RPi Shield

User Guide

Table of Contents

Notes:

Adding Python 3 (IDLE) to the menu

- The Python 3 (IDLE) may not be in the menu, if you are using a new SD card (containing the Raspbian OS for Raspberry Pi).

- To add Python 3 (IDLE) to the menu, launch a Terminal and type
  sudo apt-get install idle3

- After this, you will be able to run the IDLE by clicking: Raspberry -> Programming -> Python 3 (IDLE)

Enabling SPI & I2C (for ADC, RFID, LCD, 3-Axis Accelerometer)

- Before you use some of these devices, go to Raspberry Pi Configuration -> Interfaces and enable these: SPI & I2C.

Installing SPI-Py library (for ADC, RFID)

- The RFID reader requires an additional library "SPI-Py" which can be downloaded from:
  https://github.com/lthiery/SPI-Py

- After downloading, extract it to a convenient location/folder e.g. /home/pi

- Then launch a Terminal and navigate to that location:
  cd SPI-Py-master

- And type this to install SPI-Py
  sudo python3 setup.py install

Updating rpi.gpio library (for Keypad)

- The rpi.gpio library has been updated for Raspberry Pi 4.

- To update the library in the SD card, launch a Terminal and type
  sudo apt-get update followed by
  sudo apt-get install rpi.gpio

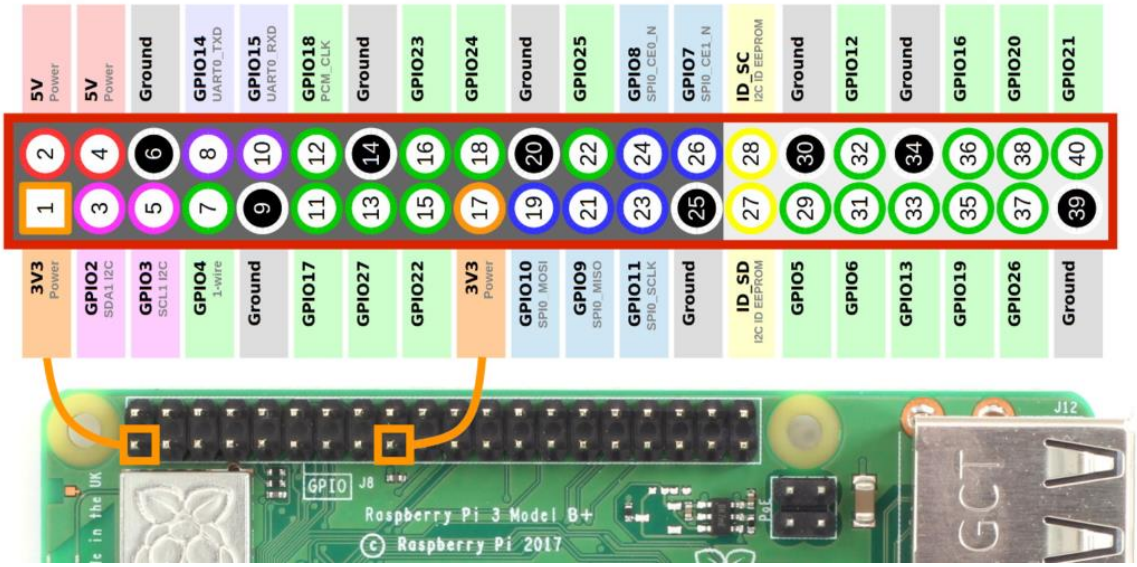
Connecting an external power supply (for DC Motor, Servo Motor, Ultrasonic Ranger – all using 5V)

- An external power adaptor is to be connected to supply more power for the DC motor, servo motor & ultrasonic ranger to work.

Part 1 – Basic IO Devices

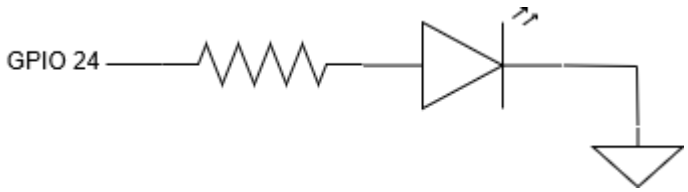
Basic IO Devices use the GPIO (General Purpose Input Output) pins on the Raspberry Pi.

The (colour-coded) pin diagram below shows the 40 pins in the GPIO pin header.



1.1 LED

The LED on the shield has been connected to GPIO24 (or physical pin 18).



To turn the LED on, run "**ledon.py**" (in the Sample Codes folder):

```
import RPi.GPIO as GPIO #import RPi.GPIO module, rename it as GPIO

GPIO.setmode(GPIO.BCM) #choose BCM mode, refer to pins as GPIO no.
GPIO.setwarnings(False)
GPIO.setup(24,GPIO.OUT) #set GPIO 24 as output

GPIO.output(24,1) #output logic high/'1'
```

Questions:

- ✓ What if you simply "import RPi.GPIO"?
- ✓ What other mode is there, beside BCM mode? (Hint: GPIO.BOARD, and refer to the LED pin as pin 18.)
- ✓ What is the effect of "GPIO.setwarnings (False)"?
- ✓ How can you set a pin as input?
- ✓ How can you turn the LED off?

To blink the LED, run "**ledblink.py**" (in the Sample Codes folder):

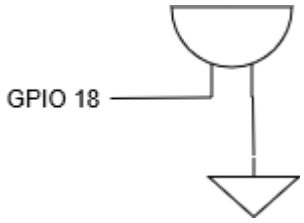
```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep #used to create delays

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(24,GPIO.OUT) #set GPIO 24 as output

while True: #loops the next 4 lines
    GPIO.output(24,1) #output logic high/'1'
    sleep(1) #delay 1 second
    GPIO.output(24,0) #output logic low/'0'
    sleep(1) #delay 1 second
```

Remarks:

- ✓ These lines of code should be self-explanatory.

1.2 Buzzer

The Buzzer on the shield has been connected to GPIO18 (or physical pin 12).



