WEEK 5:

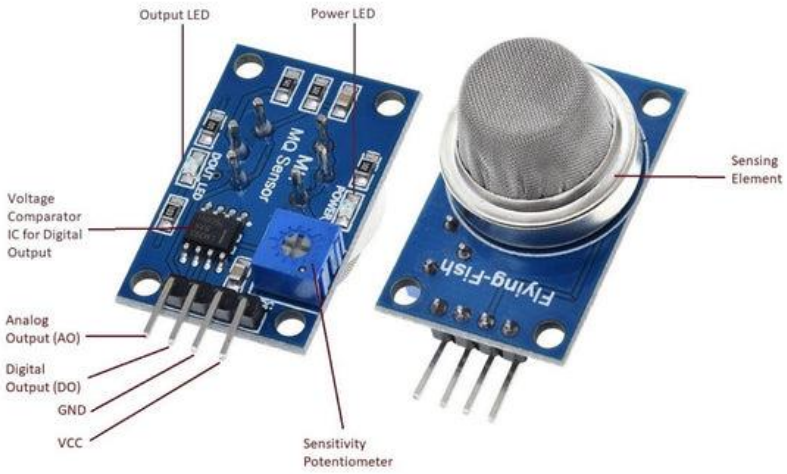
3.Design and implement to capture Gas sensor and send sensor data to cloud from your NodeMCU device Using Arduino IDE

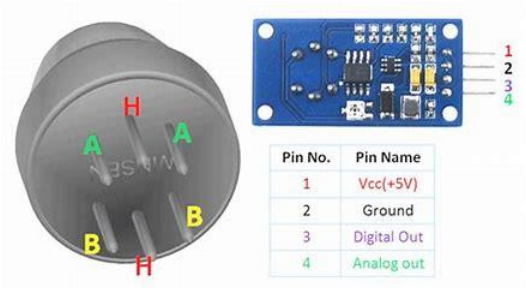
AIM: To capture gas sensor and send sensor data to cloud from NodeMCU device using Arduino IDE.

Components Required:

- MQ-6 Gas sensor
- NodeMCU (ESP8266)
- Arduino IDE
- Jumper wires
- Bread board
- Thingspeak account

Gas Sensor: (MQ-06)





NodeMCU:

Today, IOT applications are on the rise, and connecting objects are getting more and more important. There are several ways to connect objects such as Wi-Fi protocol.

NodeMCU is an open-source platform based on ESP8266 which can connect objects and let data transfer using the Wi-Fi protocol. In addition, by providing some of the most important features of microcontrollers such as GPIO, PWM, ADC, and etc., it can solve many of the project's needs alone.

The general features of this board are as follows:

Easy to use

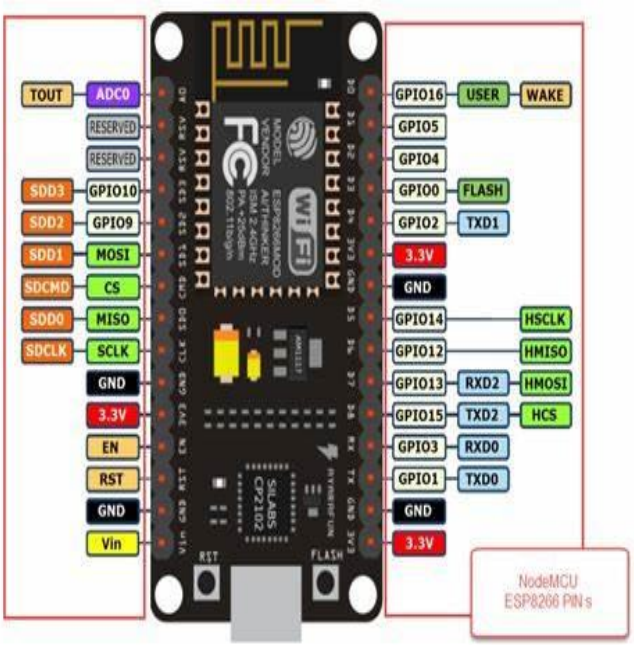
Programmability with Arduino IDE or IUA languages

Available as an access point or station

practicable in Event-driven API applications

Having an internal antenna

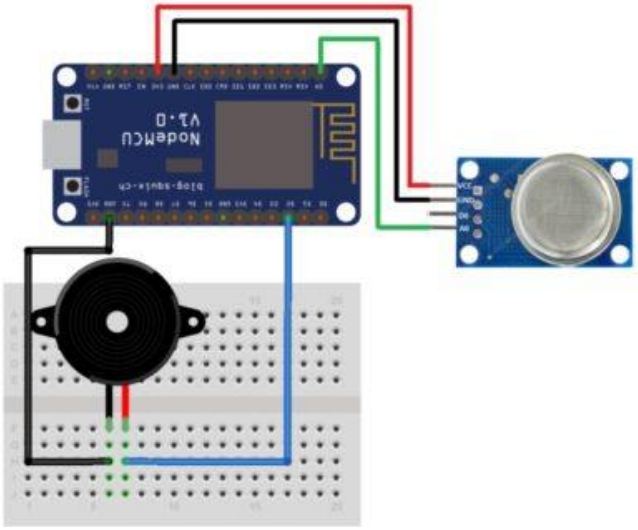
Containing 13 GPIO pins, 10 PWM channels, I2C, SPI, ADC, UART, and 1-Wire



Steps to follow

- Install Arduino IDE
- Set Up Cloud Account on Thingsio.AI
- Set Up NodeMCU
- Charts and Visualizations on Thingsio.AI
- Register the account in http://thingsio.ai/#/register.
- You will get the notification for sign up.
- Go in your email address and verify your account. You will get the notification for the email verification

Connections:



Procedure:

- Connect the DHT11 sensor with nodeMCU, VCC- 3V3, GND -GND, DATA OUT – A0 (as per code)
- Write the code in Arduino IDE
- Select nodeMCU board and Port in Tools
- Compile and upload the code

