STM32 高教高教板

实验指导书

线上商城：立创商城 https://list.szlcsc.com/catalog/953.html

立创开源论坛：https://oshwhub.com/

EDA 软件：https://lceda.cn

技术支持 QQ：715844371

目录

引言

Cortex-M3 采用 ARM V7 构架，不仅支持 Thumb-2 指令集，而且拥有很多新特性。较之 ARM7 TDMI，Cortex-M3 拥有更强劲的性能、更高的代码密度、位带操作、可嵌套中断、低成 本、低功耗等众多优势。

STM32 的优异性体现在如下几个方面：

1、超低的价格。以 8 位机的价格，得到 32 位机，是 STM32 最大的优势。

2、超多的外设。STM32 拥有包括：FSMC、TIMER、SPI、IIC、USB、CAN、IIS、SDIO、ADC、DAC、RTC、DMA 等众多外设及功能，具有极高的集成度。

3、丰富的型号。STM32 仅 M3 内核就拥有 F100、F101、F102、F103、F105、F107、F207、F217 等 8 个系列上百种型号，具有 QFN、LQFP、BGA 等封装可供选择。同时 STM32 还推出了 STM32L 和 STM32W 等超低功耗和无线应用型的 M3 芯片。

4、优异的实时性能。84 个中断，16 级可编程优先级，并且所有的引脚都可以作为中断输入。

5、杰出的功耗控制。STM32 各个外设都有自己的独立时钟开关，可以通过关闭相应外设的时钟来降低功耗。

6、极低的开发成本。STM32 的开发不需要昂贵的仿真器，只需要一个串口即可下载代码，并且支持 SWD 和 JTAG 两种调试口。SWD 调试可以为你的设计带来跟多的方便，只需要 2 个 IO 口，即可实现仿真调试。

学习 STM32 有两份不错的中文资料：

《STM32 参考手册》中文版 V10.0

《Cortex-M3 权威指南》中文版（宋岩译）

前者是 ST 官方针对 STM32 的一份通用参考资料，内容翔实，但是没有实例，也没有对 Cortex-M3 构架进行多少介绍，读者只能根据自己对书本的理解来编写相关代码。后者是专门介绍 Cortex-M3 构架的书，有简短的实例，但没有专门针对 STM32 的介绍。所以，在学习 STM32 的时候，必须结合这份资料来看。

STM32 拥有非常多的寄存器，对于新手来说，直接操作寄存器有很大的难度，所以 ST 官方提供了一套固件库函数，大家不需要再直接操作繁琐的寄存器，而是直接调用固件库函数即可实现操作寄存器的目的。

本指南将结合《STM32 参考手册》和《Cortex-M3 权威指南》两者的优点，并从固件库级别出发，深入浅出，向读者展示 STM32 的各种功能。总共配 18 个基础实例，基本上每个实例在均配有软硬件设计，先介绍硬件设计原理，后附上实例代码讲解，并带有详细注释及说明，让读者快速理解代码。

这些基础实例涵盖了 STM32 的大部分内部资源，大家只需下载程序到 STM32 高教高教板，即可验证实验。不管你是一个 STM32 初学者，还是一个老手，本指南都非常适合。尤其对于初学者，本指南将手把手的教你如何使用 MDK，包括新建工程、编译、仿真、下载调试等一系列步骤，让你轻松上手。本指南适用于想通过库函数学习 STM32 的读者，大家可以结合官方提供的库函数实例对照学习。

俗话说：人无完人。本指南也不例外，在编写过程中如遇到错误，还请告诉我们，我们的技术 QQ：715844371，也可以在立创技术论坛留言，在此先向各位朋友表示衷心的感谢。

STM32 简介

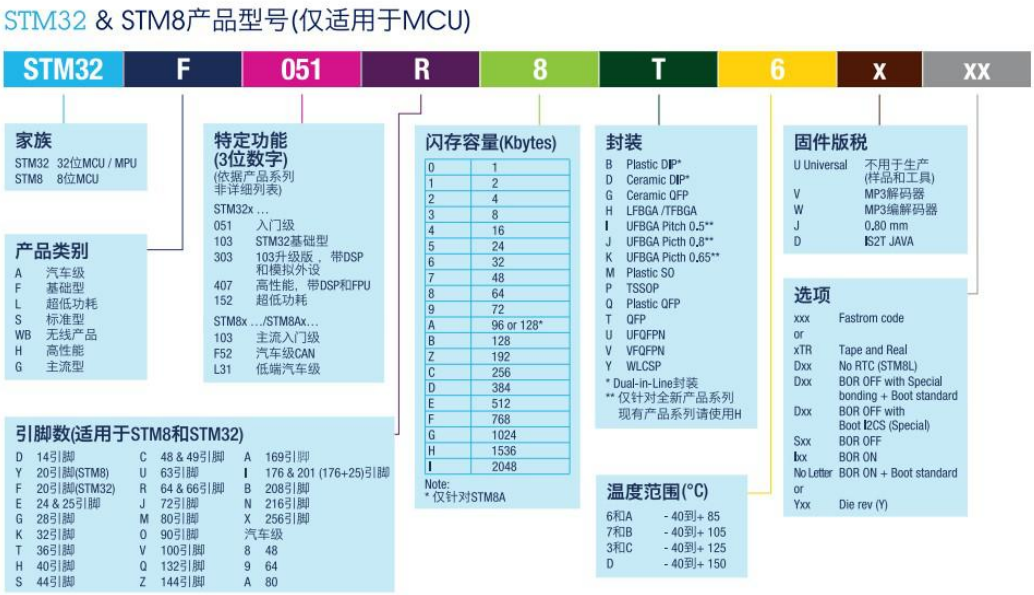
1. 什么是 STM32？

从名字上就可以看出，ST 为公司的名字，该公司是意法半导体，M 为 Microelectronics 的缩写，表示为处理器，32 为 32bit；这款产品就是意法半导体公司出品的 32 位微处理器。

2. STM32 能做什么？

我们可以在淘宝内进行 STM32 相关的搜索，我们搜索出许多与 STM32 许多相关的生活中常见的电子产品如：微型四轴无人机、智能手环、平衡车、机器人等。在新兴起的物联网中，STM32 也同样扮演着重要的角色。

3. STM32 的命名规则



4. STM32 高教高教板

JL-DB-GJB-V1.0 高教高教板，主要是引导学生从零开始学习主流的嵌入式单片机，并参与到电子竞赛、学习实验中去。着重培养学生的嵌入式单片机基础设计及基础程序编写，有助于培养电子兴趣爱好，提高教学质量，提高学生的动手能力，提高应届毕业生就业率等等。

第一部分 硬件篇

本高教高教板（以下简称高教板）拥有四部分：核心板部分、底板部分、TFT LCD 触摸显示屏、外置试验模块部分，高教板的特色为核心板与底板分离，底板拥有外置实验的接口与外置的试验模块，传统高教板都是整体型的，如果学生在做实验时不慎将高教板部分电路短路，将单片机或其他芯片烧坏，出现这样的情况时，传统的高教板故障对于初学着来说，维修是一个不小的难度与挑战，甚至会大大降低的学生的学习热情。本高教板采用全部模块化的设计理念，可对高教板大部分的电路模块单独更换，大大提高了可维护性及操作容易性，而且可更换的模块具有很高的性价比，价格低廉，极大的降低了实验板的维护成本。

第一章 高教板硬件组成

1.1 高教板之一 核心板

如图 1.1.1、1.1.2 所示：

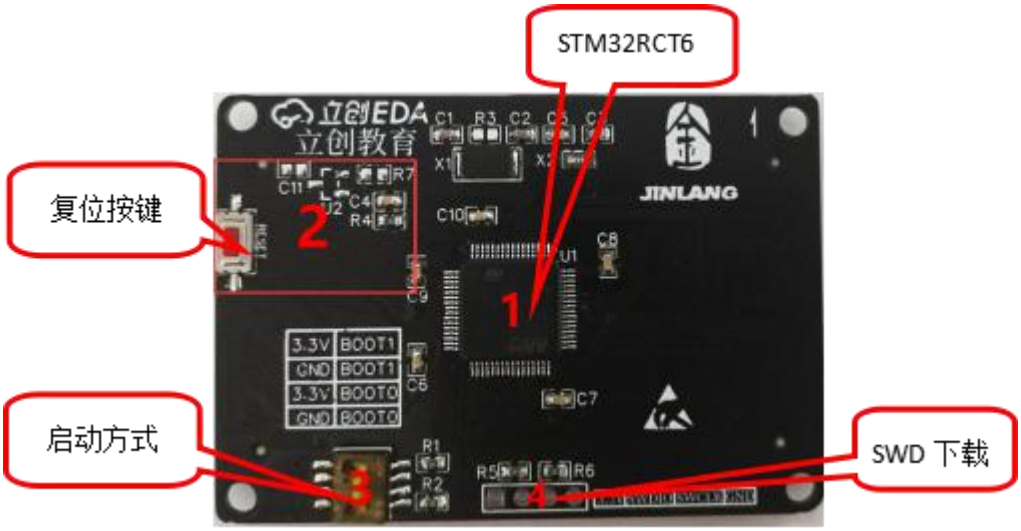


图 1.1.1 核心板正面

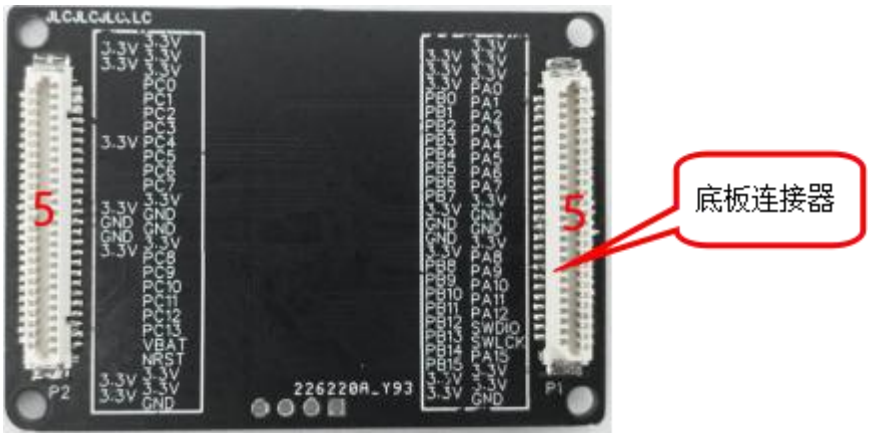


图 1.1.2 核心板背面

1.1.1 STM32 高教板核心板板载资源如下：

1. CPU:采用 STM32F103RCT6
2. 工作电压：2V~3.6V
3. Cortex-M3 架构内核的 32 位处理器
4. 封装：LQFP-64
5. 内置 256KB 的 Flash
6. 48K 的 RAM

7. 3x12 位 ADC
8. 2x12 位 DAC
9. 2 个 16 位基本定时器
10. 2 个 DMA 控制器
11. 3 个 SPI
12. 2 个 IIC
13. 3 路 USART 通讯口
14. 5 个串口
15. 1 个 USB
16. 1 个 CAN
17. 输入/输出数：51 个
18. 时钟频率最高可达 72MHz

1.1.2 复位

该部分为核心板板载的复位按键，当 STM32 系统需要复位时，可按下复位键，死机时可操作此按键。此按键还具有复位液晶的功能，因为液晶模块的复位引脚和 STM32 的复位引脚是连接在一起的，当按下该键的时候，液晶屏也一并被复位。

1.1.3 启动方式

在 ISP 下载电路中，我们需要配置 BOOT 引脚，有关 BOOT 引脚不同的配置会产生不同的启动方式，具体见表格 1.1.3.1 BOOT 配置。

BOOT0	BOOT1	启动方式	启动说明
0	X	内部 FLASH	用户闪存存储区，也就是 FLASH 启动
1	0	系统存储器	系统存储器启动，用于串口下载
1	1	内部 SRAM	SRAM 启动，用于 SRAM 中调试代码

表 1.1.3.1

1.1.4 SWD 下载

由于 STM32 支持 SWD 调试，该部分为 SWD 下载，该模式只需要 2 个 IO 口，节约 IO 口数量，当我们的进行实验时，如果占用的 IO 口过多，导致 JTAG

无法下载，我们选择 SWD 模式下载，JTAG 模式与 SWD 模式实现的功能是一致的。

1.2 高教板之二 底板

如图 1.2.1、1.2.2 所示：

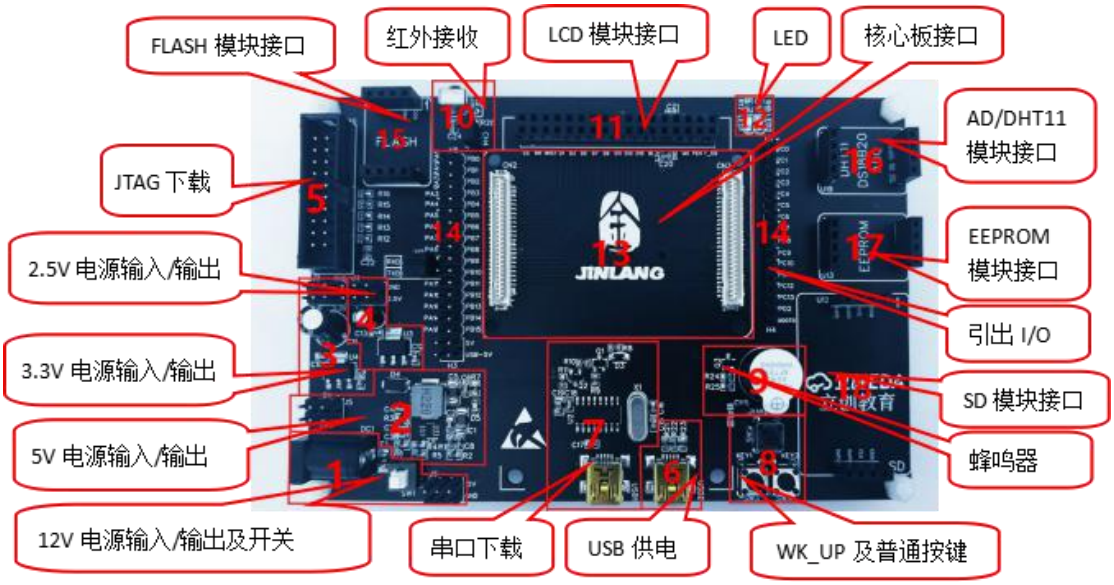


图 1.2.1 底板正面

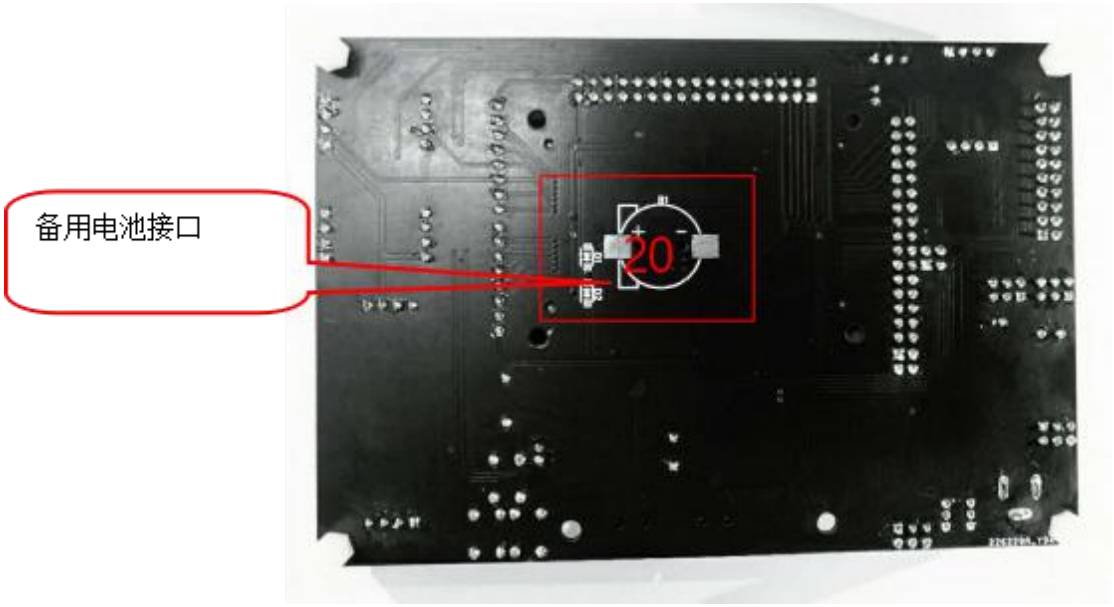


图 1.2.2 底板背面

底板的外形尺寸为 13.4cm*89.9cm 大小，板子的设计充分考虑了人性化设计，采用全模块化设计思路，经过反复修改，最终确了定这样的设计。

STM32 高教高教板的特点包括：

1）接口丰富。板子提供多种标准接口，可以方便的进行各种外设的实验和开发。
2）设计灵活。板上很多资源都可以灵活配置，以满足不同条件下的使用。我们引出了除晶振占用的 IO 口外的所有 IO 口，可以极大的方便大家扩展及使用。
3）人性化设计。各个接口都有丝印标注，使用起来一目了然；接口位置设计安排合理，方便顺手。

1.3 高教板之三 TFT LCD 2.8 寸显示屏/触摸屏

该显示屏如图 1.3.1、1.3.2 所示，采用 2.8 寸的 TFT 电阻触摸显示屏，电阻触摸屏的主要部分是一块与显示器表面非常配合的电阻薄膜屏，这是一种多层的复合 薄膜，它以一层玻璃或硬塑料平板作为基层，表面涂有一层透明氧化金属（透明的导电电阻）导电层，上面再盖有一层外表面硬化处理、光滑防擦的塑料层、它的内表面也涂有一层涂层、在他们之间有许多细小的（小于 1/1000 英寸）的透明隔离点把两层导电层隔开绝缘。当手指触摸屏幕时，两层导电层在触摸点位置就有了接触，电阻发生变化，在 X 和 Y 两个方向上产生 信号，然后送触摸屏控制器。控制器侦测到这一接触并计算出（X，Y）的位置，再根据获得的位置模拟鼠标的方式运作。这就是电阻技术触摸屏的最基本的原理。

触摸屏控制芯片为 XPT2046 如图 1.3.2 所示的 U3 芯片。XPT2046 是一款 4 导线制触摸屏控制器，内含 12 位分辨率 125KHz 转换速率逐步逼近型 A/D 转换器。XPT2046 支持从 1.5V 到 5.25V 的低电压 I/O 接口。XPT2046 能通过执行两次 A/D 转换查出被按的屏幕位置，除此之外，还可以测量加在触摸屏上的压力内部自带 2.5V 参考电压可以作为辅助输入、温度测量和电池监测模式之用，电池监测的电压范围可以从 0V 到 6V。XPT2046 片内集成有一个温度传感器。在 2.7V 的典型工作状态下，关闭参考电压，功耗可小于 0.75mW。该屏幕采用的 XPT2046 采用封装形式：TSSOP-16，工作温度范围为-40℃～+85℃。

图 1.3.1 寸 TFTLCD 正面

图 1.3.2 寸 TFTLCD 背面

1.4 高教板之四 基础外设模块

1.4.1 FLASH W25Q16 模块

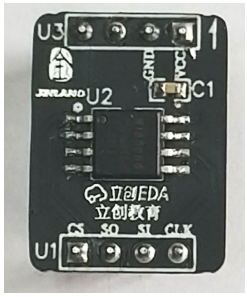


图 1.4.1 FLASH 模块

1.4.2 DS18b20/DHT11/ADC 模块

图 1.4.2 DS18B20/DHT11/ADC 模块

1.4.3 EEPROM 24C02 模块

图 1.4.3 EEPROM 模块

1.4.4 SD 卡模块

图 1.4.4 SD 卡模块

第二章 高教板原理图

2.1 电源

本高教板的电源部分有三部分，第一部分为板载一个外部电源输入口，如图 2.1.1 所示，采用标准的内径 2.5MM 直流电源插座，我们可以输入 9V-12V 的电压，我们在电源处板载一个保险丝，当某部分发生短路时电流增大从而使保险丝熔断，保护电路的安全。

