**Siddaganga Institute of Technology, Tumakuru**

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

**A Report on Project titled**

**“SIMPLE HEALTH MONITORING SYSTEM”**

submitted

*in the partial fulfillment of the requirements for IV semester*

*Bachelor of Engineering*

In

*Electronics and Communication*

by

**ARAVIND V A – 1SI20EC011**

**NIKHIL K G – 1SI20EC059**

**HARSHA S N – 1SI20EC033**

**SUNIL DANAPPA BIRADAR – 1SI20EC097**

**RUDRESH A N - 1SI20EC077**

**BHARATH GOWDA N – 1SI20EC013**



**Department Of Electronics and Communication**

**Siddaganga Institute of Technology**

B.H Road, Tumakuru-572 103, Karnataka, India.

**Siddaganga Institute of Technology, Tumakuru**

(An Autonomous Institution affiliated to Visvesvaraya Technological University, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified)

**Department of** **Electronics and Communication**



CERTIFICATE

This is to certify that project titled **“PATIENT HEALTH MONITORING SYSTEM”** is a bonafide work carried out by **ARAVIND V A (1SI20EC011), NIKHIL K G (1SI20EC059), HARSHA S N (1SI20EC033), SUNIL DANAPPA BIRADAR (1SI20EC097), RUDRESH A N (1SI20EC077), BHARATH GOWDA N (1SI20EC013)** of IV semester **Bachelor of Engineering in Electronics and Communication** of the **SIDDAGANGA INSTITUTE OF TECHNOLOGY** (An Autonomous Institution, affiliated to VTU, Belagavi, Approved by AICTE, New Delhi, Accredited by NAAC and ISO 9001:2015 certified) during the academic year 2021-2022.

**Name of the Panel Members Signature with Date**

**1. SAIKIRAN SINGARAO**

**2. BANDENAWAZ BAGWAN**

**3. MARIPI SRAVAN KUMAR**

**TABLE OF CONTENTS:**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Contents** | **Page No.** |
| 1 | **INTRODUCTION TO RASPBERRY**  **PI 3** | 06 |
| 2 | **DS18B20 TEMPERATURE SENSOR** | 08 |
| 3 | **MCP3008 ADC** | 10 |
| 4 | **PULSE SENSOR OR HEART BEAT SENSOR** | 12 |
| 5 | **CIRCUIT DIAGRAM** | 13 |
| 6 | **MQTT** | 15 |
| 7 | **WORKING** | 16 |
| 8 | **CODE FOR SENSORS AND ADC** | 22 |
| 9 | **CONCLUSION** | 23 |

# Raspberry Pi 3

Table

Description automatically generatedA picture containing text, electronics, circuit

Description automatically generated

**RASPBERRY PI 3** is a development board in PI series. It can be considered as a single board computer that works on LINUX operating system. The board not only has tons of features it also has terrific processing speed making it suitable for advanced applications. PI board is specifically designed for hobbyist and engineers who are interested in LINUX systems and IoT (Internet of Things).

# Where RASPBERRY PI 3 is Used?

**Raspberry pi** platform is most used after [ADRUINO](https://components101.com/microcontrollers/arduino-uno). Although overall applications of PI are less it is most preferred when developing advanced applications. Also the RASPBERRY PI is an open source platform where one can get a lot of related information so you can customize the system depending on the need.

Here are few examples where RASPBERRY PI 3 is chosen over other microcontrollers and [development boards](https://components101.com/development-boards):

1. Where the system processing is huge. Most ARDUINO boards all have clock speed of less than 100MHz, so they can perform functions limited to their capabilities. They cannot process high end programs for applications like Weather Station, Cloud server, gaming console etc. With **1.2GHz clock speed** and **1 GB RAM, Raspberry pi**  can perform all those advanced functions.

2. Where wireless connectivity is needed. RASPBERRY PI 3 has wireless LAN and Bluetooth facility by which you can setup WIFI HOTSPOT for internet connectivity. For **Internet of Things** this feature is best suited.

3. **Raspberry pi**  has dedicated port for connecting touch LCD display which is a feature that completely omits the need of monitor.

4. RASPBERRY PI also has dedicated camera port so one can connect camera without any hassle to the PI board.

5. RASPBERRY PI also has PWM outputs for application use.

There are many other features like HD steaming which further promote the use of RASPBERRY PI.