Sketch:

```
#include <ESP8266WiFi.h>
String apiKey = "SKP9YQY2CFVNK919"; // Enter your Write API key from ThingSpeak
const char *ssid = "daya"; // replace with your wifi ssid and wpa2 key
const char *pass = "12345678";
const char* server = "api.thingspeak.com";
WiFiClient client;
void setup()
{
Serial.begin(115200);
delay(10);
Serial.println("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, pass);
while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
```

```
Serial.println("");
Serial.println("WiFi connected");
}
void loop()
{
float h = analogRead(A0);
if (isnan(h))
{
Serial.println("Failed to read from MQ-5 sensor!");
return;
}

if (client.connect(server, 80)) // "184.106.153.149" or api.thingspeak.com
{
String postStr = apiKey;
postStr += "&field1=";
postStr += String(h/1023*100);
postStr += "r\n";
client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
client.print("Content-Type: application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(postStr.length());
client.print("\n\n");
client.print(postStr);
Serial.print("Gas Level: ");
Serial.println(h/1023*100);
Serial.println("Data Send to Thingspeak");
}
delay(500);
client.stop();
Serial.println("Waiting...");
```

```
// thingspeak needs minimum 15 sec delay between updates.

delay(1500);

}
```

Result: After the code is uploaded and the connections are made, the buzzer should make noise when a flammable gas (or a gas from the list) is detected by the MQ-6

WEEK 6:

4. Design and Implementation of Humidity and Temperature monitoring using Arduino and upload to cloud using MQTT

AIM: To display Humidity and Temperature using Arduino and upload to cloud using MQTT

Components Required:

- NodeMCU
- DHT11 Wires
- Jumper Wires

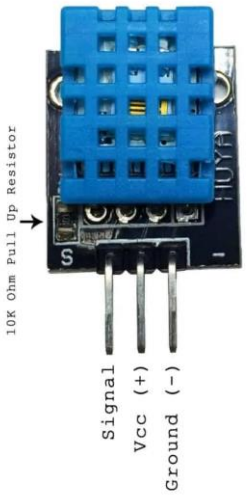
DHT11:

DHT11 is a low-cost digital sensor for sensing temperature and humidity. This sensor can be easily interfaced with any micro-controller such as Arduino, Raspberry Pi etc… to measure humidity and temperature instantaneously.

DHT11 humidity and temperature sensor is available as a sensor and as a module. The difference between this sensor and module is the pull-up resistor and a power-on LED. DHT11 is a relative humidity sensor. To measure the surrounding air this sensor uses a thermistor and a capacitive humidity sensor.

DHT11 Sensor and Its Working

Humidity is the measure of water vapour present in the air. The level of humidity in air affects various physical, chemical and biological processes. In industrial applications, humidity can affect the business cost of the products, health and safety of the employees. So, in semiconductor industries and control system industries measurement of humidity is very important. Humidity measurement determines the amount of moisture present in the gas that can be a mixture of water vapour, nitrogen, argon or pure gas etc… Humidity sensors are of two types based on their measurement units. They are a relative humidity sensor and Absolute humidity sensor. DHT11 is a digital temperature and humidity sensor.



Working Principle of DHT11 Sensor

DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. Change in the capacitance value occurs with the change in humidity levels. The IC measure, process this changed resistance values and change them into digital form.

For measuring temperature this sensor uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. To get larger resistance value even for the smallest change in temperature, this sensor is usually made up of semiconductor ceramics or polymers.

The temperature range of DHT11 is from 0 to 50 degree Celsius with a 2-degree accuracy. Humidity range of this sensor is from 20 to 80% with 5% accuracy. The sampling rate of this sensor is 1Hz .i.e. it gives one reading for every second. DHT11 is small in size with operating voltage from 3 to 5 volts. The maximum current used while measuring is 2.5mA

Applications

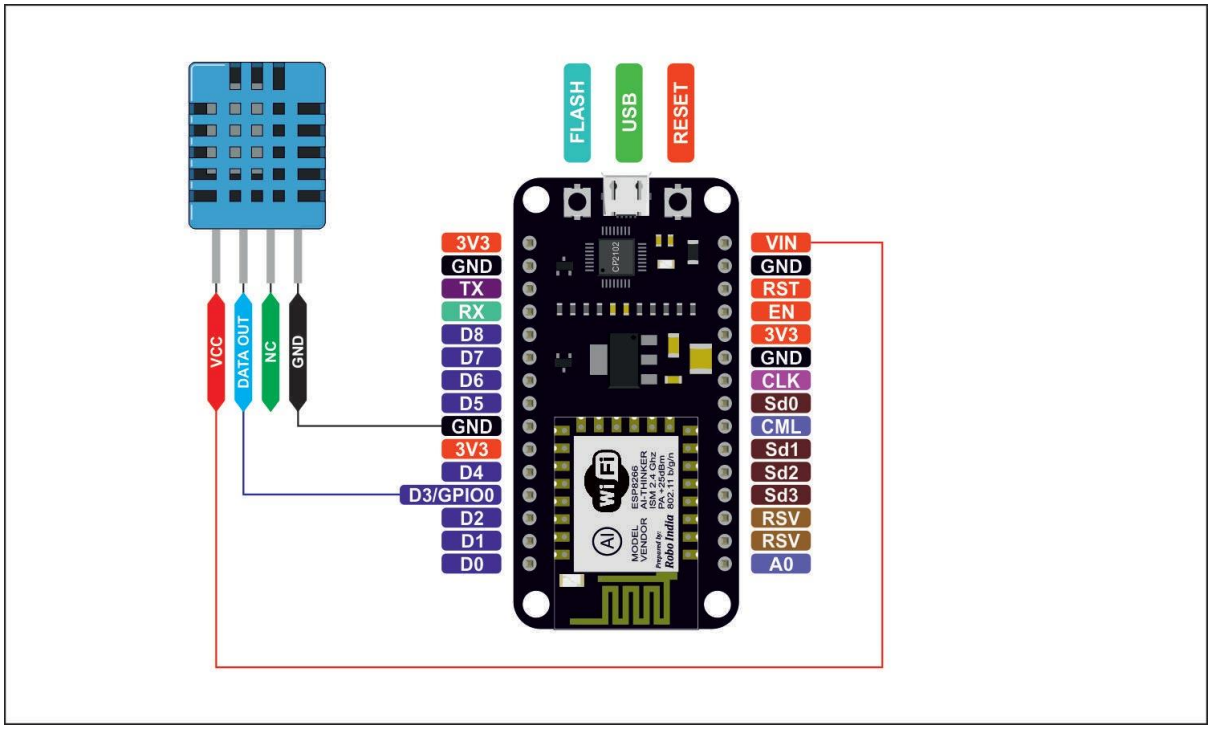
This sensor is used in various applications such as measuring humidity and temperature values in heating, ventilation and air conditioning systems. Weather stations also use these sensors to predict weather conditions. The humidity sensor is used as a preventive measure in homes where people are affected by humidity. Offices, cars, museums, greenhouses and industries use this sensor for measuring humidity values and as a safety measure.

