第 5 章 chapter 5

输入输出控制

本章学习目标

- 掌握 GPIO 的基本原理和应用。
- 掌握 LED 灯点亮方法，并在基于开发板调试。
- 熟悉 PWM 和 ADC 的基础原理。
- 熟悉各个寄存器功能。
- 掌握 SPI 的基本概念与工作方式。

本章先向读者介绍 GPIO 的基本概念与工作方式，然后介绍基于开发板的 LED 灯的点亮，之后介绍 PWM 和 ADC 的基本概念与工作方式，最后介绍 SPI 的基本概念与工作方式。

5.1 GPIO 简 介

5.1.1 概述

GPIO 是可编程的通用输入/输出接口，用于生成和采集特定应用的输入或输出信号，实现系统和外设之间的通信，方便系统控制外设。

如图 5.1 所示，GPIO 模块主要接口如下：

（1）APB 接口；

（2）I/O pad 的外部数据接口；

（3）中断信号接口。

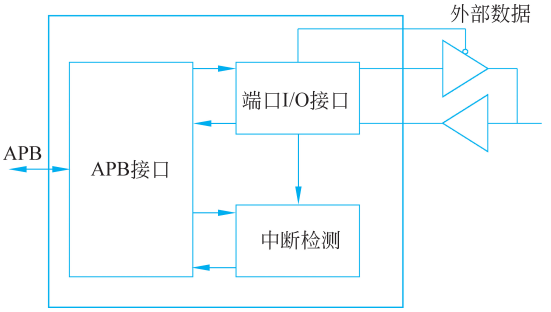

图 5.1 GPIO 模块原理图

5.1.2 GPIO 功能

GPIO 接口具有以下功能特点。

(1) 时钟源可选择：工作模式晶体时钟 24M/40M、低功耗模式 32K 时钟。

(2) 1 组 GPIO，共 15 个独立的可配置引脚。

(3) 每个 GPIO 引脚都可单独控制传输方向。

(4) 每个 GPIO 可以单独被配置为外部中断源。

(5) GPIO 用作中断时有 4 种中断触发方式：

① 上升沿触发；

② 下降沿触发；

③ 高电平触发；

④ 低电平触发。

(6) GPIO 上报一个中断，CPU 查询上报的 GPIO 编号。

(7) 每个中断支持独立屏蔽的功能，脉冲中断支持可清除功能。

5.1.3 GPIO 工作方式

GPIO 可以配置为输入或输出方式，通过 GPIO_SWPORT_DDR 配置。

(1) 当配置为输入方式时，不但可以通过引脚输入到输入端口寄存器 GPIO_EXT_PORT，还可以作为外部中断源，以及外部的睡眠唤醒信号。

(2) 当配置为输出方式时，配置 GPIO_SWPORT_DR 可以将配置数据输入引脚上。

① 中断模式可选择电平触发或边沿触发，通过 GPIO_INTTYPE_LEVEL 配置。

② 电平触发可选择高电平触发或低电平触发，通过 GPIO_INT_POLARITY 配置。

③ 单沿触发可选择上升沿触发或下降沿触发，通过 GPIO_INT_POLARITY 配置。

④ 每个中断支持独立的使能，通过 GPIO_INTEN 配置。

⑤ 每个中断支持独立的屏蔽，通过 GPIO_INTMASK 配置。每个中断的状态支持可查询：屏蔽前的原始中断状态，通过 GPIO_RAWINTSTATUS 查询；屏蔽后的最终中断状态，通过 GPIO_INTSTATUS 查询。

⑥ 脉冲中断支持可清除，通过 GPIO_PORT_EOI 配置。

5.2 点亮 LED 灯

LED 灯位于开发板的左侧，图 5.2 中显示了 NFC 指示灯、运行指示灯、电源指示灯的位置。我们需要控制运行指示灯。

图 5.3 为运行指示灯电路原理图，一端接 3.3V 高电平电源，另一端接 PCF8574 IO 扩展芯片的 P2 引脚。PCF8574 的功能为用 I^2C 来读取 P0～P7 这 8 个接口的开关状态。引脚数量是有限的，利用 PCF8574 IO 扩展芯片只需要占用主控芯片的 3 个引脚，就可以