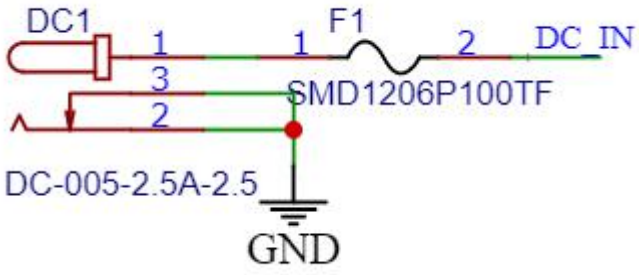


图 2.1.1 电源输入

第二部分为板载的电源开关，用于控制整个高教板的供电，如果切断，则整个高教板都将断电。

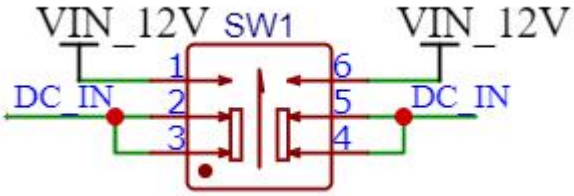


图 2.1.2 电源开关

第三部分为电源电压输出接口，该部分可以将电源输入的 12V 电压进行外部供电，也可以从外部输入 12V 给实验板进行供电。

2.2 5V 稳压电路

该电路采用 MP4420 为主的稳压电路如图 2.2 所示，该芯片的特点为可输出的电压为 0.8V-27V，输出电流为 2A，最小输入电压 4V，最大输入电压 30V。

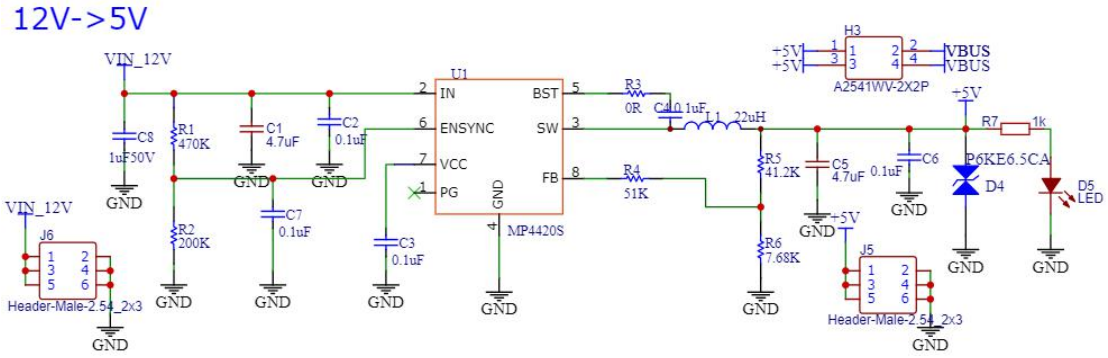


图 2.2 5V 稳压电路

2.3 3.3V 稳压电路

该电路采用 AMS117-3.3V 的稳压芯片组成的电路如图 2.3 所示，该电路为单片机及一些模块进行供电，如：STM32F103RCT6 核心板、LCD 屏幕、LED、JTAG 下载电路、红外接收头、蜂鸣器、E2PROM 外置模块、DHT11/DS18B20/ADC 通用外置模块、FLASH 外置模块、SD 外置模块、同时该稳压电路进行了 3.3V 电压的引出，该该接口可以输出 3.3V 的电压进行外部供电，也可以从外部输入 3.3V 给实验板进行供电。

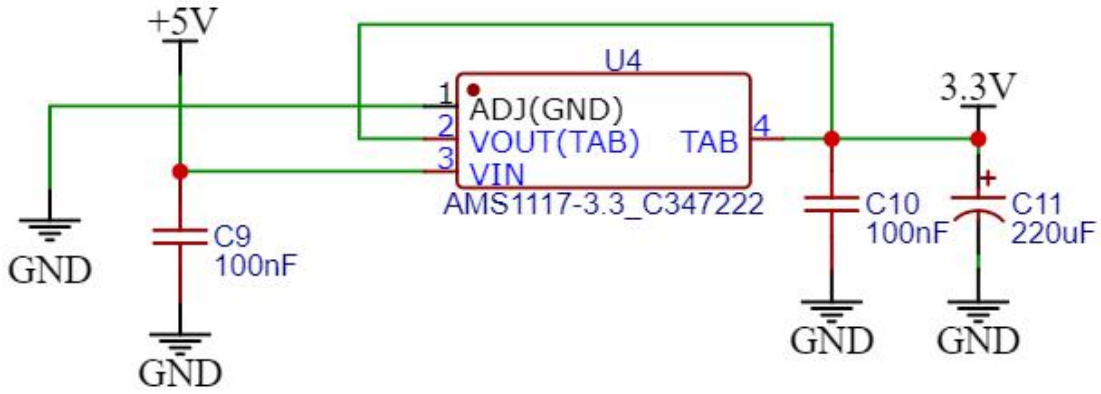


图 2.3 3.3V 稳压电路

2.4 2.5V 稳压电路

该电路采用 AMS117-2.5V 的稳压芯片组成的电路如图 2.4 所示，该稳压电路进行了 2.5V 电压的引出。

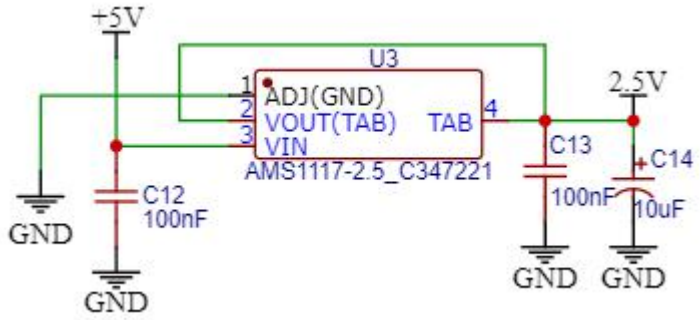


图 2.4 2.5V 稳压电路

2.5 JTAG 下载

该下载口采用板载的 20 针标准 JTAG 调试口（JTAG）如图 2.5 所示， 该接口直接可以和 ULINK、 JLINK 或者 STLINK 等调试器（仿真器）连接。

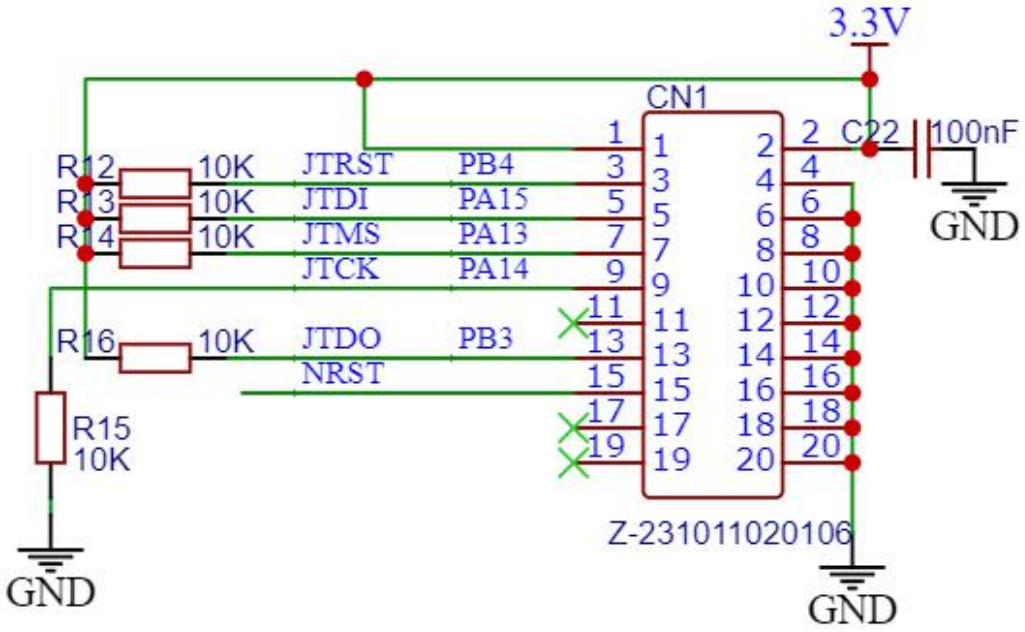


图 2.5 JTAG 下载电路

2.6 USB 供电接口

该部分为板载的 USB 接口如图 2.6 所示，可以为整个电路板供电。

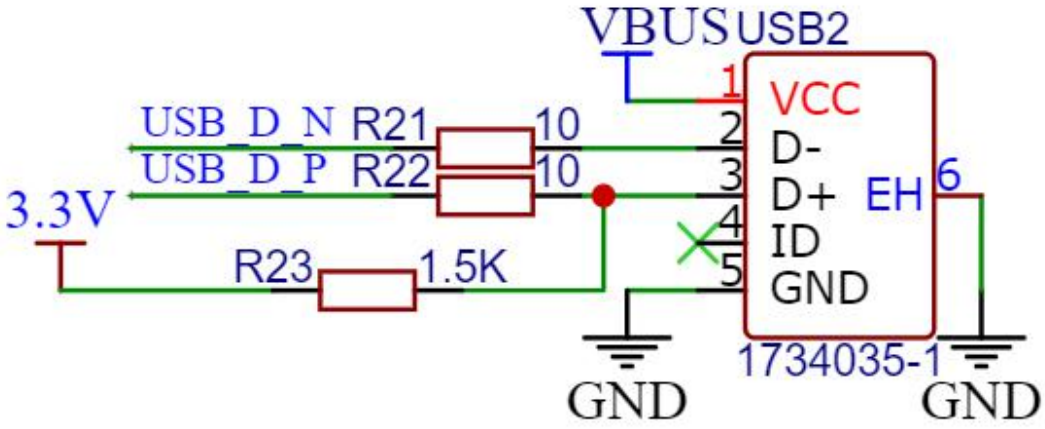


图 2.6 USB 供电

2.7 USB 转串口

将 USB 与 STM32 串口进行转换如图 2.7 所示，该部分采用的芯片为 CH340。配合上位机使用，可实现自动下载功能。

该芯片为板载的 CH340 芯片，为利用 USART 实现高教板与电脑之间的通信，这时我们需要一个 USB 转 USART 的 IC，我们选择 CH340 芯片来实现这个功能，该芯片是一款 USB 总线的转接芯片，实现 USB 转 USART， CH340 的 TXD 引脚与 USART1 的 RX 引脚连接，CH340 的 RXD 引脚与 USART1 的 TX 引脚连接。

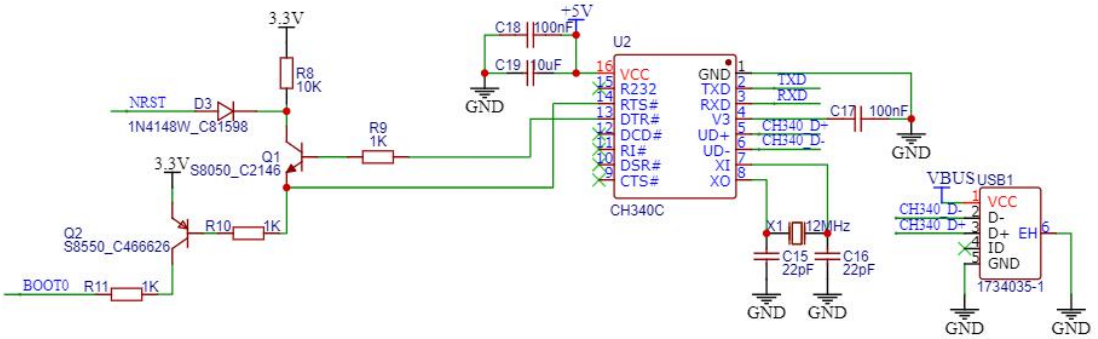


图 2.7 USB 转串口

2.8 按键部分

该高教板板载三个按键如图 2.8 所示，其中 KEY1、KEY2 为普通按键连接在 I/O 口上，可用于人机交互，其中 WK_UP 为唤醒单机状态下的单片机连接在 PA0

上；这里注意 WK_UP 是高电平有效，而 KEY1 和 KEY2 是低电平有效。

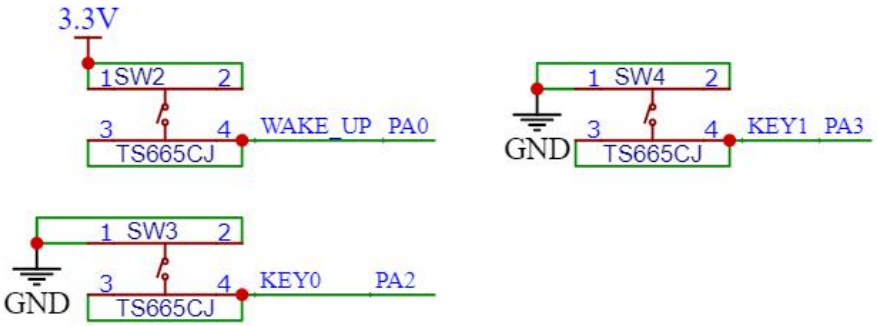


图 2.8 按键

2.9 蜂鸣器

该部分为板载的有源蜂鸣器电路如图 2.9 所示，用于蜂鸣器相关的实验。

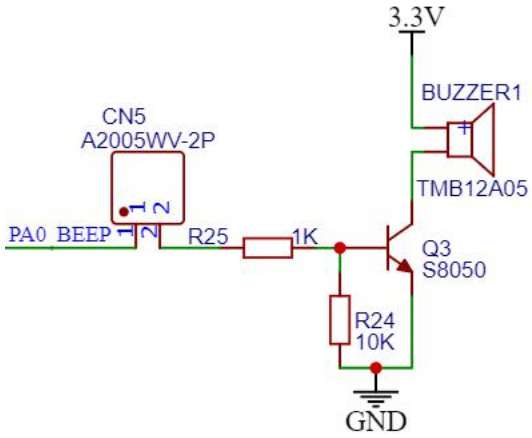


图 2.9 蜂鸣器

2.10 红外接收

这是高教板的红外接收电路如图 2.10 所示，可以实现红外遥控功能。

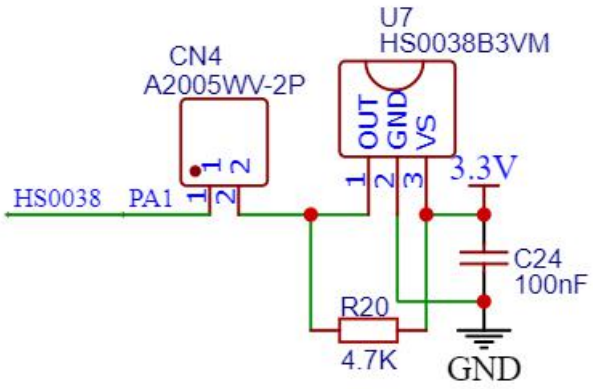


图 2.10 红外接收

2.11 TFT LCD 液晶屏接口

该屏幕接口该接口位于高教板的偏顶端电路如图 2.11，兼容 2.4/2.8 寸 LCD 屏幕模块。

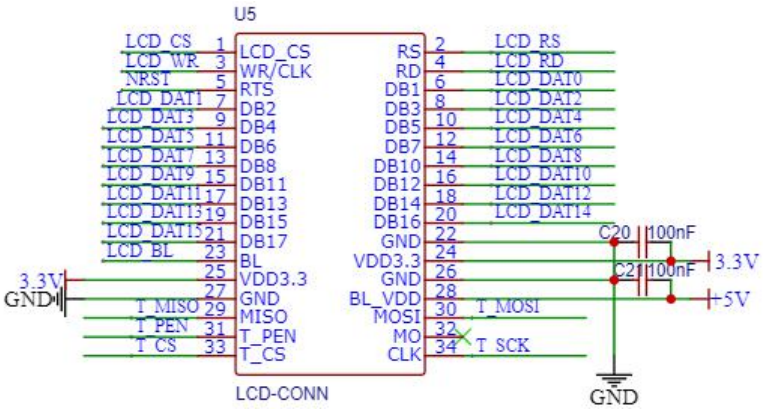


图 2.11 LCD 屏幕接口

2.12 LED 流水灯

该电路板板载 2 个 LED 灯电路如图 1.12 所示，可用于做 LED 灯的相关实验。

图 2.12 LED

2.13 核心板连接器

该部分为 STM32F103RB6 核心板的接口电路如图 2.13 所示，接口处方向的标识防止核心板插反，该部分的优点是便于更换核心板，便于维修。

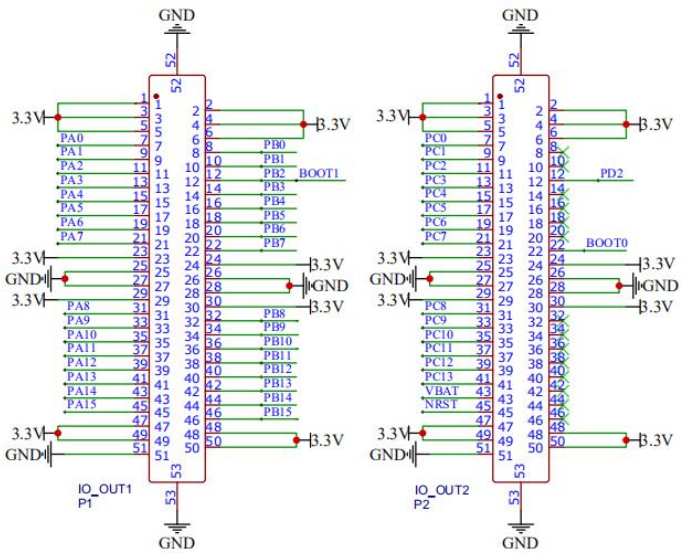


图 2.13 核心板连接器

2.14 引出 I/O 口

这里将 PA0~PA15、PB0~PB15、PC0~PC13 电路如图 2.14 所示。

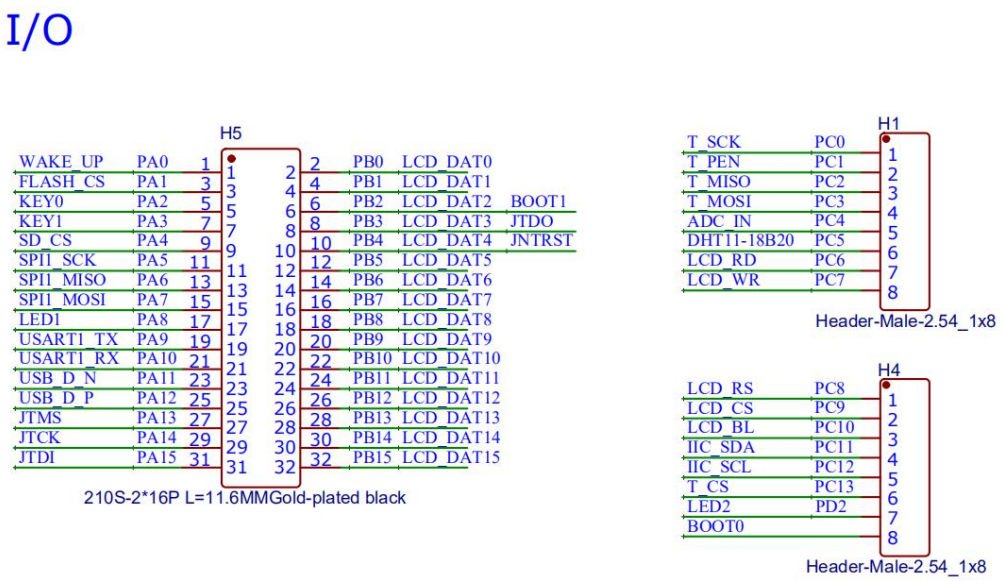
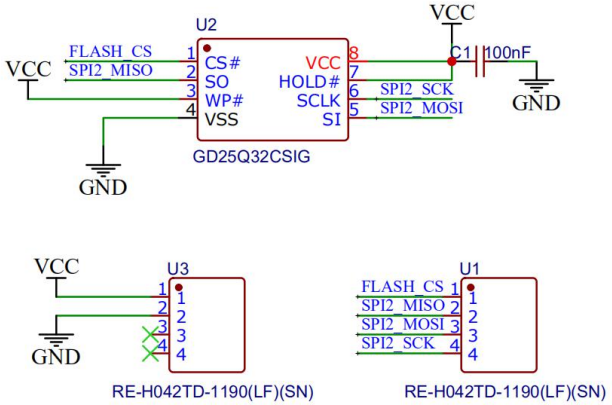


