PROM: an introduction to Pi, GPIO and I2C

The aim of this lab is to introduce the programming environment and hardware interfaces used on the Raspberry Pi i.e. General Purpose Input Output (GPIO) and the Inter-Integrated Circuit (I2C) serial communications bus. The desktop computer must be booted into Linux, this laboratory will **not** work in the Windows operating system. When prompted enter your user name and password as shown in figure 1.

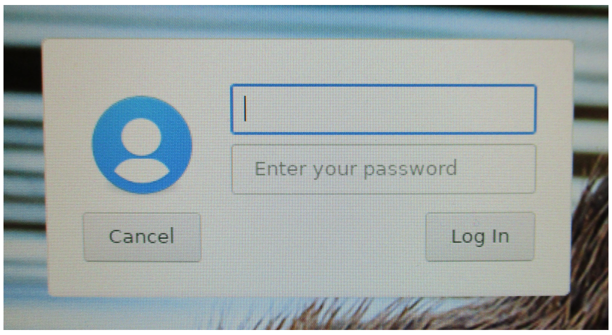


Figure 1: login prompt

Next, connect the Raspberry Pi as shown in figure 2. You will find the Ethernet cables already installed, only use the YELLOW or PURPLE Ethernet cable, other cables will not work. The Pi and its power supply cable are available from the front desk.

IMPORTANT: only use the fixed +5V from the bench power supply, plugs from other power supply modules may fit into this socket, however, this will destroy the raspberry pi!

Plug in the power and Ethernet cables as shown in figure 2 and turn on the Pi. When the Pi has finished booting the top LED bar will flash 5 times.

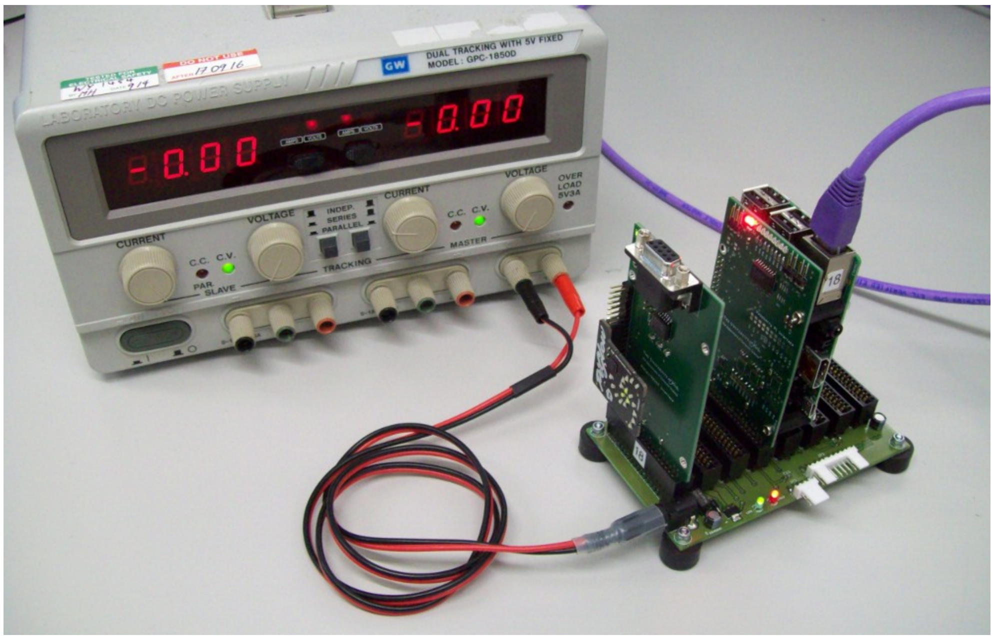
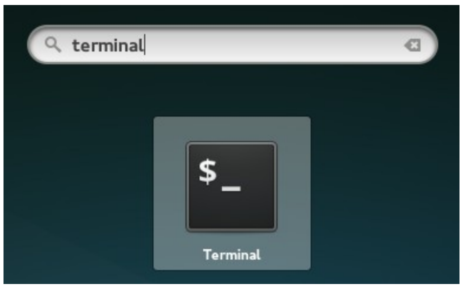
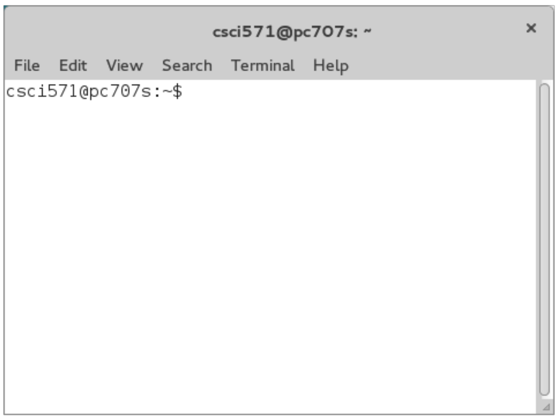


Figure 2: Raspberry Pi module (number of cards may vary)

Configuring the Desktop (Local) Computer

To start a program on the PC left click on the Activities button, top left of the screen. This will open the main options menu bar on the left hand side and a search box top centre. In the search box type the name `terminal` and press enter. This will launch a terminal or command prompt, as shown in figure 3. The command prompt will indicate your user and machine name e.g. in figure 3 my user name is `csci571` and the local machine name is `pc707s`.





```
csci571@pc707s:~$
```

Figure 3: command terminal

Note, the keyboard shortcut to open a terminal is : `CTRL + ALT + t`

To connect to the Raspberry Pi we will be using the Secure SHell (SSH) protocol. The network address used by the Pi is dynamically allocated by the local computer. At the command prompt type :

```
dudewheresmypi
```

Tip, type 'dude' then press the `TAB` key to auto complete. This will return the IPv4, four byte, private network address, typically `192.168.1.2`. If this script returns the message "no Pi detected" you may need to wait a minute for the required log files to update on the PC. Once you have the network address, to log into the Pi type into the command prompt:

`ssh -lguest 192.168.1.2` or `ssh guest@192.168.1.2`

Note, you may need to change the address i.e. `192.168.1.X`. This will open a secure shell within the terminal window using the user name `guest` and the Pi network address `192.168.1.2`. Password is also `guest`.

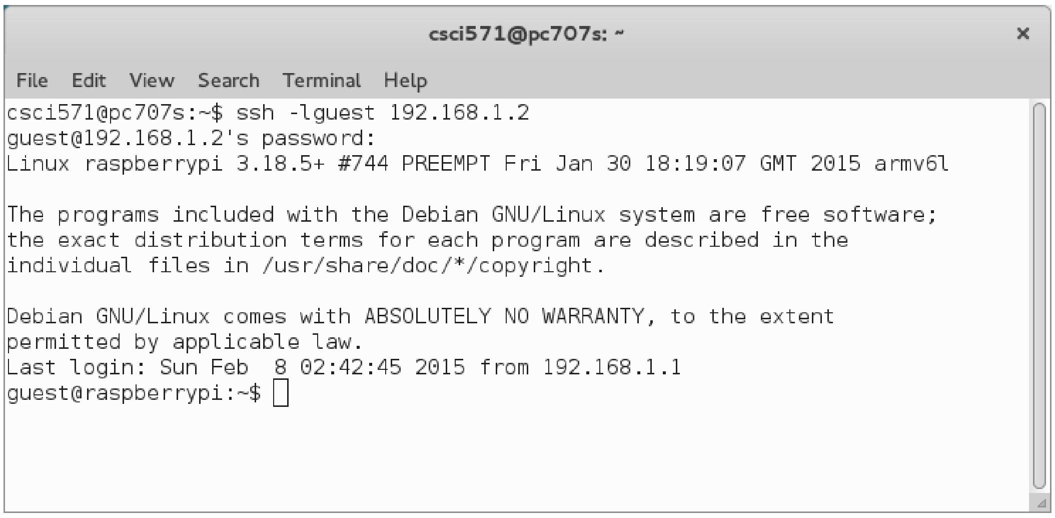
If this is the first time you have logged into the Pi a warning message may appear, as shown below. At the command prompt type: `yes`.

```
The authenticity of host '192.168.1.2 (192.168.1.2)' can't be established.
ECDSA key fingerprint is 2f:7b:a6:88:41:a3:d6:23:df:c0:90:59:03:63:5b:98.
Are you sure you want to continue connecting (yes/no)? yes
```

You will now be asked to type in a password, enter: `guest`.

This process opens a terminal / command prompt on the Pi, indicated by the change of user and machine name in the terminal window i.e. user name `guest` and machine name `raspberrypi`, as shown in figure 4. We can now use this terminal to execute a python program on the Pi.

Failure to connect : one possible reason is that the Pi has not finished its boot sequence (normally takes approximately 1 – 2 minutes, red LED strip will flash 5 times), or has failed to boot. Turn the power supply off, then on again, wait approximately a minute (until the green LED on the Pi stops flashing) and try again. If all else fails please ask for assistance.



```
csci571@pc707s:~$ ssh -lguest 192.168.1.2
guest@192.168.1.2's password:
Linux raspberrypi 3.18.5+ #744 PREEMPT Fri Jan 30 18:19:07 GMT 2015 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Feb  8 02:42:45 2015 from 192.168.1.1
guest@raspberrypi:~$ 
```

Figure 4: ssh login

To view files on the local machine and the Pi left click on the Activities button, top left of the screen. Next, click on the file browser icon on the left hand side menu bar. If you wish to search the available applications click on the browse icon.

