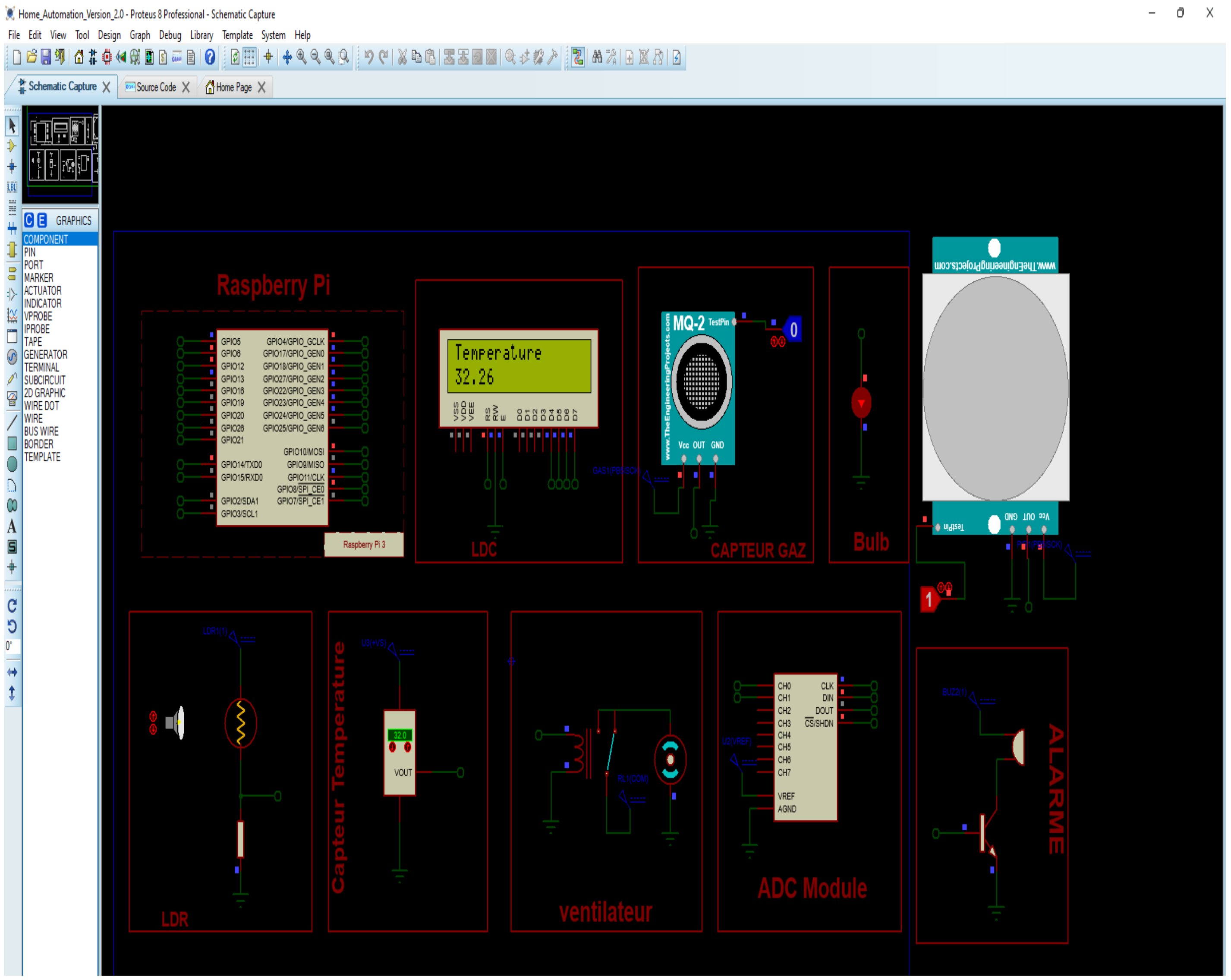
Home_Automation_Version_2.0 - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
Source Code
Home Page
GRAPHICS
COMPONENT
PIN
PORT
MARKER
ACTUATOR
INDICATOR
VPROBE
IPROBE
TAPE
GENERATOR
TERMINAL
SUBCIRCUIT
2D GRAPHIC
WIRE DOT
WIRE
BUS WIRE
BORDER
TEMPLATE
Raspberry Pi
GPIO5
GPIO6
GPIO12
GPIO13
GPIO16
GPIO19
GPIO20
GPIO26
GPIO21
GPIO14/TXD0
GPIO15/RXD0
GPIO2/SDA1
GPIO3/SCL1
GPIO4/GPIO_GCLK
GPIO17/GPIO_GEN0
GPIO18/GPIO_GEN1
GPIO27/GPIO_GEN2
GPIO22/GPIO_GEN3
GPIO23/GPIO_GEN4
GPIO24/GPIO_GEN5
GPIO25/GPIO_GEN6
GPIO10/MOSI
GPIO9/MISO
GPIO11/CLK
GPIO8/SPI_CE0
GPIO7/SPI_CE1
Raspberry Pi 3
Temperature
32.26
LDC
MQ-2
TestPin
www.TheEngineeringProjects.com
Vcc OUT GND
CAPTEUR GAZ
Bulb
www.TheEngineeringProjects.com
Vcc OUT GND
TestPin
LDR
Capteur Temperature
32.0
VOUT
ventilateur
CH0
CH1
CH2
CH3
CH4
CH5
CH6
CH7
VREF
AGND
CLK
DIN
DOUT
CS/SHDN
ADC Module
ALARME

```
import spidev

import time

import RPi.GPIO as GPIO

import pio

import Ports


GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)


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


# 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

bulb_pin =  32

motor_pin =  18

pir_pin = 31

gas_pin = 29

buzzer_pin =33

# Define sensor channels
```

```
ldr_channel = 0

temp_channel  = 1

'''

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(bulb_pin, GPIO.OUT) # DB7

GPIO.setup(motor_pin, GPIO.OUT) # DB7

GPIO.setup(buzzer_pin, GPIO.OUT) # DB7

GPIO.setup(gas_pin, GPIO.IN) # DB7

GPIO.setup(pir_pin, GPIO.IN) # DB7

# 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)
time.sleep(1)
while 1:
  gas_data =  GPIO.input(gas_pin)
  if(gas_data == True):
   lcd_byte(0x01,LCD_CMD) # 000001 Clear display
   lcd_string("Fire Detected",LCD_LINE_1)
   lcd_string("Buzzer On",LCD_LINE_2)
   GPIO.output(bulb_pin, False)
   GPIO.output(motor_pin, False)
