Cohort Session 3, Week 3
Prelab-Activity

Objectives

1. Assemble Raspberry Pi.
2. Get familiar with the Raspbian OS.
3. Learning the basics of the General Purpose Input/Output (GPIO).

Please work on this prelab-activty in a group of **five**. Be sure to email your partner all the modified code, printouts and data. You may have to use them during your exams.

1 Equipment & Software

Each group should have:

1. A Raspberry Pi.
2. A cobbler.
3. An LCD touch screen.
4. A wireless keyboard.
5. A wireless mouse.
6. An LED.
7. A switch.
8. A Resistor (330Ω).
9. A few jumper wires.
10. `raspberrypi_sample.py`, which contains a sample code to control an LED with a switch.

2 Assembling the Raspberry Pi

Task
Assemble the Raspberry Pi.

Instructions:

1. Follow the following steps:
   a. Connect the Cobbler to the Raspberry Pi and a breadboard. You may find the diagrams in **GPIO Pins and Breadboard Layout** to be useful when working with the breadboard.

   b. Connect the various peripherals (WiFi & Bluetooth dongle, mouse and keyboard).

   WARNING!
   Do not power up the Raspberry Pi yet! You should complete the rest of the following steps first.

2. If you have already connected wires to provide a 5V supply to the touch screen, remove them for now. We need to verify that the Cobbler connection is correct before we power up the touch screen. Otherwise, it is possible to damage the touch screen.
3. Verify that the Cobbler connection is correct. First, wire up an LED and a resistor as shown.

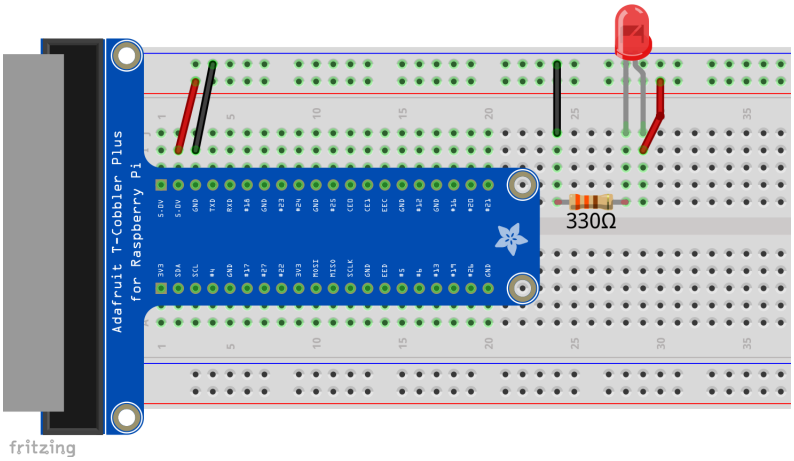


Figure 1: Note that the longer leg of the LED is the positive end. Hence it should be connected to the 5V.

4. When the Raspberry Pi has been powered up, the LED should light up. This indicates that Cobbler connection is correct. If it does not light up, you may have connected the Cobbler connector at the Raspberry Pi end wrongly. Remove the connector from the Raspberry Pi, flip it, and reconnect. If this does not help, you should approach an instructor.
5. You may remove the LED and resistor once the testing is complete.

3 Booting into the Raspberry Pi

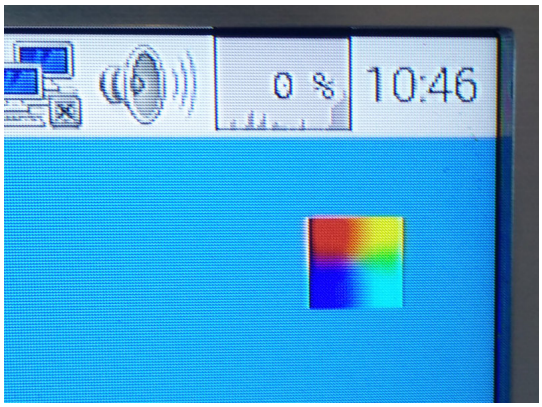
Task
Boot into the Raspberry Pi Operating System.

This section describes two methods to use the Raspberry Pi. The first method allows you to work with the Raspberry Pi software directly with a touch screen display along with the wireless keyboard and mouse. The second method allows you to work from your own PC by controlling the Raspberry Pi desktop remotely from your PC.

