AC632N 用户手册

珠海市杰理科技股份有限公司

Zhuhai Jieli Technologyco.,LTD

2021 年 4 月

修改记录

版本	更新日期	描述
V1.0	2021-04-08	初版

目录

序言

本系列芯片是低成本超低功耗蓝牙数传系统级 SOC，本芯片面向智能家居，物联网控制等无线数传透传智能设备。芯片集成了 32 位 CPU 支持浮点与数学函数加速运算，并内置蓝牙调制解调器、基带及模拟 RF 模块，支持蓝牙 V2.1/V4.2/V5.1 版本，支持超低功耗蓝牙广播、连接待机、组网传输等，具有低延时远距离数据传输能力，满足一般音乐、通话、智能控制、传感收集数据传输及处理要求。同时自带 PMU 模块提供多种低功耗工作模式，能使用 LDO 或 DCDC 供电模式，并可为锂电池充电；此外还提供了 USB、按键 ADC、IIC、SPI、Q-decoder、MCPWM、LED CONTROL 等丰富的外设接口。

本芯片主推 BluetoothSmart 和 SmartReady 应用方案，并提供了蓝牙双模的单芯片解决方案，减少设计的工作量让产品更快赢得市场。

本芯片集成了低成本、超低功耗、2.4G 射频模块，并提供极低的射频活动功耗和 MCU 功耗，支持低功耗模式，出色的电池使用寿命使其适合功耗敏感的应用。

本芯片提供了完整的 FCC 与 BQB 的认证，使客户在该系列下开发自己的产品时减少了认证的费用。

本芯片集成了一整套的蓝牙，为用户开发提供了方便的环境，为产品提供了无限的可能。

第 1 章.总体介绍

1.1. 概述

CPU

❖32 位浮点 CPU，最高 96MHz

Memory

❖片上集成了共 73K 字节 SRAM

❖8K 2-Way Icache + 1k 1way Rocache (占 用 SRAM)

中断控制器

❖64 个中断源，8 级可编程中断优先级

❖支持 16 个外部 I/O 中断，中 8 个可低功耗唤醒

❖带软中断（虚拟中断）功能，优先级可以配置

数字 I/O

❖最多 30 个可编程数字 I/O 引脚

❖不同的功能引脚有不同的电流选择，最大拉/灌电流为 64 mA（HD IO）24mA（其它 IO）

❖可配置上拉 10K、下拉 10K 功能

复杂设备

❖两路独立 Full Speed USB OTG 控制器，除 EP0 外带四个 In EP、四个 Out EP

❖V2.1/ V4.2/ V5.1 版本蓝牙

❖数学加速引擎（三角函数、对数、幂、开方以及单精度浮点运算）

数字模块

❖4 个 32 位异步分频可重载定时器，可用作计时、捕捉，PWM 模式 (Timer0/1/2/3)

❖硬件看门狗（Watchdog）

❖3 个 UART 串口控制器,支持 DMA

❖3 个 SPI 控制器，支持 1/2/4 线，支持 DMA

❖SPI FLASH 控制器，只支持跑代码

❖IIC 控制机，支持主从机

❖3 个正交解码器 ，可同时工作

❖支持低功耗 RTC，闹钟和时基唤醒

❖MCPWM 电机驱动模块，3 对 6 路 PWM 输出，支持故障保护及死区控制

❖支持低功耗指示灯

❖支持灯带控制

❖支持 crossbar 功能，数字模块输入输出可以影射到任意 IO。

❖支持 12 路 IO 低功耗唤醒

模拟模块

❖10 位精度 16 通道 ADC（ADC Key/电压检测等）

❖LVD，支持掉电保护

❖PMU，支持软开关功能，蓝牙 sniff 低功耗，支持电池充电，可独立供电。

❖PLL，中心频率为 96-192MHz

❖RC，内部 RC 时钟，频率在 16MHz 左右

❖LRC，内部低温漂 RC 时钟，频率在 32kHz 左右

❖高精度高速 OSC（12/24/40M）

❖高精度低速 32k OSC

❖64 位 EFUSE

工作范围

❖工作电压 1.8～5.5V

❖工作温度（-40° C 至+105° C）

1.2. 系统框图

本芯片系统框图如下：

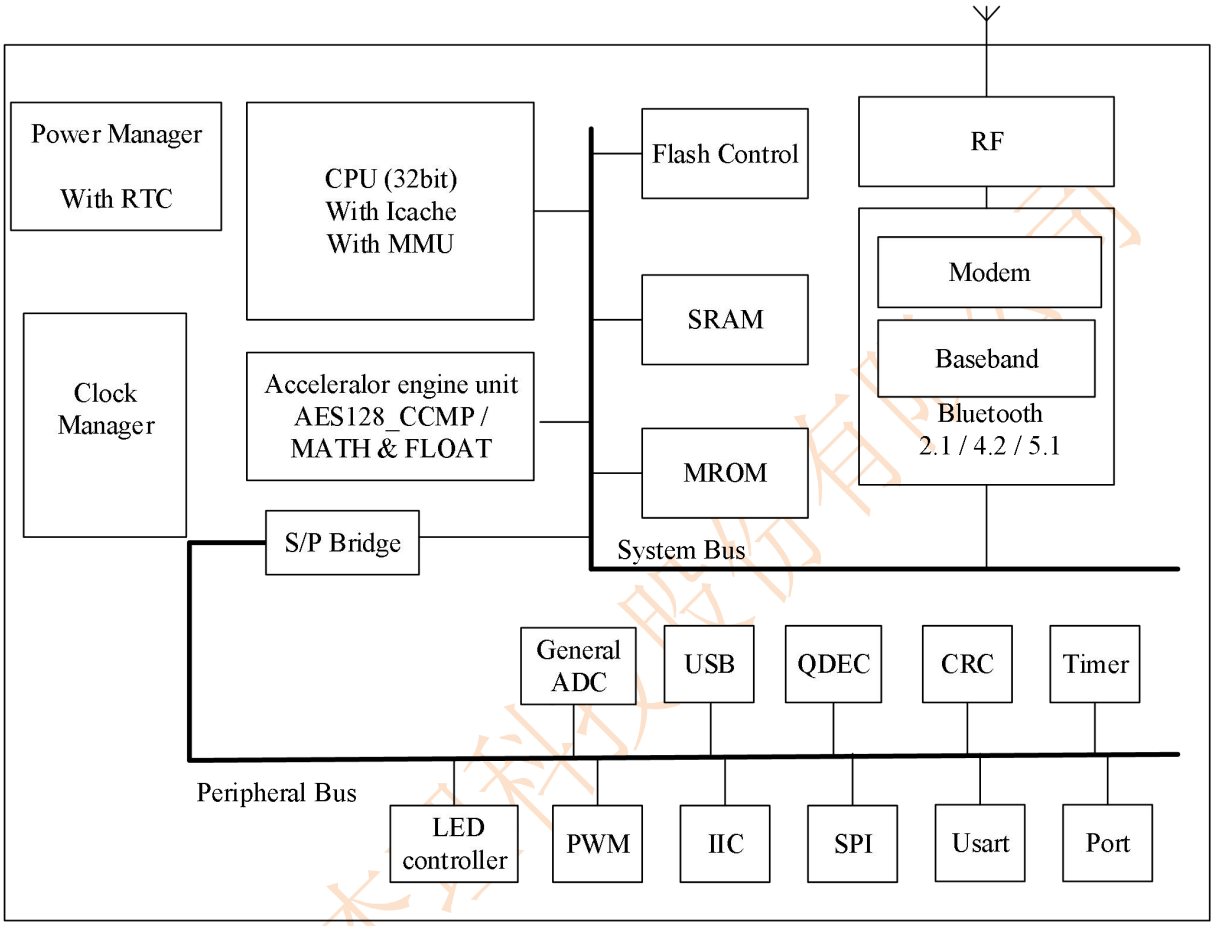


图 1-1 本芯片系统框图

第 2 章.时钟系统（Clock_System）

2.1. 概述

1. 本芯片具备如下 2 个原生时钟源，可直接驱动系统运行：

rc_clk: 来自 RC 振荡器，振荡频率约 16MHz，随电压和温度不同会有较大变化

bt_osc:来自蓝牙模块的振荡器，外部需在 BTXOSCO 和 BTXOSCI 引脚挂载 12-26MHz 晶振（内部已有谐振电容，也可外置）， 可输出 bt_osc_x2 频率。

32k_osc: 来源于 RTC， 需要在引脚挂负载电容。

2.额外的，本芯片还具备如下 1 个原生时钟源，不可直接驱动系统运行，只能驱动 PMU：

lrc_clk：来自 PMU 的特殊 RC 振荡器，振荡频率约 32KHz，随电压和温度变化较小，可用作蓝牙 sniff 或实时时钟计时。

3.本芯片还具备如下 1 个衍生时钟源：

pll_src:来自片内 PLL 的输出，范围位 96MHz~240MHz，同时输出 PLL1D，PLL1.5D，PLL2.5D，PLL3.5D，PLL4.5D 的时钟，即如果 PLL 设定为 192MHz，则能输出 192MHz、128MHz、76.8MHz、54.86MHz 和 42.67MHz；每个时钟都有独立的使能端(PLL_CON3[0:4])。在不使用该时钟时，软件应关闭其使能端以防止额外电源消耗。

第 3 章.输入/输出（I/O）

3.1. IOMC

3.1.1. 概述

IOMC 控制寄存器主要用于控制 IO 除 crossbar 之外的一些拓展功能的配置。

3.1.2. 控制寄存器

1. IOMC0

Bit	Name	RW	Default	Description
27	Lan_pa_och_en	rw	0	
26-25		rw	0	
24	Wlc_ext_ios	rw	0	
23	Wlc_freq_ioen	rw	0	
22	Wlc_sta_ioen	rw	0	
21	wlc_act_ioen	rw	0	
20	cap_mux_edge	rw	0	input channel 2 的 CAP 边沿选择（用于 GPCNT、PCNT） 0：上升沿； 1：下降沿；
19	spi0_mix_mode	rw	0	SPI0 输入结果产生选项 0：SPI0 内部输入为 DI； 1：SPI0 内部输入为 DI&DO；
18-17	sdtap_ios	rw	0	见下表
16	sfc_ios	rw	0	SFC 模块 IO 选择

	00	01	10	11
CK	PC4	USBDP	PB1	PB6
DA	PC5	USBDM	PB2	PB7

				sel	CS	CK	D0	D1	D2	D3
				0	PD3	PD0	PD1	PD2	PB4	PC3
				1	PD0	PD6	PD7	PD1	PC5	PA5

2. IOMC1

Bit	Name	RW	Default	Description
19-16	wl_ich_sel	rw	0	输入选择：(同 wkup_ich_sel)
15-12	clk_ich_sel	rw	0	输入选择：(同 wkup_ich_sel)
11-8	cap_ich_sel	rw	0	输入选择：(同 wkup_ich_sel)
7-4	irflt_ich_sel	rw	0	输入选择：(同 wkup_ich_sel)
3-0	wkup_ich_sel	rw	0	输入选择： 0：GP_ICH0 1：GP_ICH1 2：GP_ICH2 3：GP_ICH3 4：GP_ICH4 5：GP_ICH5 6：GP_ICH6 7：GP_ICH7 8：TMR0_PWM 9：TMR1_PWM

3. IOMC2

Bit	Name	RW	Default	Description
29-25	gp_och5_sel	rw	0	输出选择：（同 gp_och0_sel）
24-20	gp_och4_sel	rw	0	输出选择：（同 gp_och0_sel）
19-15	gp_och3_sel	rw	0	输出选择：（同 gp_och0_sel）
14--10	gp_och2_sel	rw	0	输出选择：（同 gp_och0_sel）

9-5	gp_och1_sel	rw	0	输出选择：（同 gp_och0_sel）
4-0	gp_och0_sel	rw	0	输出选择： 0：PMU_DBG 1：CLK_OUT0 2：CLK_OUT1 3：CLK_OUT2 4：CLK_OUT3 5：WLC_INT_ACT 6：WLC_INT_STA 7：WLC_INT_FRQ 8：WL_DBG0 9：WL_DBG1 10：WL_DBG2 11：WL_DBG3 12：WL_DBG4 13：WL_DBG5 14：WL_DBG6 15：WL_DBG7 16：Lan_pa_och_en？ PA_EN： DCDC_TEST0 17：Lan_pa_och_en？ LNA_EN： DCDC_TEST1

4. IOMC3

Bit	Name	RW	Default	Description
29-25	gp_och11_sel	rw	0	输出选择：（同 gp_och0_sel）
24-20	gp_och10_sel	rw	0	输出选择：（同 gp_och0_sel）
19-15	gp_och9_sel	rw	0	输出选择：（同 gp_och0_sel）
14-10	gp_och8_sel	rw	0	输出选择：（同 gp_och0_sel）

9-5	gp_och7_sel	rw	0	输出选择：（同 gp_och0_sel）
4-0	gp_och6_sel	rw	0	输出选择：（同 gp_och0_sel）

3.2. 引脚通用功能表

端口	可复用的功能								
PA0	耐高压	CLKOUT1						UART2TXB/RXB	PWMCH0H
PA1			PWM0	UART1_CTS	Q-decoder0_0		ADC0	UART0TXC	FPIN3
PA2			CAP3	UART1_RTS	Q-decoder0_1			UART0RXC	FPIN0
PA3		SPI1DAT(2)	CAP2		BT_Active	IIC_SCL_D	ADC1	UART2TXA	PWMCH0L
PA4		SPI1DAT(3)	PWM1		Wlan_Active	IIC_SDA_D		UART2RXA	FPIN1
PA5			TMR0	SPI2DIB	BT_priority		ADC2	UART0TXA	TMR0CK
PA6			CAP0	SPI1DIA(1)	BT_Freq			UART0RXA	TMR1CK
PA7		PA_EN	TMR1	SPI1CLKA		IIC_SCL_C	ADC3	UART1TXC	PWMCH1H
PA8		LNA_EN	TMR3	SPI1DOA(0)		IIC_SDA_C	ADC4	UART1RXC	PWMCH1L
PA9(上拉)		EVDD/PVDD	默认长按Reset				ADC8		
USB1DP(下拉)				SPI1CLKB		IIC_SCL_B	ADC5	UART2TXD	

UDB1DM(下拉)				SPI1DOB(0)		IIC_SDA_B	ADC6	UART2RXD	
PB0	耐高压	CLKOUT0						UART1TXB	TMR2CK
PB1		LVD	PWM2				ADC7	UART1RXB	
PB2(上拉)		MCLR			Q-decoder1_0			UART0TXB	PWMCH2H
PB3	耐高压				Q-decoder1_1			UART0RXB	PWMCH2L
PB4			TMR2	SPI1DIB(1)	Q-decoder2_0		ADC9	UART1TXA	PWMCH3H
PB5	耐高压			SPI2DIA	Q-decoder2_1			UART1RXA	PWMCH3L
PB6				SPI2CLKA			ADC12	UART2TXC	TMR3CK
PB7	耐高压			SPI2DOA				UART2RXC	FPIN2
PB8		32K_OSCO	SFC_DAT3A(3)(flash_hold)	spi0_DAT3A(3)					
PB9		32K_OSCI	SFC_DAT2A(2)	spi0_DAT2A(2)					
PD0			SFC_CLKA	spi0_CLKA					
PD1			SFC_DOA(0)	spi0_DOA(0)					
PD2			SFC_DIA(1)	spi0_DIA(1)					

PD3			SFC_CSA	spi0_CSA					
PD4			Flash 供电脚						
USB0DP(下拉)				SPI2CLKB		IIC_SCL_A	ADC10	UART1TXD	
UDB0DM(下拉)				SPI2DOB		IIC_SDA_A	ADC11	UART1RXD	
PP0(LDO5v)	耐高压		PWM3/CAP1					UART0TXD/UART0RXD	
P00	耐高压				小 CPU 专用 IO				

3.3. PORT CONTROL CROSSBAR

3.3.1. 概述

Crossbar 是一个 IO 输入输出任意映射模块，通过 Crossbar 可以把一些特定通用模块的输出映射到任意 IO 口输出，同时也可以通过 Crossbar 把任意一个 IO 的输入映射到特定通用模块。本芯片支持 Crossbar 功能的模块有：OUTPUTCHANNEL0/1/2/3/4/5/6/7 、INPUTCHANNEL0/1/2/3/4/5/6/7、TIMER0/1/2/3、SPI0/1/2/3、UART0/1/2、IIC、MCPWM、RDEC0/1/2、LDC 。

