牛艾科技

电子钟

Python base program

目录 Contents

第一部分 数码管简介

第二部分 AIP1638简介

第三部分 认识RTC

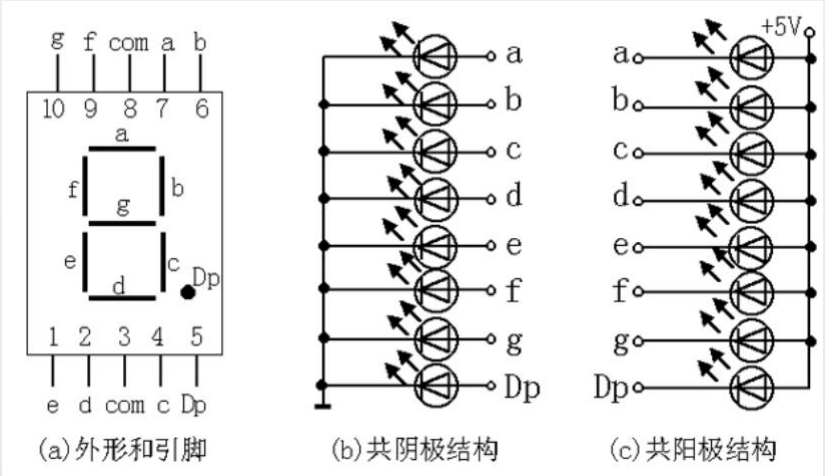
第四部分 电子钟实验

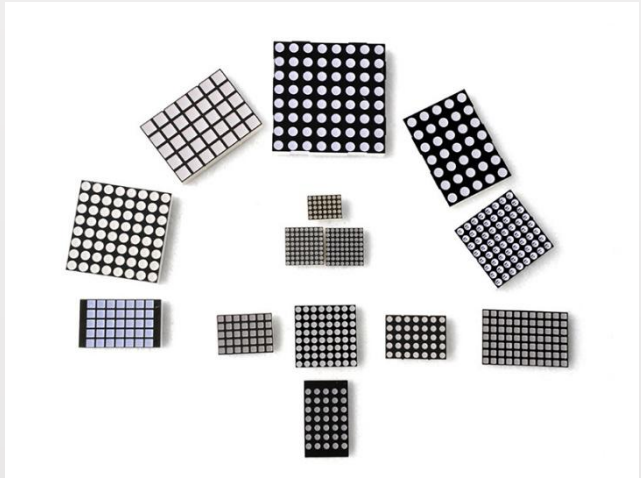
常见的数码管显示

- 电子称，温湿度计，电饭煲，电子日历等
- 还有哪里用到?

7段、8段数码管

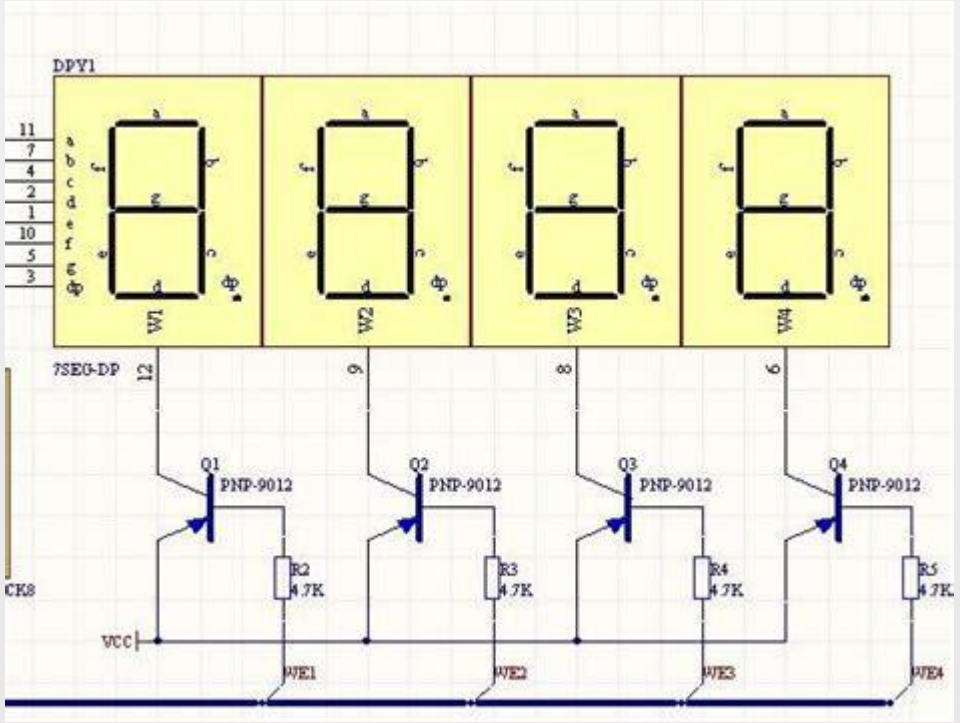
- **数码管**的一种是半导体发光器件，数码管可分为七段数码管和八段数码管，区别在于八段数码管比七段数码管多一个用于显示小数点的发光二极管单元DP（decimal point），其基本单元是发光二极管。结构上分为共阳极和共阴极，另外位数，大小，颜色，显示内容(段数7、8、9..LED数量)等区别。
- 下图为1位数码管的外形和原理结构，右图为其他形态的数码管，LED点阵也属于数码管