3.1 Method One: Direct usage of Raspberry Pi

Connect the touch screen display and power up the system. Once you have powered up the system, you should be able to see the the booting up process on the touch screen. This may take awhile to complete.

Note
If you see a rainbow square at the top right hand corner of the screen, it means that the Raspberry Pi is not receiving enough power. This may cause unexpected behaviour such as random movement of the mouse cursor or unreliable WiFi connection.



Generally, such a situation is caused by:

1. Powering the Raspberry Pi from a laptop/PC USB port
2. USB peripherals that takes a considerable amount of power
3. Inadequate power supply
4. Low quality microUSB cable

It is recommended to use the power supply and microUSB cable provided to power up the Raspberry Pi.

3.2 Method Two: Remote Destop

This method allows you to work with the Raspberry Pi software remotely from your own PC. As such, it is not necessary to have the touch screen display connected to the Raspberry Pi.

Instructions:

1. Power up the Raspberry Pi.
2. Follow the video tutorials on **Raspberry Pi Remote Desktop** to set up your PC to establish a remote desktop connection with the Raspberry Pi.
3. The vnc client will prompt you for a password for login. The password is: raspberry

> Note
> If you have the touch screen display connected and powered up, you may notice that the behaviour of the remote desktop of the Raspberry Pi on your PC does not correspond to what is shown on the touch screen. This is because the remote desktop is a separate desktop session. However, modifying things (such as renaming a file on the desktop) on the remote desktop would also cause changes to appear on the touch screen.

4 Navigating in Raspbian OS

> Task
> Get familiar with the Raspbian OS and run a Python program on it.

Instructions:

1. Follow the video tutorial on **Getting Started with Raspberry Pi** to:
   a. Use the web browser to download the software required for the project in this week.
   b. Use the file manager to navigate through the directories in the system.
   c. Run a python program.

5 GPIO Basics

> Task
> Understand and test the `raspberrypi_sample.py` code on a breadboard with an LED and switch.

Open the `raspberrypi_sample.py` file in your Raspberry Pi.

```
1 import RPi.GPIO as GPIO
```

The first two lines imports the necessary modules so that we can make use of the sleep function and manipulate the GPIOs on the Raspberry Pi.

```
4 GPIO.setmode(GPIO.BCM)
5
6 # Use GPIO23 for LED and GPIO18 for switch
7 led = 23
8 switch = 18
```

There are two types of numbering scheme. The Broadcom GPIO numbers (BCM) or the pin numbers (BOARD).

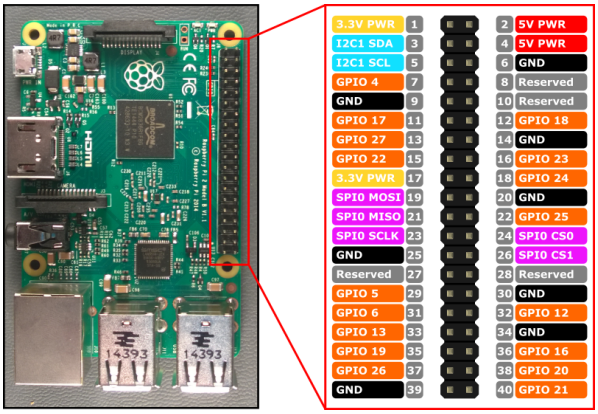


Figure 2: GPIO layout of Raspberry Pi showing the two numbering schemes.

The pin numbers are those in the grey boxes as shown in Figure 2. For example, pin 1 is the pin at the top left corner (3.3V PWR) and pin 40 is the one at the bottom right corner (GPIO21). On the other hand, the BCM numbers are those in the orange boxes. GPIO21 would correspond to pin 40, and GPIO4 would correspond to pin 7.

Line 5 sets the numbering scheme to be the BCM numbers. As such, the numbers in line 8 and 9 refer to GPIO23 and GPIO18 rather than pin 23 and pin 18.