图 2.14 引出 I/O

2.15 外置 FLASH

外置的实验接口，需要与 FLASH 模块配合使用，如图 2.15.1、2.15.2 所示，在使用时该模块有方向标识要与底板上的标识一致，该模块的优点在于便于更换与维修。



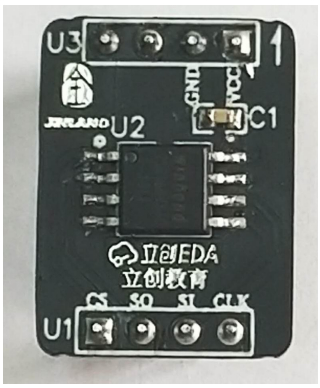


图 2.15.1 模块原理图

图 2.15.2 实物图

2.16 外置 DHT11/DS18B20/ADC 通用接口

外置的实验接口，这是一个复用的实验接口，如图 2.16.1、2.16.2 所示，需要与 DHT11/DS18B20/ADC 模块配合使用，该部分可以做测量温、温湿度及 ADC 的实验，我们在使用时，该模块有方向标识要与底板上的标识一致，该模块的优点在于便于更换与维修。

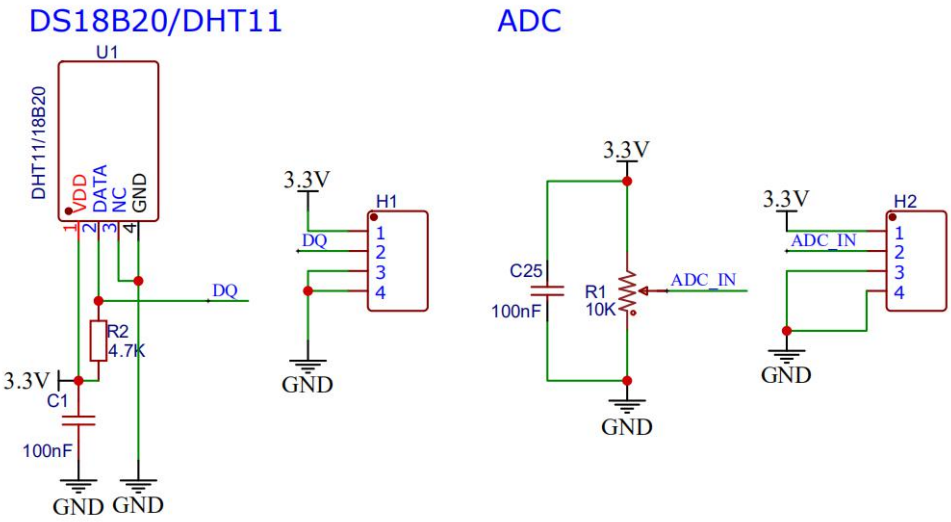


图 2.16.1 DHT11/DS18B20/ADC 模块原理图

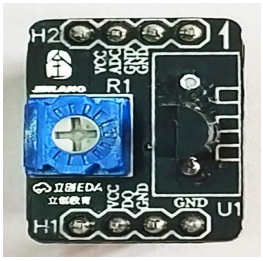


图 2.16.2 实物图

2.17 外置 EEPROM 接口

拥有外置的实验接口，需要与 EEPROM 模块配合使用，如图 2.17.1、2.17.2 所示，该 EEPROM 芯片为 24C02，容量为 2Kb，也就是 256 字节，用于存储一些掉电不能丢失的重要数据，有了它就可以方便的实现数据保存，在使用时该模块有方向标识要与底板上的标识一致，该模块的优点在便于更换与维修。

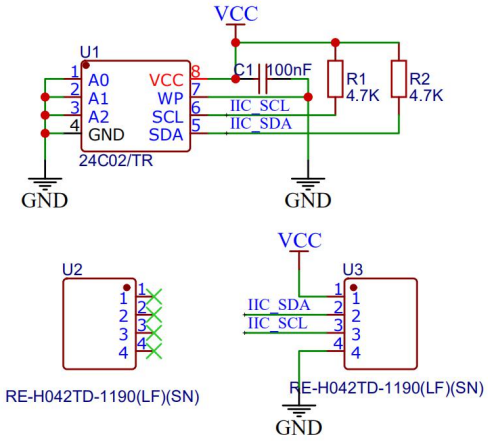


图 2.17.1 EEPROM 模块原理图

图 2.17.2 实物图

2.18 外置 SD 接口

外置的实验接口，需要与 SD 模块配合使用，如图 2.18.1、2.18.2 SD 所示，该模块是一个标准 SD 卡接口（SD_CARD）采用大 SD 卡接口（即相机卡，也可以是 TF 卡+卡套的形式），SDIO 方式驱动，有了这个 SD 卡接口，就可以满足海量数据存储的需求；在使用时该模块有方向标识要与底板上的标识一致，该模块的优点在于便于更换与维修。

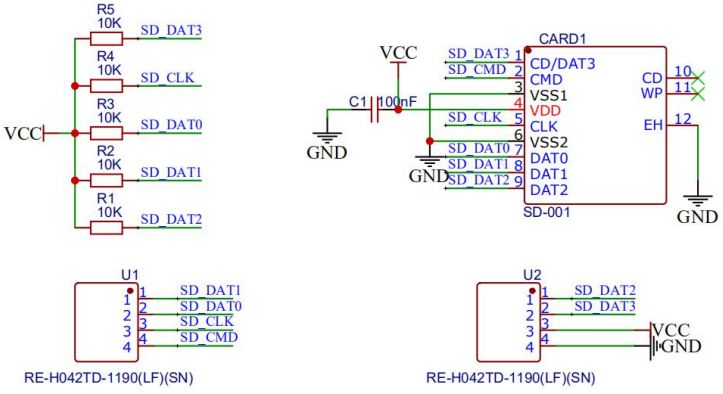


图 2.18.1 SD 模块原理图

图 2.18.2 实物图

第二部分 软件篇

实验篇

第三章 流水灯实验

3.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 流水灯的编程

3.2 实验原理

单片机流水的实质是单片机各引脚在规定的时间逐个上电，使 LED 灯能逐个亮起来但过了该引脚通电的时间后便灭灯的过程，实验中使用了单片机的 PA8、PD2 端口，对 2 个 LED 灯进行控制，要实现逐个亮灯即将 PA8、PD2 的端口逐一置零，中间使用时间间隔隔开各灯的亮灭。

3.3 实验例程

（参考开源实验例程）实验一 流水灯

3.4 实验程序配置解析

```
#include "stm32f10x.h"


#define LED1_ON()  {GPIO_ResetBits(GPIOA, GPIO_Pin_8);}
//定义亮点 LED1    PA8
#define LED1_OFF() {GPIO_SetBits(GPIOA, GPIO_Pin_8);}
//定义熄灭 LED1    PA8
#define LED2_ON()  {GPIO_ResetBits(GPIOD, GPIO_Pin_2);}
//定义亮点 LED2    PD2
#define LED2_OFF() {GPIO_SetBits(GPIOD, GPIO_Pin_2);}
//定义熄灭 LED2    PD2


/*****************************************************************
* 函数名称  : Delay
* 功能介绍 ： 短延时
```

```
* 输      入 : nCount  延时时间
    * 输      出 : 无
    * 返 回 值 : 无
*******************************************************************************/
void Delay(__IO uint32_t nCount)
{
    for(; nCount !=0;nCount--);
}

/*******************************************************************
    * 函数名称  :main.c
    * 功能介绍  :主函数
    * 输      入 :无
    * 输      出 :无
    * 返 回 值 : 无
*******************************************************************/
int main()
{
    SystemInit();                    //系统始终初始化 72M
    {
        GPIO_InitTypeDef GPIO_InitStructure;
        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA                    |
RCC_APB2Periph_GPIOD, ENABLE ); //开启 GPIO 时钟

        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;          //LED1
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
//输出模式配置
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //输出频率
        GPIO_Init(GPIOA, &GPIO_InitStructure);
```

```
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2;              //LED2
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;   //输出模式配置
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;   //输出频率
    GPIO_Init(GPIOD, &GPIO_InitStructure);

    GPIO_SetBits(GPIOA, GPIO_Pin_8 );                      //熄灭 LED1
    GPIO_SetBits(GPIOD, GPIO_Pin_2 );                      //关闭 LED2
}
while (1)
{
    uint8_t i,j;
    LED1_ON();                  //点亮 LED1
    Delay(0xCFFFF);             //短延时
    LED1_OFF();                 //熄灭 LED1
    Delay(0xCFFFF);             //短延时
    LED2_ON();                  //点亮 LED2
    Delay(0xCFFFF);             //短延时
    LED2_OFF();                 //熄灭 LED2
    Delay(0xCFFFF);             //短延时
    for(i = 0;i < 3;i++)
    {
        for(j = 2;j > 0;j--)
        {
            LED1_ON();                  //点亮 LED1
            LED2_ON();                  //点亮 LED2
            Delay(0xCFFFF);             //短延时
            LED1_OFF();                 //熄灭 LED1
            LED2_OFF();                 //熄灭 LED2
```

```
                Delay(0xCFFFF);                    //短延时
            }
        }
    }
}
```

3.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果，我们会看到 LED1,LED2 轮流点亮。

第四章 蜂鸣器实验

4.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 让蜂鸣器发声的编程

4.2 实验原理

在前面我们已经对 STM32 的 GPIO 做了简单介绍，并且还使用了其中 IO 口直接控制高教板上的 LED。对于本章要实现蜂鸣器的控制，我们能否直接使用 STM32 的 IO 口驱动呢？根据 STM32F103 芯片数据手册可知， 单个 IO 口的最大输出电流是 25mA，而蜂鸣器的驱动电流是 30mA 左右，两者非常接近，有的朋友就想直接用 IO 口来驱动，但是有没有考虑到整个芯片的输出电流，整个芯片的输出电流最大也就 150mA，如果在驱动蜂鸣器上就耗掉了 30mA，那么 STM32 其他的 IO 口及外设电流就非常拮据了。所以我们不会直接使用 IO 口驱动蜂鸣器，而是通过三极管把电流放大后再驱动蜂鸣器，这样 STM32 的 IO 口只需要提供不到 1mA 的电流就可控制蜂鸣器。所以我们也经常说到 STM32 芯片是用来做控制的，而不是驱动。

4.3 实验例程

（参考开源实验例程）实验二 蜂鸣器

4.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "BEEP.h"

/*******************************************************************
* 函数名称 : Delay
* 功能介绍 : 短延时
* 输 入 : nCount 延时时间
* 输 出 : 无
* 返 回 值 : 无
```

```
**********************************************************************/
void Delay(__IO uint32_t nCount)
{
    for(; nCount !=0;nCount--);
}

/**********************************************************************
    * 函数名称 :main.c
    * 功能介绍 :主函数
    * 输      入 :无
    * 输      出 :无
    * 返 回 值 : 无
**********************************************************************/
int main()
{
    SystemInit();                    //系统始终初始化 72M
    BEEP_Init();                     //蜂鸣器 GPIP 初始化配置（具体配置参
考例程）
    while (1)                        //主程序
    {
        BEEP_ON();                   //蜂鸣器发声
        Delay(0x80ffff);             //短延时
        BEEP_OFF();                  //关闭蜂鸣器
        Delay(0x60ffff);             //短延时
        BEEP_ON();                   //蜂鸣器发声
        Delay(0x10ffff);             //短延时
        BEEP_OFF();                  //关闭蜂鸣器
        Delay(0x10ffff);             //短延时
        BEEP_ON();                   //蜂鸣器发声
```

```
        Delay(0x10ffff);                //短延时
        BEEP_OFF();                     //关闭蜂鸣器
        Delay(0x40ffff);                //短延时
    }
}
```

4.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果，通过改变延时时间，蜂鸣器发出不同频率的声响。

第五章 按键点亮 LED 实验

5.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 的按键点亮 LED 的编程

5.2 实验原理

按键点亮 LED 实质是单片机 IO 口获取按键信号以后，点亮某个或者多个 LED 灯的过程，实验中使用了单片机的 PA8、PD2、PA2、PA3 端口，用两个按键分别控制两个 LED 灯亮灭。

5.3 实验例程

（参考开源实验例程）实验三 按键点亮 LED

5.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "KEY.h"
#include "DELAY.h"

/*************************************************************
    * 函数名称 :main.c
    * 功能介绍 :主函数
    * 输    入 :无
    * 输    出 :无
    * 返 回 值 : 无
*************************************************************/
int main(void)
{
    SystemInit();              //系统时钟配置初始化 72M
    GPIO_Config();             //LED、按键 GPIO 初始化配置
    while(1)
```

```
    {
        key_input();                    //检测是否有按键按下并有相应的 LED 点亮熄灭
    }
}
```

5.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果：KEY1 按下，LED1 先亮后灭，KEY2 按下，LED2 先亮后灭。

第六章 中断控制(中断方式点亮 LED)实验

6.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 按键中断方式点亮 LED 的编程

6.2 实验原理

按键点亮 LED 实质是单片机 IO 口获取按键信号以后，点亮某个或者多个 LED 灯的过程，实验中使用了单片机的 PA8、PD2、PA2、PA3 端口，用两个按键（采用中断方式）分别控制两个 LED 灯亮灭。

6.3 实验例程

（参考开源实验例程）实验三 按键点亮 LED

6.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "KEY.h"
#include "GPIO.h"
#include "exti.h"
#include "Delay.h"
/**************************************************************
 * 函数名称 :main.c
 * 功能介绍 :主函数
 * 输    入 :无
 * 输    出 :无
 * 返 回 值 : 无
 **************************************************************/
int main(void)
{
    SystemInit();                   //系统时钟配置初始化 72M
    GPIO_Config();                  //LED、按键 GPIO 初始化配置
    EXTI_PA_Config();               //按键中断配置初始化
```

```
    NVIC_Configuration();          //中断向量配置初始化
     while(1)                      //等待中断产生
     {


     }
}
```

6.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果：KEY1 按下，LED1、LED2 被点亮，KEY2 按下，LED1、LED2 熄灭。

第七章 串口通讯实验

7.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 的串口的编程

7.2 实验原理

串口是一种非常通用的设备通信的协议（不要与通用串行总线 Universal Serial Bus(USB)混淆）。大多数计算机包括两个基于 RS232 的串口。串口同一时候也是仪器仪表设备通用的通信协议；非常多 GPIB 兼容的设备也带有 RS-232 口。同一时候，串口通信协议也能够用于获取远程採集设备的数据。

串口通信的概念非常 easy，串口按位（bit）发送和接收字节。

虽然比按字节（byte）的并行通信慢，可是串口可以在使用一根线发送数据的同一时候用还有一根线接收数据。

它非常 easy 而且可以实现远距离通信。比方 IEEE488 定义并行通行状态时。规定设备线总长不得超过 20 米。而且随意两个设备间的长度不得超过 2 米；而对于串口而言。长度可达 1200 米。典型地。串口用于 ASCII 码字符的传输。通信使用 3 根线完毕：（1）地线，（2）发送，（3）接收。

因为串口通信是异步的，port 可以在一根线上发送数据同一时候在还有一根线上接收数据。其它线用于握手，但不是必须的。串口通信最重要的参数是波特率、数据位、停止位和奇偶校验。对于两个进行通行的 port，这些参数必须匹配：

波特率

这是一个衡量符号传输速率的参数。它表示每秒钟传送的符号的个数。比如 300 波特表示每秒钟发送 300 个符号。当我们提到时钟周期时，我们就是指波特率，比如假设协议须要 4800 波特率，那么时钟是 4800Hz。这意味着串口通信在数据线上的採样率为 4800Hz。通常电话线的波特率为 14400。28800 和 36600。波特率能够远远大于这些值，可是波特率和距离成反比。高波特率经常常使用于放置的非常近的仪器间的通信，典型的样例就是 GPIB 设备的通信。

数据位

这是衡量通信中实际数据位的参数。

当计算机发送一个信息包，实际的数据不会是 8 位的，标准的值是 5、6、7 和 8 位。怎样设置取决于你想传送的信息。比方。标准的 ASCII 码是 0～127（7 位）。扩展的 ASCII 码是 0～255（8 位）。

假设数据使用简单的文本（标准 ASCII 码），那么每一个数据包使用 7 位数据。每一个包是指一个字节，包含開始/停止位。数据位和奇偶校验位。因为实际数据位取决于通信协议的选取。术语“包”指不论什么通信的情况。

停止位

用于表示单个包的最后一位。

典型的值为 1，1.5 和 2 位。

因为数据是在传输线上定时的。而且每个设备有其自己的时钟，非常可能在通信中两台设备间出现了小小的不同步。

因此停止位不不过表示传输的结束，而且提供计算机校正时钟同步的机会。适用于停止位的位数越多。不同一时候钟同步的容忍程度越大，可是传输数据率同一时候也越慢。

奇偶校验位

在串口通信中一种简单的检错方式。有四种检错方式：偶、奇、高和低。当然没有校验位也是可以的。

对于偶和奇校验的情况，串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位。比如。假设数据是 011，那么对于偶校验，校验位为 0，保证逻辑高的位数是偶数个。假设是奇校验。校验位为 1，这样就有 3 个逻辑高位。高位和低位不是真正的检查数据，简单置位逻辑高或者逻辑低校验。

这样使得接收设备可以知道一个位的状态。有机会推断是否有噪声干扰了通信或者是否传输和接收数据是否不同步。

7.3 实验例程

（参考开源实验例程）实验四 串口实验

7.4 实验程序配置解析

```
#include "stm32f10x.h"
#include <stdio.h>
```

```
#include "usart_printf.h"
#include "systick.h"
#include "gpio.h"

/************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
************************************************************/
int main()
{
    u8 i;
    u8 len;
    u16 time=0;
    SystemInit();                    //配置系统时钟 72M
SysTick_Init();             //配置滴答定时器
uart_init(9600);            //串口初始化
GPIO_Config();              //gpio 初始化配置
while (1)                        //主程序 while(1)
    {
        if(USART_RX_STA&0x8000)
        {
            len=USART_RX_STA&0x3fff;              //得到此次接收到的数
据长度
            printf("\r\n 您发送的消息为:\r\n");
            for(i=0;i<len;i++)
            {
```

