**Omarichet Satellite Learning Platform**

**Sensor Unit System Development with the Raspberry Pi**

**1.0 Introduction**

The raspberry pi offers a powerful processing that rivals personal computers. It supplies interfaces from which an individual may connect other hardware modules and one may also have to write code to allow use of the newly added peripheral devices. The main peripheral interfaces include USB, HDMI, Ethernet, display and camera serial interfaces. Another set of interfaces fall under the General-Purpose Input/Output interfaces which is a set of programmable digital pins from which the raspberry pi may receive digital input and send out outputs. The digital input often may receive data from sensors. This will be a critical capability that the platform will require.

**1.1 Special considerations while working with GPIOs on a Raspberry pi**

Despite being a powerful device, the raspberry pi has limits within which one must work within to ensure safe and efficient operation. These limits are:

* The raspberry pi makes use of 3.3 Volt logic. This means that a high on the digital pins is not more than 3.3V or else there is a risk of damage. Thus, connections especially inputs must be 3.3V this includes powering the sensors with 3.3V if they allow. If not, regulation must be carried out and a simple voltage divider may suffice.
* The raspberry pi has no analogue read function on its pins and hence limits one to digital sensors only. One way to go around this is interfacing with an ADC. This can quickly use up pins. The other method is using an analogue-capable microcontroller like the Arduino uno and sending the data to the Raspberry Pi via UART. The latter method will be explored later.

**2.0 Sensors**

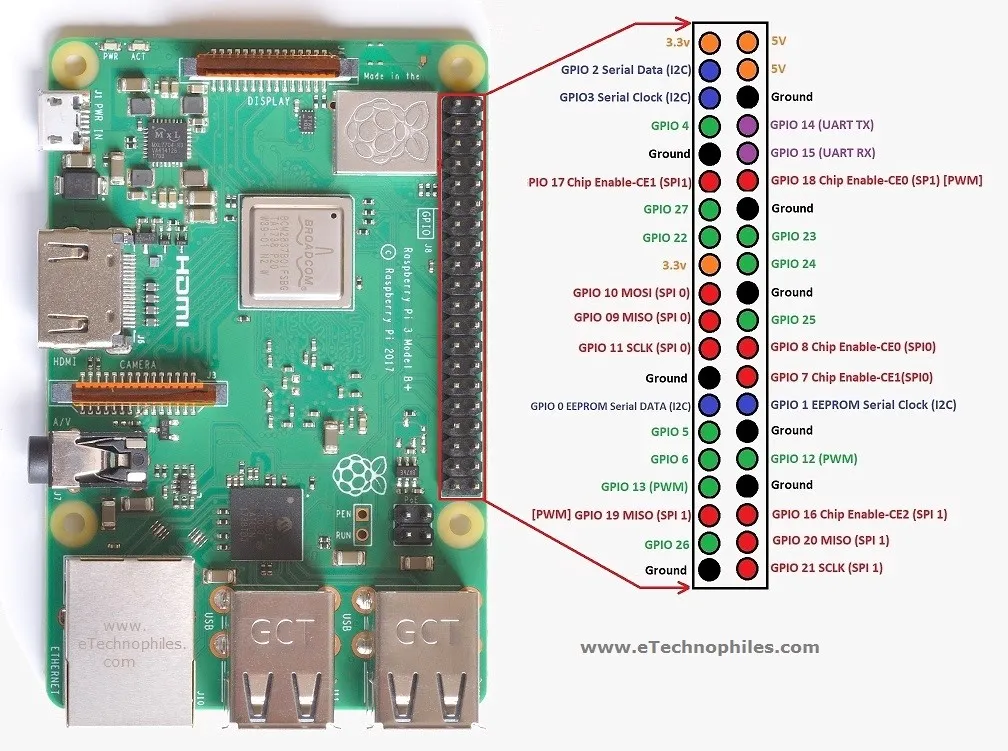
These are devices that will produce an output signal proportional to some physical quantity. They will enable our system to interact with the physical environment. The physical quantities that will be under test in our platform will include temperature, displacement, velocity, orientation, acceleration, luminosity, voltage and current from the power supply. Respective sensors are chosen that will effectively measure the above and they are as follows.

* DHT sensor – A digital sensor that measures temperature and humidity.
* MPU-6050 – A digital sensor that has an accelerometer and a gyroscope. It can measure displacement, velocity, orientation, acceleration and gravitational pull.
* HMC5883L – This is a digital magnetometer and may be used to measure magnetic strength and field and use this to determine orientation. Often combined with the MPU6050 to achieve 9DOF.
* Photoresistor?Light Dependent Resistor – This is an analogue sensor whose resistance varies with luminosity or ambient light.
* GPS module – A digital module that provides location and timing information.
* BME280 – This is a digital sensor that measures absolute barometric pressure.
* BC25 – An analogue Voltage sensor
* ACS712 – An analogue Current Sensor

The digital sensors send out data over different protocols which we will be discussed for every sensor.