To produce a tone on the Buzzer using PWM, run "**buzzerpwm.py**" (in the Sample Codes folder):

```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep #used to create delays

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT) #set GPIO 18 as output

PWM = GPIO.PWM(18,100) #set 100Hz PWM output at GPIO 18

while True: #loops the next 3 lines
    for i in range(0,101,20):
        PWM.start(i)
        sleep(0.5)
```

Questions:

- ✓ What can you hear when the program is run?
- ✓ What will happen if the PWM frequency is changed?
- ✓ What does PWM stand for?

To beep the Buzzer, run "**buzzeronoff.py**" (in the Sample Codes folder):

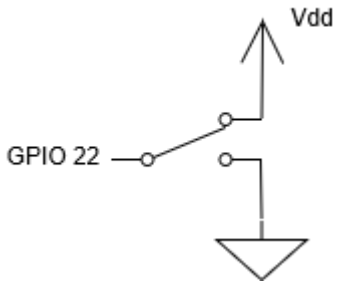
```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep #used to create delays

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(18,GPIO.OUT) #set GPIO 18 as output

while True: #loops the next 4 lines
    GPIO.output(18,1) #output logic high/'1'
    sleep(1) #delay 1 second
    GPIO.output(18,0) #output logic low/'0'
    sleep(1) #delay 1 second
```

Remarks:

- ✓ This program is almost identical to the "ledblink.py".

1.3 Slide Switch

The Slide Switch on the shield has been connected to GPIO22 (or physical pin 15).



To read the state of the Slide Switch, run "**slideswitch.py**" (in the Sample Codes folder):

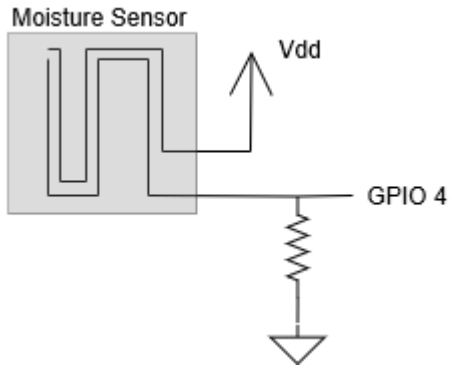
```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(22,GPIO.IN) #set GPIO 22 as input

while(True): #loop
    if GPIO.input(22): #if read a high at GPIO 22
        print("detected HIGH i.e. slider at 3.3V side")
    else: #otherwise (i.e. read a low) at GPIO 22
        print("detected LOW i.e. slider at GND side")
    sleep(0.5) # to limit print() frequency
```

Remarks:

- ✓ Printing to the monitor is a useful debugging technique.

1.4 Moisture Sensor

The Moisture Sensor on the shield has been connected to GPIO4 (or physical pin 7). This behaves like a normally open switch.

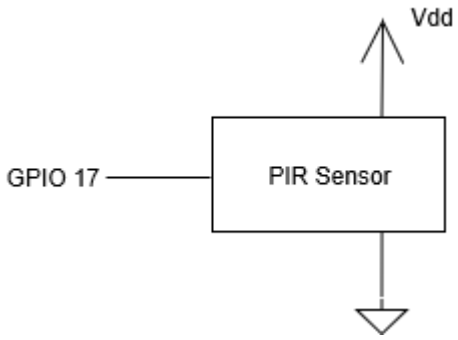


To read the state of the Moisture Sensor, run "**moisturesensor.py**" (in the Sample Codes folder):

```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(4,GPIO.IN) #set GPIO 4 as input
while (True):
    if GPIO.input(4): #if read a high at GPIO 4, moisture present
        print('detected HIGH i.e. moisture')
    else: #otherwise (i.e. read a low) at GPIO 4, no moisture
        print('detected LOW i.e. no moisture')
    sleep(0.5) # to limit print() frequency
```

1.5 PIR Sensor

The PIR Sensor on the shield has been connected to GPIO17 (or physical pin 11).



To read the state of the PIR Sensor, run "**pirsensor.py**" (in the Sample Codes folder):

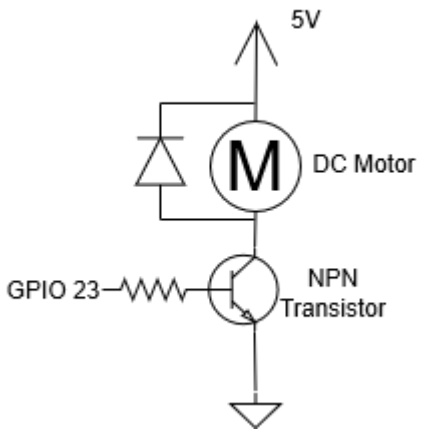
```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(17,GPIO.IN) # set GPIO 17 as input

sleep(5) #to allow sensor time to stabilize
PIR_state=0 #use this, so that only a change in state is reported
while (True):
    if GPIO.input(17): #read a HIGH i.e. motion is detected
        if PIR_state==0:
            print('detected HIGH i.e. motion detected')
            PIR_state=1
    else: #read a LOW i.e. no motion is detected
        if PIR_state==1:
            print('detected LOW i.e. no motion detected')
            PIR_state=0
    sleep(1)
    print('…')
```

Remarks:

- ✓ Both the Slide Switch program and the Moisture Sensor program can be modified, so that only a change in state in reported.

1.6 DC Motor

The DC Motor on the shield has been connected to GPIO23 (or physical pin 16). An NPN transistor is used to increase the current going through the motor. A diode is used to prevent damage to the motor circuit.



