LCD1602

Overview

This lesson will use ‘‘raspberry pi’’ with 1602 LCD display

Experimental Materials:

Raspberry Pi *1

T-type expansion board *1

Breadboard *1

LCD1602 *1

Potentiometer *

Some DuPont lines

Poduct description:

● Function: The LCD1602A character LCD module is a dot matrix LCD module designed to display letters, numbers, symbols, etc. 4-bit and 8-bit data transmission.

● Applications: More and more widely used in low-power applications

Technical Parameters：

Display capacity is 16 x 2 characters;

Chip operating voltage is 4.5 ~ 5.5V;

Operating current is 2.0mA (5.0V);

The best operating voltage of the module is 5.0V;

Character size 2.95 x 4.35 (W * H) mm.

Interface pin description：

Pin no	symbol	description
1	VSS	Negative power supply
2	VDD	Positive power supply
3	VO	Contrast settings
4	RS	Instruction/data selection

5	**RW**	**Read/write data**
6	**E**	**enable**
7	**D0**	**Data 0**
8	**D1**	**Data 1**
9	**D2**	**Data 2**
10	**D3**	**Data 3**
11	**D4**	**Data 4**
12	**D5**	**Data 5**
13	**D6**	**Data 6**
14	**D7**	**Data 7**
15	**A**	**Positive backlight**
16	**K**	**Backlight negative**

interface specification ：

1, a group of two groups of power supply is a module of the power supply is a group of backlight power supply is generally used 5V power supply. This test backlight using 3.3V power supply can work.

2、V0 is a regulator of the contrast of the pin, series is not greater than 5K potentiometer to adjust. This experiment uses 1K ohm resistor to set the contrast. Its connection points high potential and low potential connection, the use of low potential connection, series 1K ohm resistance after GND.

3、RS is a lot of liquid crystal on the pin is the command / data selection pin the pin level is high when the data indicated that the data will be carried out.。

4、RW is also a lot of liquid crystal on the pin is the choice of reading and writing the pin level is high is the representation of the liquid crystal to read operation for the low time that the write operation.

5、E also a lot of liquid crystal module this pin is usually on the bus signal stability after the signal to a positive pulse notice to read the data in this pin for the high level when the bus is not allowed to change.

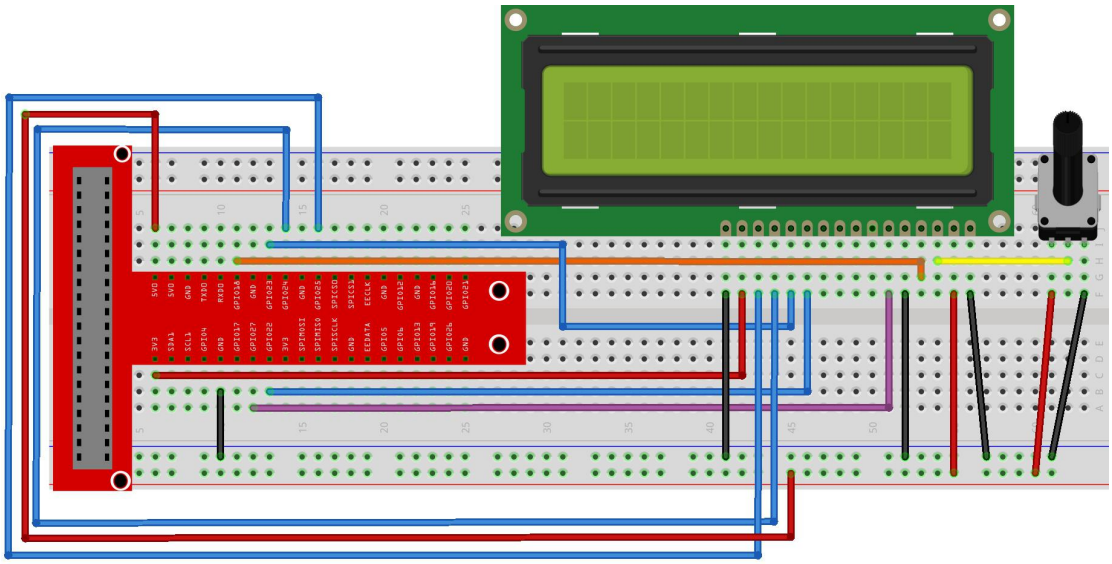
6、D7 - D0 8 two-way parallel bus, used to transmit commands and data.

7、 BLA is the back light source anode, BLK is the back light source cathode.