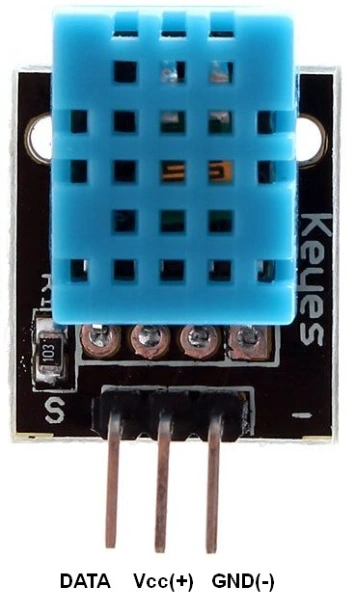
**2.1 Digital Sensor interfacing with the Raspberry Pi**

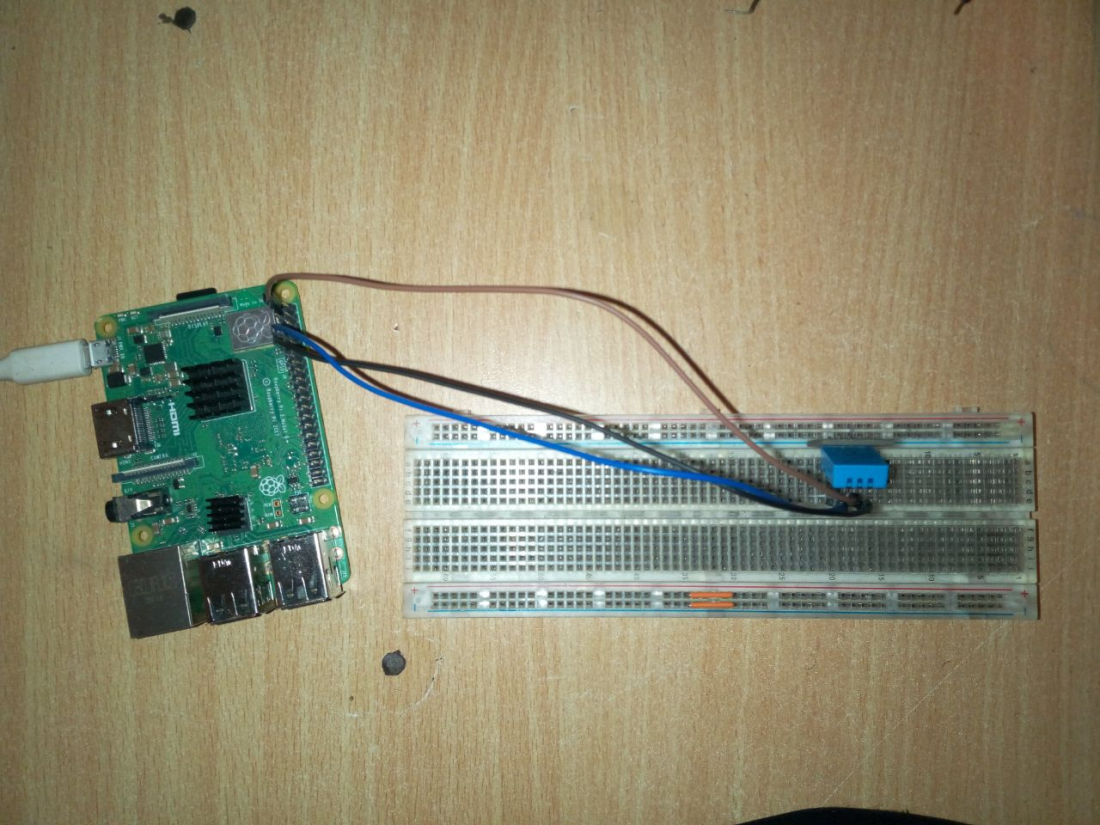
Most sensors work with the 3.3V or 5V logic state. This means that the digital pins record highs of up to 3.3 or 5V. The raspberry pi’s GPIO’s are not 5V tolerant. To interface sensors with the raspberry pi, a voltage divider may be considered as a method of voltage regulation. This will not affect the underlying protocol whether UART or SPI etc. A voltage divider that will drop the voltage by a third i.e., from 5V to 3.3V maybe used. Alternatively, the sensor may be powered from 3.3V if it allows. The raspberry Pi GPIO pin configuration is as shown below.



**2.1.1 DHT sensor**

This is a digital sensor that can be powered with 3.3V or 5V power. Connection will be made with the raspberry pi’s 3.3V supply, ground and GPIO 4 pins.





**Library installation**

To use python to access data from the DHT sensor, we install the Adafruit DHT library using the following command on the terminal:

git clone <https://github.com/adafruit/Adafruit_Python_DHT.git>

This is installed in a folder called Adafruit\_Python\_DHT. Go into the folder and run the following commands to install python within.

sudo python3 setup.py install

Or

sudo python setup.py install

The GPIO is confirmed to be enabled using

sudo raspi-config

Some extra process will be required to make the Adafruit libraries compatible with RPI4

**DHT Code**

The following code is written to get temperature and humidity data from the sensor.