It's compact size and sampling rate made this sensor popular among hobbyists. Some of the sensors which can be used as an alternative to DHT11 sensor are DHT22, AM2302, SHT71.

MQTT protocol (Message Queuing Telemetry Transport)

MQTT stands for Message Queuing Telemetry Transport. MQTT is a machine-to-machine internet of things connectivity protocol. It is an extremely lightweight and publish-subscribe messaging transport protocol. This protocol is useful for the connection with the remote location where the bandwidth is a premium. These characteristics make it useful in various situations, including constant environment such as for communication machine to machine and internet of things contexts. It is a publish and subscribe system where we can publish and receive the messages as a client. It makes it easy for communication between multiple devices. It is a simple messaging protocol designed for the constrained devices and with low bandwidth, so it's a perfect solution for the internet of things applications.

Connections:



Procedure:

- Connect the DHT11 sensor with nodeMCU, VCC- 3V3, GND -GND, DATA OUT – GPIO 0 (as per code)
- Write code in Arduino IDE
- Select nodeMCU board and Port in Tools
- Compile and upload the code

Sketch:

```
#include "DHT.h"        // including the library of DHT11 temperature and humidity sensor

#define DHTTYPE DHT11   // DHT 11

#define dht_dpin 0
```

```
DHT dht(dht_dpin, DHTTYPE);
void setup(void)
{
  dht.begin();
  Serial.begin(9600);
  Serial.println("Humidity and temperature\n\n");
  delay(700);
}
void loop() {
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    Serial.print("Current humidity = ");
    Serial.print(h);
    Serial.print("%  ");
    Serial.print("temperature = ");
    Serial.print(t);
    Serial.println("C  ");
  delay(800);
}
```

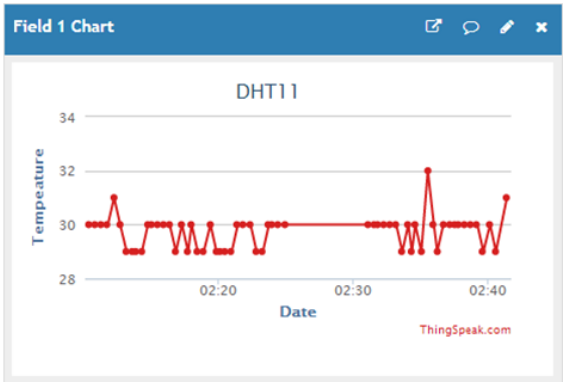
Thingspeak

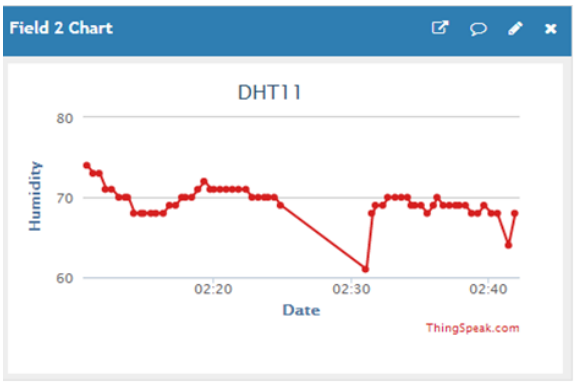
```
// Simple code upload the tempeature and humidity data using thingspeak.com
// Hardware: NodeMCU,DHT11
#include <DHT.h>  // Including library for dht
#include <ESP8266WiFi.h>
String apiKey = " API KEY";     //  Enter your Write API key from ThingSpeak
const char *ssid =  "WIFI NAME";     // replace with your wifi ssid and wpa2 key
const char *pass =  "PASSWORD";
const char* server = "api.thingspeak.com";
#define DHTPIN 0         //pin where the dht11 is connected
DHT dht(DHTPIN, DHT11);
WiFiClient client;
void setup()
{
      Serial.begin(115200);
      delay(10);
```

```
    dht.begin();
     Serial.println("Connecting to ");
    Serial.println(ssid);
     WiFi.begin(ssid, pass);
    while (WiFi.status() != WL_CONNECTED)
  {
       delay(500);
       Serial.print(".");
  }
   Serial.println("");
   Serial.println("WiFi connected");
}

void loop()
{
     float h = dht.readHumidity();
   float t = dht.readTemperature();
            if (isnan(h) || isnan(t))
          {
             Serial.println("Failed to read from DHT sensor!");
              return;
          }
              if (client.connect(server,80))   //   "184.106.153.149" or api.thingspeak.com
             {
                 String postStr = apiKey;
                  postStr +="&field1=";
                  postStr += String(t);
                  postStr +="&field2=";
                  postStr += String(h);
                  postStr += "\r\n\r\n";

                  client.print("POST /update HTTP/1.1\n");
                  client.print("Host: api.thingspeak.com\n");
                  client.print("Connection: close\n");
                  client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
```

```
                client.print("Content-Type: application/x-www-form-urlencoded\n");
                client.print("Content-Length: ");
                client.print(postStr.length());
                client.print("\n\n");
                client.print(postStr);

                Serial.print("Temperature: ");
                Serial.print(t);
                Serial.print(" degrees Celcius, Humidity: ");
                Serial.print(h);
                Serial.println("%. Send to Thingspeak.");
              }
     client.stop();
      Serial.println("Waiting...");
  // thingspeak needs minimum 15 sec delay between updates, i've set it to 30 seconds
 delay(10000);
}
```

Result: Sensor detects and display the current humidity and temperature





WEEK 7:

5. Design and Implementation of an IOT ECG (Electrocardiogram) system to record hearts electrical activity

AIM: To implement an IOT ECG System to record hearts electrical activity.