```
                USART1->DR=USART_RX_BUF[i];
                while((USART1->SR&0X40)==0);    //等待发送结束
            }
            printf("\r\n\r\n"); //插入换行
        USART_RX_STA=0;
        }else
        {
            time++;
            if(time%5000==0)
            {
                printf("\r\n LCEDASTM32 高教板  串口实验\r\n");
                printf("EasyEDA\r\n\r\n\r\n");
            }
            if(time%200==0)printf("请输入数据,以回车键结束\r\n");
            if(time%30==0)
            {
                    LED1_ON();                  //LED1 点亮
                    delay_ms(500);
                    LED1_OFF();             //LED1 熄灭
                delay_ms(500);
            }
        }
    }
}
```

7.5 实验结果

连接高教板串口至 PC 端，打开串口调试助手，配置完串口参数后，在发送去输入字符串后回车，可在接受去收到输入的字符串，串口通讯成功。

第八章 LCD 显示实验

8.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 的 LCD 驱动及显示的程序编程

8.2 实验原理

TFT-LCD 简介

TFT-LCD 即薄膜晶体管液晶显示器。TFT-LCD 与无源 TN-LCD、STN-LCD 的简单矩阵不同，它在液晶显示屏的每一个象素上都设置有一个薄膜晶体管（ TFT），可有效地克服非选通时的串扰，使显示液晶屏的静态特性与扫描线数无关，因此大大提高了图像质量。 TFT-LCD 也被叫做真彩液晶显示器。JL_MOD_LCD_2.8-V2.0 TFTLCD 模块采用 16 位的并方式与外部连接。

80 并口有如下一些信号线如图 10.2.1 所示：

注：TFTLCD 模块的 RST 信号线是直接接到 STM32 的复位脚上，并不由软件控制，这样可以省下来一个 IO 口。另外我们还需要一个背光控制线来控制 TFTLCD 的背光。所以，我们总共需要的 IO 口数目为 21 个。

信号线	作用
CS	TFTLCD 片选信号
WR	向 TFTLCD 写入数据
RD	从 TFTLCD 读取数据
D[15：0]	16 位双向数据线
RST	硬复位 TFTLCD
RS	命令/数据标志（0，读写命令；1，读写数据）

图 10.2.1 IO 介绍

ILI9341 控制器介绍

ILI9341 液晶控制器自带显存，其显存总大小为 172800（ 24032018/8），即 18 位模式（ 26 万色）下的显存量。在 16 位模式下，ILI9341 采用 RGB565 格式存储颜色数据，此时 ILI9341 的 18 位数据线与 MCU 的 16 位数据线以及 LCD GRAM 的对应关系如图 10.2.2 所示:

9341总线	D17	D16	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MCU数据（16位）	D15	D14	D13	D12	D11	NC	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	NC
LCD GRAM（16位）	R[4]	R[3]	R[2]	R[1]	R[0]	NC	G[5]	G[4]	G[3]	G[2]	G[1]	G[0]	B[4]	B[3]	B[2]	B[1]	B[0]	NC

图 10.2.2 对应关系

ILI9341 在 16 位模式下面，数据线有用的是： D17~D13 和 D11~D1， D0 和 D12 没有用到，实际上在我们 LCD 模块里面， ILI9341 的 D0 和 D12 压根就没有引出来，这样，ILI9341 的 D17~D13 和 D11~D1 对应 MCU 的 D15~D0。

MCU 的 16 位数据，最低 5 位代表蓝色，中间 6 位为绿色，最高 5 位为红色。数值越大，表示该颜色越深。另外，特别注意 ILI9341 所有的指令都是 8 位的（高 8 位无效）。且参数除了读写 GRAM 的时候是 16 位，其他操作参数，都是 8 位的。

其余特性在这里不多做介绍了。

8.3 实验例程

（参考开源实验例程）实验五 LCD 显示实验

8.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "usart_printf.h"
#include <stdio.h>
#include "systick.h"
void RCC_Configuration(void);                    //系统始终配置
/*************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
*************************************************************/
int main(void)
{
    u8 lcd_id[12];          //存放 LCD ID 字符串
```

```
    SysTick_Init();         //嘀嗒定时器初始化
    RCC_Configuration();    //系统时钟初始化
    GPIO_Configuration();//LED_GPIO 引脚配置初始化
    uart_init(9600);        //串口初始化
    LCD_Init();             //LCD 初始化
    LCD_Clear(BROWN);       //清屏为棕色
    sprintf((char*)lcd_id,"LCD ID:%04X",lcddev.id);          // 将 LCD
ID 打印到 lcd_id 数组。
    while(1)
    {
        LCD_Clear(WHITE);               //清屏为白色
        POINT_COLOR=BLUE;               //画笔颜色蓝色
        LCD_ShowString(30,40,200,24,24,"EasyEDA");    //显示字符串
        LCD_ShowString(30,70,200,24,24,"Layout & LCEDA");//显示字符串
        LCD_ShowString(30,100,200,24,24,lcd_id);    //显示 LCD ID
        LCD_ShowString(30,130,200,24,24,"2020/2/18");     //显示日期
        LED1_ON();          //LED1 点亮
        delay_ms(500);
        LED1_OFF();    //LED1 熄灭
        delay_ms(500);
    }
}
```

8.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果：

LCD 显示：EasyEDA

LCD 显示：Layout & LCEDA

LCD 显示：9341

LCD 显示：2020/2/18

第九章 窗口看门狗实验

9.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 窗口看门狗的程序编程

9.2 实验原理

窗口看门狗的原理与独立看门狗一致，之所以称之为窗口看门狗就是因为其喂狗时间是一个有上下限的范围（窗口）如图 9.2.1 所示，我们可以通过设定相关寄存器，设定其上限时间（下限固定），喂狗的时间不能过早也不能过晚，应该在下限时间（固定值）— 上限时间（可自己设定）之间，这里注意自己设置的上限时间应大于下限时间。而独立看门狗限制喂狗时间在 0-x 内，x 由相关寄存器决定。无论是独立看门狗还是窗口看门狗，喂狗的时间不能过晚。

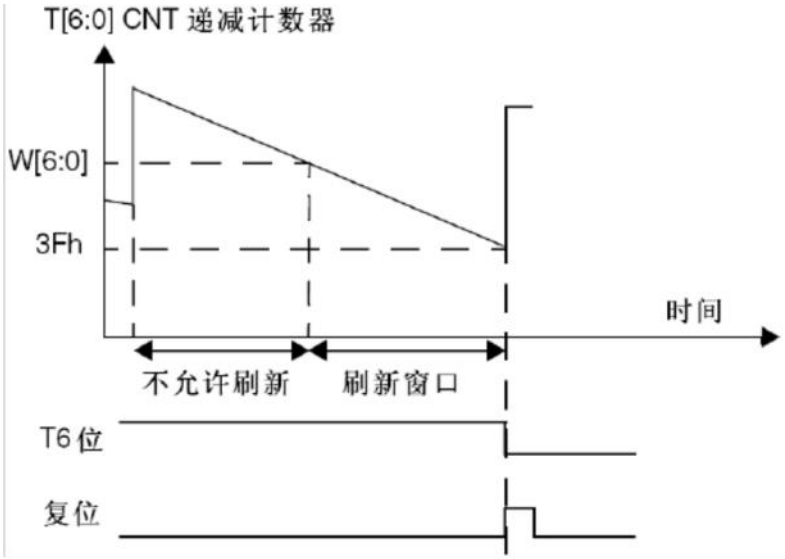


图 9.2.1 时间关系

接下来，介绍和窗口看门狗相关的 3 个寄存器。

1.控制寄存器（WWDG_CR），该寄存器的各位描述如下图 9.2.2 所示：

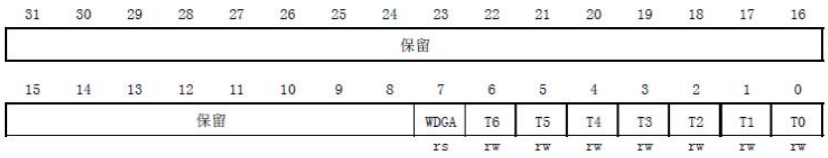


图 9.2.2 控制寄存器

WWDG_CR 只有低八位有效，T[6：0]用来存储看门狗的计数器值，随时更

新的，每个窗口看门狗计数周期（4096×2^WDGTB）减 1。当该计数器的值从 0X40 变为 0X3F 的时候，将产生看门狗复位。WDGA 位则是看门狗的激活位，该位由软件置 1，以启动看门狗，并且一定要注意的是该位一旦设置，就只能在硬件复位后才能清零了。

2.配置寄存器（WWDG_CFR），该寄存器的各位及其描述如图 9.2.3 所示：

位31:8	保留。
位9	**EWI**: 提前唤醒中断 此位若置1，则当计数器值达到40h，即产生中断。 此中断只能由硬件在复位后清除。
位8:7	**WDGTB[1:0]**: 时基 预分频器的时基可根据如下修改： 00: CK计时器时钟(PCLK1除以4096)除以1 01: CK计时器时钟(PCLK1除以4096)除以2 10: CK计时器时钟(PCLK1除以4096)除以4 11: CK计时器时钟(PCLK1除以4096)除以8
位6:0	**W[6:0]**: 7位窗口值 这些位包含了用来与递减计数器进行比较用的窗口值。

图 9.2.3 配置寄存器

该位中的 EWI 是提前唤醒中断，也就是在快要产生复位的前一段时间（T[6:0]=0X40） 来提醒我们，需要进行喂狗了，否则将复位！因此，我们一般用该位来设置中断，当窗口看门狗的计数器值减到 0X40 的时候，如果该位设置，并开启了中断，则会产生中断，我们可以在中断里面向 WWDG_CR 重新写入计数器的值，来达到喂狗的目的。注意这里在进入中断后，必须在不大于 1 个窗口看门狗计数周期的时间（在 PCLK1 频率为 36M 且 WDGTB 为 0 的条件下，该时间为 113us）内重新写 WWDG_CR，否则，看门狗将产生复位！

3.状态寄存器（WWDG_SR），该寄存器用来记录当前是否有提前唤醒的标志。该寄存器仅有位 0 有效，其他都是保留位。当计数器值达到 40h 时，此位由硬件置 1。它必须通过软件写 0 来清除。对此位写 1 无效。即使中断未被使能，在计数器的值达到 0X40 的时候，此位也会被置 1。

9.3 实验例程

（参考开源实验例程）实验六 窗口看门狗实验

9.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
void Delay(__IO uint32_t nCount);
/*******************************************************************
* 函数名    : main
* 描述      : 主函数
* 输入      : None
* 输出      : None
* 返回      : None
*******************************************************************/
int main(void)
{
    SystemInit();                       //--配置系统主频为 72MHz
    SysTick_Configuration();            //--SysTick 配置初始化
    NVIC_Configuration();               //--中断配置初始化
    GPIO_Configuration();               //--GPIO 初始化配置
    Test();                             //--LED1 点亮一次
    Delay(0XFFFF);                      //延时一段时间
    WWDG_Configuration();               //--窗口看门狗配置初始化
    while (1)                           //等待窗口看门触发
    {
        LED1_OFF();                     //LED1 熄灭
    }
}
```

9.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果，LED1 点亮一次，然后是 LED2 不停的闪烁。

第十章 独立看门狗实验

10.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 独立看门狗的程序编程

10.2 实验原理

单片机系统在外界的干扰下（比如磁场等）会出现程序跑飞的现象导致出现死循环，从而系统无法正常工作。 看门狗电路就是为了避免这种情况的发生而设计的。 看门狗的作用就是如果在一定时间内（通过定时计数器实现） 没有接收喂狗信号（表示 MCU 已经挂了），便实现处理器的自动复位重启（发送复位信号），以实现系统的正常运行。看门狗就是定期检查芯片内部程序运行情况，运行一旦发生错误就向芯片发出重启信号的电路。看门狗在程序的中断中拥有最高的优先级。

下面先了解几个与独立看门狗相关联的寄存器之后讲解怎么通过库函数来实现配置。首先是键值寄存器 IWDG_KR，该寄存器的各位描述如图 10.2.1 所示：

位31:16	保留，始终读为0。
位15:0	**KEY[15:0]**: 键值(只写寄存器，读出值为0x0000) 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。 写入0x5555表示允许访问IWDG_PR和IWDG_RLR寄存器。 写入0xCCCC，启动看门狗工作(若选择了硬件看门狗则不受此命令字限制)。

图 10.2.1 键值寄存器

在键值寄存器(IWDG_KR)中写入 0xCCCC，开始启用独立看门狗；此时计数器开始从其复位值 0xFFF 递减计数。当计数器计数到末尾 0x000 时，会产生一个复位信号(IWDG_RESET)。无论何时，只要键寄存器 IWDG_KR 中被写入 0xAAAA，IWDG_RLR 中的值就会被重新加载到计数器中从而避免产生看门狗复位。IWDG_PR

和 IWDG_RLR 寄存器具有写保护功能。要修改这两个寄存器的值，必须先向 IWDG_KR 寄存器中写入 0x5555。

将其他值写入这个寄存器将会打乱操作顺序，寄存器将重新被保护。重装载操作(即写入 0xAAAA)也会启动写保护功能。还有两个寄存器，一个预分频寄存器（IWDG_PR），该寄存器用来设置看门狗时钟的分频系数。 另一个重装载寄存器。该寄存器用来保存重装载到计数器中的值。该寄存器也是一个 32 位寄存器，但是只有低 12 位是有效的。只要对以上三个寄存器进行相应的设置，我们就可以启动 STM32 的独立看门狗。

10.3 实验例程

（参考开源实验例程）实验七 独立看门狗实验

10.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include <stdio.h>
#include "usart_printf.h"
#include "SysTick.h"

void RCC_Configuration(void);
/*************************************************************
* 函数名    : main
* 描述      : 主函数
* 输入      : None
* 输出      : None
* 返回      : None
*************************************************************/
int main(void)
{
    RCC_Configuration();     //系统时钟初始化配置
    SysTick_Init();          //嘀嗒定时器初始化
    GPIO_Configuration(); //LED KEY_WAKE GPIO 初始化配置
```

```
uart_init(9600);          //串口初始化
LCD_Init();               //LCD 初始化配置
delay_ms(500);            //让人看得到灭
IWDG_WriteAccessCmd(IWDG_WriteAccess_Enable);     //使能对寄存器 IWDG_PR 和 IWDG_RLR 的写操作
IWDG_SetPrescaler(IWDG_Prescaler_64);// 设置 IWDG 预分频值:设置 IWDG 预分频值为 64
IWDG_SetReload(625); //设置 IWDG 重装载值
IWDG_ReloadCounter(); //按照 IWDG 重装载寄存器的值重装载 IWDG 计数器
IWDG_Enable();            //使能 IWDG  喂狗时间 1S
LED1_ON();                //点亮 LED1
LCD_Clear(WHITE);         //清屏背景色白色
POINT_COLOR=BLUE;         //设置画笔颜色蓝色
LCD_ShowString(30,40,200,24,24,"IWDG text");        //显示字符串
 while(1)
{
    if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)==0x01)//按键按下:高电平有效  若，蜂鸣器部分跳线帽未取下，按下此按键蜂鸣器会发生
    {
        IWDG_ReloadCounter();
//按照 IWDG 重装载寄存器的值重装载 IWDG
    }
}
```

10.5 实验结果

下载完程序以后，在未按下 KEY_WAKE 按键时，会观察到 LED1 闪烁以及 LCD 在 1S 时间节点复位，若在 1S 内按下 KEY_WAKE，会观察到 LED1 常量，LCD 显示 IWDG text。

第十一章 定时器实验

11.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 定时器的程序编程

11.2 实验原理

STM32F103ZET6 一共有 8 个定时器，其中分别为：

高级定时器（TIM1、TIM8）；通用定时器（TIM2、TIM3、TIM4、TIM5）；基本定时器（TIM6、TIM7）。

他们之间的区别情况如图 11.2.1：

STM32定时器的区别

定时器种类	位数	计数器模式	发出DMA请求	捕获/比较通道个数	互补输出	特殊应用场景
高级定时器	16	向上、向下、向上/下	可以	4	有	带死区控制盒紧急刹车，可应用于PWM电机控制
通用定时器	16	向上、向下、向上/下	可以	4	无	通用。定时计数，PWM输出，输入捕获，输出比较
基本定时器	16	向上、向下、向上/下	可以	0	无	主要应用于驱动DAC

图 11.2.1 定时器区别

通用定时器功能特点描述

STM32 的通用定时器是由一个可编程预分频器（PSC）驱动的 16 位自动重装载计数器（CNT）构成，可用于测量输入脉冲长度（输入捕获）或者产生输出波形（输出比较和 PWM）等。

STM3 的通用 TIMx（TIM2、TIM3、TIM4 和 TIM5）定时器功能特点包括：位于低速的 APB1 总线上（注意：高级定时器是在高速的 APB2 总线上）；16 位向上、向下、向上/向下（中心对齐）计数模式，自动装载计数器（TIMx_CNT）；16 位可编程（可以实时修改）预分频器（TIMx_PSC），计数器时钟频率的分频系数 为 1～65535 之间的任意数值；4 个独立通道（TIMx_CH1~4），这些通道可以用来作为：

输入捕获

输出比较

PWM 生成（边缘或中间对齐模式）

单脉冲模式输出

可使用外部信号（TIMx_ETR）控制定时器和定时器互连（可以用 1 个定时器控制另外一个定时器）的同步电路。

如下事件发生时产生中断/DMA（6 个独立的 IRQ/DMA 请求生成器）：

更新：计数器向上溢出/向下溢出，计数器初始化(通过软件或者内部/外部触发)

触发事件（计数器启动、停止、初始化或者由内部/外部触发计数）

输入捕获

输出比较

支持针对定位的增量（正交）编码器和霍尔传感器电路

触发输入作为外部时钟或者按周期的电流管理

STM32 的通用定时器可以被用于：测量输入信号的脉冲长度（输入捕获）或者产生输出波形（输出比较和 PWM）等。

使用定时器预分频器和 RCC 时钟控制器预分频器，脉冲长度和波形周期可以在几个微秒到几个毫秒间调整。STM32 的每个通用定时器都是完全独立的，没有互相共享的任何资源。

计数器模式

通用定时器可以向上计数、向下计数、向上向下双向计数模式如图 11.2.2 所示。

向上计数模式：计数器从 0 计数到自动加载值（TIMx_ARR），然后重新从 0 开始计数并且产生一个计数器溢出事件。

向下计数模式：计数器从自动装入的值（TIMx_ARR）开始向下计数到 0，然后从自动装入的值重新开始，并产生一个计数器向下溢出事件。

中央对齐模式（向上/向下计数）：计数器从 0 开始计数到自动装入的值-1，产生一个计数器溢出事件，然后向下计数到 1 并且产生一个计数器溢出事件；然后再从 0 开始重新计数。

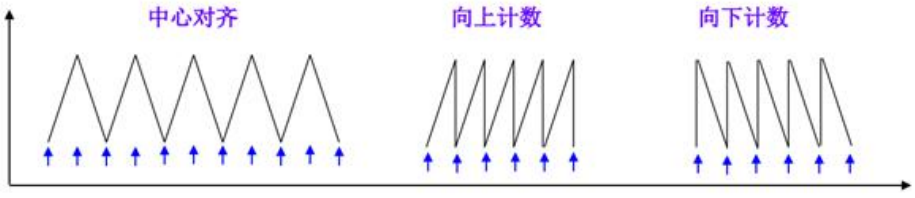


图 11.2.2 计数器模式

通用定时器工作流程

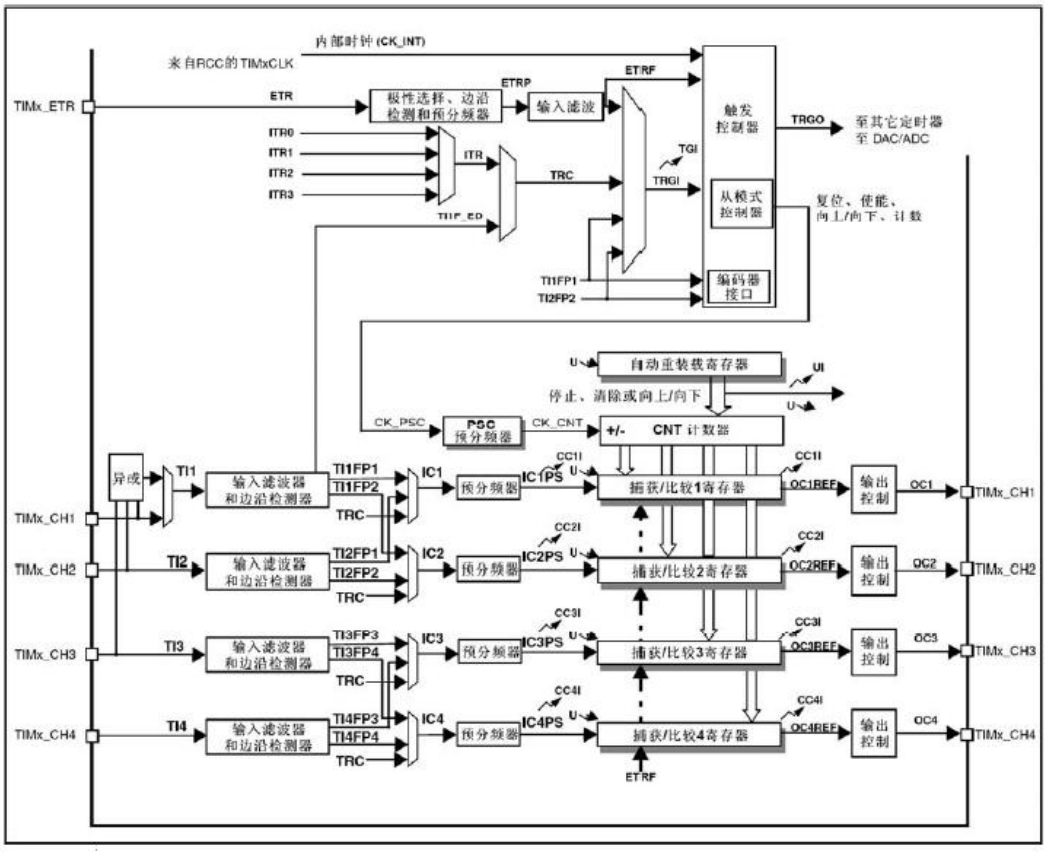


图 11.2.3 定时器框图

对于如图 11.2.3 所示的定时器框图，分成四部分来讲：最顶上的一部分（计数时钟的选择）、中间部分（时基单元）、左下部分（输入捕获）、右下部分（PWM 输出）。这里主要介绍一下前两个，后两者的内容会在后面的文章中讲解到。