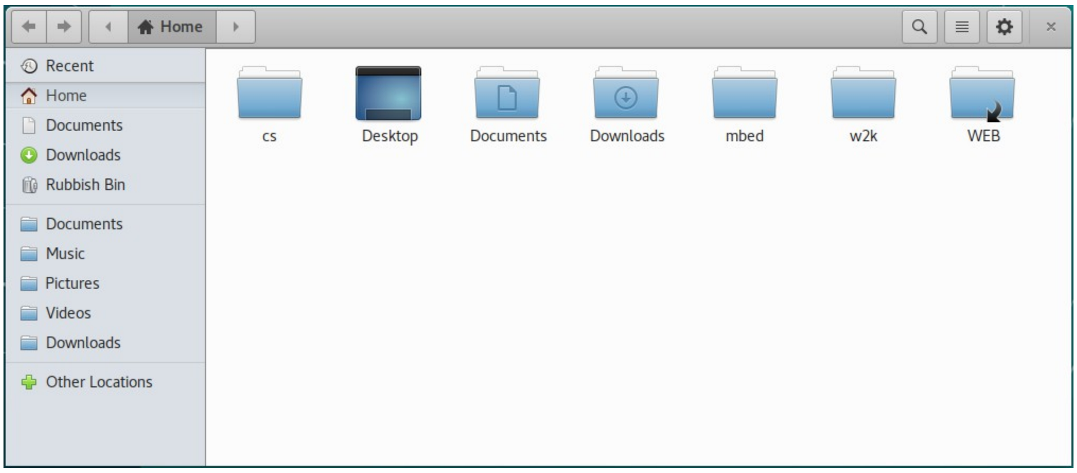


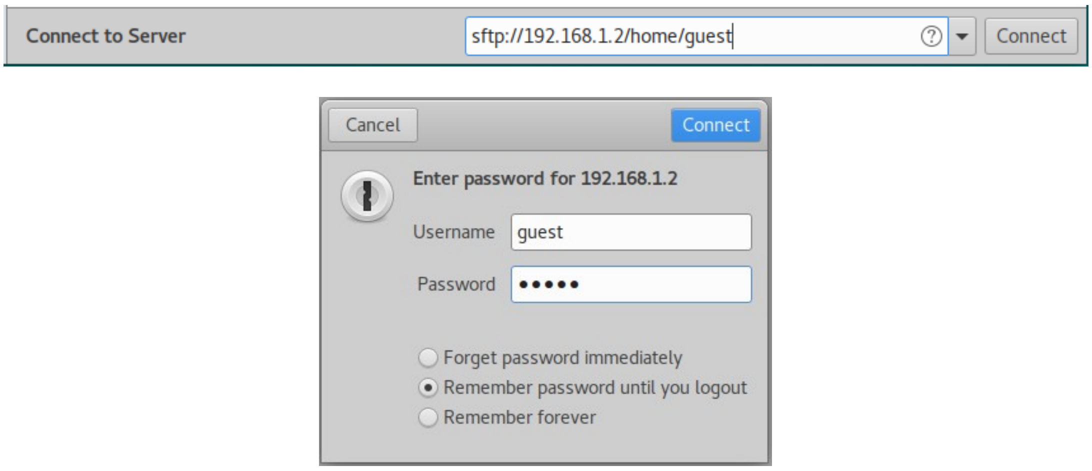
Figure 5: file browser

The easiest way to move files onto and off the Pi is to use the Secure File Transfer Protocol (SFTP). To mount the Pi file system left click on the Other Locations button, bottom left of the file browser window. This will open the 'Connect to Server' panel, as shown in figure 6. Within the 'Server Address' box type the IP address of the Pi in this example `192.168.1.2`:

```
sftp://192.168.1.2/home/guest
```

Note, If this is the first time the Pi has been connected to the local computer a "`Can't verify the identify`" message will be displayed, click on the "`Log in Anyway`" button.

You will then be asked to enter your user name and password, enter `guest` for both. When you are connected the file browser window will update to show you the contents of this (empty) directory on the Pi, as shown in figure 7.



Connect to Server: sftp://192.168.1.2/home/guest

Enter password for 192.168.1.2

Username: guest

Password: •••••

Forget password immediately [✓ Remember password until you logout] Remember forever

Figure 6: sftp login

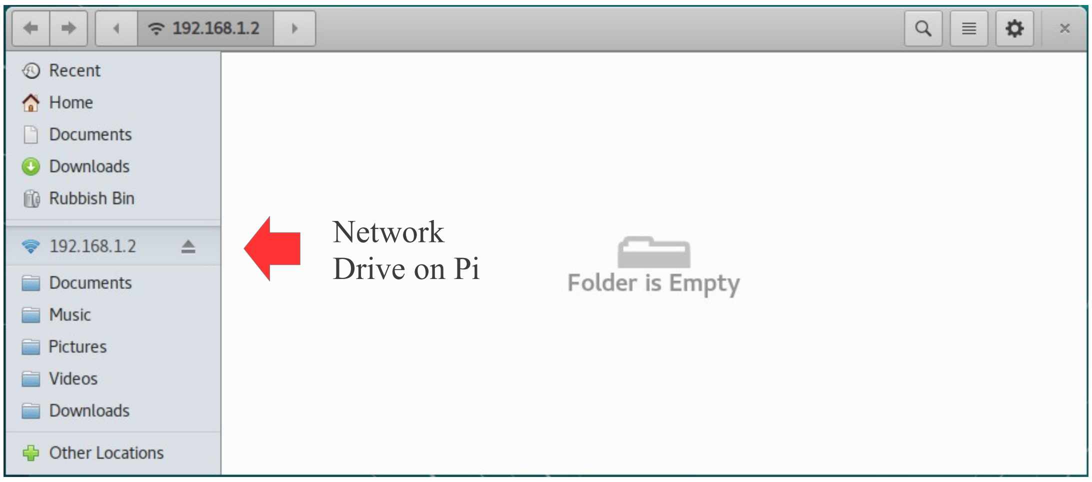


Figure 7: file browser - home directory on Pi

To enter your python code you can either use a text editor within the terminal window or use an editor on the local computer. My preferred text editor is `vi`, however, you may find this a little tricky to use, if so skip forward in the laboratory script to the description of `gedit`.

Within the Raspberry Pi terminal running `SSH` type:

```
vi test.py
```

This will open a new file called `test.py` in the `vi` text editor. This editor has two main modes of operation: edit and view. To allow you to edit (insert) this file press the `i` key, then type in the code:

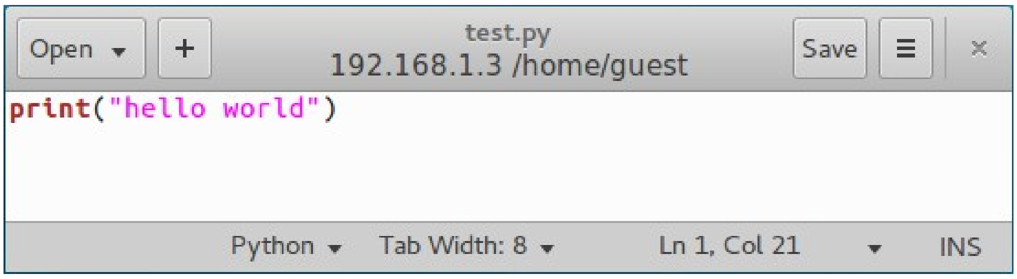
```
print("Hello World")
```

To return back to view mode press the `ESC` key. To save and exit, hold down the shift key and press the `z` key twice i.e. enter `ZZ`. This will return you back to the command prompt.

Note, at first `vi` is tricky to use, but I would recommend you stick with it as it's one of the standard editors used in Linux. Some other useful commands:

- x : delete a character
- dd : delete a line
- yy : copy (yank) a line
- p : paste a line
- A : insert characters at the end of a line
- Cursor Keys : use in view mode to move around a document

An alternative to `vi` is `nano`. Type `nano` at the command prompt to start this editor. Commands are displayed at the bottom of the screen. Note, '^' is the `CTRL` button, therefore to save a file press: `CTRL + o`. If you prefer a GUI based editor you can use `gedit` on the local computer.



```
print("hello world")
```

Figure 8: gedit

To start `gedit` on the local computer, left click on the Activities button, then in the search box type `gedit` and press enter. This will launch a standard text editor, as shown in figure 8. Enter the code:

```
print("Hello World")
```

To save this program on the Pi's filing system left click on the Save button, top right. Within the 'Save As' window enter the file name `test.py`. Next, click on the mounted Pi home directory 192.168.1.3 button. Finally click `Save`. Within the file browser press the `F5` key to refresh the display, a new python file icon should appear named `test.py`.

