《微控制器原理与应用开发实践》

指导书

先进制造实验中心

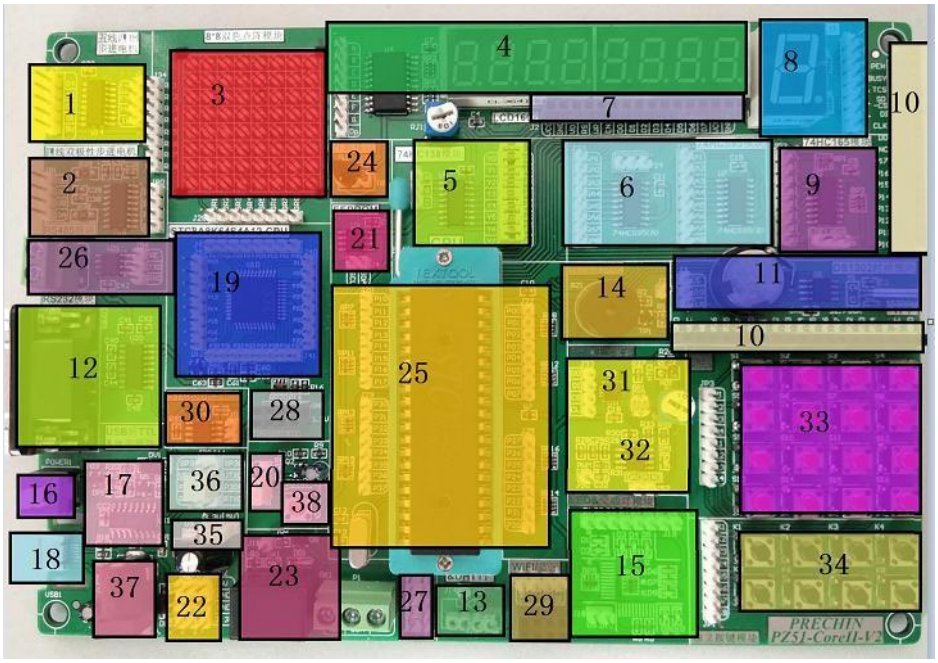
2022.3

目录

单片机开发板说明

1 开发板

开发板采用的是双 CPU 设计，分别是 STC89C516 和 STC8A8K64S4A12 芯片，它们都是 51 内核的单片机，不过 STC 公司最新的 STC8A8K64S4A12 芯片的处理速度比 STC89C516 快 12 倍左右，而且还自带在线仿真功能，非常实用，而且本款开发板各模块独立，即拥有了此款开发板就拥有了两块开发板--STC89C516 开发板和 STC8A8K64S4A12 开发板。



序号	模块	功能描述
1	五线四相步进电机驱动模块	使用 ULN2003 芯片，可驱动直流电机、五线四相步进电机等
2	四线双极性步进电机/直流电机驱动模块	使用 TC1508S 芯片，可驱动直流电机（正反转控制）、四线双极性进电机等
3	8*8 双色 LED 点阵模块	可独立控制双色点阵显示数字、字符、简单汉字图形等
4	动态数码管模块	使用 74HC245 芯片驱动 2 个四位一体共阴数码管
5	74HC138 译码器模块	使用 74HC138 译码器，默认用于控制动态数码管位选

6	74HC595 模块	使用 2 块 74HC595 芯片级联扩展 IO，可用来控制 8*8LED 双色点阵、数码管
7	LCD1602 液晶接口	连接 LCD1602 液晶屏
8	静态数码管模块	使用一位共阳数码管
9	74HC165 模块	使用 74HC165 芯片，可实现 IO 口扩展（并转串） 功能
10	LCD12864/TFT 彩屏接口	可兼容不带字库 LCD1286、带字库 LCD12864 以及 TFTLCD 彩屏
11	DS1302 时钟模块	使用 DS1302 时钟芯片， 可实现数字时钟功能
12	RS232 模块	使用 MAX232 芯片，可实现 RS232 串口通信及程序下载
13	DS18B20&DHT11 接口	可兼容 DS18B20 温度传感 器和 DHT11 温湿度传感器
14	蜂鸣器模块	使用无源蜂鸣器，可实现 报警提示等功能
15	LED&交通灯模块	使用 10 个小灯（红黄绿），按照交通灯模型排列，既可实现 LED 流水灯控制又 可实现交通灯控制
16	电源开关	控制系统电源
17	USB 转 TTL 串口模块	使用 CH340 芯片，可实现 USB 转 TTL 串口功能，既可下载程序，又可实现串 口通信
18	Mini USB 接口	既可支持配置的 USB 线又可支持安卓手机 USB 数据线
19	STC8A8K64S4A12-CPU 核心模块	增强型 51 单片机，处理速度是 STC89C5x 的 12 倍左右，具备大存储容量， 64KB FLASH 和 8KB SRAM， 自带在线仿真功能，内部 含有丰富资源外设，如 ADC、SPI、USART、TIME、PWM、I2C、INT、CAP 等

20	单片机复位选择切换端子	51 单片机是高电平复位， 所以选择短接片短接到 H 侧，对于 AVR 等低电平复位的单片机，将短接片短接到 L 侧
21	EEPROM 模块	使用 AT24C02 芯片，存储容量为 256 字节，可实现 IIC-EEPROM 功能，存储的数据掉电不丢失
22	电源输出端子	可输出 5V 和 3.3V 电压供外部使用
23	继电器模块	使用直流继电器，建议在低压段控制，不要使用高压 220V，以免造成人身安全问题
24	红外接收模块	使用一体化红外接收头，可实现红外遥控通信
25	STC89C516 单片机/ARM 核心板/AVR 核心板接口座和 IO 管脚	可固定单片机，并将单片机 IO 口全部引出，方便用户二次开发
26	RS485 模块	使用 MAX485 芯片，可实现 RS485 通信
27	ISP 接口	使用 AVR 或 AT 芯片时，可实现 ISP 下载
28	STC8A 核心内 NRF24L01 模块接口	支持 NRF24L01 模块，可实现 2.4G 无线通信
29	WIFI/蓝牙模块接口	使用 PZ-ESP8266-WIFI 模块或者 PZ-HC05 蓝牙模块，配合 APP 可实现 WIFI 或者蓝牙无线控制
30	STC8A 核心内 RS485 模块	使用 MAX485 芯片，该模块属于 STC8A 核心的 RS485 模块，可与 STC89C516 内的 RS485 模块相互通信
31	ADC 模块	使用 XPT2046 芯片，可实现模拟信号采集转换，可设计简易电压表等
32	DAC(PWM)模块	使用 LM358 芯片，可实现模拟信号输出、PWM 控制
33	矩阵键盘模块	使用 4*4 矩阵键盘，可实现键盘输入控制
34	独立按键模块	使用 2*4 按键，可实现按键控制

35	系统电源切换端子	3.3V 和 5V 电源切换，使 用 51 单片机要切换到 5V，默认也是短接到 5V 端， 使用 STM32 时可切换至 3.3V 端
36	USB 转 TTL 串口&RS232 模块&STC8A 单片机下载切换端子	①如果使用 USB 转 TTL 模块给 STC89C516 进行串口通信或下载时，短接片短接到中间端（URXD->P31T，UTXD->P30R），默认出厂就是短接到该处；②如果使用 USB 转 TTL 模块给 STC8A8K64S4A12 进行串口通信或下载时，短接片短接到上端（URXD->SP31，UTXD->SP30）；③如果使用 RS232 模块给 STC89C516 进行串口通信或下载时，短接片短接到下端（RRXD->P31T，RTXD->P30R）；
37	火牛接口	可接入 DC5V 电压，切勿超过此电压，否则烧坏开发板上芯片
38	复位按键	系统复位按钮

2.开发板使用方法

（1）参考普中 51 单片机开发攻略——

2.2 开发板使用方法 2.2.1 CH340 驱动安装

2.2.2 程序的烧录（下载）（将开发板上的 USB 转 TTL 模块的 J39 和 J44 端子使用短接 片短接好，出厂的时候开发板默认已经短接好（URXD 与 P31T，UTXD 与 P30R）， 用户无需改动）

自学 2.2.3 仿真器调试

（2）参考普中 STM32 开发攻略，学习 STM32 芯片的使用。

3.注意事项

实验过程中应先按实验指导书连线，连线完毕后再开电源。

4. 实验要求

（1）上机时每人自带 U 盘一个，实验结束时把自己的代码拷贝，带回去打印。

（2）按照给定的实验报告格式要求填写个人上机基本信息、实验步骤，打印实验结果；将实验报告格式要求和打印出来的代码用胶水粘在一起，每一位同学就交两张纸（A4 纸）。

（3）实验报告上交规定：在每一次上机前交上一周所作实验报告，由学习委员收齐后按学号顺序排好后交给实验室老师，最后一次实验报告按规定时间交到办公室。

实验一 直流电动机控制实验

一、实验目的

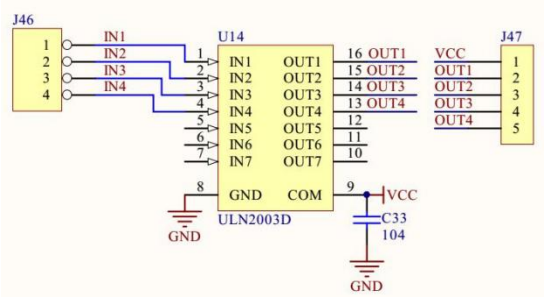
1．学习用直流电机；

2．熟悉直流电动机的工作特性。

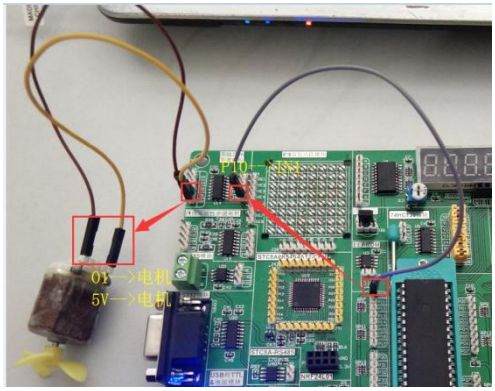
二、实验说明

本实验（2 课时）以 51 单片机为例进行说明，具体如下：

本实验使用到硬件资源如下： （1）五线四项步进电机驱动模块 我们开发板上的五线四项步进电机驱动模块电路如下图所示：



从上图中可以看出，该电路是独立的，芯片的输入通过 J46 端子提供，芯片 的输出由 J47 端子引出。J46 输入对应 J47 输出。 由于该模块电路是独立的，所以使用任意单片机管脚都可以，为了与我们例 程程序配套，这里使用 P1.0 管脚来控制 ULN2003 输出，即将 P1.0 管脚连接到 J46 的 IN1，直流电机的两根线接 J47 端子的 1、2 脚即 5V 和 O1。其对应的连接 图如下所示（具体可参考实验现象接线图）：注意：直流电机无方向，线接反只是转动方向不同而已。




```
#include "reg52.h"
#include<intrins.h>
typedef unsigned int u16;
typedef unsigned char u8;
sbit moto=P1^0;
void delay(u16 i)
{
while(i--);
```

```
}
void main()
{
u8 i;
moto=0; //关闭电机
for(i=0;i<100;i++) //循环 100 次，也就是大约 5S
{
moto=1; //开启电机
delay(5000); //大约延时 50ms
}
moto=0; //关闭电机
while(1)
{
}}
```

