San Francisco Bay University

CE450 Fundamentals of Embedded Engineering
Lab 9 LCD Display

Objectives:

In this week, students will design the program to display the characters and strings through GPIO ports on Raspberry Pi bord and do hands-on exercise through lab assignments

Introduction:

We will learn how to use LCD1602 to display characters and strings as the exercises in the lab section

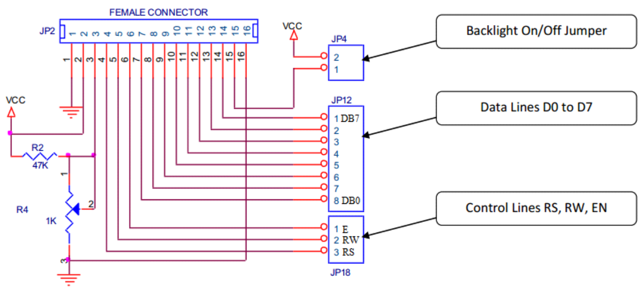
Equipment:

The equipment you require is as follows:

- Laptop & Raspberry Pi 3 model Board
- SunFounder Super Starter Kit V2.0 for Raspberry Pi
- LCD1602 display

The Laboratory Procedure:

1. **Hardware connection**



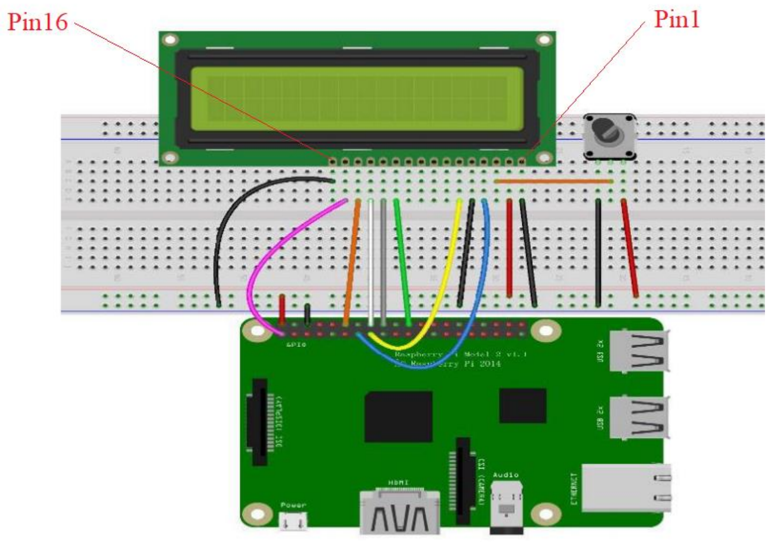
P15-P16: Backlight
P3: LCD Contrast Adjustment

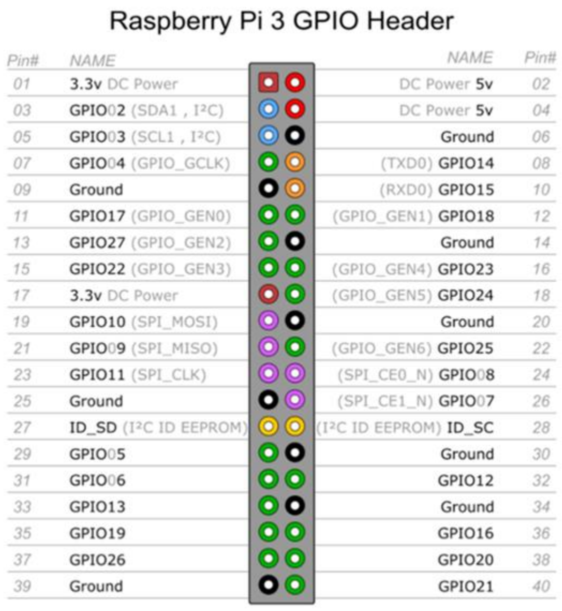
DB0-DB7: Data & Command inputs
RS--- Instruction/Date Selection
E--- Enable
R/W=0 Always in Write Mode

PIN NO.	SYMBOL	DESCRIPTION	FUNCTION
1	VSS	GROUND	0V (GND)
2	VCC	POWER SUPPLY FOR LOGIC CIRCUIT	+5V
3	VEE	LCD CONTRAST ADJUSTMENT	

