保持原来的状态	0x0	RW
5:4	DCAEVT1_FRCEN	**DCA EVT1 软件触发行为** 0x0：EPWMxA 输出高阻态 0x1：EPWMxA 输出高电平 0x2：EPWMxA 输出低电平 0x3：EPWMxA 保持原来的状态	0x0	RW
3:2	TZB	**当 trip-zone 触发时，EPWMxB 行为设置** 0x0：EPWMxB 输出高阻态 0x1：EPWMxB 输出高电平 0x2：EPWMxB 输出低电平 0x3：EPWMxB 保持原来的状态	0x0	RW
1:0	TZA	**当 trip-zone 触发时，EPWMxA 行为设置** 0x0：EPWMxA 输出高阻态 0x1：EPWMxA 输出高电平 0x2：EPWMxA 输出低电平 0x3：EPWMxA 保持原来的状态	0x0	RW

16.4.3.22. EPWMx_TZFLAG

Bit(s)	Name	Description	R/W	Reset
31	reserved	-	-	-
30	DCBEVT2INTE	**DCB EVT2 中断使能** 0x0：关闭 0x1：打开	RW	0x0
29	DCBEVT1INTE	**DCB EVT1 中断使能** 0x0：关闭 0x1：打开	RW	0x0
28	DCAEVT2INTE	**DCA EVT2 中断使能** 0x0：关闭 0x1：打开	RW	0x0
27	DCAEVT1INTE	**DCA EVT1 中断使能** 0x0：关闭 0x1：打开	RW	0x0
26	OSTINTE	**One-Shot Trip Event 中断使能** 0x0：关闭 0x1：打开	RW	0x0
25	CBCINTE	**Cycle-by-Cycle Trip Event 中断使能**	RW	0x0

		0x0：关闭 0x1：打开		
24:23	reserved	-	-	-
22	CLRDCBEVT2	**DCB EVT2 标志位清除** 写 1 清除标志位	WO	0x0
21	CLRDCBEVT1	**DCB EVT1 标志位清除** 写 1 清除标志位	WO	0x0
20	CLRDCAEVT2	**DCA EVT2 标志位清除** 写 1 清除标志位	WO	0x0
19	CLRDCAEVT1	**DCA EVT1 标志位清除** 写 1 清除标志位	WO	0x0
18	CLROST	**One-Shot Trip Event 标志清除** 写 1 清除标志位	WO	0x0
17	CLRCBC	**Cycle-by-Cycle Trip Event 标志位清除** 写 1 清除标志位	WO	0x0
16	CLRINT	**中断标志清除** 写 1 清除标志位	WO	0x0
15	reserved	-	-	-
14	FRCDCBEVT2	**软件强制使能 Digital Compare Output B Event 2** 0x0：关闭 0x1：打开	WO	0x0
13	FRCDCBEVT1	**软件强制使能 Digital Compare Output B Event 1** 0x0：关闭 0x1：打开	WO	0x0
12	FRCDCAEVT2	**软件强制使能 Digital Compare Output A Event 2** 0x0：关闭 0x1：打开	WO	0x0
11	FRCDCAEVT1	**软件强制使能 Digital Compare Output A Event 1** 0x0：关闭 0x1：打开	WO	0x0
10	FRCOST	**软件强制使能 One-Shot Trip Event** 0x0：关闭 0x1：打开	WO	0x0
9	FRCCBC	**软件强制使能 Cycle-by-Cycle Trip Event** 0x0：关闭 0x1：打开	WO	0x0
8:7	reserved	-	-	-
6	DCBEVT2_FLAG	**Digital Compare Output B Event 2 标志位** 0x0：没有触发 0x1：有信号触发	RW	0x0
5	DCBEVT1_FLAG	**Digital Compare Output B Event 1 标志位**	RW	0x0

		0x0：没有触发 0x1：有信号触发		
4	DCAEVT2_FLAG	**Digital Compare Output A Event 2 标志位** 0x0：没有触发 0x1：有信号触发	RW	0x0
3	DCAEVT1_FLAG	**Digital Compare Output A Event 1 标志位** 0x0：没有触发 0x1：有信号触发	RW	0x0
2	OST	**One-Shot Trip 状态标志位** 0x0：没有触发 0x1：有信号触发	RW	0x0
1	CBC	**Cycle-By-Cycle Trip Event 状态标志位** 0x0：没有触发 0x1：有信号触发	RW	0x0
0	INT	**TZ 中断的标志位** 0x0：没有中断触发 0x1：有中断触发	RW	0x0

16.4.3.23. EPWMx_DCCAP

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	-	-	
15:0	DCCAP	**DC 模块捕获的数值** 使能 DCCTL[CAPE]后，捕获的数据值会存在该寄存器中 ● 如果 DCCTL[SHDWMODE] = 0，则启用影子模式。在这种模式下，寄存器被复制到由 dcctl [PULSESEL]位定义的 TBCTR = TBPRD 或 TBCTR = 0 上的影子寄存器。CPU 读取此寄存器将返回影子寄存器值。 ● 如果 DCCTL[SHDWMODE] = 1，则影子寄存器被禁用。在这种模式下，cpu 读取时将返回寄存器值。	RW	0x0