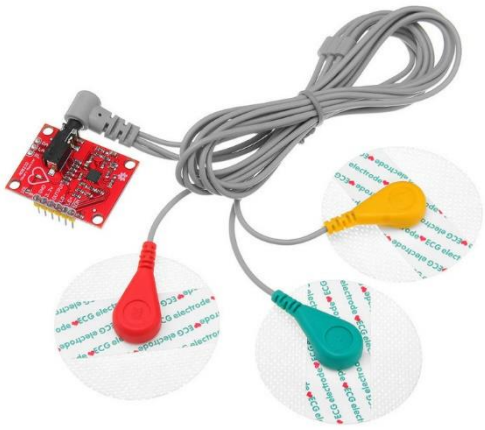
Components Required:

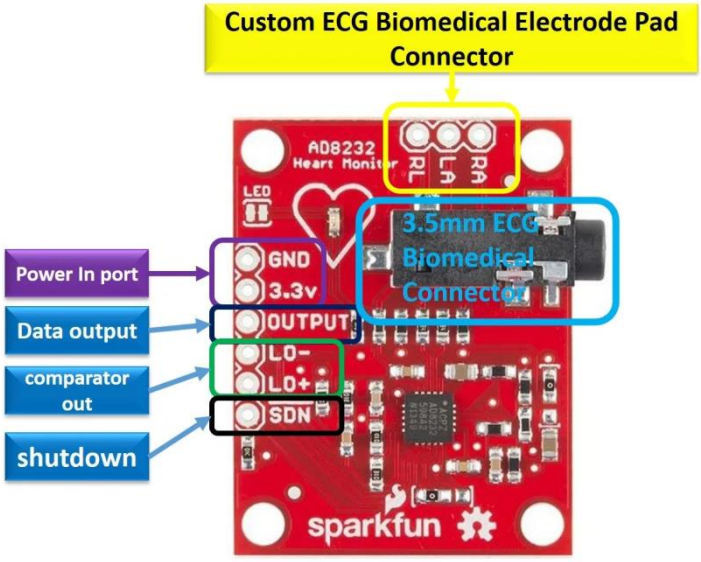
- Arduino Uno / Mega / Nano
- ECG module (AD8232)
- ECG electrode - 3 pieces
- ECG electrode connector - 3.5 mm
- DATA Cable
- Jumper Wires

ECG (AD8232):

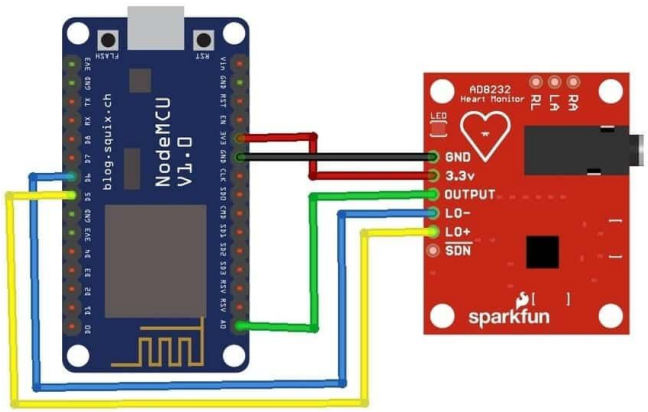
This sensor is a cost-effective board used to measure the electrical activity of the heart. This electrical activity can be charted as an ECG or Electrocardiogram and output as an analog reading. ECGs can be extremely noisy, the AD8232 Single Lead Heart Rate Monitor acts as an op-amp to help obtain a clear signal from the PR and QT Intervals easily.

The AD8232 is an integrated signal conditioning block for ECG and other biopotential measurement applications. It is designed to extract, amplify, and filter small biopotential signals in the presence of noisy conditions, such as those created by motion or remote electrode placement



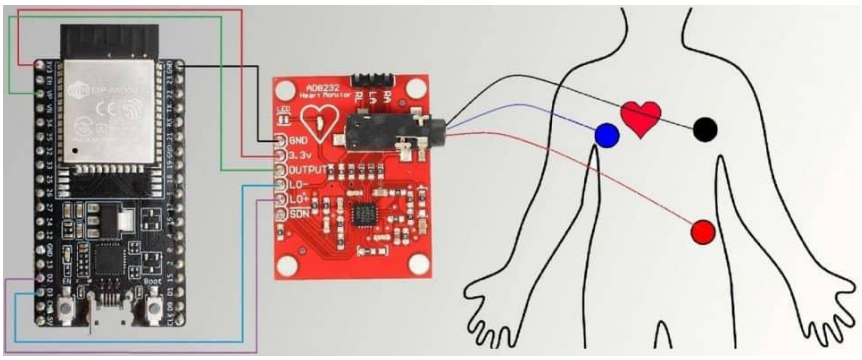


Connections:



- Power Input Pins (3.3, GND)
- Electrode pad connector pins (RA, LA, RL, 3.5 mm female jack)
- Data output pin (Output)
- Leads off detection output pins(LO-, LO+)
- Shutdown control pin(~SND)
- LED

The module AD8232 uses AD8232 analog IC. AD8232 IC is the main component of this ECG module. This chip performs three functions on small bi-potential signals in noisy conditions, such as extraction, amplification, and filtration. The final filtered signal provides a plot of an Electrocardiogram (ECG) of the heart, representing the heart's electrical activity.



Procedure:

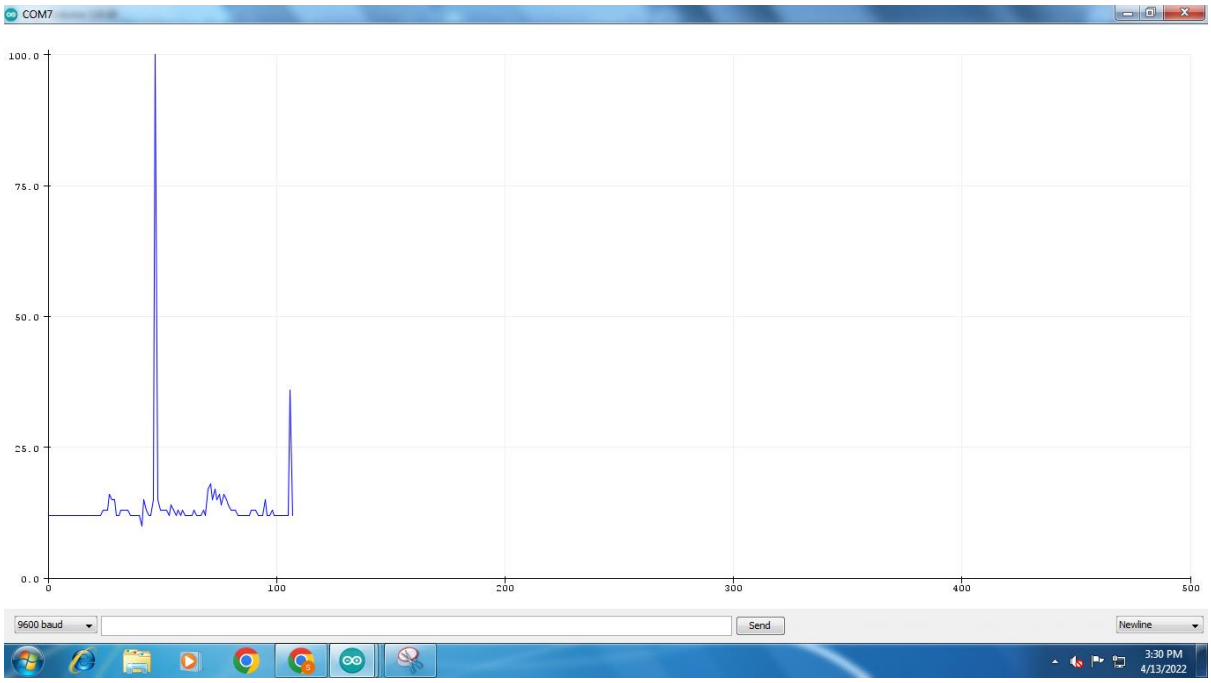
- Connect the DHT11 sensor with nodeMCU, VCC- 3V3, GND -GND, DATA OUT – GPIO 0 (as per code)
- Write code in Arduino IDE
- Select nodeMCU board and Port in Tools
- Compile and upload the code

