江西理工大学机电工程学院机器人实验室

2019 年度机器人协会电控培训-暑假电控培训使用资料

STM32F103 实验教程

机器人协会 编

第二版

姓名：_____

日期：_____

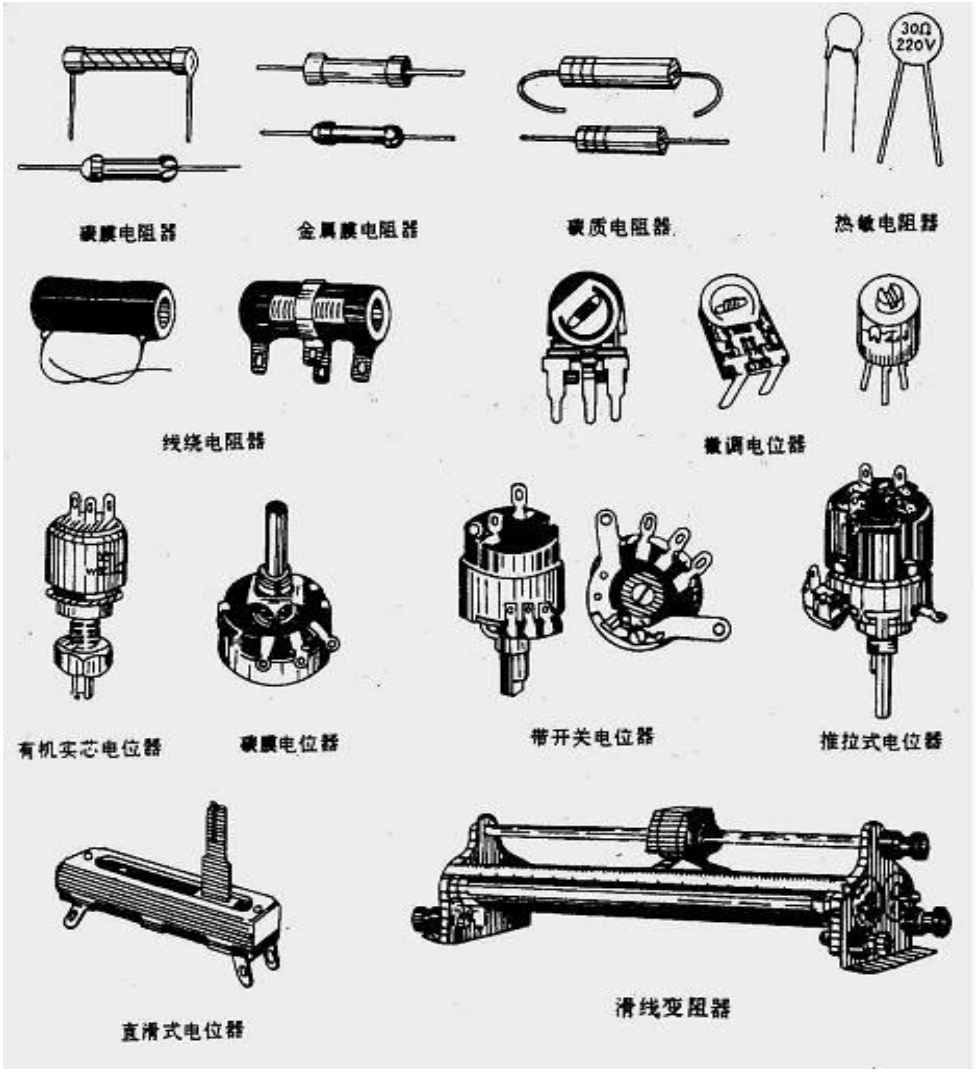
常用电子元件识别

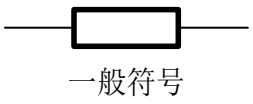
1.1 电子元件

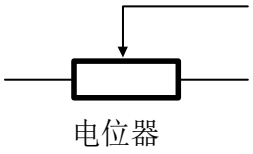
（1）〈电阻〉

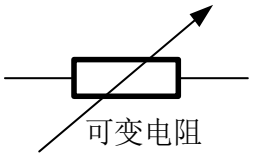
电阻在电路中用“R”加数字表示，如：R1 表示编号为 1 的电阻。电阻在电路中的主要作用为：分流、限流、分压、偏置等。

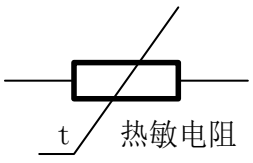
1、电阻器的外形与符号











2、参数识别：电阻的单位为欧姆（Ω），倍率单位有：千欧（KΩ），兆欧（MΩ）等。换算方法是：1 兆欧=1000 千欧=1000000 欧电阻的参数标注方法有 3 种，即直标法、色标法和数标法。 a、数标法主要用于贴片等小体积的电路，如： 472 表示 47×100Ω（即 4.7K）；104 则表示 100K b、色环标注法使用最多

3、万用表测量：旋钮至欧姆档，两表笔短路校 0，度数为相应的刻度盘数 x 档位

注意事项：

① 在测量电阻时，不能用双手同时捏住电阻或测试笔，因为那样的话，人体电阻将会与被测电阻并联在一起，表头上指示的数值就不单纯是被测电阻的阻值了。

② 不能带电测电阻

1.2 电子元件

（2）<电容>

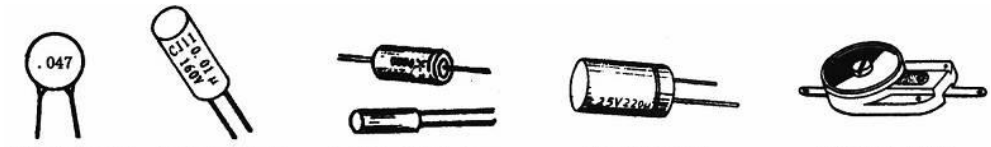


图 1.3.10 部分电容器的外形图

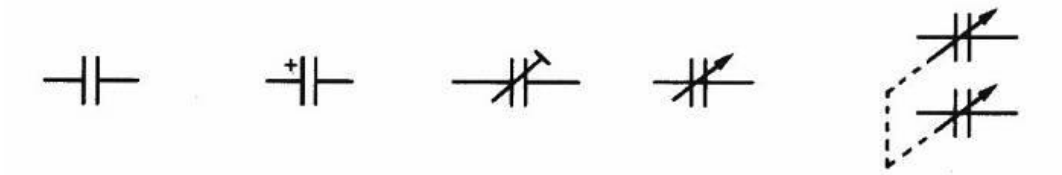


图 1.3.11 电容器的符号

1、电容在电路中一般用“C”加数字表示（如 C13 表示编号为 13 的电容）。电容是由两片金属膜紧靠，中间用绝缘材料隔开而组成的元件。电容的特性主要是隔直流通交流。电容容量的大小就是表示能贮存电能的大小，电容对交流信号的阻碍作用称为容抗，它与交流信号的频率和电容量有关。常用于级间耦合、滤波、去耦、旁路及信号调谐(选择电台)等。常用电容的种类有电解电容、瓷片电容、贴片电容、独石电容、钽电容和涤纶电容等。

2、识别方法：电容的识别方法与电阻的识别方法基本相同，分直标法、色标法和数标法 3 种。电容的基本单位用法拉（F）表示，其它单位还有：毫法（mF）、微法（uF）、纳法(nF)、皮法(pF)。其中：1 法拉=1000 毫法=1000000 微法=1000 000 000 纳法=1000000000000 皮法.容量大的电容其容量值在电容上直接标明，如 10 uF/16V 容量小的电容其容量值在电容上用字母表示或数字表示字母表示法：1m=1000uF；1P2=1.2PF；1nF=1000PF 数字表示法：一般用三位数字表示容量大小，前两位表示有效数字，第三位数字是倍率。如：102 表示 10×100PF=1000PF； 224 表示 22×10000PF=0.22 uF。

3、电容器的正确选用

⑴ 类型选择

电容器类型一般根据它在电路中的作用及工作环境来决定。

例如：

应用在高频电路中的电容器要求其高频特性好

应用在高压环境下的电容器，要求它具有较高的耐压性能，

在电源滤波、去耦、低频级间耦合等电路中，要求容量大的电容器，误差等级都可以。

⑵耐压值的选择

为保证电容器的正常工作，被选用的电容器的耐压值不仅要大于其实际工作电压，而且还要有足够的余地，一般选耐压值为实际工作电压的两倍以上。

4、电容器的实验测试

一般，我们利用万用表的欧姆挡就可以粗略地测量出电容器的优劣情况，粗略地辨别其漏电、容量大小或失效的情况。具体方法是:(与测量电阻方法类似)，根据阻值的变化情况可估判

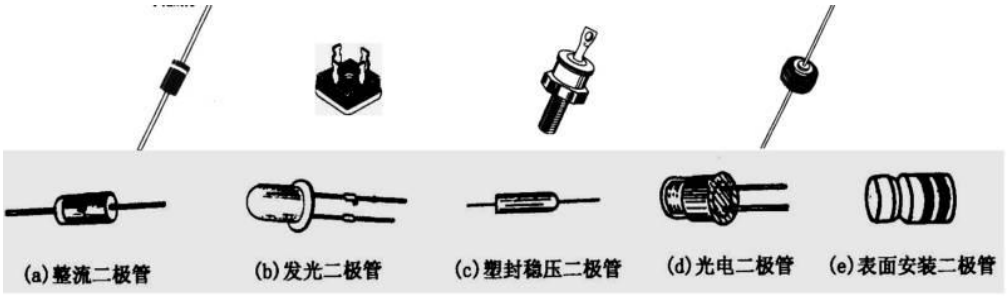
电容器质量。阻值变化快容量小、阻值变化慢容量大、阻值为零电容短路、阻值为无穷大电容可能失效或容量很小。

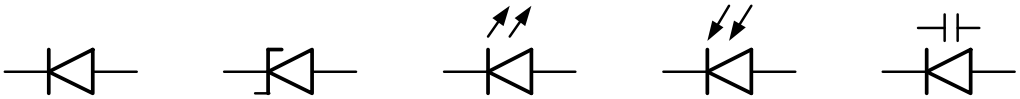
1.3 电子元件

（3）<晶体二极管>

1．二极管的种类和型号

晶体二极管在电路中常用“D”加数字表示，如： D5 表示编号为 5 的二极管。





一般二极管　稳压二极管　发光二极管　光电二极管　变容二极管

2．作用：二极管的主要特性是单向导电性，也就是在正向电压的作用下，导通电阻很小；而在反向电压作用下导通电阻极大或无穷大。正因为二极管具有上述特性，常把它用在整流、隔离、稳压、极性保护、编码控制、调频调制和静噪等电路中。按作用可分为：整流二极管（如 1N4004）、隔离二极管（如 1N4148）、肖特基二极管（快恢复二极管）、发光二极管、稳压二极管等。