**import** Adafruit\_DHT

sensor **=** Adafruit\_DHT.DHT11

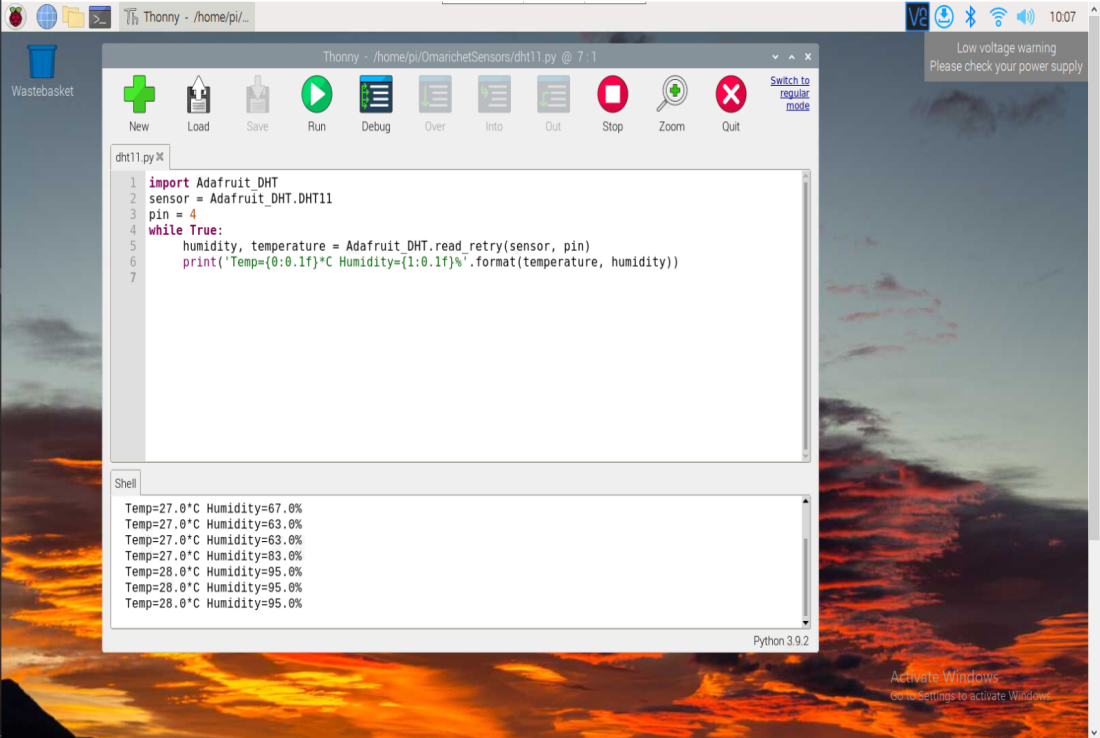
pin **=** 4

**while** True:

humidity, temperature **=** Adafruit\_DHT.read\_retry(sensor, pin)

print('Temp={0:0.1f}\*C Humidity={1:0.1f}%'.format(temperature, humidity))

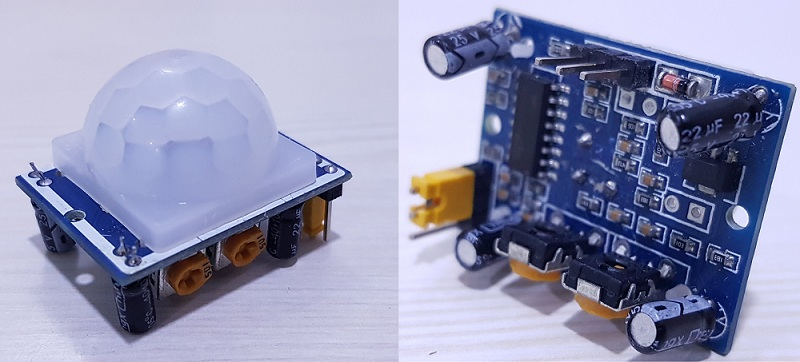
**DHT Output**



From the image above, the temperature and humidity were recorded and were seen to vary when a steaming cup of coffee were introduced next to the system. The code ran in an infinite loop.

**2.1.2 PIR sensor**

This sensor provides a means to detect human movement by detecting changes in radiation emitted by humans or other infrared emitting bodies. It can detect to a distance up to 8 metres. Once a human is sensed, the output pin remains high for a duration of time. This makes the PIR a digital sensor. It has 3 pins, a VCC power pin, ground pin and an output pin. The PIR will be powered via the 3.3V power supply and the output connected to pin 4.



**Library Installation**

No special libraries are required to get data from the PIR as it is merely a digital signal. However, the GPIO library will be required to read from the digital pin. This library is available natively in the raspberry pi.

**PIR code**

The PIR script basically sets GPIO 4 as an input and reads the state of the pin whether high in the presence of an infra-red emitting source or low in the absence. A message is then printed in either of the case.

import RPi.GPIO as GPIO

import time

pir = 4

GPIO.setmode(GPIO.BCM)

GPIO.setup(pir, GPIO.IN)

print("Initializing PIR...")

time.sleep(12)

print("Ready...")

try:

while True:

if GPIO.input(pir):

print("Motion Detected")

time.sleep(2)

else:

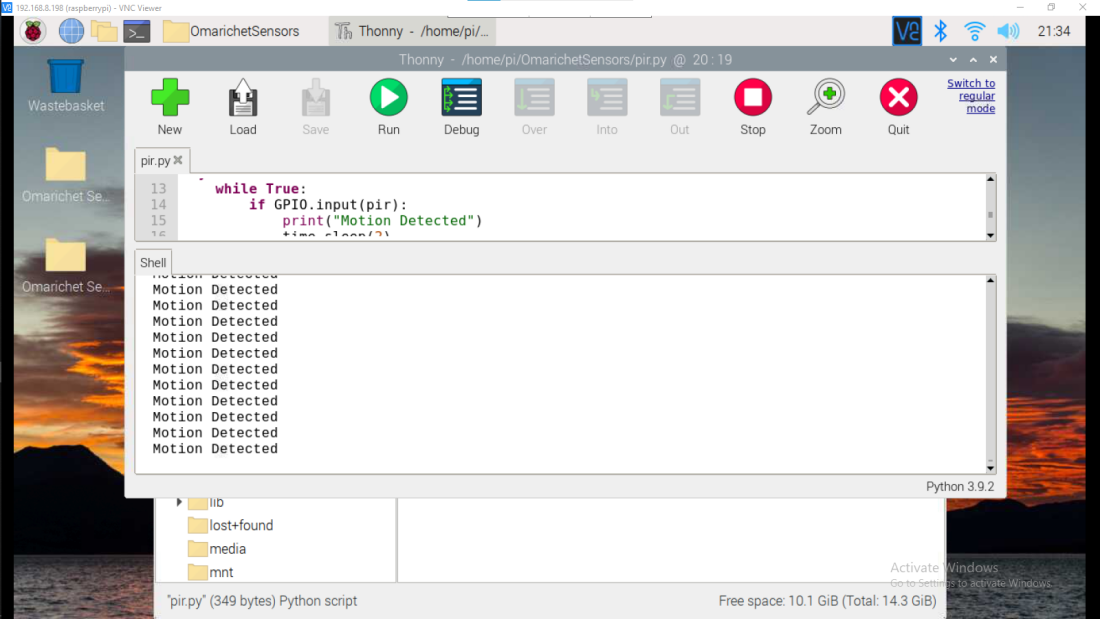
print("")

except KeyboardInterrupt:

GPIO.cleanup()

**PIR Output**

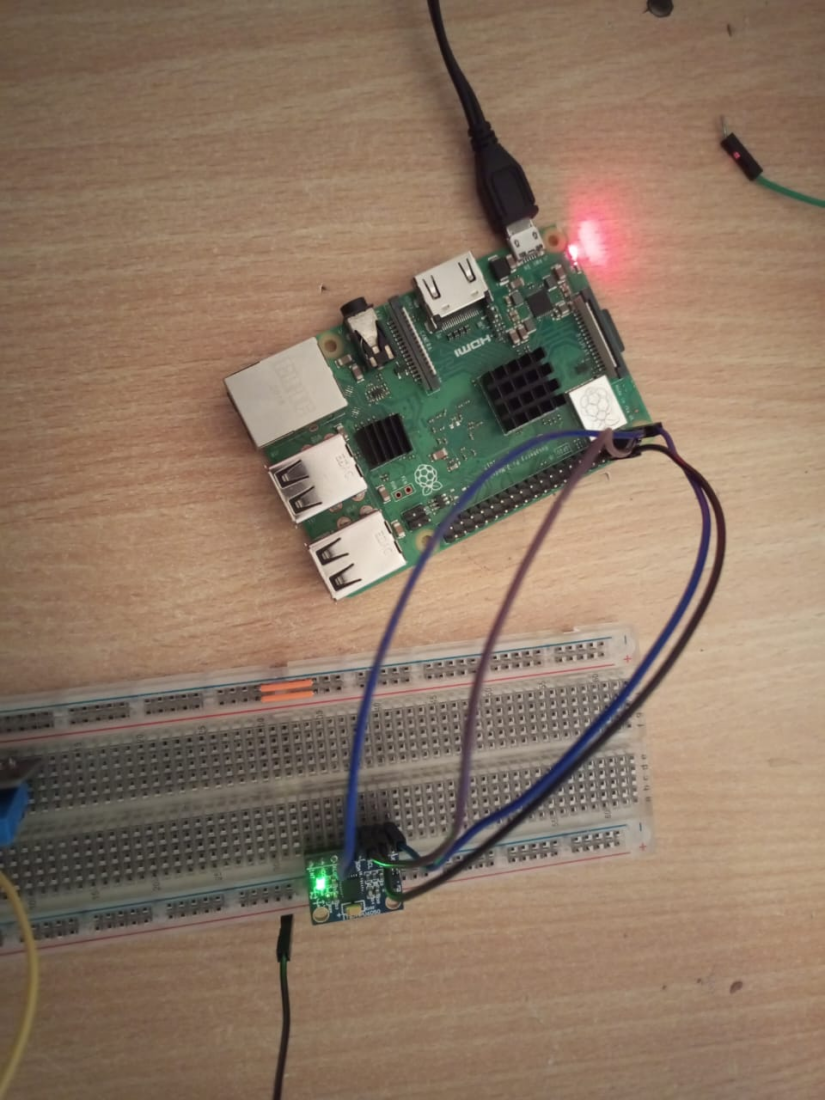
The PIR sensor actively senses places of motion of an infrared emitting body in this case a human hand.



**2.1.3 MPU-6050**

This sensor consists of a 3-axis accelerometer, 3-axis gyroscope and a temperature sensor. It can also be interfaced with other pins such as the magnetometer to increase its degrees of freedom. It communicates its data over the i2c interface. The MPU-6050 has 8 pins with 4 being a must use. These pins include a VCC pin which can be connected to either 5V or 3.3V, a ground pin, SDA pin which carries data and an SCL pin which contains the clock pulse. The latter pins are connected directly to corresponding i2c pins i.e., GPIO 2 and 3 respectively. IT is worth noting that the mpu6050 uses the 3.3V logic state and is hence safe for the raspberry pi.





**Library Installation**

Firstly, we need to ensure I2C is enabled on the raspberry pi using sudo raspi-config > interface option > Enable I2C. Reboot the PI if I2C has just been enabled.

You can use the following commands on the terminal to check if i2c ports are loaded and to check the address of the devices connected respectively.