Sketch:

```
void setup() {

// initialize the serial communication:

Serial.begin(9600);

pinMode(10, INPUT); // Setup for leads off detection LO +

pinMode(11, INPUT); // Setup for leads off detection LO -
```

```
 }
 void loop() {
 if((digitalRead(10) == 1)||(digitalRead(11) == 1)){
Serial.println('!');
}
else{
// send the value of analog input 0:
Serial.println(analogRead(A0));
}
//Wait for a bit to keep serial data from saturating
delay(1);
```

Output:



Result: The IOT ECG recorded hearts electrical activity.

WEEK 8:

6. Design and Simulate controlling on LED 7 segment Display with RASPBERRY PI

AIM: To control an LED 7 segment display with RASPBERRY PI.

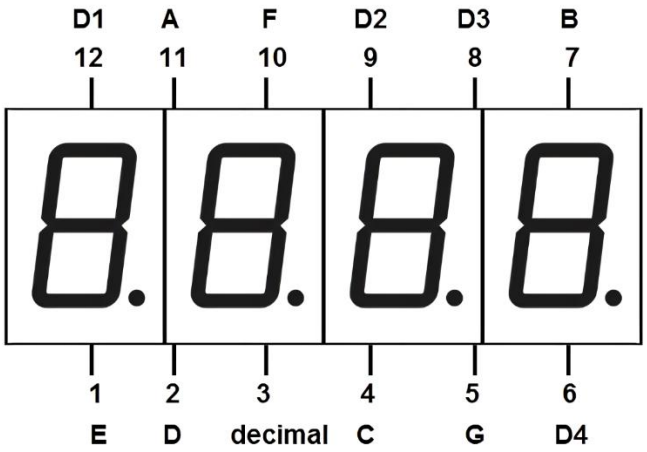
Components Required:

- Raspberry Pi3 Model B's with Installed Raspbian
- 8GB microSD cards
- Internet connection (Wired or Wireless) to access Pi Desktop
- VNC client on a wired or wireless device
- Breadboard
- Jumper Wires
- 5V Power Supply or USB
- 1 × seven-segment display
- 1 × USB cable
- 4 Digit -7 segment Display

4 digit-7 Segment Display:

7 Segment Display has seven segments in it and each segment has one LED inside it to display the numbers by lighting up the corresponding segments. Like if you want the 7-segment to display the number "5" then you need to glow segments a,f,g,c, and d by making their corresponding pins high. There are two types of 7-segment displays: Common Cathode and Common Anode, here we are using Common Cathode seven-segment display

Pin Diagram:



Raspberry Pi:

The Raspberry Pi is a low-cost, credit-card-sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

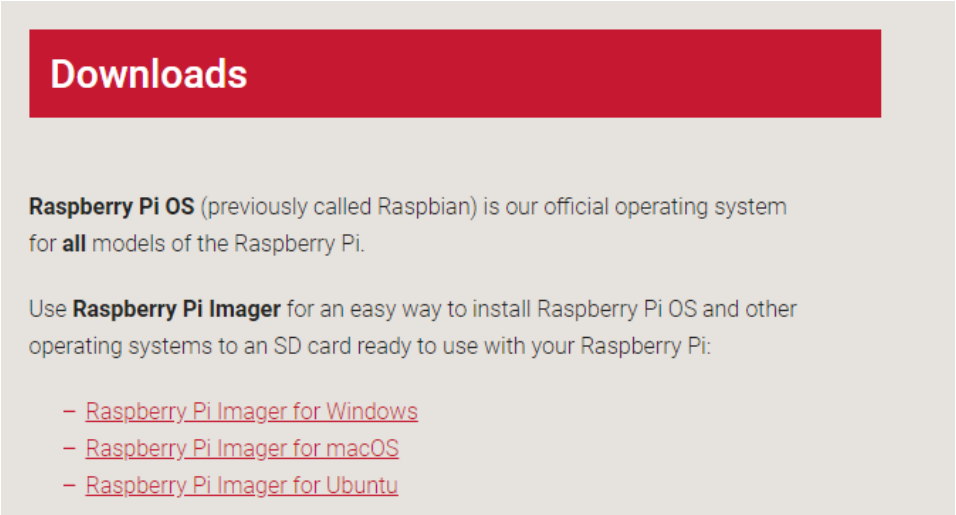
Installation of Raspbian OS

- Install Raspberry Pi OS on your SD card with the Raspberry Pi Imager
- Many vendors sell SD cards with a simple Raspberry Pi OS installer called NOOBS preinstalled but you can really easily install Raspberry Pi OS yourself using a computer that has an SD card port or using an SD card reader.
- Using the Raspberry Pi Imager is the easiest way to install Raspberry Pi OS on your SD card.

- Note: More advanced users looking to install a particular operating system should use this guide to installing operating system images.

Download and launch the Raspberry Pi Imager

- Visit the Raspberry Pi downloads page.
- Click on the link for the Raspberry Pi Imager that matches your operating system.



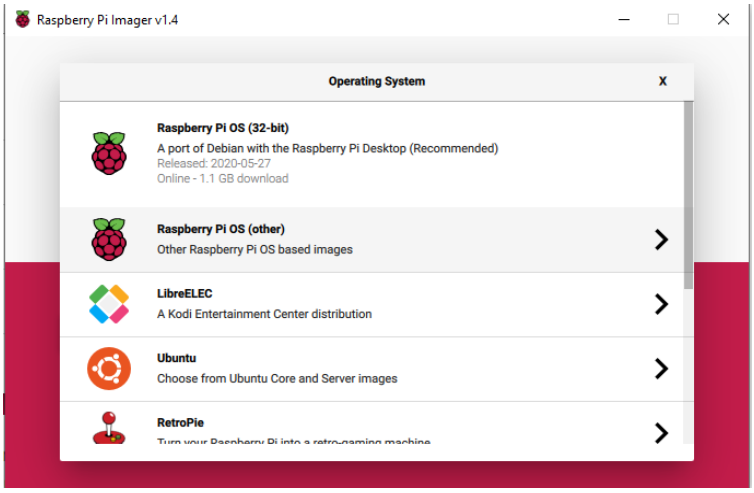
- When the download finishes, click on it to launch the installer.