3．识别方法：二极管的识别很简单，小功率二极管的 N 极（负极），在二极管外表大多采用一种色圈标出来，银色圈的为负。

4.测试注意事项：用指针式万用表去测二极管时，黑表笔接二极管的正极，红表笔接二极管的负极，此时测得的阻值才是二极管的正向导通阻值，这与数字式万用表的表笔接法刚好相反。

<稳压二极管>

稳压二极管在电路中常用“ZD”加数字表示，如：ZD5 表示编号为 5 的稳压管。

1、稳压二极管的稳压原理：稳压二极管的特点就是击穿后，其两端的电压基本保持不变。这样，当把稳压管接入电路以后，若由于电源电压发生波动，或其它原因造成电路中各点电压变动时，负载两端的电压将基本保持不变。

<发光二极管>

发光二极管（LED）具有正向通过 3-10mA 左右的电流时就发光的性质。发光二极管的测试方法：万用表指针摆动，则黑色表笔所接的就是正极。

实验一 电源制作

一、实验目的：

1. 通过稳压电源的制作，深入理解变压器原理；
2. 初步认识实验室常用的电子元器件；
3. 为实验室后续试验提供所需电源（1Ⅴ～13Ⅴ）；

二、实验器材：

12V 电源一个，船型开关,整流二极管四个，大电容（220uF）四个，LM317 一个，240 欧姆电阻一个，1.5K 欧姆电阻一个，10K 电位器一个，发光二极管一个，电源插头一个，导线一段，电烙铁，小号万用板各一，锡丝若干；

三、实验电路图：

图 1：稳压电源制作基本图

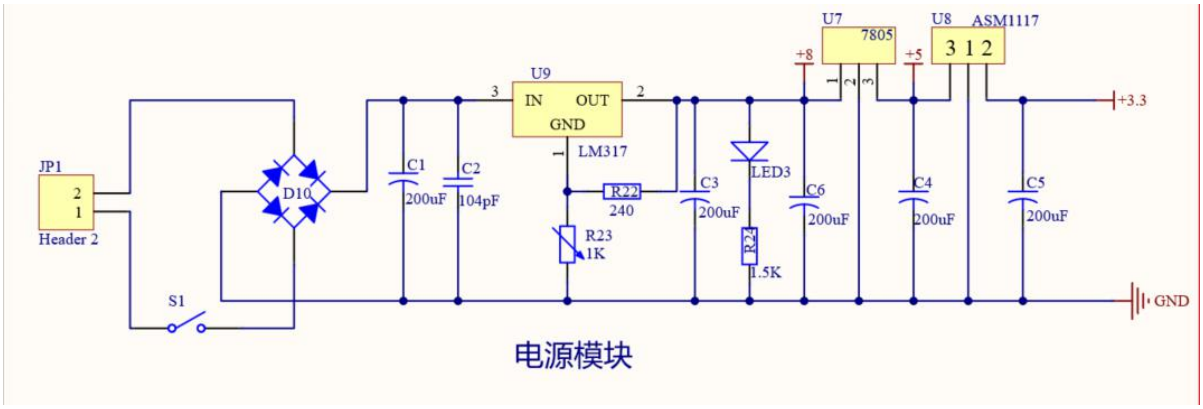


图 2：功能扩展图

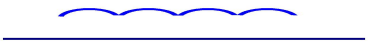
四、内容说明：

1、变压器的基本原理是电磁感应原理,现以单相双绕组变压器为例说明其基本工作原理。

2、整流分为半波整流，桥式整流，全波整流，在本实验中我们用到的是桥式整流，通过四个整流二极管的作用将 220V 交流电变为直流电，但是此时的电压并不未定，联想到电容的稳压特性，接下来我们要在电路中并联电容。

（1）、不接电容时输出的电压波形：

（2）、接上电容后输出的电压波形：



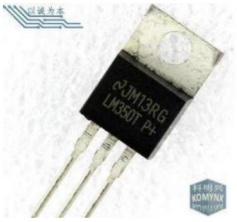
备注：所并联的电容越大，输出电压波形的波动就越小。

3、元件 LM317，7805，ASM1117 简介：

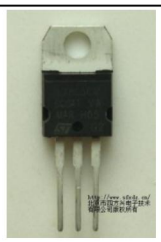
LM317 ： 一种常用三端稳压芯片 ，通过 2 个电阻的比值进行正电压稳压，对于大电流的稳压要加散热片。

7805： 输入电压为 8 伏，输出电压 5 伏、负极接公共地的稳压集成电路。

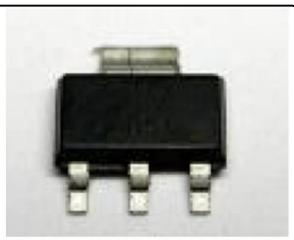
ASM1117：输入电压为 5V，输出电压为 3.3V，因此可以为单片机提供电源。



LM317 图



7805 图



ASM1117 图

备注：三者均有三个引脚，正面向上，引脚向下，引脚顺序自左往右依次为 1、2、3。为了作图方便，在上电路图中所画引脚的顺序并不是实际的引脚顺序,在焊接时要注意这一点。

4、将元件焊接在万用板上以后，要认真检查，在确认焊接无误后接通电源。

（1）、调节电位器，用万用表检查其输出电压的范围是否在 0V～13V 之间；

（2）、观察指示灯（发光二极管）是否随输出电压的变化而呈现明暗变化：

（3）、拔下电源插头后，观察指示灯是否慢慢熄灭，如果是，则说明大电容工作正常。

（原因：电容是一种储能元件，当电源断开后，电容、发光二极管与 1.5K 电阻构成回路，电容发生放电作用。）

五、实验总结：

1、在焊接元器件前先要认真了解各个元器件的功能与特点；

2、焊接时要尽量使融化的焊锡以“圆锥形”的方式包围元件的插脚，这样焊接的元件既牢固又不容易发生“虚焊”，但要注意焊接时烙铁与元件的接触时间，以免温度过高烧坏元器件或万用板；

3、要学会正确使用万用表，并利用它来检查所焊接的电路；

4、最重要的一点——让思维沸腾起来，让智慧行动起来！加入实验室不同于上教条死板的实验课，机械地完成实验项目不是我们的目的，重要的是在实验过程中我们有没有认真观察过、思考过；是否能用学到的理论知识对实验现象进行合理的假设或解释以及在实验完成后我们有哪些收获。博学之，审问之，慎思之，明辨之，笃行之，欢迎你们，大二的学弟学妹们！让我们携手奋进，共同探索，将实验室办得越来越好！

实验二 呼吸灯

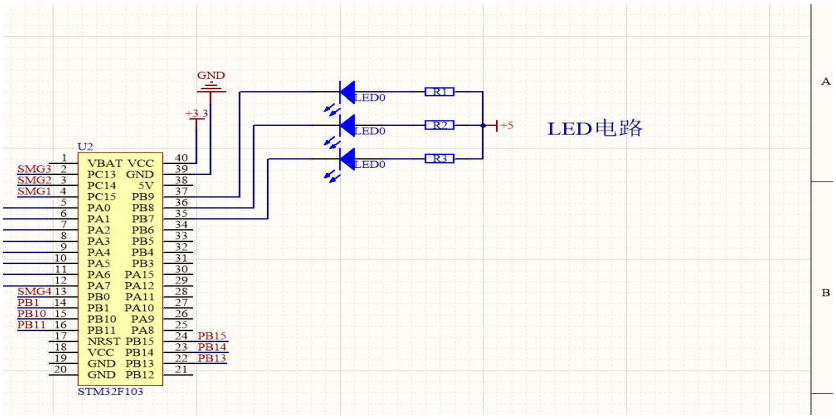
一、实验目的：

初步认识 STM32F103C8T6 这款单片机的各个引脚及功能；学会如何配置 I/O 端口；学习如何新建 stm32 程序工程以及使用 MDK5 软件编写程序并下载。

二、实验器材

STM32F103C8T6 单片机一块，大号万用板一块，500 欧电阻三个，，发光二极管三个，电烙铁一把，排插，导线，锡丝若干。

三、实验电路图



四、实验内容讲解

本试验将要实现的是控制 STM32F103C8T6 上的四个 LED 实现一个类似跑马灯的效果，该实验的关键在于如何控制 STM32 的 IO 口输出。了解了 STM32 的 IO 口如何输出的，就可以实现跑马灯了。通过此实验的学习，你将初步掌握 STM32 基本 IO 口的使用，而这是迈向 STM32 的第一步。

STM32的IO口相比51而言要复杂得多，所以使用起来也困难很多。首先STM32的IO口可以由软件配置成如下8种模式：

1、输入浮空　　2、输入上拉

3、输入下拉　　4、模拟输入

5、开漏输出　　6、推挽输出

7、推挽式复用功能　8、开漏复用功能

简单的解释一下开漏和推挽的区别：

简单的说开漏输出只能输出低电平，要输出高电平，必须加上拉电阻，相当于三极管的集电极，适合于做电流型驱动，而推挽输出和开漏不一样，可以输出高电平，也可以输出低电平，不需要上拉电阻。本实验将用IO口驱动led灯所以将其配置为开漏输出。（后续的实验会继续讲到IO口是模式）每个IO口可以自由编程，但IO口寄存器必须要按32位字被访问。STM32的很多IO口都是5V兼容的，这些IO口在与5V电平的外设连接的时候很有优势，具体哪些IO口是5V兼容的，可以从该芯片的数据手册管脚描述章节查到（I/O Level标FT的就是5V电平兼容的）。

STM32的每个IO端口都有7个寄存器来控制。他们分别是：配置模式的2个32位的端口配置寄存器CRL和CRH；2个32位的数据寄存器IDR和ODR；1个32位的置位/复位寄存器BSRR；一个16位的复位寄存器BRR；1个32位的锁存寄存器LCKR。大家如果想要了解每个寄存器的详细使用方法，可以参考《STM32中文参考手册V10》P105~P129。