**2.DS18B20 TEMPERATURE SENSOR**

# ****DS18B20 Sensor Specifications****

* Programmable Digital Temperature Sensor
* Communicates using 1-Wire method
* Operating voltage: 3V to 5V
* Temperature Range: -55°C to +125°C
* Accuracy: ±0.5°C
* Output Resolution: 9-bit to 12-bit (programmable)
* Unique 64-bit address enables multiplexing
* Conversion time: 750ms at 12-bit
* Programmable alarm options
* Available as To-92, SOP and even as a waterproof sensor

# ****Pin Configuration:****

|  |  |  |
| --- | --- | --- |
| **No:** | **Pin Name** | **Description** |
| **1** | **Ground** | Connect to the ground of the circuit |
| **2** | **Vcc** | Powers the Sensor, can be 3.3V or 5V |
| **3** | **Data** | This pin gives output the temperature value which can be read using 1-wire method |

# ****Where to use DS18B20 Sensor****

The **DS18B20** is a 1-wire programmable Temperature sensor from maxim integrated. It is widely used to measure temperature in hard environments like in chemical solutions, mines or soil etc. The constriction of the sensor is rugged and also can be purchased with a waterproof option making the mounting process easy. It can measure a wide range of temperature from **-55°C to +125°** with a decent accuracy of **±5°C**. Each sensor has a unique address and requires only one pin of the MCU to transfer data so it a very good choice for measuring temperature at multiple points without compromising much of your digital pins on the microcontroller.

A picture containing text

Description automatically generated

**3.MCP3008 ADC**

**Features:**

# • 10-bit resolution.

# • ± 1 LSB max DNL.

# • ± 1 LSB max INL.

# • 4 (MCP3004) or 8 (MCP3008) input channels.

# • Analog inputs programmable as single-ended or pseudo-differential pairs.

# • On-chip sample and hold.

# • SPI serial interface (modes 0,0 and 1,1).

# • Single supply operation: 2.7V - 5.5V.

# • 200 ksps max. sampling rate at VDD = 5V.

# • 75 ksps max. sampling rate at VDD = 2.7V.

# • Low power CMOS technology.

# • 5 nA typical standby current, 2 µA max.

# • 500 µA max. active current at 5V.

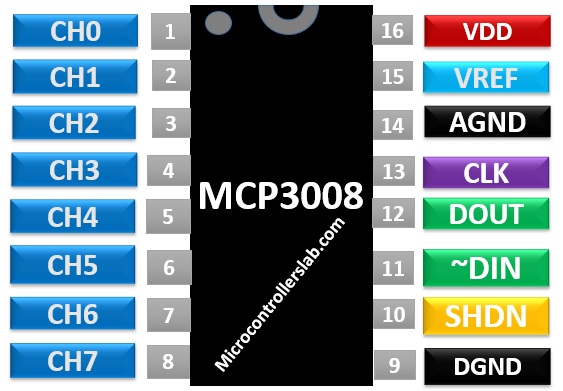
# • Industrial temp range: -40°C to +85°C.

# • Available in PDIP, SOIC and TSSOP packages.

## ****Where to use it?****

There are some devices like Raspberry pi which don’t have hardware for analog to digital converter and therefore they can’t read analog inputs. So, you need a circuit for this conversion. For such devices, you can use the **MCP3008** chip. This chip uses an SPI interface for communication. In Raspberry Pi, only four GPIO pins are required. So, you can get 8 additional analog inputs by using this chip.

Sensors use analog outputs. Therefore, many devices need an ADC converter to read these outputs. The MCP3008 can be used for converting these analog signals into digital signals.