To turn the DC Motor on and off, run "**motoronoff.py**" (in the Sample Codes folder):

```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(23,GPIO.OUT) #set GPIO 23 as output

while True: #loops the next 4 lines
    GPIO.output(23,1) #output logic high/'1'
    sleep(1) #delay 1 second
    GPIO.output(23,0) #output logic low/'0'
    sleep(1) #delay 1 second
```

Remarks:

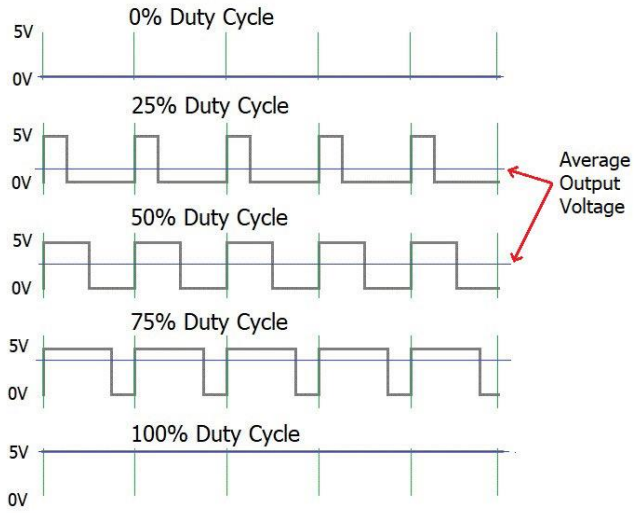
- ✓ This program is almost identical to the "ledblink.py" & "buzzeronoff.py".

To control the DC Motor speed using PWM, run "**motorpwm.py**" (in the Sample Codes folder):

```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(23,GPIO.OUT) #set GPIO 23 as output

PWM = GPIO.PWM(23,100) #set 100Hz PWM output at GPIO 23

while True: #loops the next 3 lines
    for i in range(0,101,25):
        PWM.start(i)
        sleep(0.5)
```

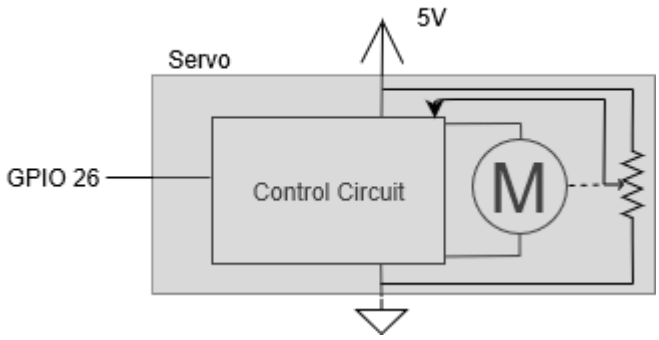
Remarks:

- ✓ This program is almost identical to the "buzzerpwm.py".
- ✓ PWM can also be used to control the brightness of an LED.
- ✓ In the program above, the duty cycle changes from 0 to 100, in steps of 25. So the average output voltage goes up, resulting in motor turning faster and faster.



1.7 Servo Motor

The Servo Motor on the shield has been connected to GPIO26 (or physical pin 37).



To sweep the Servo Motor arm, run "**servomotor.py**" (in the Sample Codes folder):

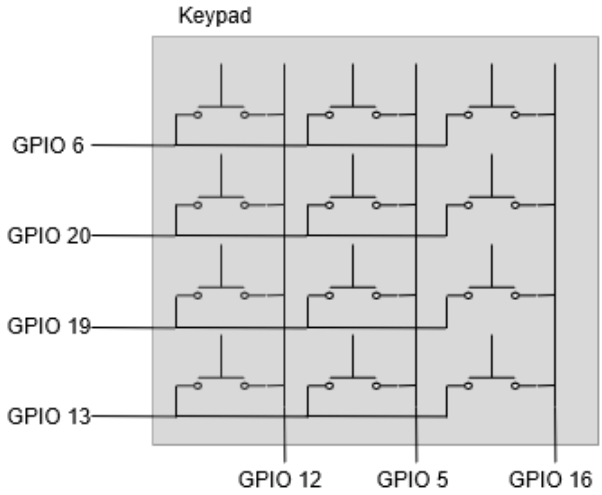
```
import RPi.GPIO as GPIO #import RPi.GPIO module
from time import sleep

GPIO.setmode(GPIO.BCM) #choose BCM mode
GPIO.setwarnings(False)
GPIO.setup(26,GPIO.OUT) #set GPIO 26 as output

PWM=GPIO.PWM(26,50) #set 50Hz PWM output at GPIO26
while (True):
    PWM.start(3) #3% duty cycle
    print('duty cycle:', 3) #3 o'clock position
    sleep(4) #allow time for movement
    PWM.start(13) #13% duty cycle
    print('duty cycle:', 13) #9 o'clock position
    sleep(4) #allow time for movement
```

Remarks:

- ✓ In this case, changing the PWM duty cycle changes the position of the servo motor arm.
- ✓ The PWM frequency is usually 50Hz i.e. period of 20ms. The exact duty cycle to be used has to be determined via trial & error.

1.8 Keypad

The Keypad on the shield has been connected to GPIO6/20/19/13 (or physical pins 31/38/35/33) for the row signals and GPIO12/5/16 (or physical pins 32/29/36) for the column signals.



Try to understand the code with the help of the comments.

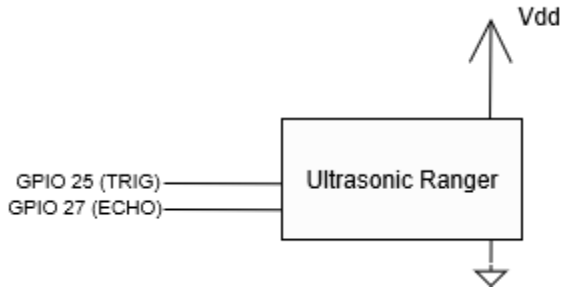
If the program does not work as expected, use these lines at a terminal to update the rpi.gpio library
sudo apt-get update
sudo apt-get install rpi.gpio

To read the Keypad, run "**keypad.py**" (in the Sample Codes folder):

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

MATRIX=[ [1,2,3],
         [4,5,6],
         [7,8,9],
         ['*',0,'#']] #layout of keys on keypad
ROW=[6,20,19,13] #row pins
COL=[12,5,16] #column pins

#set column pins as outputs, and write default value of 1 to each
for i in range(3):
    GPIO.setup(COL[i],GPIO.OUT)
    GPIO.output(COL[i],1)