CRL和CRH控制着每个IO口的模式及输出速率。

<table>
<tr><th colspan="2">配置模式</th><th>CNF1</th><th>CNF0</th><th colspan="2">MODE1　MODE0</th><th>PxODR寄存器</th></tr>
<tr><td rowspan="2">通用输出</td><td>推挽式(Push-Pull)</td><td rowspan="2">0</td><td>0</td><td colspan="2" rowspan="4">01
10
11
见表3.1.2</td><td>0 或 1</td></tr>
<tr><td>开漏(Open-Drain)</td><td>1</td><td>0 或 1</td></tr>
<tr><td rowspan="2">复用功能输出</td><td>推挽式(Push-Pull)</td><td rowspan="2">1</td><td>0</td><td>不使用</td></tr>
<tr><td>开漏(Open-Drain)</td><td>1</td><td>不使用</td></tr>
<tr><td rowspan="4">输入</td><td>模拟输入</td><td rowspan="2">0</td><td>0</td><td colspan="2" rowspan="4">00</td><td>不使用</td></tr>
<tr><td>浮空输入</td><td>1</td><td>不使用</td></tr>
<tr><td>下拉输入</td><td rowspan="2">1</td><td rowspan="2">0</td><td>0</td></tr>
<tr><td>上拉输入</td><td>1</td></tr>
</table>

STM32的IO口位配置表

MODE[1:0]	意义
00	保留
01	最大输出速度为10MHz
10	最大输出速度为2MHz
11	最大输出速度为50MHz

STM32输出模式配置表

接下来我们看看端口低配置寄存器CRL的描述，如图：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:30 27:26 23:22 19:18 15:14 11:10 7:6 3:2	**CNFy[1:0]**：端口x配置位(y = 0…7) 软件通过这些位配置相应的I/O端口，请参考表15端口位配置表。 在输入模式(MODE[1:0]=00)： 00：模拟输入模式 01：浮空输入模式(复位后的状态) 10：上拉/下拉输入模式 11：保留 在输出模式(MODE[1:0]>00)： 00：通用推挽输出模式 01：通用开漏输出模式 10：复用功能推挽输出模式 11：复用功能开漏输出模式
位29:28 25:24 21:20 17:16 13:12 9:8, 5:4 1:0	**MODEy[1:0]**：端口x的模式位(y = 0…7) 软件通过这些位配置相应的I/O端口，请参考表15端口位配置表。 00：输入模式(复位后的状态) 01：输出模式，最大速度10MHz 10：输出模式，最大速度2MHz 11：输出模式，最大速度50MHz

端口低配置寄存器CLR

该寄存器的复位值为0X4444 4444，从图6.1.1可以看到，复位值其实就是配置端口为浮空输入模式。从上图还可以得出：STM32的CRL控制着每组IO端口（A~G）的低8位的模式。每个IO端口的位占用CRL的4个位，高两位为CNF，低两位为MODE。这里我们可以记住几个常用的配置，比如0X0表示模拟输入模式（ADC用）、0X3表示推挽输出模式（做输出口用，50M速率）、0X8表示上/下拉输入模式（做输入口用）、0XB表示复用输出（使用IO口的第二功能，50M速率）。

CRH的作用和CRL完全一样，只是CRL控制的是低8位输出口，而CRH控制的是高8位输出口。这里我们对CRH就不做详细介绍了。

给个实例，比如我们要设置PORTC的11位为上拉输入，12位为推挽输出。代码如下：

```
GPIOC->CRH&=0XFFF00FFF;//清掉这2个位原来的设置，同时也不影响其他位的设置
GPIOC->CRH|=0X00038000;  //PC11输入，PC12输出
GPIOC->ODR=1<<11;//PC11上拉
```

通过这3句话的配置，我们就设置了PC11为上拉输入，PC12为推挽输出。

IDR是一个端口输入数据寄存器，只用了低16位。该寄存器为只读寄存器，并且只能以16位的形式读出。该寄存器各位的描述如图6.1.2所示：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															

端口输入数据寄存器IDR

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

位31:16	保留，始终读为0。
位15:0	**IDRy[15:0]**：端口输入数据(y = 0…15) 这些位为只读并只能以字(16位)的形式读出。读出的值为对应I/O口的状态。

要想知道某个IO口的状态，你只要读这个寄存器，再看某个位的状态就可以了。使用起来是比

较简单的。ODR是一个端口输出数据寄存器，也只用了低16位。该寄存器为可读写，从该寄存器读出来的数据可以用于判断当前IO口的输出状态。而向该寄存器写数据，则可以控制某个IO口的输出电平。该寄存器的各位描述如图6.1.3所示：

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
保留															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

位31:16	保留，始终读为0。
位15:0	**ODRy[15:0]**：端口输出数据(y = 0…15) (Port output data) 这些位可读可写并只能以字(16位)的形式操作。 注：对GPIOx_BSRR(x = A…E)，可以分别地对各个ODR位进行独立的设置/清除。

端口输出数据寄存器ODR各位描述

了解了这几个寄存器，我们就可以开始真正的配置寄存器了。下面我们来几个例子：

```
void led_init()
{
RCC->APB2ENR|=1<<3;      //使能PORTB时钟,我们在使用任何一个外设的时候都要先使能他的时钟，stm32的时钟是一个比较复杂的系统，在这里不细讲，但是这是一个非常重要的概念，想学的同学一定要重视它。
RCC->APB2ENR|=1<<6;      //使能PORTE时钟
GPIOB->CRL&=0XFF0FFFFF;
GPIOB->CRL|=0X00300000;//PB.5 推挽输出
GPIOB->ODR|=1<<5;        //PB.5 输出高
GPIOE->CRL&=0XFF0FFFFF;
GPIOE->CRL|=0X00300000;//PE.5推挽输出
GPIOE->ODR|=1<<5;        //PE.5输出高
}
```

该函数的功能就是用来实现配置PB5和PE5为开漏输出。使用寄存器配置 IO口我们就将到这里，下面我们重点讲解使用stm32的固件库来配置和编程实现流水灯的效果。

简单的说固件库就是更加方便我们配置 stm32 的硬件，我们可以不用去查询每个寄存器的值表示什么，就能完成对他的配置。ST 公司官方推出的固件库将对寄存器的操作封装成的函数，也就是说开发人员只需要调用对应的函数就可以操作底层的寄存器了，提高了开发效率。但是不能因为有了固件库就可以说不要学习寄存器，只有两者配合学习才能将 32 学精。GPIO 相关的函数和定义分布在固件库文件 stm32f10x_gpio.c 和头文件 stm32f10x_gpio.h 文件中。

在固件库开发中，操作寄存器CRH和CRL来配置IO口的模式和速度是通过GPIO初始化函数完成：

void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)；

这个函数有两个参数，第一个参数是用来指定GPIO，取值范围为GPIOA~GPIOG。

第二个参数为初始化参数结构体指针，结构体类型为GPIO_InitTypeDef。下面我们看看这个结构体的定义。首先我们打开我们的实验程序，然后找到FWLib组下面的stm32f10x_gpio.c文件，定位到GPIO_Init函数体处，双击入口参数类型GPIO_InitTypeDef后右键选择“Go to definition of …”可以查看结构体的定义：

```
typefef sstruct
{   uint16_t GPIO_Pin;
GPIOSpeed_TypeDef GPIO_Speed;
```

GPIOMode_TypeDef GPIO_Mode;
} GPIO_InitTypeDef;
下面我们通过一个GPIO初始化实例来讲解这个结构体的成员变量的含义。
通过初始化结构体初始化GPIO的常用格式是：
GPIO_InitTypeDef GPIO_InitStructure;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;//LED0-->PB.5 端口配置
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP; //推挽输出
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;//速度50MHz
GPIO_Init(GPIOB, &GPIO_InitStructure);//根据设定参数配置GPIO
上面代码的意思是设置GPIOB的第5个端口为推挽输出模式，同时速度为50M。从上面初始化代码可以看出，结构体GPIO_InitStructure的第一个成员变量GPIO_Pin用来设置是要初始化哪个或者哪些IO口；第二个成员变量GPIO_Mode是用来设置对应IO端口的输出输入模式，这些模式是上面我们讲解的8个模式，在MDK中是通过一个枚举类型定义的：
typedef enum
{ GPIO_Mode_AIN = 0x0,//模拟输入
GPIO_Mode_IN_FLOATING = 0x04,//浮空输入
GPIO_Mode_IPD = 0x28,//下拉输入
GPIO_Mode_IPU = 0x48,//上拉输入
GPIO_Mode_Out_OD = 0x14,//开漏输出
GPIO_Mode_Out_PP = 0x10,//通用推挽输出
GPIO_Mode_AF_OD = 0x1C,//复用开漏输出
GPIO_Mode_AF_PP = 0x18//复用推挽
}GPIOMode_TypeDef;
第三个参数是IO口速度设置，有三个可选值，在MDK中同样是通过枚举类型定义：
typedef enum
{
GPIO_Speed_10MHz = 1,
GPIO_Speed_2MHz,
GPIO_Speed_50MHz
}GPIOSpeed_TypeDef;
要想知道某个IO口的电平状态，你只要读这个寄存器，再看某个位的状态就可以了。使用起来是比较简单的。
在固件库中操作IDR寄存器读取IO端口数据是通过GPIO_ReadInputDataBit函数实现的：
uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
比如我要读GPIOA.5的电平状态，那么方法是：
GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_5);
返回值是1(Bit_SET)或者0(Bit_RESET);
在STM32固件库中，通过BSRR和BRR寄存器设置GPIO端口输出是通过函数
GPIO_SetBits()和函数GPIO_ResetBits()来完成的。
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
在多数情况下，我们都是采用这两个函数来设置GPIO端口的输入和输出状态。比如我们要设置GPIOB.5输出1，那么方法为：
GPIO_SetBits(GPIOB, GPIO_Pin_5);
反之如果要设置GPIOB.5输出位0，方法为：
GPIO_ResetBits(GPIOB, GPIO_Pin_5);

GPIO相关的函数我们先讲解到这里。虽然IO操作步骤很简单，这里我们还是做个概括性的总结，操作步骤为：

1） 使能IO口时钟。调用函数为RCC_APB2PeriphClockCmd()。

2） 初始化IO参数。调用函数GPIO_Init();