4	RS	INSTRUCTION/DATA REGISTER SELECTION	RS = 0 : INSTRUCTION REGISTER
			RS = 1 : DATA REGISTER

5	R/W	READ/WRITE SELECTION	R/W = 0 : REGISTER WRITE R/W = 1 : REGISTER READ
6	E	ENABLE SIGNAL	
7	DB0	DATA INPUT/OUTPUT LINES	8 BIT: DB0-DB7
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		
15	LED+	SUPPLY VOLTAGE FOR LED+	+5V
16	LED-	SUPPLY VOLTAGE FOR LED-	0V




Raspberry Pi 3 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I²C)	DC Power 5v	04
05	GPIO03 (SCL1 , I²C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)	(I²C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

2. Control program in Python

```
# Python Program

from time import sleep

class LCD:
        # commands
        LCD_CLEARDISPLAY                = 0x01         # 0000_0001
        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=27, pin_e=22, pins_db=[25, 24, 23, 18], 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()
```

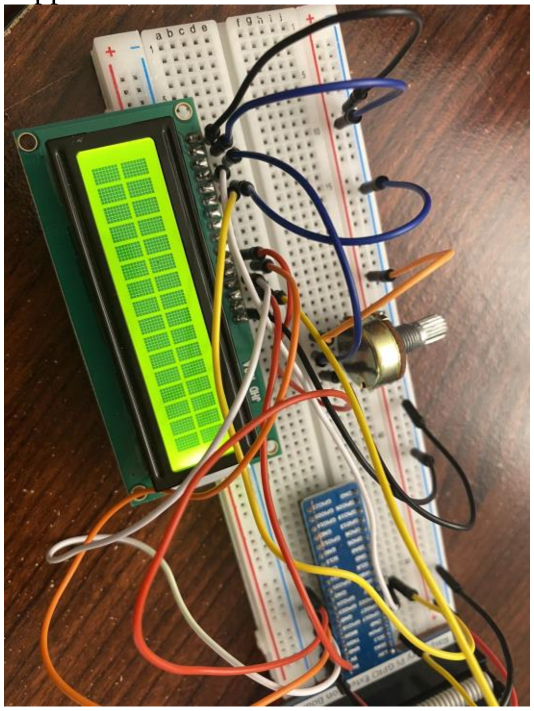
```
def begin(self, cols, lines):
        if (lines > 1):
                self.numlines = lines
                self.displayfunction |= self.LCD_2LINE
                self.currline = 0

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): # one of functions in class LCD
        # 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 ("|    ------------------------------    |")
    print ("|         D4 connect to BCM25          |")
    print ("|         D5 connect to BCM24          |")
    print ("|         D6 connect to BCM23          |")
    print ("|         D7 connect to BCM18          |")
    print ("|         RS connect to BCM27          |")
    print ("|         CE connect to bcm22          |")
    print ("|          RW connect to GND           |")
    print ("|                                      |")
    print ("|           Control LCD1602            |")
    print ("|                                      |")
    print ("|                            SunFounder|")
    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()            # obj: lcd
    line0 = "  sunfounder.com"
    line1 = "---SUNFOUNDER---"

    lcd.clear()
    lcd.message("Welcome to --->\n  sunfounder.com")
    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()
```

**Note: Hardware connection reference and running command*
https://learn.sunfounder.com/lesson-16-lcd1602/
https://learn.sunfounder.com/category/super-kit-v3-0-for-raspberry-pi/

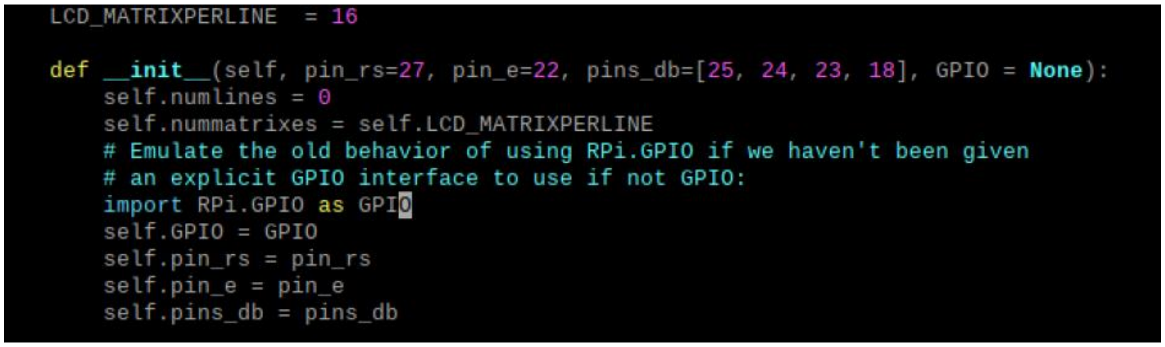
The Laboratory Assignments:

1. Build up the hardware circuit and run the example program to observe what will happen

Youtube link:

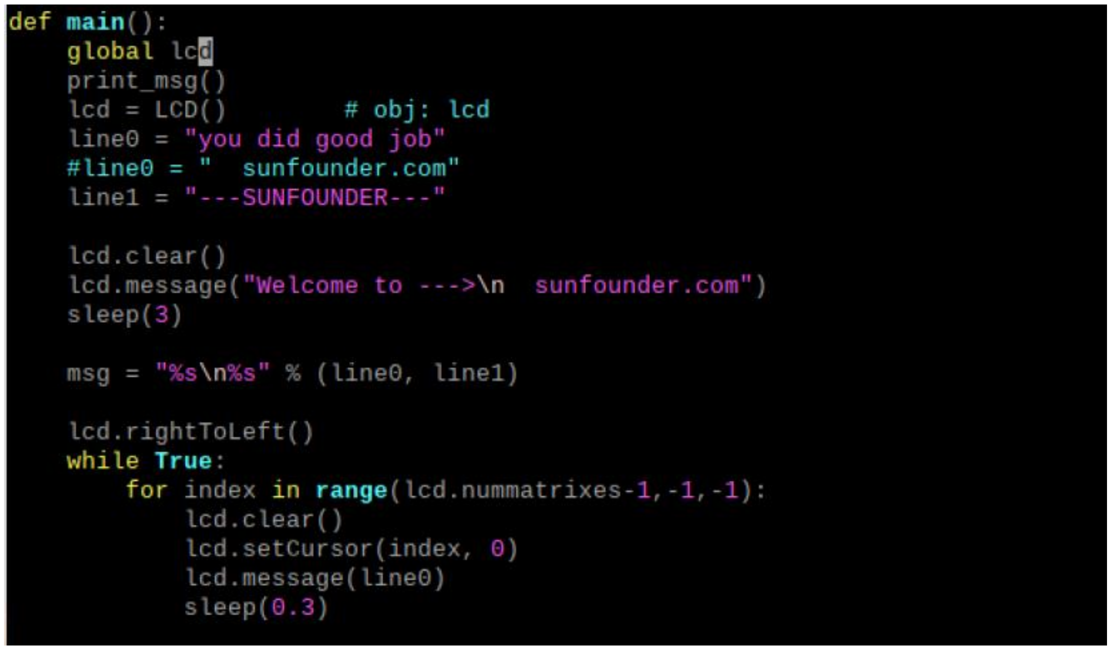
https://youtube.com/shorts/8EaOnyF-IwM?feature=share

2. Display “you did good job” in LCD by left shifting from right.

Code:



```
    LCD_MATRIXPERLINE   = 16

    def __init__(self, pin_rs=27, pin_e=22, pins_db=[25, 24, 23, 18], GPIO = None):
        self.numlines = 0
        self.nummatrixes = self.LCD_MATRIXPERLINE
        # 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
```



```
def main():
    global lcd
    print_msg()
    lcd = LCD()         # obj: lcd
    line0 = "you did good job"
    #line0 = "  sunfounder.com"
    line1 = "---SUNFOUNDER---"

    lcd.clear()
    lcd.message("Welcome to --->\n  sunfounder.com")
    sleep(3)

    msg = "%s\n%s" % (line0, line1)

    lcd.rightToLeft()
    while True:
        for index in range(lcd.nummatrixes-1,-1,-1):
            lcd.clear()
            lcd.setCursor(index, 0)
            lcd.message(line0)
            sleep(0.3)
```

Youtube link:
https://youtube.com/shorts/8epgbHchUp0?feature=share