#set row pins as inputs, with pull up
for j in range(4):
    GPIO.setup(ROW[j],GPIO.IN,pull_up_down=GPIO.PUD_UP)

#scan keypad
while (True):
    for i in range(3): #loop thru' all columns
        GPIO.output(COL[i],0) #pull one column pin low
        for j in range(4): #check which row pin becomes low
            if GPIO.input(ROW[j])==0: #if a key is pressed
                print (MATRIX[j][i]) #print the key pressed
                while GPIO.input(ROW[j])==0: #debounce
                    sleep(0.1)
         GPIO.output(COL[i],1) #write back default value of 1
```

1.9 Ultrasonic Ranger (SRF05)

The Ultrasonic Range on the shield has been connected to GPIO25 (or physical pin 22) for TRIG ("Trigger") and GPIO27 (or physical pin 13) for ECHO.



To read an obstacle distance using the Ultrasonic Ranger, run "**ultrasonicranger.py**" (in the Sample Codes folder):

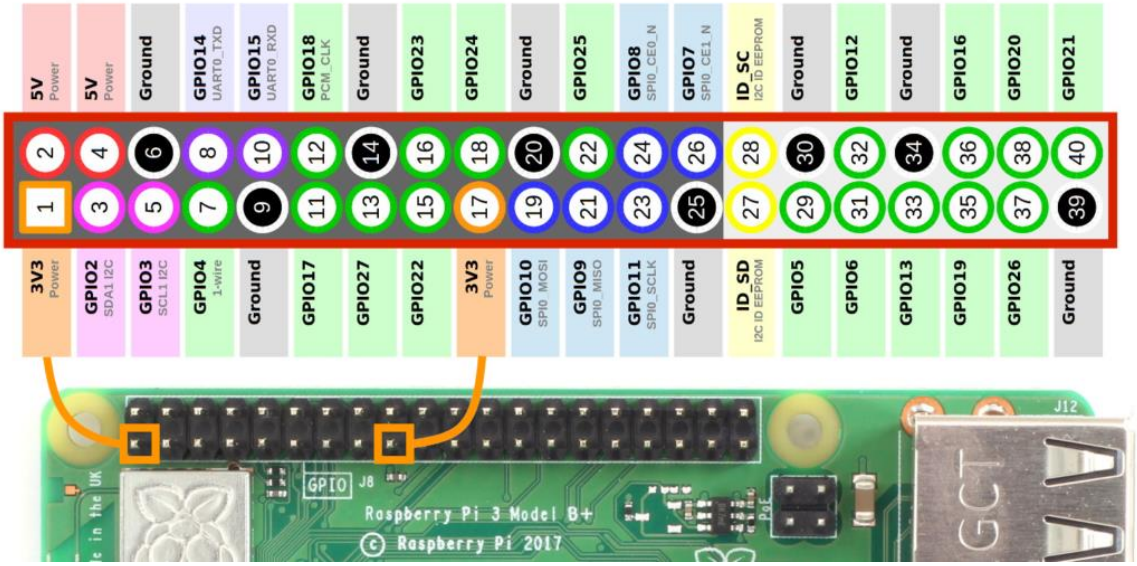
```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(25,GPIO.OUT) #GPIO25 as Trig
GPIO.setup(27,GPIO.IN) #GPIO27 as Echo

#define a function called distance below:
def distance():
    #produce a 10us pulse at Trig
    GPIO.output(25,1)
    time.sleep(0.00001)
    GPIO.output(25,0)

    #measure pulse width (i.e. time of flight) at Echo
    StartTime=time.time()
    StopTime=time.time()
    while GPIO.input(27)==0:
        StartTime=time.time() #capture start of high pulse
    while GPIO.input(27)==1:
        StopTime=time.time() #capture end of high pulse
    ElapsedTime=StopTime-StartTime

    #compute distance in cm, from time of flight
    Distance=(ElapsedTime*34300)/2
       #distance=time*speed of ultrasound,
       #/2 because to & fro
    return Distance