3） 操作IO。

下面我们就来编程实现流水灯的效果。

星星点灯实验我们主要用到的固件库文件是：

stm32f10x_gpio.c /stm32f10x_gpio.h

stm32f10x_rcc.c/stm32f10x_rcc.h

misc.c/misc.h

stm32f10x_usart /stm32f10x_usart.h

其中stm32f10x_rcc.h头文件在每个实验中都要引入，因为系统时钟配置函数以及相关的外设时钟使能函数都在这个其源文件stm32f10x_rcc.c中。stm32f10x_usart.h和misc.h头文件在我们SYSTEM

右图为主函数；

包括

```
#ifndef _LED_H
#define  _LED_H

#include "sys.h"
#include "delay.h"              // 包 含 使用文件

#define  LED1 PBout(7)          // 宏定义 意思是编译遇到 LED1 全部用 PBout(7)来替换
#define  LED2 PBout(8)
#define  LED3 PBout(9)
void LED_Init(void);
void LED_SHOW(void);
#endif
```

```
#include "LED.H"
#include "Light.H"
#include "stdio.h"
#include "math.h"
#include "string.h"
void LED_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;                  //推挽输出
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_9|GPIO_Pin_8|GPIO_Pin_7;   //配置所需引脚
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;                     //配置工作时的速度
    GPIO_Init(GPIOB,&GPIO_InitStructure);                               //初始GPIO
    GPIO_SetBits(GPIOB,GPIO_Pin_9|GPIO_Pin_8|GPIO_Pin_7);               //初始化为高电平
}
/*******************************************************************************
*                函数名    : LED_SHOW
*                函数功能 : LED显示
*                输入      : 无
*                输出      : 无
*                说明      : 无
*******************************************************************************/
void LED_SHOW(void)
{
    u16 time,n;
    for(n=3;n>0;n--)
        for(time=3;time>0;time--)
        {
```

```
            switch(time)
            {
                case(1)   :   LED1=0;LED2=LED3=1;    delay_ms(500);         break;
                case(2)   :   LED2=0;LED1=LED3=1;    delay_ms(500);         break;
                case(3)   :   LED3=0;LED1=LED2=1;    delay_ms(500);         break;
            }
        }
    for(time=500;time>0;time-=50)
    {
        LED1=LED2=LED3=0;
        delay_ms(time);
        LED1=LED2=LED3=1;
        delay_ms(time);
    }
}}此处在led.h里面最关键就是4个宏定义：
#define   LED1 PBout(7)          //宏定义 意思是编译遇到LED1全部用PBout(7)来替换
#define   LED2 PBout(8)
#define   LED3 PBout(9)
```

这里使用的是位带操作来实现操作某个IO口的1个位的。这里我们来讲解一下，操作IO口输出高低电平的三种方法。

通过位带操作PB5输出高低电平从而控制LED0的方法如下：

LED1=1; //通过位带操作控制LED0的引脚PB6输出高电平

LED1=0; //通过位带操作控制LED0的引脚PB6输出低电平

同样我们也可以使用固件库操作和寄存器操作来实现IO口操作。库函数操作方法如下：

GPIO_SetBits(GPIOB,GPIO_Pin_5);//设置GPIOB.5输出1,等同LED0=1;

GPIO_ResetBits(GPIOB,GPIO_Pin_5);//设置GPIOB.5输出0,等同LED0=0;

到这里我们就简单的讲解完的如何操作stm32的GPIO，希望对大家的深入学习有帮助。

实验三 按键显示

一、实验目的：

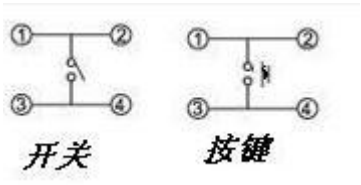
1、按键显示的原理

2、按键控制 LED 状态

二、实验器材：

STM32F103C8T6 单片机，按键 3 个，稳压电源，跳线，插槽，电烙铁，锡丝若干。

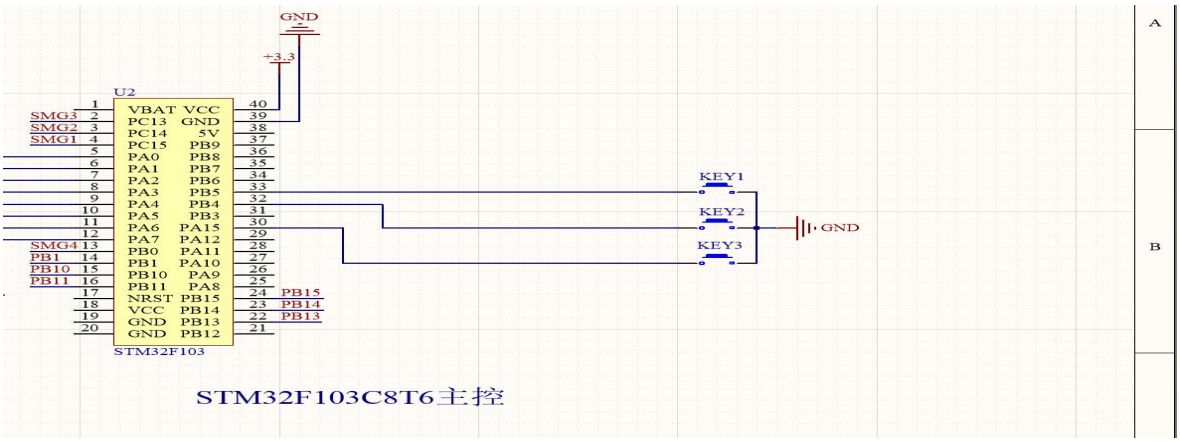
三、按键原理：



四脚按键开关的连接

上图是接线图，1、2 脚是一端，3、4 脚是另一端，不同厂家标示不同。轻触开关是按一下接通电路，手松开后电路依然接通，再按一次电路断开；轻触按键是按下去接通电路，手松开电路就断开。

四、实验电路图：



四、软件设计：

```
#include "KEY.H"
void KEY_INIT(void)
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOA, ENABLE);  //使能 GPIOB 时钟
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_IN_FLOATING;      //浮空输入
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;          //配置速度
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_4|GPIO_Pin_6;       //配置引脚
    GPIO_Init(GPIOB,&GPIO_InitStructure);
    GPIO_SetBits(GPIOB,GPIO_Pin_4|GPIO_Pin_6);
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_15;                 //配置引脚
    GPIO_Init(GPIOA,&GPIO_InitStructure);
    GPIO_SetBits(GPIOA,GPIO_Pin_15);
}
```

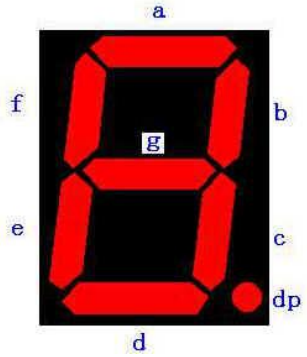
实验四 数码管显示

一、实验目的：

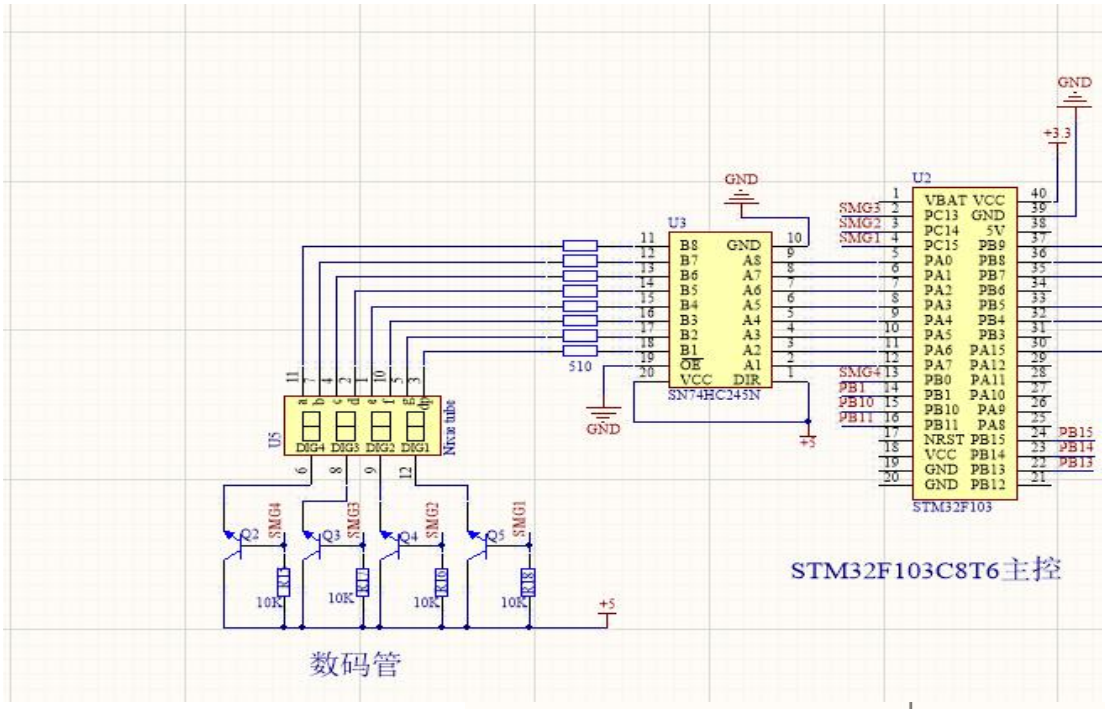
初步认识 STM32F103C8T6 这款单片机的各个引脚及功能；学会如何配置 I/O 端口；学习如何新建 stm32 程序工程以及使用 MDK5 软件编写程序并下载。