Using the Raspberry Pi Imager

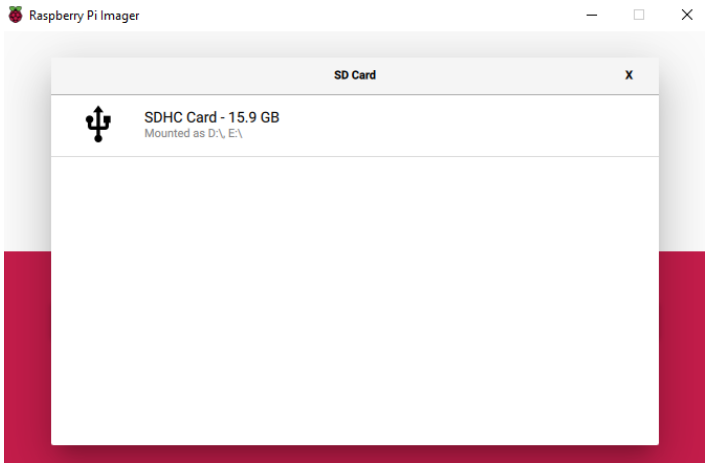
All data stored on the SD card will be overwritten during formatting and lost permanently, so make sure that you back up the card or any files, you want to keep before running the installer.

When you launch the installer, your operating system may try to block you from running it. For example, Windows may give the following message:

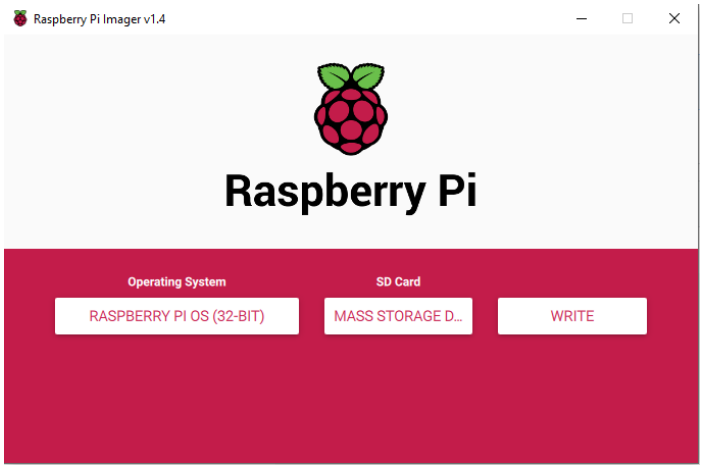
- Insert your SD card into the computer or laptop's SD card slot.
- In the Raspberry Pi Imager, select the OS that you want to install. The first option, Raspberry Pi O, is the recommended OS.



- Select the SD card you would like to install it on. Different platforms will display the drives in different ways. Mac OS, for example, will show you all drives including your main operating system.
- Note: Make sure you are selecting the correct drive. The drive's memory capacity can be a useful indication of which drive you are selecting.

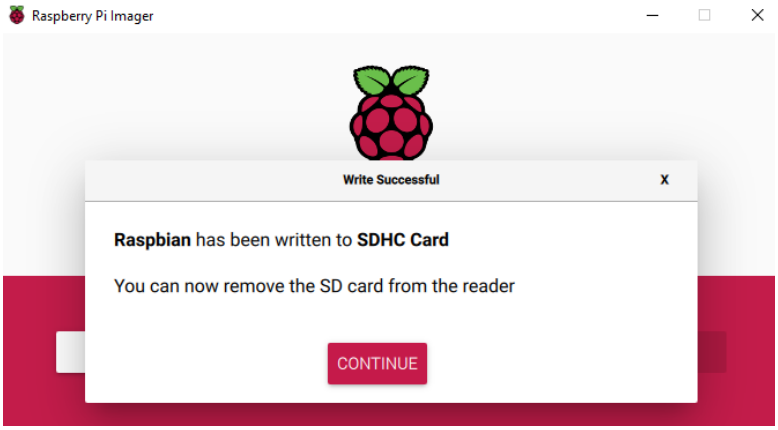


- Once you have selected both the OS and the SD card, a new WRITE button will appear.

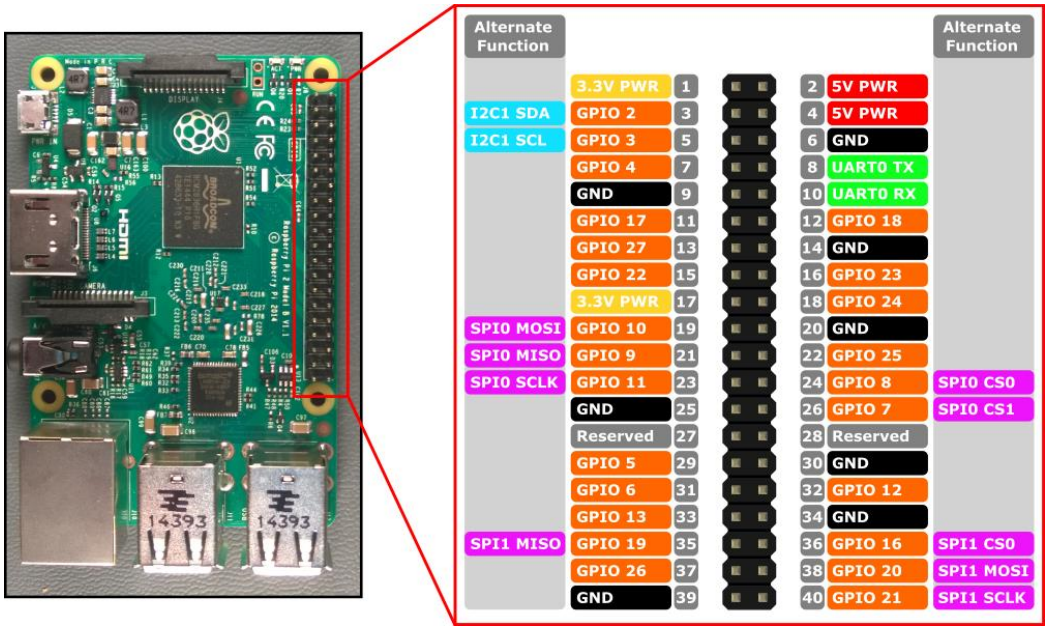


- Then simply click the WRITE button.
- Wait for the Raspberry Pi Imager to finish writing.