计数时钟的选择

计数器时钟可由下列时钟源提供：

内部时钟（TIMx_CLK）

外部时钟模式 1：外部捕捉比较引脚（TIx）

外部时钟模式 2：外部引脚输入（TIMx_ETR）

内部触发输入（ITRx）：使用一个定时器作为另一个定时器的预分频器，如可以配置一个定时器 Timer1 而作为另一个定时器 Timer2 的预分频器。

内部时钟源

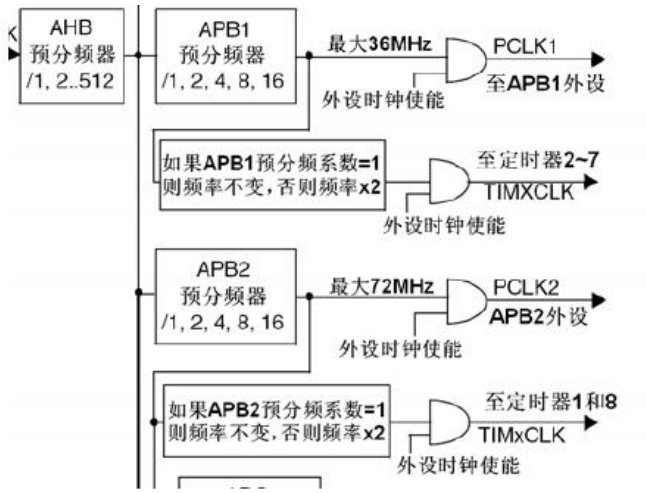


图 11.2.3 内部时钟源

从图 11.2.3 中可以看出：由 AHB 时钟经过 APB1 预分频系数转至 APB1 时钟，再通过某个规定转至 TIMxCLK 时钟（即内部时钟 CK_INT、CK_PSC）。最终经过 PSC 预分频系数转至 CK_CNT。

那么 APB1 时钟怎么转至 TIMxCLK 时钟呢？除非 APB1 的分频系数是 1，否则通用定时器的时钟等于 APB1 时钟的 2 倍。

例如：默认调用 SystemInit 函数情况下：SYSCLK=72M、AHB 时钟=72M、APB1 时钟=36M，所以 APB1 的分频系数=AHB/APB1 时钟=2。所以，通用定时器时钟 CK_INT=2*36M=72M。最终经过 PSC 预分频系数转至 CK_CNT。

时基单元

时基单元包含：计数器寄存器（TIMx_CNT）、预分频器寄存器（TIMx_PSC）、自动装载寄存器（TIMx_ARR）三部分。

对不同的预分频系数，计数器的时序图为：

当预分频器的参数从 1 变到 2 时，计数器的时序图

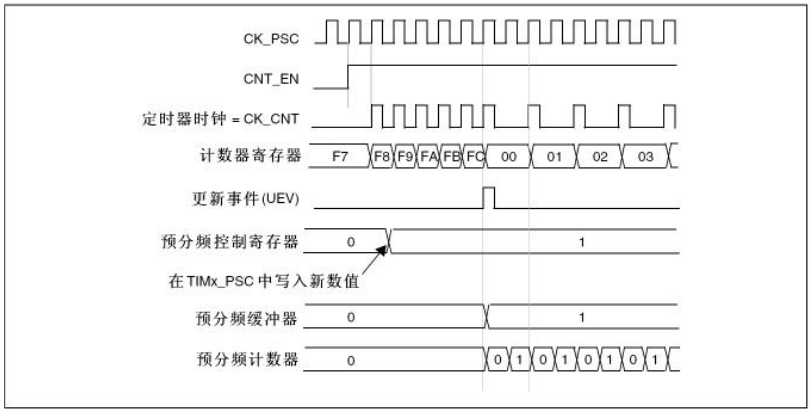


图 11.2.4 计数器时序图

计数模式

此时，再来结合时钟的时序图和时基单元，分析一下各个计数模式：

向上计数模式

计数器时序图，内部时钟分频因子为 2

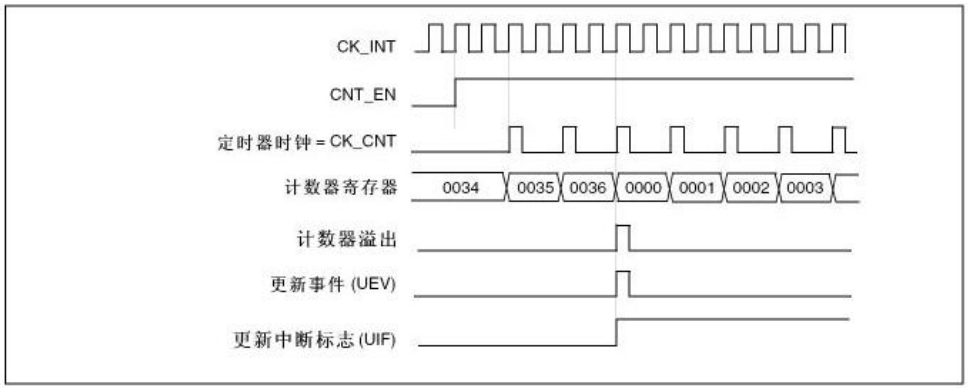


图 11.2.5 向上计数模式

向下计数模式

计数器时序图，内部时钟分频因子为 2

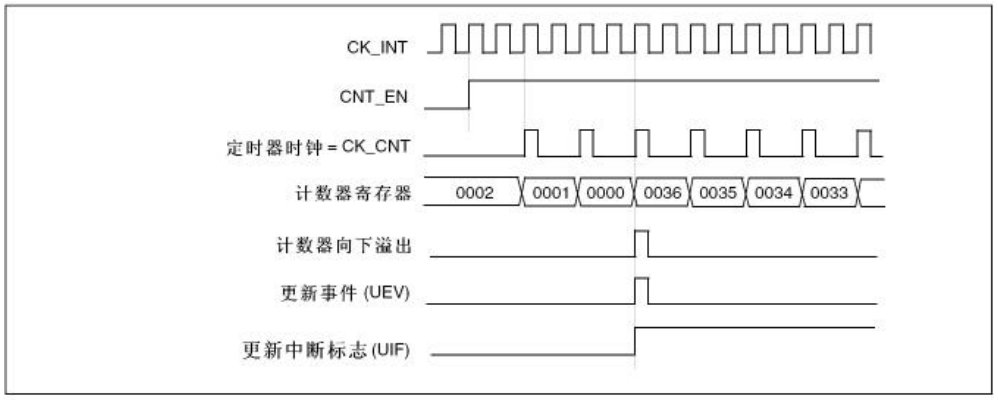


图 11.3.6 向下计数模式

中央对齐模式

计数器时序图，内部时钟分频因子为 2

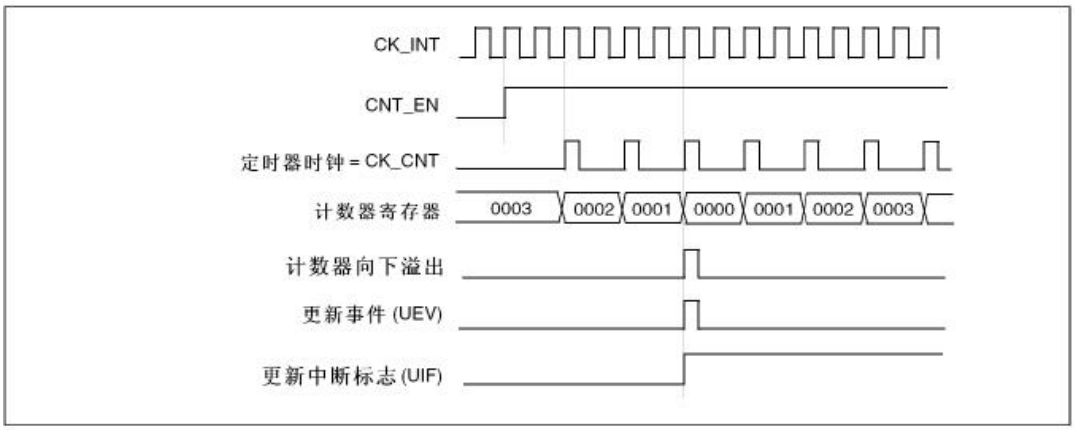


图 11.3.7 中央对齐模式

具体配置，参考数据手册定时器章节内容。

11.3 实验例程

（参考开源实验例程）实验八 定时器实验

11.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "TIM.h"
#include "GPIO.h"

void NVIC_Configuration(void);
void RCC_Configuration(void);
u8 tim_flag;
/*************************************************************
* 函数名    : main
* 描述      : 主函数
* 输入      : None
* 输出      : None
* 返回      : None
*************************************************************/
int main(void)
```

```
{
    tim_flag=0;
    RCC_Configuration();                    //系统时钟配置
    GPIO_Configuration();                   //GPIO 初始化配置
    NVIC_Configuration();                   //中断初始化配置
    TIM3_Configuration();                   //定时器初始化配置
     while(1)                               //等待定时 3 中断到达
    {
    }
}
```

11.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果：

LED1、LED2 同时被点亮与熄灭，时间间隔为 1S。

第十二章 待机实验

12.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 待机唤醒功能的程序编程

12.2 实验原理

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。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。

STM32F10xxx 有三种低功耗模式：

睡眠模式(Cortex™-M3 内核停止，所有外设包括 Cortex-M3 核心的外设，如 NVIC、系统时钟(SysTick)等仍在运行)

停止模式(所有的时钟都已停止)

待机模式(1.8V 电源关闭)

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。

低功耗模式如图 12.2.1 所示

模式	进入	唤醒	对1.8V区域时钟的影响	对V_{DD}区域时钟的影响	电压调节器
睡眠(SLEEP-NOW或SLEEP-ON-EXIT)	WFI	任一中断	CPU时钟关，对其他时钟和ADC时钟无影响	无	开
	WFE	唤醒事件			
停机	PDDS和LPDS位+SLEEPDEEP位+WFI或WFE	任一外部中断(在外部中断寄存器中设置)	关闭所有1.8V区域的时钟	HSI 和HSE的振荡器关闭	开启或处于低功耗模式(依据电源控制寄存器(PWR_CR)的设定)
待机	PDDS位+SLEEPDEEP位+WFI或WFE	WKUP引脚的上升沿、RTC闹钟事件、NRST引脚上的外部复位、IWDG复位			关

图 12.2.1 低功耗模式

外部时钟的控制

在运行模式下，任何时候都可以通过停止为外设和内存提供时钟(HCLK 和

PCLKx)来减少功耗。为了在睡眠模式下更多地减少功耗，可在执行 WFI 或 WFE 指令前关闭所有外设的时钟。

通过设置 AHB 外设时钟使能寄存器(RCC_AHBENR)、APB2 外设时钟使能寄存器(RCC_APB2ENR)和 APB1 外设时钟使能寄存器(RCC_APB1ENR)来开关各个外设模块的时钟。

待机模式

待机模式可实现系统的最低功耗。该模式是在 Cortex-M3 深睡眠模式时关闭电压调节器。整个 1.8V 供电区域被断电。PLL、HSI 和 HSE 振荡器也被断电。SRAM 和寄存器内容丢失。只有备份的寄存器和待机电路维持供电

待机模式如图 12.2.2 所示

待机模式	说明
进入	在以下条件下执行WFI(等待中断)或WFE(等待事件)指令： - 设置Cortex™-M3系统控制寄存器中的SLEEPDEEP位 - 设置电源控制寄存器(PWR_CR)中的PDDS位 - 清除电源控制/状态寄存器(PWR_CSR)中的WUF位
退出	WKUP引脚的上升沿、RTC闹钟事件的上升沿、NRST引脚上外部复位、IWDG复位。
唤醒延时	复位阶段时电压调节器的启动。

图 12.2.2 待机模式

退出待机模式

当一个外部复位(NRST 引脚)、IWDG 复位、WKUP 引脚上的上升沿或 RTC 闹钟事件的上升沿发生时，微控制器从待机模式退出。从待机唤醒后，除了电源控制/状态寄存器(PWR_CSR)，所有寄存器被复位。从待机模式唤醒后的代码执行等同于复位后的执行(采样启动模式引脚、读取复位向量等)。电源控制/状态寄存器(PWR_CSR)将会指示内核由待机状态退出。

12.3 实验例程

（参考开源实验例程）实验九 待机实验

12.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
```

```
#include "bmp.h"
#include "wakeup.h"
#include "Systick.h"
#include "usart_printf.h"

void RCC_Configuration(void);
void NVIC_Configuration(void);
/*************************************************************
* 函数名称      : int main(void)
* 功能介绍      : 主函数
* 输      入      : 无
* 输      出      : 无
* 返 回 值      : 无
*************************************************************/
int main(void)
{
    RCC_Configuration();                  //系统始终初始化
    GPIO_Configuration();                 //GPIO 初始化
    SysTick_Init();                       //滴答定时器配置初始化
    uart_init(9600);                      //串口初始化
    NVIC_Configuration();                 //中断配置初始化
    LCD_Init();                           //LCD 配置初始化
    WKUP_Init();                          //待机配置初始化
    LCD_Clear(CYAN);                      //清屏淡蓝色
   while(1)
    {
        LCD_Clear(WHITE);                 //清屏为白色
        delay_ms(50);
        POINT_COLOR=BLUE;                 //画笔颜色蓝色
```

```
            LCD_ShowString(30,40,200,24,24,"EasyEDA");                    //
显示字符串
            LCD_ShowString(30,70,200,24,24,"Layout & LCEDA");
    //显示字符串
             LED1_ON();
            LED2_OFF();
            delay_ms(500);
            LED2_ON();
            LED1_OFF();
            delay_ms(500);
        }
    }
```

12.5 实验结果

在程序下载完之后，高教板检测不到 WK_UP 的持续按下（3 秒以上），所以直接进入待机模式，看起来和没有下载代码一样。此时，长按 WAKE_UP 按键 3S 以上，系统开机，液晶屏显示"EasyEDA"、"Layout & LCEDA"两行字符串，同时 LED1、LED2 轮流点亮。

第十三章 PWM 实验

13.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 输出 PWM 的程序编程

13.2 实验原理

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。简单一点，就是对脉冲宽度的控制，PWM 原理如图 15.2.1 所示

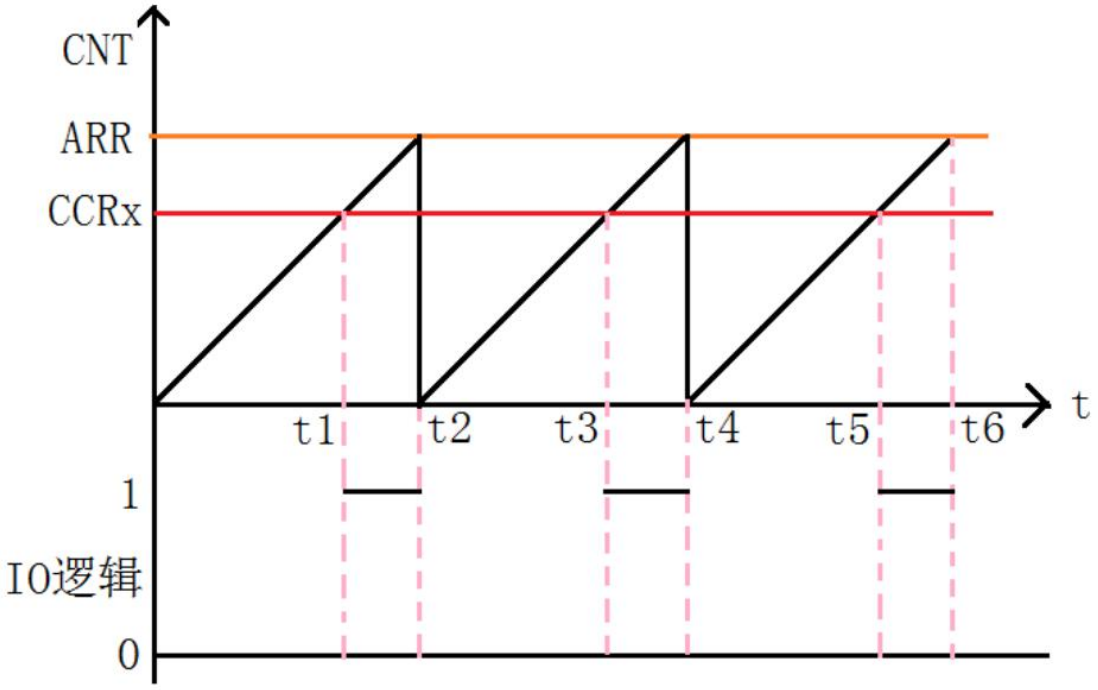


图 13.2.1 PWM 原理图

图 13.2.1 就是一个简单的 PWM 原理示意图。图中，我们假定定时器工作在向上计数 PWM 模式，且当 CNT<CCRx 时，输出 0，当 CNT>=CCRx 时输出 1。那么就可以得到如上的 PWM 示意图：当 CNT 值小于 CCRx 的时候，IO 输出低电平(0)，当 CNT 值大于等于 CCRx 的时候，IO 输出高电平(1)，当 CNT 达到 ARR 值的时候，重新归零，然后重新向上计数，依次循环。改变 CCRx 的值，就可以改变 PWM 输出的占空比，改变 ARR 的值，就可以改变 PWM 输出的频率，这就是 PWM 输出的原理。

13.3 实验例程

（参考开源实验例程）实验十 PWM 实验

13.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "Systick.h"
#include "pwm.h"
#include "GPIO.h"


/*********************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
*********************************************************************/
int main()
{
    u16 pwmval = 0;
    u8 val = 1;
    SysTick_Init();                 //嘀嗒定时器配置初始化
    GPIO_Config();                  //gpio 配置初始化
    TIM1_PWM_Init(899,0);           //PWM 频率=72000/(899+1)=80Khz
     while(1)
    {
        delay_ms(1);
        if(val)
        {
            pwmval++;
```