**lsmod | grep i2c**

**Sudo i2cdetect –y 1**

Install python-pip and GPIO library using the command:

**sudo apt-get install build-essential python-dev python-pip**

**and**

**sudo pip installs RPi.GPIO**

After thus we need to install smbus library as well as the mpu6050 library.

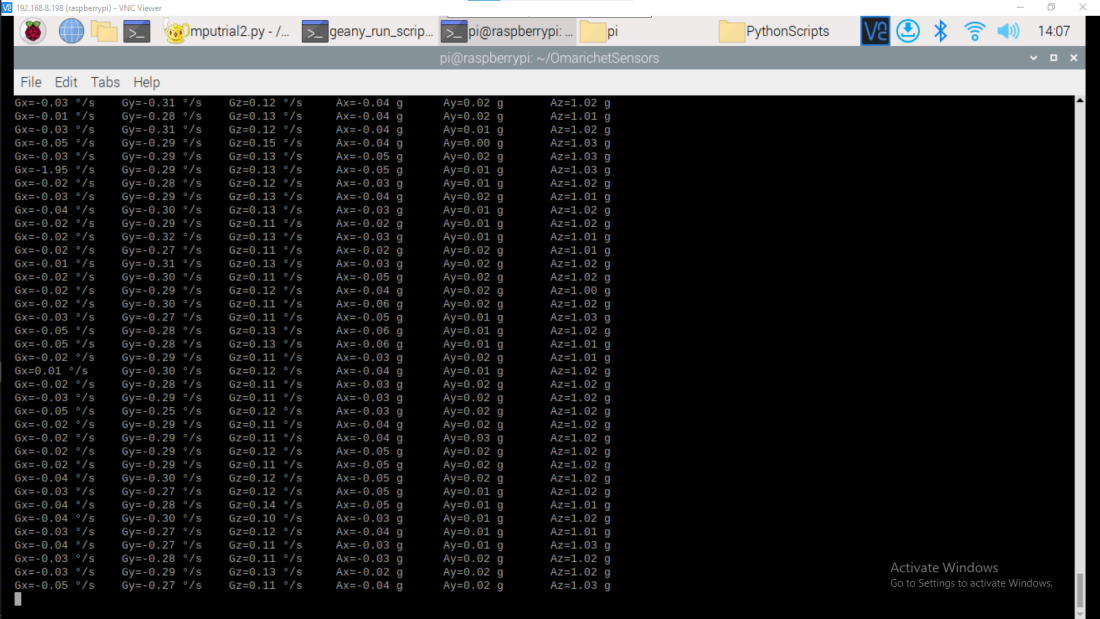
**sudo apt-get install python-smbus**

**sudo pip install mpu6050**

**MPU6050 code**

import smbus #import SMBus module of I2C  
from time import sleep #import  
  
#some MPU6050 Registers and their Address  
PWR\_MGMT\_1 = 0x6B  
SMPLRT\_DIV = 0x19  
CONFIG = 0x1A  
GYRO\_CONFIG = 0x1B  
INT\_ENABLE = 0x38  
ACCEL\_XOUT\_H = 0x3B  
ACCEL\_YOUT\_H = 0x3D  
ACCEL\_ZOUT\_H = 0x3F  
GYRO\_XOUT\_H = 0x43  
GYRO\_YOUT\_H = 0x45  
GYRO\_ZOUT\_H = 0x47  
  
  
def MPU\_Init():  
 #write to sample rate register  
 bus.write\_byte\_data(Device\_Address, SMPLRT\_DIV, 7)  
   
 #Write to power management register  
 bus.write\_byte\_data(Device\_Address, PWR\_MGMT\_1, 1)  
   
 #Write to Configuration register  
 bus.write\_byte\_data(Device\_Address, CONFIG, 0)  
   
 #Write to Gyro configuration register  
 bus.write\_byte\_data(Device\_Address, GYRO\_CONFIG, 24)  
   
 #Write to interrupt enable register  
 bus.write\_byte\_data(Device\_Address, INT\_ENABLE, 1)  
  
def read\_raw\_data(addr):  
 #Accelero and Gyro value are 16-bit  
 high = bus.read\_byte\_data(Device\_Address, addr)  
 low = bus.read\_byte\_data(Device\_Address, addr+1)  
   
 #concatenate higher and lower value  
 value = ((high << 8) | low)  
   
 #to get signed value from mpu6050  
 if(value > 32768):  
 value = value - 65536  
 return value  
  
  
bus = smbus.SMBus(1) # or bus = smbus.SMBus(0) for older version boards  
Device\_Address = 0x68 # MPU6050 device address  
  
MPU\_Init()  
  
print (" Reading Data of Gyroscope and Accelerometer")  
  
while True:  
   
 #Read Accelerometer raw value  
 acc\_x = read\_raw\_data(ACCEL\_XOUT\_H)  
 acc\_y = read\_raw\_data(ACCEL\_YOUT\_H)  
 acc\_z = read\_raw\_data(ACCEL\_ZOUT\_H)  
   
 #Read Gyroscope raw value  
 gyro\_x = read\_raw\_data(GYRO\_XOUT\_H)  
 gyro\_y = read\_raw\_data(GYRO\_YOUT\_H)  
 gyro\_z = read\_raw\_data(GYRO\_ZOUT\_H)  
   
 #Full scale range +/- 250 degree/C as per sensitivity scale factor  
 Ax = acc\_x/16384.0  
 Ay = acc\_y/16384.0  
 Az = acc\_z/16384.0  
   
 Gx = gyro\_x/131.0  
 Gy = gyro\_y/131.0  
 Gz = gyro\_z/131.0  
   
  
 print ("Gx=%.2f" %Gx, u'\u00b0'+ "/s", "\tGy=%.2f" %Gy, u'\u00b0'+ "/s", "\tGz=%.2f" %Gz, u'\u00b0'+ "/s", "\tAx=%.2f g" %Ax, "\tAy=%.2f g" %Ay, "\tAz=%.2f g" %Az)   
 sleep(1)

**MPU6050 Output**



Prefix G represents Gyroscope data while A represents Accelerometer data. X, y and z represent the three axis planes in a 3-D space. The data was collected while system was at rest and this data conforms to the expected out.

