**Source code for proteus software :**

#!/usr/bin/python

import spidev

import time

import os

import RPi.GPIO as GPIO

import pio

import Ports

pio.uart=Ports.UART () # Define serial port

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

# Open SPI bus

spi = spidev.SpiDev()

spi.open(0,0)

# Define GPIO to LCD mapping

LCD\_RS = 7

LCD\_E = 11

LCD\_D4 = 12

LCD\_D5 = 13

LCD\_D6 = 15

LCD\_D7 = 16

Motor\_1= 29

Motor\_2 =31

# Define sensor channels

temp\_channel = 0

'''

define pin for lcd

'''

# Timing constants

E\_PULSE = 0.0005

E\_DELAY = 0.0005

delay = 1

GPIO.setup(LCD\_E, GPIO.OUT) # E

GPIO.setup(LCD\_RS, GPIO.OUT) # RS

GPIO.setup(LCD\_D4, GPIO.OUT) # DB4

GPIO.setup(LCD\_D5, GPIO.OUT) # DB5

GPIO.setup(LCD\_D6, GPIO.OUT) # DB6

GPIO.setup(LCD\_D7, GPIO.OUT) # DB7

GPIO.setup(Motor\_1, GPIO.OUT) # Motor\_1

GPIO.setup(Motor\_2, GPIO.OUT) # Motor\_1

# Define some device constants

LCD\_WIDTH = 16 # Maximum characters per line

LCD\_CHR = True

LCD\_CMD = False

LCD\_LINE\_1 = 0x80 # LCD RAM address for the 1st line

LCD\_LINE\_2 = 0xC0 # LCD RAM address for the 2nd line

'''

Function Name :lcd\_init()

Function Description : this function is used to initialized lcd by sending the different commands

'''

def lcd\_init():

# Initialise display

lcd\_byte(0x33,LCD\_CMD) # 110011 Initialise

lcd\_byte(0x32,LCD\_CMD) # 110010 Initialise

lcd\_byte(0x06,LCD\_CMD) # 000110 Cursor move direction

lcd\_byte(0x0C,LCD\_CMD) # 001100 Display On,Cursor Off, Blink Off

lcd\_byte(0x28,LCD\_CMD) # 101000 Data length, number of lines, font size

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

time.sleep(E\_DELAY)

'''

Function Name :lcd\_byte(bits ,mode)

Fuction Name :the main purpose of this function to convert the byte data into bit and send to lcd port

'''

def lcd\_byte(bits, mode):

# Send byte to data pins

# bits = data

# mode = True for character

# False for command

GPIO.output(LCD\_RS, mode) # RS

# High bits

GPIO.output(LCD\_D4, False)

GPIO.output(LCD\_D5, False)

GPIO.output(LCD\_D6, False)

GPIO.output(LCD\_D7, False)

if bits&0x10==0x10:

GPIO.output(LCD\_D4, True)

if bits&0x20==0x20:

GPIO.output(LCD\_D5, True)

if bits&0x40==0x40:

GPIO.output(LCD\_D6, True)

if bits&0x80==0x80:

GPIO.output(LCD\_D7, True)

# Toggle 'Enable' pin

lcd\_toggle\_enable()

# Low bits

GPIO.output(LCD\_D4, False)

GPIO.output(LCD\_D5, False)

GPIO.output(LCD\_D6, False)

GPIO.output(LCD\_D7, False)

if bits&0x01==0x01:

GPIO.output(LCD\_D4, True)

if bits&0x02==0x02:

GPIO.output(LCD\_D5, True)

if bits&0x04==0x04:

GPIO.output(LCD\_D6, True)

if bits&0x08==0x08:

GPIO.output(LCD\_D7, True)

# Toggle 'Enable' pin

lcd\_toggle\_enable()

'''

Function Name : lcd\_toggle\_enable()

Function Description:basically this is used to toggle Enable pin

'''

def lcd\_toggle\_enable():

# Toggle enable

time.sleep(E\_DELAY)

GPIO.output(LCD\_E, True)

time.sleep(E\_PULSE)

GPIO.output(LCD\_E, False)

time.sleep(E\_DELAY)

'''

Function Name :lcd\_string(message,line)

Function Description :print the data on lcd

'''

def lcd\_string(message,line):

# Send string to display

message = message.ljust(LCD\_WIDTH," ")

lcd\_byte(line, LCD\_CMD)

for i in range(LCD\_WIDTH):

lcd\_byte(ord(message[i]),LCD\_CHR)

# Function to read SPI data from MCP3008 chip

# Channel must be an integer 0-7

def ReadChannel(channel):

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

data = ((adc[1]&3) << 8) + adc[2]

return data

# Function to calculate temperature from

# TMP36 data, rounded to specified

# number of decimal places.

def ConvertTemp(data,places):

# ADC Value

# (approx) Temp Volts

# 0 -50 0.00

# 78 -25 0.25

# 155 0 0.50

# 233 25 0.75

# 310 50 1.00

# 465 100 1.50

# 775 200 2.50

# 1023 280 3.30

temp = ((data \* 330)/float(1023))

temp = round(temp,places)

return temp

# Define delay between readings

delay = 5

lcd\_init()

lcd\_string("welcome ",LCD\_LINE\_1)

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

lcd\_string("Data ",LCD\_LINE\_1)

while 1:

Data=pio.uart.recv()

pio.uart.print(Data)

if(Data == "1"):

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

lcd\_string(" Mask Detected",LCD\_LINE\_1)

time.sleep(1)

temp\_level = ReadChannel(temp\_channel)

temp = ConvertTemp(temp\_level,2)

# Print out results

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

lcd\_string("Temperature ",LCD\_LINE\_1)

lcd\_string(str(temp),LCD\_LINE\_2)

if((temp > 90) & (temp < 100)):

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

lcd\_string("Gate Open",LCD\_LINE\_2)

GPIO.output(Motor\_1, True)

GPIO.output(Motor\_2, False)

time.sleep(1)

GPIO.output(Motor\_1, False)

GPIO.output(Motor\_2, True)

time.sleep(1)

GPIO.output(Motor\_1, False)

GPIO.output(Motor\_2, False)

time.sleep(1)

else :

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

lcd\_string("High Temperature ",LCD\_LINE\_1)

lcd\_string("Gate close",LCD\_LINE\_2)

GPIO.output(Motor\_1, False)

GPIO.output(Motor\_2, False)

time.sleep(1)

else:

lcd\_byte(0x01,LCD\_CMD) # 000001 Clear display

lcd\_string(" Mask not ",LCD\_LINE\_1)

lcd\_string(" Detected ",LCD\_LINE\_2)

time.sleep(0.5)