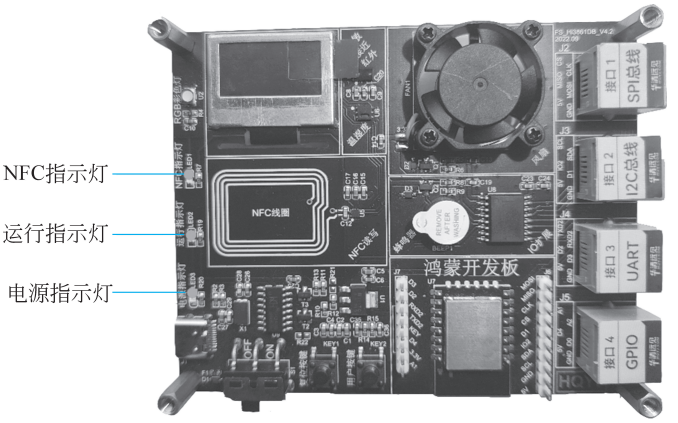

图 5.2　开发板指示灯位置

控制 8 个接口的开关状态，这样就实现了 IO 的扩展。

下面为 PCF8574 IO 扩展芯片读写协议，要求第 1 字节为地址数据。地址数据规定如图 5.4 所示，前四位为 0100，A2、A1、A0 三个数据根据电路原理图规定引脚高低电平。如图 5.5 所示，A1、A2 为低电平，即为 0；A0 为高电平，即为 1。所以从地址整体连起来为 0100001X，其中 X 位为读写控制位，读模式 X 为 1，写模式 X 为 0。