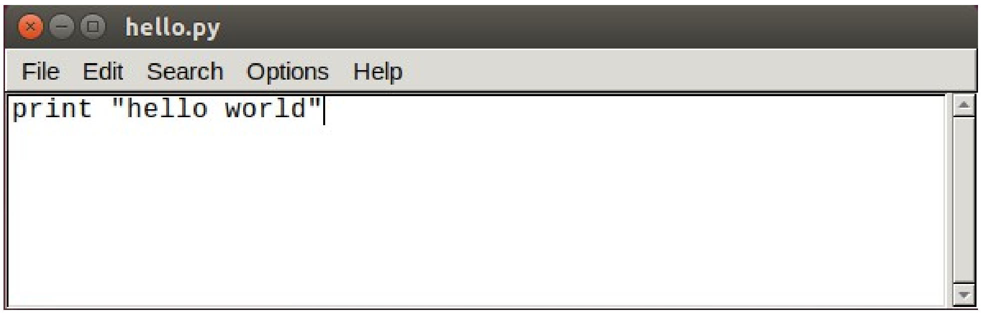
Note, do not click the unmount button ⏏. If you accidentally unmount the Pi filing system repeat the previous steps using `sftp`.

A final method to open a text editor on the Pi is to enable X11 forwarding i.e. run a GUI based editor on the Pi, but display the widow on the PC. To enable this method you must `SSH` into the Pi with the `-X` option, as shown below:

```
ssh -X -lguest 192.168.1.2   or  ssh -X guest@192.168.1.2
```

Note, again, you may need to change the address i.e. `192.168.1.X`.

Next, within this terminal window enter the command: `leafpad &`, this will open the `leafpad` text editor GUI on the PC, as shown in figure 9, allowing you to edit and save files on the Pi. Note, the "`&`" means run in the background.



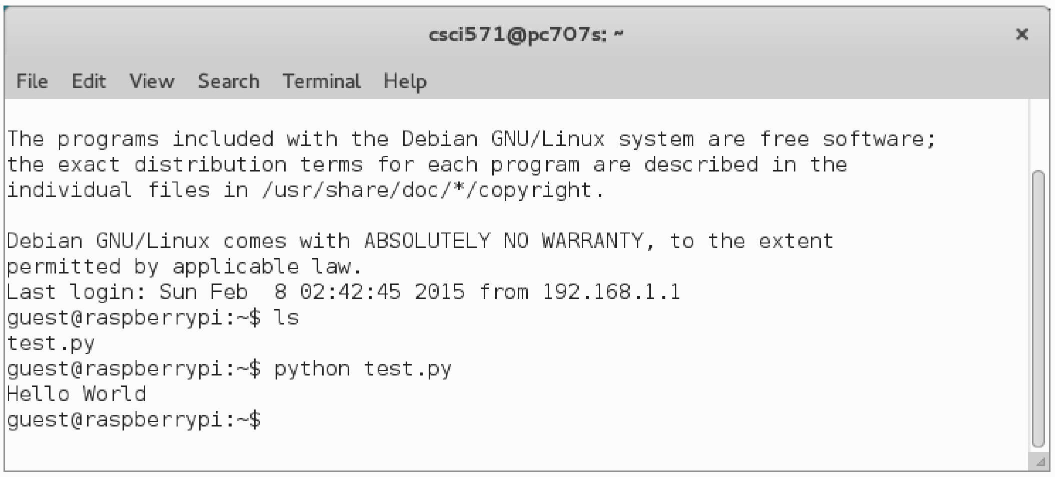
```
print "hello world"
```

Figure 9: leafpad

Once you have entered the test program, within the command terminal running `SSH` you can list the files within the working directory by typing `ls`. To execute this program type:

```
python test.py
```

This will print `Hello World` to the terminal, as shown in figure 10.



```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Feb  8 02:42:45 2015 from 192.168.1.1
guest@raspberrypi:~$ ls
test.py
guest@raspberrypi:~$ python test.py
Hello World
guest@raspberrypi:~$
```

Figure 10: executing a program on the Pi

WARNING: all files will be deleted from the guest account when the Raspberry Pi is turned off. Make sure you copy any files you wish to keep to the local computer using the file browser before shutting down the Pi.

To shut down the Pi, within the command terminal running `SSH` type:

```
sudo halt
```

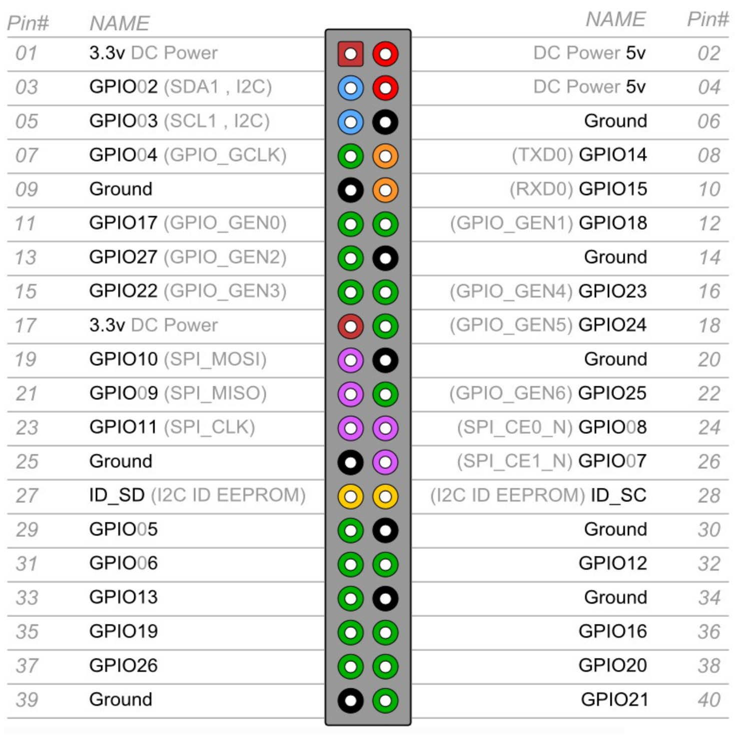
Note, sudo allows a user to run a program with different security privileges, in this case root (administrator).

General Purpose Input Output (GPIO)

The Pi can be interfaced to external hardware using a 40 pin header, as shown in figure 11. The GPIO pins used in this laboratory are GPIO10 and GPIO9, however the same python code can be used to control any of the GPIO pins.

To protect the Pi, GPIO signals are passed through a buffer board. This allows you to interface the Pi to a wide range of logic devices i.e. IC families that use different voltages (+3V to +5V) to represent a logic '1'. Without these level shifters (voltage converts / buffers) you could damage the Pi. If your not sure about anything do ask.

NEVER CONNECT +5V DIRECTLY TO A GPIO PIN ON OTHER RASPBERRY PIs, AS IT WILL DESTROYED THE HARDWARE



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

Figure 11: Pi GPIO header

To use GPIO pin 10 as an output, save the code listed in figure 12 to the file `output.py`. If you are using `gedit` transfer this file to the Pi guest directory using the file browser.

```
import RPi.GPIO as GPIO         #IO library
import time                     #Time library

GPIO.setwarnings(False)         #disable runtime warnings
GPIO.setmode(GPIO.BCM)          #use Broadcom GPIO names

GPIO.setup(10, GPIO.OUT)        #set pin 10 as output

while True:                     #infinite loop
     GPIO.output(10, True)      #set pin 10 high
     time.sleep(0.5)            #wait 1/2 sec
     GPIO.output(10, False)     #set pin 10 low
     time.sleep(0.5)            #wait 1/2 sec
```

Figure 12: Pi GPIO output code

This program will output a square wave, switching between 0V and +3.3V, with a period of approximately 1 second i.e. a frequency of 1 Hz. GPIO Pins 9, 10 and 11 can be accessed from the Pi base board as shown in figure 13.

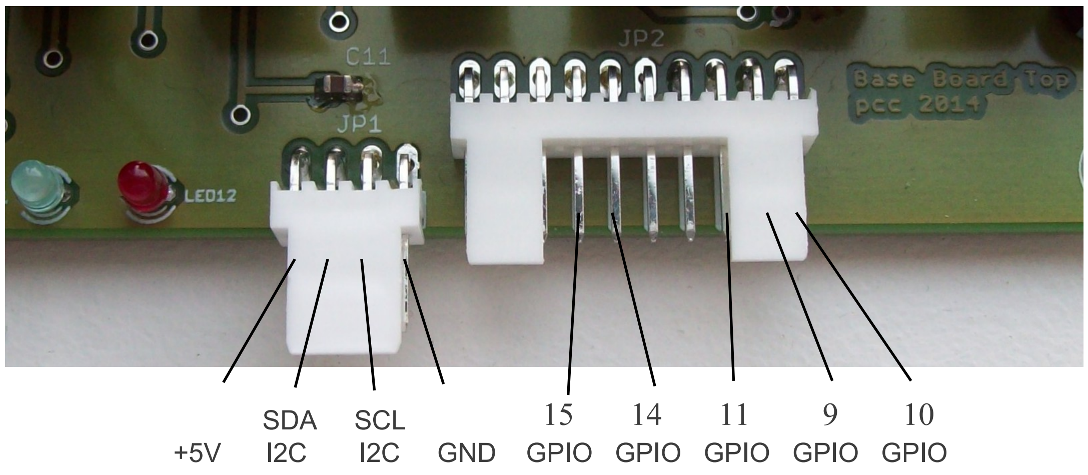


Figure 13: Pi GPIO output pins (hidden under white plastic)

To view the state of GPIO pin 10 an LED can be connected to this pin. Construct the circuit shown in figures 14, 15 and 16. All components are available from the component draws at the front of the lab, bread-board and connecting wires are available from the side desk.