```
        }
        else pwmval--;
        if(pwmval>200)val=0;
        if(pwmval==0)val=1;
        TIM_SetCompare1(TIM1,pwmval);
    }
}
```

13.5 实验结果

在完成软件设计之后，将我们将编译好的文件下载到高教板上，观看其运行结果，我们将看 LED1 不停的由暗变到亮，然后又从亮变到暗。

第十四章 ADC 实验

14.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 ADC 采样的程序编程

14.2 实验原理

ADC 的基本介绍

ADC 的基本定义

Analog-to-Digital Converter 的缩写。指模/数转换器或者模拟/数字转换器。是指将连续变量的模拟信号转换为离散的数字信号的器件。

典型的模拟数字转换器将模拟信号转换为表示一定比例电压值的数字信号。

ADC 的主要特征

12 位逐次逼近型的模拟数字转换器；

最多带 3 个 ADC 控制器，可以单独使用，也可以使用双重模式提高采样率；

最多支持 23 个通道，可最多测量 21 个外部和 2 个内部信号源；

支持单次和连续转换模式；

转换结束，注入转换结束，和发生模拟看门狗事件时产生中断；

通道 0 到通道 n 的自动扫描模式；

自动校准；

采样间隔可以按通道编程；

规则通道和注入通道均有外部触发选项；

转换结果支持左对齐或右对齐方式存储在 16 位数据寄存器；

ADC 转换时间：最大转换速率 1us（最大转换速度为 1MHz，在 ADCCLK=14M，采样周期为 1.5 个 ADC 时钟下得到）；

ADC 供电要求：2.4V-3.6V；

ADC 输入范围：VREF-≤ VIN ≤ VREF+。

STM32F10x 系列芯片 ADC 通道和引脚对应关系

ADC 通道和引脚对应关系如图 14.2.1 所示

	ADC1	ADC2	ADC3
通道0	PA0	PA0	PA0
通道1	PA1	PA1	PA1
通道2	PA2	PA2	PA2
通道3	PA3	PA3	PA3
通道4	PA4	PA4	PF6
通道5	PA5	PA5	PF7
通道6	PA6	PA6	PF8
通道7	PA7	PA7	PF9
通道8	PB0	PB0	PF10
通道9	PB1	PB1	
通道10	PC0	PC0	PC0
通道11	PC1	PC1	PC1
通道12	PC2	PC2	PC2
通道13	PC3	PC3	PC3
通道14	PC4	PC4	
通道15	PC5	PC5	
通道16	温度传感器		
通道17	内部参照电压		

图 14.2.1 对应关系

由上图中可以看出，STM32F103ZET6 带 3 个 ADC 控制器，一共支持 23 个通道，包括 21 个外部和 2 个内部信号源；但是每个 ADC 控制器最多只可以有 18 个通道，包括 16 个外部和 2 个内部信号源。

ADC 的基本原理

ADC 的工作框图，如图 14.2.1 所示

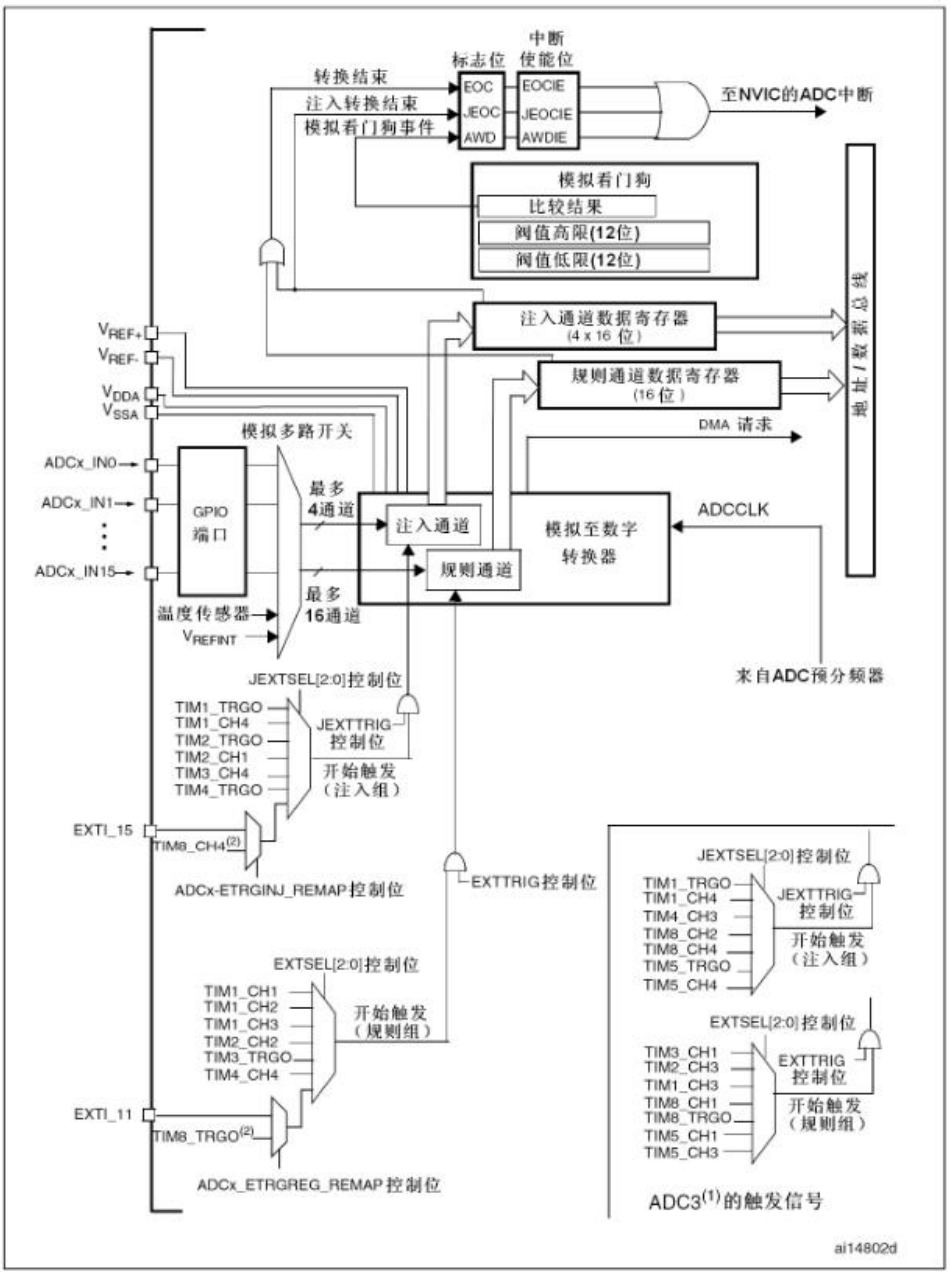


图 14.2.1 工作框图

ADC 模块的框图看起来比较复杂，具体 ADC 使用配置，参考 STM32F10X 使用手册。

14.3 实验例程

（参考开源实验例程）实验十一 ADC 实验

14.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
```

```
#include "lcd.h"
#include "ADC.h"
#include "systick.h"
#include "usart_printf.h"

void RCC_Configuration(void);
/*************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
*************************************************************/
int main(void)
{
    u16 adcx;
    float temp;
   RCC_Configuration();                    //系统时钟初始化
    SysTick_Init();                        //配置滴答定时器初始化
    uart_init(9600);                       //串口初始化，波特率 9600
    GPIO_Configuration();                  //GPIO 初始化
    ADC_Configuration();                   //ADC 初始化
    LCD_Init();                            //LCD 初始化
    LCD_Clear(CYAN);                       //清屏为淡蓝色
    POINT_COLOR=BLUE;                          //设置字体为红色

    LCD_ShowString(60,50,200,16,16,"Layout & LCEDA");
//显示字符串"Layout & LCEDA"
    LCD_ShowString(60,70,200,16,16,"EasyEDA");                    //
```

显示字符串"EasyEDA"

```
        POINT_COLOR=RED;                    //设置字体为蓝色
        LCD_ShowString(60,130,200,16,16,"ADC_VAL:      ");
    //显示 ADC 采样值
        LCD_ShowString(60,150,200,16,16,"ADC_VOL:0.000V");
    //显示 ADC 电压值
       while(1)
        {
            adcx=TestAdc();                 //ADC 采样
                LCD_ShowxNum(124,130,adcx,4,16,0);   //显示 ADC 的值
                temp=(float)adcx*(3.3/4096); //取整数
                adcx=temp;
                LCD_ShowxNum(124,150,adcx,1,16,0);       //显示电压值
                temp-=adcx;                 //取小数
                temp*=1000;
                LCD_ShowxNum(140,150,temp,3,16,0X80);   //显示小数值
                LED1_ON();
                delay_ms(100);
                LED1_OFF();
                delay_ms(200);
        }
    }
```

14.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"EasyEDA" //显示的字符串

"ADC_VAL:1836 " //显示当前 ADC 采样值

"ADC_VOL:1.479V" //显示转换后的电压值

第十五章 CPU 温度检测实验

15.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 CPU 温度采样的程序编程

15.2 实验原理

STM32 有一个内部的温度传感器，可以用来测量 CPU 及周围的温度(TA)。该温度传感器在内部和 ADCx_IN16 输入通道相连接，此通道把传感器输出的电压转换成数字值。温度传感器模拟输入推荐采样时间是 17.1μs。 STM32 的内部温度传感器支持的温度范围为：-40~125 度，精度为±1.5℃左右（实际效果不咋地）。

STM32 内部温度传感器的使用很简单，只要设置一下内部 ADC，并激活其内部通道就差不多了。关于 ADC 的设置，我们在第上一章已经进行了详细的介绍，这里就不再多说。接下来我们介绍一下和温度传感器设置相关的 2 个地方。

第一个地方，我们要使用 STM32 的内部温度传感器，必须先激活 ADC 的内部通道，这里通过 ADC_CR2 的 AWDEN 位（bit23）设置。设置该位为 1 则启用内部温度传感器。

第二个地方，STM32 的内部温度传感器固定的连接在 ADC 的通道 16 上，所以，我们在设置好 ADC 之后只要读取通道 16 的值，就是温度传感器返回来的电压值了。根据这个值，我们就可以计算出当前温度。计算公式如下：

T（℃）={（V25-Vsense）/Avg_Slope}+25

上式中：

V25=Vsense 在 25 度时的数值（典型值为：1.43）。

Avg_Slope=温度与 Vsense 曲线的平均斜率（单位：mv/℃或 uv/℃）（典型值：4.3mv/℃）。

利用以上公式，我们就可以方便的计算出当前温度传感器的温度了。

15.3 实验例程

（参考开源实验例程）实验十二 CPU 温度检测实验

15.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "ADC.h"
#include "systick.h"
#include "usart_printf.h"

void RCC_Configuration(void);

/*************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
*************************************************************/
int main(void)
{
    u16 adcx;
    float temp;
    float temperate;
    RCC_Configuration();            //系统时钟初始化
    SysTick_Init();                 //滴答定时器配置初始化
    uart_init(9600);                //串口初始化为 9600
    GPIO_Configuration();           //GPIO 初始化
    ADC_Configuration();            //ADC 初始化
    LCD_Init();                     //LCD 初始化
    LCD_Clear(CYAN);                //清屏为淡蓝色
```

```
    POINT_COLOR=RED;                    //设置字体为红色
    LCD_ShowString(60,50,200,16,16,"Layout & LCEDA");
    LCD_ShowString(60,70,200,16,16,"Temperature TEST");
    POINT_COLOR=BLUE;                   //设置字体为蓝色
    LCD_ShowString(60,130,200,16,16,"TEMP_VAL:");
    LCD_ShowString(60,150,200,16,16,"TEMP_VOL:0.000V");
    LCD_ShowString(60,170,200,16,16,"TEMPERATE:00.00C");
  while(1)
    {
        adcx=T_Get_Adc_Average(ADC_CH_TEMP,10);
//获取 ADC 采样值
        LCD_ShowxNum(132,130,adcx,4,16,0);
//显示 ADC 的值
        temp=(float)adcx*(3.3/4096);
        temperate=temp;                 //保存温度传感器的电压值
        adcx=temp;
        LCD_ShowxNum(132,150,adcx,1,16,0);    //显示电压值整数部分
        temp-=(u8)temp;                  //减掉整数部分
        LCD_ShowxNum(148,150,temp*1000,3,16,0X80);
//显示电压小数部分
        temperate=(1.43-temperate)/0.0043+25;
//计算出当前温度值
        LCD_ShowxNum(140,170,(u8)temperate,2,16,0);
//显示温度整数部分
        temperate-=(u8)temperate;
        LCD_ShowxNum(164,170,temperate*100,2,16,0X80);
//显示温度小数部分
        LED1_ON();
        delay_ms(100);
```

```
            LED1_OFF();
            delay_ms(200);
        }
    }
```

15.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"Temperature TEST" //显示的字符串

"TEMP_VAL:1749 " //显示当前 ADC 采样值

"TEMP_VOL:1.409V" //显示转换后的电压值

"TEMPERATE:29.85C" //显示转换后的温度值

第十六章 RTC 实验

16.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 RTC 时钟的程序编程

16.2 实验原理

RTC 简介

实时时钟是一个独立的定时器内部框图如图 16.2.1 所示。RTC 模块拥有一组连续计数的计数器，在相应软件配置下，可提供时钟日历的功能。修改计数器的值可以重新设置系统当前的时间和日期。

RTC 模块和时钟配置系统(RCC_BDCR 寄存器)处于后备区域，即在系统复位或从待机模式唤醒后，RTC 的设置和时间维持不变。

系统复位后，对后备寄存器和 RTC 的访问被禁止，这是为了防止对后备区域(BKP)的意外写操作。执行以下操作将使能对后备寄存器和 RTC 的访问：

- 设置寄存器 RCC_APB1ENR 的 PWREN 和 BKPEN 位，使能电源和后备接口时钟

- 设置寄存器 PWR_CR 的 DBP 位，使能对后备寄存器和 RTC 的访问。

主要特性

- 可编程的预分频系数：分频系数最高为 2^20
- 32 位的可编程计数器，可用于较长时间段的测量。
- 2 个分离的时钟：用于 APB1 接口的 PCLK1 和 RTC 时钟
  (RTC 时钟的频率必须小于 PCLK1 时钟频率的四分之一以上)。
- 可以选择以下三种 RTC 的时钟源：
  - HSE 时钟除以 128；
  - LSE 振荡器时钟；
  - LSI 振荡器时钟
- 2 个独立的复位类型：
  - APB1 接口由系统复位；

— RTC 核心(预分频器、闹钟、计数器和分频器)

只能由后备域复位

- 3 个专门的可屏蔽中断：
  - — 闹钟中断，用来产生一个软件可编程的闹钟中断。
  - — 秒中断，用来产生一个可编程的周期性中断信号(最长可达 1 秒)。
  - — 溢出中断，指示内部可编程计数器溢出并回转为 0 的状态。

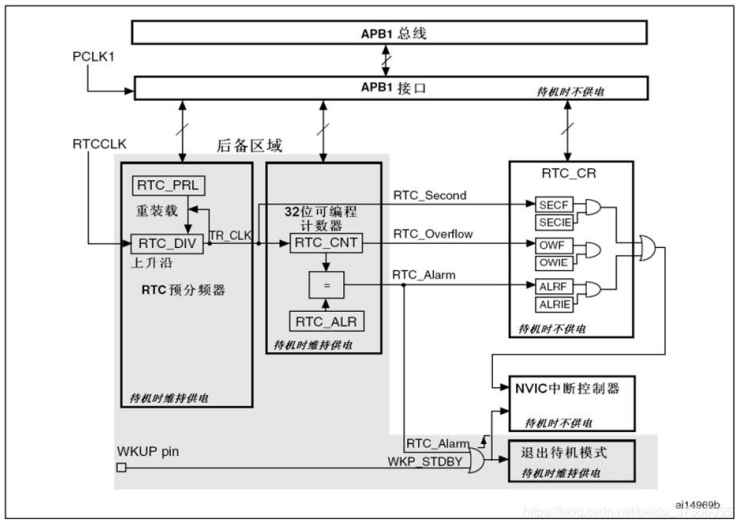


图 16.2.1 实时时钟

16.3 实验例程

（参考开源实验例程）实验十三 RTC 实验

16.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "rtc.h"
#include "systick.h"
#include "usart_printf.h"
```

```
void NVIC_Configuration(void);
void RCC_Configuration(void);
/*************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
*************************************************************/
int main(void)
{
  u8 t;
  RCC_Configuration();              //系统时钟初始化
    SysTick_Init();                 //滴答定时器配置初始化
    GPIO_Configuration();           //GPIO 初始化
    uart_init(9600);                //串口初始化为 9600
    LCD_Init();                     //LCD 初始化
    LCD_Clear(CYAN);                    //清屏为淡蓝色
    POINT_COLOR=RED;                        //设置字体为红色
    LCD_ShowString(60,90,200,16,16,"Layout & LCEDA");
    LCD_ShowString(60,110,200,16,16,"RTC TEST");
    while(RTC_Init())           //RTC 初始化，一定要初始化成功
    {
        LCD_ShowString(60,130,200,16,16,"RTC ERROR!     ");
        delay_ms(800);
        LCD_ShowString(60,130,200,16,16,"RTC Trying...");
    }
    RTC_Set(2020,3,2,10,11,45);             //设置时间
    /***************** 显示时间 *****************/
```

```
POINT_COLOR=BLUE;//设置字体为蓝色
LCD_ShowString(60,130,200,16,16,"      -  -       ");
LCD_ShowString(60,162,200,16,16,"   :  :   ");
 while(1)
{
   if(t!=timer.sec)
     {
          t=timer.sec;
          LCD_ShowNum(60,130,timer.w_year,4,16);
          LCD_ShowNum(100,130,timer.w_month,2,16);
          LCD_ShowNum(124,130,timer.w_date,2,16);
          switch(timer.week)
          {
               case 0:
                    LCD_ShowString(60,148,200,16,16,"Sunday     ");
                    break;
               case 1:
                    LCD_ShowString(60,148,200,16,16,"Monday     ");
                    break;
               case 2:
                    LCD_ShowString(60,148,200,16,16,"Tuesday    ");
                    break;
               case 3:
                    LCD_ShowString(60,148,200,16,16,"Wednesday");
                    break;
               case 4:
                    LCD_ShowString(60,148,200,16,16,"Thursday ");
                    break;
               case 5:
```

```
                    LCD_ShowString(60,148,200,16,16,"Friday    ");
                    break;
                case 6:
                    LCD_ShowString(60,148,200,16,16,"Saturday ");
                    break;
            }
            LCD_ShowNum(60,162,timer.hour,2,16);
            LCD_ShowNum(84,162,timer.min,2,16);
            LCD_ShowNum(108,162,timer.sec,2,16);
            LED1_ON();
            delay_ms(100);
            LED1_OFF();
            delay_ms(100);
        }
        delay_ms(10);
    }
}
```

16.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"RTC TEST" //显示的字符串

"2020- 3- 2" //显示当前设定是年月日

"Monday" //显示当前设定的星期

"10:12: 1" //显示当前设定的时间

第十七章 EEPROM 实验

17.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 24C02 的程序编程

17.2 实验原理

IIC 的基本介绍

IIC 的简介

IIC（Inter－Integrated Circuit）总线是一种由 PHILIPS 公司在 80 年代开发的两线式串行总线，用于连接微控制器及其外围设备。它是半双工通信方式。

IIC 总线最主要的优点是其简单性和有效性。由于接口直接在组件之上，因此 IIC 总线占用的空间非常小，减少了电路板的空间和芯片管脚的数量，降低了互联成本。总线的长度可高达 25 英尺，并且能够以 10Kbps 的最大传输速率支持 40 个组件。

IIC 总线的另一个优点是，它支持多主控(multimastering)， 其中任何能够进行发送和接收的设备都可以成为主总线如图 17.2.1 所示。一个主控能够控制信号的传输和时钟频率。当然，在任何时间点上只能有一个主控。

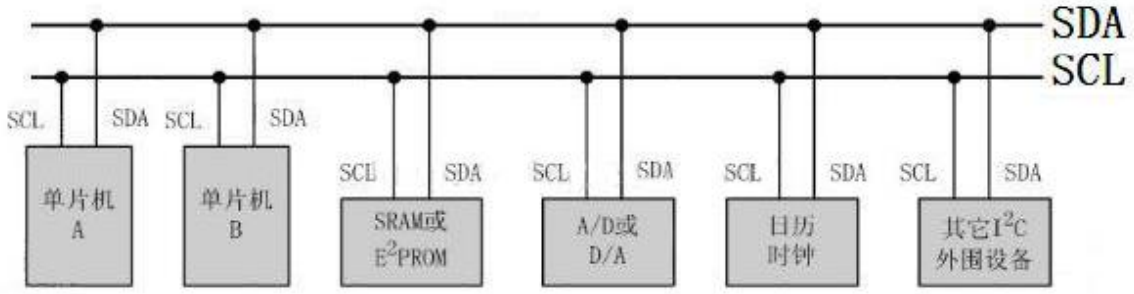
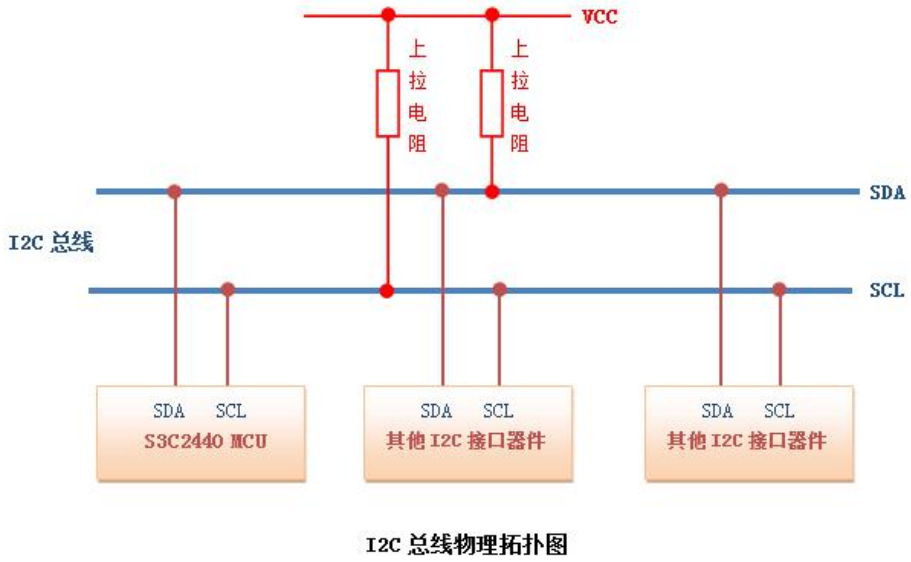


