**SENSOR CODE EXPLANATION**

**TEMPERATURE SENSOR: (**DS18B20, 3 pins-vcc,gnd,data**)**

* modprobe w1-therm: utility which is used to add loadable modules into linux kernel. (add, view and remove.)
* w1-therm: Upon import of the w1thermsensor package the w1-therm and w1-gpio kernel modules get loaded automatically.
* W1-gpio: The Raspberry Pi doesn't have a dedicated hardware controller for 1-wire devices (some SOCs do), but 1-wire is a very slow bus and it doesn't take much to bit bang the communication using a GPIO pin. Linux has a standard driver to do this calledw1-gpio.
* 1-wire: 1-Wire is a device communications bus system designed by Dallas Semiconductor Corp. that provides low-speed data, signalling, and power over a single conductor.
* W1-slave: shows raw temperature reading output by the sensor
* Modprobe w1-therm is run every time to read the sensor.

Example: after execution of modprobe w1-therm two lines are displayed:

af 01 4b 46 7f ff 01 10 bc : crc=bc YES //if NO then error has occurred.

af 01 4b 46 7f ff 01 10 bc t=26937 //hence t is divided by 1000

* Reading the temperature takes place in two functions, read\_temp\_raw just fetches the two lines of the message from the interface. The read\_temp function wraps this up checking for bad messages and retrying until it gets a message with 'YES' on end of the first line. The function returns two values, the first being the temperature in degrees C and the second in degree F. The main loop of the program simply loops, reading the temperature and printing it, before sleeping for a second.

**Program:**

import os # import os module

import glob # import glob module

import time # import time module

os.system('modprobe w1-gpio')

os.system('modprobe w1-therm')

base\_dir = '/sys/bus/w1/devices/' # point to the address

device\_folder = glob.glob(base\_dir + '28\*')[0] # find device with address

device\_file = device\_folder + '/w1\_slave' # store the details

def read\_temp\_raw():

f = open(device\_file, 'r')

lines = f.readlines() # read the device details

f.close()

return lines

def read\_temp():

lines = read\_temp\_raw()

while lines[0].strip()[-3:] != 'YES': # ignore first line

time.sleep(0.2)

lines = read\_temp\_raw()

equals\_pos = lines[1].find('t=') # find temperature in the details

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0 # convert to Celsius

return temp\_c

while True:

print(read\_temp()) # Print temperature

time.sleep(1)

**MOISTURE SENSOR: (**KG003, output-analog/digital, sensor senses resistance value, sensor has 4 pins-vcc, gnd,A0,D0, ADS1115(ADC)-10 pins:vdd, gnd, scl, sda, addr, alrt, A0,A1,A2,A3**)**

* Adafruit\_python\_ADS1x15: library which provides functions to read values from ADC (read\_adc()).
* Read\_adc(): takes 2 parameters, index and gain. Output is mapped between 0-100 using (((value-oldmin)\*(newmax-newmin))/(oldmax-oldmin)) where, oldmin= moisture level of dry soil (from ADC), oldmax = moisture level of wet soil (from ADC), newmin=0, newmax=100.

**Program:**

# Import the ADS1x15 module.

import Adafruit\_ADS1x15

# Create an ADS1115 ADC (16-bit) instance.

adc = Adafruit\_ADS1x15.ADS1115()

# Choose a gain of 1 for reading voltages from 0 to 4.09V.

GAIN = 1

while True:

value=adc.read\_adc(0, gain=GAIN)

pval=(((value-26000)\*(100-0))/(14000-26000))

print (pval)

time.sleep(1)

**ULTRASONIC SENSOR: (**HC-SR 04**)**

* In RPi.GPIO you can use either pin numbers (BOARD) or the Broadcom GPIO numbers (BCM), **but you can only use one system in each program.**
* **Syntax:** GPIO.setup(Port\_or\_pin, GPIO.IN)
* time.time returns the time in seconds since the epoch, i.e., the point where the time starts(January 1, 1970).
* Time.sleep():it pauses the Python program, time.sleep() is the equivalent to the Bash shell's sleep command.