**2.2 Analogue Sensor Interfacing with the Raspberry Pi**

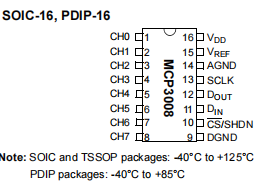
The Raspberry Pi lacks an Analogue to Digital Converter in its architecture making it unable to directly receive analogue signals from analogue-based devices. To receive analogue signals, ADC chips or microprocessors with ADCs i.e. Atmega-328p maybe used that utilize communication protocols already available on the RPI to communicate the values such as SPI etc.

One such ADC chip is the MCP3008.

**2.2.1 MCP3008**

This chip is a 10-bit, 8-channel ADC communication over the SPI protocol. It allows upto 8 individual analogue signals to be passed into it. Analogue signals have step values of between 0 to 1023. It operates on a single supply operation of 2.7-5.5V.

**Pin Configuration of the MCP3008**



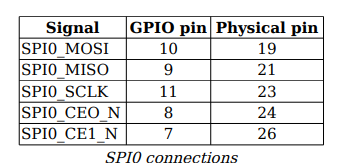
The above chip image indicates how to connect the MCP3008 pins to various peripheral circuits.

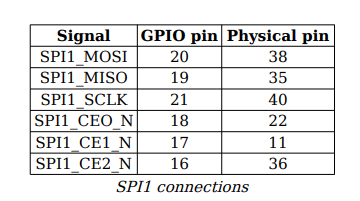
**Recommended circuit**

The chip has a Vref pin that needs to be powered at Vdd or lower voltage. A higher voltage than this would be destructive. The chip may be powered by 3.3V but this would not be conducive for the current sensor that requires 5V only. A logic translator maybe used in such a case, and this is what will be done in this project. The MCP3008 requires a bypass capacitor connected to the Vdd pin to smooth out the current. The data-sheet also recommends a design with minimal interference such as placing the chip far from high speed lines, or routing beneath the chip. Proper grounding is also recommended. The pins can be connected as follows :

* MCP3008 VDD to Raspberry Pi 3.3V
* MCP3008 VREF to Raspberry Pi 3.3V
* MCP3008 AGND to Raspberry Pi GND
* MCP3008 DGND to Raspberry Pi GND
* MCP3008 CLK to Raspberry Pi pin 18
* MCP3008 DOUT to Raspberry Pi pin 23
* MCP3008 DIN to Raspberry Pi pin 24
* MCP3008 CS/SHDN to Raspberry Pi pin 25

It is important to note that the raspberry pi has two sets of SPI pins as indicated below. The first set is ideal.





**MCP3008 CODE**

The chip outputs data via the SPI protocol. This must be enabled using

sudo raspi-config

To get an output the following installations must be done

sudo apt install python3-spydev

However, spydev comes shipped natively in the latest raspberrypi os.

The code to fetch data is as follows:

import spidev, time

spi = spidev.SpiDev()

spi.open(0,1)

def analog\_read(channel):

r = spi.xfer2([1, (8+channel) << 4, 0])

adc\_out = ((r[1]&3)<<8)+r[2]

return adc\_out

while True:

reading = analog\_read(0)

voltage = reading \* 3.3 /1024

print("Reading=%d\tVoltage=%f"%(reading, voltage))

time.sleep(1)

**Output**

The channel is connected to a light dependent resistor and gives an output corresponding to the voltage across the LDR