16.4.3.24. EPWM_TTCTL

Bit(s)	Name	Description	R/W	Reset
31:24	reserved	-	-	-
23	LVDERR	**LVD error 标志位**	RO	0x0
22	WDTERR	**Watchdog error 标志位**	RO	0x0
21	HOCERR	**HighspeedOSC error 标志位**	RO	0x0
20	SYSERR	**System error 标志位**	RO	0x0
19	LVDERR_TZEN	**LVD error to TripZone 5 使能** 0x0：关闭	RW	0x0

		0x1：打开		
18	WDTERR_TZEN	**Watchdog error to TripZone 5 使能** 0x0：关闭 0x1：打开	RW	0x0
17	HOCERR_TZEN	**Highspeed OSC error to TripZone 5 使能** 0x0：关闭 0x1：打开	RW	0x0
16	SYSERR_TZEN	**System error to Tripzone 5 使能** 0x0：关闭 0x1：打开	RW	0x0
15	EPWM3_TZINT	**EPWM3 Trip zone 标志位**	R	0x0
14	EPWM2_TZINT	**EPWM2 Trip zone 标志位**	R	0x0
13	EPWM1_TZINT	**EPWM1 Trip zone 标志位**	R	0x0
12	EPWM0_TZINT	**EPWM0 Trip zone 标志位**	R	0x0
11	EPWM3_ETINT	**EPWM3 event triger 标志位**	R	0x0
10	EPWM2_ETINT	**EPWM2 event triger 标志位**	R	0x0
9	EPWM1_ETINT	**EPWM1 event triger 标志位**	R	0x0
8	EPWM0_ETINT	**EPWM0 event triger 标志位**	R	0x0
7:5	reserved	-	-	-
4	CPUTZ	**软件控制 TZ 数值** Write 1 will force tz6 to 0. Write 0 will force tz6 to 1.	RW	0x0
3	EPWM3_EN	**EPWM3 使能** 0x0：关闭 0x1：打开	RW	0x0
2	EPWM2_EN	**EPWM2 使能** 0x0：关闭 0x1：打开	RW	0x0
1	EPWM1_EN	**EPWM1 使能** 0x0：关闭 0x1：打开	RW	0x0
0	EPWM0_EN	**EPWM0 使能** 0x0：关闭 0x1：打开	RW	0x0

17. WDT

17.1. 简介

Watchdog 工作时钟为 32K。默认定时 2S，可以通过修改分频系数改变喂狗时间。通过

配置寄存器，可以选择当计时溢出时，复位系统或者产生中断。

17.2. 寄存器

17.2.1. 寄存器基地址

Name	Base Address	Description
WDT	0x40001000	WDT 基地址

17.2.2. 寄存器列表

Offset Address	Name	Description
0x0000	WDT_CON	控制寄存器
0x0004	WDT_KEY	秘钥寄存器

17.2.3. 寄存器详细说明

17.2.3.1. WDT_CON

Bit(s)	Name	Description	R/W	Reset
31:10	reserved		-	-
9	wake_en	**Wakeup 使能** 0x0：关闭 0x1：打开	RO	0x0
8:7	reserved	-	-	-
6	wdt_pend	**WDT 计数满标志** WDT_KEY 写 0xaaaa，清除 wdt_pending;	RO	0x0
5	int_enable	**中断使能** 0x0：计数满时复位系统 0x1：计数满时产生中断	RO	0x0
4	wdte	写 WDT_KEY=0xCCCC，置位 写 WDT_KEY=0xDDDD，复位	RW	0x1
3:0	wdt_psr	**分频系数配置** 每次配置该位域之前必须先写 WDT_KEY=0x5555 0x0：不分频	RW	0x8

		0x1：2 0x2：4 0x3：8 0x4：16 0x5：32 0x6：64 0x7：128 0x8：256 0x9：512 0xA：1024 0xB：2048 0xC：4096 0xD：8192 0xE：16384 0xF：32768 看门狗复位时间=1/32K*256*分频系数		

17.2.3.2. WDT_KEY

Bit(s)	Name	Description	R/W	Reset
31:16	reserved	-	-	-
15:0	KEY	KEY[15：0]：键值（只写寄存器，读出值为 0x0000）（Key value） 软件必须以一定的间隔写入 0xAAAA，否则，当计数器为 0 时，看门狗会产生复位。当 pending 为 1 的时候，写 0xAAAA 会清除 pending。 写入 0x5555 表示允许访问 WDT_PSR 写入 0xCCCC，启动看门狗工作 写入 0xDDDD，关闭看门狗 写 0xaaaa，清除 wdt_pending 写入 0x55AA，开启中断使能 写入 0xAA55，关闭中断使能 写入 0x5A5A，开启 wake up 使能 写入 0xA5A5，关闭 wake up 使能	RO	0x0