We will be using the BCM numbers in this sample code, since it is easy to identify the number from the labelling on the Cobbler. However, if you would like to use the pin numbers, you can replace line 4 and 5 with:

```
# Use the pin numbers as the numbering scheme
GPIO.setmode(GPIO.BOARD)
```

Furthermore, you may want to change line 7-9 to:

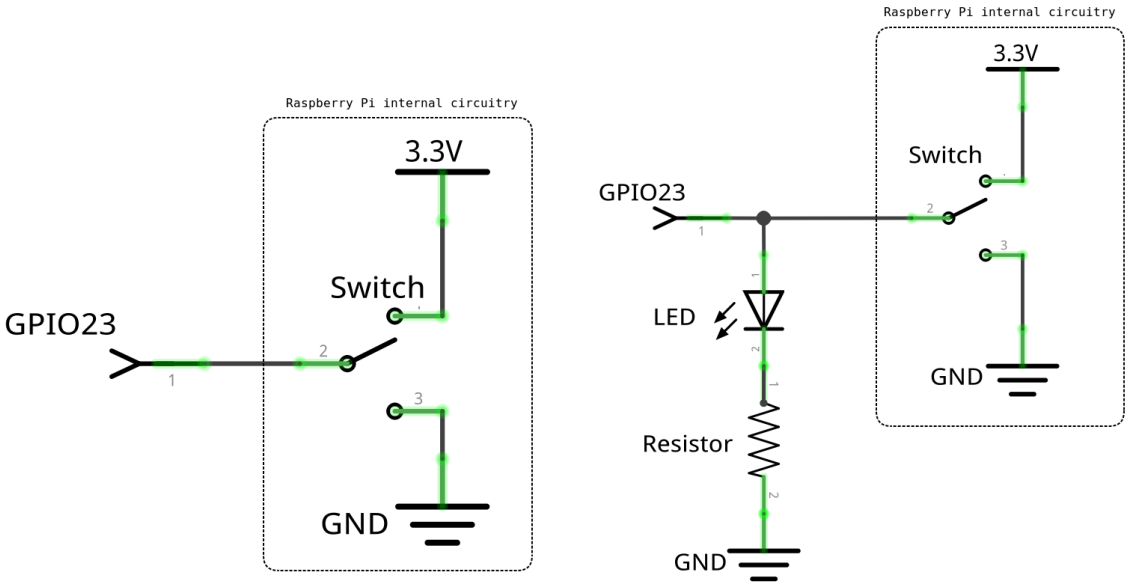
```
# Use pin 16 for LED and pin 12 for switch
led = 16
switch = 12
```

By using the pin number scheme and changing line 8-9 to use the updated pins, you would not have to change any physical wiring on the breadboard if you have already wired the components.

Unless you are aware of what you are doing, we would recommend that you leave the sample code as it is and use the default BCM number scheme so that you can follow along the rest of this guide without any confusion.

```
11 GPIO.setup(led, GPIO.OUT)
```

The `GPIO.OUT` in line 12 tells the function that we want to set GPIO23 as an output. Setting it as an output allows us to manipulate the port to be HIGH (3.3V) or LOW (0V). This would be equivalent to the following circuit.



(a) A GPIO output without any connections.

(b) A GPIO output with an LED and resistor connected.

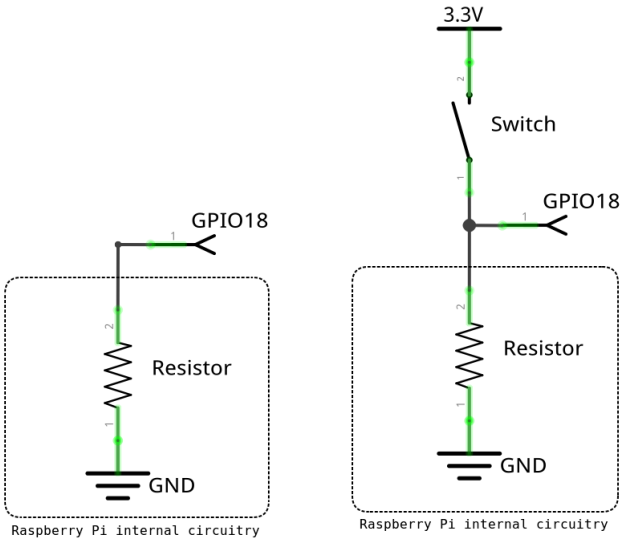
Figure 3: Schematic of a GPIO set as output.

