|  |  |  |  |
| --- | --- | --- | --- |
| **7** | **Interfacing Raspberry Pi with 16x2 LCD using I2C module.** | | |
|  | Step 1 – Connect LCD Screen to the Pi  The I2c module can be powered with either 5V or 3.3V but the screen works best if it provided with 5V. However the Pi’s GPIO pins aren’t 5V tolerant so the I2C signals need to be level shifted. To do this I used an I2C level shifter.  This requires a high level voltage (5V) and a low level voltage (3.3V) which the device uses as a reference. The HV pins can be connected to the screen and two of the LV pins to the Pi’s I2C interface.    While experimenting I found that it worked fine without the level shifting but I couldn’t be certain this wasn’t going to damage the Pi at some point. So it’s probably best to play it safe!  Step 2 – Download the Example Python Script  The example script will allow you to send text to the screen via I2C. It is very similar to my scripts for [the normal 16×2 screen](https://www.raspberrypi-spy.co.uk/2012/07/16x2-lcd-module-control-using-python/). To download the script directly to your Pi you can use : | | |
|  | wget https://bitbucket.org/MattHawkinsUK/rpispy-misc/raw/master/python/lcd\_i2c.py |  |
|  | | |

T.Y.B.Sc. (I.T.) 2019

|  |  |
| --- | --- |
|  | Step 3 – Enable the I2C Interface  In order to use I2C devices you must enable the interface on your Raspberry Pi. This can be done by following m[y “Enabling The I2C Interface On The Raspberry Pi](https://www.raspberrypi-spy.co.uk/2014/11/enabling-the-i2c-interface-on-the-raspberry-pi/)” tutorial. By default the I2C backpack will show up on address 0x27.  Step 4 – Run LCD Script  The script can be run using the following command : sudo python lcd\_i2c.py |
| Code:  import smbus import time  # Define some device parameters I2C\_ADDR = 0x27 # I2C device address  LCD\_WIDTH = 16 # Maximum characters per line  # Define some device constants LCD\_CHR = 1 # Mode - Sending data  LCD\_CMD = 0 # Mode - Sending command  LCD\_LINE\_1 = 0x80 # LCD RAM address for the 1st line LCD\_LINE\_2 = 0xC0 # LCD RAM address for the 2nd line LCD\_LINE\_3 = 0x94 # LCD RAM address for the 3rd line LCD\_LINE\_4 = 0xD4 # LCD RAM address for the 4th line  LCD\_BACKLIGHT = 0x08 # On #LCD\_BACKLIGHT = 0x00 # Off  ENABLE = 0b00000100 # Enable bit # Timing constants  E\_PULSE = 0.0005  E\_DELAY = 0.0005  #Open I2C interface  #bus = smbus.SMBus(0) # Rev 1 Pi uses 0 bus = smbus.SMBus(1) # Rev 2 Pi uses 1  def lcd\_init(): |

T.Y.B.Sc. (I.T.) 2019

|  |  |
| --- | --- |
|  | # Initialise display  lcd\_byte(0x33,LCD\_CMD) # 110011 Initialise lcd\_byte(0x32,LCD\_CMD) # 110010 Initialise lcd\_byte(0x06,LCD\_CMD) # 000110 Cursor move direction lcd\_byte(0x0C,LCD\_CMD) # 001100 Display On,Cursor Off, Blink Off  lcd\_byte(0x28,LCD\_CMD) # 101000 Data length, number of lines, font size lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display  time.sleep(E\_DELAY)  def lcd\_byte(bits, mode):  # Send byte to data pins # bits = the data  # mode = 1 for data # 0 for command  bits\_high = mode | (bits & 0xF0) | LCD\_BACKLIGHT bits\_low = mode | ((bits<<4) & 0xF0) | LCD\_BACKLIGHT  # High bits bus.write\_byte(I2C\_ADDR, bits\_high) lcd\_toggle\_enable(bits\_high)  # Low bits bus.write\_byte(I2C\_ADDR, bits\_low) lcd\_toggle\_enable(bits\_low)  def lcd\_toggle\_enable(bits):  # Toggle enable time.sleep(E\_DELAY)  bus.write\_byte(I2C\_ADDR, (bits | ENABLE)) time.sleep(E\_PULSE) bus.write\_byte(I2C\_ADDR,(bits & ~ENABLE)) time.sleep(E\_DELAY)  def lcd\_string(message,line):  # Send string to display  message = message.ljust(LCD\_WIDTH," ") lcd\_byte(line, LCD\_CMD)  for i in range(LCD\_WIDTH): lcd\_byte(ord(message[i]),LCD\_CHR)  def main():  # Main program block |

T.Y.B.Sc. (I.T.) 2019

|  |  |
| --- | --- |
|  | # Initialise display lcd\_init()  while True:  # Send some test  lcd\_string("RPiSpy <",LCD\_LINE\_1) lcd\_string("I2C LCD <",LCD\_LINE\_2)  time.sleep(3)  # Send some more text  lcd\_string("> RPiSpy",LCD\_LINE\_1) lcd\_string("> I2C LCD",LCD\_LINE\_2)  time.sleep(3)  if name == ' main ': try:  main()  except KeyboardInterrupt: pass  finally:  lcd\_byte(0x01, LCD\_CMD) |