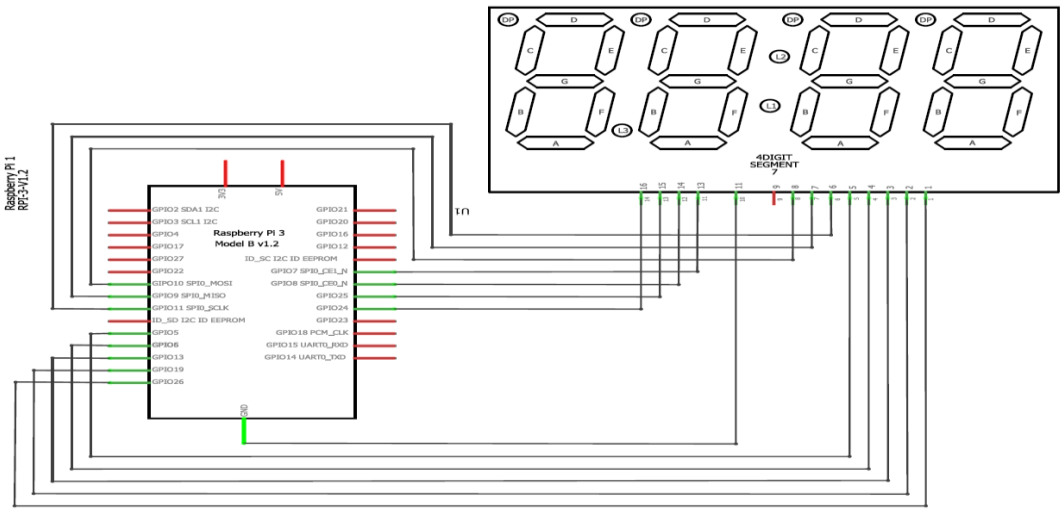
- Once you get the following message, you can eject your SD card.



Raspberry Pi Pin Diagram:

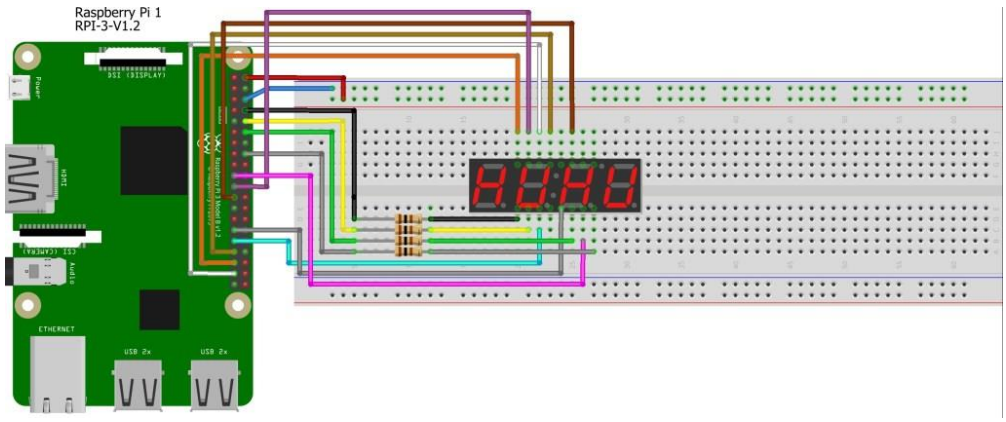


Circuit Diagram:



S.No	Rsp Pi GPIO number	Rsp Pi PIN number	7-Segment name	7-Seg pin number (here in this module)
1	GPIO 26	PIN 37	Segment a	1
2	GPIO 19	PIN 35	Segment b	2
3	GPIO 13	PIN 33	Segment c	3
4	GPIO 6	PIN 31	Segment d	4
5	GPIO 5	PIN 29	Segment e	5
6	GPIO 11	PIN 23	Segment f	6
7	GPIO 9	PIN 21	Segment g	7
8	GPIO 10	PIN 19	Segment DP	8
9	GPIO 7	PIN 26	Digit 1	13
10	GPIO 8	PIN 24	Digit 2	14
11	GPIO 25	PIN 22	Digit 3	15
12	GPIO 24	PIN 18	Digit 4	16
13	Ground	Ground	Ground	11

Connections:



Procedure:

- Connect Raspberry Pi monitor
- Connect key board, mouse, internet to Raspberry pi
- Open Raspbian os – programming- Python IDE
- Write the code in Python IDE
- Install necessary libraries and run

Source Code:

time.py

```
import sys
import time
import datetime
import RPi.GPIO as GPIO
import tm1637

Display = tm1637.TM1637(23,24,tm1637.BRIGHT_TYPICAL)
Display.Clear()
Display.SetBrightnes(1)

while(Time):
now = datetime.datetime.now()
hour = now.hour
minute = now.minute
second = now.second
Display.Clear()
val = [(int(hour/10)), (hour % 10), (int(minute / 10)), (minute % 10)]
Display.Show(val)
Display.ShowDoublepoint((second %2))
time.sleep(0.25)
```