Once we set GPIO23 as an output, we can connect an LED and a resistor in series from the GPIO23 to the ground. Setting GPIO23 to HIGH would be equivalent to toggling the switch so that the GPIO23 is connected to 3.3V. This would light up the LED. If GPIO23 is LOW, then it would be connected to ground, and the LED would not light up.

```
14 GPIO.setup(switch, GPIO.IN, GPIO.PUD_DOWN)
```

The `GPIO.IN` in line 15 specifies that the GPIO18 behave as an input. Furthermore, `GPIO.PUD_DOWN` would connect a resistor that pulls GPIO18 to ground. Hence the name

pull-down resistor. This is equivalent to figure 4(a). Setting GPIO18 as an input allows us to determine whether it is HIGH or LOW. By default, it is pulled to ground, hence it is LOW. We can add a switch to GPIO18 that has 3.3V connected to the other end of the switch as shown in figure 4(b). Toggling this switch such that it forms a close circuit would cause GPIO18 to be connected directly to 3.3V. As such, it would be HIGH. We can put this signal into good use as we shall see in the next few lines of code.

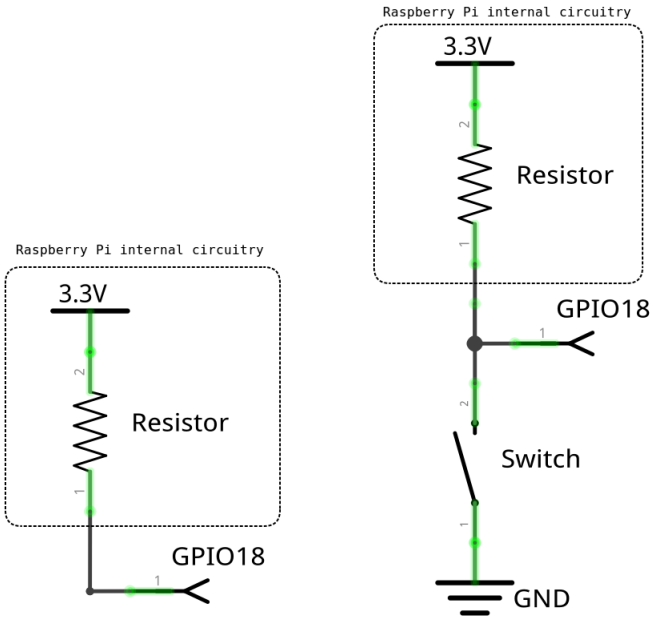


(a) A GPIO input without any connections.

(b) A GPIO input with a switch.

Figure 4: Schematic of a GPIO set as input with pull-down resistor.

Another input option is to use the pull-up resistor instead of the pull-down resistor. Using a pull-up resistor would result in an internal (in Raspberry Pi) configuration as shown in figure 5(a). The working principle is similar to the pull-down counter part, except that everything would be reversed. By default, the signal at GPIO18 would be HIGH since it is pulled up to 3.3V by the resistor. If we connect a switch as shown in figure 5(b), we can close the switch so that GPIO18 would be directly connected to ground, causing the signal at GPIO18 to be LOW.



(a) A GPIO input without any connections.

(b) A GPIO input with a switch.

Figure 5: Schematic of a GPIO set as input with pull-up resistor.

```
17      if GPIO.input(switch) == GPIO.HIGH:  # switch is connected
18          GPIO.output(led, GPIO.HIGH)  # turn on the LED
19      else:  # switch is disconnected
20          GPIO.output(led, GPIO.LOW)  # turn off the LED
```

This is the part of the code that does some useful work. Recall that we have assigned the variable `switch` to be GPIO18 and led to be GPIO23. Furthermore, we have also set the GPIO18 to be an input with a pull-down resistor at line 15 and set the GPIO23 to be an output at line 12.

In line 18, `GPIO.input(switch)` returns the signal of GPIO18. Referring back to figure 4(b), if we close the switch, the signal at GPIO18 should be HIGH. When the switch is open, the signal should be LOW. We use `GPIO.HIGH` to refer to a HIGH signal and `GPIO.LOW` to refer to LOW. Hence line 18 is essentially checking whether the switch is closed. If the switch is closed, line 19 would be executed. Otherwise, line 21 would be executed.

Line 19 and 21 sets GPIO23 to be HIGH and LOW respectively. Recall that setting GPIO23 to be HIGH or LOW is equivalent to toggling the internal switch in figure 3(b) to connect GPIO23 to 3.3V or ground respectively. Hence, when setting GPIO23 to HIGH, the led that is connected to GPIO23 should light up. Conversely, the led should switch off when we set GPIO23 to LOW.

The checking of the switch's status is placed in a while True block. Since the expression True is always true, the statements in the while loop are always executed. We want this behaviour so that the program would continuously check the status of the switch. Suppose line 17 is omitted, the program would only run line 18-21 only once. This mean that if you toggle the switch any time after the program has executed line 18-21, nothing would happen.