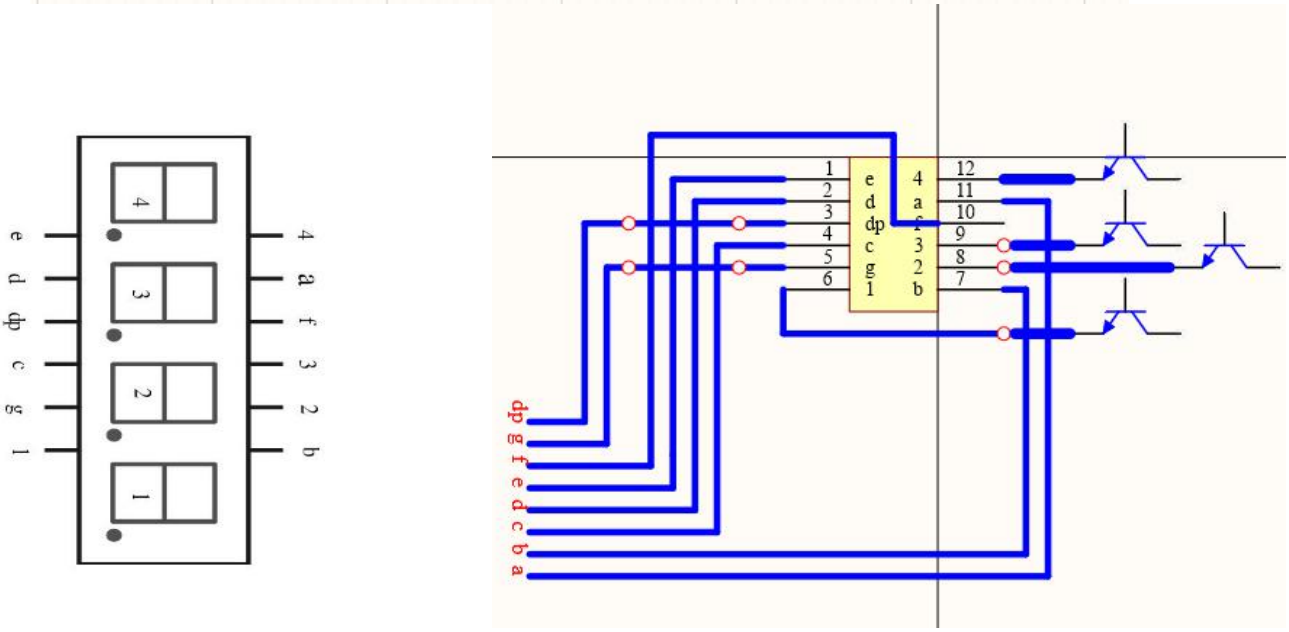
二、实验器材

STM32F103C8T6 单片机一块，大号万用板一块，500 欧电阻四个，，发光二极管四个，电烙铁一把，排插，导线，锡丝若干。



三、实验电路图





注意：看清楚数码的小数点的位置（上面两个图对照），数码管跳线参考焊接方式，粗线是借助数码管的长管脚从洞洞板上方绕过焊接的锡线!!!

动态显示驱动：下面我们介绍一下数码管的显示原理,每个数码管共有八位,分别是 A、B、C、D、E、F、G、DP（即小数点），每一位实际上相当于一个发光二极管。当数码管的共阳极接电源正极后，此时如果把对应每一位的数码管引脚接电源的负极则相应的位便会发光（当然要注意选择合适的电压否则会烧坏数码管），当我们让这八位中的某些位按照一定的规则同时发光则可以显示出我们所需要的数字。

接下来我们要做的是让多个数码管同时闪亮，在星星点灯实验中当我们将延时时间减小到一定程度时我们便会观察到四个二极管同时发光，这就是利用了人的视觉暂留现象。其实这四个二极管并不是同时发光的，只是因为延时时间过短使得它们发光太快以至于我们的视觉根本无法区分，利用这个原理我们也可以让多个数码管看起来同时发光。

从图中我们可以看到：三个共阳极双数码管的每一位（A、B、C……）是连在一起的，而六个三极管则相当于每一数码管（每一个双数码管是由 L/R 两个数码管组成）的开关，这就相当于

将六个数码管并联并在每一并联支路上加一开关，当开关闭合时则该支路的数码管将显示出数字。三极管的开关作用是利用控制基极电位来实现的，当基极加高电位，则基极与发射极间的 PN 结将被打开，当基极加低电位时不能打开该 PN 结。本实验中我们是用 P1 来控制基极电位的。
如图所示，A——PA0；B——PA1；C——PA2；D——PA3；E——PA4；F——PA5；G——PA6；DP——PA7；自左往右控制六个数码管的 PA 引脚分别是 PA5、PA4、PA3、PA2、PA1、PA0；
下面来介绍如何用数码管的八位来显示数值，以显示“1”为例，当显示“1”时我们需要让位 B 与位 C 亮，其余位不亮，因此我们要将 PA1 与 PA2 置 0（低电平），其余的设置为 1（高电平），所以 PA=1111 1001，即十六进制下的 PA=0XF9;同样道理我们可以得到：
0——0XC0（1100 0000）；1——0XF9（1111 1001）；2——0XA4（1010 0100）；3——0XB0（1011 0000）；4——0X99（1001 1001）；5——0X92（1001 0010）；6——0X82（1000 0010）；7——0XF8（1111 1000）；8——0X80（1000 0000）；9——0X90（1001 0000）；

此外，由于我们采用的是四段数码管，因此我们还需要利用四个引脚来控制对应的数码管开启，来保证四个数码管可以同时工作显示数字，在这里我们使用的是 PC15、PC14、PC13、PB0，分别用来控制四位数码管的显示，因此我们可以定义一个数组，用来实现数字的循环改变，如下：

数组中的每一个数字分别代表一个数码管显示的数。

四、软件设计：

```
#include "Nixie tube.H"
void Nixie_tube_Init(void)
{
    GPIO_InitTypeDef     GPIO_InitTStrctrue;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB|RCC_APB2Periph_GPIOC,ENABLE);//使能时钟
    GPIO_InitTStrctrue.GPIO_Mode=GPIO_Mode_Out_PP;                          //推挽输出
    GPIO_InitTStrctrue.GPIO_Speed=GPIO_Speed_50MHz;                         //配置速度

    GPIO_InitTStrctrue.GPIO_Pin=GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7;//配置引脚
    GPIO_Init(GPIOA,&GPIO_InitTStrctrue);

    GPIO_InitTStrctrue.GPIO_Pin=GPIO_Pin_0;                                 //配置引脚
    GPIO_Init(GPIOB,&GPIO_InitTStrctrue);                                   //初始化引脚

    GPIO_InitTStrctrue.GPIO_Pin=GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15;    //配置引脚
    GPIO_Init(GPIOC,&GPIO_InitTStrctrue);                                   //初始化引脚

    GPIO_SetBits(GPIOA,GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3|GPIO_Pin_4|GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7);//初始化引脚电平
    GPIO_ResetBits(GPIOB,GPIO_Pin_0);                                       //初始化引脚电平
    GPIO_ResetBits(GPIOC,GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15);              //初始化引脚电平
}
/*******************************************************************************
*                    函数名    ：Nixie_tube_Show
*                    函数功能  ：显示数字
```

```
*                    输入      ：对应要显示的数字
*                    输出      ：无
*                    说明      ：从左到右分别显示的数码管位 4-->1
*******************************************************************************/
void Nixie_tube_Show(u8 chioce,u8 number)
{
    switch(chioce)
    {
        case(4)   :   SMG3=1;  SMG1=SMG2=SMG4=0;   break;
        case(1)   :   SMG4=1;  SMG1=SMG2=SMG3=0;   break;
        case(2)   :   SMG1=1;  SMG2=SMG3=SMG4=0;   break;
        case(3)   :   SMG2=1;  SMG1=SMG3=SMG4=0;   break;
    }
    GPIO_Write(GPIOA,number);                          //写入数据
    delay_ms(2);                                       //延时一段时间
}
```

实验效果：我们可以看到四段数码管每隔 500ms，轮流同步显示数字循环，从 0~9 循环显示

五、实验总结：

虽然本次实验的原理与前面的大致相同，但是在硬件电路上明显的复杂了很多，所以在焊接硬件部分的时候我们一定要细心，如果只求速度而粗心的焊接好实验板，那么如果一旦线路焊接有问题，那就要花费更大的精力去检查错误，所以请大家一定要细心看电路图，耐心焊接电路——细节决定成败。

实验五 光敏计数

一：实验目的

通过实验，掌握光敏接收二极管，LM393 运算放大器的工作原理，并熟练运用。了解并掌握 C8051f310 单片机的计数器及中断功能，并实现程序控制。

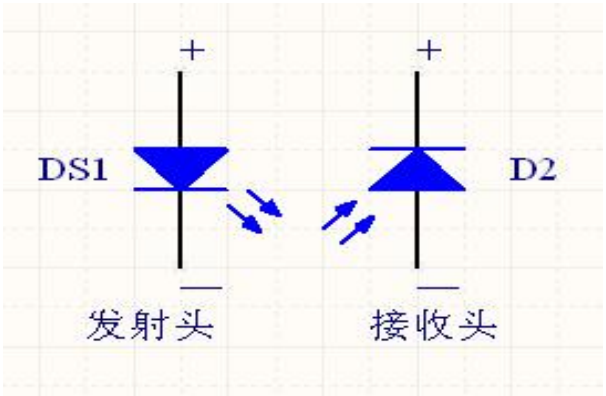
二：实验器材

STM32F103C8T6 单片机，LED 显示模板，光敏接收二极管，LM393，电位器各一个，电源，杜邦线若干。

三：实验原理

1. 光敏发射接收器

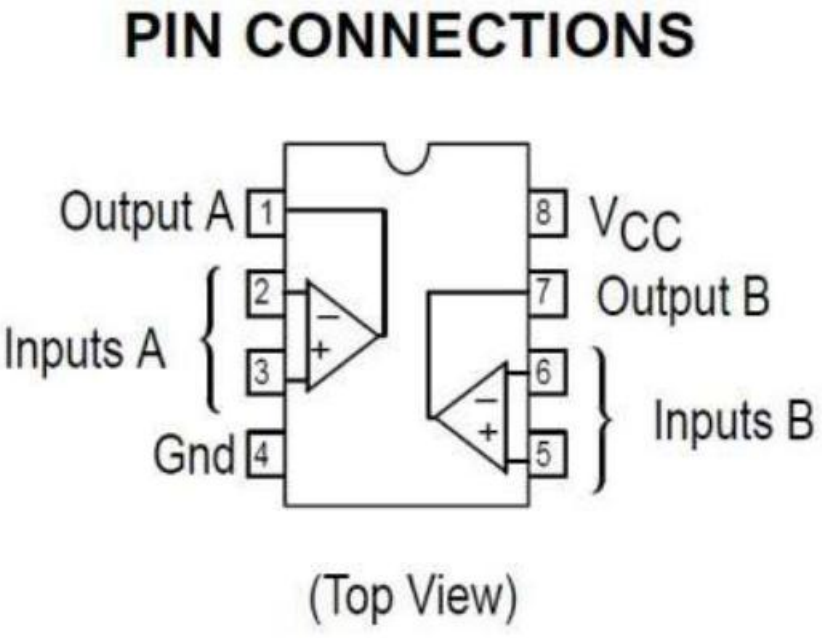
光敏发射接收器由一个发射头和一个接收头组成，发射头是白色的，接受头是平头的，原理图如下所示：