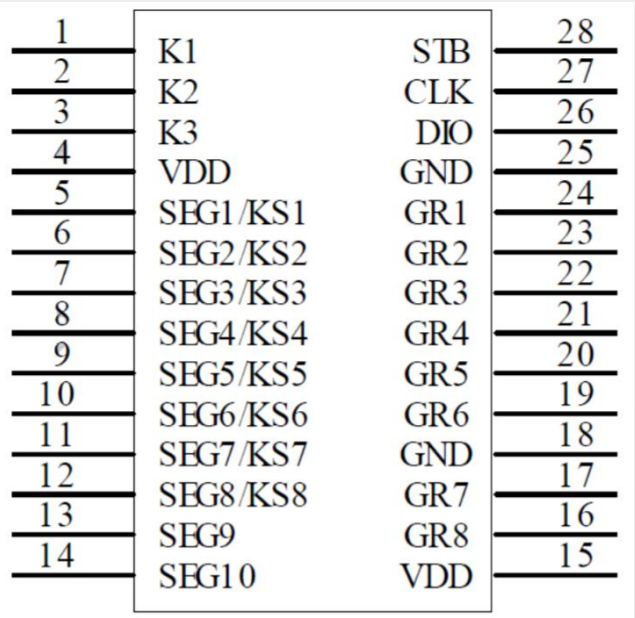
数码管显示控制原理

- 根据LED点阵学习可了解点阵显示需要不断刷新LED才能显示不同内容，数码管也是一样的a-g一般共用，公共端为片选端，一般由三极管控制。参考原理如下图，Q1-Q4分时选通，当Q1选通时，a-g上为低(0)的位置被点亮
- 因为需要不断刷新，所以很占用处理资源
- 那么是否可以做个专门的芯片来刷新呢?
- 答案：AiP1620，AIP1638等。
- 选择AIP1638支持10段8位，



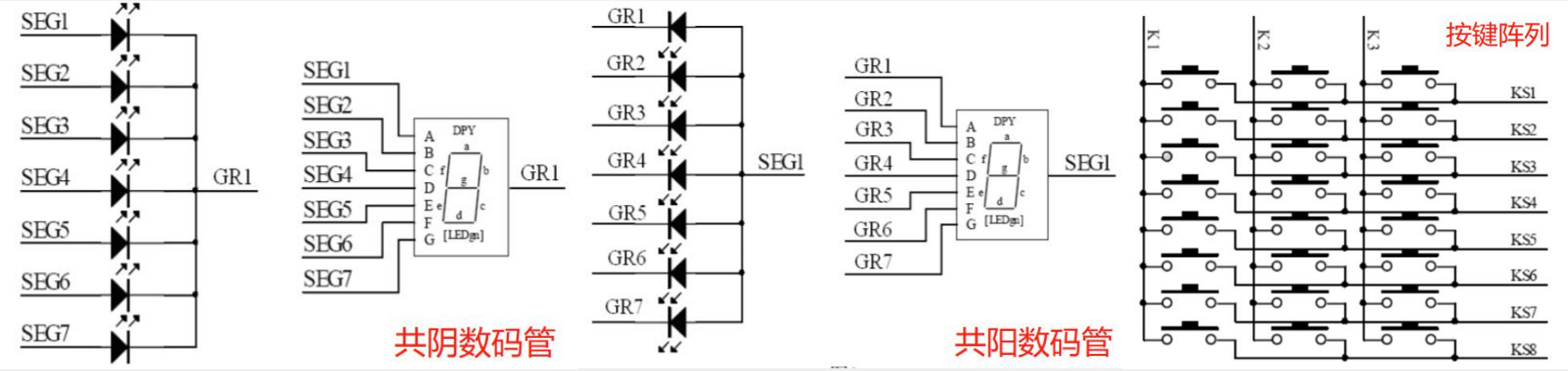
AIP1638

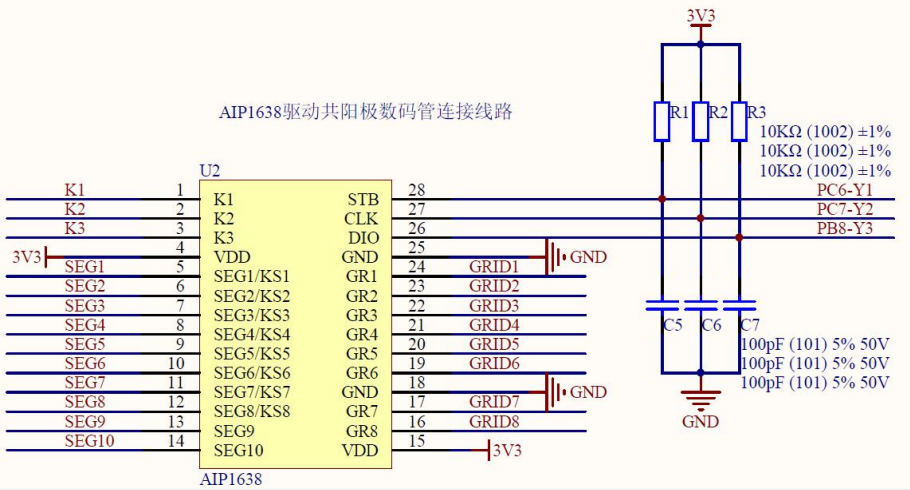
- AIP1638是带键盘扫描接口的LED驱动专用电路，内置键盘扫描接口。MCU接口为3线SPI接口，显示模式最大支持10段x8位。芯片内部有寄存器存储数码管各位、段的显示值，并自动根据寄存器内容刷新数码管。

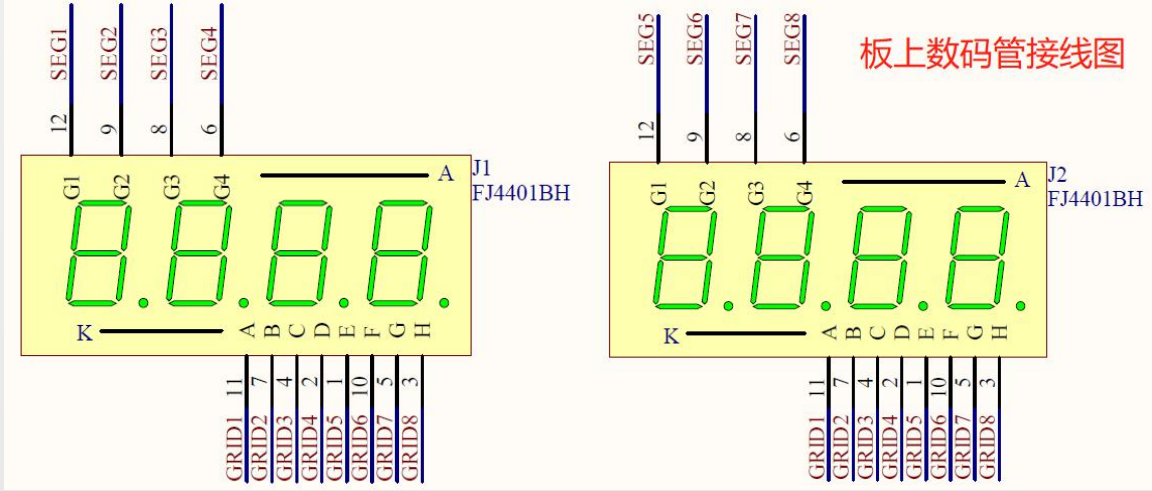


引脚	符　号	引脚名称	功　能
1-3	K1～K3	键扫数据输入	输入该脚的数据在显示周期结束后被锁存
4、15	VDD	逻辑电源	5V±10%
5～12	SEG1/KS1～SEG8/KS8	输出（ 段）	段输出（也用作键扫描）,P管开漏输出
13、14	SEG9、SEG10	输出（ 段）	段输出， P管开漏输出
16、17	GR7、GR8	输出（ 位）	位输出， N管开漏输出
18、25	GND	逻辑地	接系统地
19～24	GR1～GR6	输出（ 位）	位输出， N管开漏输出
26	DIO	数据输入/输出	在时钟上升沿输入/输出串行数据，从低位开始；
27	CLK	时钟输入	上升沿输入/输出串行数据
28	STB	片选	在上升或下降沿初始化串行接口， 随后等待接收指令。STB 为低后的第一个字节作为指令，当处理指令时，当前其它处理被终止。当STB 为高时， CLK 被忽略

AIP1638 数码管接法

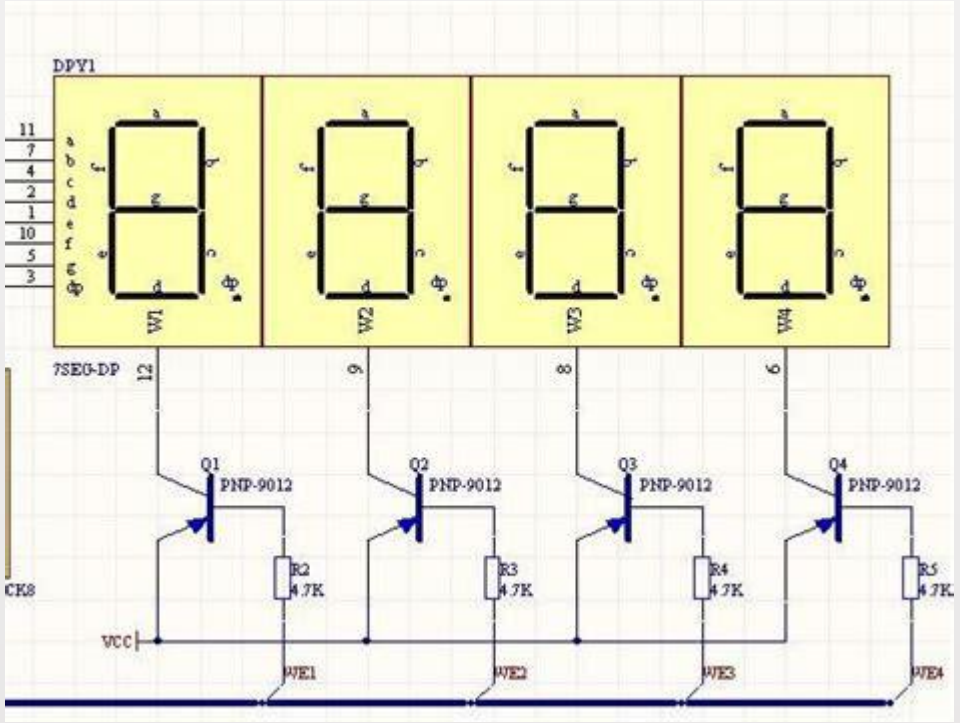






数码管显示控制原理

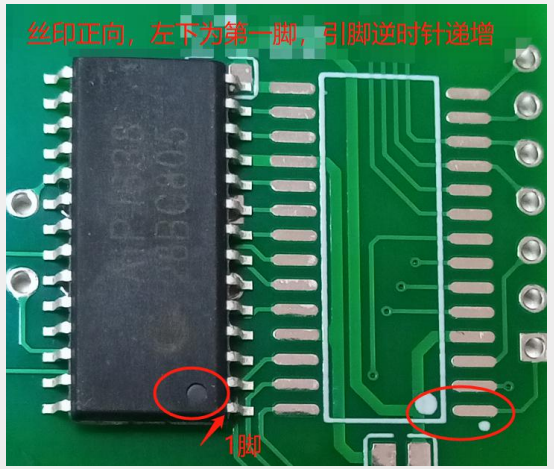
- 根据LED点阵学习可了解点阵显示需要不断刷新LED才能显示不同内容，数码管也是一样的a-g一般共用，公共端为片选端，一般由三极管控制。参考原理如下图，Q1-Q4分时选通，当Q1选通时，a-g上为低(0)的位置被点亮
- 因为需要不断刷新，所以很占用处理资源
- 那么是否可以做个专门的芯片来刷新呢?
- 答案：AiP1620，AIP1638等。
- 选择AIP1638支持10段8位，



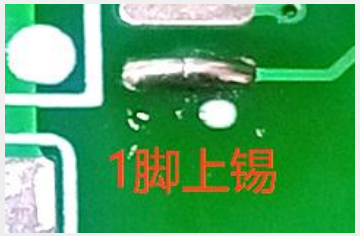
焊接方法

- IC的焊接步骤
  - PCB上IC的1脚焊盘上锡
  - IC1脚对齐1脚焊盘，调整好其他引脚，焊接1脚并检测其他脚是否对齐，若不齐重复该步骤
  - 焊接最后1脚，焊接过程中暂时不处理短路情况
  - A. 其他管脚适当堆锡，使用拖焊完成; B. IC脚间距大时可逐个焊接

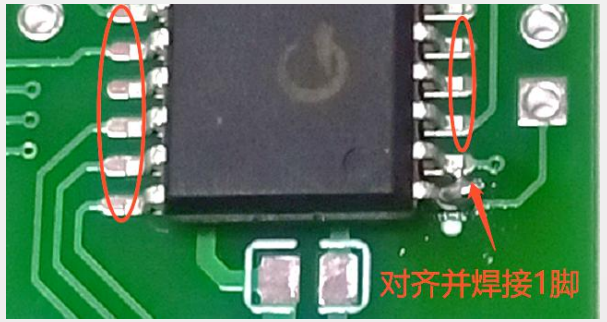
焊接方法-对齐固定



确认方向



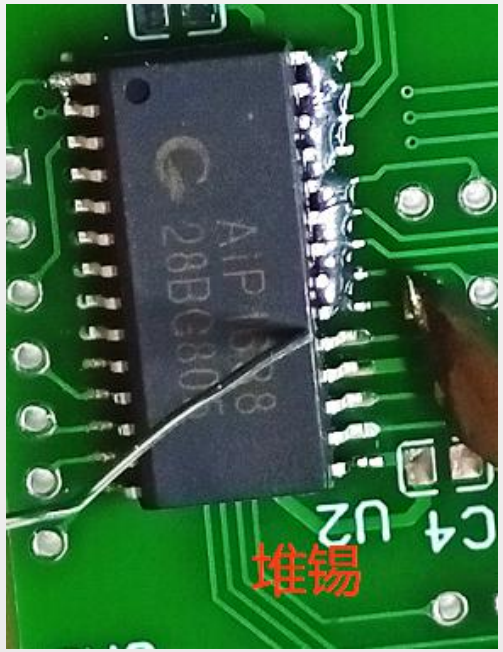
1脚上锡



1脚焊接-固定

各脚焊接

焊接方法-拖焊



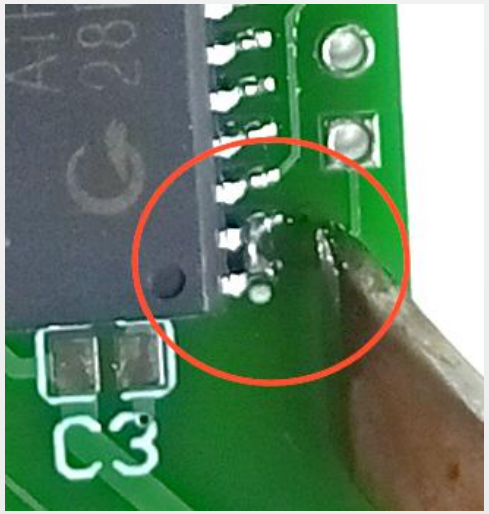
堆锡

拖焊1

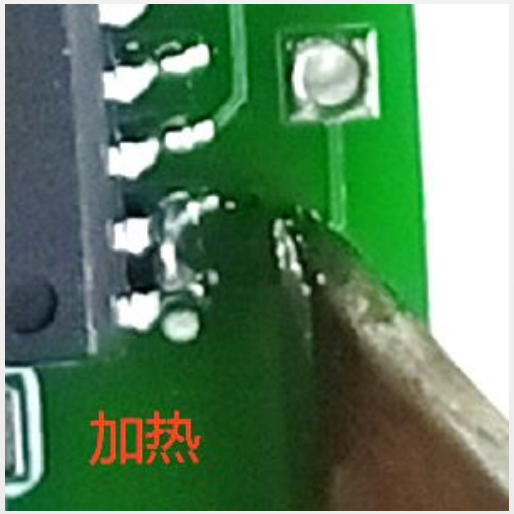
拖焊2

拖焊中要加适量的焊锡，同时保持焊锡的流动性很重要，此时就需要用到助焊剂：松香或焊锡膏

焊接方法-短路处理



短路位置



加热，必要时加助焊剂

利用流动性烙铁头带走多余焊锡

保持焊锡的流动性很重要，需要用到助焊剂：松香或焊锡膏

SPI总线

- SPI是串行外设接口（Serial Peripheral Interface）的缩写。SPI，是一种高速的，全双工，同步的通信总线
  - SDI – **S**erial**D**ata **In,**串行数据输入；
  - SDO – **S**erial**D**ata**O**ut,串行数据输出；
  - SCLK – Serial Clock,时钟信号，由主设备产生；
  - CS – Chip Select,从设备使能信号，由主设备控制，不同设备可接不同CS信号共用其他信号

根据数据手册编写AIP1638驱动

AIP1638有三个引脚用以初始化

26	DIO	数据输入/输出	在时钟上升沿输入/输出串行数据，从低位开始；
27	CLK	时钟输入	上升沿输入/输出串行数据
28	STB	片选	在上升或下降沿初始化串行接口， 随后等待接收指令。STB 为低后的第一个字节作为指令，当处理指令时，当前其它处理被终止。当STB 为高时， CLK 被忽略

其中有两个引脚已经在SPI中介绍过，因此在初始化时只需要额外设定cs引脚即可

```
class AIP1638:
    def __init__(self, spi, cs):
        self.spi = spi
        self.cs = cs

    def _data(self, data):
        self.cs(0)
        pyb.delay(1)
        self.spi.write_data(data)
        pyb.delay(1)
        self.cs(1)

    def disp_on(self):
        self._data(b"\x8d")

    def disp_off(self):
        self._data(b"\x80")

def disp_test():
    spi = USR_SPI(scl=Pin('X1',Pin.OUT_PP), sda=Pin('X3',Pin.OUT_PP))
    display = AIP1638(spi,cs=Pin('X2',Pin.OUT_PP))
    print("AIP1638 control interface init done")

    display.disp_on()
    display._data(b"\x40")
    display._data(b"\xC0\xf0\x00\x0f\x00\x0f\x00\xf0\x00\xf0\x00\xf0\x00\xf0\x00\xf0\x00")

    while True:
        print("AIP1638 disp")
        pyb.delay(1000)
        display.disp_on()
        pyb.delay(1000)
        display.disp_off()

disp_test()
```

对AIP1638的设定

根据数据手册控制AIP1638

控制8段LED数码管代码如下：

```
display.disp_on()
display._data(b"\x40")#设为开始输入数据
display._data(b"\xC0\xf0\x00\x0f\x00\x0f\x00\xf0\x00\xf0\x00\xf0\x00\xf0\x00\xf0\x00")
```

第二行：输入为01000000，根据手册内容，设置为写数据到显示寄存器

4.3.1、数据命令设置

该指令用来设置数据写和读，B1和B0位不允许设置01或11。

B7	B6	B5	B4	B3	B2	B1	B0	功能	说明
0	1	无关项，填0				0	0	数据读写模式设置	写数据到显示寄存器
0	1					1	0		读键扫数据
0	1				0			地址增加模式设置	自动地址增加
0	1				1				固定地址
0	1			0				测试模式设置（内部使用）	普通模式
0	1			1					测试模式

根据数据手册控制AIP1638

第三行：开头输入为11000000

```
display.disp_on()
display._data(b"\x40")#设为开始输入数据
display._data(b"\xC0\xf0\x00\x0f\x00\x0f\x00\xf0\x00\xf0\x00\xf0\x00\xf0\x00\xf0\x00")
```

设置地址开始位置为00H，之后将后面的依次输入寄存器内，其地址依次增加

MSB LSB

B7	B6	B5	B4	B3	B2	B1	B0	显示地址
1	1			0	0	0	0	00H
1	1			0	0	0	1	01H

根据数据手册控制AIP1638

本电路焊接为共阳极，根据手册示例，若想在第一位处显示0，则需要将00H，02H，04H，06H，08H，0AH地址内写入01H，其余地址单元全部写0

2、驱动共阳数码管：

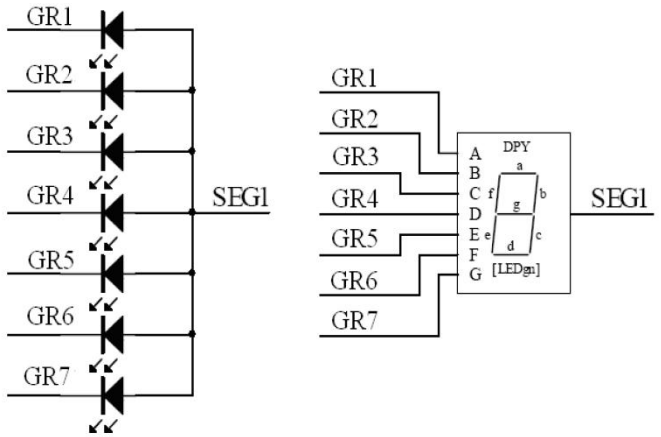


图9、

图9给出共阳数码管的连接示意图，如果让该数码管显示“0”，那么在GR1，GR2，GR3，GR4，GR5，GR6为低电平时SEG1为高电平，在GR7为低电平时SEG1为低电平。要向地址单元00H，02H，04H，06H，08H，0AH里面分别写数据01H，其余的地址单元全部写数据00H。

SEG8	SEG7	SEG6	SEG5	SEG4	SEG3	SEG2	SEG1	
0	0	0	0	0	0	0	1	00H
0	0	0	0	0	0	0	1	02H
0	0	0	0	0	0	0	1	04H
0	0	0	0	0	0	0	1	06H
0	0	0	0	0	0	0	1	08H
0	0	0	0	0	0	0	1	0AH
0	0	0	0	0	0	0	0	0CH
B7	B6	B5	B4	B3	B2	B1	B0	

注：SEGn为P管开漏输出，GRn为N管开漏输出，在使用时候，SEGn只能接LED的阳极，GRn只能接LED的阴极，不可反接。

根据数据手册控制AIP1638

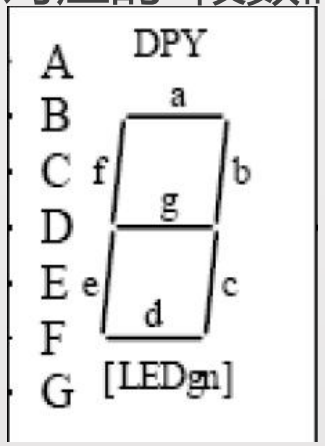
根据手册示例，本次所给示例写入地址情况如右图所示

显示结果如下图所示：

第一行最后一个E所对应的数字为右图红框所圈，可以简单的理解为：

10011111

对应的8段数码管如图：



除b、c两段之外全亮

```
\xC0 #设置地址为0
\xf0\x00#
1111 0000   00H
0000 0000   01H
\x0f\x00#
0000 1111   00H
0000 0000   03H
\x0f\x00#
0000 1111   04H
0000 0000   05H
\xf0\x00#
1111 0000   06H
0000 0000   07H
\xf0\x00#
1111 0000   08H
0000 0000   09H
\xf0\x00#
1111 0000   0AH
0000 0000   0BH
\xf0\x00#
1111 0000   0CH
0000 0000   0DH
\xf0\x00#
1111 0000   0EH
0000 0000   0FH
```

控制数码管显示0-9

根据所给示例，将0-9对应的数码管段选写入数组

```
for i in range(8):
    for j in range(8):
        temp[i][j]=temp[i][j]<<i
```

第一个循环从0-7中依次选中数字，第二个循环将选中数字的每个元素都左移相应的位数

```
for i in range(8):
    for j in  range(8):
        result[i]=result[i]+temp[j][i]
```

之后将每个数字的相应元素依次相加，这样就能保证写入寄存器内的数据按照二进制展开后从上到下对应的位数符合我们的需求

```
Number=[#0
    [1,1,1,1,1,1,0,0],
    #1
    [0,1,1,0,0,0,0,0],
    #2
    [01,01,00,01,01,00,01,00],
    #3
    [01,01,01,01,00,00,01,00],
    #4
    [00,01,01,00,00,01,01,00],
    #5
    [01,00,01,01,00,01,01,00],
    #6
    [01,00,01,01,01,01,01,00],
    #7
    [01,01,01,00,00,00,00,00],
    #8
    [01,01,01,01,01,01,01,00],
    #9
    [01,01,01,01,00,01,01,00]
]
```

认识RTC

- RTC（Real-Time Clock）即实时时钟
- PC主板上的晶振及相关电路组成的时钟电路的生成脉冲，RTC经过8254电路的变频产生一个频率较低一点的OS(系统)时钟TSC，系统时钟每一个cpu周期加一，每次系统时钟在系统初起时通过RTC初始化
- 提供稳定的时钟信号给后续电路用。主要功能有:时钟，日历，闹钟，周期性中断输出，32KHz时钟输出。

认识RTC

- RTC类中有四个方法：
- RTC.datetime([datetimetuple])：设置RTC的日期与时间，其中datetimetuple的格式为：(year, month, day, weekday, hours, minutes, seconds, subseconds)
- Reout, callback=None)：唤醒函数，第一个参数为时间长度，单位为微TC.wakeup(tim秒，第二个参数为函数入口
- RTC.info()：获取关于启动时间以及重置源的信息
- RTC.calibration(cal)：校准RTC

显示时间

- 通过RTC类的datetime（）方法对RTC进行时间的设定
- 实例化一个RTC对象，之后对齐进行设计，方法变量为一个数组，按照年、月、日、星期、小时、分、秒、亚秒（1/255秒）进行赋值

```
rtc = pyb.RTC()
rtc.datetime((2014, 5, 1, 4, 13, 0, 0, 0))
print(rtc.datetime())
```

- 调用datetime（）方法查看RTC时间，该方法会返回一个长度为8的数组

```
>>> rtc.datetime()
(2014, 5, 1, 4, 13, 3, 28, 142)
>>> rtc.datetime()
(2014, 5, 1, 4, 13, 3, 29, 82)
>>> rtc.datetime()
(2014, 5, 1, 4, 13, 3, 30, 192)
>>> rtc.datetime()
(2014, 5, 1, 4, 13, 3, 30, 83)
```

电子表设计

- 在调用rtc.datatime()函数后，会返回一个长度为8的数组，我们将其赋值给变量a，之后将各个元素依次赋值给相应的变量

```
a=rtc.datetime()
year=a[0]
month=a[1]
day=a[2]
weekday=a[3]
hours=a[4]
minutes=a[5]
seconds=a[6]
subseconds=a[7]
```

- 然后调用之前写好的驱动将这些变量分为两部分，年月日为一部分，时分秒星期为第二部分
- 至此，我们已经知道了如何在8段LED上显示数字以及如何获取内部时间，之后按照之前所讲即可设计出电子表，具体思路不在赘述，代码在Clock.py文件内。

设计音乐闹钟

根据之前学过的蜂鸣器播放音乐部分，将乐谱写入文件，按照不同的频率及间隔控制蜂鸣器的响声，以此来播放乐曲

在第一次读取时间的时候将其存入变量中，之后在更新数码管的时候对是否到时间进行判断，到达设定时间时即开始播放音乐，同时不断对LED屏进行更新

设计音乐闹钟

具体代码如下：

```
a=rtc.datetime()
clk_hours=a[4]
clk_minutes=a[5]
clk_seconds=a[6]
clk=5
```

获取开始时间并定时为5秒

```
temp=(hours-clk_hours)*3600+(minutes-clk_minutes)*60+(seconds-clk_seconds)*1
if temp==clk:
    x1 = Pin('X4',Pin.OUT_PP)
    tm3=Timer(2, freq=MyScore[i][0])
    led3=tm3.channel(4, Timer.PWM, pin=x1,pulse_width_percent=50)
    for i in range(16):
        b=rtc.datetime()
        b_hours=b[4]
        b_minutes=b[5]
        b_seconds=b[6]
        print(b_hours,b_minutes,b_seconds)
        tm3.freq(MyScore[i][0])
        show(b_hours,b_minutes,b_seconds)
        display.disp_on()
        display._data(b"\x40")#设为开始输入数据
        clock_s=(b"\xC0")
        for  i in  range(8):
            clock_s+=pack('<H', result[i])
        display._data(clock_s)
        display.disp_on()
        pyb.delay(int(MyScore[i][1]))
```

判断是否到设定时间并播放乐曲

实现多位数字显示

```
temp= Number
result=[0,0,0,0,0,0,0,0]
for i in range(8):
    for j in range(8):
        temp[i][j]=temp[i][j]<<i
for i in range(8):
    for j in  range(8):
        result[i]=result[i]+temp[j][i]

def disp_test():
    spi = USR_SPI(scl=Pin('X1',Pin.OUT_PP), sda=Pin('X3',Pin.OUT_PP))
    display = AIP1638(spi,cs=Pin('X2',Pin.OUT_PP))
    print("AIP1638 control interface init done")

    display.disp_on()
    display._data(b"\x40")#设为开始输入数据
    s=(b"\xC0")
    for  i in  range(8):
        s+=pack('<H', result[i])
    print(s)
    display._data(s)

    while True:
        print("AIP1638 disp")
        pyb.delay(1000)
        display.disp_on()
        pyb.delay(1000)
        display.disp_off()

disp_test()
```