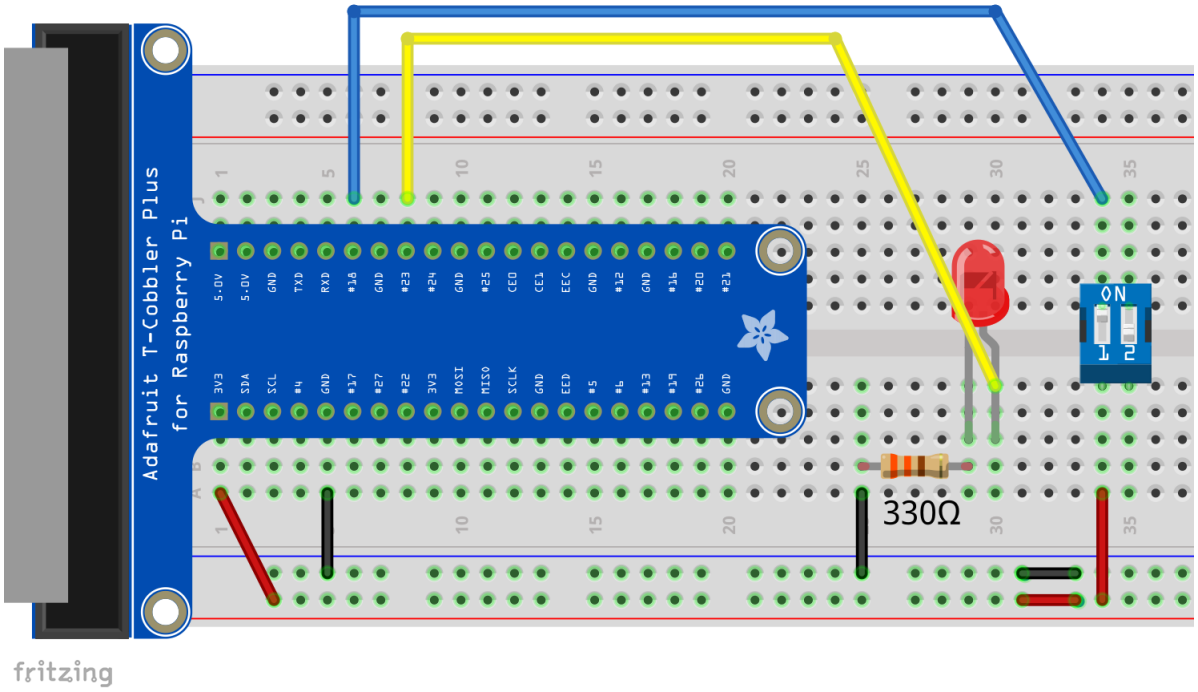


Figure 6: Breadboard diagram showing how the connection should be done. Note that the longer leg of the LED is the positive end. Hence it should be connected to the GPIO pin.

If you feel that you have understood the sample code, you may refer to the diagram in figure 6 to wire up all the components necessary to test out this sample code. You may find the diagrams in **GPIO Pins and Breadboard Layout** to be useful when working with the breadboard. Once the wiring is done, run the python program (if you do not know how to run a python program in the Raspberry Pi, refer to the video tutorial mentioned in previous section). Try toggling the switch and verify that the LED lights up when the switch is closed and that the LED turns off when the switch is opened.

WARNING!
The state of the GPIOs will be retained after the program terminates. That is, if any GPIO output was HIGH at the moment the program terminates, it would remain as HIGH even after the program terminates.

Consequently, it is possible to accidentally damage the Raspberry Pi by connecting GPIOs that was HIGH directly to ground. In order to 'reset' the GPIOs, go to IDLE3 in your raspberry pi. Type the following into the shell.

```
>>> import RPi.GPIO as GPIO
>>> GPIO.cleanup()
```

The program sets every GPIO as input. As such, they would not get damaged even if they are connected directly to either HIGH or LOW.

6 Something to Try (Optional)

Task
Experiment with different set ups and gain a deeper insight to the fundamentals of GPIO.

Instructions:

1. Use the pin numbers scheme instead of the BCM. What happens when you try to use pin 1 or 2?
2. Change the input configuration to use a pull-up resistor. What are the changes to be made in the code? What are the changes to be made on the breadboard?