主函数功能非常简单，首先定义变量 i 用于循环次 数累计，初始时关闭电机即使 moto 为 0，对应输出就为高电平，电机不工作。 然后使用 for 循环 100 次，开启电机，每次约延时 50ms，for 循环结束后关闭电 机。最后进入 while 循环。 延时时间长短可通过修改 i 值或者 delay 实现，但是要注意 i 值的变量类型， 不能超过 255，如果超过可使用 u16 类型来定义。

三、实验内容及步骤

1．使用 USB 线将开发板和电脑连接成功后（电脑能识别开发板上 CH340 驱动串 口），把编写好的程序编译后将编译产生的.hex 文件烧入到芯片内，按照如下 图所示的接线方式可以看到直流电机旋转约 5 秒后停止。

四、作业

1、使用独立按键控制直流电机的转动和停止。

2、提交实验步骤页+作业 1 的代码打印。

实验二　步进电动机控制实验

一、实验目的

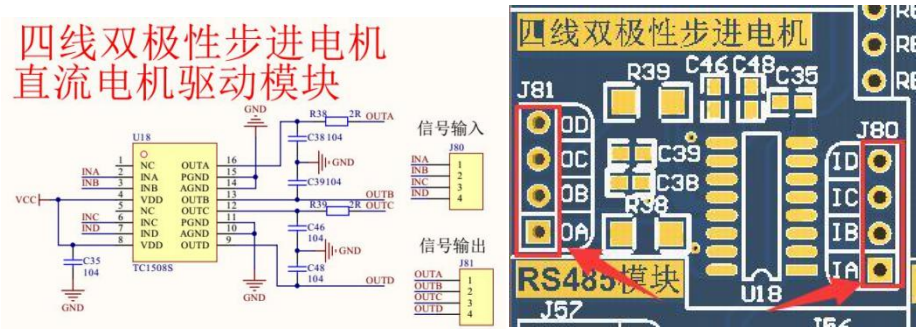
1．掌握采用单片机控制步进电机的硬件接口技术。

2．掌握步进电机驱动程序的设计和调试方法。

二、实验说明

本实验（2 课时）以 51 单片机为例进行说明，具体如下：

本章所要实现的功能是：控制步进电机旋转。使用到硬件资源如下：　（1）四线双极性步进电机驱动模块 我们开发板上的四线双极性步进电机驱动模块电路如下图所示：



从上图中可以看出，该电路是独立的，芯片的输入通过 J80 端子提供，芯片 的输出由 J81 端子引出。由于该模块电路是独立的，所以使用任意单片机管脚都 可以，为了与我们例程程序配套，这里使用 P1.0-P1.3 管脚来控制 TC1508S 输入， 即将 P1.0-P1.3 管脚依次连接到 J80 的 INA、INB、INC、IND 管脚，四线双极性 步进电机的 4 根线接到 J81 端子，即 A-接 OUTB，A+接 OUTA，B-接 OUTD，B+接 OUTC。其对应的连接图如下所示（具体可参考实验现象接线图）：注意：步进电机的 4 条控制线如果与 J81 接反，其旋转方向变为反向。

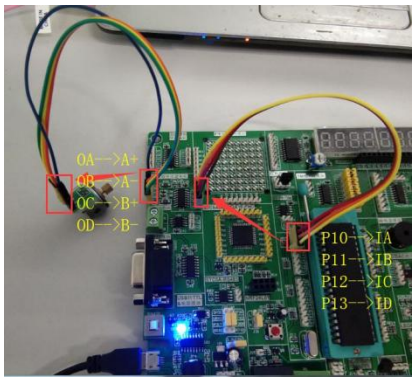
```
/**********************
实验现象：下载程序后，步进电机旋转
1，单片机-->四线双极性步进电机模块
P10-->IA
P11-->IB
P12-->IC
P13-->ID
2，四线双极性步进电机模块输出-->步进电机
OA-->A+
OB-->A-
OC-->B+
OD-->B-
********/
#include "reg52.h"
#include<intrins.h>
```

```
typedef unsigned int u16;
typedef unsigned char u8;
sbit MOTOA = P1^0;
sbit MOTOB = P1^1;
sbit MOTOC = P1^2;
sbit MOTOD = P1^3;
#define SPEED 200
//修改此值可改变电机旋转速度，不能过大或过小
void delay(u16 i)
{
while(i--);
}
void main()
{
P1=0X00;
while(1)
{
MOTOA = 1;
MOTOB = 0;
MOTOC = 1;
MOTOD = 1;
delay(SPEED);
MOTOA = 1;
MOTOB = 1;
MOTOC = 1;
MOTOD = 0;
delay(SPEED);
MOTOA = 0;
MOTOB = 1;
MOTOC = 1;
MOTOD = 1;
delay(SPEED);
MOTOA = 1;
MOTOB = 1;
MOTOC = 0;
MOTOD = 1;
delay(SPEED);
} }
```

三、实验内容及步骤

1．使用 USB 线将开发板和电脑连接成功后（电脑能识别开发板上 CH340 驱动串 口），把编写好的程序编译后将编译产生的.hex 文件烧入到芯片内，按照如下 图所示的接线方式可以看到步进电机不停的

在旋转。



四、作业

1、使用独立按键控制步进电机的正反转和停止。 （温馨提示：电机正反转控制即是让 P1 口输入与之前电平相反的时序）

2、提交实验步骤页+作业 1 的代码打印。

实验三 DS18B20 温度传感器实验

一、实验目的

1．掌握读取 DS18B20 采集温度的方法，通过看时序图能实现单总线控制 DS18B20 进行温度读取。

2．掌握将读取的温度通过数码管显示的方法。

二、实验说明

本实验（4 课时）以 51 单片机为例进行说明，具体如下：

1、DS18B20 介绍

DS18B20 是由 DALLAS 半导体公司推出的一种的“一线总线（单总线）”接口的温度传感器。与传统的热敏阻等测温元件相比，它是一种新型的体积小、适用电压宽、与微处理器接口简单的数字化温度传感器。

DS18B20 温度传感器具有如下特点：

（1）适应电压范围更宽，电压范围：3.0～5.5V，在寄生电源方式下可由数据线供电。

（2）独特的单线接口方式，DS18B20 在与微处理器连接时仅需要一条口线即可实现微处理器与 DS18B20 的双向通讯。

（3）DS18B20 支持多点组网功能，多个 DS18B20 可以并联在唯一的三线上， 实现组网多点测温。

（4）DS18B20 在使用中不需要任何外围元件，全部传感元件及转换电路集成在形如一只三极管的集成电路内。

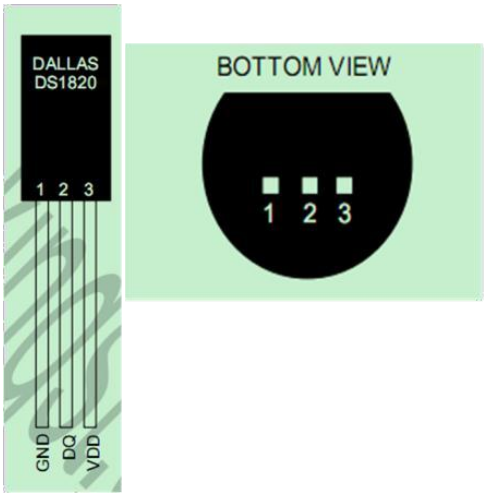
（5）温范围－55℃～+125℃，在-10～+85℃时精度为±0.5℃

（6）可编程的分辨率为 9～12 位，对应的可分辨温度分别为 0.5℃、0.25℃、0.125℃ 和 0.0625℃，可实现高精度测温。

（7）在 9 位分辨率时最多在 93.75ms 内把温度转换为数字，12 位分辨率时最多在 750ms 内把温度值转换为数字，速度更快。

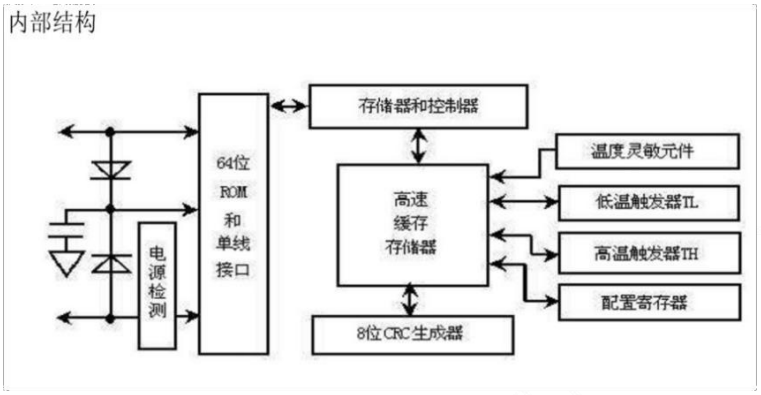
（8）测量结果直接输出数字温度信号，以"一根总线"串行传送给 CPU，同时可传送 CRC 校验码，具有极强的抗干扰纠错能力。

（9）负压特性：电源极性接反时，芯片不会因发热而烧毁，但不能正常工作。DS18B20 外观实物如下图所示：



从 DS18B20 外观图可以看到，当我们正对传感器切面（传感器型号字符那一面）时，传感器的管脚顺序是从左到右排列。管脚 1 为 GND，管脚 2 为数据 DQ，管脚 3 为 VDD。如果把传感器插反，那么电源将短路，传感器就会发烫， 很容易损坏，所以一定要注意传感器方向，通常我们在开发板上都会标出传感器的凸起出，所以只需要把传感器凸起的方向对着开发板凸起方向插入即可。

DS18B20 内部结构如下图所示：



ROM 中的 64 位序列号是出厂前被光刻好的，它可以看作是该 DS18B20 的地址序列号。64 位光刻 ROM 的排列是：开始 8 位（28H）是产品类型标号，接着的 48 位是该 DS18B20 自身的序列号，最后 8 位是前面 56 位的循环冗余校验码。光刻 ROM 的作用是使每一个 DS18B20 都各不相同，这样就可以实现一根总线上挂接多个 DS18B20 的目的。

DS18B20 温度传感器的内部存储器包括一个高速的暂存器 RAM 和一个非易失性的可电擦除的 EEPROM，后者存放高温度和低温度触发器 TH、TL 和配置寄存器。