**Program:**

import RPi.GPIO as GPIO #import GPIO library

import time #import time library

GPIO.setmode(GPIO.BCM) #set GPIO pin numbering

TRIG = 23 #Assign name to GPIO pin

ECHO = 24 #Assign name to GPIO pin

print "Distance Measurement In Progress"

GPIO.setup(TRIG,GPIO.OUT) #Set pin mode

GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)

print "Waiting For Sensor To Settle"

time.sleep(2)

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:

pulse\_start = time.time()

while GPIO.input(ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration x 17150 #speed of sound(**343**ms/2)

distance = round(distance, 2)

print "Distance:",distance,"cm"

GPIO.cleanup()

**INTERGRATION OF ALL SENSORS:**

import os

import glob

import time

import socket

import sys

import logging

from suds.client import Client

logging.getLogger('suds.client').setLevel(logging.DEBUG)

import Adafruit\_ADS1x15

import RPi.GPIO as GPIO

#Declare vars

TRIG=23

ECHO=24

RELAY=25

water\_tank\_height=14

water\_level=0

moisture\_level\_low=40

moisture\_level\_high=50

GAIN = 1

adc = Adafruit\_ADS1x15.ADS1115()

prev\_temp = 0

prev\_mois = 0

notify=0

GPIO.setmode(GPIO.BCM)

#Functions.....

#Get serial ID

def getserial():

# Extract serial from cpuinfo file

cpuserial = "0000000000000000"

try:

f = open('/proc/cpuinfo','r')

for line in f:

if line[0:6]=='Serial':

cpuserial = line[10:26]

f.close()

except:

cpuserial = "ERROR000000000"

return cpuserial

myserial=getserial()

os.system('modprobe w1-gpio')

os.system('modprobe w1-therm')

base\_dir = '/sys/bus/w1/devices/'

device\_folder = glob.glob(base\_dir + '28\*')[0]

device\_file = device\_folder + '/w1\_slave'

#Read temp

def read\_temp\_raw():

f=open(device\_file,'r')

lines=f.readlines()

f.close()

return lines

#Read moisture

def read\_moisture():

value=adc.read\_adc(0, gain=GAIN)

rawpval=(((value-11500)\*(100-0))/(27000-11500))

per=100-rawpval

if per>100:

per=100

return per

#Read temp in C

def read\_temp():

lines = read\_temp\_raw()

while lines[0].strip()[-3:] != 'YES':

time.sleep(0.2)

lines = read\_temp\_raw()

equals\_pos = lines[1].find('t=')

if equals\_pos != -1:

temp\_string = lines[1][equals\_pos+2:]

temp\_c = float(temp\_string) / 1000.0

return temp\_c

#Measure water level in tank

def measure\_water\_level():

GPIO.output(TRIG, False)

time.sleep(2)

GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)

while GPIO.input(ECHO)==0:

pulse\_start = time.time()

while GPIO.input(ECHO)==1:

pulse\_end = time.time()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration \* 17150

distance = round(distance, 2)

GPIO.cleanup()

return int(100-((100\*distance)/water\_tank\_height))

#Switch on water pump

def switch\_on\_water\_pump():

GPIO.setmode(GPIO.BCM)

GPIO.setup(RELAY,GPIO.OUT)

GPIO.output(RELAY,True)

#Switch off water pump

def switch\_off\_water\_pump():

GPIO.setmode(GPIO.BCM)

GPIO.setup(RELAY,GPIO.OUT)

GPIO.output(RELAY,False)

GPIO.cleanup()

#Main Loop

while True:

try:

GPIO.setmode(GPIO.BCM)

GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)

GPIO.setup(RELAY,GPIO.OUT)

curr\_temp=read\_temp()

curr\_mois=read\_moisture()

water\_tank\_level=measure\_water\_level()

if 1==1:

s = socket.socket()

port = 12345

s.connect(('192.168.43.180', port))

# Define the port on which you want to connect

flag=0

if curr\_mois<moisture\_level\_low:

flag=1

switch\_on\_water\_pump()

notify=1

msg=str(curr\_temp)+","+str(curr\_mois)+","+str(myserial)+","+str(water\_tank\_level)+",On"+","

+str(notify)

s.send(msg)

while flag==1:

current\_mois=read\_moisture()

if current\_mois>=moisture\_level\_high:

switch\_off\_water\_pump()

notify=1

msg=str(curr\_temp)+","+str(curr\_mois)+","+str(myserial)+","+str(water\_tank\_level)+",Off"+","

+str(notify)

s.send(msg)

flag=0

break;

else:

time.sleep(3)

else:

notify=0

msg=str(curr\_temp)+","+str(curr\_mois)+","+str(myserial)+","+str(water\_tank\_level)+",Off"+","

+str(notify)

# connect to the server on local computer

s.send(msg)

prev\_mois=curr\_mois

prev\_temp=curr\_temp

s.close()

time.sleep(1800)

else:

print(read\_temp(),read\_moisture(),myserial)

except IOerror:

print()

finally:

GPIO.cleanup()