1602 the basic operation of the liquid crystal is divided into the following four kinds:

Read the state	INPUT	RS=L, R/W=H, E=H	OUTPUT	D0~D7= Status word
Written instructions	INPUT	RS=L, R/W=L, D0~D7=Order code E=High pulse	OUTPUT	NC
Read the data	INPUT	RS=H, R/W=H, E=H	OUTPUT	D0~D7= Data
Write the data	INPUT	RS=H, R/W=L, D0~D7=Data E= High pulse	OUTPUT	NC

Wiring diagram：



C code：

```
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <lcd.h>

const unsigned char Buf[] = "------world-------";
const unsigned char myBuf[] = "-----hello-------";

int main(void)
{
    int fd;
    int i;

    if(wiringPiSetup() == -1){
        exit(1);
    }

    fd = lcdInit(2, 16, 4,  1, 2,  3, 4, 5, 6, 0, 0, 0, 0); //see /usr/local/include/lcd.h
    printf("%d",  fd);
```

```
    if (fd == -1){
        printf("lcdInit 1 failed\n") ;
        return 1;
    }
    sleep(1);
    lcdClear(fd);
    lcdPosition(fd, 0, 0);
    lcdPuts(fd, "Welcome To--->");

    lcdPosition(fd, 0, 1);
    lcdPuts(fd, "    raspberry pi");

    sleep(1);
    lcdClear(fd);

    printf("\n");
    printf("\n");
    printf("========================================\n");
    printf("|           Control LCD1602            |\n");
    printf("========================================\n");
    printf("\n");
    printf("\n");

    while(1){
        lcdClear(fd);
        for(i=0; i<16; i++){
            lcdPosition(fd, i, 0);
            lcdPutchar(fd, *(myBuf+i));
            delay(100);
        }
        for(i=0;i<sizeof(Buf)-1;i++){
            lcdPosition(fd, i, 1);
            lcdPutchar(fd, *(Buf+i));
            delay(200);
        }
        sleep(0.5);
    }
    return 0;
}
```