****

**4.PULSE SENSOR OR HEART BEAT SENSOR**

# ****Pulse Sensor Features and Specifications****

* Biometric Pulse Rate or Heart Rate detecting sensor
* Plug and Play type sensor
* Operating Voltage: +5V or +3.3V
* Current Consumption: 4mA
* Inbuilt Amplification and Noise cancellation circuit.
* Diameter: 0.625”
* Thickness: 0.125” Thick

# ****Pin Configuration****

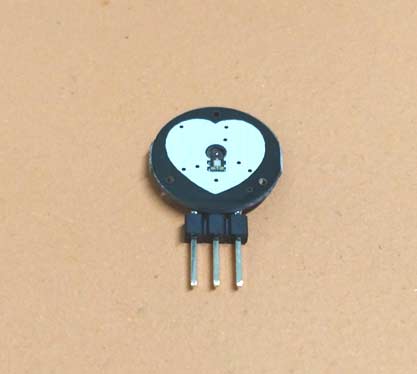
|  |  |  |  |
| --- | --- | --- | --- |
| Pin number | Pin name | ****Wire Colour**** | ****Description**** |
| 1. | Ground | Black | Connected to the ground of the system |
| 2. | VCC | Red | Connect to +5V or +3.3V supply voltage |
| 3. | Signal | Purple | Pulsating output signal. |

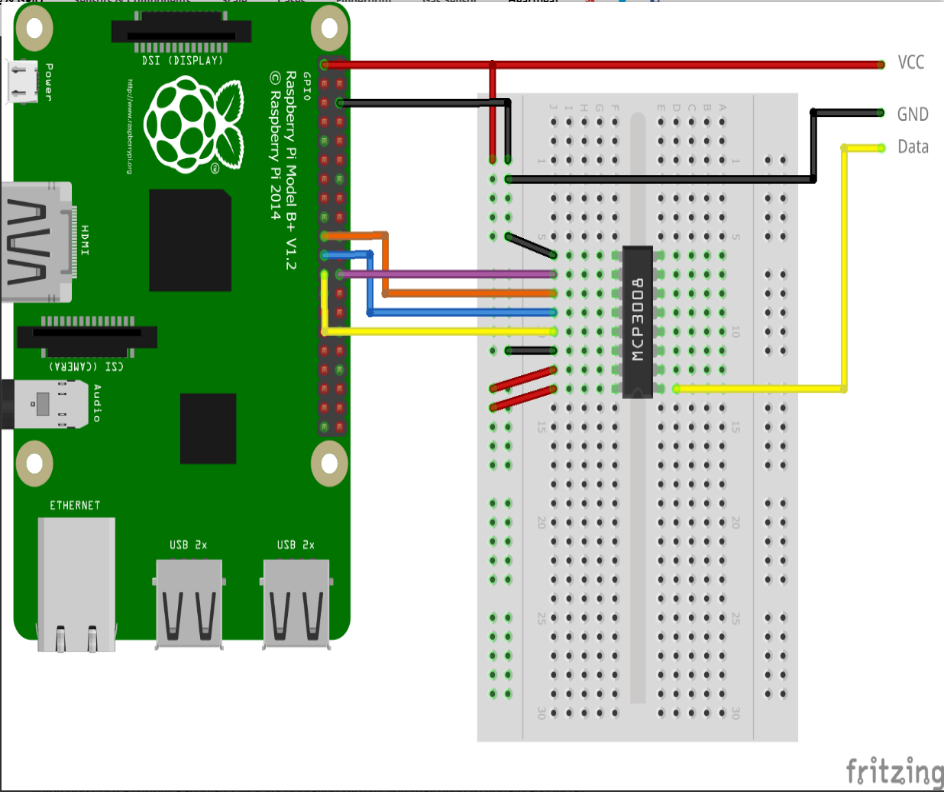
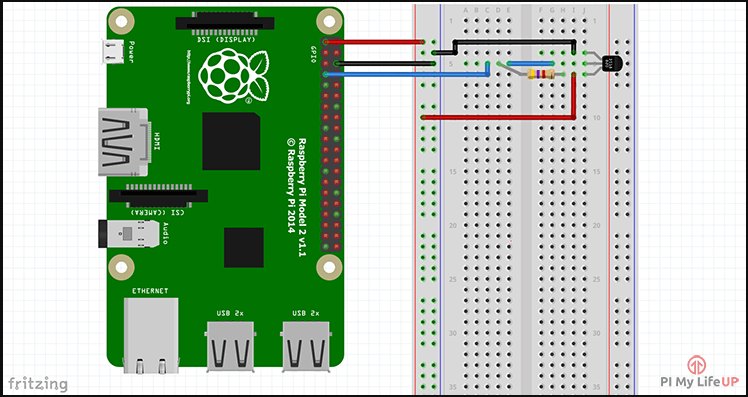
# ****How Pulse Sensor Works****

The working of the **Pulse/Heart beat sensor** is very simple. The sensor has two sides, on one side the LED is placed along with an ambient light sensor and on the other side we have some circuitry. This circuitry is responsible for the amplification and noise cancellation work. The LED on the front side of the sensor is placed over a vein in our human body. This can either be your Finger tip or you ear tips, but it should be placed directly on top of a vein.