配置寄存器是配置不同的位数来确定温度和数字的转化，配置寄存器结构如下：

TM	R1	R0	1	1	1	1	1

低五位一直都是"1"，TM 是测试模式位，用于设置 DS18B20 在工作模式还是在测试模式。在 DS18B20 出厂时该位被设置为 0，用户不需要去改动。R1 和 R0 用来设置 DS18B20 的精度（分辨率），可设置为 9，10，11 或 12 位，对应的分辨率温度是 0.5℃，0.25℃，0.125℃和 0.0625℃。R0 和 R1 配置如下图：

R1	R0	精度	最大转换时间	
0	0	9-bit	93.75 ms	(t_{CONV}/8)
0	1	10-bit	187.5 ms	(t_{CONV}/4)
1	0	11-bit	375 ms	(t_{CONV}/2)
1	1	12-bit	750 ms	(t_{CONV})

在初始状态下默认的精度是 12 位，即 R0=1、 R1=1。高速暂存存储器由 9 个字节组成，其分配如下：

寄存器内容	字节地址
温度值低位（LS Byte）	0
温度值高位（MS Byte）	1
高温限值（TH）	2
低温限值（TL）	3
配置寄存器	4
保留	5
保留	6
保留	7
CRC校验值	8

当温度转换命令（44H）发布后，经转换所得的温度值以二字节补码形式存放在高速暂存存储器的第 0 和第 1 个字节。存储的两个字节，高字节的前 5 位是符号位 S，单片机可通过单线接口读到该数据，读取时低位在前，高位在后， 数据格式如下：

温度寄存器格式 图 2

	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
LS Byte	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
MS Byte	S	S	S	S	S	2^6	2^5	2^4

如果测得的温度大于 0，这 5 位为‘ 0’，只要将测到的数值乘以 0.0625（默认精度是 12 位）即可得到实际温度；如果温度小于 0，这 5 位为‘ 1’， 测到的数值需要取反加 1 再乘以 0.0625 即可得到实际温度。温度与数据对应关系如下：

温度/数据关系 表 2

温度 ℃	数据输出（二进制）	数据输出（十六进制）
+125	0000 0111 1101 0000	07D0h
+85	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Eh
-55	1111 1100 1001 0000	FC90h

*上电复位时温度寄存器默认值为+85℃

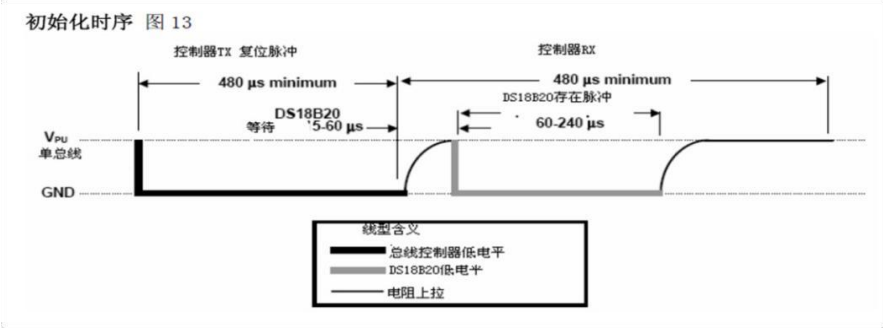
比如我们要计算+85 度，数据输出十六进制是 0X0550，因为高字节的高 5 位为 0，表明检测的温度是正温度，0X0550 对应的十进制为 1360，将这个值乘以 12 位精度 0.0625，所以可以得到+85 度。

知道了怎么计算温度， 接下来我们就来看看如何读取温度数据， 由于 DS18B20 是单总线器件，所有的单总线器件都要求采用严格的信号时序，以保证数据的完整性。DS18B20 时序包括如下几种：初始化时序、写（0 和 1）时序、读（0 和 1）时序。 DS18B20 发送所有的命令和数据都是字节的低位在前。这里我们简单介绍这几个信号的时序：

（1） 初始化时序

单总线上的所有通信都是以初始化序列开始。主机输出低电平，保持低电平时间至少 480us（该时间的时间范围可以从 480 到 960 微妙），以产生复位脉冲。接着主机释放总线，外部的上拉电阻将单总线拉高，延

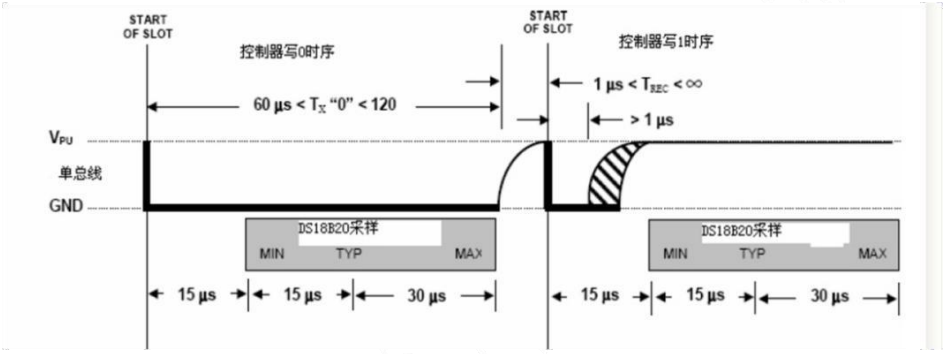
初始化时序 图 13



时 15～60us，并进入接收模式。接着 DS18B20 拉低总线 60~240 us，以产生低电平应答脉冲，若为低电平，还要做延时，其延时的时间从外部上拉电阻将单总线拉高算起最少要480 微妙。初始化时序图如下：

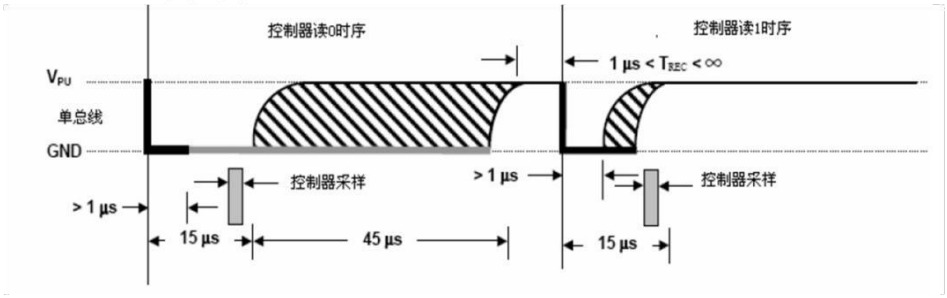
（2）写时序

写时序包括写 0 时序和写 1 时序。所有写时序至少需要 60us，且在 2 次独立的写时序之间至少需要 1us 的恢复时间，两种写时序均起始于主机拉低总线。写 1 时序：主机输出低电平，延时 2us，然后释放总线，延时 60us。写 0 时序：主机输出低电平，延时 60us，然后释放总线，延时 2us。写时序图如下：



（3）读时序

单总线器件仅在主机发出读时序时，才向主机传输数据，所以，在主机发出读数据命令后，必须马上产生读时序，以便从机能够传输数据。所有读时序至少需要 60us，且在 2 次独立的读时序之间至少需要 1us 的恢复时间。每个读时序都由主机发起，至少拉低总线 1us。主机在读时序期间必须释放总线，并且在时序起始后的 15us 之内采样总线状态。读时序图如下：



典型的读时序过程为：主机输出低电平延时 2us，然后主机转入输入模式延时 12us，然后读取单总线当前的电平，然后延时 50us。

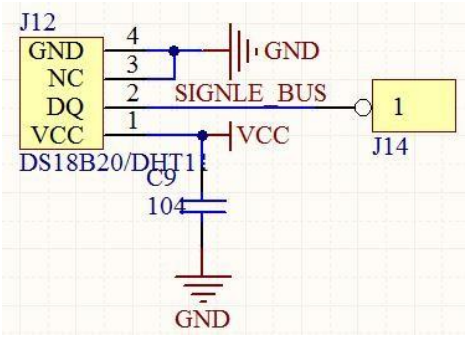
在了解了单总线时序之后，我们来看看 DS18B20 的典型温度读取过程， DS18B20 的典型温度读取过程为：复位→发 SKIP ROM 命令（0XCC）→发开始转换命令（0X44）→延时→复位→发送 SKIP ROM 命令（0XCC）→发读存储器命令（0XBE）→连续读出两个字节数据(即温度)→结束。

本实验使用到硬件资源如下：

（1）动态数码管

（2）DS18B20

动态数码管电路在前面章节都介绍过，这里就不再重复。下面我们来看下开发板上 DS18B20 模块电路，如下图所示：

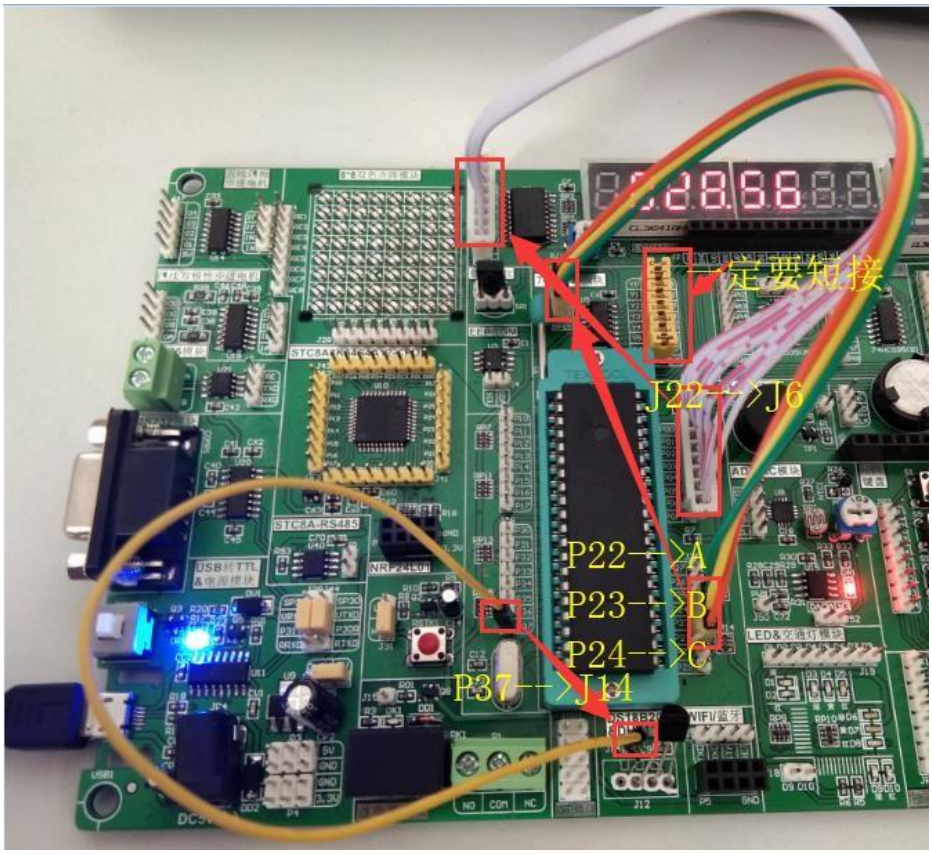


从上图中可以看出，该电路是独立的，并且该接口可以支持 DS18B20 温度传感器和DHT11 温湿度传感器。传感器接口的单总线管脚接至 J14 端子上，在介绍单总线的时候我们说过，为了让单总线默认为高电平，通常会在单总线上接上拉电阻，在图中并没有看到有上拉电阻，这是因为我们单片机 IO 都外接了 10K 上拉电阻，当单片机IO 口连接到传感器的总线管脚时即相当于它们外接上拉电阻， 所以此处可以省去。