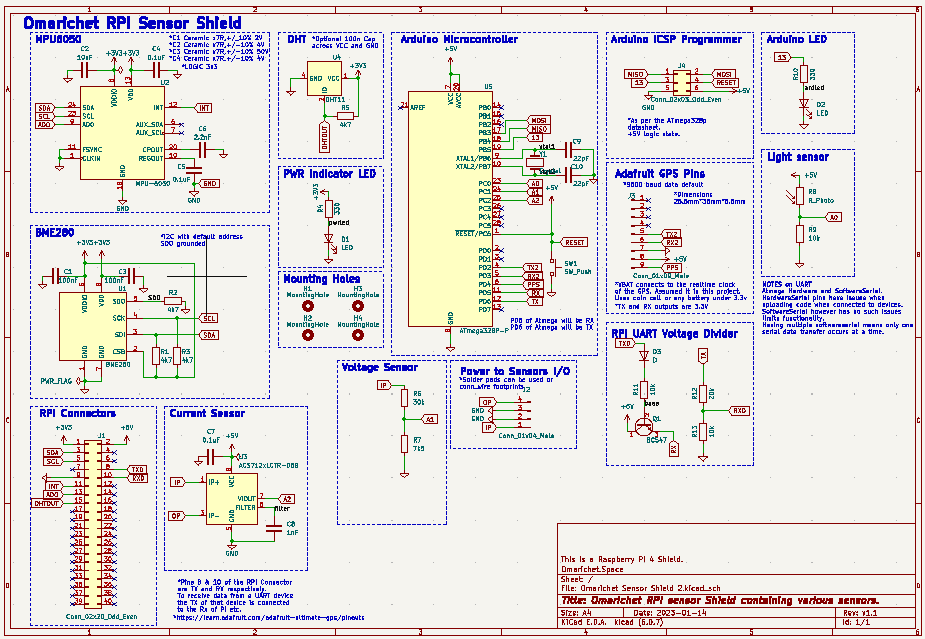
**3.0 PCB Design**

A printed Circuit Board brings together components in a circuit within a physical form and provides pads and tracks that allow for the interconnection of those devices. A good PCB design provides the circuit with low impedance signal paths, low noise and proper component layout and routing. The concepts of transmission lines need to be considered especially at high frequency or signal speeds. Kicad is a PCB design software that is open source and easy to start using. This will be my design software of choice. The PCB will be designed in a manner that will allow for stackup.

**3.1 Schematic Design**

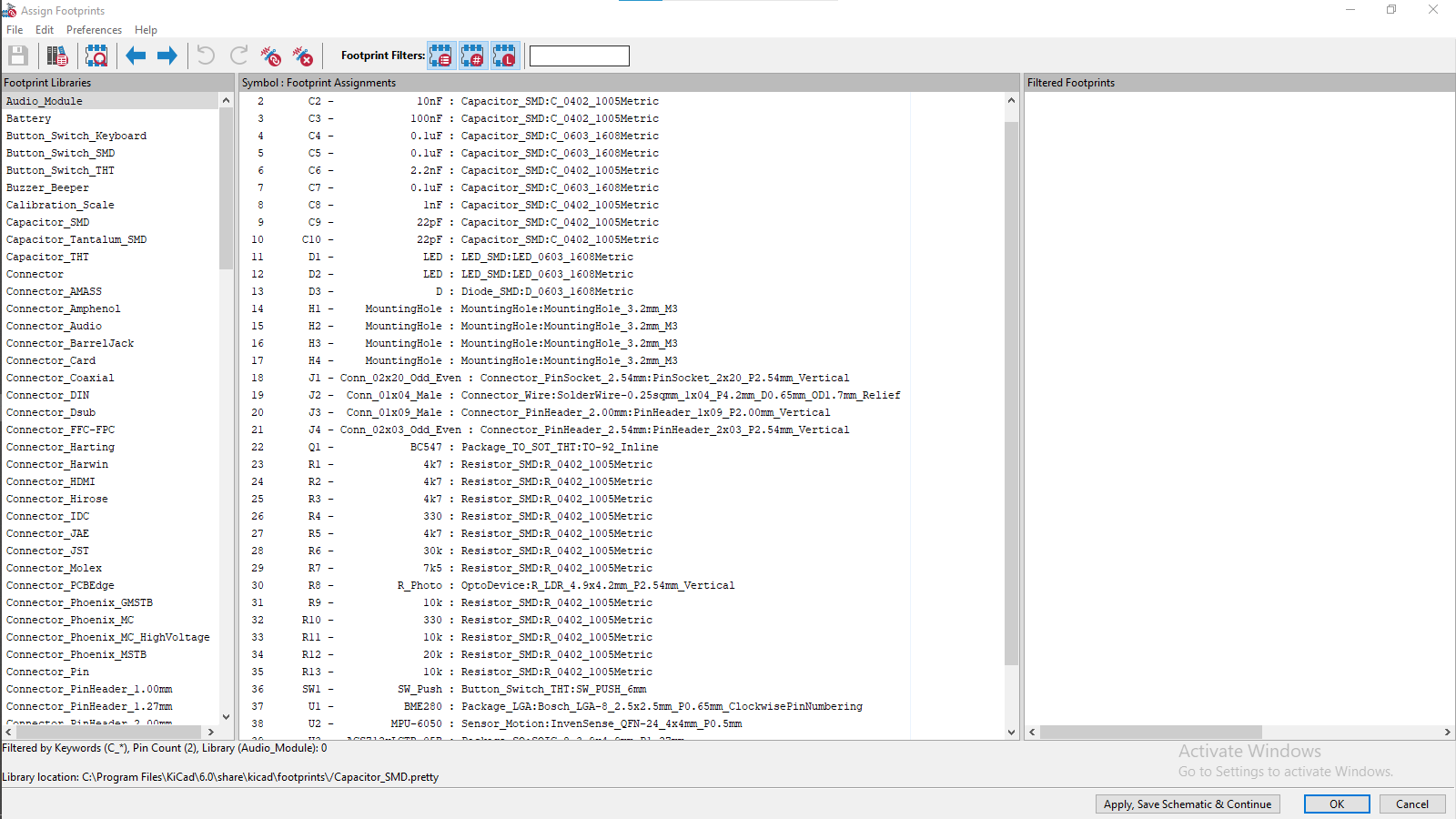
The schematic design involves use of electrical component symbols to indicate interconnection between the various components forming the circuit. This is the first section of the design phase. To come up with a good schematic, each module stated in earlier sections of this document is broken down into its individual components and this components are then placed in the schematic. Manufacturer datasheets of the main driver chips provide the recommended design parameters and schematics. This are looked into and utilised accordingly. Appropriate power and grounding plane must also be provided as well as connectors. Connectors will allow for data and power transfer from source to destination and vice versa.