while (True):
    print("Measured distance = {0:0.1f} cm".format(distance()))
    time.sleep(1)
```

Try to understand the code with the help of the comments.

Part 2 – Advanced IO Devices

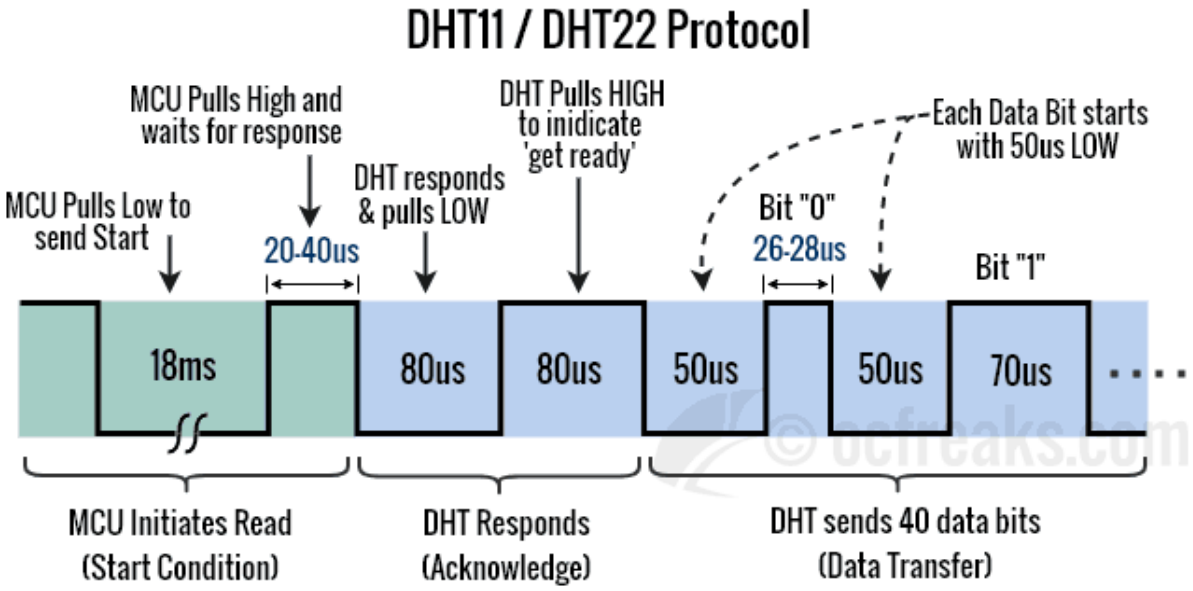
Advanced IO devices use alternate pin functions of the Raspberry Pi pins, to communicate via data transfer protocols such as I2C/TWI or SPI.

As it is difficult to write Python code from scratch for these data transfer protocols, we will use libraries of functions written by others.

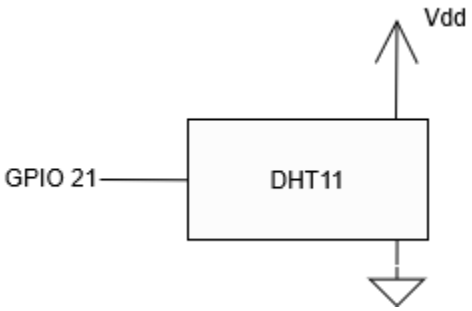


2.1 Temperature & Humidity Sensor (DHT11, using proprietary 1-wire interface)

The diagram below shows how a typical MCU (Micro-Controller Unit) can communicate with a DHT11 using 1-wire interface. Note that this data transfer protocol is proprietary & unique to DHT11, and is not the same as Dallas Semiconductor's 1-wire bus.



The DHT11 on the shield has been connected to GPIO21 (or physical pin 40).



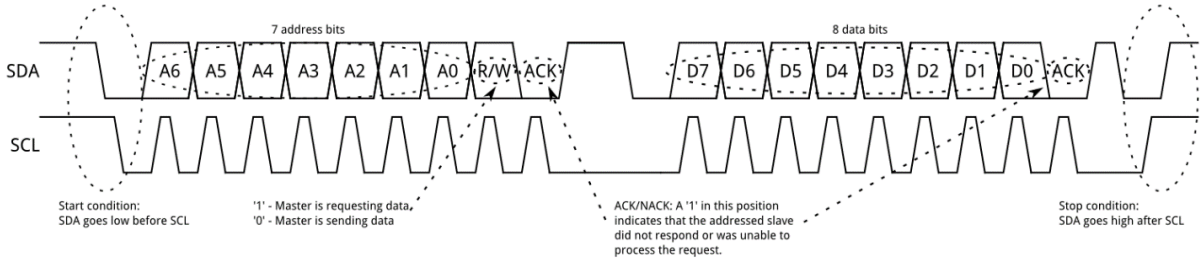
To read the DHT11, run "**dht11_example.py**" (in the Sample Codes folder, which also contains dht11.py, the library file):

```
import RPi.GPIO as GPIO
import dht11
import time
import datetime

GPIO.setwarnings(True)
GPIO.setmode(GPIO.BCM)

instance = dht11.DHT11(pin=21) #read data using pin 21

try:
    while True: #keep reading, unless a key is pressed on keyboard
        result = instance.read()
        if result.is_valid(): #print date, time and sensor values
            print("Last valid input: " +
                str(datetime.datetime.now()))
            print("Temperature: %-3.1f C" % result.temperature)
            print("Humidity: %-3.1f %%" % result.humidity)
        time.sleep(0.5) #short delay between reads

except KeyboardInterrupt:
    print("Cleanup")
    GPIO.cleanup() #Google what this means…
```

2.2 I2C / TWI

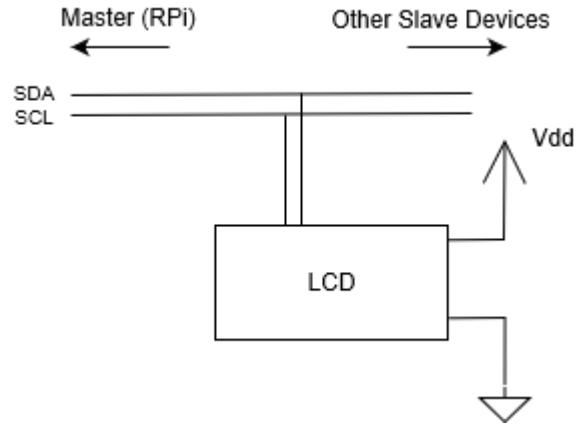
Inter-Integrated Circuit (I2C) or Two Wire Interface (TWI) is a serial data transfer protocol that uses 2 wires/lines: SDA (data) & SCL (clock). A master device (e.g. a RPi) and multiple slave devices (e.g. LCD or 3-axis sensor) can be connected to these 2 lines. Each slave has a unique address, and the master identifies which slave it wants to communicate with, by putting the slave address on the data line. It also indicates whether it is reading to writing to the slave, by using the R/W bit. The slave, if it is ready to communicate, will acknowledge (ACK) by pulling the SDA line low. After that, communication takes place.

A typical exchange is shown in the figure below:



2.3 LCD

The LCD on the shield uses TWI and has been connected to SDA1 I2C (or GPIO2 or physical pin 3) and SCL1 I2C (or GPIO3 or physical pin 5).



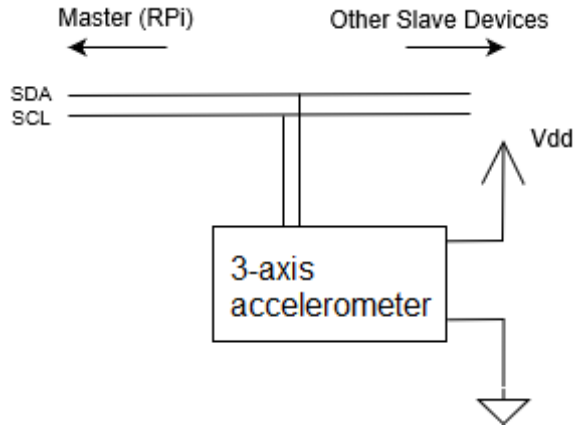
Our LCDs use the slave address 0x27. (There are LCDs that use the slave address 0x38.)

To display a line of text on the LCD, run "**lcdbasic.py**" (in the Sample Codes folder, which also contains I2C_LCD_driver.py, the library file):