Note, if possible its good practice to colour code your connecting wires. In the example shown in figure 15, **GREEN** is a signal (GPIO10), **RED** is VCC (+5V, not used in this example) and **BLACK** is 0V. To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python output.py
```

You should see the LED flash ON for half a second and OFF for half a second. To stop the program press `CTRL C`.

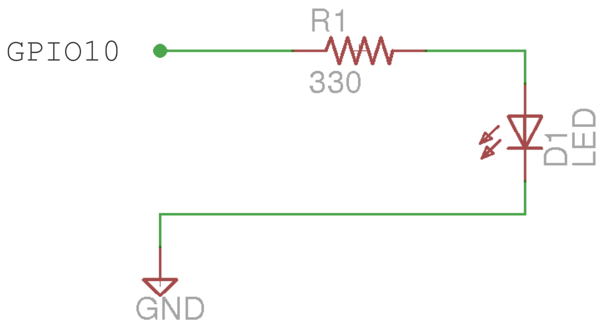


Figure 14: LED display

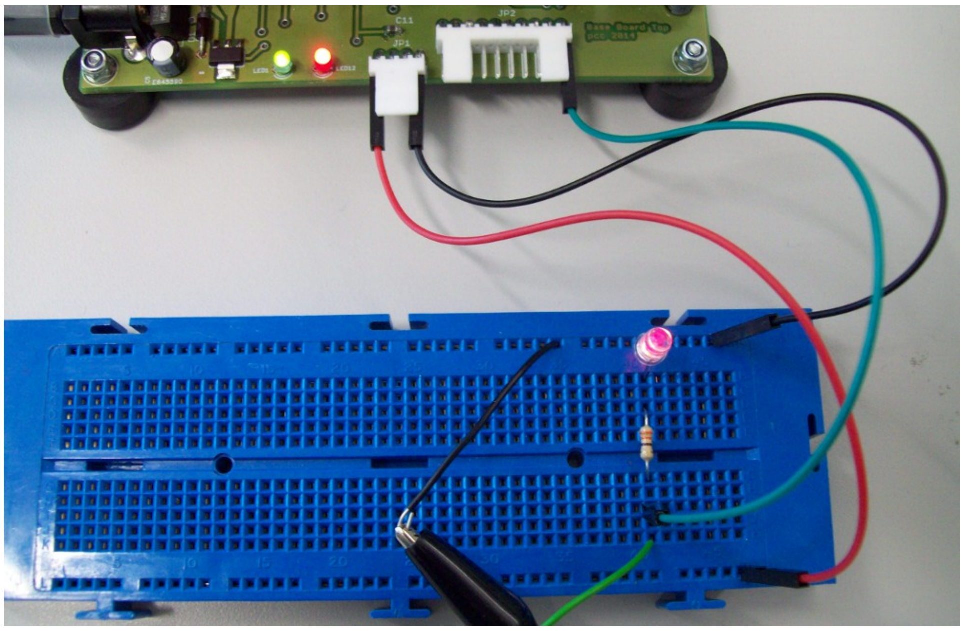


Figure 15: LED display wiring

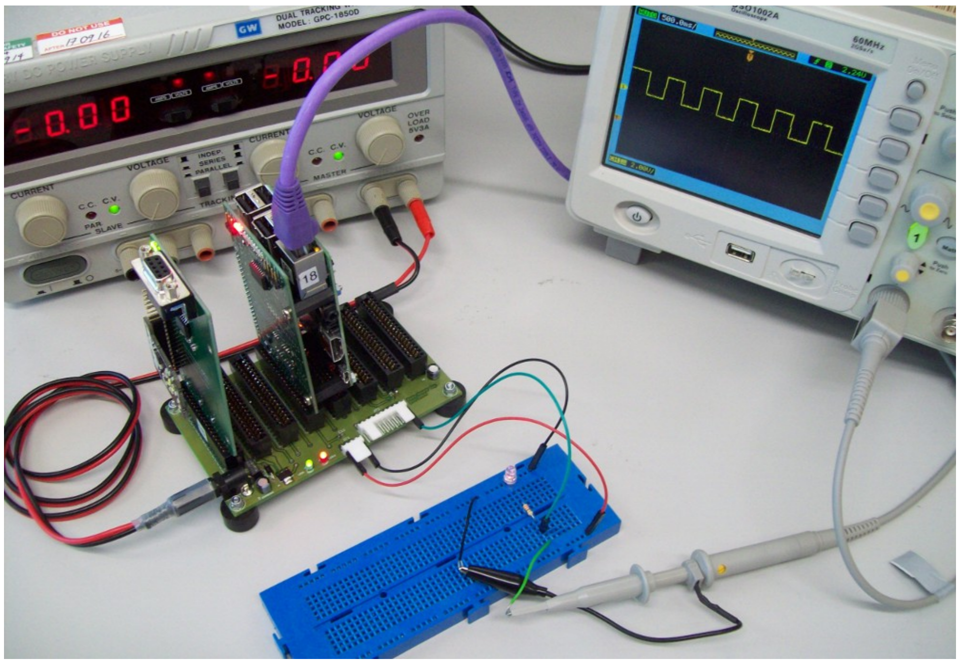


Figure 16: LED test system

Try increasing the output frequency e.g. reduce the time delay from 0.5s to 0.05s. At this speed you can not see the LED flash, however as the LED is on for 50% of the time and off for 50% of the time it will appear dimmer i.e. persistence of vision.

This technique i.e. pulse width modulation (PWM), can be used to control the brightness of the LED and has been built into the GPIO library. Enter and save the file `pulse.py` shown in figure 17. If you are using `gedit` transfer this file to the Pi guest directory using the file browser.

```
import RPi.GPIO as GPIO        #IO library
import time                    #Time library

GPIO.setwarnings(False)        #disable runtime warnings
GPIO.setmode(GPIO.BCM)         #use Broadcom GPIO names

GPIO.setup(10, GPIO.OUT)       #set pin 10 as output
PWM = GPIO.PWM(10, 100)        #set freq 100Hz
PWM.start(0)                   #turn off LED

try:
     while True:
          for i in range(5,100,5):
               PWM.ChangeDutyCycle(i)
               time.sleep(0.1)
          for i in range(100,0,-5):
               PWM.ChangeDutyCycle(i)
               time.sleep(0.1)

except KeyboardInterrupt:
     pass

PWM.stop()
GPIO.cleanup()
```

Figure 17: Pi GPIO output code

To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python pulse.py
```

You should see the LED increase then decrease in brightness. To stop the program press `CTRL C`.

Note, a try–except block has been used in this program, this give a move graceful exit path, allowing the program to turn off GPIO pins etc.

To hear the state of GPIO pin 10 a piezoelectric buzzer can be connected to this pin. Construct the circuit shown in figures 18 and 19. Again, components are available from the component draws at the front of the lab.

Execute the program `output.py` as previously described. You should hear a clicking sound from the buzzer. **Note**, the buzzer is very quiet, you may need to attach a plastic/paper cup to amplify the sound (available on the side desk). Edit you program to increase the frequency of the buzzer e.g. a period of 10 ms.