   GPIO.output(buzzer_pin, True)
   time.sleep(0.5)
   while(1):
    lcd_byte(0x01,LCD_CMD) # 000001 Clear display
    lcd_string("Sending Message",LCD_LINE_1)
    pio.uart.println("AT")
    pio.uart.println("AT+CMGF=1")
    pio.uart.println("AT+CMGS=\"+919922512017\"\r")
    pio.uart.println("Fire Detected")
  pir_data =  GPIO.input(pir_pin)
  if(pir_data  ==  True):
   light_level = ReadChannel(ldr_channel)
   time.sleep(0.2)
   lcd_byte(0x01,LCD_CMD) # 000001 Clear display
   lcd_string("Person Detected  ",LCD_LINE_1)
   time.sleep(0.5)
```

```
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
lcd_string("Automatic Light and  ",LCD_LINE_1)
lcd_string("Fan System Active ",LCD_LINE_2)
time.sleep(0.5)
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
lcd_string("Light Intencsty  ",LCD_LINE_1)
lcd_string(str(light_level),LCD_LINE_2)
time.sleep(0.5)
if(light_level < 100 ):
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Bulb ON",LCD_LINE_2)
 GPIO.output(bulb_pin, True)
 time.sleep(0.5)
else:
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Bulb OFF",LCD_LINE_2)
 GPIO.output(bulb_pin, False)
 time.sleep(0.5)
# Print out results
temp_level = ReadChannel(temp_channel)
time.sleep(0.5)
temperature      = ConvertTemp(temp_level,2)
lcd_byte(0x01,LCD_CMD) # 000001 Clear display
lcd_string("Temperature  ",LCD_LINE_1)
lcd_string(str(temperature),LCD_LINE_2)
time.sleep(0.5)
if(temperature > 30):
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 lcd_string("Fan ON  ",LCD_LINE_1)
```

```
  GPIO.output(motor_pin, True)
  time.sleep(0.5)
 else:
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Fan Off  ",LCD_LINE_1)
  GPIO.output(motor_pin, False)
  time.sleep(0.5)
else:
  lcd_byte(0x01,LCD_CMD) # 000001 Clear display
  lcd_string("Person Not Detected",LCD_LINE_1)
  GPIO.output(motor_pin, False)
  GPIO.output(bulb_pin, False)
  time.sleep(0.5)
```