3.3.2. Crossbar 功能结构图

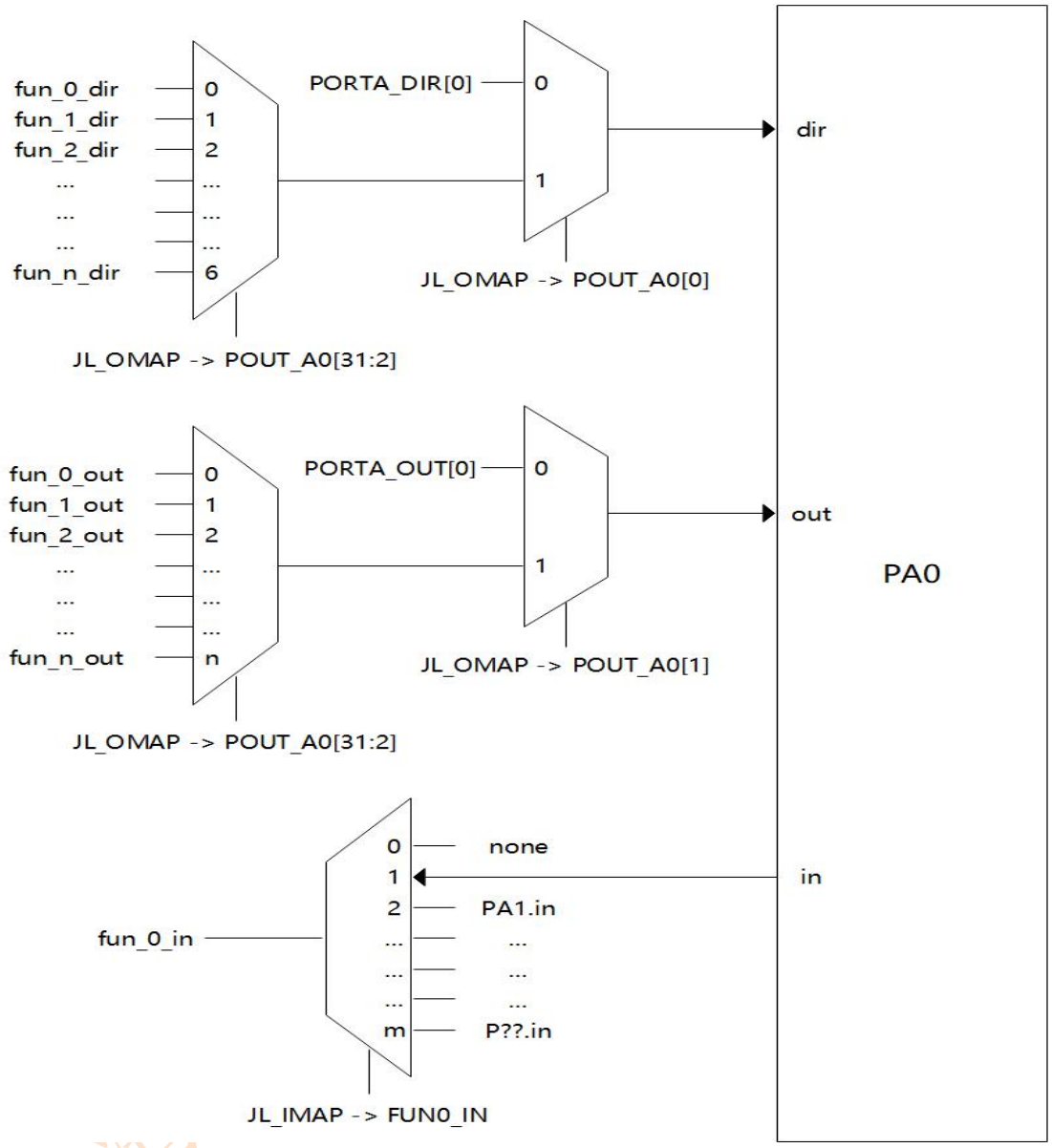


图 3- 1 Crossbar 功能结构图

3.3.3. 寄存器列表

Crossbar 模块输入映射寄存器	Crossbar IO 输出映射寄存器
JL_IMAP->FI_GP_ICH0	JL_OMAP->PA0_OUT
JL_IMAP->FI_GP_ICH1	JL_OMAP->PA1_OUT

JL_IMAP->FI_GP_ICH2	JL_OMAP->PA2_OUT
JL_IMAP->FI_GP_ICH3	JL_OMAP->PA3_OUT
JL_IMAP->FI_GP_ICH4	JL_OMAP->PA4_OUT
JL_IMAP->FI_GP_ICH5	JL_OMAP->PA5_OUT
JL_IMAP->FI_GP_ICH6	JL_OMAP->PA6_OUT
JL_IMAP->FI_GP_ICH7	JL_OMAP->PA7_OUT
JL_IMAP->FI_TMR0_CIN	JL_OMAP->PA8_OUT
JL_IMAP->FI_TMR0_CAP	JL_OMAP->PA9_OUT
JL_IMAP->FI_TMR1_CIN	JL_OMAP->PB0_OUT
JL_IMAP->FI_TMR1_CAP	JL_OMAP->PB1_OUT
JL_IMAP->FI_TMR2_CIN	JL_OMAP->PB2_OUT
JL_IMAP->FI_TMR2_CAP	JL_OMAP->PB3_OUT
JL_IMAP->FI_TMR3_CIN	JL_OMAP->PB4_OUT
JL_IMAP->FI_TMR3_CAP	JL_OMAP->PB5_OUT
JL_IMAP->FI_SPI0_CLK	JL_OMAP->PB6_OUT
JL_IMAP->FI_SPI0_DA0	JL_OMAP->PB7_OUT
JL_IMAP->FI_SPI0_DA1	JL_OMAP->PB8_OUT
JL_IMAP->FI_SPI0_DA2	JL_OMAP->PB9_OUT
JL_IMAP->FI_SPI0_DA3	JL_OMAP->PD0_OUT
JL_IMAP->FI_SPI1_CLK;	JL_OMAP->PD1_OUT
JL_IMAP->FI_SPI1_DA0;	JL_OMAP->PD2_OUT
JL_IMAP->FI_SPI1_DA1;	JL_OMAP->PD3_OUT
JL_IMAP->FI_SPI1_DA2;	JL_OMAP->PD4_OUT
JL_IMAP->FI_SPI1_DA3;	JL_OMAP->USB0DP_OUT
JL_IMAP->FI_SPI2_CLK;	JL_OMAP->USB0DM_OUT
JL_IMAP->FI_SPI2_DA0;	JL_OMAP->USB1DP_OUT
JL_IMAP->FI_SPI2_DA1;	JL_OMAP->USB1DM_OUT
JL_IMAP->FI_SPI2_DA2;	JL_OMAP->PP0_OUT

JL_IMAP->FI_SPI2_DA3; JL_IMAP->FI_MCPWM_TMR0_CLKI JL_IMAP->FI_MCPWM_TMR1_CLKI JL_IMAP->FI_MCPWM_TMR2_CLKI JL_IMAP->FI_MCPWM_TMR3_CLKI JL_IMAP->FI_MCPWM_FPIN_A JL_IMAP->FI_MCPWM_FPIN_B JL_IMAP->FI_MCPWM_FPIN_C JL_IMAP->FI_MCPWM_FPIN_D JL_IMAP->FI_IIC_SCL JL_IMAP->FI_IIC_SDA JL_IMAP->FI_UART0_RX JL_IMAP->FI_UART1_RX JL_IMAP->FI_UART1_CTS JL_IMAP->FI_UART2_RX JL_IMAP->FI_RDEC0_DAT0 JL_IMAP->FI_RDEC0_DAT1 JL_IMAP->FI_RDEC1_DAT0 JL_IMAP->FI_RDEC1_DAT1 JL_IMAP->FI_RDEC2_DAT0 JL_IMAP->FI_RDEC2_DAT1	

3.3.4. 寄存器功能

1. JL_OMAP -> XX_OUT ： IO 输出功能映射配置

Bit	Name	RW	Default	Description
31-2	FUN_SELECT	rw	0	Crossbar 输出功能选择： FO_GP_OCH0 0

				FO_GP_OCH1	1
				FO_GP_OCH2	2
				FO_GP_OCH3	3
				FO_GP_OCH4	4
				FO_GP_OCH5	5
				FO_GP_OCH6	6
				FO_GP_OCH7	7
				FO_GP_OCH8	8
				FO_GP_OCH9	9
				FO_GP_OCH10	10
				FO_GP_OCH11	11
				FO_TMR0_PWM	12
				FO_TMR1_PWM	13
				FO_TMR2_PWM	14
				FO_TMR3_PWM	15
				FO_SPI0_CLK	16
				FO_SPI0_DA0	17
				FO_SPI0_DA1	18
				FO_SPI0_DA2	19
				FO_SPI0_DA3	20
				FO_SPI1_CLK	21
				FO_SPI1_DA0	22
				FO_SPI1_DA1	23
				FO_SPI1_DA2	24
				FO_SPI1_DA3	25
				FO_SPI2_CLK	26
				FO_SPI2_DA0	27
				FO_SPI2_DA1	28

				FO_SPI2_DA2 29 FO_SPI2_DA3 30 FO_MCPWM_CH0L 31 FO_MCPWM_CH0H 32 FO_MCPWM_CH1L 33 FO_MCPWM_CH1H 34 FO_MCPWM_CH2L 35 FO_MCPWM_CH2H 36 FO_MCPWM_CH3L 37 FO_MCPWM_CH3H 38 FO_IIC_SCL 39 FO_IIC_SDA 40 FO_LEDC0_DO 41 FO_LEDC1_DO 42 FO_UART0_TX 43 FO_UART1_TX 44 FO_UART1_RTS 45 FO_UART2_TX 46
1	FUN_OUT_EN	rw	0	Crossbar 数据控制使能： 0：CPU 数据 1：外设数据
0	FUN_DIR_EN	rw	0	Crossbar 方向控制使能： 0：CPU 控制 1：外设控制 （注：若所选外设无方向控制功能，则都由 CPU 控制）

2. JL_IMAP -> FI_XX ： 模块功能输入 IO 映射配置

Bit	Name	RW	Default	Description
31-0	INPUT_SELECT	rw	0	Crossbar 输入引脚选择： PA0_IN 1 PA1_IN 2 PA2_IN 3 PA3_IN 4 PA4_IN 5 PA5_IN 6 PA6_IN 7 PA7_IN 8 PA8_IN 9 PA9_IN 10 PB0_IN 11 PB1_IN 12 PB2_IN 13 PB3_IN 14 PB4_IN 15 PB5_IN 16 PB6_IN 17 PB7_IN 18 PB8_IN 19 PB9_IN 20 PD0_IN 21 PD1_IN 22 PD2_IN 23 PD3_IN 24 PD4_IN 25 USB0DP_IN 26

				USB0DM_IN 27 USB1DP_IN 28 USB1DM_IN 29 PP0_IN 30

第 4 章.32 位定时器(TIMER32)

4.1. 概述

Timer32 是一个集合了定时/计数/捕获功能于一体的多动能 32 位定时器。它的驱动源可以选择片内时钟或片外信号。它带有一个可配置的最高达 64 的异步预分频器，用于扩展定时时间或片外信号的最高频率。它还具有上升沿/下降沿捕获功能，可以方便的对片外信号的高电平/低电平宽度 进行测量。

4.2. 控制寄存器

寄存器列表	TIMER0	TIMER1	TIMER2	TIMER3
Tx_CON	T0_CON	T1_CON	T2_CON	T3_CON
Tx_CNT	T0_CNT	T1_CNT	T2_CNT	T3_CNT
Tx_PRD	T0_PRD	T1_PRD	T2_PRD	T3_PRD
Tx_PWM	T0_PWM	T1_PWM	T2_PWM	T3_PWM

4.3. 寄存器说明

1.Tx_CON： timer x control register

Bit	Name	RW	Default	Description
31-16	reserved	r	0	预留
15	PND	r	0	PND： 中断请求标志，当 timer 溢出或产生捕获动作时会被硬件置 1，需要由软件清 0。
14	PCLR	w	0	PCLR： 软件在此位写入‘1’将清除 PND 中断请求标志。
13-10	SSEL[3:0]	rw	0	SSEL3-0：timer 驱动源选择 注意：选中的时钟需要使用上面 PSET 分频到(lsb_clk/2)以下！

				0：使用 LSB 时钟作为 timer 的驱动源； 1：使用 OSC 口信号作为 timer 的驱动源； 2：使用 HTC 时钟作为 timer 的驱动源； 3：使用 LRC 时钟作为 timer 的驱动源； 4：使用 RC_16M 时钟作为 timer 的驱动源； 5：使用 STD_12M 时钟作为 timer 的驱动源； 6：使用 STD_24M 时钟作为 timer 的驱动源； 7：使用 STD_48M 时钟作为 timer 的驱动源； 15：使用 IO mux in 时钟作为 timer 的驱动源；（crossbar）
9	PWM_INV	rw	0	PWM_INV：PWM 信号输出反向。
8	PWM_EN	rw	0	PWM_EN： PWM 信号输出使能。此位置 1 后，相应 IO 口的功能将会被 PWM 信号输出替代。
7-4	PSET[3:0]	rw	0	PSET3-0：预分频选择位 0000：预分频 1 0001：预分频 4 0010：预分频 16 0011：预分频 64 0100：预分频 1*2 0101：预分频 4*2 0110：预分频 16*2 0111：预分频 64*2 1000：预分频 1*256 1001：预分频 4*256 1010：预分频 16*256 1011：预分频 64*256 1100：预分频 1*2*256 1101：预分频 4*2*256 1110：预分频 16*2*256

				1111：预分频 64*2*256
3-2	CSEL[1:0]	rw	0	捕获模式端口选择： 0：IO mux in（crossbar） 1：IRFLT_OUT
1-0	MODE[1:0]	rw	0	MODE1-0：工作模式选择 00： timer 关闭； 01： 定时/计数模式； 10： IO 口上升沿捕获模式（当 IO 上升沿到来时，把 TxCNT 的值捕捉到 TxPR 中）； 11： IO 口下降沿捕获模式（当 IO 下降沿到来时，把 TxCNT 的值捕捉到 TxPR 中）。

2.Tx_CNT：timer x counter register

Bit	Name	RW	Default	Description
31-0	Tx_CNT	rw	0	timer32 的计数寄存器

3.Tx_PR：timer x period register

Bit	Name	RW	Default	Description
31-0	Tx_PR	rw	0	timer32 的周期寄存器

在定时/计数模式下，当 Tx_CNT == Tx_PR 时，Tx_CNT 会被清 0。

在上升沿/下降沿捕获模式下，TxPR 是作为捕获寄存器使用的，当捕获发生时，Tx_CNT 的值会被复制到 Tx_PR 中。而此时 Tx_CNT 自由的由 0-4294967295-0 计数，不会和 Tx_PR 进行比较清 0。

4.Tx_PWM：timer x PWM register

Bit	Name	RW	Default	Description
31-0	Tx_PWM	rw	0	Timer32 的 PWM 设置寄存器

在 PWM 模式下，此寄存器的值决定 PWM 输出的占空比。占空比 N 的计算公式如下：

N = (Tx_PWM / Tx_PR) * 100%

此寄存器不带有缓冲，写此寄存器的动作将可能导致不同步状态产生的 PWM 波形占空比瞬间过大或过小的问题。

第 5 章.ADC

5.1. 概述

10Bit ADC(A/D 转换器)，其时钟最大不可超过 1MHz。

5.2. 寄存器说明

1. ADC_CON: ADC control register

Bit	Name	RW	Default	Description
31-18	-	-	0	预留
17	ADC_CLKEN	rw	0	ADC 控制器工作时钟 adc_clk 使能
16	ADCISEL	rw	0	ADC CMP 功率选择，写‘1’选择低功率
15-12	WAIT_TIME	rw	0	WAIT_TIME： 启动延时控制，实际启动延时为此数值乘 8 个 ADC 时钟
11-8	CH_SEL	rw	0	CH_SEL：通道选择 0000：选择 PA1 0001：PA3 0010：PA5 0011：PA7 0100：PA8 0101：USBD1P 0110：USBD1M 0111：PB1 1000：PA9 1001：PB4 1010：USBD0P 1011：USBD0M 1100：PB6

				1101：PMU_SYSPLL 1110：OSC32k 1111：BT
7	PND	r	0	PND：只读，中断请求位，当 ADC 完成一次转换后，此位会被设置为‘1’，需由软件清‘0’
6	CPND	w	0	CPND：只写，写‘1’清除中断请求位，写‘0’无效(该 bit 充当 kst，对该写‘1’将启动 ADC 转换)
5	ADC_IE	rw	0	ADC_IE：ADC 中断允许
4	ADC_EN	rw	0	ADC_EN：ADC 控制器使能
3	ADC_AE	rw	0	ADC_AE：ADC 模拟模块使能
2-0	ADC_BAUD	rw	0	ADC_BAUD： ADC 时钟频率选择 000：LSB 时钟 1 分频 001：LSB 时钟 6 分频 010：LSB 时钟 12 分频 011：LSB 时钟 24 分频 100：LSB 时钟 48 分频 101：LSB 时钟 72 分频 110：LSB 时钟 96 分频 111：LSB 时钟 128 分频

2.P3_ANA_CON4 ANA control SFR（PMU to ADC）

Bit	Name	RW	default	Description
7	WVBG_TOADC_EN	rw	0	WVBG output to ADC enable
6	MVBG_TOADC_EN	rw	0	MVBG output to ADC enable
5	VBG_BUFFER_EN	rw	0	VBG voltage ADCDET enable
4-1	CHANNEL_ADC_S3-0	rw	0	ADC channel select: 0000(default): VBG08 0001: VDC12

				0010: SYSVDD 0011: VTEMP 0100: PROGF 0101: 1/4 VBAT 0110: 1/4 LDO5v 0111: WVDD 1000: PVDD 1001: RVDD 1010: VSW 1011: PROGI 1100: EVDD 1101: VBGW08 1110: - 1111: -
0	PMU_DET_OE	rw	0	PMU voltage detect to ADC output enable

3.PLL_CON0: pll to adc

Bit	Name	RW	Default	Description
23	PLL_TEST_EN	rw	0	pll to adc enable
22-21	PLL_TEST_S<1:0>	rw	0	00: test 5u current 01: test the 1.1v VCO voltage 10: test the 1.1v DIV voltage 11: test output 192meg colck

4.BT to adc

en: WLA_CON4[27]