Python code：

```
#!/usr/bin/env python

from time import sleep

class LCD:
    # commands
    LCD_CLEARDISPLAY        = 0x01
    LCD_RETURNHOME          = 0x02
    LCD_ENTRYMODESET        = 0x04
    LCD_DISPLAYCONTROL      = 0x08
    LCD_CURSORSHIFT         = 0x10
    LCD_FUNCTIONSET         = 0x20
    LCD_SETCGRAMADDR        = 0x40
    LCD_SETDDRAMADDR        = 0x80

    # flags for display entry mode
    LCD_ENTRYRIGHT      = 0x00
    LCD_ENTRYLEFT       = 0x02
    LCD_ENTRYSHIFTINCREMENT    = 0x01
    LCD_ENTRYSHIFTDECREMENT    = 0x00

    # flags for display on/off control
    LCD_DISPLAYON       = 0x04
    LCD_DISPLAYOFF      = 0x00
    LCD_CURSORON        = 0x02
    LCD_CURSOROFF       = 0x00
    LCD_BLINKON         = 0x01
    LCD_BLINKOFF        = 0x00

    # flags for display/cursor shift
    LCD_DISPLAYMOVE     = 0x08
    LCD_CURSORMOVE      = 0x00

    # flags for display/cursor shift
    LCD_DISPLAYMOVE     = 0x08
    LCD_CURSORMOVE      = 0x00
    LCD_MOVERIGHT       = 0x04
    LCD_MOVELEFT        = 0x00

    # flags for function set
    LCD_8BITMODE        = 0x10
    LCD_4BITMODE        = 0x00
```

```
    LCD_2LINE           = 0x08
    LCD_1LINE           = 0x00
    LCD_5x10DOTS        = 0x04
    LCD_5x8DOTS         = 0x00

    def __init__(self, pin_rs=18, pin_e=27, pins_db=[22, 23, 24, 25], GPIO =
None):
        # Emulate the old behavior of using RPi.GPIO if we haven't been given
        # an explicit GPIO interface to use
        if not GPIO:
            import RPi.GPIO as GPIO
            self.GPIO = GPIO
            self.pin_rs = pin_rs
            self.pin_e = pin_e
            self.pins_db = pins_db

            self.used_gpio = self.pins_db[:]
            self.used_gpio.append(pin_e)
            self.used_gpio.append(pin_rs)

            self.GPIO.setwarnings(False)
            self.GPIO.setmode(GPIO.BCM)
            self.GPIO.setup(self.pin_e, GPIO.OUT)
            self.GPIO.setup(self.pin_rs, GPIO.OUT)

            for pin in self.pins_db:
                self.GPIO.setup(pin, GPIO.OUT)

        self.write4bits(0x33) # initialization
        self.write4bits(0x32) # initialization
        self.write4bits(0x28) # 2 line 5x7 matrix
        self.write4bits(0x0C) # turn cursor off 0x0E to enable cursor
        self.write4bits(0x06) # shift cursor right

        self.displaycontrol = self.LCD_DISPLAYON | self.LCD_CURSOROFF |
self.LCD_BLINKOFF

        self.displayfunction = self.LCD_4BITMODE | self.LCD_1LINE |
self.LCD_5x8DOTS
        self.displayfunction |= self.LCD_2LINE

        """ Initialize to default text direction (for romance languages) """
        self.displaymode =  self.LCD_ENTRYLEFT | self.LCD_ENTRYSHIFTDECREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode) #  set the
```

```
entry mode

        self.clear()

    def begin(self, cols, lines):
        if (lines > 1):
            self.numlines = lines
            self.displayfunction |= self.LCD_2LINE
            self.currline = 0

    def home(self):
        self.write4bits(self.LCD_RETURNHOME) # set cursor position to zero
        self.delayMicroseconds(3000) # this command takes a long time!

    def clear(self):
        self.write4bits(self.LCD_CLEARDISPLAY) # command to clear display
        self.delayMicroseconds(3000)  # 3000 microsecond sleep, clearing the
display takes a long time

    def setCursor(self, col, row):
        self.row_offsets = [ 0x00, 0x40, 0x14, 0x54 ]

        if ( row > self.numlines ):
            row = self.numlines - 1 # we count rows starting w/0

        self.write4bits(self.LCD_SETDDRAMADDR | (col +
self.row_offsets[row]))

    def noDisplay(self):
        # Turn the display off (quickly)
        self.displaycontrol &= ~self.LCD_DISPLAYON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def display(self):
        # Turn the display on (quickly)
        self.displaycontrol |= self.LCD_DISPLAYON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noCursor(self):
        # Turns the underline cursor on/off
        self.displaycontrol &= ~self.LCD_CURSORON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def cursor(self):
```

```
        # Cursor On
        self.displaycontrol |= self.LCD_CURSORON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noBlink(self):
        # Turn on and off the blinking cursor
        self.displaycontrol &= ~self.LCD_BLINKON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def noBlink(self):
        # Turn on and off the blinking cursor
        self.displaycontrol &= ~self.LCD_BLINKON
        self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

    def DisplayLeft(self):
        # These commands scroll the display without changing the RAM
        self.write4bits(self.LCD_CURSORSHIFT | self.LCD_DISPLAYMOVE |
self.LCD_MOVELEFT)

    def scrollDisplayRight(self):
        # These commands scroll the display without changing the RAM
        self.write4bits(self.LCD_CURSORSHIFT | self.LCD_DISPLAYMOVE |
self.LCD_MOVERIGHT);

    def leftToRight(self):
        # This is for text that flows Left to Right
        self.displaymode |= self.LCD_ENTRYLEFT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode);

    def rightToLeft(self):
        # This is for text that flows Right to Left
        self.displaymode &= ~self.LCD_ENTRYLEFT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def autoscroll(self):
        # This will 'right justify' text from the cursor
        self.displaymode |= self.LCD_ENTRYSHIFTINCREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

    def noAutoscroll(self):
        # This will 'left justify' text from the cursor
        self.displaymode &= ~self.LCD_ENTRYSHIFTINCREMENT
        self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)
```

```
    def write4bits(self, bits, char_mode=False):
        # Send command to LCD
        self.delayMicroseconds(1000) # 1000 microsecond sleep
        bits=bin(bits)[2:].zfill(8)
        self.GPIO.output(self.pin_rs, char_mode)
        for pin in self.pins_db:
            self.GPIO.output(pin, False)
        for i in range(4):
            if bits[i] == "1":
                self.GPIO.output(self.pins_db[::-1][i], True)
        self.pulseEnable()
        for pin in self.pins_db:
            self.GPIO.output(pin, False)
        for i in range(4,8):
            if bits[i] == "1":
                self.GPIO.output(self.pins_db[::-1][i-4], True)
        self.pulseEnable()

    def delayMicroseconds(self, microseconds):
        seconds = microseconds / float(1000000)   # divide microseconds by 1
million for seconds
        sleep(seconds)

    def pulseEnable(self):
        self.GPIO.output(self.pin_e, False)
        self.delayMicroseconds(1)      # 1 microsecond pause - enable pulse
must be > 450ns
        self.GPIO.output(self.pin_e, True)
        self.delayMicroseconds(1)      # 1 microsecond pause - enable pulse
must be > 450ns
        self.GPIO.output(self.pin_e, False)
        self.delayMicroseconds(1)      # commands need > 37us to settle

    def message(self, text):
        # Send string to LCD. Newline wraps to second line
        print "message: %s"%text
        for char in text:
            if char == '\n':
                self.write4bits(0xC0) # next line
            else:
                self.write4bits(ord(char),True)

    def destroy(self):
        print "clean up used_gpio"
```

```
        self.GPIO.cleanup(self.used_gpio)

def print_msg():
    print ("========================================")
    print ("|                LCD1602                |")
    print ("|           Control LCD1602             |")
    print ("========================================\n")
    print 'Program is running...'
    print 'Please press Ctrl+C to end the program...'
    raw_input ("Press Enter to begin\n")

def main():
    global lcd
    print_msg()
    lcd = LCD()
    line0 = "-----hello------"
    line1 = "-----world------"

    lcd.clear()
    lcd.message("Welcome to --->\n    raspberry pi")
    sleep(3)

    msg = "%s\n%s" % (line0, line1)
    while True:
        lcd.begin(0, 2)
        lcd.clear()
        for i in range(0, len(line0)):
            lcd.setCursor(i, 0)
            lcd.message(line0[i])
            sleep(0.1)
        for i in range(0, len(line1)):
            lcd.setCursor(i, 1)
            lcd.message(line1[i])
            sleep(0.1)
        sleep(1)

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        lcd.clear()
        lcd.destroy()
```