```
import I2C_LCD_driver #import the library
from time import sleep

LCD = I2C_LCD_driver.lcd() #instantiate an lcd object, call it LCD
sleep(0.5)
LCD.backlight(0) #turn backlight off
sleep(0.5)
LCD.backlight(1) #turn backlight on
LCD.lcd_display_string("LCD Display Test", 1) #write on line 1
LCD.lcd_display_string("Address = 0x27", 2, 2) #write on line 2
              #starting on 3rd column

sleep(2) #wait 2 sec
LCD.lcd_clear() #clear the display
```

2.4 Three-axis accelerometer (ADXL345)

The 3-axis accelerometer on the shield uses TWI and has been connected to SDA1 I2C (or GPIO2 or physical pin 3) and SCL1 I2C (or GPIO3 or physical pin 5) (– the same connections as the LCD).



Our 3-axis accelerometers use the slave address 0x53.

To read the 3-axis accelerometer, run "**adxl345data.py**" (in the Sample Codes folder, which also contains adxl345.py, the library file):

```
import adxl345 #import the library
from time import sleep

ADDRESS=0x53

acc=adxl345.ADXL345(i2c_port=1,address=ADDRESS) #instantiate
acc.load_calib_value() #load calib. values in accel_calib
acc.set_data_rate(data_rate=adxl345.DataRate.R_100) #see datasheet
acc.set_range(g_range=adxl345.Range.G_16,full_res=True) # ..
acc.measure_start()

#acc.calibrate() #calibrate only one time

while(True):
        x,y,z=acc.get_3_axis_adjusted()
        print(x,y,z)
        sleep(0.5)
```

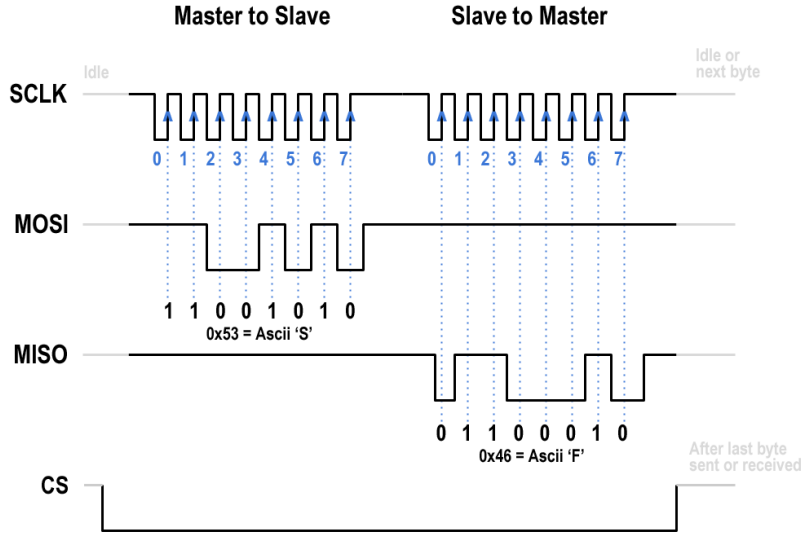
The raw data (x, y, z) read from the accelerometer is not useful. The accelerometer comes with an on-board processor, which interprets the raw data and translates to more useful information, such as tap detection. Try this python program ("**adxl345tap.py**" in the Sample Codes folder) too, which can differentiate single tap from double tap:

```
import adxl345 #import the library
from time import sleep

ADDRESS=0x53

acc=adxl345.ADXL345(i2c_port=1,address=ADDRESS) #instantiate
acc.load_calib_value()#load calib. values in accel_calib
acc.set_data_rate(data_rate=adxl345.DataRate.R_100) #see datasheet
acc.set_range(g_range=adxl345.Range.G_16, full_res=True) # ..
acc.measure_start()
acc.setTapDetection()

#acc.calibrate() #calibrate only one time

while(True):
        tap=acc.getTapDetection()
        if(tap==1):
            print("Single Tap Detected.")
        if(tap==2):
            print("Double Tap Detected.")
        sleep(0.5)
```

2.5 SPI

Serial Peripheral Interface is another serial data transfer protocol that uses a few wires/lines: SCLK (Serial CLocK), MOSI (Master Out Slave In), MISO (Master In Slave Out) and a CS (Chip Select) line for each slave. The master identifies which slave it wants to communicate with, by pulling its CS line low. After that, communication takes place by the master producing a series of clock pulses at SCLK, to move data from master to slave (via MOSI) and from slave to master (via MISO).

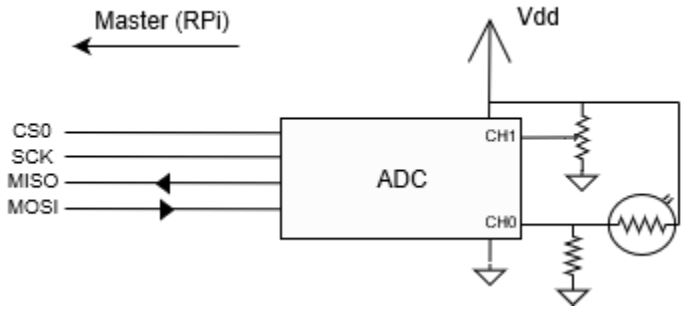
A typical exchange is shown in the figure below:



2.6 ADC (MCP3008) – potentiometer, LDR

Raspberry Pi does not have analogue input pins. So, to read analogue sensors such as brightness sensor, an external ADC (Analogue to Digital Converter) must be used. The ADC on the shield uses SPI interface and has been connected to SPI0_CE0_N (CS0 or GPIO8 or physical pin 24), SPI0_SCLK (SCK or GPIO11 or physical pin 23), SPI0_MISO (MISO or GPIO9 or physical pin 21) and SPI0_MOSI (MOSI or GPIO10 or physical pin 19).

This ADC, MCP3008 (from Microchip) gives a 10-bit conversion result. It has 8 analogue input channels, CH0 to CH7. On the shield, CH0 is connected to an LDR in series with a fixed resistor, while CH1 is connected to a potentiometer.

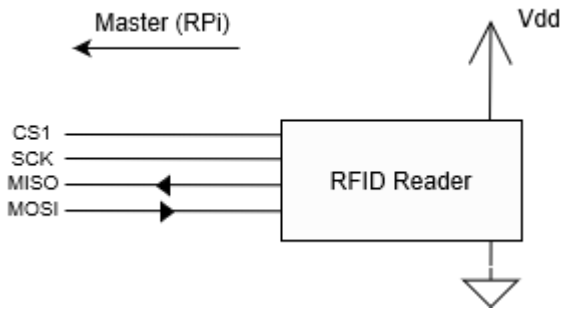


To read from the ADC, run "**adc.py**" (in the Sample Codes folder):