代码请参考例程序。

三、实验内容及步骤

使 用 USB 线将开发板和电脑连接成功后（使用短接片将 USB 转串口模块上J39 端子的 P31T 与URXD 短接，J44 端子的P30R 与UTXD 短接，电脑能识别开发板上 CH340 驱动串口），把编写好的程序编译后将编译产生的.hex 文件烧入到芯片内， 按照如下图所示的接线方式可以看到：插上 DS18B20 温度传感器，数码管显示检测的温度值。



四、作业

利用 DS18B20 设计一个智能温度控制系统，具有温度上下限值设定， 当温度高于上限值电机开启同时蜂鸣器报警，当温度低于下限值时继电器工作加热，同时蜂鸣器报警，当温度处于上下限间正常工作。（温馨提示：把本章实验结合按键控制实验）

实验四 LCD1602 液晶显示实验

一、实验目的

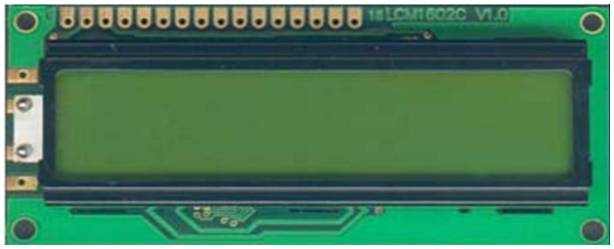
1．学会使用 LCD1602 液晶屏；

2．掌握 LCD1602 液晶显示的控制方法。

二、实验说明

本实验（4 课时）以 STM32 单片机为例进行说明，具体如下：

1602 液晶也叫 1602 字符型液晶，它能显示 2 行字符信息，每行又能显示 16 个字符。它是一种专门用来显示字母、数字、符号的点阵型液晶模块。它是由若干个5x7 或者5x10 的点阵字符位组成，每个点阵字符位都可以用显示一个字符，每位之间有一个点距的间隔，每行之间也有间隔，起到了字符间距和行间距的作用，正因为如此，所以它不能很好的显示图片。其实物图如下所示：



大家手上拿到的 LCD1602 外观可能和上图不一样，这是由于不同厂家设计所致，使用方法是一样的。在上图中可以看到有 16 个管脚孔，从左至右管脚编号顺序是 1-16，其功能定义如下所示：

编号	符号	引脚说明	编号	符号	引脚说明
1	VSS	电源地	9	D2	Data I/O
2	VDD	电源正极	10	D3	Data I/O
3	VL	液晶显示偏压信号	11	D4	Data I/O
4	RS	数据/命令选择端（H/L）	12	D5	Data I/O
5	R/W	读/写选择端（H/L）	13	D6	Data I/O
6	E	使能信号	14	D7	Data I/O
7	D0	Data I/O	15	BLA	背光源正极
8	D1	Data I/O	16	BLK	背光源负极

下面对几个管脚做下说明：

3 脚：VL，液晶显示偏压信号，用于调整 LCD1602 的显示对比度，一般会外接电位器用以调整偏压信号，注意此脚电压为 0 时可以得到最强的对比度。

4 脚：RS，数据/命令选择端，当此脚为高电平时，可以对 1602 进行数据字节的传输操作，而为低电平时，则是进行命令字节的传输操作。命令字节，即是用来对 LCD1602 的一些工作方式作设置的字节；数据字节，即使用以在 1602 上显示的字节。值得一提的是，LCD1602 的数据是 8 位

的。

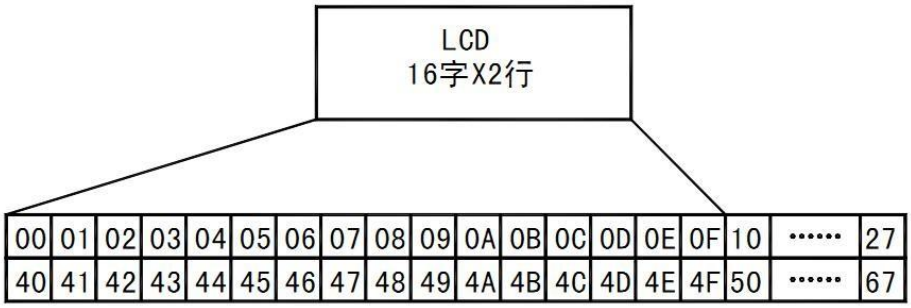
5 脚：R/W，读写选择端。当此脚为高电平可对 LCD1602 进行读数据操作， 反之进行写数据操作。

6 脚：E，使能信号，其实是 LCD1602 的数据控制时钟信号，利用该信号的上升沿实现对 LCD1602 的数据传输。

7~14 脚：8 位并行数据口，而 51 单片机一组 IO 也是 8 位，使得对 LCD1602 的数据读写大为方便。

在 LCD1602 内部含有 80 个字节的 DDRAM，它是用来寄存显示字符的。其地址和屏幕的对应关系如下表：

	显示位置	1	2	3	4	5	6	7	……	40
DDRAM 地址	第一行	00H	01H	02H	03H	04H	05H	06H	……	27H
	第二行	40H	41H	42H	43H	44H	45H	46H	……	67H



从上图可知，不是所有的地址都可以直接用来显示字符数据，只有第一行中的 00-0F，第二行中的 40-4F 才能显示，其他地址只能用于存储。要显示字符时要先输入显示字符地址，也就是告诉模块在哪里显示字符，例如第二行第一个字符的地址是 40H，那么是否直接写入 40H 就可以将光标定位在第二行第一个字符的位置呢？这样不行，因为写入显示地址时要求最高位 D7 恒定为高电平 1 所以实际写入的数据应该是 01000000B(40H）

+10000000B(80H)=11000000B(C0H)。在 1602 中我们就用前 16 个就行了。第二行也一样用前 16 个地址。

LCD1602 常用指令

在使用 LCD1602 时，我们需要掌握一些常用的指令，这些指令对于 LCD1602 初始化是必须的。

（1） 清屏指令

指令功能	指令编码										执行时间 /ms
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
清屏	0	0	0	0	0	0	0	0	0	1	1.64

功能：

<1> 清除液晶显示器，即将 DDRAM 的内容全部填入"空白"的 ASCII 码 20H;

<2> 光标归位，即将光标撤回液晶显示屏的左上方;

<3> 将地址计数器(AC)的值设为 0。

（2） 模式设置指令

指令功能	指令编码										执行时间 /us
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
进入模式设置	0	0	0	0	0	0	0	1	I/D	S	40

功能：

设定每次写入 1 位数据后光标的移位方向，并且设定每次写入的一个字符是否移动。

I/D：0=写入新数据后光标左移　　1=写入新数据后光标右移

S：0=写入新数据后显示屏不移动　　1=写入新数据后显示屏整体右移 1 个字符

（3） 显示开关控制指令

指令功能	指令编码										执行时间 /us
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
显示开关控制	0	0	0	0	0	0	1	D	C	B	40

功能：

控制显示器开/关、光标显示/关闭以及光标是否闪烁。

D：0=显示功能关　　1=显示功能开

C：0=无光标　　1=有光标

B：0=光标闪烁　　1=光标不闪烁

（4）功能设定指令

指令功能	指令编码										执行时间 /us
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
功能设定	0	0	0	0	1	DL	N	F	X	X	40

功能：

设定数据总线位数、显示的行数及字型。

DL：0=数据总线为 4 位 1=数据总线为 8 位

N：0=显示 1 行　　1=显示 2 行

F：0=5×7 点阵/每字符　1=5×10 点阵/每字符

要使用 LCD1602，首先需要对其初始化，即通过写入一些特定的指令实现。然后选择要在 LCD1602 的哪个位置显示并将所要显示的数据发送到 LCD 的 DDRAM。使用 LCD1602 通常都是用于写数据进去，很少使用读功能。LCD1602 操作步骤如下所示：

（1） 初始化

（2） 写命令（RS=L），设置显示坐标

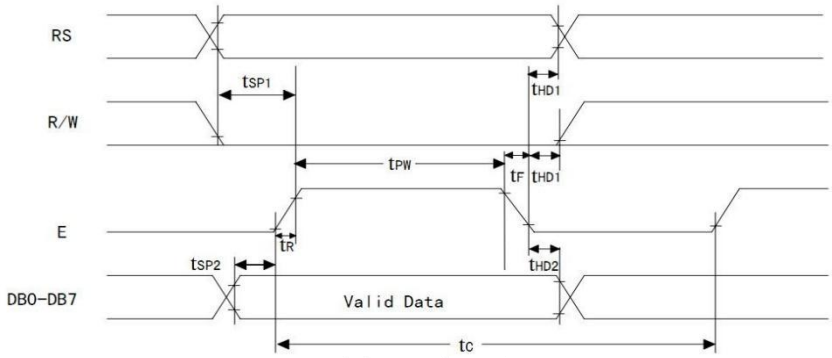
（3） 写数据（RS=H）

在此，不需要读出它的数据的状态或者数据本身。所以只需要看两个写时序：

① 当要写指令字，设置 LCD1602 的工作方式时：需要把 RS 置为低电平，RW 置为低电平，然后将数据送到数据口 D0~D7，最后 E 引脚一个高脉冲将数据写入。

② 当要写入数据字，在 1602 上实现显示时：需要把 RS 置为高电平，RW 置为低电平，然后将数据送到数据口 D0~D7，最后 E 引脚一个高脉冲将数据写入。写指令和写数据，差别仅仅在于 RS 的电平不一样而已。以下是 LCD1602 的

时序图：



从上图可以看到，以上给的时间参数全部是 ns 级别的，而51 单片机的机器周期是 1us，指令周期是 2-4 个机器周期，所以即便在程序里不加延时程序，也可以很好的配合 LCD1602 的时序要求了。