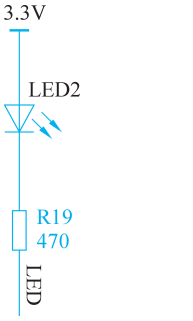

图 5.3　指示灯电路原理图

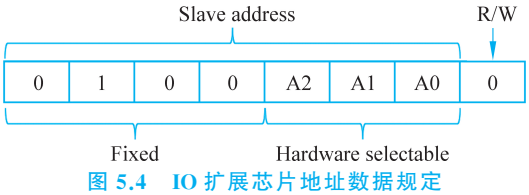

图 5.4　IO 扩展芯片地址数据规定

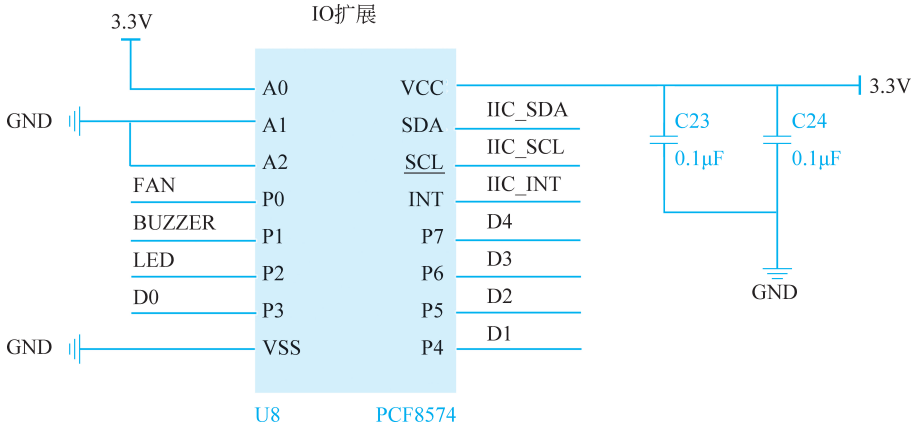

图 5.5　IO 扩展芯片电路原理图

若想读数据，则将 R/W 位设置为 1，图 5.6 所示为读模式。

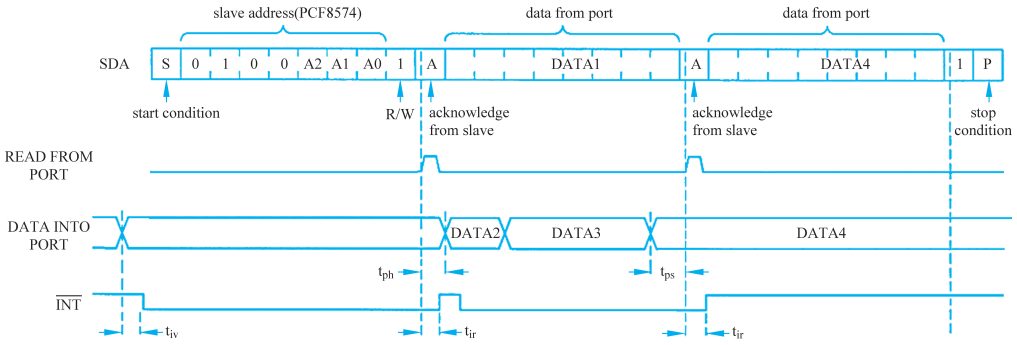

图 5.6 IO 扩展芯片读写协议（读模式）

若想要写数据，例如点亮 LED 灯，则将 R/W 位设置为 0，图 5.7 所示为写模式。

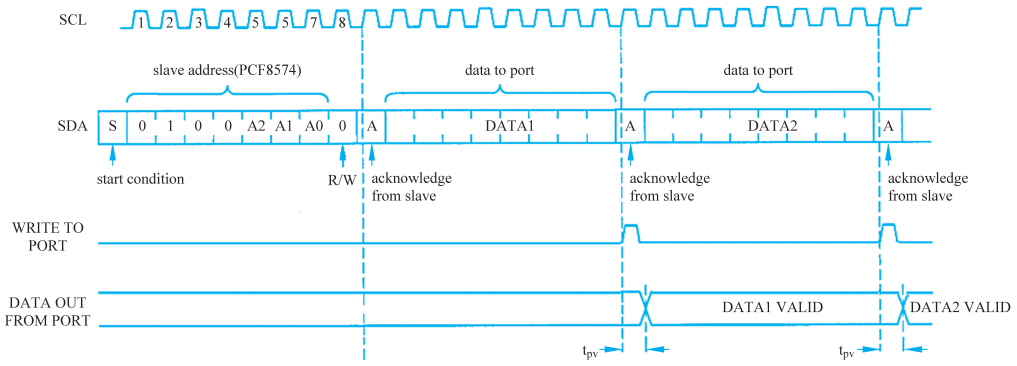

图 5.7 IO 扩展芯片读写协议（写模式）

首先创建任务，在主循环中点亮或熄灭 LED 灯。

```
/* *
  * @description: 初始化并创建任务
  * @param { * }
  * @return { * }
  */
static void base_04_i2c_pcf8574_led_demo(void)
{
    console_log_info("[demo] Enter base_04_i2c_pcf8574_led_demo()!");
    PCF8574_Init();
    osThreadAttr_t taskOptions;
    taskOptions.name = "Task1";                    // 任务的名字
    taskOptions.attr_bits = 0;                     // 属性位
    taskOptions.cb_mem = NULL;                     // 堆空间地址
    taskOptions.cb_size = 0;                       // 堆空间大小
    taskOptions.stack_mem = NULL;                  // 栈空间地址
    taskOptions.stack_size = 1024;                 // 栈空间大小 单位:字节
    taskOptions.priority = osPriorityNormal;       // 任务的优先级
    Task1_ID = osThreadNew((osThreadFunc_t)Task1, NULL, &taskOptions);
                                                   // 创建任务 1
    If (Task1_ID != NULL)
    {
```

```
        console_log_info("ID = %d, Create Task1_ID is OK!", Task1_ID);
        }
    }
    SYS_RUN(base_04_i2c_pcf8574_led_demo);
```

由于 LED 灯连接的是 IO 扩展芯片的 P2 引脚，所以若点亮 LED 灯，则让 P2 引脚为低电平，若熄灭 LED 灯，则让 P2 引脚为高电平。

```
void Task1(void * argument)
{
    (void)argument;
    console_log_info("enter Task 1.......");
    While (1)
    {
        pcf8574_io.bit.p2 = 1;                          // LED 灯熄灭
        PCF8574_Write(pcf8574_io.all);
        sleep(1);
        pcf8574_io.bit.p2 = 0;                          // LED 灯点亮
        PCF8574_Write (pcf8574_io.all);
        sleep (1);
    }
}
```