第 6 章. CRC

6.1. 概述

CRC16 计算器，每次运算 8bit，多项式为： $X^{16}+X^{12}+X^{5}+X^{1}$ 。

6.2. 寄存器说明

1. JL_CRC->REG: CRC16 校验码

Bit	Name	RW	Default	Description
31-16	-	-	0	预留
15-0	CRC_REG	rw	0	写入初始值，CRC 计算完毕，读取校验码

2.JL_CRC->FIFO: 运算数据输入

Bit	Name	RW	Default	Description
31-8	-	-	0	预留
7-0	CRC_FIFO	rw	0	运算数据输入，HSB 先运算，LSB 后运算

第 7 章.GPCNT

7.1. 概述

GPCNT 为时钟脉冲计数器，用于计算两个时钟周期的比例，即用一个已知时钟（**主时钟**）计算另一个时钟（**次时钟**）的周期。

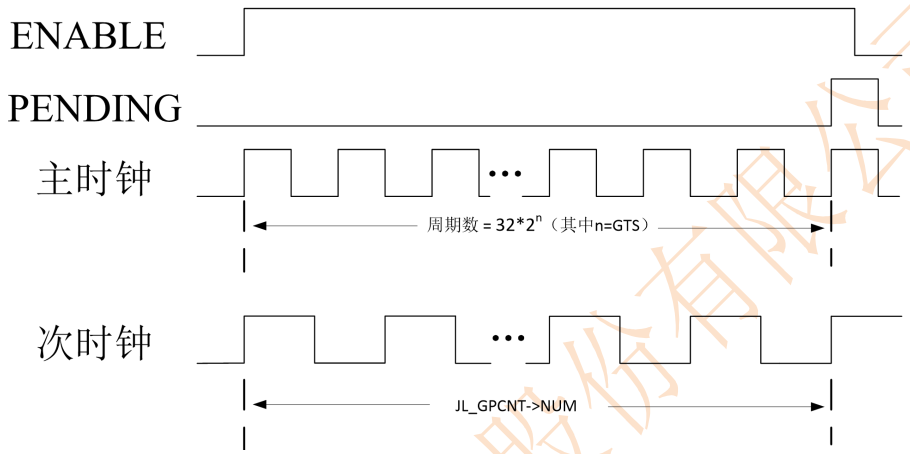


图 7-1 GPCNT 时钟计算示意图

7.2. 寄存器说明

1. JL_GPCNT->CON: GPCNT configuration register

Bit	Name	RW	Default	Description
14-12	GSS	rw	0	主时钟选择： 000: lsb_clk 001: osc_clk 010: input channel2（见 IOMAP_CON1[11:8]） 011: input channel3（见 IOMAP_CON1[15:12]） 100: 时钟系统输入（见 CLK_CON2[31:29]） 101: ring_osc 110: pll_d1p0 111: input channel1（见 IOMAP_CON1[7:4]）

11-8	GTS	rw	0	主时钟周期数选择： 主时钟周期数 = $32*2^n$ （其中 n=GTS）
7	PND	r	0	中断请求标志（当 JL_GPCNT->CON[0]置 1 后，主时钟达到 JL_GPCNT->CON[11:8]设定的周期数后，PENDING 置 1，并请求中断）： 0: 无 PENDING； 1: 有 PENDING
6	CLR_PND	w	0	清除中断请求标志位： 0: 无效； 1: 清 PENDING
5-4	reserved	rw	0	
3-1	CSS	rw	0	次时钟选择： 000: lsb_clk 001: osc_clk 010: input channel2（见 IOMAP_CON1[11:8]） 011: input channel3（见 IOMAP_CON1[15:12]） 100: 时钟系统输入（见 CLK_CON2[31:29]） 101: ring_osc 110: pll_d1p0 111: input channel1（见 IOMAP_CON1[7:4]）
0	ENABLE	rw	0	GPCNT 模块使能位： 0: 不使能； 1: 使能

【注意】：需配置好其他位，才将 ENABLE 置 1。

2.JL_GPCNT->NUM: The number of GPCNT clock cycle register

Bit	Name	RW	Default	Description
31-0	NUM	r	0	在 JL_GPCNT->CON[0]置 1 到 PENDING 置 1（中断到来）之间，次时钟跑的周期数。

第 8 章.IIC

8.1. 概述

IIC 接口是一个标准的遵守 IIC 协议的串行通讯接口。在上面传输的数据以 Byte（8bit）为最小单位，且永远是 MSB 在前。

IIC 接口支持主机和从机两种模式

主机：1）IIC 接口时钟由本机产生，提供给片外 IIC 设备使用；

2) IIC 接口的驱动时钟可配置，频率范围为 F = Fsystem/(IIC_BAUD*2) + 电阻上拉的时间。上拉电阻越大，频率越低。

从机：1）IIC 接口时钟由片外 IIC 设备产生，提供给本机使用；

2）IIC 总线上每一个 start/restart 位后接从机地址，此时当外部 IIC 设备所发的地址与我们匹配时，硬件会自动回应（ack 位上为“0”）；

IIC 接口使用下降沿更新数据，上升沿采集数据。

IIC 接口的发送寄存器和接收寄存器在物理上是分开的，但在逻辑上它们一起称为 IIC_BUF 寄存器，使用相同的 SFR 地址。当写这个 SFR 地址时，写入至发送寄存器。当读这个 SFR 地址时，从接收寄存器读出。

当作为 Master 的传送数据，可以在发送完 1byte 的数据之后，待 pnding 起来后，通过往 buf 中写数据再次传输数据，也可以配置 CON0[5]来结束本次传输。

如果是作为 Master 接收数据，那么传送完地址后，接收到第一笔数据，就自动结束本次传输。

寄存器基地址为（0x24）

8.2. 寄存器说明

1.IIC_CON0: IIC control register0

Bit	Name	RW	Default	Description
31	iic_pnding	r	0	PND：中断请求标志，当完成 1Byte 加 1 应答位的传输后会被硬件置“1”；清除此标志用软件方式：写 bit[7]。
30	iic_pnding_ab	r	0	当作为主机时，发送的地址，没有从机应答的 pnding，

				或者，发送的数据没有应答的 pnding，写 bit[7]清除该 pnding
29	iic_stop_pnding	r	0	当 IIC 停止时，pnding 标志位（不产生中断）
28	slv_is_t	r	0	master 要求从机是发送的标志位 1：从机是发送 0：从机是接收
27	slv_t_done	r	0	当作为 slave，是发送时，数据结束的标志，会在 master 下次发送 start 时，自动清 0 1：数据传输结束（master 回应了一个 NACK） 0：数据还要继续传输（master 回应了一个 ACK）
26-19	reserved	rw	0	reserved
18	mst_no_ack_allow	rw	0	作为主机时，no ack 允许 0：不允许接收 no ack 1：允许接收 no ack
17	mst_dma_en	rw	0	作为主机时的 dma 模式使能，仅作用于主机发送
16	mst_rd_start	w	0	作为主机，接收时，触发下一笔读操作
15	slv_dma_en	rw	0	当作为 slave，且接收时，dma 的使能 0：slave dma 关闭 1：slave dma 打开（使能打开后，在写满 buf 后，才起 pnding，而且在写满 buf 后，如果主机继续发数据，SLV 会根据 BIT[13]来决定是否拉住时钟。在 DMA 期间，master 不能 restart，不然 slv 会把 restart 后的地址当做数据处理）
14	iic_auto_stop_en	rw	0	IIC 硬件自动 stop 使能 1：硬件检测到 NACK，自动进入 stop 0：硬件检测到 NACK，会拉低 SCL，等待（是否进入 restart，或者继续发送/读取数据，或者进入 stop 的状态）

13	iic_restart	w	0	iic 重新开始位。（在 iic_pnding 产生之后，配置该 iic_restart，然后再配置 iic_buf，写入新的控制 byte 后，硬件会自动 restart 并发送新的控制 byte，发送完成后，产生 iic_pnding）
12	slv_clk_en	rw	0	作为 SLV 时，在传输完一个 btye 的数据（不包含地址）之后，是否要拉住时钟 0：不拉 1：拉住时钟（将 SCL 拉为 0）
11	iic_conf_ack	rw	0	iic 可配置 ack（最好一直设为 0，由硬件自动回复） 0：硬件自动回复 ack/no_ack 1：硬件强制回复 no_ack
10	iic_end_rd	w	0	当为主机接收，且为连续 byte 接收模式时，写”1”表示完成下一个 byte 数据交易后，stop 数据接收，将 SCL 拉低（须在开启最后一笔交易之后，pnding 起来之前配置）
9	iic_rd_sel	rw	0	当为主机接收时 0：单 byte 接收模式 1：连续 byte 接收模式
8	iic_int_en	rw	0	IIC 中断使能。 0: 禁止 iic 中断 1: 允许中断允许
7	iic_pnding_clr	w	0	IIC 的 pnding 清除
6	iic_slv_continue	w	0	当作为从机时，完成 1byte 的数据传送时（不包括地址的校验），会将 SCL 拉低（配合 bit[12]使用），该 bit 写“1”会将 SCL 释放，继续交易数据。
5	iic_end	w	0	IIC 为主机时，且选择发送时，写“1”停止发送（须在 pnding 起来之后，清 pnding 之前配置）； IIC 为主机时，且选择接收时，写“1”停止接收（在 pnding

				起来之后配置）；
4	iic_dat_dir	rw	0	当前的发送和接收选择 0：发送 1：接收
3	iic_isel	rw	0	iic_cki 和 iic_di 输入选择： 0：选择 IO 直接输入 1：选择 IO 经过 filter 后再输入
2	iic_slave	rw	0	IIC 接口模式选择 0：主机模式 1：从机模式
1	iic_start	w	0	IIC 开始准备工作
0	iic_en	rw	0	IIC 接口使能。 0: 关闭 IIC 接口 1: 打开 IIC 接口

2.IIC_BUF： IIC control register1

Bit	Name	RW	Default	Description
7-0	IIC_BUF	rw	0	

发送寄存器和接收寄存器共用此 SFR 地址。写入至发送寄存器，从接收寄存器读出。

3.IIC_BAUD：IIC control register2

Bit	Name	RW	Default	Description
15-0	IIC_BAUD	rw	0	

1) IIC 接口为主机时，IIC_BAUD 做为时钟设置寄存器。
2) 频率范围为 F = Fsystem/(IIC_BAUD*2) + 电阻上拉的时间。上拉电阻越大，频率越低。
3) IIC 接口为从机时，IIC_BAUD 做为从机地址(7bits)设置寄存器。

从机地址 = IIC_BAUD[7：1]

当地址匹配时硬件自动应答（ACK 位为“0”）

4.IIC_ADR：IIC ADR

Bit	Name	RW	Default	Description
31-0	IIC_ADR	rw	0	IIC DMA 模式时，dma 的起始地址

5.IIC_CNT：IIC CNT

Bit	Name	RW	Default	Description
31-0	IIC_CNT	rw	0	IIC DMA 模式时，已经写入/读出了多少个 8bit 的数据

6.IIC 时序图

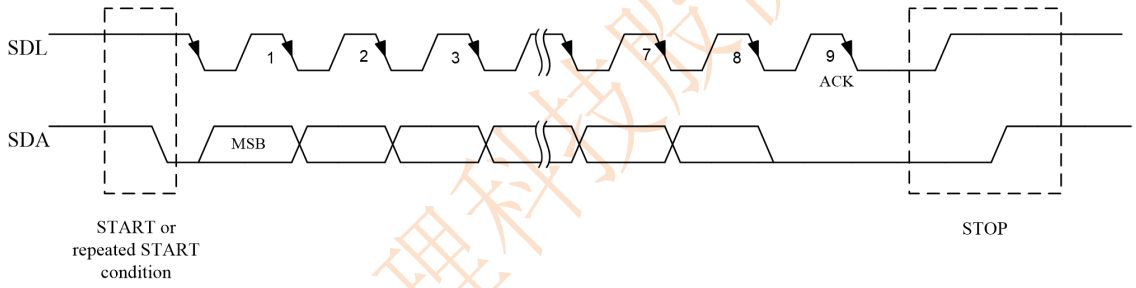


图 8-1 IIC 时序图

程序示例如下：

作为 Master 和 transmitter：

```
SFR(JL_IIC->CON0,  2,  1,  0); //iic_mst
SFR(JL_IIC->CON0,  3,  1,  1); //iic_isel
SFR(JL_IIC->CON0,  4,  1,  0); //0:transmitter 1:receiver
SFR(JL_IIC->CON0,  8,  1,  1); //iic_int_en
SFR(JL_IIC->CON0,  9,  1,  0); //0:1byte_rd 1:continuous_rd
```

```
SFR(JL_IIC->BAUD,    0,   16, 20); //baud
SFR(JL_IIC->BUF,     0,   8,  4); //initial_adr and r/w
SFR(JL_IIC->CON0,    0,   1,  1); //iic_en

SFR(JL_IIC->CON0,    1,   1,  1); //iic_start

for(int i=0;i<10;i++) {
while(!((JL_IIC->CON0)>>31)); //wait pnding
SFR(JL_IIC->CON0,    7,   1,  1); //iic_pnding_clr
asm("csync");
SFR(JL_IIC->BUF,     0,   8, i+5); //initial_adr and r/w
}

while(!((JL_IIC->CON0)>>31)); //wait pnding
SFR(JL_IIC->CON0,    7,   1,  1); //iic_pnding_clr
SFR(JL_IIC->CON0,    5,   1,  1); //iic_wr_end
delay(500);
```

作为 slv 和 receiver：

```
SFR(JL_IIC->CON0,    2,   1,  1); //0:mst 1:slave
SFR(JL_IIC ->CON0,    3,   1,  1); //iic_isel
SFR(JL_IIC->CON0,    4,   1,  1); //0:transmitter 1:receiver
SFR(JL_IIC->CON0,    8,   1,  1); //iic_int_en

SFR(JL_IIC->BAUD,    0,   16, 1); //baud
SFR(JL_IIC->CON0,    0,   1,  1); //iic_en
SFR(JL_IIC->CON0,    12,  1,  0); //0:slv no stretch clk 1:slv stretch clk
```

```
SFR(JL_IIC->CON0,    1,   1,   1); //iic_start
```

作为 Mst 和 receiver，而且连续读：

```
SFR(JL_IIC->CON0,    2,   1,   0); //iic_mst
SFR(JL_IIC->CON0,    3,   1,   1); //iic_isel
SFR(JL_IIC->CON0,    4,   1,   1); //0:transmitter 1:receiver
SFR(JL_IIC->CON0,    8,   1,   1); //iic_int_en
SFR(JL_IIC->CON0,    9,   1,   1); //0:1byte_rd 1:continuous_rd

SFR(JL_IIC->BAUD,    0,   16,  20); //baud
SFR(JL_IIC->BUF,     0,   8,   4); //initial_adr and r/w
SFR(JL_IIC->CON0,    0,   1,   1); //iic_en

SFR(JL_IIC->CON0,    1,   1,   1); //iic_start

for(int i=0;i<10;i++) {
while(!((JL_IIC->CON0)>>31)); //wait pnding
SFR(JL_IIC->CON0,    7,   1,   1); //iic_pnding_clr
asm("csync");
int tmp = JL_IIC->BUF;
}

while(!((JL_IIC->CON0)>>31)); //wait pnding
SFR(JL_IIC->CON0,    10,   1,   1); //iic_end_rd
SFR(JL_IIC->CON0,    7,   1,   1); //iic_pnding_clr
int tmp = JL_IIC->BUF;  //lst rd

delay(500);
```

作为 Mst 和 receiver，读 1byte：

```
SFR(JL_IIC->CON0,   2,  1,  0); //iic_mst
SFR(JL_IIC->CON0,   3,  1,  1); //iic_isel
SFR(JL_IIC->CON0,   4,  1,  1); //0:transmitter 1:receiver
SFR(JL_IIC->CON0,   8,  1,  1); //iic_int_en
SFR(JL_IIC->CON0,   9,  1,  0); //0:1byte_rd 1:continuous_rd

SFR(JL_IIC->BAUD,   0,  16, 20); //baud
SFR(JL_IIC->BUF,    0,  8,  4); //initial_adr and r/w
SFR(JL_IIC->CON0,   0,  1,  1); //iic_en

SFR(JL_IIC->CON0,   1,  1,  1); //iic_start

while(!((JL_IIC->CON0)>>31)); //wait pnding
int tmp = JL_IIC->BUF;  //lst rd
SFR(JL_IIC->CON0,   7,  1,  1); //iic_pnding_clr

delay(500);
```

作为 slave 和 transmitter：

```
SFR(JL_IIC->CON0,   2,  1,  1); //iic_mst
SFR(JL_IIC->CON0,   3,  1,  1); //iic_isel
SFR(JL_IIC->CON0,   4,  1,  0); //0:transmitter 1:receiver
SFR(JL_IIC->CON0,   8,  1,  1); //iic_int_en

SFR(JL_IIC->BAUD,   0,  16, 2); //baud
SFR(JL_IIC->CON1,   0,  8,  85); //wr_buf
```

```
SFR(JL_IIC->CON0,    0,   1,   1); //iic_en
SFR(JL_IIC->CON0,    12,   1,  0); //0:slv no stretch clk 1:slv stretch clk

SFR(JL_IIC->CON0,    1,   1,   1); //iic_start
```

第 9 章.红外过滤（IRFLT）

9.1. 概述

IRFLT 是一个专用的硬件模块，用于去除掉红外接收头信号上的窄脉冲信号，提升红外接收解码的质量。IRFLT 使用一个固定的时基对红外信号进行采样，必须连续 4 次采样均为‘1’时，输出信号才会变为‘1’；必须连续 4 次采样均为‘0’时，输出信号才会变为‘0’。换言之，脉宽小于 3 倍时基的窄脉冲将被滤除。改变该时基的产生可兼容不同的系统工作状态，也可在一定范围内调整对红外信号的过滤效果。通过对 IOMC（IO re-mapping）寄存器的配置，可以将 IRFLT 插入到系统 6 个 timer 中某一个的捕获引脚之前。