当要写命令字节的时候，时间由左往右，RS 变为低电平，R/W 变为低电平， 注意看是 RS 的状态先

时序参数	符号	极限值			单位
		最小值	典型值	最大值	
E 信号周期	t_C	400	–	–	ns
E 脉冲宽度	t_{PW}	150	–	–	ns
E 上升沿/下降沿时间	t_R, t_F	–	–	25	ns
地址建立时间	t_{SP1}	30	–	–	ns
地址保持时间	t_{HD1}	10	–	–	ns
数据建立时间(读操作)	t_D	–	–	100	ns
数据保持时间(读操作)	t_{HD2}	20	–	–	ns
数据建立时间(写操作)	t_{SP2}	40	–	–	ns
数据保持时间(写操作)	t_{HD2}	10	–	–	ns

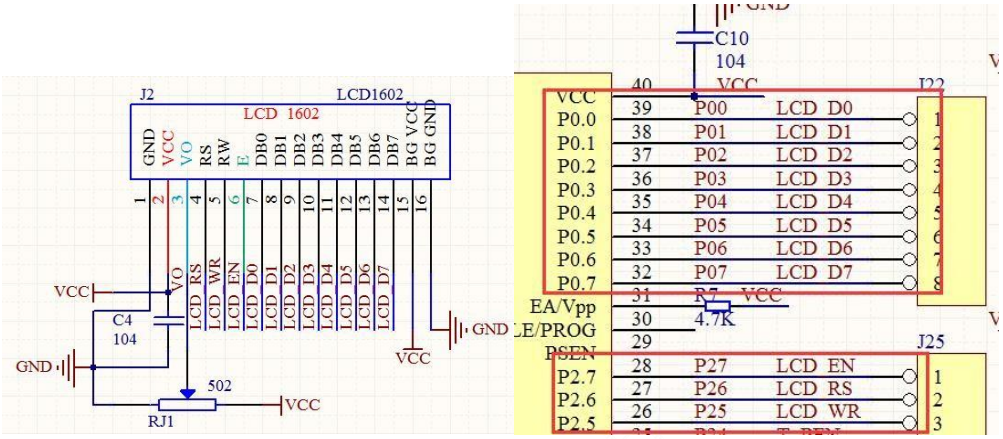
变化完成。然后这时，DB0~DB7 上数据进入有效阶段，接着 E 引脚有一个整脉冲的跳变，接着要维持时间最小值为 tpw=400ns 的 E 脉冲宽度。然后 E 引脚负跳变，RS 电平变化，R/W 电平变化。这样便是一个完整的 LCD1602 写命令的时序。

注意：这里介绍的是 8 位 LCD1602，现在某些公司为简化引脚数，使用 4 位 LCD1602。使用 4 位 LCD1602 时，应该多看手册，找到不同点，对原有程序加以修改。我们例程也做了 4 位和 8 位 LCD1602 的兼容，具体后面会介绍。

本实验使用到硬件资源如下：

（1）LCD1602 液晶

开发板上集成了一个 LCD1602 液晶接口，下面我们来看下开发板上 LCD1602 液晶接口电路，如下图所示：



从上图中可以看出，该电路是并不是独立的，LCD1602的8 位数据口DB0-DB7 与单片机的P00-P07 管脚连接，LCD1602的RS、RW、E 脚与单片机的P26、P25、P27 管脚连接。RJ1 是一个电位器，用来调节 LCD1602 对比度即显示亮度。

注意：这里原理图是使用的 8 位 LCD1602 接口设计，如果使用 4 位 LCD1602 则需要使用转接板。在出厂配套 LCD1602 时，有的是 8 位的，有的是4 位的。出厂的 4 位 LCD1602 已带有转接板，仅仅将 8 位 LCD1602 引脚的高四位用于 4 位 LCD1602。也就是说在传输数据的时候需要将 8 位的数据截成两段，先发送高四位，在发送低四位。其它引脚操作方法不变。

做本章实验时只需要将配置的 LCD1602 液晶插到开发板的LCD1602 接口处，无需额外接线，如下所示（具体可参考实验现象接线图）：

代码请参考例程序

三、实验内容及步骤

使用 USB 线将开发板和电脑连接成功后（使用短接片将 USB 转串口模块上J39 端子的P31T 与JRXD 短接，J44 端子的P30R 与JTXD 短接，电脑能识别开发板上CH340 驱动串口），把编写好的程序编译后将编译产生的.hex 文件烧入到芯片内，按照如下图所示的接线方式可以看到：LCD1602 第一行显示“PechinScience”。

四、课后作业

使用 LCD1602 显示时钟或上节课的温度。

（温馨提示：LCD1602 只能以ASCII 字符显示，所以在显示时钟数值时，需要将时钟数据或温度数据转换为字符，比如数值 1 转换为ASCII 字符可这样操作：1+0X30， 或者1+’’）

实验五 I2C-EEPROM 实验

一、实验目的

1．学使用 STM32F1 的IO 口模拟 I2C 时序；

2．实现与 AT24C02（EEPROM）之间的双向通信。

二、实验说明

本实验（4 课时）以 STM32 单片机为例进行说明，具体如下：

1、I2C 介绍

I2C（Inter－IntegratedCircuit）总线是由 PHILIPS 公司开发的两线式串行总线，用于连接微控制器及其外围设备。是微电子通信控制领域广泛采用的一种总线标准。它是同步通信的一种特殊形式，具有接口线少，控制方式简单，器件封装形式小，通信速率较高等优点。I2C 总线只有两根双向信号线。一根是数据线 SDA，另一根是时钟线 SCL。由于其管脚少，硬件实现简单，可扩展性强等特点，因此被广泛的使用在各大集成芯片内。下面我们就从 I2C 的物理层与协议层来了解 I2C。

I2C 通信设备常用的连接方式如图 5.1 所示：

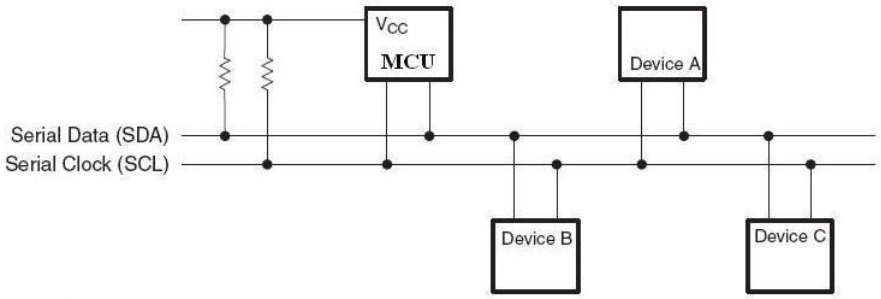


图 5.1 常用的 I2C 通信系统图

它的物理层有如下特点：

它是一个支持多设备的总线。“总线”指多个设备共用的信号线。在一个 I2C 通讯总线中，可连接多个 I2C 通讯设备，支持多个通讯主机及多个通讯从机。

一个 I2C 总线只使用两条总线线路，一条双向串行数据线(SDA) ，一条串行时钟线(SCL)。数据线即用来表示数据，时钟线用于数据收发同步。

每个连接到总线的设备都有一个独立的地址，主机可以利用这个地址进行不同设备之间的访问。

总线通过上拉电阻接到电源。当 I2C 设备空闲时，会输出高阻态，而当所有设备都空闲，都输出高阻态时，由上拉电阻把总线拉成高电平。

多个主机同时使用总线时，为了防止数据冲突，会利用仲裁方式决定由哪个设备占用总线。

具有三种传输模式：标准模式传输速率为 100kbit/s ，快速模式为 400kbit/s ，高速模式下可达 3.4Mbit/s，但目前大多 I2C 设备尚不支持高速模式。

连接到相同总线的 IC 数量受到总线的最大电容 400pF 限制。下面我们来了解下 I2C 总线常用的一些术语：

主机：启动数据传送并产生时钟信号的设备； 从机：被主机寻址的器件；

多主机：同时有多于一个主机尝试控制总线但不破坏传输；

主模式：用 I2CNDAT 支持自动字节计数的模式；位 I2CRM,I2CSTT,I2CSTP 控制数据的接收和发送；

从模式：发送和接收操作都是由 I2C 模块自动控制的；

仲裁：是一个在有多个主机同时尝试控制总线但只允许其中一个控制总线并使传输不被破坏的过程；

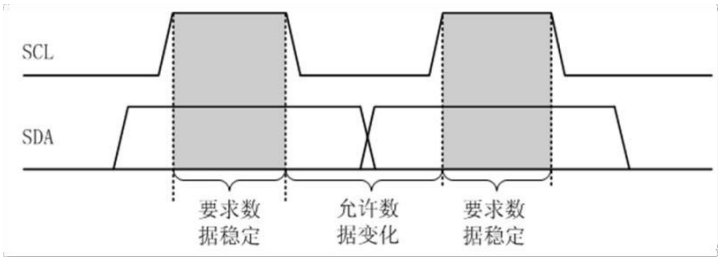
同步：两个或多个器件同步时钟信号的过程； 发送器：发送数据到总线的器件；

接收器：从总线接收数据的器件。

I2C 的协议定义了通信的起始和停止信号、数据有效性、响应、仲裁、时钟同步和地址广播等环节。下面我们就来简单介绍下。

（1） 数据有效性规定

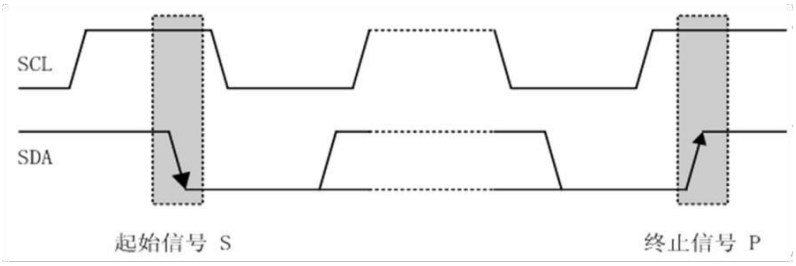
I2C 总线进行数据传送时，时钟信号为高电平期间，数据线上的数据必须保持稳定，只有在时钟线上的信号为低电平期间，数据线上的高电平或低电平状态才允许变化。如下图：