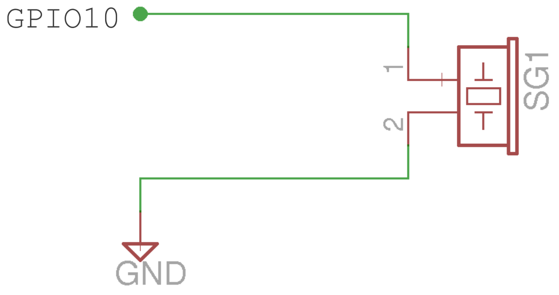


Figure 18: Piezoelectric buzzer circuit

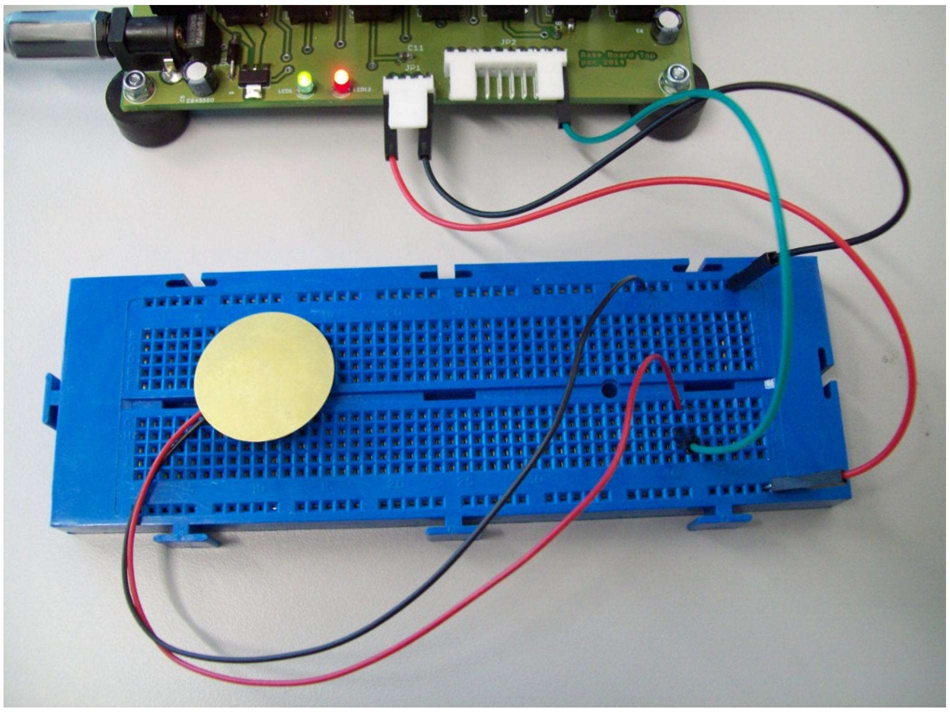


Figure 19: Piezoelectric buzzer test system

Note, the maximum output frequency is limited by the time it takes to execute each python instruction. Removing the sleep function, the maximum frequency is approximately 100KHz. However, as the Pi is running other software i.e. the operating system (OS), its not particularly square, this can be observed using an oscilloscope.

Task 1: write a program to incrementally increase the output frequency from 100Hz to 1KHz and then back down again i.e. ramp up, then ramp down. You may want to base your solution on the PWM function used to control the LED. The PWM output frequency can be changed using the function: `PWM.ChangeFrequency(100)`, in this example will set the output frequency to 100Hz.

Tip, don't forget to use `PWM.ChangeDutyCycle()` to set the mark/space ratio.

To use GPIO pin 9 as an input, save the code listed in figure 20 to the file input.py. If you are using gedit transfer this file to the Pi guest directory using the file browser.

When setting up an input, the default state of the input pin can be configured using the PUD_UP (set to high) or PUD_DOWN (set to low) option. These commands connect internal pull-up or pull-down resistors to the selected input pin within the Pi.

```
import RPi.GPIO as GPIO         #IO library
import time                     #Time library

GPIO.setwarnings(False)         #disable runtime warnings
GPIO.setmode(GPIO.BCM)          #use Broadcom GPIO names

                                #set pin 9 as input
                                #set default input to high
GPIO.setup(9, GPIO.IN,
           pull_up_down=GPIO.PUD_UP)

while True:                     #infinite loop
     if (GPIO.input(9)==1):
          print("BYE")          #if pin 9 high print BYE
     else:
          print("HI")           #if pin 9 low print HI
     time.sleep(1)              #wait 1 sec
```

Figure 20: Pi GPIO input code

To control the state of GPIO pin 9 a simple single-poll-single-through (SPST) switch can be connected to this pin, as shown in figure 21. This push switch can be found in the component box, in your desk drawn. To connect this switch to the bread-board you will need to use a terminal block and single core wire.

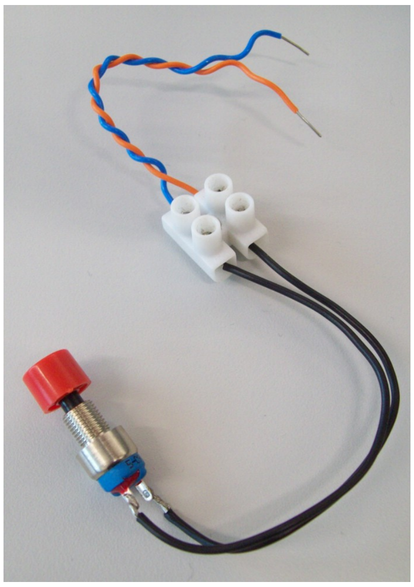
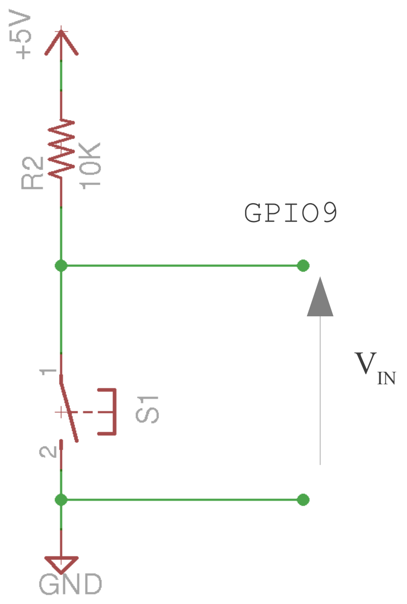



Figure 21: Pi GPIO input switch

Construct the circuit shown in figures 21 and 22. **Note**, pin 9 is next to pin 10. To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python input.py
```

When the switch is pressed, V_{IN} is connected to 0V i.e. the input pin, GPIO9 is set to a logic '0'. When released the 10KΩ resistor will pull the voltage up to a logical '1'. The Raspberry Pi will now poll this input every second, when the switch is not pressed `BYE` will be printed to the screen. When the button is pressed `HI` will be printed to the screen. This example shows the main limitation of polling i.e. if the button is pressed during the 1 second delay no message is displayed.

Note, the 10KΩ resistor is not strictly needed as the python code configures the Pi to use its internal pull-up resistor. If you remove the 10KΩ resistor the program will still work.

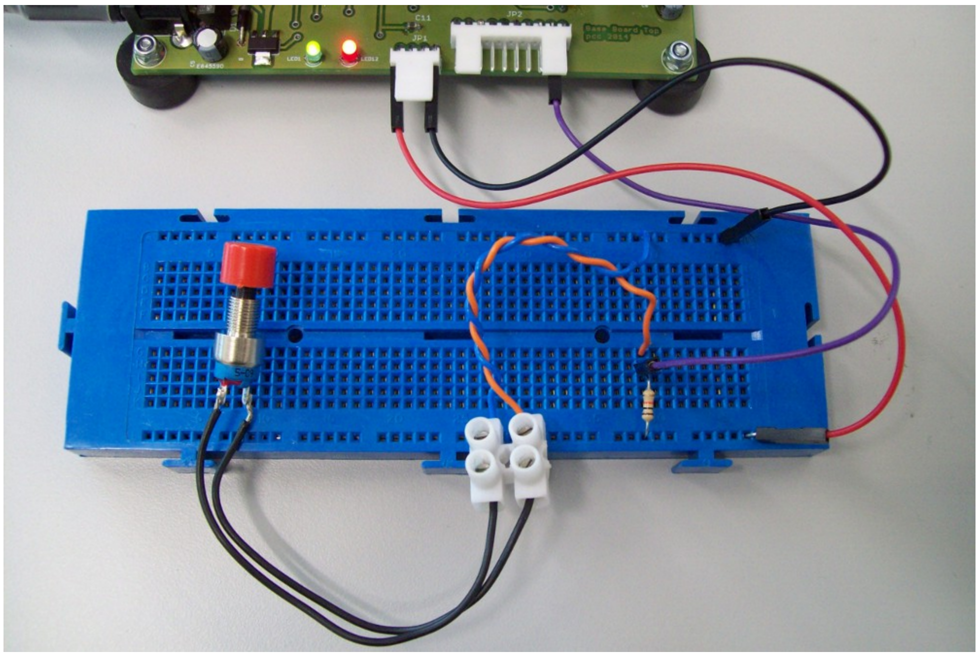


Figure 22: Pi GPIO input test system

Task 2: to overcome the problem of missed button presses increase the polling rate of the program in figure 20. Your solution should still print the message `BYE` when the switch is not pressed and `HI` if the switch was pressed during the one second delay i.e. the screen update rate should not be affected by the increased polling rate, the system should print a message to the screen every 1 second.

Task 3: write a program to produce a 500Hz tone when the switch is pressed. Then sent a message to your lab partner using Morse code, as shown in figure 23.

The python program `input.py` uses polling to read the state of the input pin. An alternative method of inputting data is to define an event i.e. an interrupt. To use GPIO pin 9 as an interrupt source save the code shown in figure 24 to the file `event.py`, then transfer this file to the Pi.

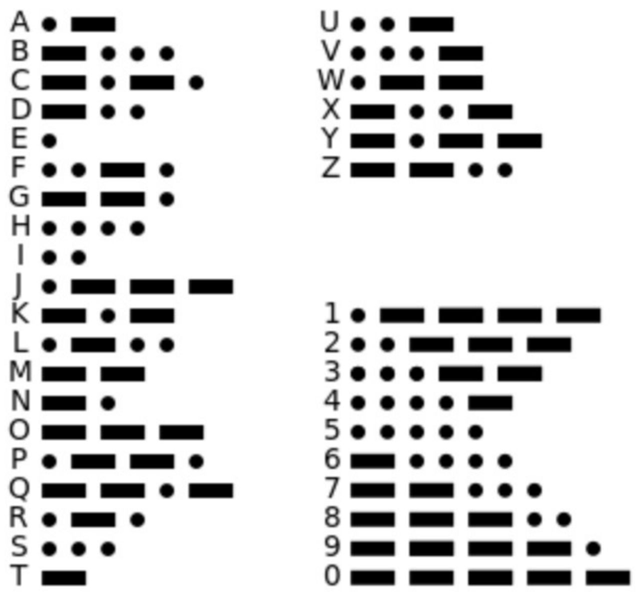

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
1
2
3
4
5
6
7
8
9
0

Figure 23: Morse code