例如通过 IOMC 寄存器选择了 IRFLT 对 timer1 有效，并且 IRFLT_EN 被使能之后，则 IO 口的信号会先经过 IRFLT 进行滤波，然后再送至 timer1 中进行边沿捕获。

9.1.1. 硬件接口

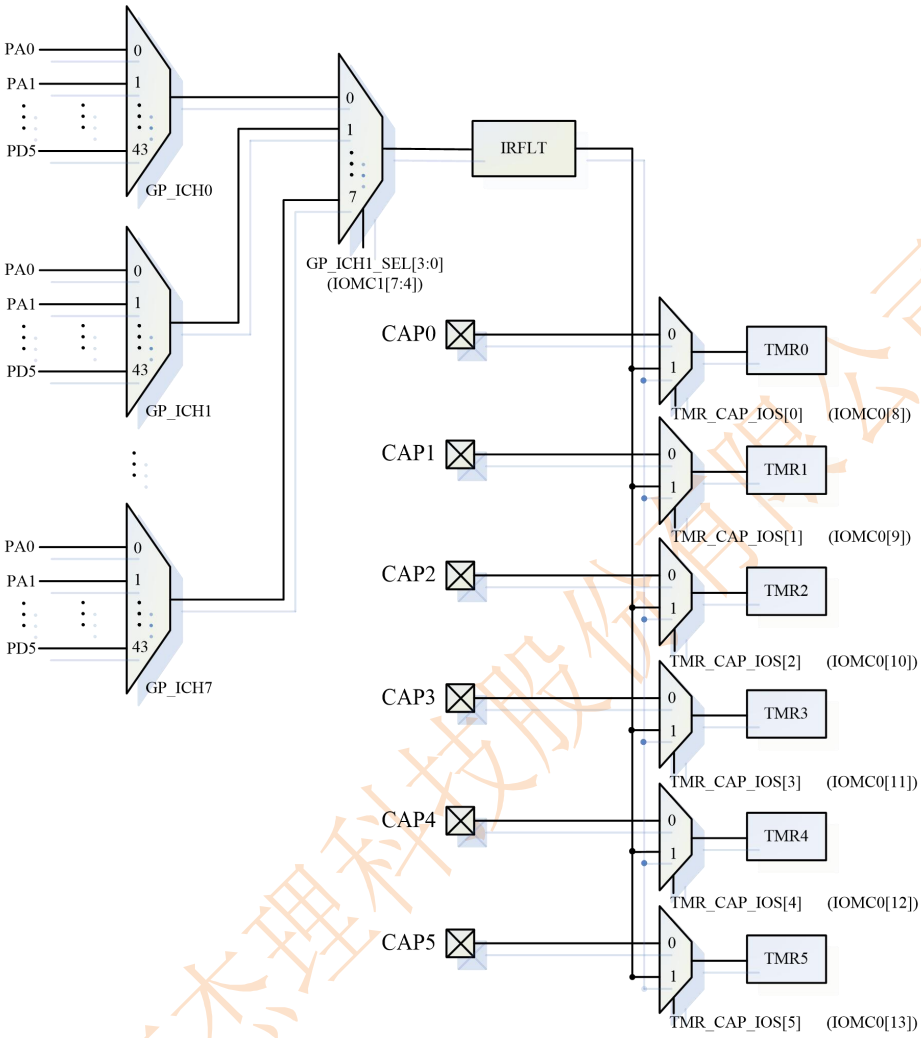


图 9-1 硬件接口示意图

9.1.2. 时基选择

PSEL 选定的分频倍数 N 和 TSRC 选定的驱动时钟的周期 Tc 共同决定了 IRFLT 用于采样红外接收信号的时基 Ts：

$$Ts = Tc * N$$

例如，当选择 32KHz 的 OSC 时钟，并且分频倍数为 1 时，Ts = 30.5uS。根据 IRFLT 的工作规则，所有小于(30.5*3=91.5uS)的窄脉冲信号，均会被滤除。

又如，当选择 48MHz 的系统时钟，并且分频倍数为 1024 时，Ts = 21.3uS。根据 IRFLT 的工作

规则，所有小于(21.3*3=63.9uS)的窄脉冲信号，均会被滤除。

9.2. 寄存器说明

1. IRFLT_CON: irda filter contrl register (8bit addressing)

Bit	Name	RW	Default	Description
7-4	PSEL[3:0]	rw	0	PSEL[3-0]：时基发生器分频选择 0000：分频倍数为 1； 0001：分频倍数为 2； 0010：分频倍数为 4； 0011：分频倍数为 8； 0100：分频倍数为 16； 0101：分频倍数为 32； 0110：分频倍数为 64； 0111：分频倍数为 128； 1000：分频倍数为 256； 1001：分频倍数为 512； 1010：分频倍数为 1024； 1011：分频倍数为 2048； 1100：分频倍数为 4096； 1101：分频倍数为 8192； 1110：分频倍数为 16384； 1111：分频倍数为 32768；
3-2	TSRC[1-0]	rw	0	TSRC[1-0]：时基发生器驱动源选择 00：选择 LSB_CLK 来驱动时基发生器； 01：选择 RC 时钟来驱动时基发生器； 10：选择 OSC_CLK 时钟来驱动时基发生器； 11：选择 PLL_48M 时钟来驱动时基发生器；
1	reserved	r	0	预留

0	IRFLT_EN	rw	0	IRFLT_EN： IRFLT 使能 0：关闭 IRFLT； 1：打开 IRFLT；

9.3. 例程

```
#define IR_PA8          0//PA8
void IR_init(void)
{
    ///选择 IO
    PORTA_DIR |= BIT(8);

    IOMC0 &= ~(0xF<<8);
    IOMC0 |= (IR_CTR<<8);//选择红外输入脚
    IOMC0 |= (2<<4);        //选择捕获 timer

    if(get_apb_clk() >= 12000000L)
    {
        IRFLT_CON = 0xa1;    //1024 分频倍数，使能 IR
        TMR2_CON  = 0x33;    //预分频 64，IO 口下降沿捕获
    }
    else
    {
        IRFLT_CON = 0x41;    //4096 分频倍数，使能 IR
        TMR2_CON  = 0x23;    //预分频 64，IO 口下降沿捕获
    }

    timer2_pad = (get_apb_clk()/1000)/64;
```

```
    printf("timer 2 init\n");
    int_enter_pr[TIMER2_INT-1] = timer2_isr;
}
```

第 10 章.LEDC

10.1.概述

LEDC 应用于 LED 灯带驱动和控制。通过配置，可以使 led 灯带显示不同颜色变化。支持 dma 数据传输，支持两路独立显示驱动。

10.2.控制寄存器

LEDX 为 0/1 通道相同寄存器

1. LED_CLK

Bit	Name	RW	Default	Description
15-8	LEDC_BAUD	rw	0	CNT_CK = baud clock / (LEDC_BAUD + 1)
7-2				
1-0	CLK_SEL	rw	0	0-内部状态机复位，无时钟 1-lsb_clk 2-std_24m 3-std_48m

2.LEDx_CON

Bit	Name	RW	Default	Description
7	PAND	r	0	完成标志，1 表示完成一次循环
6	CLR	w	0	置 1 清除完成标志
5	IE	rw	0	中断允许
4	IDLE_LEVEL	rw	0	空闲时输出电平
3	OUT_INV	rw	0	0-默认信号正向输出，1-输出信号反向
2-1	BYTE_INV	rw	0	0-不取反 1-1byte 取反（0xaa<->0x55） 2-2byte 取反（0xaabb<->0xdd55）

				3-4byte 取反（0xaabbccdd<->0xbb33dd55）
0	EN	rw	0	置 1 启动

3.LEDx_FD

Bit	Name	RW	Default	Description
31-0	FD_LEN	rw	0	不重复 n 个 LED 数据（LED_BIT）循环，设置值为（不重复个数*单个 led 位数），例如 10 个不重复 RGB led 灯-10*24=240。

4.LEDx_LP

Bit	Name	RW	Default	Description
15-0	LP_LEN	rw	0	FD_LEN 重复次数，0 则不重复

5.LEDx_TIX

Bit	Name	RW	Default	Description
31-24	T1H	rw	0	输出 1 码高电平时间，单位 CNT_CK。实际值减 1
23-16	T1L	rw	0	输出 1 码低电平时间，单位 CNT_CK，实际值减 1
15-8	T0H	rw	0	输出 0 码高电平时间。实际值减 1
7-0	T0L	rw	0	输出 0 码低电平时间。实际值减 1

6.LEDx_RSTX

Bit	Name	RW	Default	Description
31-8	RSTL	rw	0	每帧一开始复位有效时间长度，单位 CNT_CK。
7-0	RSTH	rw	0	每帧最后一 bit 到起 pnding 的时间间隔，单位 CNT_CK。0 为不输出

7.LEDx_ADR

Bit	Name	RW	Description
31-0	ADR	rw	数据输入基地址,必须为 32 位对齐地址值，读为 DMA 当前地址值

10.3.控制寄存器

10.3.1. LEDC 流程结构

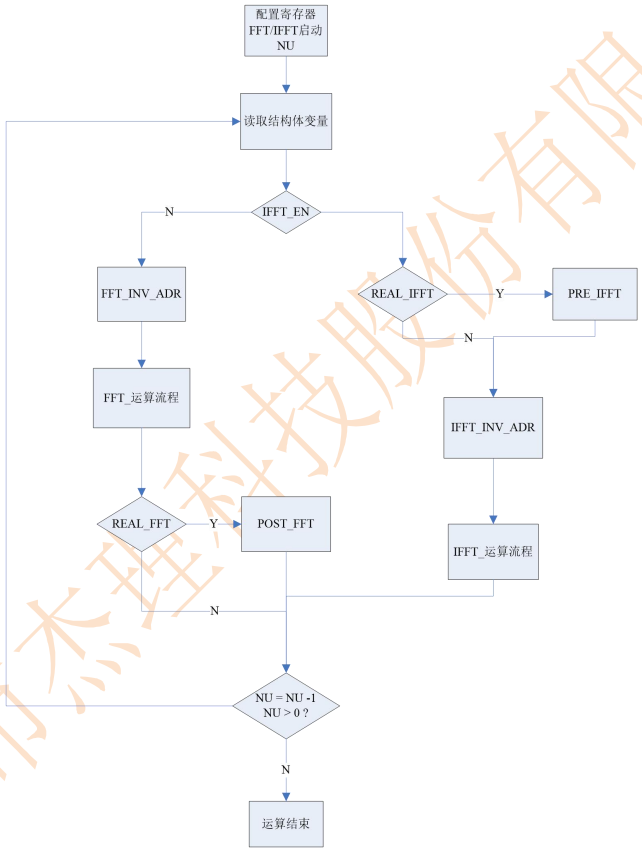


图 10-1 LEDC 流程结构

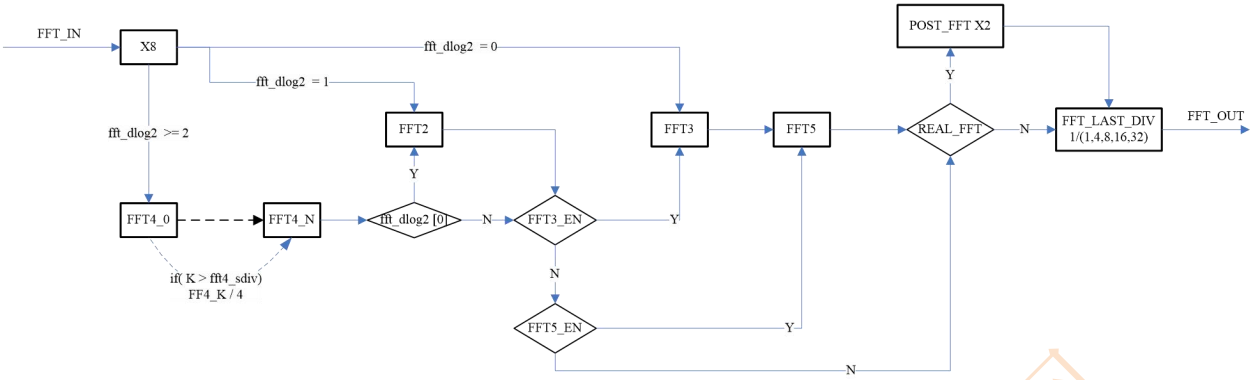


图 10-2 LEDC 图 1

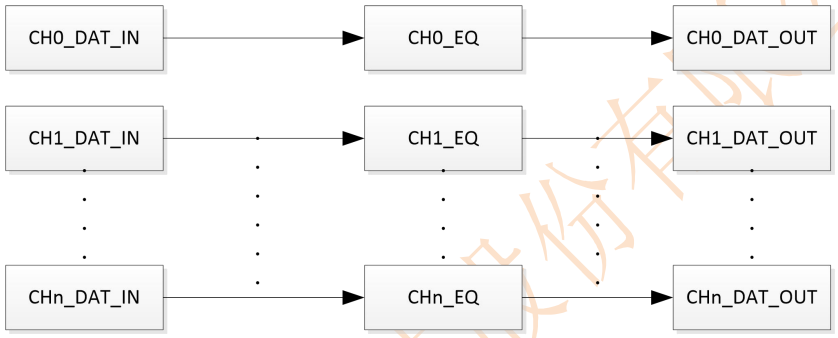


图 10-3 LEDC 图 2

10.3.2. 输入输出数据结构

输入数据结构按 byte 读取	
Byte 地址值	数据
0	Data_0
1	Data_1
2	Data_2
…	
…	
N-1	Data_N-1

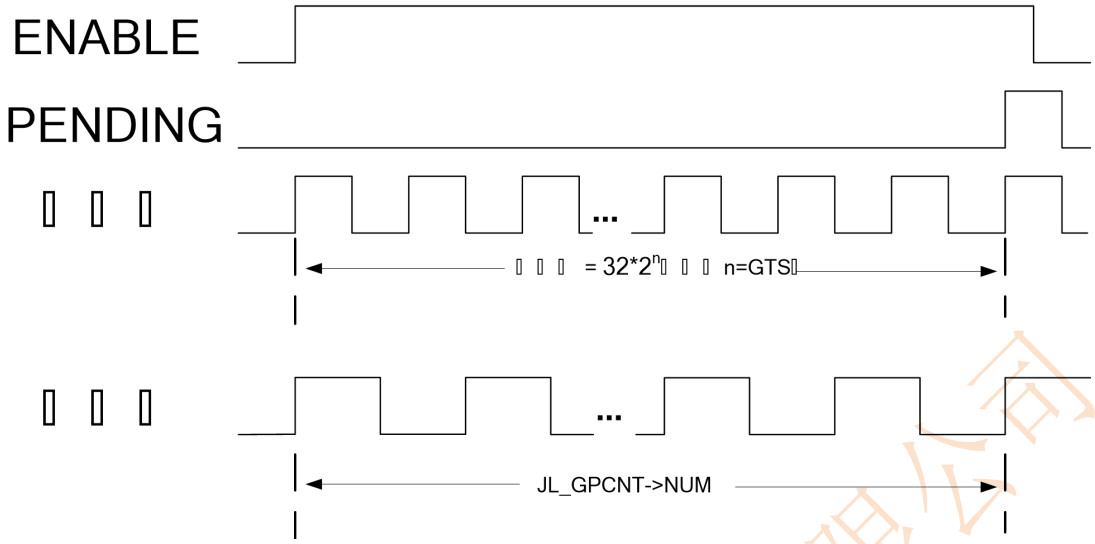


图 10-4 输入输出数据结构图

10.4.验证点

1.收发正确性，从 DMA 口读取到 IO 数据输出，数据是否一致

2.DMA 繁忙时，数据接收是否正确。

3.多通道请求 DMA 数据时，DMA 请求是否正确，数据是否给对通道。

4.波特率分频是否正确，时延参数计数是否正确。

5.CNT 输出 bit 数是否正确。

6.字段计数是否正确，中断是否正确起来。

10.5.验证方法

验证环境针对收发正确性做专门检测，支持多通道数据校验，再搭建 LED 驱动芯片简易模型，支持 1 线 2 线输入，支持 WRGB、RGB 格式，支持 0、1 电平长度可调，内建 monitor 用于监控 LEDC 输出信号是否满足驱动要求以及是否按照 SFR 配置要求。只针对上述几点验证点做验证，model 需结合实际应用，以后再考虑。

10.6.使用例程

10.6.1. LEDC 使用示例

1．第一次使用或者模块通道都处于空闲状态时，先设置 clk_sel，选择 1-3 时钟，同时使内部状态机复位信号释放回复正常。

【注意】：中途给任意通道配置时不要配置 clk_sel。

2．使用对应通道，先对该通道进行 pnd_clr，清除 pnd 标志。

3．配置 CON，中断使能,波特率 baud，空闲电平 idle_level，输出反向使能（默认为 0，即归 0 码）。

4．配置 FD，设置一帧的 led bit 数，比如 256 个 24bit 的 LED 灯，则设置为 256*24=6144，注：配置 0 时为避免出错，内部逻辑修改为 1。

5．配置 LP，设置循环次数，设置为 0 时即不循环。比如 256 个 24bit 的 led 灯，两两重复，即 FD=2*24=48，LP=256/2=128；