当发射头 DS1 导通后，会发射出白光；接受头 D2 在没有接收到光源的时候，电阻非常大，趋于截止（断开）状态，一旦接收到光线，电阻会迅速减小，阻值与光线强度大小有关系，光线越强阻值越小（可趋于 0），此时 D2 处于导通状态，所以实际应用中要加限流电阻。

2． LM393

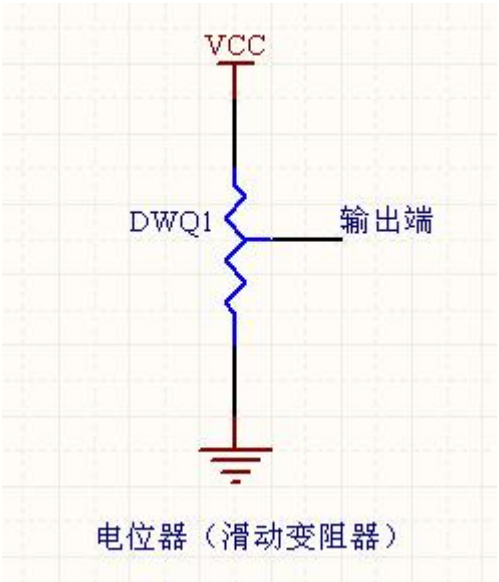
LM393 为双电压比较器集成电路，采用 8 脚双列直插塑料封装。它的内部包含两组形式完全相同的运算放大器，除电源共用外，两组运放相互独立。 如图所示：



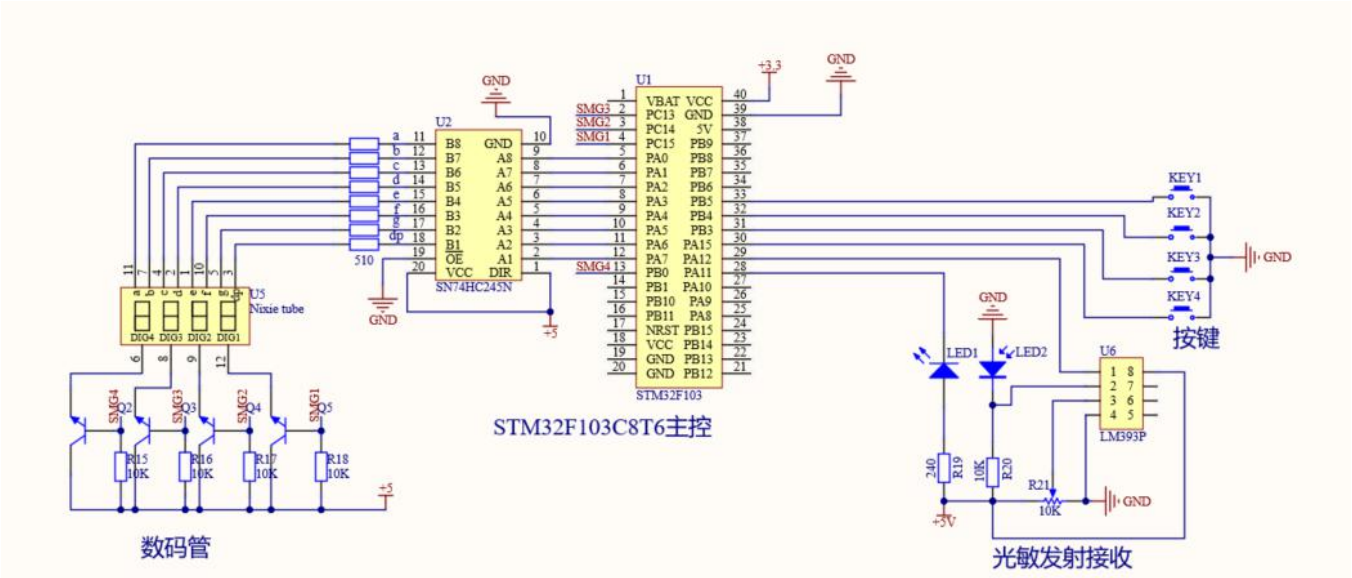
芯片 LM393 一般采用 5V 供电。在每一路运算放大器中，+号表示同相输入端，－号表示反相输入端。通常情况下，当同相输入端为某一中间电压值（如 2V），如果反相输入端电压高于同相输入端，则输出低电平，如果反相输入端电压低于通向输入端，则输出高电平。（同高为高，同低为低）

3． 电位器

利用电位器分压原理可以输出我们需要的电压。



5.实验电路图



实验六 PWM 控制直流电动机

一、实验目的

通过实验，掌握控制电机的 L298 芯片工作原理，ULN2003 及 74HC04 的工作原理，并都能熟练运用。了解并掌握 stm32f103 单片机脉宽调制（pwm）控制输出功能，并实现程序控制。

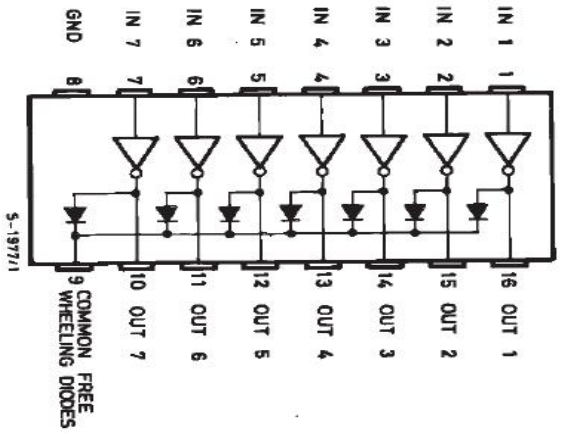
二、实验器材

stm32f103 单片机一个，ULN2003（或 74HC04）一个，直流电动机一个（或两个），单片机电源，电机电源，跳线数条。

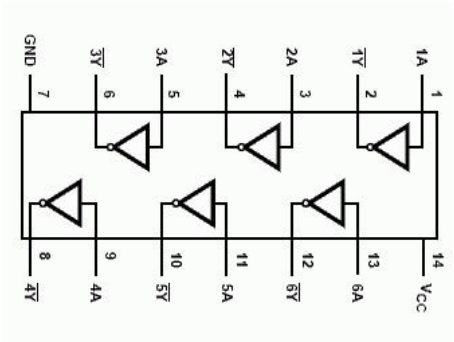
三、实验原理

1．ULN2003

芯片 ULN2003 同 74HC04 都是反相器，ULN2003 的内部结构为：



74HC04 的内部结构为：



他们的工作电压都为 5V，IN/A 为输入端，OUT/Y’为输出端。在电路中一般要接上拉电阻，反相器有两个基本功能：

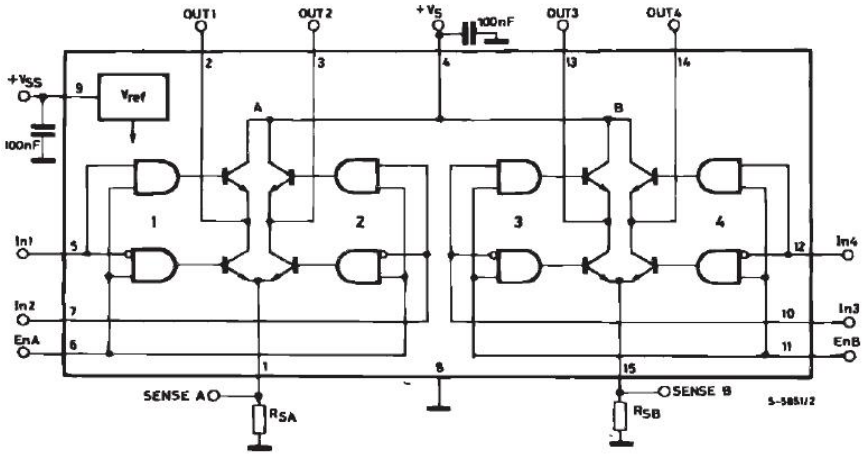
a．输入输出反相功能。输入高电平则输出低电平，输入低电平则输出高电平。

b．提高驱动的功能。用单片机直接驱动某个大元件可能会驱动能力不好或者对会单片机造成一定的影响，可以用反相器在中间提供驱动，只不过，反了一下相而已。

2．L298

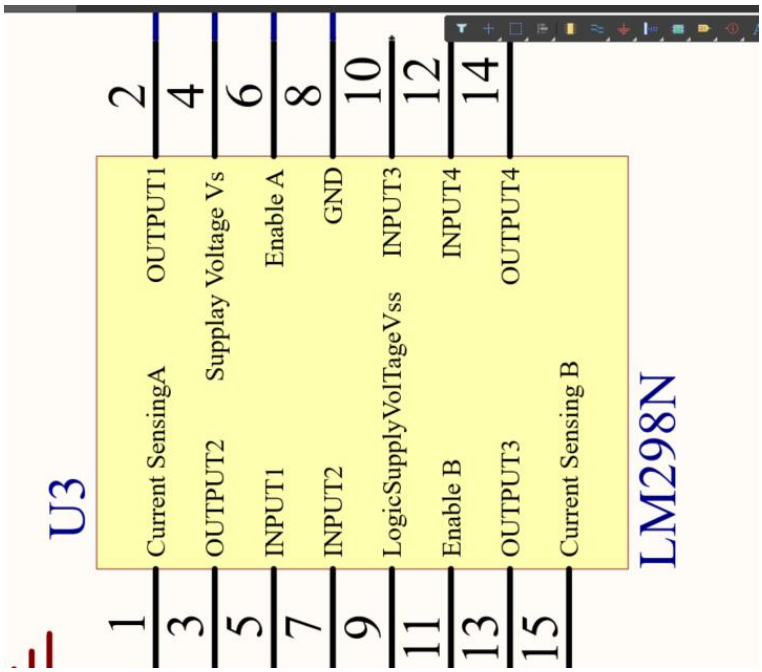
芯片 L298 内部主要利用晶体三极管（NPN）和与门组成的 H 桥式控制结构，实现电机的转速转向控制。