```
import RPi.GPIO as GPIO        #IO library
import time                    #Time library

count = 0                      #set variable to 0
                               #function: count events
def pin_event(pin):
     global count
     outputString = "Event triggered : " + str( count )
     print(outputString)
     count = count +1

GPIO.setwarnings(False)        #disable runtime warnings
GPIO.setmode(GPIO.BCM)         #use Broadcom GPIO names

GPIO.setup(10, GPIO.OUT)       #set pin 10 as output
                               #set pin 9 as input, pullup

GPIO.setup(9, GPIO.IN, pull_up_down=GPIO.PUD_UP)

                               #set rising edge event

GPIO.add_event_detect(9, GPIO.RISING, callback=pin_event,
                      bouncetime=500)

while True:                    #infinite loop
     GPIO.output(10, True)     #set pin 10 high
     time.sleep(0.5)           #wait 1/2 sec
     GPIO.output(10, False)    #set pin 10 low
     time.sleep(0.5)           #wait 1/2 sec
```

Figure 24: Pi GPIO event code

This code defines an event on the input pin. Events can be defined to detect edges of type `RISING`, `FALLING` or `BOTH`. The key word `BOTH` will trigger the event on both the falling and rising edges.

To prevent electrical noise generated during the process of pressing or releasing the push switch, a delay between events can be specified i.e. `bouncetime`, defined in milliseconds.

Note, when a switch is pressed, momentum will cause its contacts to bounce i.e. repeatedly make, then break its electrical contacts for a few milliseconds. This can produce false signals. Defining a contact bounce time disables this input for a specific time, allowing the signal to settle down to its final, steady state value.

Disconnect the buzzer and reconnect the 330Ω resistor and LED to GPIO pin 10, as shown in figure 14. To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python event.py
```

The LED connected to the output pin GPIO 10 should now flash ON for half a second and OFF for half a second. The main loop controlling the LED can be interrupted by a low to high transition on the input line GPIO 9. For each event a message is printed to the terminal window. **Note**, the parameter `pin` passed to the function `pi_event` is the GPIO pin number that triggered the event.

Modify the program to use `bouncetime=1` and `BOTH` event options. You should observe multiple count values are recorded each time the switch is pressed. When is the event message printed to the terminal window? Why do these options cause a problem?

<u>**Task 4:**</u> restructure your Morse code program to use interrupts rather than polling. **Hint**, use a variable (flag) to pass control signals between the interrupt service routine and the main program. Again, you may want to consider using the PWM functions.

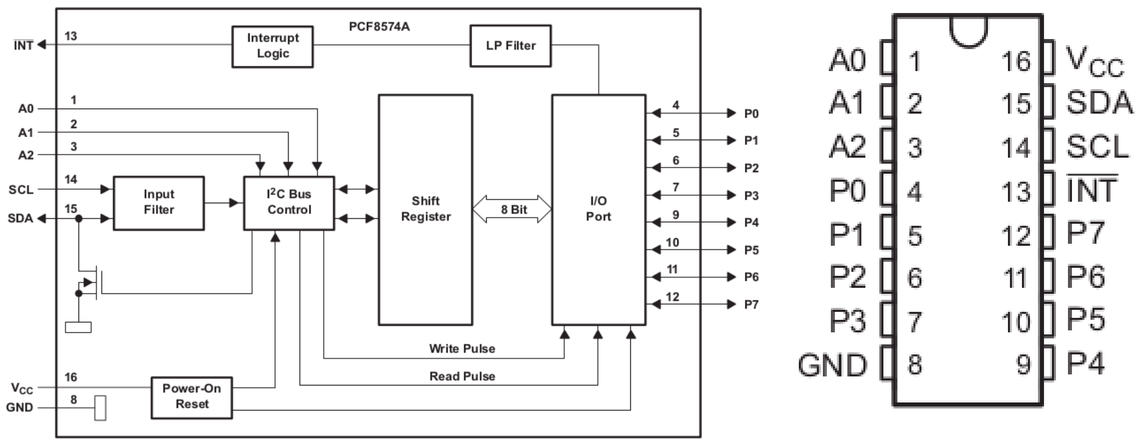


Figure 25: PCF8574 IO expander

Inter-Integrated Communications (I2C)

To expand the number of general purpose input / output lines available on the Pi the I2C serial bus can be used. The Raspberry Pi is a Master device, having a dedicated hardware I2C controller, transmitting or receiving data from one or more Slave devices. The I2C bus only uses two wires; serial data (SDA) and serial clock (SCL), which are accessible on the Pi using pins 3 and 5, or from the base, as shown in figures 11 and 13. All I2C master and slave devices are connected to the same two wires. In order to communicate with a specific slave device, each device must have an unique 7 bit address, as shown in figure 26.

Note, for more information on the I2C bus refer to the specification document on the module web page. If you are using the I2C bus you can not use pins 2 and 3 as GPIO.

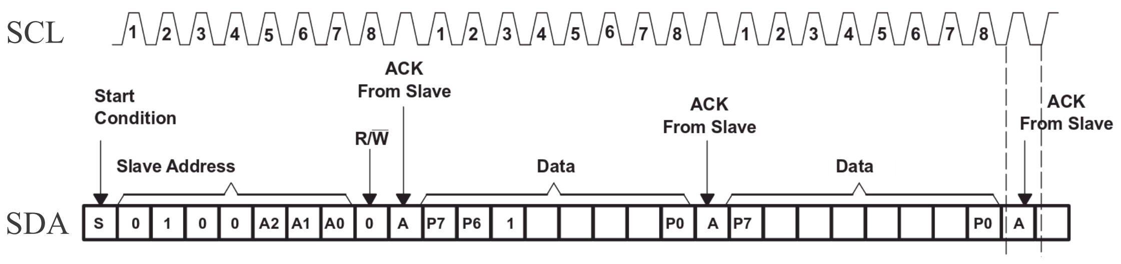


Figure 26: I2C communications (two byte write)

To illustrate the process of reading and writing data across this bus we will use the PCF8574 8bit IO expander, as shown in figure 25. This is one of the simplest I2C devices available having eight pins P0 – P7 (physical pins 4 – 7 and 9 – 12). These can be configured to be either an input or an output. Each type of I2C device is hard coded with a base address at manufacture, however, this value can be modified by tying the pins A0 – A2 to either a logical 1 or 0 e.g. if the device's base address is 0x38, setting A0 = A1 = A2 = 1 would change the address to 0x3F.

The I2C 8bit IO expander in your box is a PCF8574**N**, having the base address 0x20, as shown in figure 27. An equivalent device is also produced with a different base address: 0x38 i.e. a PCF8574**AN**, as shown in figure 28. Both are stocked and used in the laboratory so do double check (they are exactly the same except for the address).

BYTE	**BIT**							
	7 (MSB)	**6**	**5**	**4**	**3**	**2**	**1**	**0 (LSB)**
I^2C slave address	L	H	L	L	A2	A1	A0	R/$\overline{W}$
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0

Figure 27: PCF8574N address and data formats

BYTE	**BIT**							
	7 (MSB)	**6**	**5**	**4**	**3**	**2**	**1**	**0 (LSB)**
I^2C slave address	L	H	H	H	A2	A1	A0	R/$\overline{W}$
I/O data bus	P7	P6	P5	P4	P3	P2	P1	P0

Figure 28: PCF8574AN address and data formats

Construct the circuit shown in figure 29. Next, connect the Raspberry Pi I2C bus to the PCF8574 as shown in figure 30. Connecting wires, components and bread boards are available from the side desk, or front component cabinets.