6．配置 TIX，设置 1 和 0 的高低电平时间，两通道共用参数。

【注意】：在归 0 码模式，T1H 即 T1H，在归 1 码时，T1H 即 T0H，T1L 即 T0L。

7．配置 RSTX，设置一帧结尾的复位时间，RSTH 为输出的最后 1bit 到复位信号的时间，RSTL 为输出复位信号时间，归 0 码和归 1 码表述意思相同。

8．配置 ADR,设置两通道的 dma 数据地址。

9．配置 CON 使能位。

10．当 RSTH 计数时间到会起 pnd。

第 11 章.MCPWM

11.1. 概述

MCPWM 功能模块包括：8 个 MCTimer0 时基模块，8 对独立的 PWM 通道，4 路故障保护输入。框图如下：

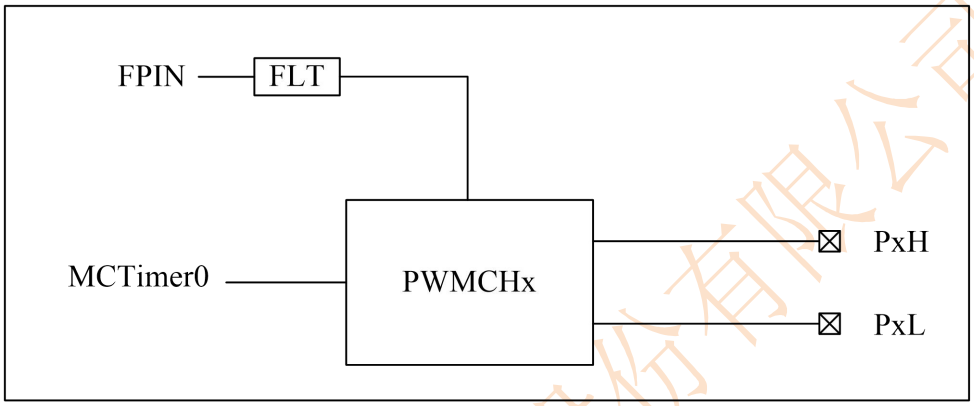


图 11-1 MCPWM 功能模块图

11.2. 定时器 MCTIMER 0/1/2/3/4/5/6/7

11.2.1. 概述

MCTimer0/1/2/3/4/5/6/7 是 16 位定时器，可作为 PWM0/1/2/3/4/5/6/7 的时基控制

11.2.2. 模块特性

1.16 位定时功能

2.带缓冲的周期寄存器

3.支持多种工作模式：递增、递增-递减、外部引脚控制递增或递减

4.多种中断模式：上溢出中断、下溢出中断、上下溢出中断

5.可配置成启动 ADC

11.2.3. 模块寄存器

1. MCTMR0_CON：MCPWM Timer contrl register (16bit addressing)计数/定时控制寄存器

Bit	Name	RW	Default	Description
31-16	reserved	r	0	保留
15	T0INCF	r	0	T0INCF：TIMR0CNT 递增递减标志位 0：递减 1：递增
13	T0UFPND	r	0	T0UFPND：TMR0CNT 递减借位标志 0：没借位 1：已发生借位
12	T0OFPN	r	0	T0OFPND：TMR0CNT 溢出（TMR0CNT == TMR0PR）标志 0：没溢出 1：已发生溢出
11	T0UFCLR	w	0	T0UFCLR: 清除 T0UFPND 标志位 0：无效 1：清除 T0UFPND
10	T0OFCLR	w	0	T0OFCLR: 清除 T0OFPND 标志位 0：无效 1：清除 T0OFPND
9	T0UFIE	rw	0	T0UFIE：TMR0CNT 定时递减借位中断控制 0：禁止 1：允许
8	T0OFIE	rw	0	T0OFIE：TMR0CNT 定时溢出（TMR0CNT == TMR0PR）中断控制 0：禁止 1：允许

7	T0CKSRC	rw	0	T0CKSRC：定时器时钟源 0：内部时钟 1：外部引脚时钟 TMR0CK(pwm0-3 有，pwm4-7 固定 1'b0)
6-3	T0CKPS	rw	0	T0CKPS：时钟预分频设置，TCK/（2^TCKPS）
2	reserved	r	0	预留
1-0	T0MODE	rw	0	T0MODE：定时器工作模式 00：保持 TMR0CNT 01：递增模式 10：递增-递减循环模式 11：由外部引脚控制递增或递减

2.2.MCTMR0_CNT：

Bit	Name	RW	Default	Description
15-0	MCTMR0_CNT	rw	0	16 位定时/计数器

3.3.MCTMR0_PR：

Bit	Name	RW	Default	Description
15-0	MCTMR0_PR	rw	0	带缓冲的 16 位周期寄存器

11.3. PWM 0/1/2/3/4/5/6/7（内置 8 对 PWM 模块）

11.3.1. 模块引脚

PWM0：PWMCH0H、PWMCH0L

PWM1：PWMCH0H、PWMCH1L

PWM2：PWMCH0H、PWMCH2L

.....

故障输入引脚：FPIN

11.3.2. 模块特性

1.边沿对齐和中心对齐输出模式

2.可运行过程中更改 PWM 频率、占空比

3.多种更新频率/占空比模式：上溢出重载、下溢出重载、上下溢出重载

4.灵活配置每对通道的有效电平状态

5.可编程死区控制

6.硬件故障输入引脚

11.3.3. 模块控制寄存器

PWM0/1/2/3/4/5/6/7 功能相同，以下描述 PWM0 的寄存器，PWM1/2/3/4/5/6/7 的寄存器对应参照 PWM0

1. MCCH0_CON0：PWM 控制寄存器 0

Bit	Name	RW	Default	Description
15-12	DTCKPS	rw	0	DTCKPS：死区时钟预分频，Tsys/（2^ DTCKPS）
11-7	DTPR	rw	0	DTPR：死区时间控制 死区时间：Tsys/(2^ DTCKPS) × (DTPR+1)
6	DTEN	rw	0	DTEN：死区允许控制，对应 PWMCH0H、PWMCH0L 0：禁止 1：允许
5	L_INV	rw	0	L_INV：对应 PWMCH0L 输出反向控制 0：反向禁止 1：反向允许
4	H_INV	rw	0	H_INV：对应 PWMCH0H 输出反向控制

				0：反向禁止 1：反向允许
3	L_EN	rw	0	L_EN：对应 PWMCH0L 输出允许控制 0：禁止 PWMCH0L 1：允许 PWMCH0L
2	H_EN	rw	0	H_EN：对应 PWMCH0H 输出允许控制 0：禁止 PWMCH0H 1：允许 PWMCH0H
1-0	CMP_LD	rw	0	CMP_LD：MCCH0_CMP 重新载入控制 00：时基 MCTMR0_CNT 等于“0”载入 01：时基 MCTMR0_CNT 等于“0”或者等于 MCTMR0_OVF 时候载入 10：时基 MCTMR0_CNT 等于 MCTMR0_OVF 时候载入 11：立即载入

2. MCCH0_CON1：PWM 控制寄存器 1

Bit	Name	RW	Default	Description
15	FPND	r	0	FPND：故障保护输入标志，只读，写无效 读 0：未发生保护 读 1：已发生保护，模块的 PWM 引脚会变成高阻态
14	FCLR	w	0	FCLR: 清除 FPND 标志位，只写，读为“0” 写 0：无效 写 1：清除 FPND
13-12	reserved	r	0	保留
11	INTEN	rw	0	INTEN: FPND 中断允许

				0：禁止 1：允许
10-8	TMRSEL	rw	0	TMRSEL：选择 MCTMR0-7 作为 PWM 时基 0-7：选择时基
7-5	reserved	r	0	保留
4	FPINEN	rw	0	FPINEN：故障保护输入允许控制 0：禁止保护 1：允许保护
3	FPINAUTO	rw	0	FPINAUTO：故障自动保护控制 0：禁止自动保护 1：允许自动保护， 当检测到故障引脚有效信号时，自动把 PWM 引脚设成高阻态，直到软件清除 FPND。
2-0	FPINSEL	rw	0	FPINSEL：故障保护输入引脚选择 0-7：选择 FPIN 作为故障保护输入(0-3 为 io 输入，4-7 固定为 0）

3. MCCH0_CMPH：

Bit	Name	RW	Default	Description
15-0	MCTMR0_PRH	rw	0	带缓冲的 16 位比较寄存器，对应 PWMCH0H 引脚的占空比控制

4. MCCH0_CMPL：

Bit	Name	RW	Default	Description
15-0	MCTMR0_PRL	rw	0	带缓冲的 16 位比较寄存器，对应 PWMCH0L 引脚的占空比控制

5. MCFPIN_CON：FPIN0/1/2/3/4/5/6/7 滤波控制寄存器，复位值：0x0

Bit	Name	RW	Default	Description
23-16	EDGE	rw	0	EDGE：FPINx 边沿选择 0：下降沿 1：上升沿
15-8	FLT_EN	rw	0	FLT_EN：FPINx 滤波使能开关 0：滤波关闭 1：滤波开启
7-6	reserved	r	0	保留
5-0	FLT_PR	rw	0	FLT_PR：滤波宽度选择 滤波宽度=16 × FLT_PR × lsb_clk

6.MCPWM_CON：使能控制寄存器，复位值：0x0

Bit	Name	RW	Default	Description
15-8	TMR_EN	rw	0	T0EN：定时器计数开关控制(tmr7-0) 0：定时器计数关闭 1：定时器计数打开
7-0	PWM_EN	rw	0	PWMEN：模块开关控制（pwm7-0） 0：模块关闭 1：模块开启

【注意】：目前模块 pwm 输出，disable 后保持最后输出状态。

第 12 章.PULSE COUNTER

12.1.概述

以下为 pulse_counter/触摸键的寄存器说明及使用方法。

12.2.寄存器说明

1. PL_CNT_CON

Bit	Name	RW	Default	Description
7-4	-	-	0	预留
3-2	clk_sel	rw	0	pulse counter 时钟选择 00: 选择 osc_clk 作为计数时钟； 01: 选择 clk_mux_in 作为计数时钟（iomc1[15:12] intput channel3 选择）； 10: 选择 pll_d1p5 作为计数时钟； 11: 选择 pll_d1p0 作为计数时钟； 备注：时钟选择的频率越大，pl_cnt_value 计数值会越大，分辨率会越高，触摸键的灵敏度也越高
1	en	rw	0	触摸键使能，当 cap_mux_in（input channel 2）为 1 时，PLCNTVL 累加计数，计满后归 0 再重新累加 0: 触摸键禁止； 1: 触摸键使能
0	test_en	rw	0	触摸键测试使能（此位专用于测试） 0: 测试模式未使能； 1: 测试模式使能；

【注意】：当 cap_mux_in（input channel 2）选择 TMRx_PWMOUT, pulse counter 时钟选择 clk_mux_in（input channel 3）时，可以用内部固定周期的 PWM，计算芯片 IO 上不确定的时钟频率。

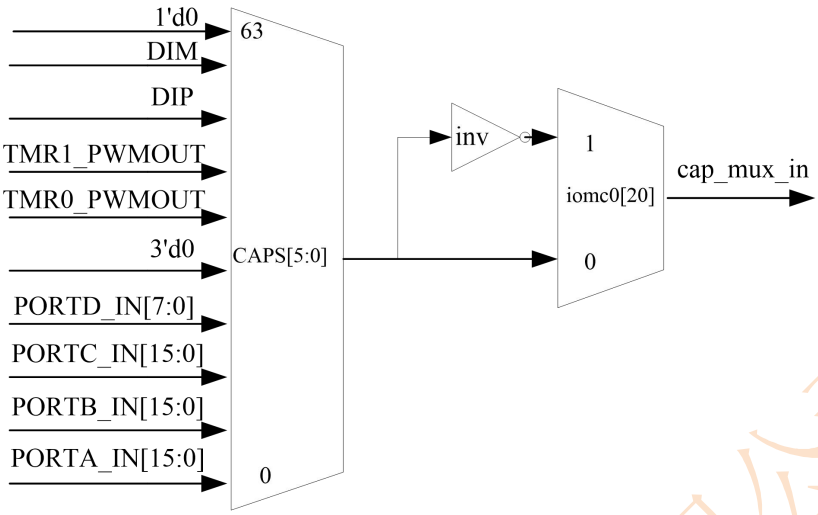


图 12- 1 cap_mux_in(input channel 2)电路控制图

12.3.示例代码

以下以 PA1 为例:

1. 变量定义：

int Touchkey_value_old，Touchkey_value_new，Touchkey_value_delta； //32bit 有符号整数

2.选择检测管脚

IOMC2 &= ~(0xff00)； //

IOMC2 |= 0x01 << 16； //选择 PA1

PORTA_OUT |= BIT(1)；

PORTA_DIR &= ~BIT(1)； //PA1 输出为 1

PORTA_PD |= BIT(1)；//PA1 下拉打开

3.初始化计数器配置

PLCNTCON &= ~(0xC)； //

PLCNTCON |= BIT(3)；//选择 pll_192m 作为触摸键时钟

PLCNTCON |= BIT(1)；//使能计数器

4. 检测电容

Touchkey_value_old = PLCNTVL；//读取旧值

PORTA_DIR |= BIT(1)；//对应管脚输入使能

While(PORTA_IN & BIT(1))；

Touchkey_value_new = PLCNTVL；

If(Touchkey_value_old > Touchkey_value_new)then

Touchkey_value_new += 0x10000；

Touchkey_value_delta = Touchkey_value_new - Touchkey_value_old；

第 13 章.RDEC

13.1.概述

RDEC（rotate decoder）是一个用于旋转编码器检测的模块，它支持两线输入的旋转编码器，可以检测旋转方向和旋转步数。

AC632N 中，一共支持 3 路 RDEC，每路 RDEC 均可独立工作且互不影响。其中，RDEC0~RDEC2 除了能当做普通旋转编码器之外，还兼容鼠标滑轮模式，而且鼠标滚轮模式下还包括定时与非定时两种方式。

RDEC0 的中断号为 25，RDEC1 的中断号为 49，RDEC2 的中断号为 50。

13.2.控制寄存器

寄存器列表	RDEC0	RDEC1	RDEC2
RDECx_CON	RDEC0_CON	RDEC1_CON	RDEC2_CON
RDECx_DAT	RDEC0_DAT	RDEC1_DAT	RDEC2_DAT
RDECx_SMP	RDEC0_SMP	RDEC1_SMP	RDEC2_SMP

13.3.寄存器说明

1. RDECx_CON： rdec control register

Bit	Name	RW	Default	Description
15~10	reserved	r	0	预留
9	RDEC_TIME	rw	0	RDEC_TIME（需要配合鼠标滑轮模式使用，即 bit8 为高时有效） 0：关闭定时器模式 1：开启定时器模式
8	RDEC_MODE	rw	0	RDEC_MODE 0：普通模式

				1：鼠标滑轮模式
7	PND	r	0	PND，只读。写入无效
6	CPND	w	0	CPND，只写。写入 1 清除 PND，读出永远为 0
5~2	RDEC_SPD	rw	0	RDEC_SPD，采样速率设置 Tsr=(2^RDEC_SPD)/Flsb Flsb 为低速外设总线的频率，软件应当设置 RDEC_SPD 使 Tsr 介于 0.5~2mS 之间
1	RDEC_POL	rw	0	RDEC_POL 0：输入引脚无信号时处于 1 状态（外部引脚上拉） 1：输入引脚无信号时处于 0 状态（外部引脚下拉）
0	EN	rw	0	RDEC_EN 0：模块关闭 1：模块打开

2.RDECx_DAT：rdec data register

Bit	Name	RW	Default	Description
7-0	RDEC_DAT	r	0	此寄存器为 8 位有符号数，表示旋转编码器正反向旋转的步数。 在普通模式下：当检测到旋转编码器动作时，PND 会设置为 1，软件可通过中断或查询方式来读取此寄存器。也可以完全不理会 PND，软件隔一段时间来访问此寄存器，但需注意检测时间不能过长，如果此寄存器的数值超过-128~127 的区间，将会产生溢出，无法表示出旋转编码器的正确动作。 在鼠标滑轮模式的非定时模式下，该寄存器的使用方式与普通模式相同，在清 PND 的动作时会更新此寄存器；在鼠标滑轮模式的定时模式下，该寄存器应配合 RDEC_SMP 寄存器使用，当采样个数满足 RDEC_SMP

				要求时，PND 会设置为 1，软件可通过中断或查询方式来读取此寄存器。也可以完全不理会 PND，软件隔一段时间来访问此寄存器，但需注意检测时间不能过长，如果此寄存器的数值超过-128~127 的区间，将会产生溢出，无法表示出旋转编码器的正确动作。而此时该寄存器里头的值表示在这段时间内鼠标滚轮的增减量。 【注意】：清 PND 的动作会更新此寄存器。

3.RDECx_SMP： rdec sample register

Bit	Name	RW	Default	Description
8-0	RDEC_SMP	rw	0	RDEC_SMP： 0000：640 0001：2559 0010：5118 0011：7676 0100：10235 0101：12793 0110：15352 0111：17910 1000：64 该寄存器通过设置采样个数，表示在 Tsmp=Tsr*RDEC_SMP 的时间内将旋转编码器的正反转步数记录在 RDEC_DAT 的寄存器中。当采样个数满足设置要求时，PND 将会被置 1。