每次数据传输都以字节为单位，每次传输的字节数不受限制。

（2） 起始和停止信号

SCL 线为高电平期间，SDA 线由高电平向低电平的变化表示起始信号；SCL 线为高电平期间，SDA 线由低电平向高电平的变化表示终止信号。如下图：

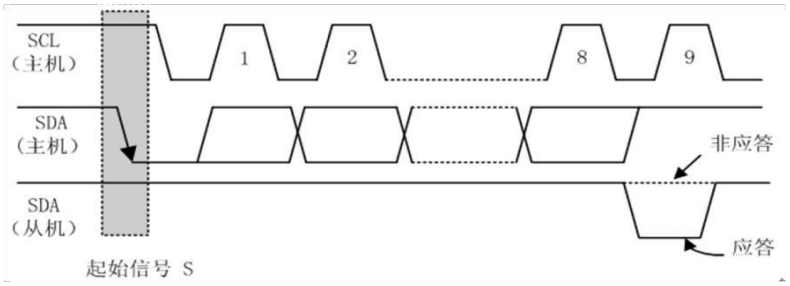


起始和终止信号都是由主机发出的，在起始信号产生后，总线就处于被占用的状态；在终止信号产生后，总线就处于空闲状态。

（3） 应答响应

每当发送器件传输完一个字节的数据后，后面必须紧跟一个校验位，这个校验位是接收端通过控制 SDA（数据线）来实现的，以提醒发送端数据我这边已经接收完成，数据传送可以继续进行。这个校验位其实就是数据或地址传输过程中的响应。响应包括“应答(ACK)”和“非应答(NACK)”两种信号。作为数据接收端时，当设备(无论主从机)接收到 I2C 传输的一个字节数据或地址后，若希望对方继续发送数据，则需要向对方发送“应答(ACK)”信号即特定的低电平脉冲， 发送方会继续发送下一个数据；若接收端希望结束数据传输，则向对方发送“非应答(NACK)”

信号即特定的高电平脉冲，发送方接收到该信号后会产生一个停止信号，结束信号传输。应答响应时序图如下：



每一个字节必须保证是 8 位长度。数据传送时，先传送最高位（MSB），每一个被传送的字节后面都必须跟随一位应答位（即一帧共有 9 位）。由于某种原因从机不对主机寻址信号应答时（如从机正在进行实时性的处理工作而无法接收总线上的数据），它必须将数据线置于高电平，而由主机产生一个终止信号以结束总线的数据传送。

如果从机对主机进行了应答，但在数据传送一段时间后无法继续接收更多的数据时，从机可以通过对无法接收的第一个数据字节的“非应答”通知主机，主机则应发出终止信号以结束数据的继续传送。

当主机接收数据时，它收到最后一个数据字节后，必须向从机发出一个结束传送的信号。这个信号是由对从机的“非应答”来实现的。然后，从机释放 SDA 线，以允许主机产生终止信号。

这些信号中，起始信号是必需的，结束信号和应答信号都可以不要。

（4） 总线的寻址方式

I2C 总线寻址按照从机地址位数可分为两种，一种是 7 位，另一种是 10 位。

采用 7 位的寻址字节（寻址字节是起始信号后的第一个字节）的位定义如下：

位：	7	6	5	4	3	2	1	0
	从机地址							R/$\overline{W}$

D7～D1 位组成从机的地址。D0 位是数据传送方向位，为“0”时表示主机向从机写数据，为“1”时表示主机由从机读数据。

10 位寻址和 7 位寻址兼容，而且可以结合使用。10 位寻址不会影响已有的

7 位寻址，有 7 位和 10 位地址的器件可以连接到相同的 I2C 总线。我们就以 7 位寻址为例进行介绍。

当主机发送了一个地址后，总线上的每个器件都将头 7 位与它自己的地址比较，如果一样，器件会判定它被主机寻址，其他地址不同的器件将被忽略后面的数据信号。至于是从机接收器还是从机发送器，都由 R/W 位决定的。

从机的地址由固定部分和可编程部分组成。在一个系统中可能希望接入多个相同的从机，从机地址中可编程部分决定了可接入总线该类器件的最大数目。如一个从机的 7 位寻址位有 4 位是固定位，3 位是可编程位，这时仅能寻址 8 个同样的器件，即可以有 8 个同样的器件接入到该 I2C 总线系统中。

（5） 数据传输

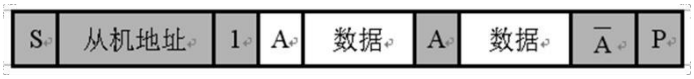
I2C 总线上传送的数据信号是广义的，既包括地址信号，又包括真正的数据信号。在起始信号后必须传送一个从机的地址（7 位），第8 位是数据的传送方向位（R/W），用“0”表示主机发送（写）数据（W），“1”表示主机接收数据（R）。每次数据传送总是由主机产生的终止信号结束。但是，若主机希望继续占用总线进行新的数据传

送，则可以不产生终止信号，马上再次发出起始信号对另一从机进行寻址。

在总线的一次数据传送过程中，可以有以下几种组合方式：

a、主机向从机发送数据，数据传送方向在整个传送过程中不变

注意：有阴影部分表示数据由主机向从机传送，无阴影部分则表示数据由从机向主机传送。A 表示应答，A 非表示非应答（高电平）。S 表示起始信号，P 表示终止信号。



b、主机在第一个字节后，立即从从机读数据

c、在传送过程中，当需要改变传送方向时，起始信号和从机地址都被重复产生一次，但两次读/写方向位正好反相

到这里我们就介绍完了 I2C 总线，现如今大部分的 MCU 都自带 I2C 总线接口， STM32F1 芯片也不例外，STM32F1 芯片自带 2 个 I2C 接口，I2C1 和 I2C2，但是本章实验我们不使用 STM32F1 自带的硬件 I2C，而采用软件模拟 I2C。主要原因是 STM32F1 的硬件 IIC 设计的比较复杂，而且稳定性不怎么好，程序移植比较麻烦， 而用软件模拟 IIC，最大的好处就是移植方便，同一个代码兼容所有单片机，任何一个单片机只要有 IO 口（不需要特定 IO），都可以很快的移植过去。有兴趣的朋友，可以结合 STM32F1 中文参考手册学习下硬件 I2C。

2.AT24C02 介绍

AT24C01/02/04/08/16...是一个 1K/2K/4K/8K/16K 位串行 CMOS，内部含有 128/256/512/1024/2048 个 8 位字节， AT24C01 有一个 8 字节页写缓冲器， AT24C02/04/08/16 有一个 16 字节页写缓冲器。该器件通过 I2C 总线接口进行操作，它有一个专门的写保护功能。我们开发板上使用的是 AT24C02（EEPROM）芯片，此芯片具有 I2C 通信接口，芯片内保存的数据在掉电情况下都不丢失，所以通常用于存放一些比较重要的数据等。AT24C02 芯片管脚及外观图如图 5.3 所示：

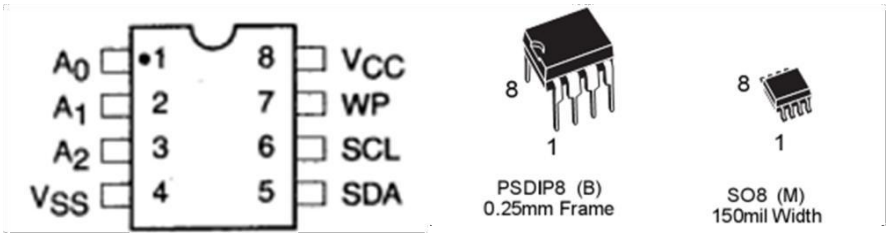


图 5.3AT24C02 芯片管脚及外观图

芯片管脚说明如图 5.4 所示：

引脚号	引脚名称	功能说明
1	A0	地址输入。A2、A1和A0是器件地址输入引脚。 24C02/32/64使用A2、A1和A0输入引脚作为硬件地址，总线上可同时级联8个24C02/32/64器件（详见器件寻址）。 24C04使用A2和A1输入引脚作为硬件地址，总线上可同时级联4个24C04器件，A0为空脚，可接地。 24C08使用A2输入引脚作为硬件地址，总线上可同时级联2个24C08器件，A0和A1为空脚，可接地。 24C16未使用器件地址引脚，总线上最多只可连接一个16K器件，A2、A1和A0为空脚，可接地。
2	A1	
3	A2	
5	SDA	串行地址和数据输入/输出。SDA是双向串行数据传输引脚，漏极开路，需外接上拉电阻到Vcc（典型值10kΩ）。
6	SCL	串行时钟输入。SCL同步数据传输，上升沿数据写入，下降沿数据读出。
7	WP	写保护。WP引脚提供硬件数据保护。当WP接地时，允许数据正常读写操作；当WP接Vcc时，写保护，只读。
4	GND	地
8	Vcc	正电源

图 5.4 AT24C02 芯片管脚说明图

AT24C02 器件地址为 7 位，高4 位固定为1010，低3 位由 A0/A1/A2 信号线的电平决定。因为传输地址或数据是以字节为单位传送的，当传送地址时，器件地址占7 位，还有最后一位（最低位 R/W）用来选择读写方向，它与地址无关。其格式如下：

24WC01/02	1	0	1	0	A2	A1	A0	R/$\overline{W}$

我们开发板已经将芯片的 A0/A1/A2 连接到GND，所以器件地址为 1010000， 即0x50（未计算最低位）。如果要对芯片进行写操作时，R/W 即为0，写器件地址即为 0XA0；如果要对芯片进行读操作时，R/W 即为1，此时读器件地址为 0XA1。

开发板上我们也将 WP 引脚直接接在 GND 上，此时芯片允许数据正常读写。I2C 总线时序如图 5.5 所示：

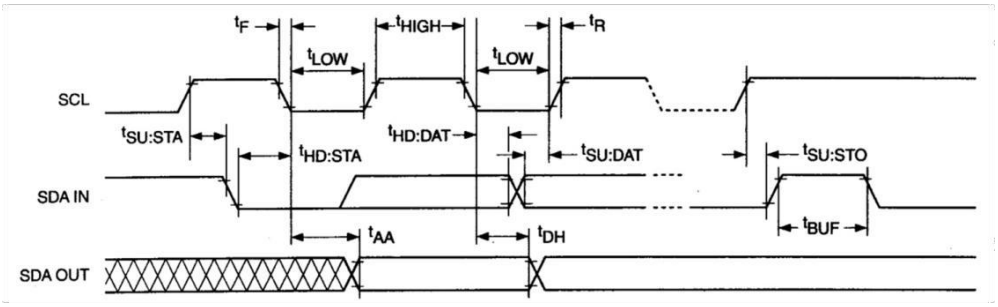


图 5.5 I2C 总线时序图

读写周期范围

符号	参数	1.8 V，2.5 V		4.5V～5.5V		单位
		最小	最大	最小	最大	
F_{SCL}	时钟频率		100		400	KHz
T_I	SCL,SDA 输入的噪声抑制时间		200		200	ns
t_{AA}	SCL 变低至 SDA 数据输出及应答信号		3.5		1	μs
t_{BUF}	新的发送开始前总线空闲时间	4.7		1.2		μs
$t_{HD:STA}$	起始信号保持时间	4		0.6		μs
t_{LOW}	时钟低电平周期	4.7		1.2		μs
t_{HIGH}	时钟高电平周期	4		0.6		μs
$t_{SU:STA}$	起始信号建立时间	4.7		0.6		μs
$t_{HD:DAT}$	数据输入保持时间	0		0		ns
$t_{SUI:DAT}$	数据输入建立时间	50		50		ns
t_R	SDA 及 SCL 上升时间		1		0.3	μs
t_F	SDA 及 SCL 下降时间		300		300	ns
$t_{SU:STO}$	停止信号建立时间	4		0.6		μs
t_{DH}	数据输出保持时间	100		100		ns

关于 AT24C02（EEPROM）的更多信息，可参考“\6--芯片资料”内 24C02 数据手册来了解。

3.硬件设计

本实验使用到硬件资源如下：D1 指示灯、K_UP 和 K_DOWN 按键、串口 1。

D1 指示灯、K_UP 和 K_DOWN 按键、串口 1 电路在前面章节都介绍过，这里就不多说，AT24C02（EEPROM）模块电路图如图 5.6 所示：

开发板 AT24C02(EEPROM)电路：

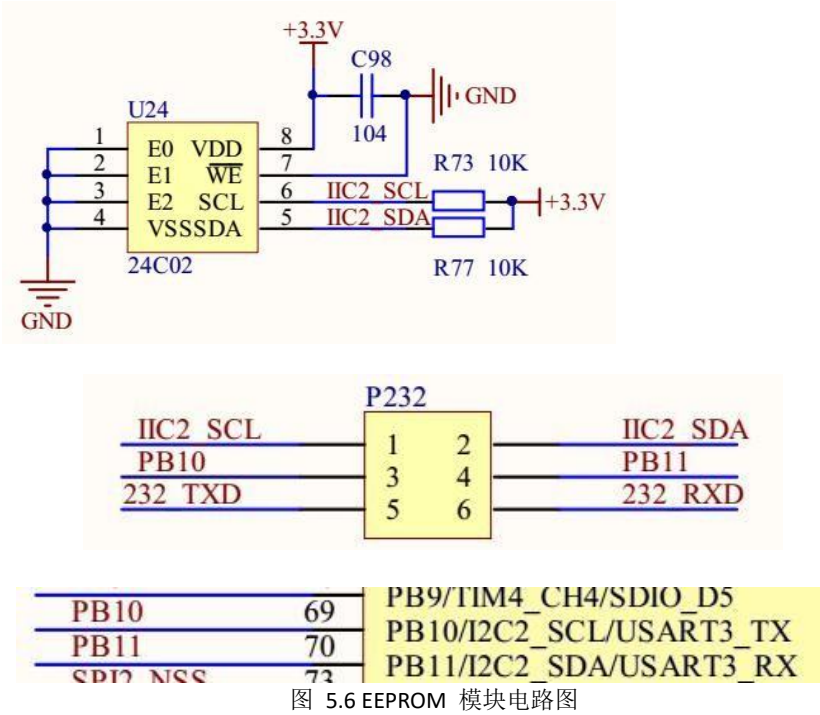


图 5.6 EEPROM 模块电路图

从电路图中可以看到，24C02 芯片的SCL和SDA 管脚连接2*3 端子P232 上，因为USART3 和IIC2 在同一管脚PB10PB11，为了能够独立使用USART3 和IIC2，通过P232 端子进行切换，如果 P232 端子的1、3 和2、4 短接则使用IIC2 功能跟EEPROM 实验，如果3、5 和4、6 短接，则使用 USART3 功能。切换到IIC2 时，管脚都上拉了一个 10K 的电阻。通过这两个管脚模拟 I2C 时序与24C02 通信，从STM32F1 芯片管脚功能图中可以看到这两个管脚本身也是 STM32F1 自带的硬件 I2C2 接口，所以也可以使用 STM32F1 硬件I2C 与24C02 芯片进行通信。

3、软件设计

本章所要实现的功能是：首先检测 AT24C02 芯片是否存在，如果存在则输出提示信息，然后通过按键K_UP 和K_DOWN 控制AT24C02 数据读写，并输出写入和读取的数据信息，最后让 D1 指示灯闪烁提示系统正常运行。程序框架如下：

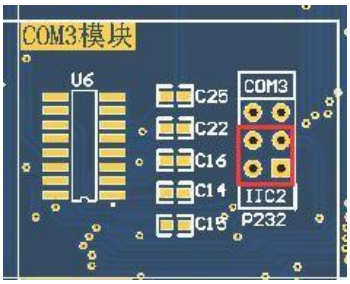
① 使能所用 GPIO 端口时钟，初始化 GPIO
② 使用软件模拟 I2C 通信时序，包含起始和停止信号、应答信号等
③ 编写 AT24C02 读写函数
④ 编写主函数

前面的实验章节都有 GPIO 的初始化，现在对大家来说应该不是问题，所以本章软件的重点在 I2C 时序的模拟及数据的读写上。下面我们打开“\4--实验程序\库函数版\26.I2C-EEPROM 实验”工程在APP 工程组中可以看到添加了 iic.c 和24cxx.c文件（iic.c 里面包含了 I2C 驱动程序，24cxx.c 里面包含了 AT24CXX 驱动程序，并且此程序支持 AT24C01/02/04/08/16/32/64/128/256 芯片），还要包含对应的头文件路径。

三、实验内容及步骤

将工程程序编译后下载到开发板内，串口会输出 AT24C02 检测正常信息，同时可以看到D1 指示灯不断闪烁，表示程序正常运行。当按下 K_UP 按键后，将数据写入到24C02 芯片内，同时串口打印输出写入的值；当按下 K_DOWN 按键后， 读取24C02 芯片内的值，同时串口打印输出读取的值。如果想在串口调试助手上看到输出信息，可以打开“\5--开发工具\4. 常用辅助开发软件\串口调试助手”，首先勾选下标号 1DTR 框，然后再取消勾选。这是因为此串口助手启动时会把系统复位住，通过 DTR 状态切换下即可。然后设置好波特率等参数后，串口助手上即会收到串口发送过来的信息。（串口助手上先勾选下标号 1DTR 框，然后再取消勾选）如图 所示：

实验说明：对于 PZ6806L 开发板用户需要将 COM3 模块上的 P232 端子短接到 IIC2 端，否则无法显示此功能，这个在硬件电路已经说过。开发板出厂默认已经短接到 IIC2 上，如下：



对于开发板的用户无需操作短接片，默认已经连接到 IIC1 上。存储在 24C02（EEPROM）芯片内的值，掉电时都不会丢失。大家可以使用一个短接片将对应的 IIC 模拟 IO 口短接，重启下系统，这个时候就会看到初始化时会报“AT24C02 检测不正常!”（注意还是要按照前面操作串口助手的方法才能看到）。

四、课后作业

使用 K_UP 按键写入多个字节数据到 AT24C02 内， 比如写入“www.prechin.cn”字符串，使用 K_DOWN 按键读取上次写入的数据，通过串口输出。（温馨提示：调用 AT24CXX_Write 与 AT24CXX_Read 函数即可）

实验六 红外遥控实验

一、实验目的

1．掌握红外遥控发送和接收信号的原理即编解码的原理；

2．掌握中断的控制方法，通过中断来正确接收红外信号。

二、实验说明

本实验（4 课时）以 STM32 单片机为例进行说明，具体如下：

我们开发板标配了一个一体化红外接收头和红外遥控器，这一章我们来学习如何使用 STM32F1 解码红外遥控器的信号。本章我们使用 STM32F1 的外部中断功能来解码红外遥控器的编码信号。本章要实现的功能是：使用外部中断功能将遥控器键值编码数据解码后通过串口打印输出，同时 D1 指示灯闪烁，提示系统运行。

1、红外遥控介绍

人的眼睛能看到的可见光按波长从长到短排列，依次为红、橙、黄、绿、青、蓝、紫。其中红光的波长范围为 0.62～0.76μm；紫光的波长范围为 0.38～0.46 μm。比紫光波长还短的光叫紫外线，比红光波长还长的光叫红外线。红外线遥控就是利用波长为 0.76～1.5μm 之间的近红外线来传送控制信号的。

2、红外遥控的原理

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便。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。

红外遥控通信系统一般由红外发射装置和红外接收设备两大部分组成。

红外发射装置，也就是通常我们说的红外遥控器是由键盘电路、红外编码电路、电源电路和红外发射电路组成。红外发射电路的主要元件为红外发光二极管。它实际上是一只特殊的发光二极管；由于其内部材料不同于普通发光二极管，因而在其两端施加一定电压时，它便发出的是红外线而不是可见光。目前大量的使用的红外发光二极管发出的红外线波长为 940nm 左右，外形与普通发光二极管相同。红外发光二极管有透明的，还有不透明的，在我们的红外遥控器上可以看到

这个红外发光二极管。红外遥控器和红外发光二极管如图所示：

红外遥控器和红外发光二极管外观图

通常红外遥控为了提高抗干扰性能和降低电源消耗，红外遥控器常用载波的方式传送二进制编码，常用的载波频率为 38kHz，这是由发射端所使用的 455kHz 晶振来决定的。在发射端要对晶振进行整数分频，分频系数一般取 12，所以455kHz÷12≈37.9kHz≈38kHz。也有一些遥控系统采用 36kHz、40 kHz、56 kHz 等，一般由发射端晶振的振荡频率来决定。所以，通常的红外遥控器是将遥控信号（二进制脉冲码）调制在 38KHz 的载波上，经缓冲放大后送至红外发光二极管，转化为红外信号发射出去的。

二进制脉冲码的形式有多种，其中最为常用的是 NEC Protocol 的 PWM 码(脉冲宽度调制)和 Philips RC-5 Protocol 的 PPM 码(脉冲位置调制码，脉冲串之间的时间间隔来实现信号调制)。如果要开发红外接收设备，一定要知道红外遥控器的编码方式和载波频率，我们才可以选取一体化红外接收头和制定解码方案。我们配套的红外遥控器使用的是 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）。而红外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。所以可以通过计算高电平时间判断接收到的数据是 0 还是 1。NEC 码位定义时序图如图所示：

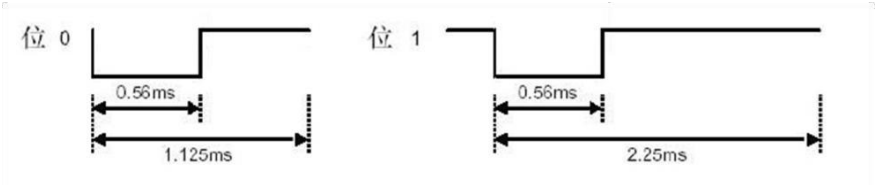


图 NEC 码位定义时序图图

NEC 遥控指令的数据格式为：引导码、地址码、地址反码、控制码、控制反码。引导码由一个 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）。数据格式如下：

NEC 码还规定了连发码(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，红外遥控器按键仍然没有放开，则发射连发码，可以通过统计连发码的次数来标记按键按下的长短或次数。

红外接收设备

红外接收设备是由红外接收电路、红外解码、电源和应用电路组成。红外遥控接收器的主要作用是将遥控发射器发来的红外光信好转换成电信号，再放大、限幅、检波、整形，形成遥控指令脉冲，输出至遥控微处理器。近几年不论是业余制作还是正式产品，大多都采用成品红外接收头。成品红外接收头的封装大致有两种：一种采用铁皮屏蔽；一种是塑料封装。均有三只引脚，即电源正（VDD）、电源负（GND）和数据输出（VOUT）。其外观实物图如下图所示：

图红外接收头外观图

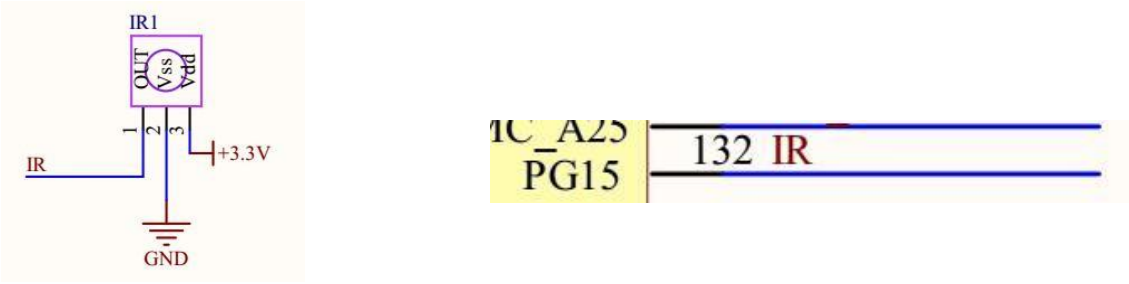
正对接收头的凸起处看，从左至右，管脚依次是 1：VOUT，2：GND，3：VDD。由于红外接收头在没有脉冲的时候为高电平，当收到脉冲的时候为低电平，

所以可以通过外部中断的下降沿触发中断，在中断内通过计算高电平时间来判断接收到的数据是 0 还是 1。外部中断的相关知识，大家可以参考前面的“外部中断实验”，如果想要更加深入的了解红外遥控通信，大家可以百度下，网上这方面的资料可谓是铺天盖地。

3、硬件设计

本实验使用到硬件资源如下：D1 指示灯，串口 1红外遥控器和红外接收头

D1 指示灯、串口 1 电路在前面章节都介绍过，这里就不多说，红外遥控器属于外部器件，只有红外接收头集成在开发板上，其电路如图 所示：



图红外接收模块电路图

从电路图中可以看到，红外接收头的数据输出管脚连接在 STM32F1 芯片的PG15 管脚上，这里我们使用 PG15 引脚内部上拉，所以可以省去一个上拉电阻， 不会有影响，当然也可以接一个 10K 的上拉电阻。通过配置 PG15 引脚为外部中断功能，按照 NEC 协议进行解码。

D1 指示灯用来提示系统运行状态，红外遥控器用来发射红外键值的编码信号，通过红外接收头进行解码，并将解码后的数据通过串口 1 打印输出。

4、软件设计

本章所要实现的功能是：使用外部中断功能将遥控器键值编码数据解码后通过串口打印输出，同时控制 D1 指示灯闪烁，提示系统运行。程序框架如下：

① 使能 PG15 端口及 AFIO 时钟，映射 PG15 至外部中断线上，初始化 EXTI等

② 编写红外解码函数（在 EXTI 中断处理）

③ 编写主函数

外部中断的配置在“外部中断实验”章节中都有介绍，不清楚的可以回过头

看下。下面我们打开“\4--实验程序\库函数版\28. 红外遥控实验”工程，在APP 工程组中可以看到添加了 hwjs.c 文件（里面包含了红外解码驱动程序）， 在 StdPeriph_Driver 工程组中添加了 stm32f10x_exti.c 库文件。EXTI 操作的库函数都放在stm32f10x_exti.c 和stm32f10x_exti.h 文件中，所以使用到 EXTI 就必须加入 stm32f10x_exti.c 文件，同时还要包含对应的头文件路径。

这里我们分析几个重要函数，其他部分程序大家可以打开工程查看。

4.1外部中断初始化函数

我们知道红外接收头数据输出管脚是接在 PG15 上，所以配置 PG15 为外部中断功能，初始化代码如下：

```
/**************************************************************
***************
函 数 名 : Hwjs_Init
函数功能 : 红外端口初始化函数    时钟端口及外部中断初始化
输    入 : 无
输    出 : 无
**************************************************************
**************/
void Hwjs_Init()
{
GPIO_InitTypeDef GPIO_InitStructure; EXTI_InitTypeDef EXTI_InitStructure; NVIC_InitTypeDef NVIC_InitStructure;

/* 开 启 GPIO 时 钟 及 管 脚 复 用 时 钟 */ RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOG|RCC_APB2Periph_AFIO,ENABLE);


GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15;// 红 外 接 收 GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IPU; GPIO_Init(GPIOG,&GPIO_InitStructure);

GPIO_EXTILineConfig(GPIO_PortSourceGPIOG, GPIO_PinSource15); // 选择 GPIO 管脚用作外部中断线路
EXTI_ClearITPendingBit(EXTI_Line15);
/* 设 置 外 部 中 断 的 模 式 */ EXTI_InitStructure.EXTI_Line=EXTI_Line15; EXTI_InitStructure.EXTI_Mode=EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger=EXTI_Trigger_Falling;
EXTI_InitStructure.EXTI_LineCmd=ENABLE; EXTI_Init(&EXTI_InitStructure);/* 设置 NVIC 参数 */
```

```
NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;  //打开全局中断

NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //抢占优先级为 0
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;  // 响应优先级为 1
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //使能NVIC_Init(&NVIC_InitStructure);

}
```

在 Hwjs_Init()函数中，首先使能 GPIOG 端口和 AFIO 时钟，然后将 PG15 映射到外部中断线 15 上，并配置 PG15 为上拉输入模式，最后初始化 EXTI 及 NVIC，这里需要注意，PG15 的外部中断通道是 EXTI15_10_IRQn。外部中断初始化过程在前面章节介绍过，不清楚的可以回过头看下。

6.2红外解码函数

初始化外部中断后，中断就已经开启了，当 PG15 引脚上来了一个下降沿， 就会触发一次中断，在中断内我们可以计算高电平时间，通过高电平时间判断是否进入引导码及数据 0 和 1。具体代码如下：

```
void EXTI15_10_IRQHandler(void) //红外遥控外部中断
{
u8 Tim=0,Ok=0,Data,Num=0;


while(1)
{
if(GPIO_ReadInputDataBit(GPIOG,GPIO_Pin_15)==1)
{
Tim=HW_jssj();//获得此次高电平时间


if(Tim>=250) break;//不是有用的信号


if(Tim>=200 && Tim<250)
{
Ok=1;//收到起始信号
}
else if(Tim>=60 && Tim<90)
{
Data=1;//收到数据 1
}
else if(Tim>=10 && Tim<50)
{
Data=0;//收到数据 0
}


if(Ok==1)
{
hw_jsm<<=1; hw_jsm+=Data;

if(Num>=32)
{
hw_jsbz=1; break;
}
}


Num++;
}
}


EXTI_ClearITPendingBit(EXTI_Line15);
```

```
}
```

中断函数内调用了 HW_jssj 函数获取高电平时间，此函数代码如下：

```
/************************************************************
**************
*                函 数 名 : HW_jssj
*  函数功能          : 高电平持续时间，将记录的时间保存在 t 中返回， 其中一
次大约 20us
*  输          入  : 无
*  输          出  : t
************************************************************
***
*******
****/ u8
HW_jssj
()
{
    u8 t=0; while(GPIO_ReadInputDataBit(GPIOG,GPIO_Pin_15)==1)//高电平
    {
        t++;
        delay_us(20);
        if(t>=250) return t;//超时溢出
    }
    return t;
}
```

通过此函数返回值就可以计算高电平时间，t 累加一次积累的时间为 20us。中断函数内，将此函数返回值保存在变量 Tim 中，通过 Tim 值就可以判断是否进入引导码，如果是引导码，就可以接着判断接收的数据是 1 还是 0，然后将数据进行按低位在前高位在后顺序保存在 hw_jsm 内， 当接收完 32 位数据后将hw_jsbz 标志位置 1，并退出解码函数。这里要注意，判断是否为引导码、数据1 和 0 时，我们不要硬性的固定在高电平区分时间上，而应该给它一个在固定值旁的范围判断，以防止因为干扰而导致误操作。引导码及数据 0 和 1 的高电平区分时间在前面介绍 NEC 码已经讲解了，不清楚的可以回过头看下。

6.3主函数

编写好外部中断初始化和红外解码函数后，接下来就可以编写主函数了，代码如下：

```
/************************************************************
**************
函 数 名       : main
函数功能       : 主函数
输          入  : 无
输          出  : 无
************************************************************
**************/ int main()
{
u8 i=0;

SysTick_Init(72);
```

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);    // 中断优先级分组 分 2 组
LED_Init(); USART1_Init(9600);
Hwjs_Init();


while(1)
{
if(hw_jsbz==1)//如果红外接收到
{
hw_jsbz=0;     //清零
printf("红外接收码  %0.8X\r\n",hw_jsm); //打印hw_jsm=0;    //接收码清零
}


i++; if(i%20==0)
{
led1=!led1;
}


delay_ms(10);
}
}
```

主函数实现的功能很简单，首先调用之前编写好的硬件初始化函数，包括SysTick 系统时钟，中断分组，LED 初始化等。然后调用我们前面编写的 Hwjs_Init函数初始化 PG15 外部中断功能，最后进入 while 循环，通过 hw_jsbz 标志判断是否解码成功，如果成功将解码后的数据 hw_jsm 打印输出，同时控制 D1 指示灯闪烁，提示系统正常运行。

三、实验内容及步骤

将工程程序编译后下载到开发板内，可以看到 D1 指示灯不断闪烁，表示程序正常运行。当按下红外遥控器键时，串口将打印输出解码后的数据（地址码+地址反码+控制码+控制反码）。如果想在串口调试助手上看到输出信息，可以打开“5--开发工具\4. 常用辅助开发软件\串口调试助手”，首先勾选下标号1DTR 框，然后再取消勾选。这是因为此串口助手启动时会把系统复位住，通过 DTR 状态切换下即可。然后设置好波特率等参数后，串口助手上即会收到串口发送过来的信息。（串口助手上先勾选下标号 1DTR 框，然后再取消勾选）如下图所示：

图 红外遥控实验图

实验说明：我们配的遥控器所有键的地址码与反码都是相同的，不同的是控制码和控制反码，其实知道了控制码就知道了控制反码，所以如果要使用红外遥控控制其他设备，可以通过区分控制码来实现。

四、课后作业

用红外遥控器控制开发板上的 2 个指示灯及蜂鸣器。（温馨提示： 对解码后的信号数据进行处理，通过比较来控制）