The various sensor circuits and peripherals have been made.



**3.2 Footprint Assignment**

Once the schematic design is done, footprint assignment is required. This enables the next phases which are layout placement and routing. Choice is made on the actual component and accessories needed for the PCB during operation and outlook.



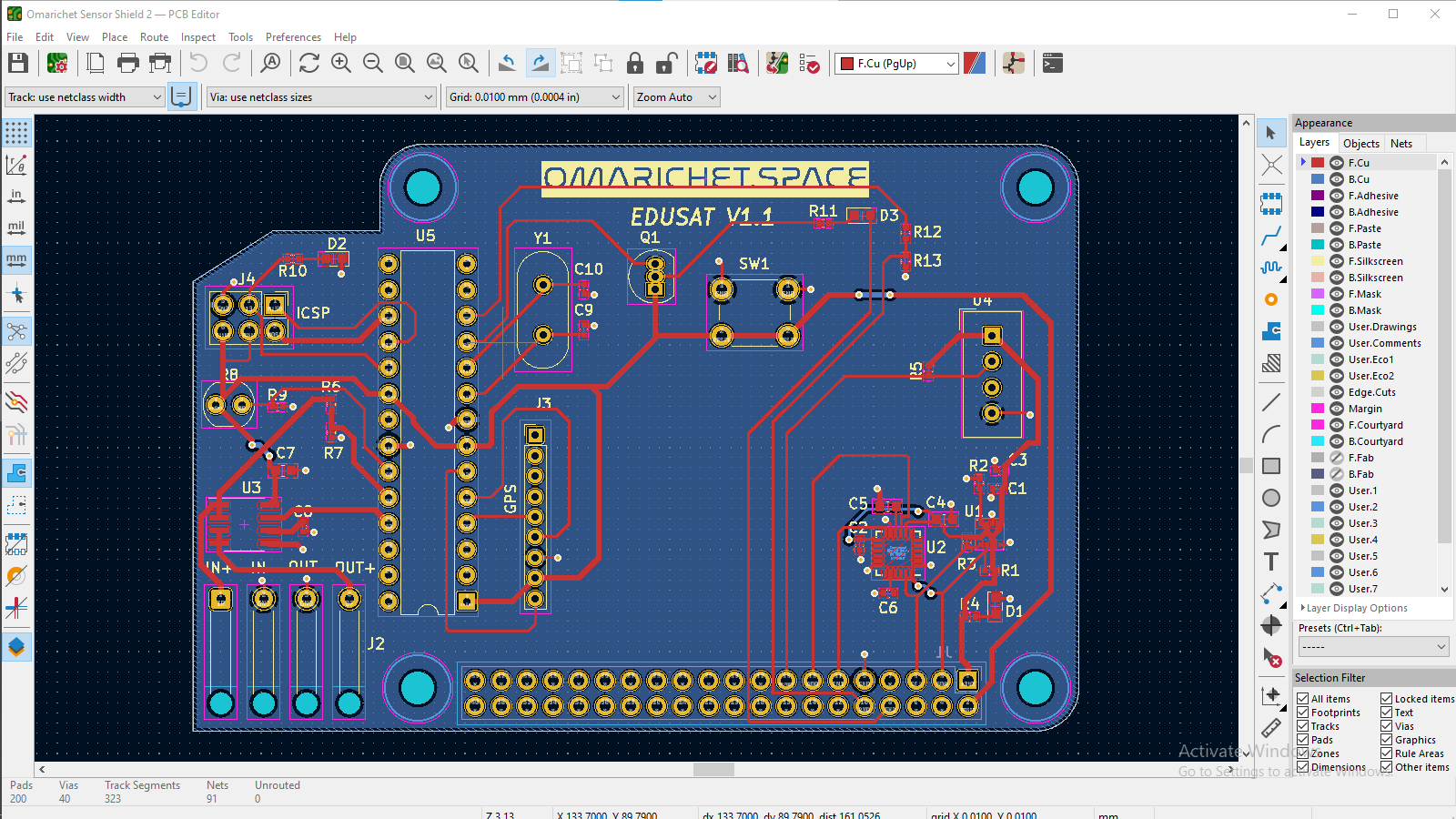
**3.3 Layout and Placement**

Proper layout out essential as it allows for proper routing that is not complicated and allows for easy routing with the shortest tracks possible.

**3.4 Routing**

The interconnection between components is done via tracks with predefined widths. Proper widths of tracks must be defined to allow proper signal or power transfer. The tracks must provide a low impedance path for the signal through the track. Ideally wider tracks provide lower impedance but the width should be commensurate with the speed and often current through it. Power tracks should be wide enough to minimize resistance and heating. Ideally power and signal track widths can be set to 0.3mm and 0.5mm. In the case of power tracks, field zones maybe utilised to lower the impedance further. A ground plane should also be provided and the ground connections made to this plane through vias.

It is important to follow design guides so as to figure out the expected routing parameters and conditions required by various components. Design constraints from the PCB manufacturer must be set to ensure the system is manufacturable. The designs made for the system was made in accordance with JLCPCB design constraints that are made available in their website at [www.jlcpcb.com](http://www.jlcpcb.com).



\*Some changes still pending to design a PCB that conforms to the 1u dimensions.

**References**

SPI trace requirements: <https://resources.altium.com/p/there-spi-trace-impedance-requirement>

MCP3008 DATASHEET <https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf>

Spidev docs <https://www.kernel.org/doc/Documentation/spi/spidev>