第 14 章.SPI

14.1.概述

SPI 接口是一个标准的遵守 SPI 协议的串行通讯接口，在上面传输的数据以 Byte（8bit）为最小单位，且永远是 MSB 在前。

【注意】：该模块所有说明及配置仅作为 SPIx 模块使用，不能套用到 SFC 中

SPI 接口支持主机和从机两种模式

主机：SPI 接口时钟由本机产生，提供给片外 SPI 设备使用

从机：SPI 接口时钟由片外 SPI 设备产生，提供给本机使用

工作于主机模式时，SPI 接口的驱动时钟可配置，范围为 系统时钟~系统时钟/256

工作于从机模式时，SPI 接口的驱动时钟频率无特殊要求，但数据速率需要进行限制，否则易出现接收缓冲覆盖错误。

SPI 接口可独立地选择在 SPI 时钟的上升沿或下降沿更新数据，在 SPI 时钟的上升沿或下降沿采样数据。

SPI 接口支持单向（Unidirection）和双向（Bidirection）模式

单向模式：使用 SPICK 和 SPIDAT 两组连线，其中 SPIDAT 为双向信号线，同一时刻数据只能单方向传输。

双向模式：使用 SPICK，SPIDI 和 SPIDO 三组连线，同一时刻数据双向传输。但 DMA 不支持双向数据传输，当在本模式下使能 DMA 时，也只有一个方向的数据能通过 DMA 和系统进行传输。

SPI 单向模式支持 1bit data、2bit data 和 4bit data 模式，即：

1bit data 模式：串行数据通过一根 DAT 线传输，一个字节数据需 8 个 SPI 时钟。

2bit data 模式：串行数据通过两根 DAT 线传输，一个字节数据需 4 个 SPI 时钟。

4bit data 模式：串行数据通过四根 DAT 线传输，一个字节数据需 2 个 SPI 时钟。

【注意】：所有 SPI 支持 1bit，2bit 和 4bit 模式

SPI 双向模式只支持 1bit data 模式，即：

1bit data 模式：串行数据通过一根 DAT 线传输，一个字节数据需 8 个 SPI 时钟。

SPI 接口在发送方向上为单缓冲，在上一次传输未完成之前，不可开始下一次传输。在接收方向上为双缓冲，如果在下一次传输完成时 CPU 还未取走本次的接收数据，那么本次的接收数据将会丢失。

SPI 接口的发送寄存器和接收寄存器在物理上是分开的，但在逻辑上它们一起称为 SPIBUF 寄存器，使用相同的 SFR 地址。当写这个 SFR 地址时，写入至发送寄存器。当读这个 SFR 地址时，从接收寄存器读出。

SPI 传输支持由 CPU 直接驱动，写 SPIBUF 的动作将启动一次 Byte 传输。

SPI 传输也支持 DMA 操作，但 DMA 操作永远是单方向的，即一次 DMA 要么是发送一包数据，要么是接收一包数据，不能同时发送并且接收一包数据，即使在双向模式下也是这样。每次 DMA 操作支持的数据量为 1-65535Byte。写 SPIADR 的动作将启动一次 DMA 传输。

14.2.控制寄存器

寄存器列表	SPI0	SPI1	SPI2
SPIx_CON	SPI0_CON	SPI1_CON	SPI2_CON
SPIx_BUF	SPI0_BUF	SPI1_BUF	SPI2_BUF
SPIx_BAUD	SPI0_BAUD	SPI1_BAUD	SPI2_BAUD
SPIx_ADR	SPI0_ADR	SPI1_ADR	SPI2_ADR
SPIx_CNT	SPI0_CNT	SPI1_CNT	SPI2_CNT

1. SPIx_CON: SPIx control register (16bit addressing)

Bit	Name	RW	Default	Description
15	PND	r	0	中断请求标志，当 1Byte 传输完成或 DMA 传输完成时

				会被硬件置 1。 有 3 种方法清除此标志 向 PCLR 写入‘1’ 写 SPIBUF 寄存器来启动一次传输 写 SPICNT 寄存器来启动一次 DMA
14	PCLR	w	0	软件在此位写入‘1’将清除 PND 中断请求标志。
13	IE	rw	0	SPI 中断使能 0: 禁止 SPI 中断 1: 允许中断允许
12	DIR	rw	0	在单向模式或 DMA 操作时设置传输的方向 0: 发送数据 1: 接收数据
11-10	DATW	r	0	SPI 数据宽度设置 00: 1bit 数据宽度 01: 2bit 数据宽度 10: 4bit 数据宽度 11：NA，不可设置为此项 （注：SPI0 支持 1bit，2bit 和 4bit 模式，SPI1/SPI2 只支持 1bit 模式）(添加了 2bit,4bit 模式)
9	reserved	r	0	0
8	reserved	r	0	0
7	CSID	rw	0	SPICS 信号极性选择 0: SPICS 空闲时为 0 电平 1: SPICS 空闲时为 1 电平
6	CKID	rw	0	SPICK 信号极性选择 0: SPICK 空闲时为 0 电平

				1: SPICK 空闲时为 1 电平
5	UE	rw	0	更新数据边沿选择 0: 在 SPICK 的上升沿更新数据 1: 在 SPICK 的下降沿更新数据
4	SE	rw	0	采样数据边沿选择 0: 在 SPICK 的上升沿采样数据 1: 在 SPICK 的下降沿采样数据
3	BIDIR	rw	0	单向/双向模式选择 0: 单向模式，数据单向传输，同一时刻只能发送或者接收数据。 数据传输方向因收发而改变，所以由硬件控制，不受写 IO 口 DIR 影响。 1: 双向模式，数据双向传输，同时收发数据，但 DMA 只支持一个方向的数据传输。 数据传输方向设置后不改变，所以由软件控制，通过写 IO 口 DIR 控制。
2	CSE	rw	0	SPICS 信号使能 0: 不使用 SPICS 信号 1: 使用 SPICS 信号
1	SLAVE	rw	0	从机模式
0	SPIE	rw	0	SPI 接口使能 0: 关闭 SPI 接口 1: 打开 SPI 接口

2.SPIx_BAUD: SPI baudrate setting register (8bit addressing, write only).

Bit	Name	RW	Default	Description
7-0	SPI_BAUD	rw	0	

SPI 主机时钟设置寄存器

SPICK = system clock / (SPIBAUD + 1)

3.SPI_BUF: SPI buffer register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	SPI_BUF	rw	0	

发送寄存器和接收寄存器共用此 SFR 地址。写入至发送寄存器，从接收寄存器读出。

4.SPI_ADR: SPI DMA start address register (25bit addressing, write only).

Bit	Name	RW	Default	Description
24-0	SPI_ADR	rw	0	

SPI DMA 起始地址寄存器，只写，读出为不确定值

5.SPI_CNT: SPI DMA counter register (16bit addressing, write only)

Bit	Name	RW	Default	Description
15-0	SPI_CNT	rw	0	

SPI DMA 计数寄存器，只写，读出为不确定值

此寄存器用于设置 DMA 操作的数目（按 Byte 计）并启动 DMA 传输。

如，需启动一次 512Byte 的 DMA 传输，写入 0x0200，此写入动作将启动本次传输。

14.3.传输波形

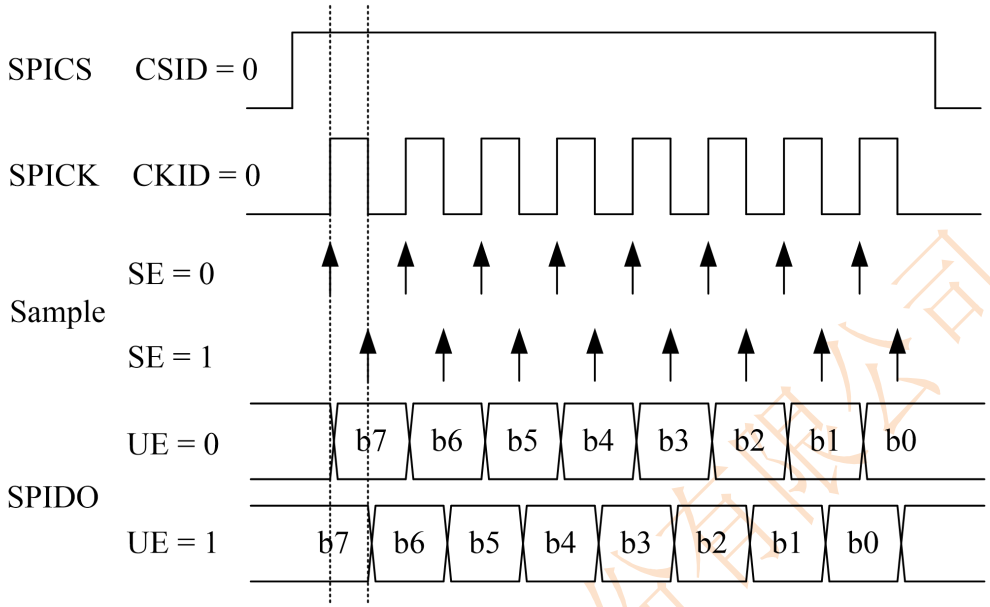


图 14-1 SPI 时序图 1

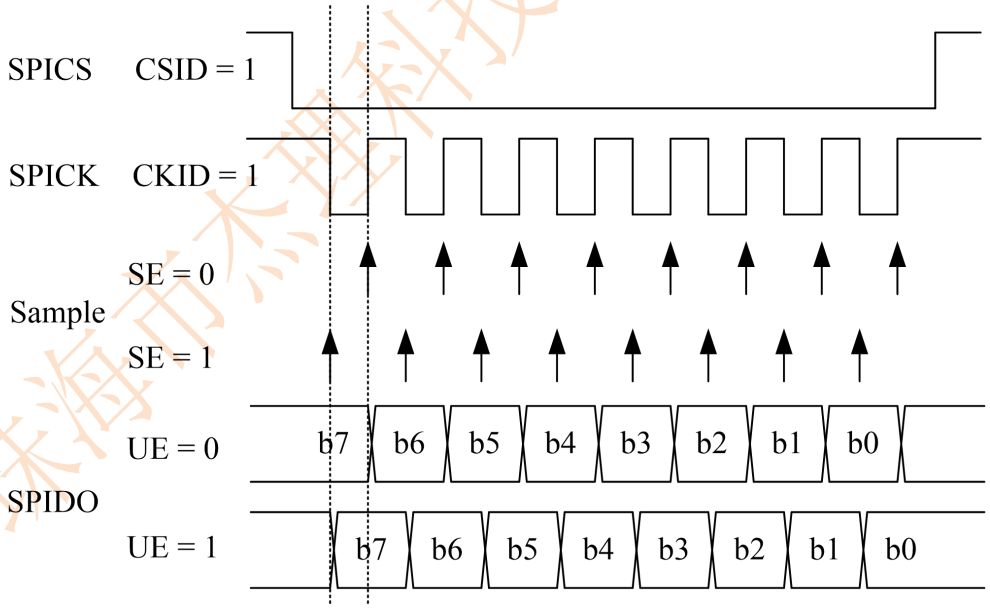


图 14-2 SPI 时序图 2

第 15 章.UART

15.1.概述

UART0、 UART1、UART2 支持接收带循环 Buffer 的 DMA 模式和普通模式。

UART0 在 DMA 接收的时候有一个循环 Buffer，UTx_RXSADR 表示它的起始，UTx_RXEADR 表示它的结束。同时，在接收过程中，会有一个超时计数器（UTx_OTCNT），如果在指定的时间里没有收到任何数据，则超时中断就会产生。超时计数器是在收到数据的同时自动清空。

15.2.控制寄存器

寄存器列表	UART0	UART1	UART2
UTx_CON(0)	UT0_CON0	UT1_CON	UT2_CON
UTx_CON1	UT0_CON1	UT1_CON1	UT2_CON1
UTx_CON2	UT0_CON2	UT0_CON2	UT0_CON2
UTx_BAUD	UT0_BAUD	UT1_BAUD	UT2_BAUD
UTx_BUF	UT0_BUF	UT1_BUF	UT2_BUF
UTx_TXADR	UT0_TXADR	UT1_TXADR	UT2_TXADR
UTx_TXCNT	UT0_TXCNT	UT1_TXCNT	UT2_TXCNT
UTx_RXCNT	UT0_RXCNT	UT1_RXCNT	UT2_RXCNT
UTx_RXSADR	UT0_RXSADR	UT1_RXSADR	UT2_RXSADR
UTx_RXEADR	UT0_RXEADR	UT1_RXEADR	UT2_RXEADR
UTx_HRXCNT	UT0_HRXCNT	UT1_HRXCNT	UT2_HRXCNT
UTx_OTCNT	UT0_OTCNT	UT1_OTCNT	UT2_OTCNT

1. UTx_CON(0): uart x control register (16bit addressing).

Bit	Name	RW	Default	Description
15	TPND	r	0	TPND:TX Pending
14	RPND	r	0	RPND:RX Pending& Dma_Wr_Buf_Empty，数据接收不完 Pending 不会为 1
13	CLRTPND	w	0	CLRTPND：清空 TX Pending
12	CLRRPND	w	0	CLRRPND：清空 RX Pending
11	OTPND	r	0	OTPND:OverTime Pending
10	CLR_OTPND	w	0	CLR_OTPND: 清空 OTPND
9			0	
8			0	
7	RDC	w	0	RDC：写 1 时，将已经收到的数目写到 UTx_HRXCNT，已收到的数目清零写 0 无效
6	RX_MODE	rw	0	RXMODE：读模式选择 0:普通模式，不用 DMA； 1:DMA 模式。
5	OT_IE	rw	0	OTIE:OT 中断允许 0：不允许； 1:允许。
4	DIVS	rw	0	DIVS:前 3 分频选择，0 为 4 分频，1 为 3 分频
3	RXIE	rw	0	RXIE:RX 中断允许 当 RX Pending 为 1，而且 RX 中断允许为 1，则会产生中断
2	TXIE	rw	0	TXIE:TX 中断允许 当 TX Pending 为 1，而且 TX 中断允许为 1，则会产生中断
1	UTRXEN	rw	0	UTRXEN:UART 模块接收使能
0	UTTXEN	rw	0	UTTXEN:UART 模块发送使能

2. UTx_CON1: uart x control register1 (16bit addressing).

Bit	Name	RW	Default	Description
15	CTSPND	r	0	CTSPND:CTS 中断 pending
14	CLR_CTSPND	wo	0	CLR_CTSPND：清楚 CTS pending
13	CLRRTS	wo	0	CLRRTS：清除 RTS 0:N/A 1:清空 RTS
12-5	reserved	rw	0	预留
4	RX_DISABLE	rw	0	关闭数据接收 0:开启输入（正常模式） 1:关闭输入（输入固定为 1）
3	CTSIE	rw	0	CTSIE:CTS 中断使能 0:禁止中断 1:中断允许 注：只有 UART1 有该功能
2	CTSE	rw	0	CTSE:CTS 使能 0:禁止 CTS 硬件流控制 1：允许 CTS 硬件流控制 注：只有 UART1 有该功能
1	reserved	rw	0	reserved
0	RTSE	rw	0	RTSE:RTS 使能 0:禁止 RTS 硬件流控制 1:允许 RTS 硬件流控制 注：只有 UART1 有该功能

3. UTx_CON2: uart x control register2 (16bit addressing).

Bit	Name	RW	Default	Description
15-3			0	
2	RB8	r	0	RB8:9Bit 模式时。RX 接收到的第 9 位
1	TB8	rw	0	TB8:9Bit 模式时，TX 发送的第 9 位
0	M9EN	rw	0	M9EN:9bit 模式使能

4. UTx_BAUD: uart x baudrate register (16bit addressing, Write Only).

Bit	Name	RW	Default	Description
15-0	UTx_BAUD	wo	0	

uart 的 UTx_DIV 的整数部分

串口频率分频器因子(UTx_DIV)的整数部分

当 DIVS=0 时，

Baudrate = Freq_sys / ((UTx_BAUD+1) *4 + BAUD_FRAC)

当 DIVS=1 时，

Baudrate = Freq_sys / ((UTx_BAUD+1) *3 + BAUD_FRAC)

(Freq_sys 是 apb_clk,指慢速设备总线的时钟，非系统时钟)

5. UTx_BUF: uart x data buffer register (8bit addressing).

Bit	Name	RW	Default	Description
8	UT0BUF	rw	0	uart 的收发数据寄存器 写 UTx_BUF 可启动一次发送；（w） 读 UTx_BUF 可获得已接收到的数据。（r）

6. UTx_TXADR：uart x TX DMA address(25bit addressing, Write Only)

Bit	Name	RW	Default	Description
24-0	UTx_TXADR	wo	0	DMA 发送数据的起始地址

7. UTx_TXCNT：uart x TX DMA count (32bit addressing, Write Only).

Bit	Name	RW	Default	Description
31-0	UTx_TXADR	wo	0	写 UTx_TXCNT，控制器产生一次 DMA 的操作，同时清空中断，当 uart 需要发送的数据达到 UTx_TXCNT 的值，控制器会停止发送数据的操作，同时产生中断（UTx_CON[15]）。

8. UTx_RXCNT：uart x receive DMA count(32bit addressing, Write Only).

Bit	Name	RW	Default	Description
31-0	UTx_RXCNT	wo	0	写 UTx_RXCNT，控制器产生一次 DMA 的操作，同时清空中断，当 uart 需要接收的数据达到 UTx_RXCNT 的值，产生中断（UTx_CON[14]）。