图 17.2.1 IIC 总线图

IIC 串行总线一般有两根信号线，一根是双向的数据线 SDA，另一根是时钟线 SCL，其时钟信号是由主控器件产生。所有接到 IIC 总线设备上的串行数据 SDA 都接到总线的 SDA 上，各设备的时钟线 SCL 接到总线的 SCL 上。对于并联在一条总线上的每个 IC 都有唯一的地址。



I2C 总线物理拓扑图

图 17.2.2 拓扑图

一般情况下，数据线 SDA 和时钟线 SCL 都是处于上拉电阻状态如图 17.2.2 所示。因为：在总线空闲状态时，这两根线一般被上面所接的上拉电阻拉高，保持着高电平。

STM32 的 IIC 接口

目前绝大多数的 MCU 都附带 IIC 总线接口，STM32 也不例外。但是在本文中，我们不使用 STM32 的硬件 IIC 来读取 24C02，而是通过软件的方式来模拟。原因是因为：STM32 的硬件 IIC 非常复杂，更重要的是它并不稳定，故不推荐使用。

17.3 实验例程

（参考开源实验例程）实验十四 24C02 实验

17.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "I2C.h"
#include "24C02.h"
#include "systick.h"
#include "usart_printf.h"
```

```
#include "KEY.h"
#define SIZE sizeof(TEXT_Buffer)
const u8 TEXT_Buffer[]={"STM32F103+24CXX!"};
 //要写入 24cxx 的字符串
void RCC_Configuration(void);
/*****************************************************************
* 函数名称 : main.c
* 功能介绍  : 主函数
* 输      入 : 无
* 输      出 : 无
* 返 回 值  : 无
*****************************************************************/
int main(void)
{
   u16 i=0;
    u8 key;
    u8 datatemp[SIZE];
    RCC_Configuration();              //系统时钟初始化
    SysTick_Init();                   //滴答定时器配置初始化
    GPIO_Configuration();             //GPIO 初始化
    uart_init(9600);                  //串口初始化 9600
    LCD_Init();                       //LCD 初始化
    KEY_Init();                       //按键配置初始化
    AT24CXX_Init();                   //24Cxx 初始化
    LCD_Clear(CYAN);                  //清屏为淡蓝色
    POINT_COLOR=RED;                  //设置字体为红色
    LCD_ShowString(60,50,200,16,16,"Layout & LCEDA");
    LCD_ShowString(60,70,200,16,16,"24CXX TEST");
```

```
while(AT24CXX_Check())          //检查 AT24C02 器件是否正常
{
    LCD_ShowString(60,150,200,16,16,"24C02 Check Failed!");
    delay_ms(200);
    LCD_ShowString(60,150,200,16,16,"Please Check!          ");
    delay_ms(200);
    LED2_ON();
    delay_ms(500);
    LED2_OFF();
    delay_ms(500);
}
LCD_ShowString(60,150,200,16,16,"24C02 Ready!");
POINT_COLOR=BLUE;                                   //设置字体为蓝色
while(1)
{
    key=KEY_Scan(0);
  if(key==key==WKUP_PRES)       //key==WKUP_PRES 按下，写入
24C02
    {
        LCD_ShowString(60,170,200,16,16,"StartWrite24C02....");
        AT24CXX_Write(0,(u8*)TEXT_Buffer,SIZE);
        LCD_ShowString(60,170,200,16,16,"24C02 Write Finished!");
  //提示传送完成
    }
    if(key==KEY2_PRES)
        //KEY2 按下,读取字符串并显示
    {
      LCD_ShowString(60,170,200,16,16,"Start Read 24C02.... ");
        AT24CXX_Read(0,datatemp,SIZE);
```

```
                LCD_ShowString(60,170,200,16,16,"The Data Readed Is:    ");//提
示传送完成
                LCD_ShowString(60,190,200,16,16,datatemp);      //显示读到的字
符串
            }
            i++;
            delay_ms(10);
            if(i==20)
            {
                LED1_ON();
                delay_ms(10);
                LED1_OFF();
                delay_ms(10);
                i=0;
            }
        }
    }
```

17.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"24CXX TEST" //显示的字符串

"24C02 Ready" //显示 24C02 检测 OK

按下 wake_up 按键后，在"24C02 Ready"下方显示

"StartWrite24C02...."写完之后显示

"24C02 Write Finished!"

按下 KEY2 按键后，在"24C02 Ready"下方显示

"The Data Readed Is:"

"STM32F103+24CXX!"

第十八章 FLASH 实验

18.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 W25Q16 的程序编程

18.2 实验原理

SPI 基础知识

SPI 特征

● 3 线全双工同步传输

● 带或不带第三根双向数据线的双线单工同步传输

● 8 或 16 位传输帧格式选择

● 主或从操作

● 支持多主模式

● 8 个主模式波特率预分频系数(最大为 fPCLK/2)

● 从模式频率(最大为 fPCLK/2)

● 主模式和从模式的快速通信

● 主模式和从模式下均可以由软件或硬件进行 NSS 管理：主/从操作模式的动态改变

● 可编程的时钟极性和相位

● 可编程的数据顺序，MSB 在前或 LSB 在前

● 可触发中断的专用发送和接收标志

● SPI 总线忙状态标志

● 支持可靠通信的硬件 CRC

─ 在发送模式下，CRC 值可以被作为最后一个字节发送

─ 在全双工模式中对接收到的最后一个字节自动进行 CRC 校验

● 可触发中断的主模式故障、过载以及 CRC 错误标志

● 支持 DMA 功能的 1 字节发送和接收缓冲器：产生发送和接受请求

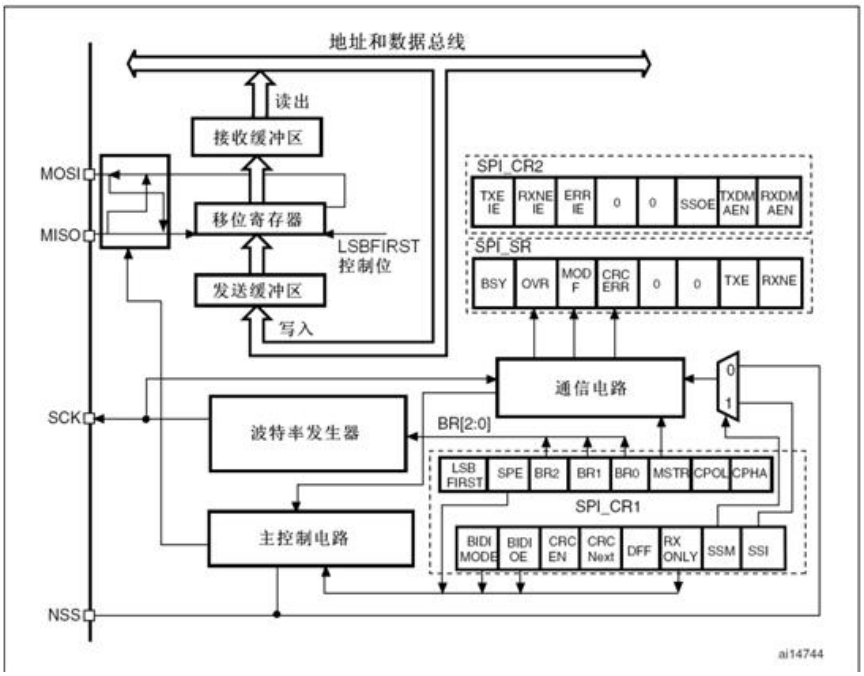


图 **18**.2.1 内部框图

从选择(NSS)脚管理

有 2 种 NSS 模式：图 **18**.2.2 所示

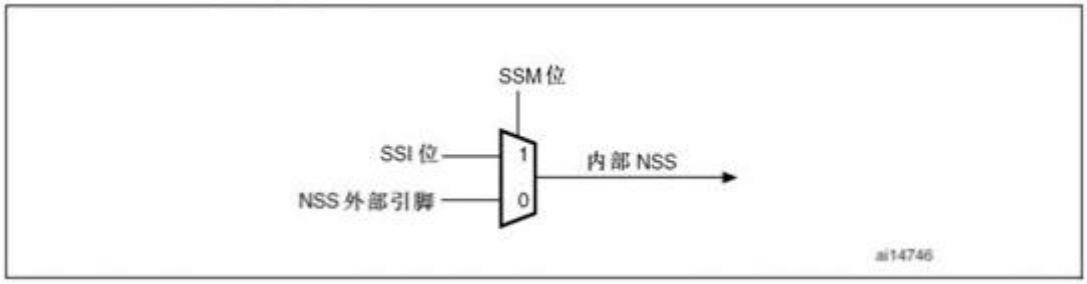


图 **18**.2.2 硬件/软件的从选择管理

● 软件 NSS 模式：可以通过设置 SPI_CR1 寄存器的 SSM 位来使能这种模式(见)。在这种模式下NSS 引脚可以用作它用，而内部NSS 信号电平可以通过写 SPI_CR1 的 SSI 位来驱动

● 硬件 NSS 模式，分两种情况：

—NSS 输出被使能：当 STM32F10xxx 工作为主 SPI，并且 NSS 输出已经通过 SPI_CR2 寄存器的 SSOE 位使能，这时 NSS 引脚被拉低，所有 NSS 引脚与这个主 SPI 的 NSS 引脚相连并配置为硬件 NSS 的 SPI 设备，将自动变成从 SPI 设备。

当一个 SPI 设备需要发送广播数据，它必须拉低 NSS 信号，以通知所有其它的设备它是主设备；如果它不能拉低 NSS，这意味着总线上有另外一个主设备在通信，这时将产生一个硬件失败错误(Hard Fault)。

—NSS 输出被关闭：允许操作于多主环境。

时钟信号的相位和极性可以组合成四种不同的模式，下面为其中来两种模式

图212 数据时钟时序图

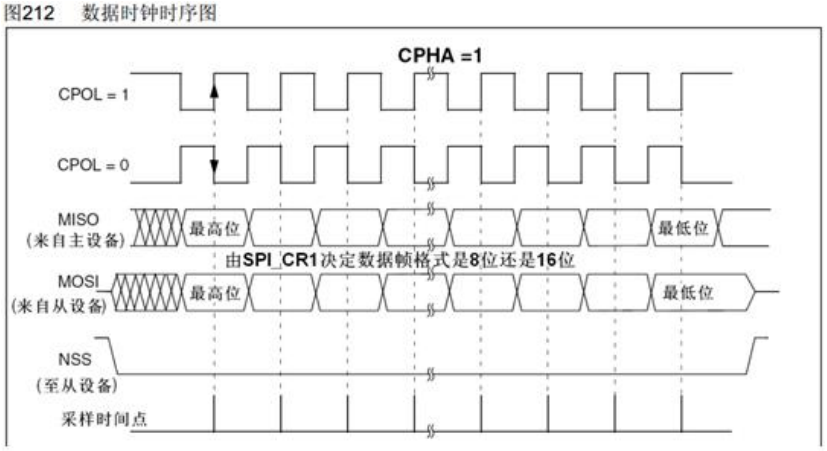


图 **18**.2.2 数据时钟时序图

数据帧格式

根据 SPI_CR1 寄存器中的 LSBFIRST 位，输出数据位时可以 MSB 在先也可以 LSB 在先。

根据 SPI_CR1 寄存器的 DFF 位，每个数据帧可以是 8 位或是 16 位。所选择的数据帧格式对发送和或接收都有效。

配置 SPI 为主模式

在主配置时，在 SCK 脚产生串行时钟。

配置步骤

1. 通过 SPI_CR1 寄存器的 BR[2:0]位定义串行时钟波特率。

2. 选择 CPOL 和 CPHA 位，定义数据传输和串行时钟间的相位关系(见图 212)。

3. 设置 DFF 位来定义 8 位或 16 位数据帧格式。

4. 配置 SPI_CR1 寄存器的 LSBFIRST 位定义帧格式。

5. 如果需要 NSS 引脚工作在输入模式，硬件模式下，在整个数据帧传输期间应把 NSS 脚连接

到高电平；在软件模式下，需设置 SPI_CR1 寄存器的 SSM 位和 SSI 位。如果 NSS 引脚工作

在输出模式，则只需设置 SSOE 位。

6. 必须设置 MSTR 位和 SPE 位(只当 NSS 脚被连到高电平，这些位才能保持置位)。

在这个配置中，MOSI 引脚是数据输出，而 MISO 引脚是数据输入。

SPI 模块能够以两种配置工作于单工方式：

● 1 条时钟线和 1 条双向数据线；

● 1 条时钟线和 1 条数据线(只接收或只发送)；

SPI 通信可以通过以下步骤使用 CRC：

● 设置 CPOL、CPHA、LSBFirst、BR、SSM、SSI 和 MSTR 的值；

● 在 SPI_CRCPR 寄存器输入多项式；

● 通过设置 SPI_CR1 寄存器 CRCEN 位使能 CRC 计算，该操作也会清除寄存器 SPI_RXCRCR

和 SPI_TXCRC；

● 设置 SPI_CR1 寄存器的 SPE 位启动 SPI 功能；

● 启动通信并且维持通信，直到只剩最后一个字节或者半字；

● 在把最后一个字节或半字写进发送缓冲器时，设置 SPI_CR1 的 CRCNext 位，指示硬件在发送完成最后一个数据之后，发送 CRC 的数值。在发送 CRC 数值期间，停止 CRC 计算；

● 当最后一个字节或半字被发送后，SPI 发送 CRC 数值，CRCNext 位被清除。同样，接收到的 CRC 与 SPI_RXCRCR 值进行比较，如果比较不相配，则设置 SPI_SR 上的 CRCERR 标志位，当设置了 SPI_CR2 寄存器的 ERRIE 时，则产生中断。

个人理解：CRC 校验，计算发送端是 CRC 值，接收端计算 CRC 值，然后发送端将计算的 CRC 值发送出去，接收端接收到之后与接收端计算的值相比较，若不同则返回错误标志

利用 DMA 的 SPI 通信

为了达到最大通信速度，需要及时往 SPI 发送缓冲器填数据，同样接收缓冲

器中的数据也必须及时读走以防止溢出。为了方便高速率的数据传输，SPI 实现了一种采用简单的请求/应答的 DMA 机制。

当 SPI_CR2 寄存器上的对应使能位被设置时，SPI 模块可以发出 DMA 传输请求。发送缓冲器和接收缓冲器亦有各自的 DMA 请求(见)。

● 发送时，在每次 TXE 被设置为'1'时发出 DMA 请求，DMA 控制器则写数据至 SPI_DR 寄存器，TXE 标志因此而被清除。

● 接收时，在每次 RXNE 被设置为'1'时发出 DMA 请求，DMA 控制器则从 SPI_DR 寄存器读出数据，RXNE 标志因此而被清除。

当只使用 SPI 发送数据时，只需使能 SPI 的发送 DMA 通道。此时，因为没有读取收到的数据，OVR 被置为'1'(译注：软件不必理会这个标志)。

当只使用 SPI 接收数据时，只需使能 SPI 的接收 DMA 通道

SPI 寄存器

SPI 控制寄存器 1　SPI_CR1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIDI MODE	BIDI OE	CRCEN	CRC NEXT	DFF	RX ONLY	SSM	SSI	LSB FIRST	SPE	BR[2:0]			MSTR	CPOL	CPHA
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

SPI 控制寄存器 2(SPI_CR2)

SPI 状态寄存器(SPI_SR)

SPI 数据寄存器(SPI_DR)

SPI CRC 多项式寄存器

SPI Rx CRC 寄存器(SPI_RXCRCR)

在启用 CRC 计算时，RXCRC[15:0]中包含了依据收到的字节计算的 CRC 数值。当在 SPI_CR1 的 CRCEN 位写入'1'时，该寄存器被复位。CRC 计算使用 SPI_CRCPR 中的多项式。

当数据帧格式被设置为 8 位时，仅低 8 位参与计算，并且按照 CRC8 的方法进行；当数据帧格式为 16 位时，寄存器中的所有 16 位都参与计算，并且按照 CRC16 的标准。

SPI Tx CRC 寄存器(SPI_TXCRCR)

在启用 CRC 计算时，TXCRC[15:0]中包含了依据将要发送的字节计算的 CRC 数值。当在 SPI_CR1 中的 CRCEN 位写入'1'时，该寄存器被复位。CRC 计算使

用 SPI_CRCPR 中的多项式。

当数据帧格式被设置为 8 位时，仅低 8 位参与计算，并且按照 CRC8 的方法进行；当数据帧格式为 16 位时，寄存器中的所有 16 个位都参与计算，并且按照 CRC16 的标准。

18.3 实验例程

（参考开源实验例程）实验十五 FLASH 实验

18.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "25WXX.h"
#include "spi.h"
#include "systick.h"
#include "usart_printf.h"
#include "key.h"

#define SIZE sizeof(TEXT_Buffer)
typedef enum { FAILED = 0, PASSED = !FAILED} TestStatus;
const u8 TEXT_Buffer[]={"FLASH TEST IS OK"};
void RCC_Configuration(void);

/*************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输 入：无
* 输 出：无
* 返 回 值 ：无
*************************************************************/
int main(void)
```

```
{
  u8 key;
    u16 i=0;
    u8 datatemp[SIZE];
    u32 FLASH_SIZE;
    RCC_Configuration();                    //RCC 配置初始化
    SysTick_Init();                         //滴答定时器配置初始化
    GPIO_Configuration();                   //GPIO 初始化
    uart_init(9600);                        //串口初始化 9600
    LCD_Init();                             //LCD 初始化
    KEY_Init();                             //按键配置初始化
    SPIx_Init();                            //PSI 配置初始化
    LCD_Clear(CYAN);                        //清屏为淡蓝色
    POINT_COLOR=BLUE;
    LCD_ShowString(40,50,200,16,16,"Layout & LCEDA");
    LCD_ShowString(40,70,200,16,16,"FLASH TEST");
    while(SPI_Flash_ReadID()!=W25Q16)      //检测不到 W25Q16
    {
        LCD_ShowString(40,150,200,16,16,"25Q16 Check Failed!");
        LED1_ON();
        delay_ms(500);
        LCD_ShowString(40,150,200,16,16,"Please Check!          ");
        LED1_OFF();
        delay_ms(500);
    }
    LCD_ShowString(40,150,200,16,16,"25Q16 Ready!");
    FLASH_SIZE=2*1024*1024;//FLASH  大小为 2M 字节
  while(1)
    {
```

```
        key=KEY_Scan(0);
          if(key==WKUP_PRES)   //WKUP_PRES 按下,写入 W25Q16
          {
              LCD_ShowString(40,170,200,16,16,
"Start Write W25Q16....");
              SPI_Flash_Write((u8*)TEXT_Buffer,FLASH_SIZE-100,SIZE);
//从倒数第 100 个地址处开始,写入 SIZE 长度的数据
              LCD_ShowString(40,170,200,16,16,
"W25Q16 Write Finished!");  //提示传送完成
          }
          if(key==KEY2_PRES)    //KEY2 按下,读取字符串并显示
          {
              LCD_ShowString(40,170,200,16,16,"Start Read W25Q16... ");
              SPI_Flash_Read(datatemp,FLASH_SIZE-100,SIZE);
//从倒数第 100 个地址处开始,读出 SIZE 个字节
              LCD_ShowString(40,170,200,16,16,"The Data Readed Is:     "); //
提示传送完成
              LCD_ShowString(40,190,200,16,16,datatemp);                  //
显示读到的字符串
          }
          i++;
          delay_ms(10);
          if(i==20)
          {
              LED2_ON();              //提示系统正在运行
              delay_ms(100);
              LED2_OFF();
              delay_ms(100);
              i=0;
```

```
        }
    }
```

18.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"FLASH TEST" //显示的字符串

"25Q16 Ready" //显示 25Q16 检测 OK

按下 wake_up 按键后，在"25Q16 Ready"下方显示

"Start Write W25Q16...."写完之后显示

"W25Q16 Write Finished!"