Now the LED emits light which will fall on the vein directly. The veins will have blood flow inside them only when the heart is pumping, so if we monitor the flow of blood we can monitor the heart beats as well.  If the flow of blood is detected then the ambient light sensor will pick up more light since they will be reflect ted by the blood, this minor change in received light is analysed over time to determine our heart beats.

**Diagram

Description automatically generated with medium confidence**

** CIRCUIT DIAGRAM:**

# 1.pulse sensor

# 2.Temperature sensor

**MQTT:**

**“MQTT is a Client Server publish/subscribe messaging transport protocol. It is light weight, open, simple, and designed so as to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium.“**

# **The basic concepts (**[**publish/subscribe**](https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/)**,**[**client/broker**](https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/)**) and basic functionality (**[**Connect**](https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/)**,**[**Publish, Subscribe**](https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe/)**) of MQTT. Then, we’ll look at the features:**[**Quality of Service**](https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/)**,**[**Retained Messages**](https://www.hivemq.com/blog/mqtt-essentials-part-8-retained-messages)**,**[**Persistent Session**](https://www.hivemq.com/blog/mqtt-essentials-part-7-persistent-session-queuing-messages/)**,**[**Last Will and Testament**](https://www.hivemq.com/blog/mqtt-essentials-part-9-last-will-and-testament)**,**[**Keep Alive**](https://www.hivemq.com/blog/mqtt-essentials-part-10-alive-client-take-over)**and**[**more**](https://www.hivemq.com/blog/mqtt-essentials-special-mqtt-over-websockets)**.**

# The abstract of the MQTT specification does a good job describing what MQTT is all about. It is a very light weight and binary protocol, and due to its minimal packet overhead, MQTT ****excels when transferring data over the wire in comparison to protocols like HTTP****.

**BASIC ADVANTAGES OF MQTT CLIENT:**

* Simple implementation
* Quality of Service data delivery
* Lightweight and bandwidth efficient
* Data agnostic
* Continuous session awareness

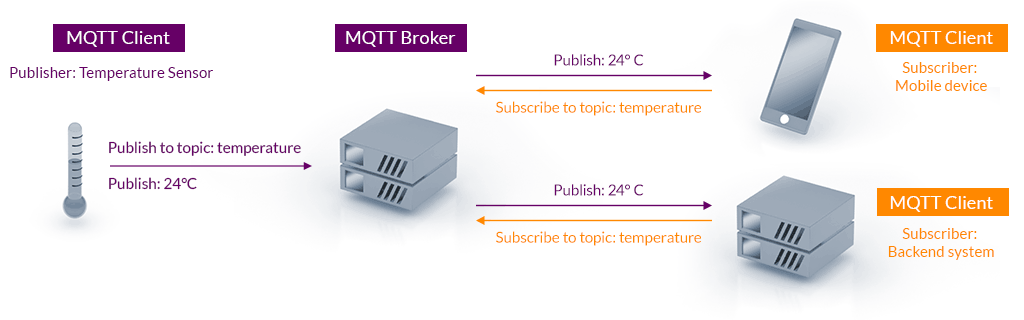
## 

## 

**KEY FEATURES:**

* MQTT decouples the publisher and subscriber spatially. To publish or receive messages, publishers and subscribers only need to know the hostname/IP and port of the broker.
* MQTT decouples by time. Although most MQTT use cases deliver messages in near-real time, if desired, the broker can store messages for clients that are not online. (Two conditions must be met to store messages: the client had connected with a persistent session and subscribed to a topic with a [**Quality of Service**](https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/) greater than 0).
* Publish/Subscribe decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers)
* Broker: The counterpart of the MQTT client is the MQTT broker. The broker is at the heart of any publish/subscribe protocol. Depending on the implementation, a broker can handle up to millions of concurrently connected MQTT clients.