9. UTx_RXSADR：uart x receive DMA address(25bit addressing, Write Only)

Bit	Name	RW	Default	Description
24-0	UTx_RXSADR	wo	0	DMA 接收数据时，循环 buffer 的起始地址

10. UTx_RXEADR：uart x receive DMA end address(25bit addressing, Write Only).

Bit	Name	RW	Default	Description
24-0	UTx_RXEADR	wo	0	DMA 接收数据时，循环 buffer 的结束地址

11. UTx_HRXCNT：uart x have receive DMA count(32bit addressing, Read Only).

Bit	Name	RW	Default	Description
31-0	UTx_HRXCNT	ro	0	当设这 RDC（UTx_CON[7]）=1 时，串口设备会将当前总共收到的字节数记录到 UTx_HRXCNT 里。

12. UTx_OTCNT：uart x OverTime count(32bit addressing, Write Only).

Bit	Name	RW	Default	Description
31-0	UTx_OTCNT	wo	0	

设置串口设备在等待多久 RX 下降沿的时间，如果在所设置的时间里没收到 RX 的下降沿，则产生 OT 中断（OT_PND）

Time(ot)= Time(uart_clk) * UTx_OTCNT

例如：波特率时间为 100ns，UTx_OTCNT = 10,那么，OT 的时间就为 1000ns

【注意】:

1)OT_PND 触发流程，满足一下（1-4）顺序，则可产生 OT_PND：

i.清 OT_PND；

ii.写 UTx_OTCNT；

iii.收到 n 个 byte 数据；

iv.从最后一个下降沿开始，等待 OT 超时；

v.当 OTCNT 与超时时间相等，OT_PND 置 1。

2)OT_PND 不受 RPND 影响，即 RX_PND 置 1 后，后面如果没有再写一次 UTx_OTCNT，OT_PND 还是会起来；

3)下降沿包括但不限于 start_bit，数据从 1 到 0 变化；

4)如果芯片有 lsb_clk，uart_clk 就是 lsb_clk，否则看该芯片的时钟说明；

第 16 章.USB BRIDGE

16.1.概述

负责控制系统与 USB 模块之间的通讯，包括：

1. 接收 CPU 的命令，控制 SIE；
2. 管理 endpoint mapping；
3. 负责 SIE 和 memory 的通讯；
4. 控制 USB PHY。

寄存器列表：

寄存器列表	USB			
USB_CONx	USB_CON0		USB_CON1	
EPx_CNT	EP0_CNT	EP1_CNT	EP2_CNT	EP3_CNT
	EP4_CNT			
EP0_ADR	EP0 _ADR			
EPx_TADR	EP0_TADR	EP1_TADR	EP2_TADR	EP3_TADR
	EP4_TADR			
EPx_RADR	EP0_RADR	EP1_RADR	EP2_RADR	EP3_RADR
	EP4_RADR			
USB_IO_CONx	USB_IO_CON0			
USB_SIE	FADDR	POWER	INTRTX1	INTRTX2
	INTRRX1	INTRRX2	INTRUSB	INTRTX1E
	INTRTX2E	INTRRX1E	INTRRX2E	INTRUSBE
	FRAME1	FRAME2	DEVCTL	
USB_EP0	CSR0		COUNT0	
USB_EPX	TXMAXP	TXCSR1	TXCSR2	RXMAXP
	RXCSR1	RXCSR2	RXCOUNT1	RXCOUNT2
	TXTYPE	TXINTERVAL	RXTYPE	RXINTERVAL

TXDLY_CON				
EPx_RX_LEN	EP1_RX_LEN	EP2_RX_LEN	EP3_RX_LEN	EP4_RX_LEN
EP1_MTX_PRD				
EP1_MTX_NUM				
EP1_MRX_PRD				
EP1_MRX_NUM				

Endpoint 大小：

EP 号	BUFFER 大小(BYTE)
EP0	64
EP1 TX	1024
EP1 RX	1024
EP2_TX	1024
EP2 RX	1024
EP3 TX	1024
EP3 RX	1024
EP4 TX	1024
EP4 RX	1024

16.2.寄存器说明

1. USB_CON0

Bit	Name	RW	Default	Description
31-28	-	-	0	预留
27	EP4_RLIM_EN	rw	0	EP4 写限制使能
26	EP3_RLIM_EN	rw	0	EP3 写限制使能
25	EP2_RLIM_EN	rw	0	EP2 写限制使能
24	EP1_RLIM_EN	rw	0	EP1 写限制使能

23	EP4_DISABLE	rw	0	关闭端点 4，不会返回 ack, nak, stall
22	EP3_DISABLE	rw	0	关闭端点 3，不会返回 ack, nak, stall
21	EP2_DISABLE	rw	0	关闭端点 2，不会返回 ack, nak, stall
20	EP1_DISABLE	rw	0	关闭端点 1，不会返回 ack, nak, stall
19	RST_STL	rw	0	EP0 从 stall 状态复位为 idle 状态
13	SOF_PND	r	0	SOF 中断请求标志
12	CLR_SOFP	w	0	写 1 清除 SOF 中断请求标志，写 0 无效
11	SIEIE	rw	0	SIE 中断使能
10	SOFIE	rw	0	SOF 中断使能
9	PDCHKDP	rw	0	DP 外接下拉检查使能 0：disable 1：enable
8-7	-	-	0	预留
6	USB_TEST	rw	0	USB 测试模式
5	VBUS	rw	0	USB 电源
4	CID	rw	0	USB 工作模式： 0：host 1：device
3	TM1	rw	0	用于缩短检测连接时间(short connect timeout)： 0：disable 1：enable
2	USB_NRST	rw	0	USB 模块复位： 0：reset 1：release reset
1	LOW_SPEED	rw	0	低速 USB_DMA 使能： 0：disable 1：enable
0	PHY_ON	rw	0	USB_PHY 使能：

				0：disable 1：enable

2. USB_CON1

Bit	Name	RW	Default	Description
31-6	-	-	0	reserved
5	ep1_mrx_pnd	r	0	ep1 连续接收完成标记位
4	ep1_mtx_pnd	r	0	ep1 连续发送完成标记位
3	ep1_mrx_pnd_clr	w	0	清除 ep1 连续接收完成标记位
2	ep1_mtx_pnd_clr	w	0	清除 ep1 连续发送完成标记位
1	ep1_mrx_en	rw	0	ep1 连续接收使能。
0	ep1_mtx_en	rw	0	ep1 连续发送使能。 注意： 连续收发期间，最后一包数据完成才会起正常传输中断，期间如 stall 等异常中断也会起来； 连续收发的数据存档地址是连续的；

3. EP0_CNT，EP1_CNT，EP2_CNT，EP3_CNT，EP4_CNT

Bit	Name	RW	Default	Description
31-0	EP(n)_CNT	w	0	usb endpoint 0-4 发送数据个数，单位为 byte

4. EP0_ADR

Bit	Name	RW	Default	Description
31-0	EP0_ADR	w	0	usb endpoint 0 发送/接收数据的起始地址

5. EP1_TADR，EP2_TADR，EP3_TADR，EP4_TADR

Bit	Name	RW	Default	Description
31-0	EP(n)_TADR	w	0	usb endpoint 1-4 发送数据的起始地址

6. EP1_RADR，EP2_RADR，EP3_RADR，EP4_RADR

Bit	Name	RW	Default	Description
31-0	EP(n)_RADR	w	0	usb endpoint 1-4 接收数据的起始地址

7. USB_IO_CON0

Bit	Name	RW	Default	Description
31-15	-	-	0	预留
14	IO_PU_MODE	rw	0	IO 上下拉分两种模式 0：USB 模式 1：普通 IO 模式
13	SR	rw	0	输出驱动能力选择 0：slow 1：fast
12	IO_MODE	rw	0	IO 模式使能 0：USB 模式 1：普通 IO 模式
11	DMDIEH	rw	0	3.3V DM 数字输入使能
10	DPDIEH	rw	0	3.3V DP 数字输入使能
9	DMDIE	rw	0	1.2V DM 数字输入使能
8	DPDIE	rw	0	1.2V DP 数字输入使能
7	DMPD	rw	0	DM 下拉
6	DPPD	rw	0	DP 下拉
5	DMPU	rw	0	DM 上拉
4	DPPU	rw	0	DP 上拉

3	DMIE	rw	0	DM 方向 0：输出 1：输入
2	DPIE	rw	0	DP 方向 0：输出 1：输入
1	DMOUT	rw	0	DM 输出电平
0	DPOUT	rw	0	DP 输出电平

8. TXDLY_CON

Bit	Name	RW	Default	Description
31-16	prd	rw	0	设置令牌包与数据包的延迟时钟个数
3-15	-	-	0	-
2	PND	r	0	延迟时钟数等于 PRD 时的标记位
1	CLR_PND	w	0	清除 pending
0	CLK_DIS	rw	0	1：开启令牌包与数据包的延迟功能 0：关闭令牌包与数据包的延迟功能

9. EPx_RX_LEN: EP1_RX_LEN, EP2_RX_LEN, EP3_RX_LEN, EP4_RX_LEN

Bit	Name	RW	Default	Description
31-11	-	-	0	reserved
10-0	EPx_RX_LEN	rw	0	EP（x）写数据最大地址长度（EPx_BASE_ADR 到 EPx_BASE_ADR+LEN-1）， 写满 len 个数后， dma 停止接收数据。 注意：每一次新的传输，dma 地址都会回到 BASE_ADR.

10. EP1_MTX_PRD

Bit	Name	RW	Default	Description
31-8	-	-	0	reserved
7-0	EP1_MTX_PRD	rw	0	EP1 连续发送的包数

11. EP1_MTX_NUM

Bit	Name	RW	Default	Description
31-8	-	-	0	reserved
7-0	EP1_MTX_NUM	r	0	EP1 连续发送中已发送包数，写 mtx_pnd_clr 会清 0

12. EP1_MRX_PRD

Bit	Name	RW	Default	Description
31-8	-	-	0	reserved
7-0	EP1_MRX_PRD	rw	0	EP1 连续接收的包数

13. EP1_MRX_NUM

Bit	Name	RW	Default	Description
31-8	-	-	0	reserved
7-0	EP1_MRX_NUM	r	0	EP1 连续发送中已接收包数，写 mtx_pnd_clr 会清 0

第 17 章.PWM LED 灯

17.1.概述

PWM LED 是通过脉冲宽度调节控制 LED 的亮度与亮灭的一个模块。放在 1.2V SYSVDD，不可以在软关机下工作。

PWM LED 模块可以配置两个周期大小不同的 PWM0、PWM1，分别控制 LED 的亮度（PWM0）与亮灭（PWM1）。

17.2.特性

1.所有 IO 可用。

2.PDOWN（蓝牙 sniff）可用，SOFF 不可用。

3.目前只放了一个模块，只支持单 IO 单 LED、单 IO 双 LED。

4.驱动档位特性：

a）PWM 推高电平: 高阻，上拉，输出 1 强驱 0，输出 1 强驱 1，输出 1 强驱 2，输出 1 强驱 3，保持，输出 1 强驱 2，输出 1 强驱 1，输出 1 强驱 0，上拉，高阻， 保持（循环以上步骤）

b）PWM 推低电平: 高阻，下拉，输出 0 强驱 0，输出 0 强驱 1，输出 0 强驱 2，输出 0 强驱 3，保持，输出 0 强驱 2，输出 0 强驱 1，输出 0 强驱 0，下拉，高阻， 保持（循环以上步骤）

c）最高驱动档位可设， 强驱及高阻保持时间可设， 驱动切换等待时间可设（所有等待时间为同样的 n 个 PWM0_CLK）

5.5. 推灯功能：

a）固定亮度

i.（单 LED）单闪 ；双闪（双闪的两个时间长度可调，两个间隔可调）；

ii.（双 LED）单闪 变色 单闪；

双闪 变色 双闪（双闪的两个时间长度可调，两个间隔可调）；

n 闪 变色 n 闪（n<7）；

a）7.2 呼吸灯

i.（单 LED）灭灯时间长度可调，亮灯时间长度可调，亮灯快慢可调，灭灯快慢可调；

ii.（双 LED）（灭灯时间长度可调，亮灯快慢可调，灭灯快慢可调）呼吸 变色 呼吸；

无灯---A 灯逐渐变到最亮---最亮变色---B 灯由最亮逐渐变到最暗---变色（无灯）---A 灯逐渐变到最亮...；

无灯---A 灯呼吸---无灯---A 灯逐渐变到 最亮---最亮变色---B 灯由最亮逐渐变到最暗---无灯---B 灯呼吸---变色（无灯）---A 灯呼吸...；

17.3.寄存器并接

1.PWM_ PRD_DIV ：pwm1_clk 分频比控制器，12bit

PWM_ PRD_DIV[11:0] = {P3_PWM_CON3[3:0],PWM_ PRD_DIVL[7:0]}；

2.PWM_BRI_PRD[9:0]： 亮度周期，10bit

PWM_BRI_PRD [9:0] = { P3_PWM_BRI_PRDH [1:0], P3_PWM_BRI_PRDL [7:0]}；

3.P3_PWM_BRI_DUTY0 [9:0]： 推高电平点亮的灯，亮度占空比，10bit

P3_PWM_BRI_DUTY0 [9:0] = { P3_PWM_BRI_PRD0H [1:0], P3_PWM_BRI_PRD0L [7:0]}；

4.P3_PWM_BRI_DUTY1 [9:0]：推低电平点亮的灯，亮度占空比，10bit

P3_PWM_BRI_DUTY1[9:0] = { P3_PWM_BRI_PRD1H [1:0], P3_PWM_BRI_PRD1L [7:0]}；

5.呼吸灯，灭灯延时计数器，16bit

{PWM_DUTY3[7:0] ,PWM_DUTY2[7:0]}；

17.4.时钟控制

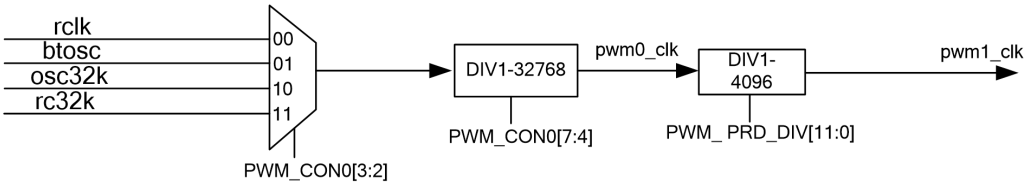


图 26-1 时钟控制框图

其中 PWM0 由 pwm0_clk 驱动，周期 PRD0 由 PWM_BRI_PRD0/1 10bit 决定，有一个循环计数 bright_cnt 决定显示亮度。

其中 PWM1 由 pwm1_clk 驱动，周期 PRD1 最多 256 个时钟，有一个循环计数 counter_cnt。特别地，当呼吸灯功能打开，PWM0、PWM1 分别控制 LED 的亮度级数变化及最亮和最暗保持的时间。

17.5.二级亮灭控制

PWM1 可以控制 LED 在一个 PRD1 周期亮灭一次或两次。

下图以一个 LED 灯 PWM 电平举例：

输出高电平亮

单周期双闪

单灯

非呼吸灯模式

配置：

PWM_DUTY3_EN = 0；

PWM_DUTY2_EN = 1；

PWM_DUTY1_EN = 1；

PWM_DUTY0_EN = 1；

PWM1_PRD_SEL = 0； //PWM1 PRD 为 256 pwm1_clk

PWM1_INV = 0;//固定为 0 不变，亮灭不取反，

如下图，周期开始时是从灯灭开始

PWM0_INV = 0；//固定为 0 不变，灯亮度控制

OUT_LOGIC = 1；//固定为 1 不变，PWM 输出逻辑控制

BREATHE_EN = 0;

SHIFT_DUTY = 0;