5.3 RGB 三色灯

RGB 彩色灯位于开发板的左上方，图 5.8 显示了 RGB 彩色灯位置。

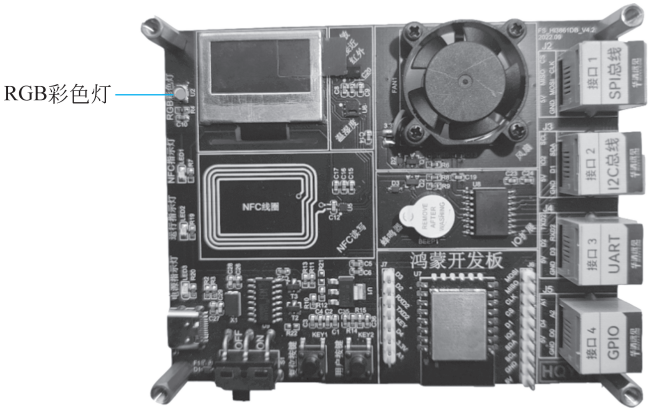

图 5.8 开发板 RGB 彩色灯位置

图 5.9 为三色 LED 灯原理图，一端连接 3.3V 高电平，另一端连接 AW2013 控制芯片，控制芯片通过 I^2C 连接 Hi3861。AW2013 控制芯片不仅可以控制 RGB 彩色灯的亮灭，还可以实现呼吸灯的效果或控制亮暗程度。三色 LED 灯支持 256 级 PWM 的亮度调节。有了 AW2013 的辅助，可以不使用 Hi3861 的 PWM，从而节省 Hi3861 的引脚。

表 5.1 中描述了芯片不同寄存器功能与含义。

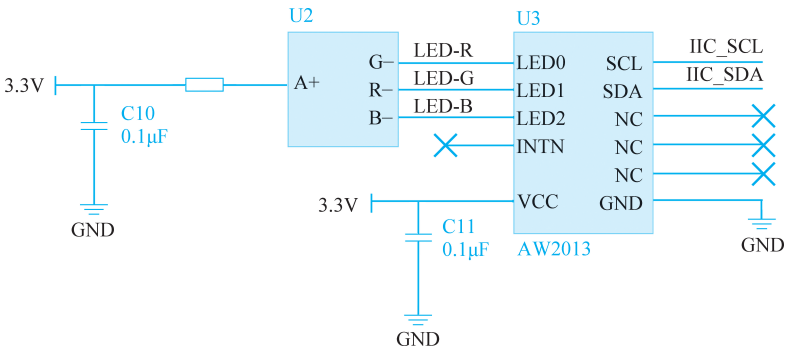

图 5.9 三色 LED 灯原理图

表 5.1 AW2013 寄存器功能与含义

地 址	寄存器名称	功 能	默认值
00h	软件复位寄存器 RSTR	软复位控制	33h
01h	全局控制寄存器 GCR	设置芯片使能和中断使能	00h
02h	中断状态寄存器 ISR	读取中断状态	00h
30h	LED 控制寄存器 LCTR	分通道控制 LED 输出使能	00h
31h～33h	LED 模式配置寄存器 LCFG	设置 LED0～LED2 工作模式	00h
34h～36h	PWM 配置寄存器 PWMx	设置 LED0～2 的 PWM 亮度等级	00h
37/3A/3Dh	T1&T2 时间寄存器	设置一次编程模式的 T1/T2 时间	00h
38/3B/3Eh	T3&T4 时间寄存器	设置一次编程模式的 T3/T4 时间	00h
39/3C/3Fh	T0 和重复次数配置寄存器	设置 T0 时间和呼吸次数	00h
77h	I²C 地址配置寄存器 IADR	修改 I²C 接口的设备地址	45h

首先初始化一些外设驱动。

```
PCF8574_Init();
```

然后进行三色 LED 灯的初始化。

```
AW2013_Init();                    // 三色 LED 灯的初始化
```

在三色 LED 灯的初始化中，首先将 GPIO9、GPIO10 复用为 I²C 的 SCL 和 SDA。

```
//GPIO_9 复用为 I2C_SCL
hi_io_set_pull(HI_IO_NAME_GPIO_9,HI_IO_PULL_UP);
hi_io_set_func(HI_IO_NAME_GPIO_9,HI_IO_FUNC_GPIO_9_I2C0_SCL);
//GPIO_10 复用为 I2C_SDA
hi_io_set_pull(HI_IO_NAME_GPIO_10,HI_IO_PULL_UP);
hi_io_set_func(HI_IO_NAME_GPIO_10,HI_IO_FUNC_GPIO_10_I2C0_SDA);
```