## MQTT Publish / Subscribe Architecture



**WORKING:**

* **Raspberry pi3 is used as medium that interacts with both sensors and output device (App).**
* **The temperature sensor does not require ADC, however it requires resistor connected across Vcc and sensor output.**
* **The temperature sensor output is connected to one of the GPIO pins of Raspberry pi, for communication it uses only one wired interface.**
* **Pulse sensor communicates through ADC(MCP3008), the data pin is connected to channel 0 of MCP3008.**
* **The analog output from Pulse sensor is converted into digital through ADC and interfaced with Raspberry pi.**
* **ADC(MCP3008) uses SAR(Successive approximation) technique to get accurate pulse readings.**
* **Raspberry pi communicates with mobile application by using MQTT client.**
* **A MQTT broker connects Raspberry pi to the cloud and cloud to mobile.**
* **The sensor output is sent to cloud through a channel and the mobile app receives the data from subscribing the channel that contains the sensor output.**
* **To collect the data for ML code is written to include the data into a csv file created locally.**
* **The csv file is used for various applications of ML.**

* **CODE FOR SENSORS AND ADC**

1. **CODE FOR MCP3008 ADC**

# First install spidev:

# Enable SPI (sudo raspi-config)

# $ sudo apt-get update

# $ sudo apt-get upgrade

# $ sudo apt-get install python-dev

# $ sudo reboot

# $ wget https://github.com/doceme/py-spidev/archive/master.zip

# $ unzip master.zip

# $ cd py-spidev-master

# $ sudo python setup.py install

from spidev import SpiDev

class MCP3008:

    def \_\_init\_\_(self, bus = 0, device = 0):

        self.bus, self.device = bus, device

        self.spi = SpiDev()

        self.open()

        self.spi.max\_speed\_hz = 1000000 # 1MHz

    def open(self):

        self.spi.open(self.bus, self.device)

        self.spi.max\_speed\_hz = 1000000 # 1MHz

    def read(self, channel = 0):

        cmd1 = 4 | 2 | (( channel & 4) >> 2)

        cmd2 = (channel & 3) << 6

        adc = self.spi.xfer2([cmd1, cmd2, 0])

        data = ((adc[1] & 15) << 8) + adc[2]

        return data

    def close(self):

        self.spi.close()

1. **CODE FOR PULSE SENSOR**

import time

import threading

from MCP3008 import MCP3008

class Pulsesensor:

    def \_\_init\_\_(self, channel = 0, bus = 0, device = 0):

        self.channel = channel

        self.BPM = 0

        self.adc = MCP3008(bus, device)

    def getBPMLoop(self):

        # init variables

        rate = [0] \* 10         # array to hold last 10 IBI values

        sampleCounter = 0       # used to determine pulse timing

        lastBeatTime = 0        # used to find IBI

        P = 512                 # used to find peak in pulse wave, seeded

        T = 512                 # used to find trough in pulse wave, seeded

        thresh = 525            # used to find instant moment of heart beat, seeded

        amp = 100               # used to hold amplitude of pulse waveform, seeded

        firstBeat = True        # used to seed rate array so we startup with reasonable BPM

        secondBeat = False      # used to seed rate array so we startup with reasonable BPM

        IBI = 600               # int that holds the time interval between beats! Must be seeded!

        Pulse = False           # "True" when User's live heartbeat is detected. "False" when not a "live beat".

        lastTime = int(time.time()\*1000)

        while not self.thread.stopped:

            Signal = self.adc.read(self.channel)

            currentTime = int(time.time()\*1000)

            sampleCounter += currentTime - lastTime

            lastTime = currentTime

            N = sampleCounter - lastBeatTime

            # find the peak and trough of the pulse wave

            if Signal < thresh and N > (IBI/5.0)\*3:     # avoid dichrotic noise by waiting 3/5 of last IBI

                if Signal < T:                          # T is the trough

                    T = Signal                          # keep track of lowest point in pulse wave

            if Signal > thresh and Signal > P:

                P = Signal

            # signal surges up in value every time there is a pulse

            if N > 250:                                 # avoid high frequency noise

                if Signal > thresh and Pulse == False and N > (IBI/5.0)\*3:

                    Pulse = True                        # set the Pulse flag when we think there is a pulse

                    IBI = sampleCounter - lastBeatTime  # measure time between beats in mS

                    lastBeatTime = sampleCounter        # keep track of time for next pulse

                    if secondBeat:                      # if this is the second beat, if secondBeat == TRUE

                        secondBeat = False;             # clear secondBeat flag

                        for i in range(len(rate)):      # seed the running total to get a realisitic BPM at startup

                          rate[i] = IBI

                    if firstBeat:                       # if it's the first time we found a beat, if firstBeat == TRUE

                        firstBeat = False;              # clear firstBeat flag

                        secondBeat = True;              # set the second beat flag

                        continue

                    # keep a running total of the last 10 IBI values

                    rate[:-1] = rate[1:]                # shift data in the rate array

                    rate[-1] = IBI                      # add the latest IBI to the rate array

                    runningTotal = sum(rate)            # add upp oldest IBI values

                    runningTotal /= len(rate)           # average the IBI values

                    self.BPM = 60000/runningTotal       # how many beats can fit into a minute? that's BPM!

            if Signal < thresh and Pulse == True:       # when the values are going down, the beat is over

                Pulse = False                           # reset the Pulse flag so we can do it again

                amp = P - T                             # get amplitude of the pulse wave

                thresh = amp/2 + T                      # set thresh at 50% of the amplitude

                P = thresh                              # reset these for next time

                T = thresh

            if N > 2500:                                # if 2.5 seconds go by without a beat

                thresh = 512                            # set thresh default

                P = 512                                 # set P default

                T = 512                                 # set T default

                lastBeatTime = sampleCounter            # bring the lastBeatTime up to date

                firstBeat = True                        # set these to avoid noise

                secondBeat = False                      # when we get the heartbeat back

                self.BPM = 0

            time.sleep(0.005)

    # Start getBPMLoop routine which saves the BPM in its variable

    def startAsyncBPM(self):

        self.thread = threading.Thread(target=self.getBPMLoop)

        self.thread.stopped = False

        self.thread.start()

        return

    # Stop the routine

    def stopAsyncBPM(self):

        self.thread.stopped = True

        self.BPM = 0

        return

1. **CODE FOR DS18B20 TEMPERATURE SENSOR**

import os

import glob

import time

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'

def read\_temp\_raw():

    f = open(device\_file, 'r')

    lines = f.readlines()

    f.close()

    return lines

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

        temp\_f = temp\_c \* 9.0 / 5.0 + 32.0

        return temp\_c, temp\_f

while True:

    print(read\_temp())

    time.sleep(1)

1. **CODE TO MEASURE HEART BEAT RATE FROM PULSE SENSOR**

from pulsesensor import Pulsesensor

import time

import RPi.GPIO as GPIO     # Import Library to access GPIO PIN

import time                 # To access delay function

GPIO.setmode(GPIO.BOARD)    # Consider complete raspberry-pi board

GPIO.setwarnings(False)     # To avoid same PIN use warning

count = 0

p = Pulsesensor()

p.startAsyncBPM()

try:

    file = open("log.csv","a")

    while True:

        bpm = p.BPM

        if (bpm-125) > 0:

            GPIO.output(LED\_PIN,GPIO.HIGH)

            print("BPM: %d" % (bpm-125))

            Pul\_string("Your Heart Rate ",PUL\_LINE\_1)

            Pul\_string(str(int(bpm-125)) + " BPM",PUL\_LINE\_2)

            file.write("{0:0.2f".format(read\_temp())+","+"{0:0.2f}".format(bpm-145)+",")

        else:

            count = 0

            print("No Heartbeat found")

            pul\_string("placed your",PUL\_LINE\_1)

            pul\_string("Finger on sensor",PUL\_LINE\_2)

        time.sleep(1)

        GPIO.output(LED\_PIN,GPIO.LOW)

        file.close()

        time.sleep(2)

except:

    p.stopAsyncBPM()

**CONCLUSION:**

**This is a simple project that handles with the sensors, CPU and mobile application. It is user friendly and hence any person of any age can use it. With further developments in hardware structure this device can be carried out anywhere. And mobile app is also a prototype, when developed the UI and usability of the app can be improved.**