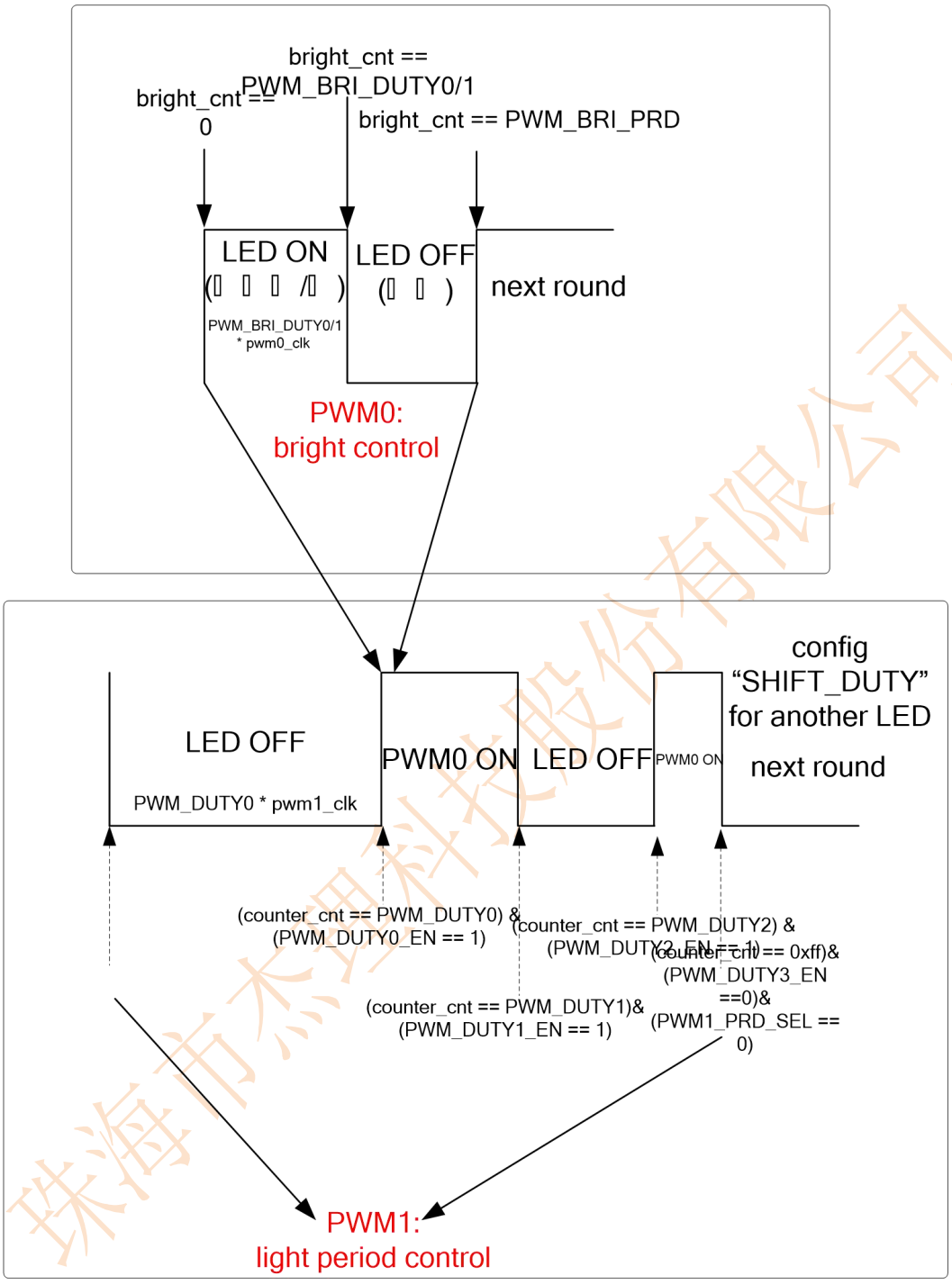
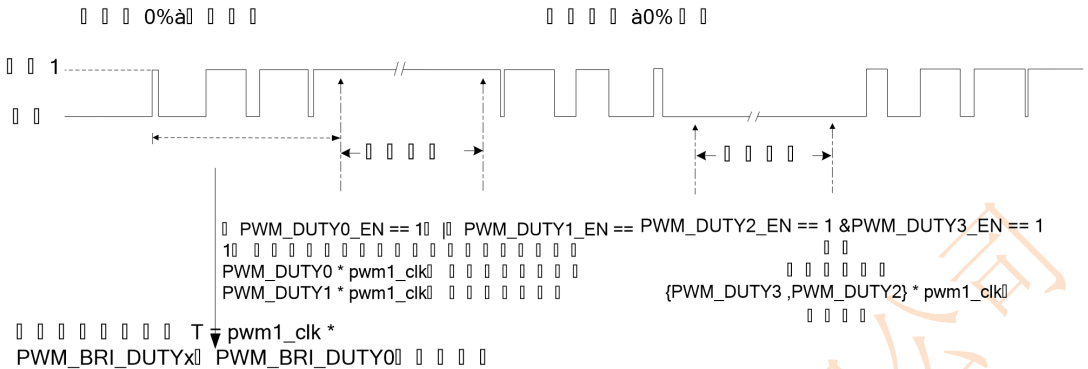


图 26-2 二级亮灭控制框图

【注意】：需要同时推两个颜色的灯时，请把 PWM1 的周期设置得尽可能短，小于人眼分辨极限即可。

17.6.呼吸灯控制



PWM_DUTY0_EN == 1 || PWM_DUTY1_EN == 1
PWM_DUTY0 * pwm1_clk
PWM_DUTY1 * pwm1_clk
PWM_DUTY2_EN == 1 &PWM_DUTY3_EN == 1
{PWM_DUTY3 ,PWM_DUTY2} * pwm1_clk

T = pwm1_clk * PWM_BRI_DUTYx PWM_BRI_DUTY0
PWM_BRI_DUTY1
pwm0_clk 25uS pwm1_clk 5mS

PWM_ PRD_DIV = 200 PWM_BRI_PRD = 200
PWM_BRI_DUTY0 = 150
PWM_BRI_DUTY1 = 170

150/200
T0 = pwm1_clk * PWM_BRI_DUTY0= 750mS
N0 = T0/pwm0_clk*PWM_BRI_PRD=150

170/200
T1 = pwm1_clk * PWM_BRI_DUTY1= 850mS
N1 = T1/pwm0_clk*PWM_BRI_PRD=170

tmp = PWM_BRI_DUTY0 - PWM_BRI_DUTY1

17.7.3.3v 系统控制寄存器说明

1.PWM_CON0: PWM control register 0 (8bit addressing)

Bit	Name	RW	Default	Description
7-4	PWM_PSET	rw	0	PWM_PSET:PWM pre_scaler setting。设置 PWM0 的分频器除法因子 [1:0]: 00:div101:div4 10:div16 11:div64

				[2] :0: x11: x2 [3] :0: x11: x256 计算方式:DIV = [1:0]的 divx 乘以 [2]倍数 乘以 [3]倍数。
3-2	PWM_SSEL	rw	0	PWM_SSEL:PWM 时钟源的选择。 00：rclk (有偏差 250KHz) 01：btosc (准确 12/24MHz) 10：osc32k (准确 32KHz) 11：led_clk
1	BREATHE_EN	rw	0	呼吸灯使能。 0：呼吸灯功能关闭 1：呼吸灯功能打开
0	PWM_EN	rw	0	PWM 使能。 0：模块总使能关闭 1：模块总使能打开

2.P2M_CLK_CON0 (P11 to main system clock control)

Bit	Name	RW	Default	Description
7-5	led_clk_sel	rw	0	LED 模块时钟 led_clk 选择： 0 : none 1 : rc_250k(有偏差 250KHz) 2 : rc_16m(有偏差 16MHz) 3 : bt_osc(准确 12/24MHz) 4 : lrc_osc(需校准 32KHz) 5 : rtc_osc (准确 32KHz)

3.PWM_CON1: PWM control register 1 (8bit addressing)

Bit	Name	RW	Default	Description

7	PWM_DUTY3_EN	rw	0	亮灭 DUTY 控制 3 开关/呼吸灯空周期延时开关高 8bit 0：亮灭 DUTY 控制 3 关闭（BREATHE_EN == 0 & PWM1_PRD_SEL == 0）/ 呼吸灯空周期延时关闭（BREATHE_EN == 1） 1：亮灭 DUTY 控制 3 开启（BREATHE_EN == 0 & PWM1_PRD_SEL == 0）/ 呼吸灯空周期延时开启（BREATHE_EN == 1）
6	PWM_DUTY2_EN	rw	0	亮灭 DUTY 控制 2 开关/呼吸灯空周期延时开关低 8bit 0：亮灭 DUTY 控制 2 关闭（BREATHE_EN == 0）/ 呼吸灯满周期延时关闭（BREATHE_EN == 1） 1：亮灭 DUTY 控制 2 开启（BREATHE_EN == 0）/ 呼吸灯满周期延时开启（BREATHE_EN == 1）
5	PWM_DUTY1_EN	rw	0	亮灭 DUTY 控制 1 开关/呼吸灯最高亮度延时开关(高电平灯) 0：亮灭 DUTY 控制 1 关闭（BREATHE_EN == 0）/ 呼吸灯空周期延时关闭（BREATHE_EN == 1） 1：亮灭 DUTY 控制 1 开启（BREATHE_EN == 0）/ 呼吸灯空周期延时开启（BREATHE_EN == 1）
4	PWM_DUTY0_EN	rw	0	PWM_DUTY0_EN:亮灭 DUTY 控制 0 开关/呼吸灯最高亮度延时开关 0：亮灭 DUTY 控制 0 关闭（BREATHE_EN == 0）/ 呼吸灯满周期延时关闭（BREATHE_EN == 1） 1：亮灭 DUTY 控制 0 开启（BREATHE_EN == 0）/ 呼吸灯满周期延时开启（BREATHE_EN == 1）
3	PWM1_PRD_SEL	rw	0	PWM1 亮灭周期选择： 0：PWM1 周期为 0xff 1：PWM1 周期为 PWM_DUTY3
2	PWM_OUTPUT_INV	rw	0	输出电平取反（周期变色 SHIFT_DUTY==0 时可用）

				0： 不取反。 1： 取反
1	PWM1_INV	rw	0	PWM1 输出的 LED 亮灭波形取反（默认由 0 开始），可以控制亮灭周期是从灯亮开始还是灯灭开始，软件常设为 0。 0：PWM1 输出的 LED 亮度波形不取反（灭灯开始的 PWM1 周期） 1：PWM1 输出的 LED 亮度波形取反（亮灯开始的 PWM1 周期）
0	PWM0_INV	rw	0	PWM0 输出的 LED 亮度波形取反（默认由 1 开始），软件常设为 0 0：PWM0 输出的 LED 亮度波形不取反 1：PWM0 输出的 LED 亮度波形取反

4.PWM_CON2: PWM control register 2 (8bit addressing)

Bit	Name	RW	Default	Description
7-4	SHIFT_DUTY	rw	0	LED 周期变色（输出电平取反），非呼吸灯 0000：不变色 0001：1 个 PWM1 周期变色 … 1111：15 个 PWM1 周期变色 LED 周期变色，呼吸灯 PWM_CON2[4]常设为 0 0000：不变色 0010：1 个周期变色（1 个周期为：最暗到最亮，再到最暗） 0100：2 个周期变色 … 1110：7 个周期变色

3-2	IO_DRV_SHIFT_CLK	rw	0	输出驱动切换时钟个数（PWM0_CLK） 00：1 01：2 10：4 11：8
1-0	IO_MAX_DRV	rw	0	IO 最大输出强度 00：2.4mA（8mA mos + 120Ω res） 01：8mA（8mA mos ） 10：18.4mA（24mA mos + 120Ω res） 11：24mA（24mA mos ）

5.PWM_CON3: PWM control register 3 (8bit addressing)

Bit	Name	RW	Default	Description
7	PENDING	r	0	亮灭周期结束，呼吸灯周期结束，pengding 置 1
6	CLEAR PENDING	w	0	清 pending 0：无效 1：清 pending
5	IE	rw	0	PWM LED pending 中断使能。
4	OUT_LOGIC	rw	0	输出逻辑， 软件常设为 1 0：LED 在低电平时为亮（PWM0 | PWM1）。 1：LED 在高电平时为亮（PWM0 & PWM1）。
3-0	PWM_PRD_DIVH	rw	0	pwm1 clk 分频比控制高位 PWM_PRD_DIV[11:8]

6.PWM_BRI_PRDL: PWM0 brightness period register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_BRI_PRD[7:0]	rw	0	亮度周期低 7bit

7.PWM_BRI_PRDH: PWM0 brightness period register (8bit addressing)

Bit	Name	RW	Default	Description
1-0	PWM_BRI_PRD[9:8]	rw	0	亮度周期高 2bit

8.PWM_BRI_DUTY0L: PWM0 brightness duty register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_BRI_DUTY0 [7:0]	rw	0	亮度控制低 7bit，例： 高电平灯亮 PWM_BRI_DUTY0 == 0； 灯最暗 PWM_BRI_DUTY0 == 1； 灯亮度增强一档 … PWM_BRI_DUTY0 == PWM_BRI_PRD； 灯最亮

9.PWM_BRI_DUTY0H: PWM0 brightness duty register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_BRI_DUTY0 [9:8]	rw	0	亮度控制高 2bit，例： 高电平灯亮

10.PWM_BRI_DUTY1L: PWM0 brightness duty register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_BRI_DUTY1 [7:0]	rw	0	亮度控制低 7bit，例： 低电平灯亮 PWM_BRI_DUTY0 == 0； 灯最暗 PWM_BRI_DUTY0 == 1； 灯亮度增强一档 … PWM_BRI_DUTY0 == PWM_BRI_PRD； 灯最亮

11.PWM_BRI_DUTY1H: PWM0 brightness duty register (8bit addressing)

Bit	Name	RW	Default	Description

7-0	PWM_BRI_DUTY1 [9:8]	rw	0	亮度控制高 2bit，例：低电平灯亮

12.PWM_ PRD_DIVL: PWM1 period divider register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_PRD_DIV[7:0]	rw	0	PWM 时钟的分频因子设置。 PWM_ PRD_DIV ：pwm1_clk 分频比控制器，12bit PWM_ PRD_DIV[11:0] = {P3_PWM_CON3[3:0],PWM_PRD_DIVL[7:0]}; 0x000:div = 1; 0x001:div = 2; … … 0xfff:div = 4096;

13.PWM_DUTY0: PWM duty0 register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_DUTY0	rw	0	BREATHE_EN == 0 & PWM_DUTY0_EN == 1: 亮灭 DUTY 控制 1 BREATHE_EN == 1 & PWM_DUTY0_EN == 1: 呼吸灯最高亮度延时(低电平灯)

14.PWM_DUTY1: PWM duty1 register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_DUTY1	rw	0	BREATHE_EN == 0 & PWM_DUTY1_EN == 1: 亮灭 DUTY 控制 1 BREATHE_EN == 1 & PWM_DUTY1_EN == 1: 呼吸灯最高亮度延时(高电平灯)

15.PWM_DUTY2: PWM duty2 register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_DUTY2	rw	0	BREATHE_EN == 0 & PWM_DUTY2_EN == 1: 亮灭 DUTY 控制 2 BREATHE_EN == 1 & PWM_DUTY2_EN == 1: 呼吸灯灭灯延时低 8bit

16.PWM_DUTY3: PWM duty3 register (8bit addressing)

Bit	Name	RW	Default	Description
7-0	PWM_DUTY3	rw	0	BREATHE_EN == 0 & PWM1_PRD_SEL == 0 & PWM_DUTY3_EN == 1: 亮灭 DUTY 控制 3 BREATHE_EN == 0 & PWM1_PRD_SEL == 1: 亮灭 PWM1 周期 PWM1_PRD BREATHE_EN == 1 & PWM_DUTY3_EN == 1: 呼吸灯灭灯延时高 8bit

17.8.软件 DEMO

```
//-------------------------------------------//
// led define for breathe demo
//-------------------------------------------//
#define led_con0_init                                   \
        /* PWM0 PDIV          4bit RW   */ ( 1<<4 )   |\
        /* CLKSEL             2bit RW   */ ( 3<<2 ) /* 0:RC250k 1:BTOSC 2:OSL 3:LRC */ |\
        /* BREATHE            1bit RW   */ ( 1<<1 )   |\
        /* EN                 1bit RW   */ ( 1<<0 )

         //PWM0 PDIV      //pre-scaler setting,      [1:0]:   00:div1      01:div4      10:div16
```

```
11:div64
                                //                          [2]:    0: x1          1: x2
                                //                          [3]:    0: x1          1: x256


#define led_con1_init                                   \
        /* PWM_DUTY3_EN        1bit RW    */ ( 1<<7 )    | \
        /* PWM_DUTY2_EN        1bit RW    */ ( 1<<6 )    | \
        /* PWM_DUTY1_EN        1bit RW    */ ( 1<<5 )    | \
        /* PWM_DUTY0_EN        1bit RW    */ ( 1<<4 )    | \
        /* PWM1_PRD_SEL        1bit RW     */ ( 0<<3 ) /* 0 pwm1 round end by 0xff; 1 pwm1
round end by pwm_duty3 */ | \
        /* PWM_OUT_INV         1bit RW    */ ( 0<<2 ) /* using in shift == 0 */ | \
        /* PWM_EDGE1           1bit RW    */ ( 0<<1 )    | \
        /* PWM_EDGE0           1bit RW    */ ( 0<<0 )


#define led_con2_init                                   \
        /* SHIFT PRD          4bit RW    */ ( 2<<4 )    | \
        /* DRVING CYCLE       2bit RW    */ ( 2<<2 )    | \
        /* DRVING MAX         2bit RW    */ ( 2<<0 )


#define led_con3_init                                   \
        /* PWM PND            1bit RO    */ ( 0<<7 )    | \
        /* PWM CLR PND        1bit WO    */ ( 1<<6 )    | \
        /* PWM IE             1bit RW    */ ( 1<<5 )    | \
        /* OUT_LOGIC          1bit RW    */ ( 1<<4 )    | \
        /* PWM1 DIVH          4bit RW    */ ( 2<<0 )
void led_init (void)
{
```

```
    p33_tx_1byte(P3_LRC_CON0,     0x61);
    p33_tx_1byte(P3_LRC_CON1,     0xaa);

    p33_tx_1byte(P3_PWM_BRI_PRDL,     0x40);
    p33_tx_1byte(P3_PWM_BRI_PRDH,     0x01);
    p33_tx_1byte(P3_PWM_BRI_DUTY0L, 0x71);
    p33_tx_1byte(P3_PWM_BRI_DUTY0H, 0x00);
    p33_tx_1byte(P3_PWM_BRI_DUTY1L, 0x90);
    p33_tx_1byte(P3_PWM_BRI_DUTY1H, 0x00);

    p33_tx_1byte(P3_PWM_PRD_DIVL,     0x40);

    p33_tx_1byte(P3_PWM_DUTY0,     0x01);
    p33_tx_1byte(P3_PWM_DUTY1,     0x09);
    p33_tx_1byte(P3_PWM_DUTY2,     0xb0);
    p33_tx_1byte(P3_PWM_DUTY3,     0x02);

    p33_tx_1byte(P3_PWM_CON3,     led_con3_init);
    p33_tx_1byte(P3_PWM_CON2,     led_con2_init);
    p33_tx_1byte(P3_PWM_CON1,     led_con1_init);
    p33_tx_1byte(P3_PWM_CON0,     led_con0_init);
    p33_or_1byte(P3_PWM_CON3,     BIT(6));//clear pending
}
```