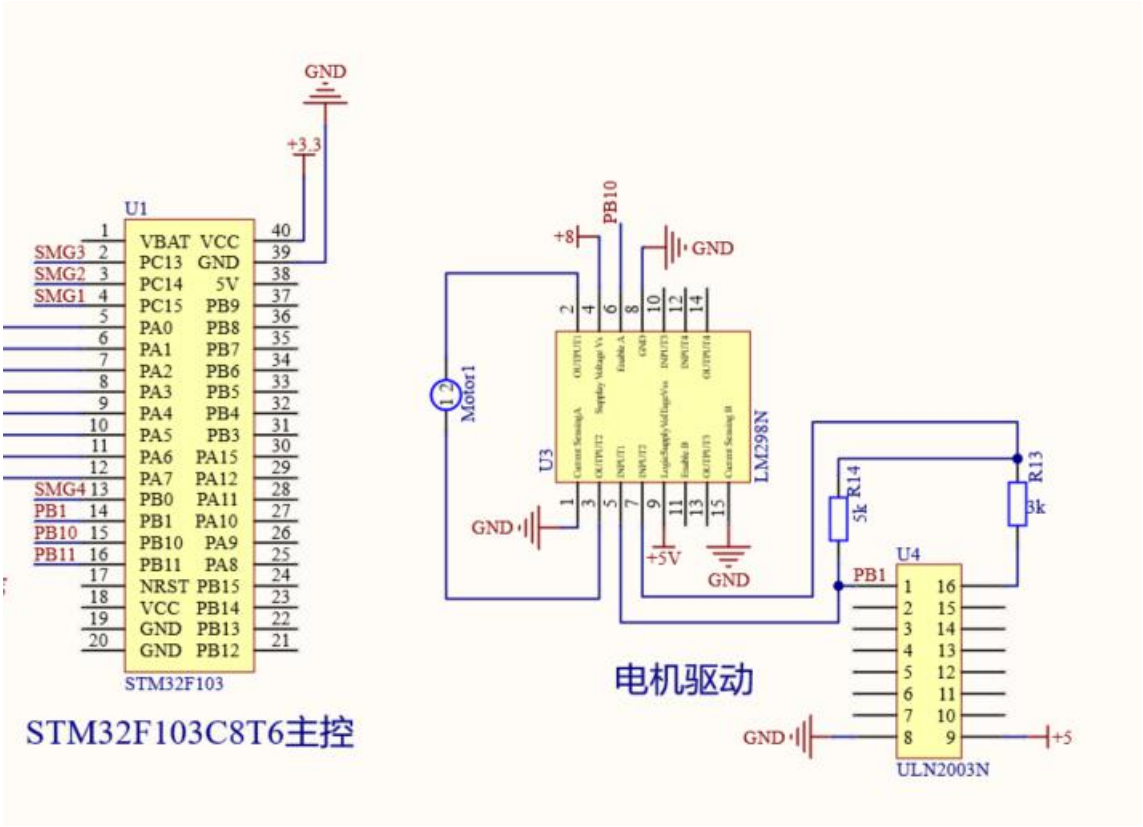
芯片内部电路结构图：



从图中可以看出，L298 内部有两路相同的电机控制电路（H 桥结构），分左右两半。以左边为例，脚 1 是电机运行时电流测试端，串联的电阻 R 是很小的，几欧左右，平时不测电流时直接接地；脚 2 和脚 3 是控制电机端，接电机；脚 4 是电压输入端，电机的额定电压是多大，一般就在此端接多大电压，但要注意不要超过 L298 的耐压值；脚 5(IN1)和脚 7(IN2)是电机运转方向控制端，一般情况下，脚 5 和脚 7 是反相的，所有我们用到反相器，例如单片机某个脚给一个高电平，输出接两根线，一根直接接脚 5(IN1)，另一根接反相器，反向后输出低电平接脚 7(IN2)，就实现了用一跟单片机管脚控制电机转向；脚 6(ENA)是脉冲输入端，控制电机转速。脚 8 接地，脚 9 接 5V 电源，给芯片供电。右边的电路结构功能和左边是一模一样的。其中 4 脚对左右两边要控制的电机供电，9 脚是对 L298 芯片供电，使芯片正常工作，8 脚是公共地。

3.实验电路图





注：控制一个电机电路连接图其中只用到了
L298 里的一路电机控制电路

4．编程引导介绍

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。 STM32 的定时器除了 TIM6 和 7。其他的定时器都可以用来产生 PWM 输出。其中高级定 时器 TIM1 和 TIM8 可以同时产生多达 7 路的 PWM 输出。而通用定时器也能同时产生

多达 4 路的 PWM 输出，这样，STM32 最多可以同时产生 30 路 PWM 输出！这里我们仅利用 TIM3 的 CH2 产生一路 PWM 输出。如果要产生多路输出，大家可以根据我们的代码稍作修改即可。

同样，我们首先通过对 PWM 相关的寄存器进行讲解，大家了解了定时器 TIM3 的 PWM 原理之后，我们再讲解怎么使用库函数产生 PWM 输出。 要使 STM32 的通用定时器 TIMx 产生 PWM 输出，除了上一章介绍的寄存器外，我们还会 用到 3 个寄存器，来控制 PWM 的。这三个寄存器分别是:捕获/比较模式寄存器（TIMx_CCMR1/2）、捕获/比较使能寄存器（TIMx_CCER）、捕获/比较寄存器（TIMx_CCR1~4）。 接下来我们简单介绍一下这三个寄存器。 首先是捕获/比较模式寄存器（TIMx_CCMR1/2），该寄存器总共有 2 个，TIMx _CCMR1 和 TIMx _CCMR2。TIMx_CCMR1 控制 CH1 和 2，而 TIMx_CCMR2 控制 CH3 和 4。该寄存器 的各位描述如图 14.1.1 所示：

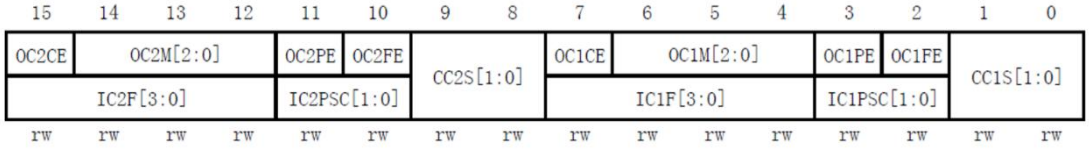


图 14.1.1 TIMx_CCMR1 寄存器各位描述

该寄存器的有些位在不同模式下，功能不一样，所以在图 14.1.1 中，我们把寄存器分了 2 层，上面一层对应输出而下面的则对应输入。关于该寄存器的详细说明，请参考《STM32 参考 手册》第 288 页，14.4.7 一节。这里我们需要说明的是模式设置位 OCxM，此部分由 3 位组成。 总共可以配置成 7 种模式，我们使用的是 PWM 模式，所以这 3 位必须设置为 110/111。这两种 PWM 模式的区别就是输出电平的极性相反。 接下来，我们介绍捕获/比较使能寄存器（TIMx_CCER），该寄存器控制着各个输入输出通 道的开关。该寄存器的各位描述如图 14.1.2 所示：

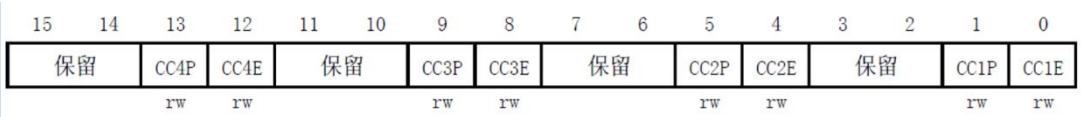


图 14.1.2 TIMx_ CCER 寄存器各位描述

该寄存器比较简单，我们这里只用到了 CC2E 位，该位是输入/捕获 2 输出使能位，要想 PWM 从 IO 口输出，这个位必须设置为 1，所以我们需要设置该位为 1。该寄存器更详细的介 绍了，请参考《STM32 参考手册》第 292 页，14.4.9 这一节。 最后，我们介绍一下捕获/比较寄存器（TIMx_CCR1~4），该寄存器总共有 4 个，对应 4 个 输通道 CH1~4。因为这 4 个寄存器都差不多，我们仅以 TIMx_CCR1 为例介绍，该寄存器的各 位描述如图 14.1.3 所示：

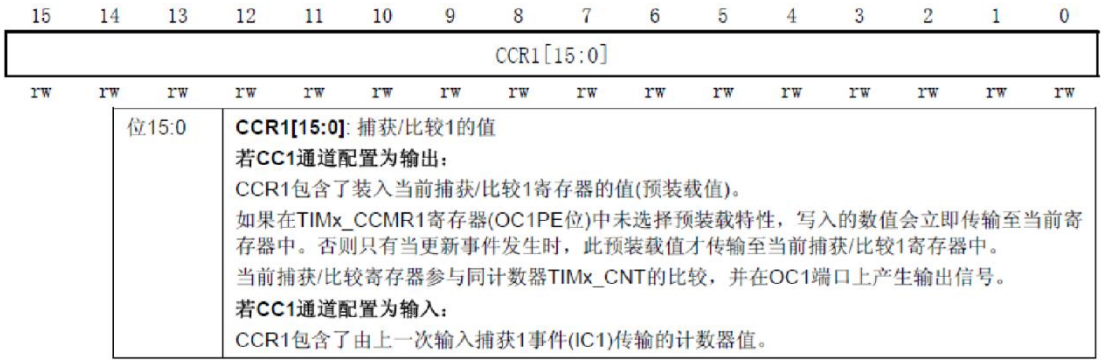


图 14.1.3 寄存器 TIMx_ CCR1 各位描述

在输出模式下，该寄存器的值与 CNT 的值比较，根据比较结果产生相应动作。利用这点， 我们通过修改这个寄存器的值，就可以控制 PWM 的输出脉宽了。本章，我们使用的是 TIM3 的通道 2，所以我们需要修改 TIM3_CCR2 以实现脉宽控制 DS0 的亮度。 我们要利用 TIM3 的 CH2 输出 PWM 来控制 DS0 的亮度，但是 TIM3_CH2 默认是接在 PA7 上面的，而我们的

DS0 接在 PB5 上面，如果普通 MCU，可能就只能用飞线把 PA7 飞到 PB5 上来实现了，不过，我们用的是 STM32，它比较高级，可以通过重映射功能，把 TIM3_CH2 映射到 PB5 上。STM32 的重映射控制是由复用重映射和调试 IO 配置寄存器（AFIO_MAPR）控制的，该 寄存器的各位描述如图 14.1.4 所示：