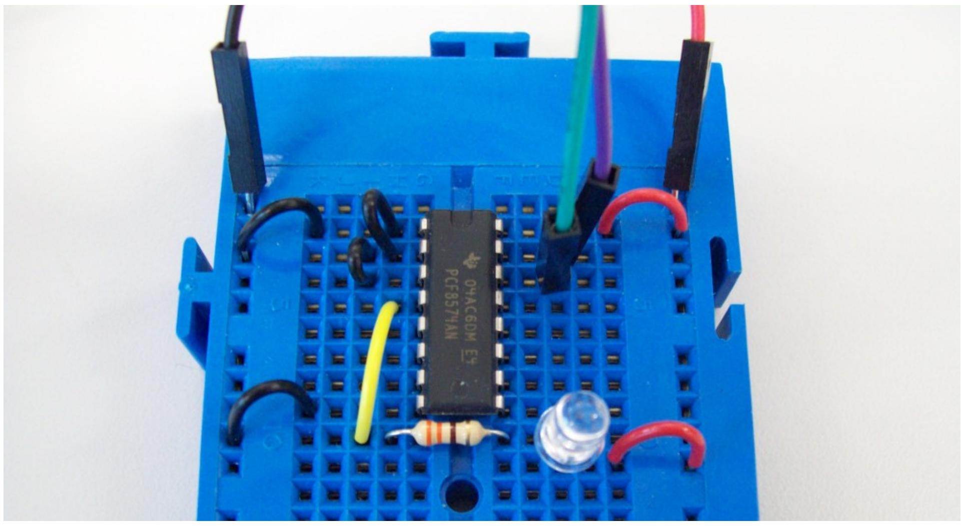


Figure 29: PCF8574 output port (**PURPLE** wire pin 2, **GREEN** wire pin 3)

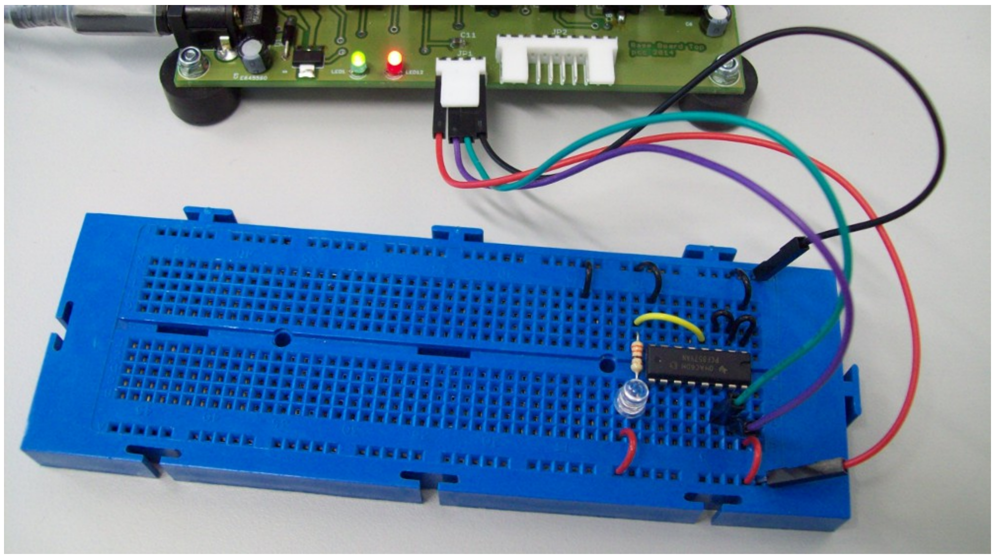


Figure 30: PCF8574 output test system

To confirm the base addresses of the I2C devices attached to the Raspberry Pi, within the command terminal running `SSH`, type:

```
i2cdetect -y 1
```

This will scan the I2C address range from 0x00 to 0xFF. If no device is present at an address `--` will be displayed in that address position. The Raspberry Pi system already contains two I2C devices at address 0x21(ADC) and 0x24 (Parallel Port).

Note, these addresses may vary as they are configured using jumpers on the Raspberry Pi PCB. However, with the PCF8574 removed two addresses will be listed.

If these are not listed then disconnect the bread-board and try again. Re-run the command, if they now appear there is a fault on your bread-board e.g. not powered, SDA and SCL wrong way round etc.

To change the values on pins P0 – P7 use your preferred text editor to save the code listed in figure 31 to the file `i2c_write.py`. If you are using `gedit` transfer this file to the Pi guest directory using the file browser.

```
import smbus                        #I2C library
import time                         #Time library

I2C_ADDR = 0x20                     #I2C base address
LED_ON = 0x00                       #value need to turn
LED_OFF = 0xFF                      #LEDS on / off

bus = smbus.SMBus(1)                #enable I2C bus

while True:                                 #infinite loop
     bus.write_byte( I2C_ADDR, LED_ON )     #set port to 0
     time.sleep(1)                          #wait 1 sec
     bus.write_byte( I2C_ADDR, LED_OFF )    #set port to 1
     time.sleep(1)                          #wait 1 sec
```

Figure 31: I2C write code

To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python i2c_write.py
```

The LED connected to this I2C chip should now flash ON for a second and OFF for a second. The anode of the LED is connected to +3.3V, therefore a '0' will turn it on i.e. the opposite voltage compared to the previous LED.

Note, if this program exits with an exception the Raspberry Pi was unable to communicate with a device at the specified I2C_ADDR i.e. did not receive back an acknowledgement (ACK). Double check the hardware and base address.

Figure 32 shows the internal hardware used for each pin. The 8574 pins are described as “quasi-bidirectional”. To use a pin as an input you will first need to set the pin to a logic '1'. As the output current for each pin is limited to 100μA this pin can be connected to 0V to enter a logic '0', or connected to +3.3V to enter a logic '1' i.e. the 100μA current source is acting as a pull-up resistor.

Note, this is only possible owing to the hardware used. Normally connecting an output to 0V would damage the device. Do **NOT** do this to the Pi GPIO outputs.

Connect one terminal (wire) of the push switch to `P4` (physical pin 9) on the PCF8574, the other to zero volts, as shown in figure 33.

To change pins P0 – P7 to inputs use your preferred text editor to save the code listed in figure 34 to the file `i2c_read.py`. If you are using `gedit` transfer this file to the Pi guest directory using the file browser.

To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python i2c_read.py
```

When the switch is not pressed the displayed value will be 255 (0xFF). When the switch is pressed the displayed value will be 239 (0xEF). Before proceeding make sure you understand why.