通过 hi_i2c_init 函数设立 I²C 的从机地址，总线号，速度。从机地址为 0×8A，总线号为 I²C0，速度 100kHz。

```
result =hi_i2c_init(AW2013_I2C_IDX,AW2013_I2C_SPEED);
if (result !=HI_ERR_SUCCESS)
{
```

```
        printf("I2C aw2013 Init status is 0x%x!!!\r\n",result);
        return result;
}
```

利用 I²C 复位芯片，向 RSTR_REG_ADDR 寄存器写入 0x55。

```
//复位芯片
result =aw2013_witeByte(RSTR_REG_ADDR,0x55);
if (result !=HI_ERR_SUCCESS)
{
    printf("I2C aw2013 RSTR_REG_ADDR status =0x%x!!!\r\n",result);
    return result;
}

usleep(TIME_RESET);
```

设置使能全局控制器，向 GCR_REG_ADDR 寄存器写入 0x01。

```
result =aw2013_witeByte(GCR_REG_ADDR,0x01);
    if (result !=HI_ERR_SUCCESS)
    {
        printf("I2C aw2013 GCR_REG_ADDR status =0x%x!!!\r\n",result);
        return result;
    }
```

设置打开 RGB 三路通道，向 LCTR_REG_ADDR 寄存器写入 0x07。

```
result =aw2013_witeByte(LCTR_REG_ADDR,0x07);
if (result !=HI_ERR_SUCCESS)
{
    printf("I2C aw2013 GCR_REG_ADDR status =0x%x!!!\r\n",result);
    return result;
}
```

设置 RGB 三路通道的工作模式 LCFG0_REG_ADDR、LCFG1_REG_ADDR、LCFG2_REG_ADDR，分别对应 RGB 灯的三个颜色。

```
result =aw2013_witeByte(LCFG0_REG_ADDR,0x63);
if (result !=HI_ERR_SUCCESS)
{
    printf ("I2C aw2013 GCR_REG_ADDR status =0x%x!!!\r\n",result);
    return result;
}
result =aw2013_witeByte(LCFG1_REG_ADDR,0x63);
if (result !=HI_ERR_SUCCESS){
    printf("I2C aw2013 GCR_REG_ADDR status =0x%x!!!\r\n",result);
    return result;
}
result =aw2013_witeByte(LCFG2_REG_ADDR,0x63);
if (result !=HI_ERR_SUCCESS)
{
        printf("I2C aw2013 GCR_REG_ADDR status =0x%x!!!\r\n",result);
        return result;
}
printf("I2C aw2013 Init is succeeded!!!\r\n");
```

```
return HI_ERR_SUCCESS;
```

在前置代码中关闭三色灯。

```
AW2013_Control_Red(RGB_OFF);
AW2013_Control_Green(RGB_OFF);
AW2013_Control_Blue(RGB_OFF);
```

在任务 1 中首先打开三色灯，而后关闭。点亮 LED 灯则 RGB_ON 为 255，关闭 LED 灯则 RGB_OFF 为 0。

```
void Task1(void)
{
        while (1)
    {
        AW2013_Control_Red(RGB_ON);
        AW2013_Control_Green(RGB_ON);
        AW2013_Control_Blue(RGB_ON);
        Sleep(TASK_DELAY_TIME);

        AW2013_Control_Red(RGB_OFF);
        AW2013_Control_Green(RGB_OFF);
        AW2013_Control_Blue(RGB_OFF);
        Sleep(TASK_DELAY_TIME);
    }
}
```

图 5.10 为 RGB 三色灯点亮效果图。

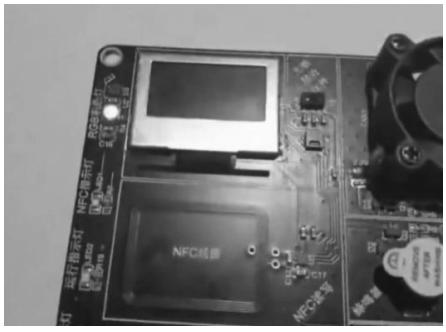

图 5.10 RGB 三色灯点亮效果图

5.4 脉冲宽度调制

5.4.1 概述

脉冲宽度调制(PWM)，是英文 Pulse Width Modulation 的缩写，简称脉宽调制，是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中。PWM 模块用于生成 PWM 信号，调节灯的亮度等功能。