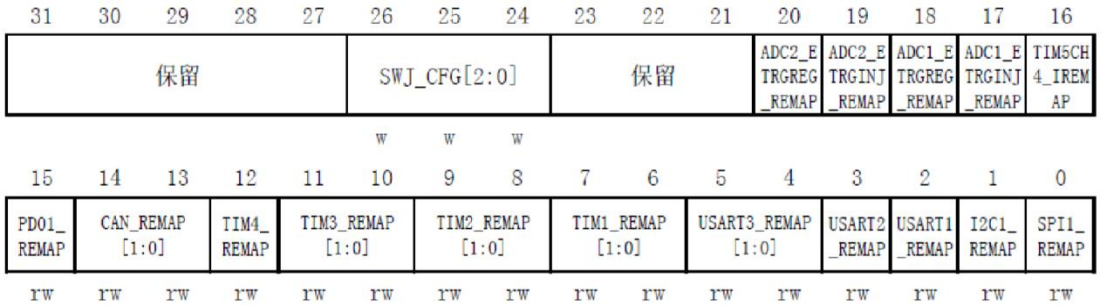


图 14.1.4 寄存器 AFIO_MAPR 各位描述

我们这里用到的是 tim4，详细参考 stm32 参考手册。

四：注意事项

1. 各实验模块之间连线应按照自己的电路图和自己的程序应用 I/O 脚连线，切勿照搬其他范例，以免出错。

2. 硬件连接注意事项

关于 L298

①：需要给 L298 供两个电源，一个是芯片工作电源 5V；一个是给电机供电电源，大小一般是电机额定电压，但不能超过 L298 的耐压值。注意两个电源一定不能接反接错，所有地线可以接到一起。

②：控制大电机或两个电机时，L298 一般会有不同程度发热，建议最好在 L298 上接上散热片。

关于反相器

③：反相器工作时，一般要接上拉电阻，上拉电压为 5V，与芯片工作电压相同。

五：范例程序 **注：本次实验使用的是定时器模拟 PWM**

```
#include "Motor.H"
u8 Speed;                                                    //控制熟读的变量
u8 aim_speed;                                                //速度自加变量
void Motor_Init_GPIO(void)
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB,ENABLE);     //使能 GPIOB 时钟
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;           //推挽输出
    GPIO_InitStructure.GPIO_Pin=GPIO_Pin_1|GPIO_Pin_10;      //配置引脚
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;          //配置速度
    GPIO_Init(GPIOB,&GPIO_InitStructure);
    GPIO_SetBits(GPIOB,GPIO_Pin_1|GPIO_Pin_10);              //配置为低电平
}
void  TIM2_Init(u16 ARR,u16 PSC)
{
    TIM_TimeBaseInitTypeDef       TIM_TimeBaseInitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);      //使能定时器 2 的时钟
```

```
    TIM_TimeBaseInitStructure.TIM_ClockDivision=0;                    //时钟分频系数
    TIM_TimeBaseInitStructure.TIM_CounterMode=TIM_CounterMode_Up;//向上计数模式
    TIM_TimeBaseInitStructure.TIM_Period=ARR;                         //自动重装在值
    TIM_TimeBaseInitStructure.TIM_Prescaler=PSC;                      //时钟预分频系数
    TIM_TimeBaseInit(TIM2,&TIM_TimeBaseInitStructure);
    TIM_ITConfig(TIM2,TIM_IT_Update,ENABLE);                          //开启更新中断
    NVIC_InitStructure.NVIC_IRQChannel=TIM2_IRQn;                     //中断通道
    NVIC_InitStructure.NVIC_IRQChannelCmd=ENABLE;                     //使能中断
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority=1;           //抢占优先级 1
    NVIC_InitStructure.NVIC_IRQChannelSubPriority=3;                  //响应优先级
    NVIC_Init(&NVIC_InitStructure);
    TIM_Cmd(TIM2,DISABLE);                                            //不使能定时器
}
void Motor_Init(u16 ARR,u16 PSC)
{
    Motor_Init_GPIO();                                                //电机 IO 初始化
    TIM2_Init(ARR,PSC);                                               //初始化定时器
}
u8 KEY_Motor_Control(void)
{
    static u8 motor_value=100;                                        //静态变量初始化电机的值
    if(KEY1==0)
    {
        delay_ms(10);                                                 //消抖
        if(KEY1==0)
        {
            motor_value-=20;                                          //值减小
            if(motor_value<=60)       motor_value=0; //限幅，值太小转不动，直接停止转动
            LED3=~LED3;
        }
        while(!KEY1);
    }
    if(KEY2==0)
    {
        delay_ms(10);
        if(KEY2==0)
        {
            motor_value+=20;
            if(motor_value>=230)motor_value=230;
            LED2=~LED2;
        }
        while(!KEY2);
    }
    return motor_value;
}
void TIM2_IRQHandler(void)                                            //中断函数
```

```
{
    if(TIM_GetITStatus(TIM2,TIM_IT_Update)!=RESET)      //判断中断是否符合更新中断条件
    {
        aim_speed++;
        if(aim_speed<=Speed)                            //计数值小于所给的速度值时候 IO 输出高电平
        {
            Motor_RUN=1;
        }
        else                                            //计数值大于所给的速度值时候 IO 输出低电平
        {
            Motor_RUN=0;
        }
        if(aim_speed>=250)   aim_speed=0;               //变量大于 250 时清零
        TIM_ClearITPendingBit(TIM2,TIM_IT_Update);      //清除更新中断标志位
    }
}
```

注意：以 pb5 的高低电平来使电机正反转。

六、应用与扩展

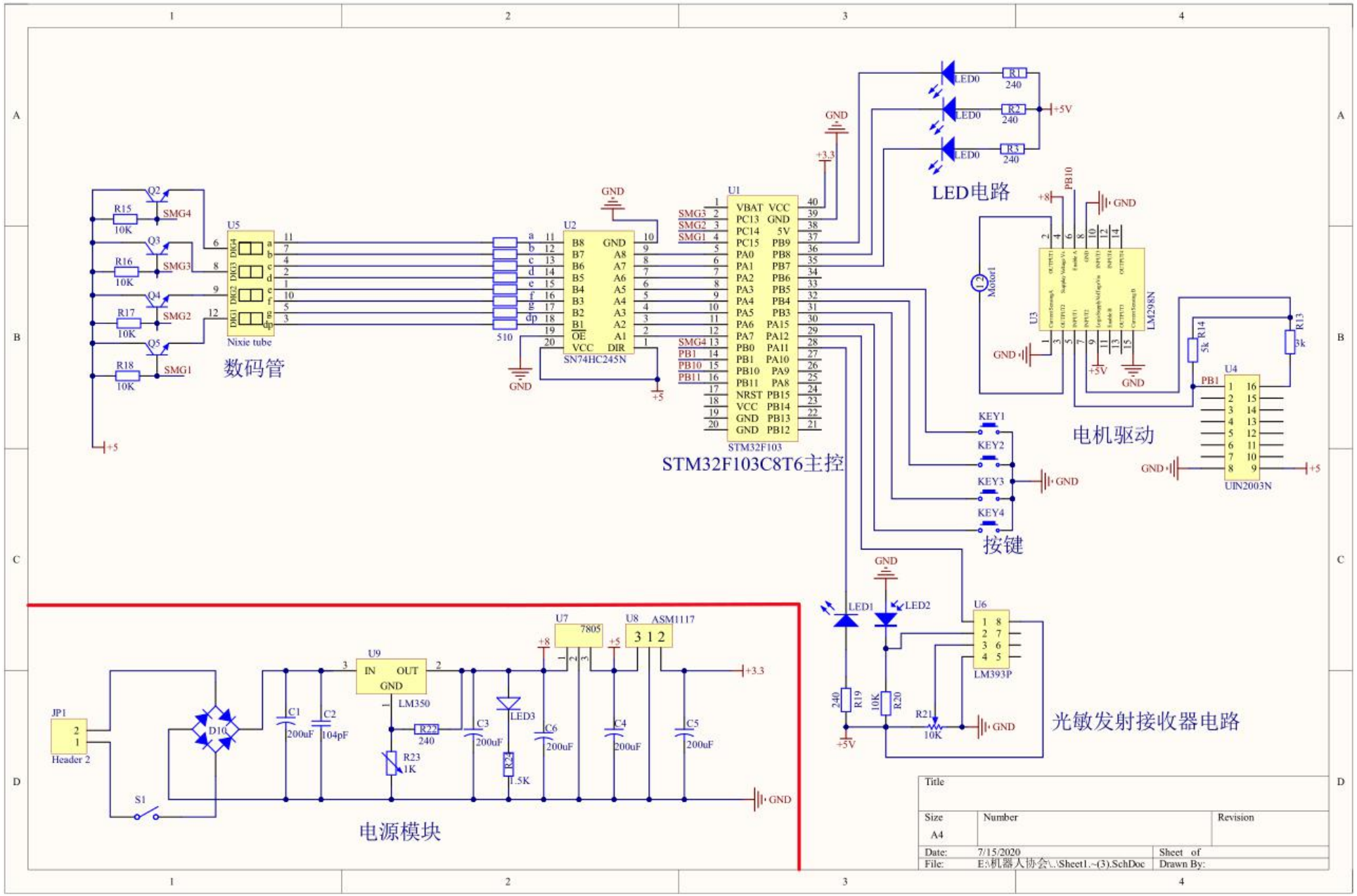
运用 Pwm 脉宽调制进行调速与传统的串电位器分压调速有以下优势：

1.调速范围宽，可以线性调速。电位器调速为非线性调速

2.损耗小，串电位器中电阻消耗大（尤其是在大功率电机中）

3.数字控制，更精准现在大部分的电机调速都是用 pwm 调速，大功率的电机用场效应管做 H 桥，实现大功率驱动。

附、总电路图


LED电路
数码管
STM32F103C8T6主控
电机驱动
按键
光敏发射接收器电路
电源模块
Nixie tube
SN74HC245N
STM32F103
LM298N
UIN2003N
LM393P
LM350
Header 2
Title
Size
A4
Number
Revision
Date: 7/15/2020
File: E:\机器人协会\..\Sheet1.~(3).SchDoc
Sheet of
Drawn By: