

智能系统与控制

树莓派：GPIO-液晶控制

于泓

鲁东大学

信息与电气工程学院

2021.11.8



液晶模块：LCD1602+I2C串口芯片PCF8574



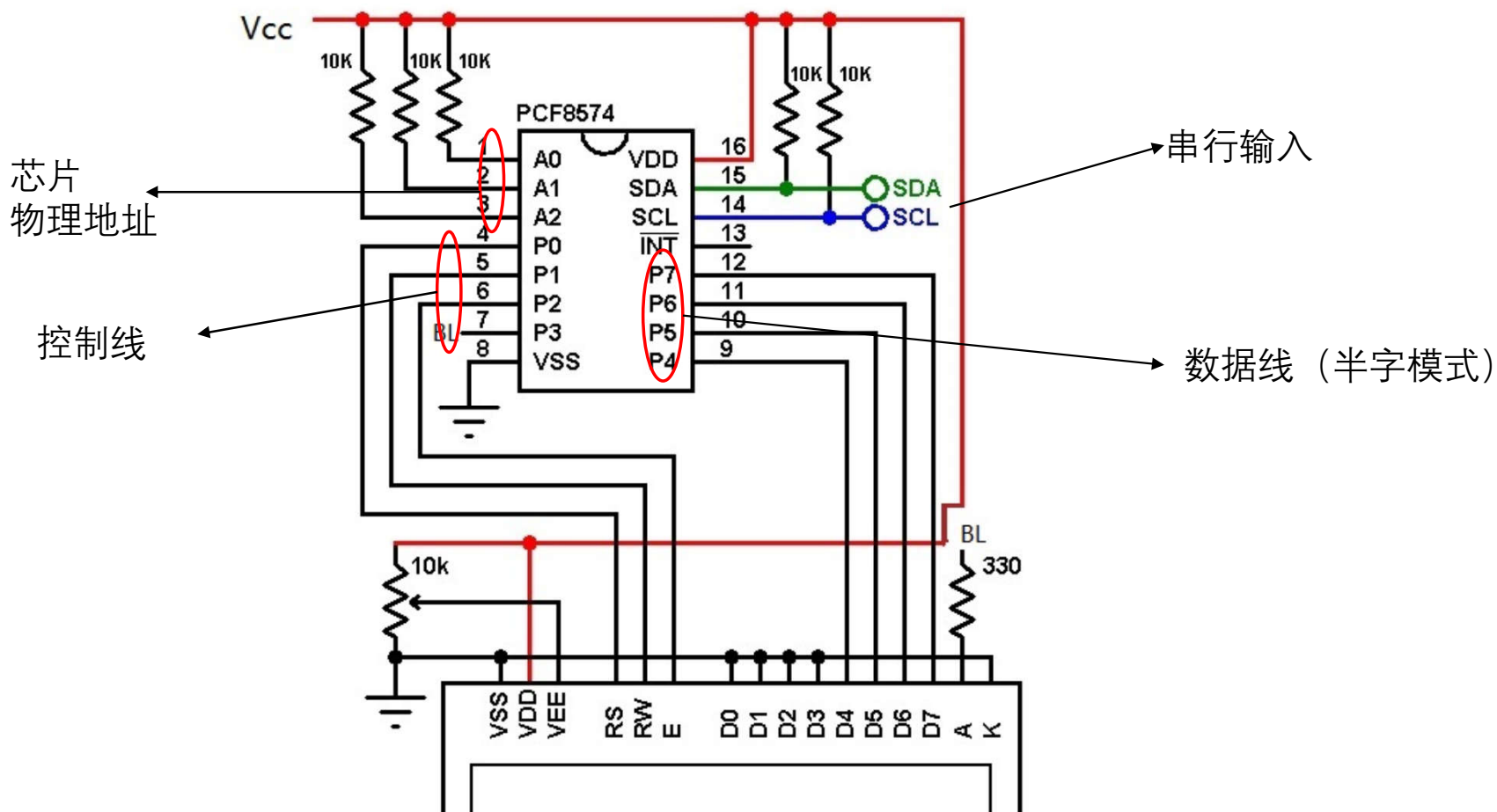
LCD1602 液晶显示器是广泛使用的一种字符型液晶显示模块。它是由字符型液晶显示屏（LCD）、控制驱动主电路 HD44780 及其扩展驱动电路 HD44100，以及少量电阻、电容元件和结构件等装配在 PCB 板上而组成。

驱动一个 LCD1602 液晶模块需要用到 8 条数据线（D0-D7），一条读/写选择线（R/W），一条数据/命令选择线（RS）以及一条使能信号线（E）共计 **11** 条 IO 线。



半字模式使用 4 条数据线，那也需要至少需要 **7** 个 IO 口来进行 I2C 液晶模块的驱动

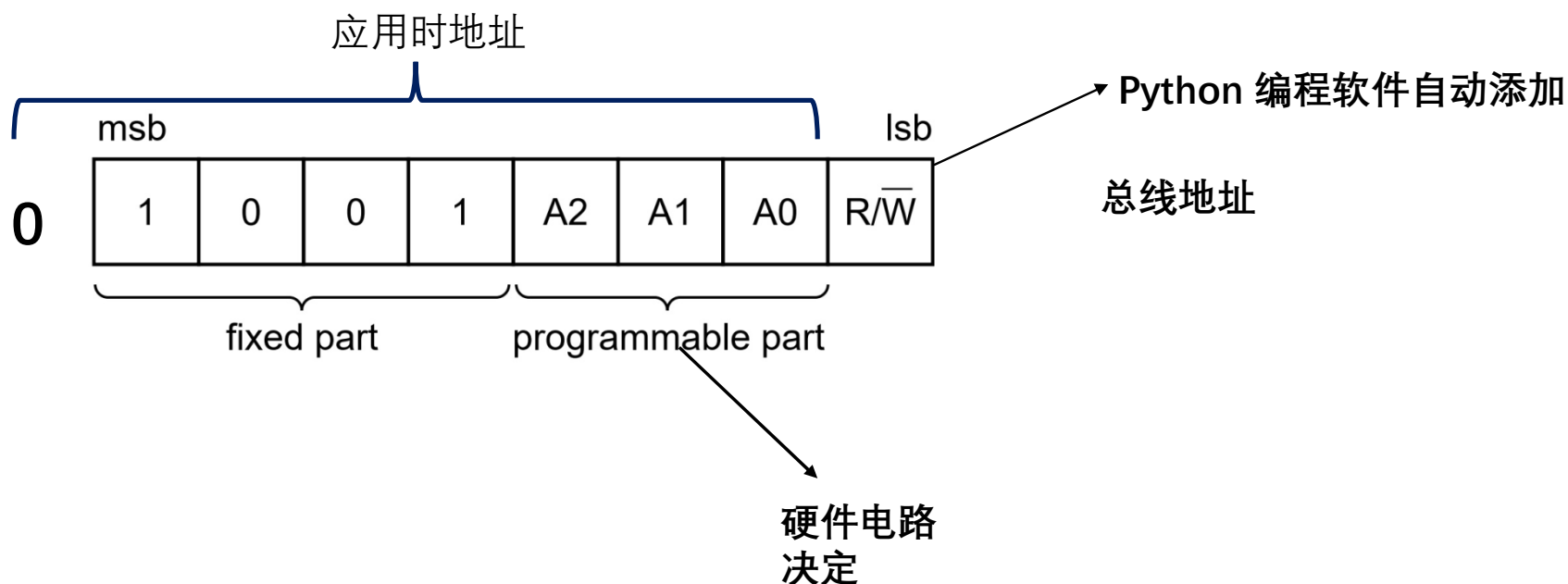
引入I2C串并转换模块**PCF8574**
只需要2根IO就可以实现液晶的驱动



I2C总线

I2C(Inter - Integrated Circuit) 总线是 80 年代 PHILIPS 公司开发的两线式**串行总线**。
在硬件上，它只需要一根数据线和一根时钟线两根线，总线接口已经集成在芯片内部，不需要特殊的接口电路，而且片上接口电路的滤波器可以滤去总线数据上的毛刺。因此 I2C 总线简化了硬件电路 PCB 布线，降低了系统成本，提高了系统可靠性。

可以挂接多个设备



I2C配置

启动I2C

`sudo raspi-config`

I2C 总线协议
设计精巧细致，
如果利用 IO
口进行直接模
拟相对比较困
难，树莓派系
统
内置了 I2C 接
口，使得我们
可以很方便的
使用它们。

```
Raspberry Pi Software Configuration Tool (raspi-config)

1 Change User Password  Change password for the 'pi' user
2 Network Options       Configure network settings
3 Boot Options          Configure options for start-up
4 Localisation Options  Set up language and regional settings to match your location
5 Interfacing Options   Configure connections to peripherals
6 Overclock             Configure overclocking for your Pi
7 Advanced Options      Configure advanced settings
8 Update                Update this tool to the latest version
9 About raspi-config    Information about this configuration tool

<Select>                                <Finish>
```

```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SSH         Enable/Disable remote command line access to your Pi using SSH
P3 VNC         Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI         Enable/Disable automatic loading of SPI kernel module
P5 I2C         Enable/Disable automatic loading of I2C kernel module
P6 Serial      Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire      Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins

<Select>                                <Back>
```

```
Would you like the ARM I2C interface to be enabled?

<Yes>                                <No>
```

安装驱动程序:

I2c的Python驱动接口

sudo apt-get install python-smbus

I2c的命令行工具包

sudo apt-get install i2c-tools

显示所有的I2C设置

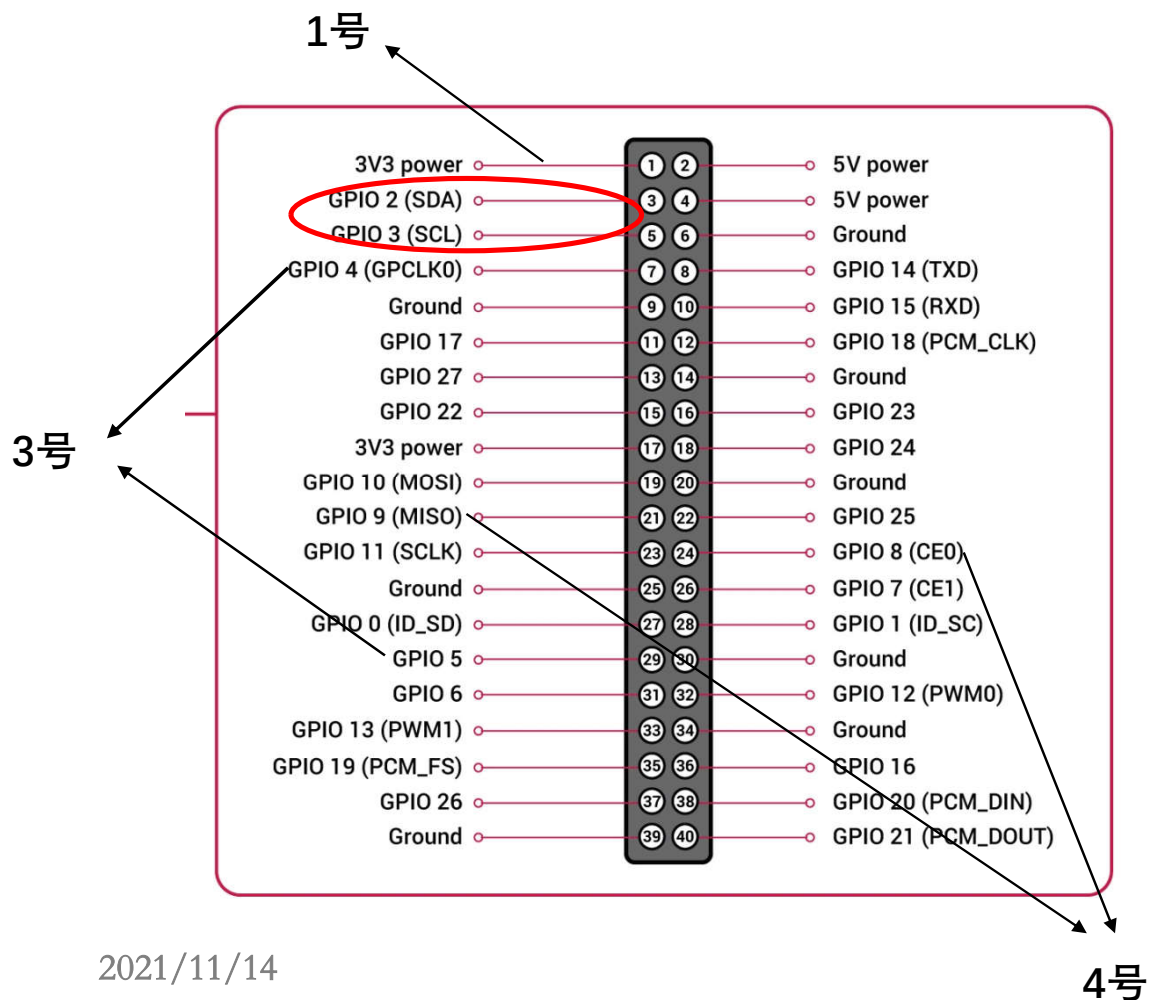
ls /dev/*i2c*

检测I2c 设备上是否有i2c器件

sudo i2cdetect -y 1

```
pi@raspberrypi:~/EXP-Raspberry/EXP_LCD $ ls /dev/*i2c*
/dev/i2c-1 /dev/i2c-4 /dev/i2c-7
pi@raspberrypi:~/EXP-Raspberry/EXP_LCD $ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- 27 -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
70:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~/EXP-Raspberry/EXP_LCD $ sudo i2cdetect -y 4
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
70:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~/EXP-Raspberry/EXP_LCD $ sudo i2cdetect -y 7
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
```


I2C总线的扩展



硬件端口扩展说明

cat /boot/overlays/README

```

Name: i2c1
Info: Change i2c1 pin usage. Not all pin combinations are usable on all
      platforms - platforms other than Compute Modules can only use this
      to disable transaction combining.
Load: dtoverlay=i2c1,<param>=<val>
Params: pins_2_3      Use pins 2 and 3 (default)
        pins_44_45    Use pins 44 and 45
        combine        Allow transactions to be combined (default
                        "yes")

Name: i2c1-bcm2708
Info: Deprecated, legacy version of i2c1, from which it inherits its
      parameters, just adding the explicit individual pin specifiers.
Load: <Deprecated>
Params: sda1_pin      GPIO pin for SDA1 (2 or 44 - default 2)
        scl1_pin      GPIO pin for SCL1 (3 or 45 - default 3)
        pin_func       Alternative pin function (4 (alt0), 6 (alt2) -
                        default 4)

Name: i2c3
Info: Enable the i2c3 bus
Load: dtoverlay=i2c3,<param>
Params: pins_2_3      Use GPIOs 2 and 3
        pins_4_5      Use GPIOs 4 and 5 (default)
        baudrate       Set the baudrate for the interface (default
                        "100000")

Name: i2c4
Info: Enable the i2c4 bus
Load: dtoverlay=i2c4,<param>
Params: pins_6_7      Use GPIOs 6 and 7
        pins_8_9      Use GPIOs 8 and 9 (default)
        baudrate       Set the baudrate for the interface (default

```

扩展i2c启动方法

sudo nano /boot/config.txt

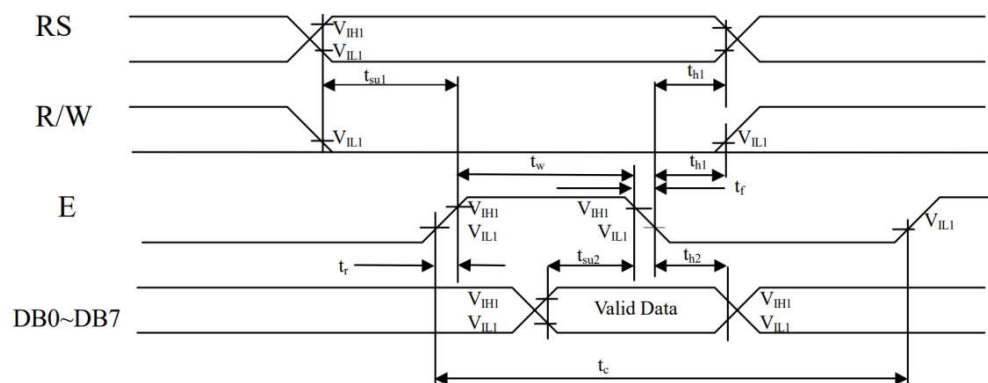
```
[all]
#dtoverlay=vc4-fkms-v3d
dtoverlay=w1-gpio
dtoverlay=i2c3
dtoverlay=i2c4
dtoverlay=i2c-gpio,bus=7

dtoverlay=gpio-ir,gpio_pin=17
```

启动
新的i2c总线

通过修改可以设置任意GPIO作为I2c的总线

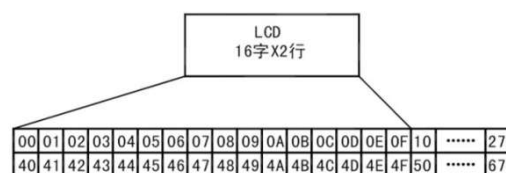
LCD1602 总线协议



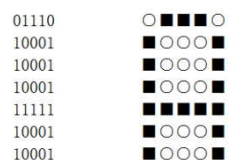
命令：工作方式
数据：显示内容

当要发送命令/数据时，首先对三条控制线进行设置：
设置 RS 的状态（发送命令时RS=0；发送数据时 RS=1），
将 RW 置 0（表示进行写操作），
将 EN 置 1（表示允许进行数据传输）；
 然后通过**数据线 D7-D0 进行数据传输**；
 延时一段时间后**将 EN 置 0**表示数据传输结束

LCD1602 工作原理



(a) DDRAM(Display Data RAM)
数据显示RAM



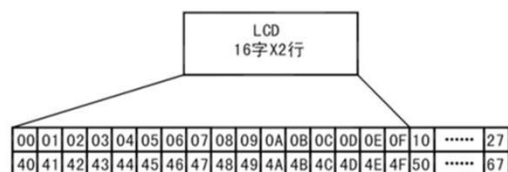
(c) 显示表例

Address	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000			00P`P													
xxxx0001	(2)		!1AQa													
xxxx0010	(3)		"2BRbr													
xxxx0011	(4)		#3CScs													
xxxx0100	(5)		\$4DTdt													
xxxx0101	(6)		%5EUeu													
xxxx0110	(7)		&6FVfv													
xxxx0111	(8)		'7GWgw													
xxxx1000	(1)		(8HXhx													
xxxx1001	(2))9IYiy													
xxxx1010	(3)		*:JZjz													
xxxx1011	(4)		+;K[k(
xxxx1100	(5)		,<L[l													
xxxx1101	(6)		-=M]m}													
xxxx1110	(7)		.>N^n~													
xxxx1111	(8)		/?O_o													

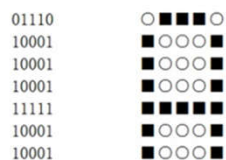
(b) CGROM(Character Generator ROM)
CGRAM(Character Generator RAM)
码表

在模块内部有一个 80 字节的 DDRAM (图a)
里面存放着需要显示数据的编码。
DDRAM 第一行地址从 00h 到 27H,
第二行地址从 40H-67H。

由于 LCD1602 共两行, 每行显示 16 个字,
所以能够用来显示的 DDRAM 地址为
00H-0FH (第一行)
40H-4FH (第二行)。
其他的地址也可以存放数据,
不过存放的数据无法在屏幕上显示出来



(a) DDRAM(Display Data RAM)
数据显示RAM



(c) 显示表例

Address	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000																
xxxx0001 (2)																
xxxx0010 (3)																
xxxx0011 (4)																
xxxx0100 (5)																
xxxx0101 (6)																
xxxx0110 (7)																
xxxx0111 (8)																
xxxx1000 (1)																
xxxx1001 (2)																
xxxx1010 (3)																
xxxx1011 (4)																
xxxx1100 (5)																
xxxx1101 (6)																
xxxx1110 (7)																
xxxx1111 (8)																

(b) CGROM(Character Generator ROM)
CGRAM(Character Generator RAM)
码表

当需要进行数据显示的时候，就向 DDRAM 的相应地址中写入需要显示字符的编码，就是显示字符的 ACSII 码。LCD1602 模块会根据写入的码字到 CGROM 构成的码表（图 b）中查询该字符对应的点阵数据。然后驱动 LED 屏幕进行点阵显示。

例如我们想在第 2 行第 3 个字符处显示一个字母 ‘A’，那就需要向 DDRAM 的 42H 地址内写入 ‘A’ 的 ACSII 码 41H，然后模块就会自动的从 CGROM 中查找到 ‘A’ 的点阵（图c）并在屏幕上进行显示。

取值、编码以及扫描的过程都是由模块自动完成的。

用户只需要向 DDRAM 中写入想要显示的数据即可

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

(3) 输入模式的设置。该命令定义了向 **DDRAM** 写入数据后光标的运动方向，以及是否将运动显示出来。I/D 为 1 表示光标左移，为 0 表示光标右移。S 为 1 表示显示移动，为 0 表示不显示移动。

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

(4) 显示控制。该命令控制屏幕是否显示内容，以及控制光标的显示方式。

D 为 1 表示显示、为 0 表示不显示；

C 为 1 表示显示光标，为 0 表示不显示；

B 为 1 表示光标闪烁，为 0 表示不闪烁。

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

(5) 光标字符移位控制。该名命令可以控制屏幕显示的内容或者光标，左移或者右移一位

S/C 为 1 表示控制文字移动，为 0 表示光标移动。

R/L 为 1 表示向右移动，为 0 表示向左移动。

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

(6) 功能设置。该命令用来对模块的整体功能进行设置。

DL 为 1 表示 8 总线模式，为 0 表示 4 总线模式（本例中就是使用 4 总线模式）。

N 为 1 表示 2 行显示，为 0 表示 1 行显示。

F 表示显示点阵数据的规格，为 1 表示 5×10 点阵，为 0 表示 5×7 点阵

通常在初始化时进行设置

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

(7) 设置字符码表地址。根据前面的讲述可知，在屏幕显示时，显示的点阵数据时从码表存储器 **CGROM** 中读取的。当我们想要显示一些码表中没有的特殊字符时，我们可以在 **CGRAM** 中人工填入点阵数据。利用该命令字就可以设置填入点阵数据在 **CGRAM** 中的存储地址，然后发送数据进行填充。

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

(8) 设置显存地址。

当需要液晶屏幕显示内容时，需要向 DDRAM 中写入数据，在写入数据之前需要调用命令设置数据写入的地址。如果只调用该命令后面不跟写入数据的操作时，该命令相当于设置光标的位置。

代码实现

```
# 命令字
LCD_CLEARDISPLAY = 0x01    #清屏
LCD_RETURNHOME = 0x02     #光标复位

LCD_SETGRAMADDR = 0x40    #字符发生器地址
LCD_SETDRAMADDR = 0x80    #显示数据存储地址

# 输入方式控制
LCD_ENTRYMODESET = 0x04    #输入方式标志位
LCD_ENTRYRIGHT = 0x00      #输入新数据光标右移动
LCD_ENTRYLEFT = 0x02      #输入新数据光标左移动
LCD_ENTRYSHIFTINCREMENT = 0x01 #显示移动
LCD_ENTRYSHIFTDECREMENT = 0x00 #不显示移动

# 显示开关控制
LCD_DISPLAYCONTROL = 0x08 #显示开关控制标志位
LCD_DISPLAYON = 0x04      #整体显示开
LCD_DISPLAYOFF = 0x00     #整体显示关
LCD_CURSORON = 0x02       #光标开
LCD_CURSOROFF = 0x00      #光标关
LCD_BLINKON = 0x01        #闪烁开
LCD_BLINKOFF = 0x00       #闪烁关
```

序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

通过**或 (|)** 操作，实现各种功能的组合

```

# 光标或字符移位控制
LCD_CURSORSHIFT = 0x10    #光标、字符移位标志位
LCD_DISPLAYMOVE  = 0x08    # 移动显示文字
LCD_CURSORMOVE   = 0x00    # 移动光标
LCD_MOVERIGHT    = 0x04    # 右移
LCD_MOVELEFT     = 0x00    # 左移

#功能设置
LCD_FUNCTIONSET  = 0x20    #功能设置标志位
LCD_8BITMODE     = 0x10    #8总线模式
LCD_4BITMODE     = 0x00    #4总线模式
LCD_2LINE        = 0x08    # 2行显示
LCD_1LINE        = 0x00    # 1行显示
LCD_5x10DOTS     = 0x04    # 5x10 字符点阵
LCD_5x8DOTS      = 0x00    # 5x8字符点阵
  
```

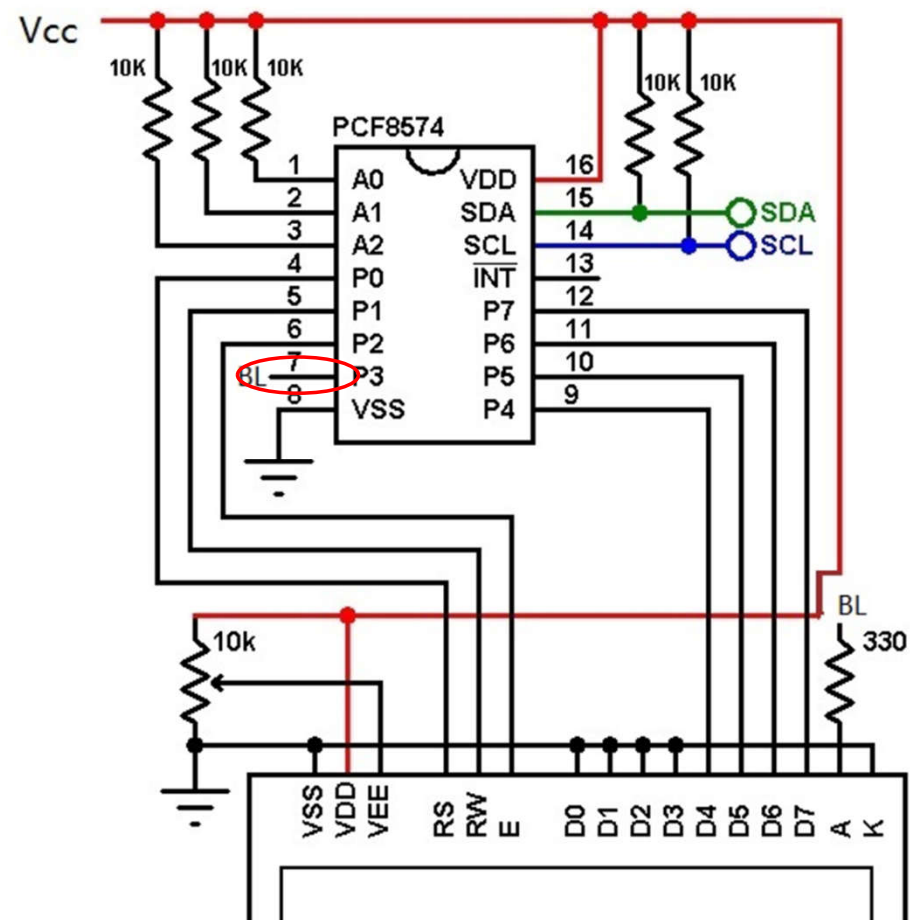
序号	功能	D7	D6	D5	D4	D3	D2	D1	D0
1	清屏	0	0	0	0	0	0	0	1
2	光标复位	0	0	0	0	0	0	1	-
3	输入模式设置	0	0	0	0	0	1	I/D	S
4	显示控制	0	0	0	0	1	D	C	B
5	光标字符移位控制	0	0	0	1	S/C	R/L	-	-
6	功能设置	0	0	1	DL	N	F	-	-
7	设置字符地址	0	1	A5	A4	A3	A2	A1	A0
8	设置显存地址	1	A6	A5	A4	A3	A2	A1	A0

```

class LCD_1602(object):
    # 初始化 Address: I2C 芯片物理地址 bus_id 总线编号 bl 是非开背光
    def __init__(self, Address=0x27, bus_id=1, bl=1):
        self.bus_id = bus_id
        self.Address = Address
        self.BUS = smbus.SMBus(self.bus_id)
        self.bl = bl

    # 设置背景光
    def set_BL(self, bl):
        self.bl = bl

    def write_word(self, data):
        # 根据背景光标志, 设置写数据方式
        temp = data
        if self.bl == 1:
            temp |= 0x08
        else:
            temp &= 0xF7
        self.BUS.write_byte(self.Address, temp)
  
```



发送命令字

```
def send_command(self,comm):
```

先送高4位

```
buf = comm & 0xF0
```

```
buf |= 0x04
```

RS = 0, RW = 0, EN = 1

```
self.write_word(buf)
```

```
time.sleep(0.002)
```

```
buf &= 0xFB
```

Make EN = 0

```
self.write_word(buf)
```

再送第4位

```
buf = (comm & 0x0F) << 4
```

```
buf |= 0x04
```

RS = 0, RW = 0, EN = 1

```
self.write_word(buf)
```

```
time.sleep(0.002)
```

```
buf &= 0xFB
```

Make EN = 0

```
self.write_word(buf)
```

发送数据

```
def send_data(self,data):
```

先送高4位

```
buf = data & 0xF0
```

```
buf |= 0x05
```

RS = 1, RW = 0, EN = 1

```
self.write_word(buf)
```

```
time.sleep(0.002)
```

```
buf &= 0xFB
```

Make EN = 0

```
self.write_word(buf)
```

再送低四位

```
buf = (data & 0x0F) << 4
```

```
buf |= 0x05
```

RS = 1, RW = 0, EN = 1

```
self.write_word(buf)
```

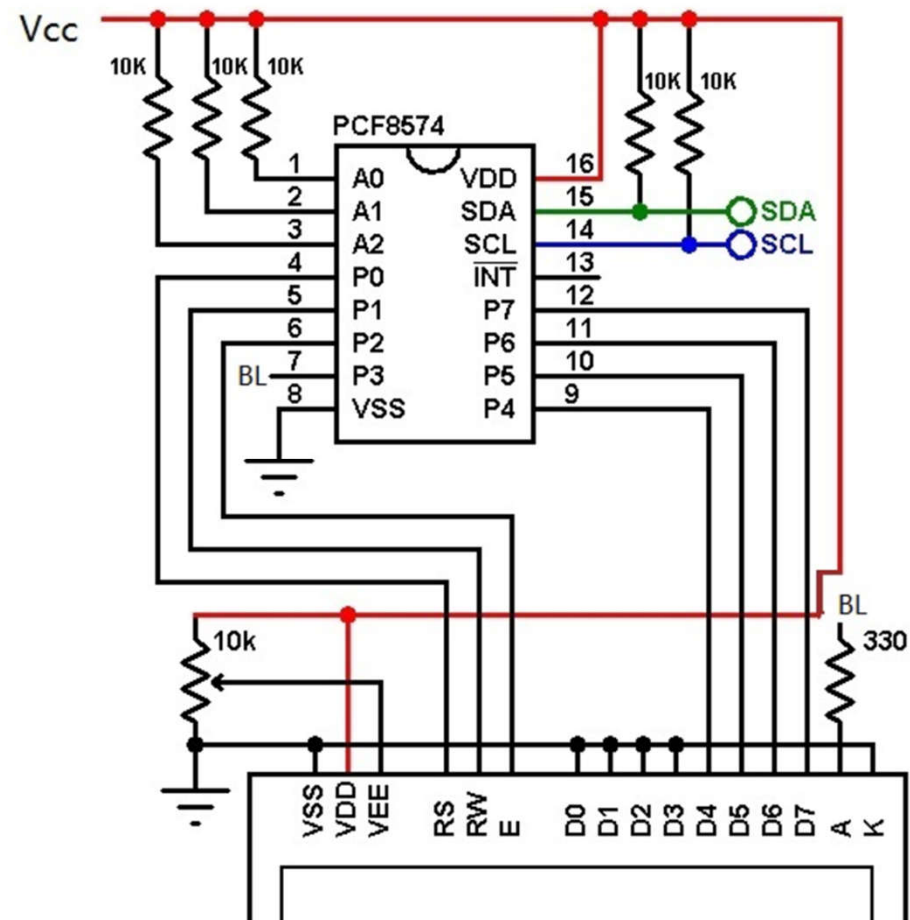
```
time.sleep(0.002)
```

```
buf &= 0xFB
```

Make EN = 0

```
self.write_word(buf)
```

2021/11/14



```

# LCD 初始化
def lcd_init(self):
    try:

        # 先初始化为 8总线模式 光标复位 清屏
        self.send_command(LCD_FUNCTIONSET|LCD_8BITMODE|LCD_CLEARDISPLAY|LCD_RETURNHOME)
        time.sleep(0.005)
        # 初始化为 8总线模式 光标复位
        self.send_command(LCD_FUNCTIONSET|LCD_8BITMODE|LCD_RETURNHOME)
        time.sleep(0.005)

        # 4总线模式 2行显示 5x8点阵
        self.send_command(LCD_FUNCTIONSET|LCD_4BITMODE|LCD_2LINE|LCD_5x8DOTS)
        time.sleep(0.005)

        # 设置数据进入方式
        self.lcd_inputset(bDirection=False, bShift=False)

        # 显示方式设置 整体显示开 、 光标关 、 闪烁关
        self.lcd_displaySwitch(bDisplay=True,bCursor=False,bBlink=False)

        # 清屏
        self.clear()

        # 背光打开
        self.openlight()

    except:
        return False
    else:
        return True

```

```

# 清屏
def clear(self):
    self.send_command(LCD_CLEARDISPLAY)
    time.sleep(0.005)

# 打开背光
def openlight(self):
    self.BUS.write_byte(self.Address,0x08)
    time.sleep(0.005)

# 光标复位 指向显示数据的起始位置
def lcd_cursorReturn(self):
    self.send_command(LCD_RETURNHOME)
    time.sleep(0.005)

```


```

# 设置显示方式 bDisplay 是否开显示 bCursor 是否显示光标 bBlink 是否闪烁
def lcd_displaySwitch(self,bDisplay=True,bCursor=False,bBlink=False):
    cmd = LCD_DISPLAYCONTROL| \
        (LCD_DISPLAYON if bDisplay else LCD_DISPLAYOFF)| \
        (LCD_CURSORON if bCursor else LCD_CURSOROFF)| \
        (LCD_BLINKON if bBlink else LCD_BLINKOFF)
    self.send_command(cmd)
    time.sleep(0.005)

# 输入移动方式设置
def lcd_inputset(self,bDirection,bShift):
    cmd = LCD_ENTRYMODESET| \
        (LCD_ENTRYLEFT if bDirection else LCD_ENTRYRIGHT)| \
        (LCD_ENTRYSHIFTINCREMENT if bShift else LCD_ENTRYSHIFTDECREMENT)
    self.send_command(cmd)
    time.sleep(0.005)

```


显示设置



```
# 设置光标的位置 x 列 y 行
def lcd_set_cursor(self,x,y):
    if x < 0:
        x = 0
    if x > 15:
        x = 15
    if y < 0:
        y = 0
    if y > 1:
        y = 1

    # Move cursor
    addr = 0x80 + 0x40 * y + x
    self.send_command(addr)

# 在指定位置输出字符串
def lcd_display_string(self,x, y, string):
    # 设置光标位置
    self.lcd_set_cursor(x,y)
    for ch in string:
        self.send_data(ord(ch))
```

```
# 光标或字符移位控制 bTarget 1 文字移动 0 光标移动 bDirection 1 右移 0 左移
def lcd_shit(self, bTarget,bDirection):
    cmd = LCD_CURSORSHIFT | \
        (LCD_DISPLAYMOVE if bTarget else LCD_CURSORMOVE) | \
        (LCD_MOVERIGHT if bDirection else LCD_MOVELEFT)
    self.send_command(cmd)
    time.sleep(0.005)
```

```
if __name__ == "__main__":  
    m_lcd = LCD_1602(Address=0x27,bus_id=1,bl=1)  
  
    try:  
        flag = m_lcd.lcd_init()  
        print(flag)  
  
        # 在指定位置显示字符串  
        m_lcd.lcd_display_string(0,0,'Welcome!!')  
        m_lcd.lcd_display_string(0,1,'WWW.LDU.EDU.CN')  
        time.sleep(5)  
  
        # 设置光标位置, 并改变该位置的光标显示方式  
        m_lcd.lcd_set_cursor(3,0)  
        m_lcd.lcd_displaySwitch(bDisplay=True,bCursor=True,bBlink=True)  
        time.sleep(5)  
  
        # 向右移动光标  
        m_lcd.lcd_shit(bTarget=0,bDirection=True)  
        time.sleep(5)  
  
        # 向右移动文字  
        m_lcd.lcd_shit(bTarget=1,bDirection=True)  
        time.sleep(5)  
  
        # 光标复位 改变显示形式  
        m_lcd.lcd_cursorReturn()  
        m_lcd.lcd_displaySwitch(bDisplay=True,bCursor=True,bBlink=False)  
        time.sleep(5)
```

```
space = '  
greetings = 'Thank you for watching the lesson of Raspberry LCD! ^_^'  
greetings = space + greetings
```

```
while True:  
    tmp = greetings  
    for i in range(0, len(greetings)):  
        m_lcd.lcd_display_string(0, 0, tmp)  
        tmp = tmp[1:]  
        time.sleep(0.8)  
        m_lcd.clear()
```

```
except KeyboardInterrupt:  
    pass
```



(a) 显示文字



(b) 设置光标



(c) 光标移位



(d) 文字移位



(e) 复位操作



(f) 显示长文

```

from test_LCD import LCD_1602
from test_ds1302 import DS1302
import RPi.GPIO as GPIO
from pin_dic import pin_dic
from datetime import datetime
import time

if __name__ == "__main__":

    # ds1302 初始化
    pin_clk = pin_dic['G4']
    pin_dat = pin_dic['G5']
    pin_rst = pin_dic['G6']

    GPIO.setmode(GPIO.BOARD)

    m_ds1302 = DS1302(pin_clk,pin_dat,pin_rst)
    # 写入当前时间
    x = datetime.now()
    m_ds1302.write_DateTime(x)

    # LCD 1602 初始化
    m_lcd = LCD_1602(Address=0x27,bus_id=1,bl=1)
    flag =m_lcd.lcd_init()
    print(flag)

```

```

try:
    while True:
        dt = m_ds1302.read_DateTime()

        if not dt:
            continue
        else:
            str_time1 = dt.strftime("%a %Y-%m-%d")
            str_time2 = dt.strftime("%H:%M:%S")

            m_lcd.lcd_display_string(0,0,str_time1)
            m_lcd.lcd_display_string(0,1,str_time2)

            time.sleep(1)
except KeyboardInterrupt:
    print('\n Ctrl + C QUIT')

finally:
    GPIO.cleanup()

```