Experimental results:

In the directory where the code file is located, execute the following command

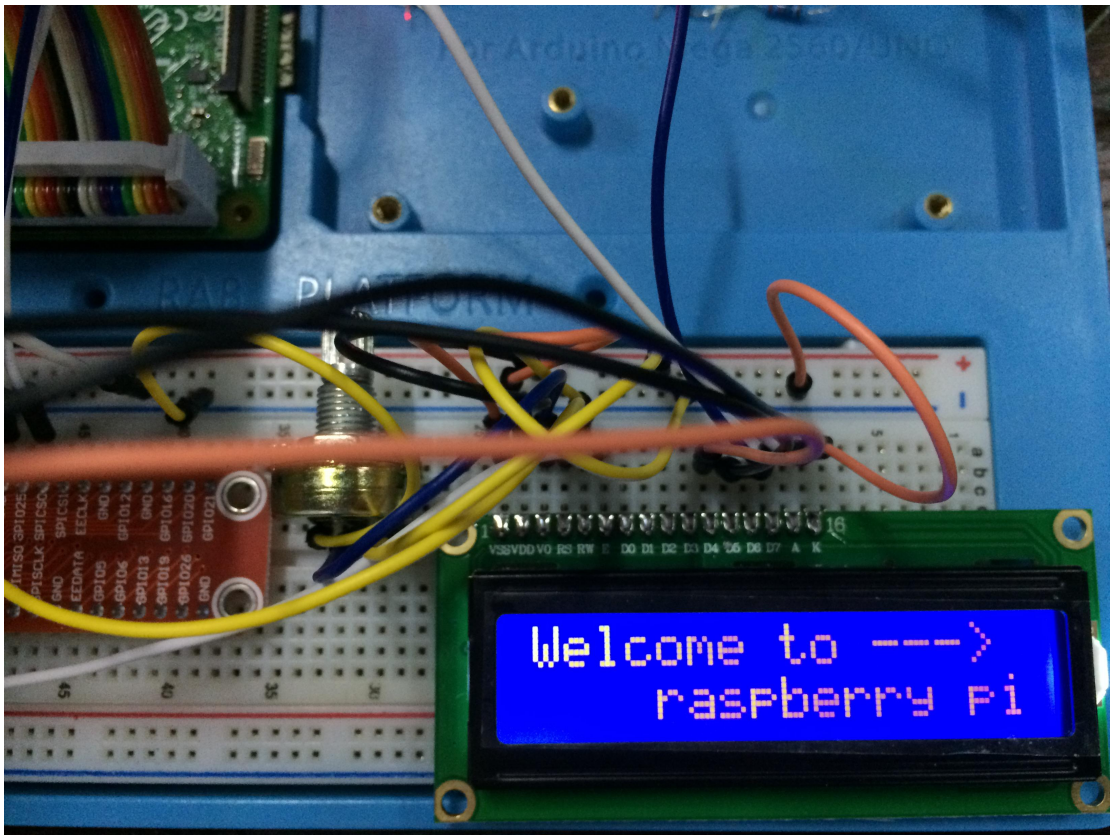
C:

```
gcc -Wall -o lcd1602 lcd1602.c -lwiringPi -lwiringPiDev
sudo ./lcd1602
```

Python:

```
python lcd1602.py
```

After the instruction is executed, lcd will display the content



Please turn the potentiometer to adjust the contrast if the LCD shows n othing