PWM 的频率是指 1 秒内信号从高电平到低电平再回到高电平的次数(一个周期)，

即说一秒内 PWM 有多少个周期，单位为 Hz。

PWM 的周期 $T=1/f$，T 为周期，f 为频率。比如 50Hz = 1/20ms 一个周期，即如果频率为 50Hz，也就是说一个周期是 20ms，那么一秒就有 50 次 PWM 周期。

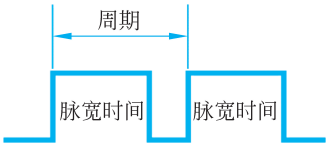

图 5.11　周期示意图

占空比是一个脉冲周期内，高电平的时间与整个周期时间的比例。周期是一个脉冲信号的时间。脉宽时间为高电平时间。如图 5.11 所示，脉宽时间占总周期时间的比例为占空比。

5.4.2　功能描述

PWM 模式具有以下功能特点：

(1) 共集成 6 路 PWM；

(2) 每一路的 PWM 时钟可以单独门控；

(3) 可对晶体时钟(40/24MHz)或 160MHz 时钟进行 1～65535 倍分频；

(4) PWM 信号占空比可调整范围：1～1/65535。

5.4.3　工作方式

PWM 模块有两个时钟可以选择：晶体时钟(40/24MHz)、160MHz。

以配置 PWM 信号对 160MHz 时钟分频、分频倍数为 25 倍、占空比为 1/5 为例，配置步骤如下：

(1) 写 CLK_SEL[pwm_clk_sel]为 0，选择 PWM 时钟为 160MHz；

(2) 写 PWM_EN[pwm_en]为 1，使能 PWM 信号输出；

(3) 写 PWM_FREQ[pwm_freq]为 0x19，确定对 PWM 时钟的分频倍数为 25；

(4) 写 PWM_DUTY[pwm_duty]为 0x5，确定 PWM 信号的占空比为 PWM_DUTY[pwm_duty]/PWM_FREQ[pwm_freq] = 1/5；

(5) 写 PWM_START[pwm_start]为 1，使能步骤(2)～步骤(4)的配置。

5.4.4　寄存器概览

如表 5.2 所示为 PWM 寄存器概览，其中 PWM0 基址是 0x4004_0000，PWM1 基址是 0x4004_0100，PWM2 基址是 0x4004_0200，PWM3 基址是 0x4004_0300，PWM4 基址是 0x4004_0400，PWM5 基址是 0x4004_0500。

表 5.2　PWM 寄存器概览

偏 移 地 址	名　　称	描　　述
0x00	PWM_EN	PWM 使能寄存器
0x04	PWM_START	PWM 配置生效寄存器
0x08	PWM_FREQ	PWM 频率控制计数值寄存器
0x0C	PWM_DUTY	PWM 占空比计数值寄存器

5.4.5 寄存器描述

1. PWM_EN

PWM_EN(如表 5.3 所示)为 PWM 使能寄存器,偏移地址为 0x00,总复位值为 0x0000_0001。

表 5.3 PWM_EN 描述

Bits	Access	名 称	描 述	复 位
[31:1]	—	reserved	保留	0x00000000
[0]	RW	pwm_en	PWM 功能使能 0:禁止,pwm_out 输出持续为 0; 1:使能	0x1

2. PWM_START

PWM_START(如表 5.4 所示)为 PWM 配置生效寄存器,偏移地址为 0x04,总复位值为 0x0000_0000。

表 5.4 PWM_START 描述

Bits	Access	名 称	描 述	复 位
[31:1]	—	reserved	保留	0x00000000
[0]	RW	pwm_start	PWM 配置生效寄存器 0:无操作; 1:此前赋值的 PWM 相关寄存器生效,逻辑自清零	0x0

3. PWM_FREQ

PWM_FREQ(如表 5.5 所示)为 PWM 频率控制计数值寄存器,偏移地址为 0x08,总复位值为 0x0000_05DC。

表 5.5 PWM_FREQ 描述

Bits	Access	名 称	描 述	复 位
[31:16]	—	reserved	保留	0x0000
[15:0]	RW	pwm_freq	PWM 频率控制计数值,即对 PWM 时钟的分频倍数,取值范围 1～65535	0x05DC

4. PWM_DUTY

PWM_DUTY(如表 5.6 所示)为 PWM 占空比计数值寄存器,偏移地址为 0x0C,总复位值为 0x0000_02EE。