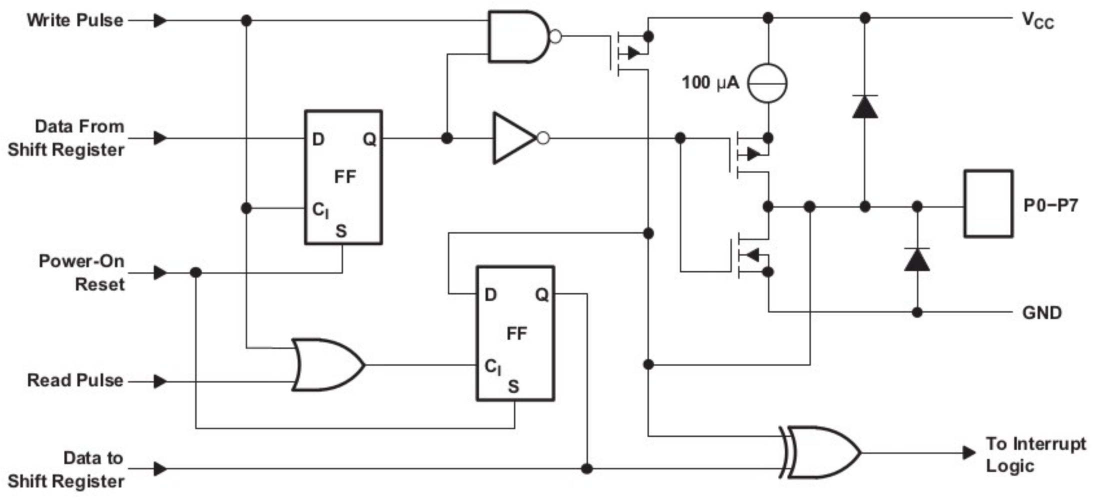


Figure 32: PCF8574A internal hardware

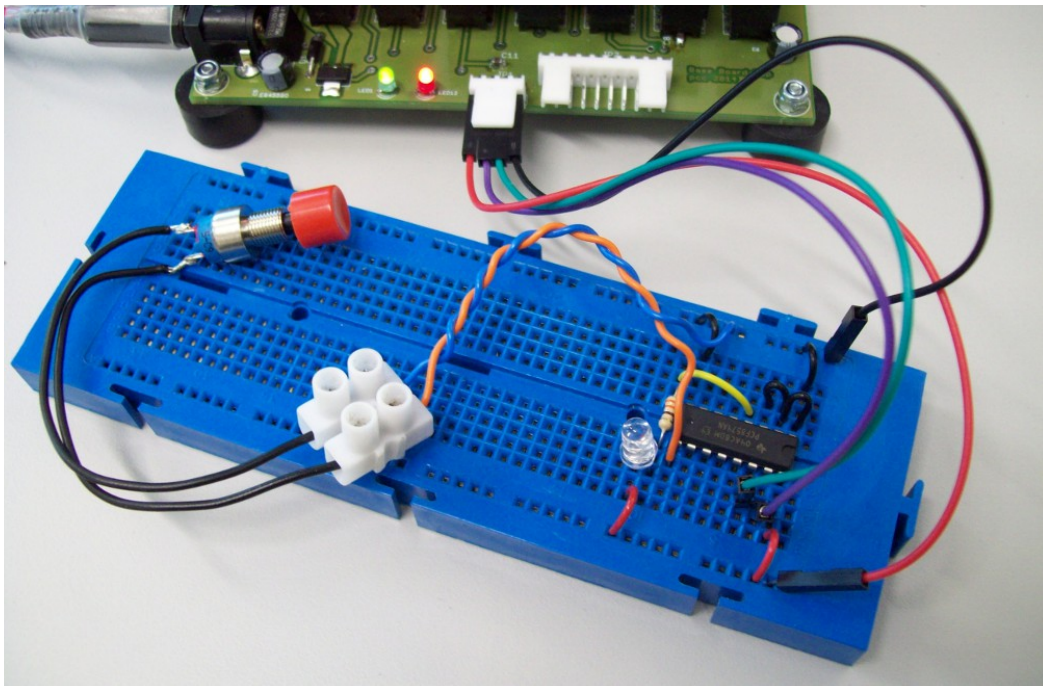


Figure 33: I2C switch

```
import smbus                        #I2C library
import time                         #Time library

I2C_ADDR = 0x20                     #I2C base address
PORT_ON = 0xFF

bus = smbus.SMBus(1)                #enable I2C bus
                                    #set Port to 1s to
                                    #allow inputs

bus.write_byte( I2C_ADDR, PORT_ON )

while True:                         #infinite loop
                                    #read I2C address

     i2cvalue = bus.read_byte( I2C_ADDR )

                                    #generate string

     outputString = "INPUT = " + str( i2cvalue )

     print( outputString )          #print string
     time.sleep(1)                  #wait 1 sec
```

Figure 34: I2C read code

Task 5: using the I2C read and write functions, write a program to read port `P4` (physical pin 9) and output its state i.e. '0' or '1', to port `P0` (physical pin 4, connected to the LED). You are not allowed to use IF statements, however, you should be able to define this functionality in a single line of code.

Hint, this task is a little tricky. I would **NOT** recommend attempting to solve these types of problems using character string manipulation i.e. numbers are not characters. You will need to use the bitwise logic functions available in python i.e. `&`, `|`, and >> . Examples:

```
~1 = -2             NOT 0001 = 1110

5 & 2 = 0      0101 AND 0010 = 0000
7 & 2 = 2      0111 AND 0010 = 0010
5 | 2 = 7      0101 OR  0010 = 0111
7 | 2 = 7      0111 OR  0010 = 0111
0 ^ 2 = 2      0000 XOR 0010 = 0010
1 ^ 2 = 3      0001 XOR 0010 = 0011
5 >> 2 = 1     0101 Right Shift 2 = 0001
7 >> 3 = 0     0111 Right Shift 3 = 0000
5 << 2 = 20    0101 Left  Shift 2 = 10100
7 << 3 = 56    0111 Left  Shift 3 = 111000

DATA=0x55
DATA=((DATA & 0xF0) | 1) << 2 = 1010000100 = 324 = 0x144
```

For more details refer to :

https://wiki.python.org/moin/BitwiseOperators

IMPORTANT: you will need to use these types of operators in the open assessment, make sure you understand these examples.

Task 6: can you write a python function to individually control each pin on the PCF8574A e.g. `setpin(5)` or `clearpin(5)`, where 5 is an output pin, numbered 0-7. When an output is changed all other pins must remain the same. **Hint**, you will need to read the output port's pin status, perform the appropriate logical operator, then write this value back to the port.

More complex I2C devices contain multiple internal registers e.g. the SN3218 is a 18 channel LED controller, used on the piGlow shown in figure 35. This device contains 23 internal registers and is assigned the I2C address 0x54. Each LED can individually be turned on or off and its brightness controlled using pulse width modulation (PWM), as shown in figure 36. For more information refer to the SN3218 data sheet on the module web page.

To read or write data in a device containing multiple registers different python functions must be used:

```
data = bus.read_byte_data( I2C_ADDR, I2C_REG )
bus.write_byte_data( I2C_ADDR, I2C_REG, I2C_DATA )
```

To write the value 0x07 to register 0x12 on device 0x54 the following function can be used:

```
bus.write_byte_data( 0x54, 0x12, 0x07 )
```

Copy the files `PyGlow.py` and `test.py` from the module web page onto the Pi. To execute this program on the Pi, within the command terminal running `SSH`, type:

```
python test.py
```

At the prompt enter the value 50 for each colour. When prompted to enter a value for `ALL` enter the value 0 to turn off the LEDS. Next, re-run the program using the values 150, 200 and 255. Ensure you turn off the LEDS at the end of each test.

Examine the data sheet and the above python programs to see how this hardware is controlled using the I2C bus.

Task 7: Write a program to turn on the white LEDs. Then using the PWM control registers incrementally increase and decrease the brightness of these LEDs.

IMPORTANT: in the open assessment you may be given an existing python library, code or other I2C devices, you will then have to integrate these components into your system.

To log off the local computer click on the log off button in the top right of the screen, as shown in figure 37.

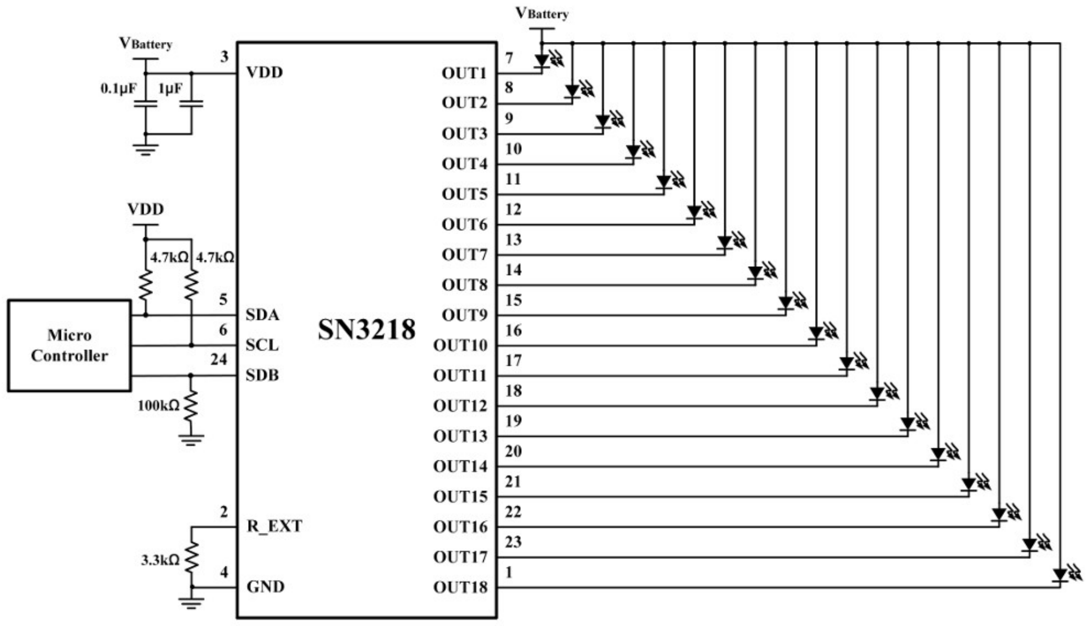


Figure 35: SN3218 PiGlow board controller

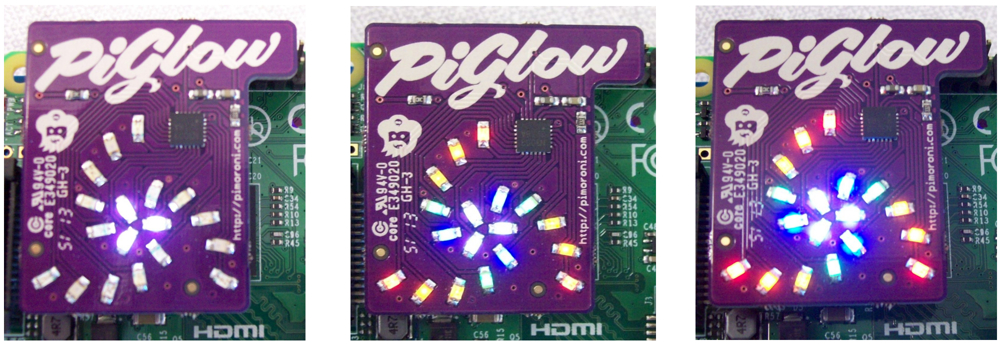


Figure 36: examples of LED control

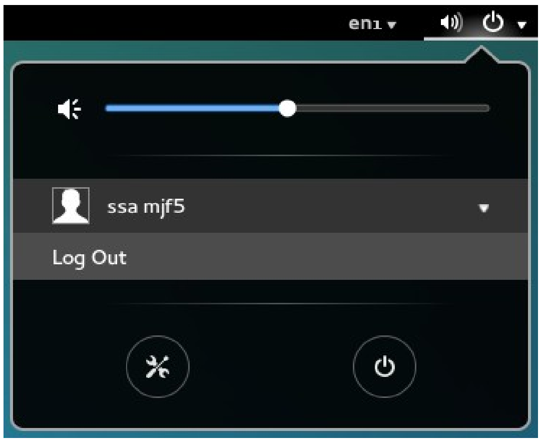


Figure 37: Logging off