按下 KEY2 按键后，在"25Q16 Ready"下方显示

"The Data Readed Is:"

"FLASH TEST IS OK"

第十九章 SD 实验

19.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 SD 读写的程序编程

19.2 实验原理

SD 卡简介

SD 卡（Secure Digital Memory Card）中文翻译为安全数码卡，它是在 MMC 的基础上发展而来，是一种基于半导体快闪记忆器的新一代记忆设备，它被广泛地于便携式装置上使用，例如数码相机、个人数码助理(PDA)和多媒体播放器等。SD 卡由日本松下、东芝及美国 SanDisk 公司于 1999 年 8 月共同开发研制。大小犹如一张邮票的 SD 记忆卡，重量只有 2 克，但却拥有高记忆容量、快速数据传输率、极大的移动灵活性以及很好的安全性。按容量分类，可以将 SD 卡分为 3 类：SD 卡、SDHC 卡、SDXC 卡。如表 **19**.2.1 所示：

容量	命名	简称
0~2G	Standard Capacity SD Memory Card	SDSC 或 SD
2G~32G	High Capacity SD Memory Card	SDHC
32G~2T	Extended Capacity SD Memory Card	SDXC

图 **19**.2.1 SD 卡按容量分类

SD 卡和 SDHC 卡协议基本兼容，但是 SDXC 卡，同这两者区别就比较大了，本章我们讨论的主要是 SD/SDHC 卡（简称 SD 卡）。

SD 卡一般支持 2 种操作模式：

1，SD 卡模式（通过 SDIO 通信）；

2，SPI 模式；

主机可以选择以上任意一种模式同 SD 卡通信，SD 卡模式允许 4 线的高速数据传输。SPI 模式允许简单的通过 SPI 接口来和 SD 卡通信，这种模式同 SD 卡模式相比就是丧失了速度。

SD 卡的引脚排序如下图 19.2.2 所示：

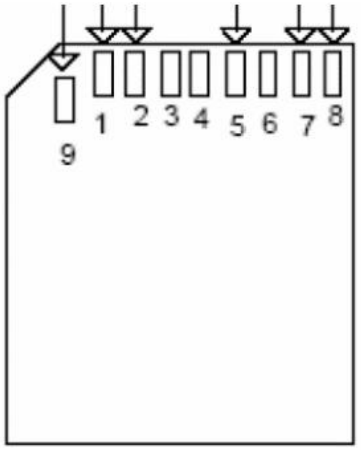


图 19.2.2 SD 卡引脚排序图

SD 卡引脚功能描述如表 19.2.3 所示：

针脚	1	2	3	4	5	6	7	8	9
SD卡模式	CD/DAT3	CMD	VSS	VCC	CLK	VSS	DATO	DAT1	DAT2
SPI模式	CS	MOSI	VSS	VCC	CLK	VSS	MISO	NC	NC

表 19.2.3 SD 卡引脚功能表

SD 卡只能使用 3.3V 的 IO 电平，所以，MCU 一定要能够支持 3.3V 的 IO 端口输出。注意：在 SPI 模式下，CS/MOSI/MISO/CLK 都需要加 10~100K 左右的上拉电阻。

SD 卡寄存器及配置请参考 Datasheet。

19.3 实验例程

（参考开源实验例程）实验十六 SD 卡实验

19.4 实验程序配置解析

```
#include "GPIO.h"
#include "key.h"
#include "lcd.h"
#include "usart_printf.h"
#include "MMC_SD.h"
#include "systick.h"
```

```
#include "spi.h"
#include "stm32f10x.h"
#include <stdio.h>
#include "malloc.h"
void RCC_Configuration(void);
/*****************************************************************
* 函数名称 : main.c
* 功能介绍  ：主函数
* 输      入：无
* 输      出：无
* 返 回 值  ：无
*****************************************************************/
 int main(void)
 {
    u8 key;
    u32 sd_size;
    u8 t=0;
    u8 *buf;
    RCC_Configuration();            //RCC 初始化
    SysTick_Init();                 //滴答定时器配置初始化
    uart_init(9600);                //串口初始化
    GPIO_Config();                  //LED 端口初始化
    LCD_Init();                     //初始化 LCD
    key_init();                     //按键初始化
    mem_init(0);                        //初始化内存池
    LCD_Clear(CYAN);                    //清屏为淡蓝色
    POINT_COLOR=BLACK;                  //设置字体为黑色
    LCD_ShowString(60,50,200,16,16,"Layout & LCEDA");
    LCD_ShowString(60,70,200,16,16,"SD CARD TEST");
```

```
        while(SD_Initialize())//检测不到 SD 卡
        {
            LCD_ShowString(60,150,200,16,16,"SD Card Error!");
            LED1_ON();
            delay_ms(500);
            LCD_ShowString(60,150,200,16,16,"Please Check! ");
            LED1_OFF();
            delay_ms(500);
        }
        POINT_COLOR=BLUE;                    //设置字体为蓝色
    /******************** 检测 SD 卡成功 ************************/
        LCD_ShowString(60,150,200,16,16,"SD Card OK      ");
        LCD_ShowString(60,170,200,16,16,"SD Card Size:       MB");
        sd_size=SD_GetSectorCount();              //得到扇区数
        LCD_ShowNum(164,170,sd_size>>11,5,16);//显示 SD 卡容量
        while(1)
        {
            key=KEY_Scan(0);
            if(key==KEY_WAKE_UP)                  //WAKE_UP 按下了
            {
                buf=mymalloc(0,512);              //申请内存
                LED2_ON();
                if(SD_ReadDisk(buf,0,1)==0)//读取 0 扇区的内容
                {
                    LCD_ShowString(60,190,200,16,16,"USART1          Sending
Data...");
                    printf(" SECTOR 0 DATA:\r\n ");
                    for(sd_size=0;sd_size<512;sd_size++)printf("%x
```

```
",buf[sd_size]);//打印 0 扇区数据
                    printf("\r\nDATA ENDED\r\n");
                    LCD_ShowString(60,190,200,16,16,"USART1 Send Data Over!");
                }
                LED2_OFF();
                myfree(0,buf);              //释放内存
            }
            t++;
            delay_ms(10);
            if(t==20)
            {
                LED2_ON();
                delay_ms(100);
                LED2_OFF();
                t=0;
            }
        }
    }
```

19.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"SD CARD TEST" //显示的字符串

"SD Card OK" //显示 SD 卡检测 OK

"SD Card Size:1876MB" //显示 SD 卡大小

按下 wake_up 按键后，在"SD Card OK"下方先显示

"USART1 Sending Data..."，后显示"USART1 Send Data Over!"，同时打开串口可以看到串口调试助手接收到的数据。

第二十章 DS18B20 实验

20.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 DS18B20 数字温度传感器的程序编程

20.2 实验原理

1.DS18B20 是 Dallas 公司生产的数字温度传感器，具有体积小、适用电压宽、经济灵活的特点。它内部使用了 onboard 专利技术，全部传感元件及转换电路集成在一个形如三极管的集成电路内。DS18B20 有电源线、地线及数据线 3 根引脚线，工作电压范围为 3～5.5 V，支持单总线接口。

2.1DS18B20 的内外结构

DS18B20 的外部结构如图 20.2.1 所示。其中，VDD 为电源输入端，DQ 为数字信号输入 / 输出端，GND 为电源地。

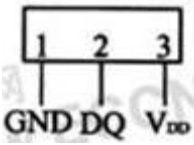


图 20.2.1 DS18B20 外部结构

DS18B20 内部结构主要包括 4 部分：64 位光刻 ROM、温度传感器、非易失的温度报警触发器 TH 和 TL、配置寄存器，如图 20.2.2 所示。

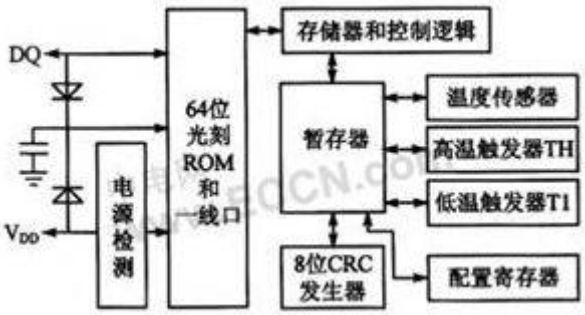


图 20.2.2 DS18B20 内部结构

64 位 ROM 中，在产品出厂前就被厂家通过光刻刻录好了 64 位序列号。该序列号可以看作是 DS18B20 的地址序列码，用来区分每一个 DS18B20，从而更好地实现对现场温度的多点测量。

图 22.2.2 中的暂存器是 DS18B20 中最重要的寄存器。暂存器由 9 个字节组成，各字节定义如表 20.2.3 所列。

字 节	定 义
0	所测温度值低 8 位
1	所测温度值高 8 位
2	高温报警值(TH)
3	低温报警值(TL)
4	配置寄存器
5～7	保留
8	循环冗余校验(CRC)值

表 20.2.3 暂存器字节分配表

配置寄存器用于用户设置温度传感器的转换精度，其各位定义如图 20.2.4 所示：

TM	R1	R0	1	1	1	1	1

图 20.2.4 转换精度

TM 位是测试模式位，用于设置 DS18B20 是工作模式(0)还是测试模式(1)，其出厂值为 0。R1、R0 用于设置温度传感器的转换精度：00，分辨率为 9 位，转换时间为 93.75ms；01，分辨率为 10 位，转换时间为 187.5 ms；10，分辨率为 11 位，转换时间为 375 ms；11，分辨为 12 位，转换时间为 750 ms。R1、R0 的出厂值为 11。其余 5 位值始终为 1。

第 0 和第 1 字节为 16 位转换后的温度二进制值，其中前 4 位为符号位，其余 12 位为转换后的数据位(分辨率为 12 位)。如果温度大于 0，则前 4 位值为 0，只要将测到的数值乘上 0.062 5 即可得到实际温度值；如果温度小于 0，则前 4 位为 1，需将测得的数值取反加 1 后，再乘上 0.062 5。第 0 和第 1 字节各位的二进制值如图 20.2.5 所示：

LSB(0)	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
HSB(1)	S	S	S	S	2^7	2^6	2^5	2^4

图 20.2.5 二进制

20.3 实验例程

（参考开源实验例程）实验十七 DS18B20 实验

20.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "usart_printf.h"
#include <stdio.h>
#include "systick.h"
#include "ds18b20.h"

void RCC_Configuration(void);
/*************************************************************
* 函数名称 : main.c
* 功能介绍 ：主函数
* 输　　入：无
* 输　　出：无
* 返 回 值 ：无
*************************************************************/
int main(void)
{
    u8 t=0;
    short temperature;
    SysTick_Init();              //嘀嗒定时器初始化
    RCC_Configuration();         //系统时钟初始化
    GPIO_Configuration();        //LED_GPIO 引脚配置初始化
    uart_init(9600);             //串口初始化
    LCD_Init();                  //LCD 初始化
    LCD_Clear(CYAN);             //清屏为淡蓝色
    POINT_COLOR=RED;             //设置字体为红色
    LCD_ShowString(60,50,200,16,16,"layout & LCEDA");
```

```
    LCD_ShowString(60,70,200,16,16,"DS18B20 TEST");
    while(DS18B20_Init())    //DS18B20 初始化
    {
        LCD_ShowString(60,130,200,16,16,"DS18B20 Error");
        LED2_ON();
        delay_ms(200);
        LCD_Fill(60,130,239,130+16,WHITE);
        LED2_OFF();
        delay_ms(200);
    }
    LCD_ShowString(60,130,200,16,16,"DS18B20 OK");
    POINT_COLOR=BLUE;                                    //设置字体为蓝色
    LCD_ShowString(60,150,200,16,16,"Temp:      . C");
  while(1)
    {
        if(t%10==0)//每 100ms 读取一次
        {
            temperature=DS18B20_Get_Temp();
            if(temperature<0)
            {
                LCD_ShowChar(60+40,150,'-',16,0);
    //显示负号
                temperature=-temperature;
            //转为正数
            }else LCD_ShowChar(60+40,150,' ',16,0);                    //
去掉负号
            LCD_ShowNum(60+40+8,150,temperature/10,2,16);              //
显示正数部分
            LCD_ShowNum(60+40+32,150,temperature%10,1,16); //显示小数
```

部分

```
            }
            delay_ms(10);
            t++;
            if(t==20)
            {
                t=0;
                LED1_ON();              //LED1 点亮
                delay_ms(100);
                LED1_OFF();         //LED1 熄灭
                delay_ms(100);
            }
        }
    }
```

20.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

"Layout & LCEDA" //显示的字符串

"DS18B20 TEST" //显示的字符串

"DS18B20 OK" //显示 DS18B20 检测 OK

"Temp:25.6C" //显示当前室温 25.6C

用手触碰 DS18B20,可以观察到 LCD 显示的温度值会增加。

第二十一章 HS0038 实验

21.1 实验目的

1.熟悉 STM32 的编程环境的使用

2.掌握 STM32 红外遥控 HS0038 的程序编程

21.2 实验原理

红外遥控是一种无线、非接触控制技术，具有抗干扰能力强，信息传输可靠，功耗低，成本低，易实现等显著优点，被诸多电子设备特别是家用电器广泛采用，并越来越多的应用到计算机系统中。

由于红外线遥控不具有像无线电遥控那样穿过障碍物去控制被控对象的能力，所以，在设计红外线遥控器时，不必要像无线电遥控器那样，每套(发射器和接收器)要有不同的遥控频率或编码(否则，就会隔墙控制或干扰邻居的家用电器)，所以同类产品的红外线遥控器，可以有相同的遥控频率或编码，而不会出现遥控信号“串门”的情况。这对于大批量生产以及在家用电器上普及红外线遥控提供了极大的方面。由于红外线为不可见光，因此对环境影响很小，再由红外光波动波长远小于无线电波的波长，所以红外线遥控不会影响其他家用电器，也不会影响临近的无线电设备。

红外遥控的编码方式目前广泛使用的是：PWM(脉冲宽度调制)的 NEC 协议和 Philips PPM(脉冲位置调制) 的 RC-5 协议的。ALIENTEK MiniSTM32 高教板配套的遥控器使用的是 NEC 协议，其特征如下：

1、8 位地址和 8 位指令长度；

2、地址和命令 2 次传输（确保可靠性）

3、PWM 脉冲位置调制，以发射红外载波的占空比代表“0”和“1”；

4、载波频率为 38Khz；

5、位时间为 1.125ms 或 2.25ms；

NEC 码的位定义：一个脉冲对应 560us 的连续载波，一个逻辑 1 传输需要 2.25ms（560us 脉冲+1680us 低电平），一个逻辑 0 的传输需要 1.125ms（560us 脉冲+560us 低电平）。而遥控接收头在收到脉冲的时候为低电平，在

没有脉冲的时候为高电平，这样，我们在接收头端收到的信号为：逻辑 1 应该是 560us 低+1680us 高，逻辑 0 应该是 560us 低+560us 高。

NEC 遥控指令的数据格式为：同步码头、地址码、地址反码、控制码、控制反码。同步码由一个 9ms 的低电平和一个 4.5ms 的高电平组成，地址码、地址反码、控制码、控制反码均是 8 位数据格式。按照低位在前，高位在后的顺序发送。采用反码是为了增加传输的可靠性（可用于校验）。

我们遥控器的按键 ok 按下时，从红外接收头端收到的波形如图 21.2.1 所示：

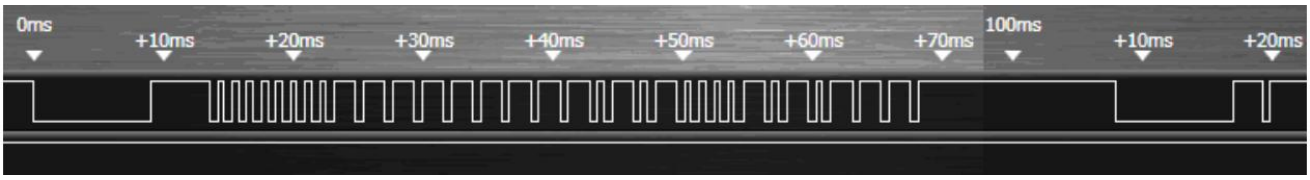


图 21.2.1 按键 OK 所对应的红外波形

从图 21.2.1 中可以看到，其地址码为 0，控制码为 168。可以看到在 100ms 之后，我们还收到了几个脉冲，这是 NEC 码规定的连发码(由 9ms 低电平+2.5m 高电平+0.56ms 低电平+97.94ms 高电平组成)，如果在一帧数据发送完毕之后，按键仍然没有放开，则发射重复码，即连发码，可以通过统计连发码的次数来标记按键按下的长短/次数。

21.3 实验例程

（参考开源实验例程）实验十八 HS0038 实验

21.4 实验程序配置解析

```
#include "stm32f10x.h"
#include "GPIO.h"
#include "lcd.h"
#include "usart_printf.h"
#include <stdio.h>
#include "systick.h"
#include "remote.h"

void RCC_Configuration(void);
```

```
/**************************************************************************
* 函数名称 : main.c
* 功能介绍  : 主函数
* 输    入 : 无
* 输    出 : 无
* 返 回 值  : 无
**************************************************************************/
int main(void)
{
    u8 key;
    u8 t=0;
    u8 *str=0;
    SysTick_Init();             //嘀嗒定时器初始化
    RCC_Configuration();        //系统时钟初始化
    GPIO_Configuration();   //LED_GPIO 引脚配置初始化
    Remote_Init();              //红外接收初始化
    uart_init(9600);            //串口初始化
    LCD_Init();                 //LCD 初始化
    POINT_COLOR=RED;                //设置字体为红色
    LCD_ShowString(60,50,200,16,16,"layout & LCEDA");
    LCD_ShowString(60,70,200,16,16,"HS0038 TEST");

    LCD_ShowString(60,90,200,16,16,"KEYVAL:");      //显示按键值
   LCD_ShowString(60,110,200,16,16,"KEYCNT:");      //显示按键次数
   LCD_ShowString(60,130,200,16,16,"SYMBOL:");      //按键号
  while(1)
    {
        key=Remote_Scan();
        if(key)
```

```
{
    LCD_ShowNum(116,90,key,3,16);          //显示键值
    LCD_ShowNum(116,110,RmtCnt,3,16);    //显示遥控按键信息
    switch(key)
    {
        case 0:str="ERROR";break;
        case 162:str="POWER";break;
        case 226:str="MENU";break;
        case 34:str="TEST";break;
        case 2:str="+";break;
        case 194:str="DELETE";break;
        case 224:str="LEFT";break;
        case 168:str="OK";break;
        case 144:str="RIGHT";break;
        case 152:str="-";break;
        case 176:str="C";break;
        case 104:str="0";break;
        case 48:str="1";break;
        case 24:str="2";break;
        case 122:str="3";break;
        case 16:str="4";break;
        case 56:str="5";break;
        case 90:str="6";break;
        case 66:str="7";break;
        case 74:str="8";break;
        case 82:str="9";break;
    }
    LCD_Fill(116,130,116+8*8,170+16,WHITE);//清楚之前的显示
    LCD_ShowString(116,130,200,16,16,str);  //显示遥控按键信息
```

```
        }else delay_ms(10);
        t++;
        if(t==20)
        {
            t=0;
            LED1_ON();              //LED1 点亮
            delay_ms(100);
            LED1_OFF();         //LED1 熄灭
            delay_ms(100);
        }
    }
}
```

21.5 实验结果

在代码编译成功之后，我们通过下载代码到高教板上，可以看到 LCD 依次显示：

```
"Layout & LCEDA"            //显示的字符串
"HS0038 TEST"               //显示的字符串
"KEYVAL:"                   //显示按键值
"KEYCNT:"                   //显示按键次数
"SYMBOL:"                   //按键号
```

当我们按下遥控器按键是，分别会显示当前按键值，按键次数，按键号信息。

至此，整个高教板的基础实验就介绍完了。很多是移植的开源程序代码，希望我们的这个代码，可以让大家有所受益，能帮助大家学习，提高对电子产品的兴趣。

本高教板实验代码比较简单，偏向于基础实验，希望可以给零基础的电子爱好者提供一个好的学习平台及学习案例，最后祝大家身体健康、学习进步！