Save the below file along with time.py in the same folder

tm1637.py

MicroPython TM1637 quad 7-segment LED display driver

```
import subprocess
from time import time, sleep, localtime
from wiringpi import wiringPiSetupGpio, pinMode, digitalRead, digitalWrite, GPIO

wiringPiSetupGpio()
```

```
TM1637_CMD1 = 0x40 # 0x40 data command

TM1637_CMD2 = 0xc0 # 0xC0 address command

TM1637_CMD3 = 0x80 # 0x80 display control command

TM1637_DSP_ON = 0x08 # 0x08 display on

TM1637_DELAY = 0.00000001 # 10us delay between clk/dio pulses

TM1637_MSB = 0x80  # msb is the decimal point or the colon depending on your display


# 0-9, a-z, blank, dash, star

_SEGMENTS =
bytearray(b'\x3F\x06\x5B\x4F\x66\x6D\x7D\x07\x7F\x6F\x77\x7C\x39\x5E\x79\x71\x3D\x76\x06\x1E\x76\x38\x55\x54\x
3F\x73\x67\x50\x6D\x78\x3E\x1C\x2A\x76\x6E\x5B\x00\x40\x63')


class TM1637(object):

    """Library for quad 7-segment LED modules based on the TM1637 LED driver."""

    def __init__(self, clk, dio, brightness=7):

        self.clk = clk

        self.dio = dio


        if not 0 <= brightness <= 7:

            raise ValueError("Brightness out of range")

        self._brightness = brightness


        pinMode(self.clk, GPIO.INPUT)

        pinMode(self.dio, GPIO.INPUT)

        digitalWrite(self.clk, 0)

        digitalWrite(self.dio, 0)


        sleep(TM1637_DELAY)


        self._write_data_cmd()

        self._write_dsp_ctrl()


    def _start(self):

        pinMode(self.dio, GPIO.OUTPUT)

        sleep(TM1637_DELAY)

        pinMode(self.clk, GPIO.OUTPUT)
```

```
    sleep(TM1637_DELAY)

  def _stop(self):
    pinMode(self.dio, GPIO.OUTPUT)
    sleep(TM1637_DELAY)
    pinMode(self.clk, GPIO.INPUT)
    sleep(TM1637_DELAY)
    pinMode(self.dio, GPIO.INPUT)
    sleep(TM1637_DELAY)

  def _write_data_cmd(self):
    # automatic address increment, normal mode
    self._start()
    self._write_byte(TM1637_CMD1)
    self._stop()

  def _write_dsp_ctrl(self):
    # display on, set brightness
    self._start()
    self._write_byte(TM1637_CMD3 | TM1637_DSP_ON | self._brightness)
    self._stop()

  def _write_byte(self, b):
    for i in range(8):
      pinMode(self.clk, GPIO.OUTPUT)
      sleep(TM1637_DELAY)
      pinMode(self.dio, GPIO.INPUT if b & 1 else GPIO.OUTPUT)
      sleep(TM1637_DELAY)
      pinMode(self.clk, GPIO.INPUT)
      sleep(TM1637_DELAY)
      b >>= 1
    pinMode(self.clk, GPIO.OUTPUT)
    sleep(TM1637_DELAY)
    pinMode(self.clk, GPIO.INPUT)
    sleep(TM1637_DELAY)
```

```
    pinMode(self.clk, GPIO.OUTPUT)
    sleep(TM1637_DELAY)

  def brightness(self, val=None):
    """Set the display brightness 0-7."""
    # brightness 0 = 1/16th pulse width
    # brightness 7 = 14/16th pulse width
    if val is None:
      return self._brightness
    if not 0 <= val <= 7:
      raise ValueError("Brightness out of range")

    self._brightness = val
    self._write_data_cmd()
    self._write_dsp_ctrl()

  def write(self, segments, pos=0):
    """Display up to 6 segments moving right from a given position.
    The MSB in the 2nd segment controls the colon between the 2nd
    and 3rd segments."""
    if not 0 <= pos <= 5:
      raise ValueError("Position out of range")
    self._write_data_cmd()
    self._start()

    self._write_byte(TM1637_CMD2 | pos)
    for seg in segments:
      self._write_byte(seg)
    self._stop()
    self._write_dsp_ctrl()

  def encode_digit(self, digit):
    """Convert a character 0-9, a-f to a segment."""
    return _SEGMENTS[digit & 0x0f]
```

```
  def encode_string(self, string):
    """Convert an up to 4 character length string containing 0-9, a-z,
    space, dash, star to an array of segments, matching the length of the
    source string."""
    segments = bytearray(len(string))
    for i in range(len(string)):
      segments[i] = self.encode_char(string[i])
    return segments

  def encode_char(self, char):
    """Convert a character 0-9, a-z, space, dash or star to a segment."""
    o = ord(char)
    if o == 32:
      return _SEGMENTS[36] # space
    if o == 42:
      return _SEGMENTS[38] # star/degrees
    if o == 45:
      return _SEGMENTS[37] # dash
    if o >= 65 and o <= 90:
      return _SEGMENTS[o-55] # uppercase A-Z
    if o >= 97 and o <= 122:
      return _SEGMENTS[o-87] # lowercase a-z
    if o >= 48 and o <= 57:
      return _SEGMENTS[o-48] # 0-9
    raise ValueError("Character out of range: {:d} '{:s}'".format(o, chr(o)))

  def hex(self, val):
    """Display a hex value 0x0000 through 0xffff, right aligned."""
    string = '{:04x}'.format(val & 0xffff)
    self.write(self.encode_string(string))

  def number(self, num):
    """Display a numeric value -999 through 9999, right aligned."""
    # limit to range -999 to 9999
    num = max(-999, min(num, 9999))
```

```
        string = '{0: >4d}'.format(num)
        self.write(self.encode_string(string))

    def numbers(self, num1, num2, colon=True):
        """Display two numeric values -9 through 99, with leading zeros
        and separated by a colon."""
        num1 = max(-9, min(num1, 99))
        num2 = max(-9, min(num2, 99))
        segments = self.encode_string('{0:0>2d}{1:0>2d}'.format(num1, num2))
        if colon:
            segments[1] |= 0x80 # colon on
        self.write(segments)

    def temperature(self, num):
        if num < -9:
            self.show('lo') # low
        elif num > 99:
            self.show('hi') # high
        else:
            string = '{0: >2d}'.format(num)
            self.write(self.encode_string(string))
        self.write([_SEGMENTS[38], _SEGMENTS[12]], 2) # degrees C

    def show(self, string, colon=False):
        segments = self.encode_string(string)
        if len(segments) > 1 and colon:
            segments[1] |= 128
        self.write(segments[:4])

    def scroll(self, string, delay=250):
        segments = string if isinstance(string, list) else self.encode_string(string)
        data = [0] * 8
        data[4:0] = list(segments)
        for i in range(len(segments) + 5):
            self.write(data[0+i:4+i])
```

```
        sleep(delay / 1000)



class TM1637Decimal(TM1637):
    """Library for quad 7-segment LED modules based on the TM1637 LED driver.

    This class is meant to be used with decimal display modules (modules
    that have a decimal point after each 7-segment LED).
    """
    def encode_string(self, string):
        """Convert a string to LED segments.

        Convert an up to 4 character length string containing 0-9, a-z,
        space, dash, star and '.' to an array of segments, matching the length of
        the source string."""
        segments = bytearray(len(string.replace('.','')))
        j = 0
        for i in range(len(string)):
            if string[i] == '.' and j > 0:
                segments[j-1] |= TM1637_MSB
                continue
            segments[j] = self.encode_char(string[i])
            j += 1
        return segments
```

Result: It displays the current time on 4 digit 7 segment display