```
import spidev #import SPI library
from time import sleep

spi=spidev.SpiDev() #create SPI object
spi.open(0,0) #open SPI port 0, device (CS) 0

def readadc(adcnum):
    #read SPI data from the MCP3008, 8 channels in total
    if adcnum>7 or adcnum<0:
        return -1
    spi.max_speed_hz = 1350000
    r=spi.xfer2([1,8+adcnum<<4,0])
        #construct list of 3 items, before sending to ADC:
        #1(start), (single-ended+channel#) shifted left 4 bits, 0(stop)
        #see MCP3008 datasheet for details
    data=((r[1]&3)<<8)+r[2]
        #ADD first byte with 3 or 0b00000011 - masking operation
        #shift result left by 8 bits
        #OR result with second byte, to get 10-bit ADC result
    return data

while (True):
    LDR_value=readadc(0) #read ADC channel 0 i.e. LDR
    print("LDR = ", LDR_value) #print result
    pot_value=readadc(1) #read ADC channel 1 i.e. potentiometer
    print("pot = ", pot_value) #print result

    sleep(1)
```

2.7 RFID Reader (MFRC522)

The RFID Reader on the shield uses SPI interface and has been connected to SPI0_CE1_N (CS1 or GPIO7 or physical pin 26), SPI0_SCLK (SCK or GPIO11 or physical pin 23), SPI0_MISO (MISO or GPIO9 or physical pin 21) and SPI0_MOSI (MOSI or GPIO10 or physical pin 19).



Note: Each RFID Card has a UID (Unique IDentity number) which can be read by the RFID Reader. Registered UID's (i.e. "authorised cards") are stored in a database, in the form of a text file named as "authlist.txt".

In the Sample Codes folder, there are 3 sample codes (Clear Database.py, Register Cards.py, Identify Cards.py), a database file (authlist.txt), and a library folder (mfrc522, which contains _init_.py, MFRC522.py, SimpleMFRC522.py)

1. "**Clear Database.py**" – running this will clear the database of registered RFID cards. Deleting the contents in the "authlist.txt" file has the same effect.

```
import sys
auth=[]
resp = input("Enter y to clear database: ")
if resp is 'y':
    print("Clearing database...")
    f=open("authlist.txt","r+")
    auth=f.read() #read list of UID's
    num=auth.count('\n') #count how many UID's in list
    for n in range(0, num): #print "Deleted entry #1.. etc
        print("Deleted entry #", n)
    f=open("authlist.txt", "w")
        #this recreates an empty authlist.txt file
    f.close()
    print("Database cleared.")
```

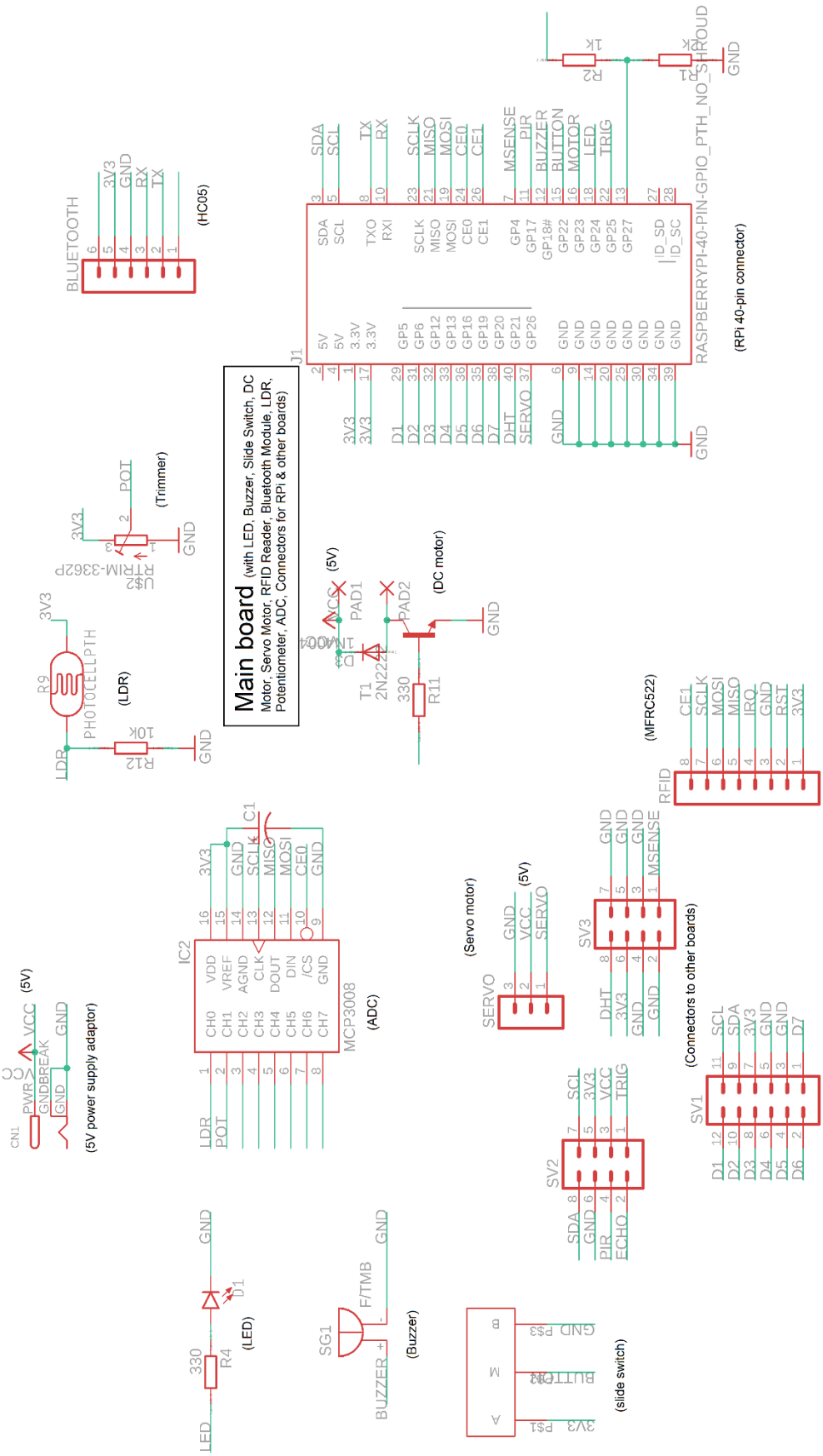
2. "**Register Cards.py**" – running this will allow a new card (i.e. new UID) to be registered.

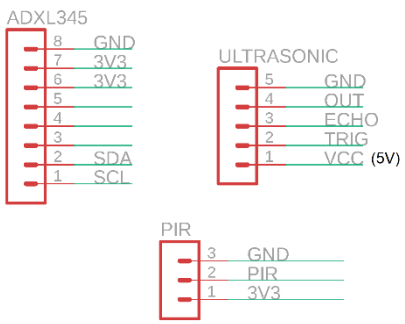
```
import RPi.GPIO as GPIO
from time import sleep
import sys
from mfrc522 import SimpleMFRC522

GPIO.setwarnings(False)
reader = SimpleMFRC522()
auth = []

while True:
        print("Hold card near the reader to register it in the database")
        id = reader.read_id()
        id = str(id)
        f = open("authlist.txt", "a+")
        f = open("authlist.txt", "r+")
        if f.mode == "r+":
              auth=f.read()
        if id not in auth:
              f.write(id)
              f.write('\n')
              f.close()
              pos = auth.count('\n')
              print("New card with UID", id, "detected; registered as entry #", pos)
        else:
              number = auth.split('\n')
              pos = number.index(id)
              print("Card with UID", id, "already registered as entry #", pos)
        sleep(2)
```

3. "**Identify Cards.py**" – running this will allow a card to be used as an access card. This is modified from the previous py file.

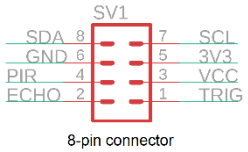
```
import RPi.GPIO as GPIO
from time import sleep
import sys
from mfrc522 import SimpleMFRC522

GPIO.setwarnings(False)
reader = SimpleMFRC522()
auth = []

while True:
        print("Hold card near the reader to check if it is in the database")
        id = reader.read_id()
        id = str(id)
        f = open("authlist.txt", "r+")
        if f.mode == "r+":
              auth=f.read()
        if id in auth:
              number = auth.split('\n')
              pos = number.index(id)
              print("Card with UID", id, "found in database entry #", pos, "; access granted")
        else:
              print("Card with UID", id, "not found in database; access denied")
        sleep(2)
```

Schematics of the various boards

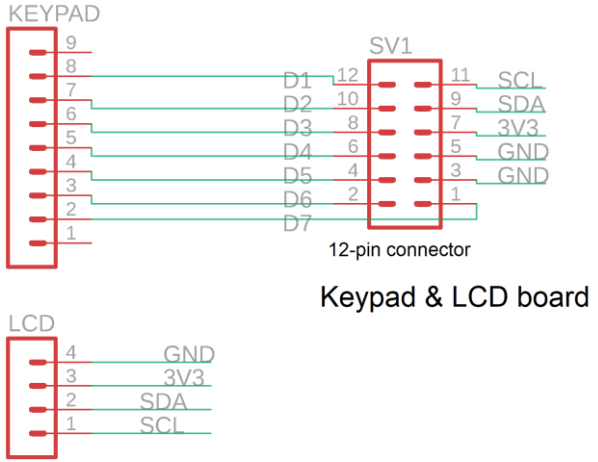




ADXL345 (3-axis accelerometer), Ultrasonic Ranger, PIR Sensor board

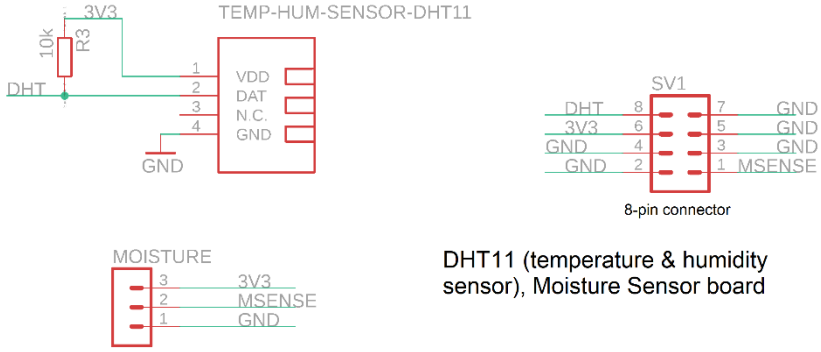


8-pin connector



12-pin connector

Keypad & LCD board



8-pin connector

DHT11 (temperature & humidity sensor), Moisture Sensor board