

PROJECT REPORT
ON
“HEART RATE MONITORING”

*Submitted in partial fulfillment of the requirements of V semester
MCA – Internet Of Things (IoT) Lab, Department of MCA, Mount
Carmel College, Autonomous*

By
API YOKA (M18MC03)

Under
Guidance of
Ms. Shaila Mary J

Batch: 2018-2021

Department Of MCA
MOUNT CARMEL COLLEGE, AUTONOMOUS,
58, PALACE ROAD, BANGALORE – 560052



Mount Carmel College Autonomous

58, Palace Road, Bengaluru-52

Certificate

This is to certify that the project entitled “**Heart Rate Monitoring for the visually impaired using Arduino**” by **API YOKA (Register No: M18MC03)**, student of **MASTER OF COMPUTER APPLICATIONS**, batch [2018-2020], has successfully completed the project in **Internet of Things (IoT) Lab**, prescribed by Board Of Studies, Department of MCA, Mount Carmel College Autonomous, Bengaluru affiliated to Bangalore University and approved by AICTE for the V semester MCA - 2018.

Project Guide

Ms. Shaila Mary J

Coordinator, Department Of MCA

Ms. Vijayalakshmi. N

Date : 28/12/2020

Register Number : M18MC03

Examiners:

College Seal

1. _____

2. _____

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Ms. Shaila Mary J, for providing me an opportunity to do the project work in Mount Carmel College Autonomous and giving us all support and guidance, which made me complete the project duly. I am extremely thankful for providing such a nice support and guidance, although he had busy schedule managing the corporate affairs.

I would like to express my gratitude towards my parents, member and mentors of Mount Carmel College Autonomous for their kind co-operation and encouragement which help me in completion of this project.

My thanks and appreciations also go to my project partner in developing the project and people who have willingly helped me out with their abilities.

API YOKA

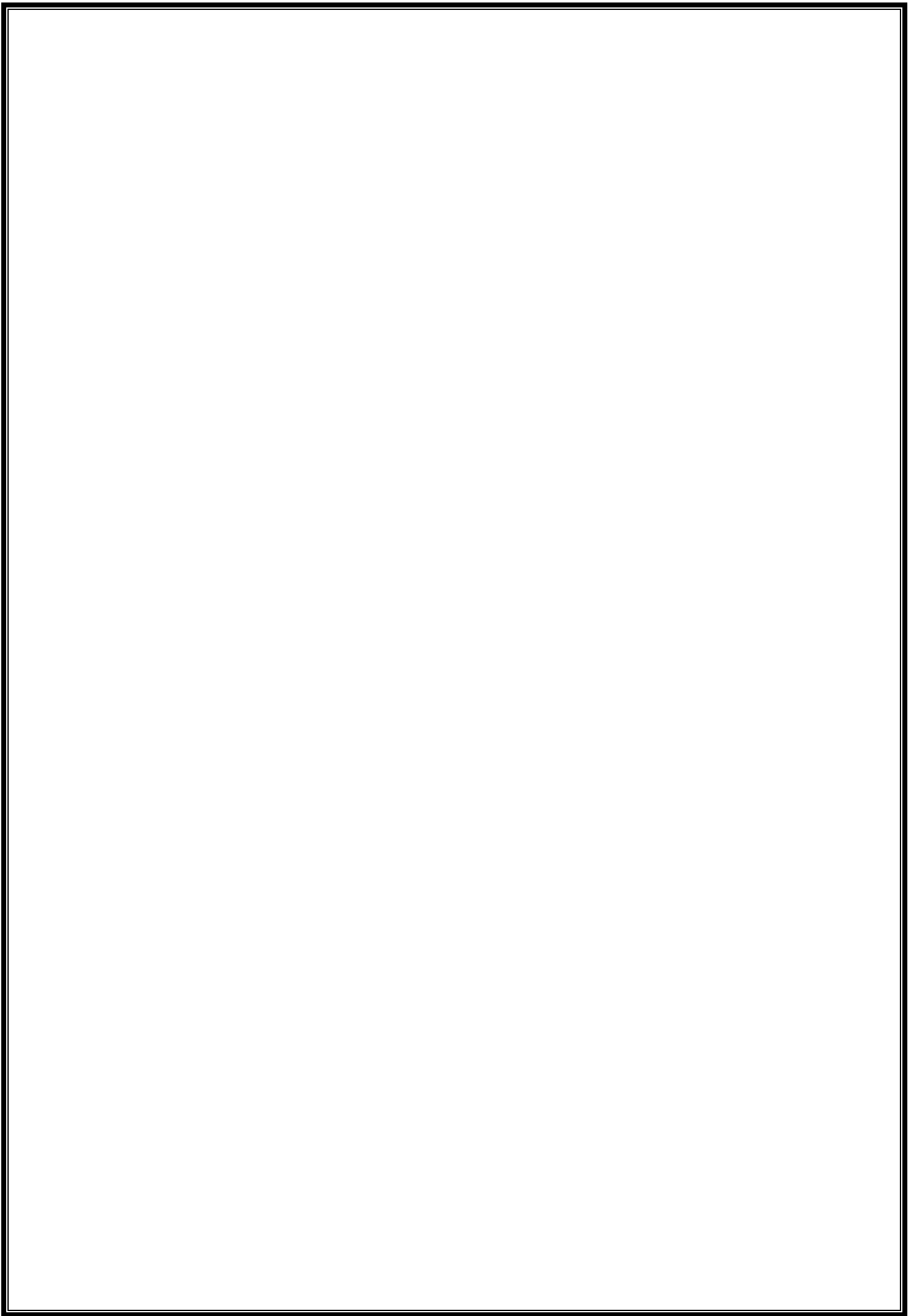
TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO.
1	Introduction and Concepts	1 -7
	1.1 Introduction	
	1.2 Definition	
	1.3 Characteristics of IoT	
	1.4 Functionalities	
	1.5 Applications	
	1.6 Challenges in IoT	
2	Objectives and Function Specification	8-9
	2.1 Objective	
	2.2 Scope	
	2.3 Benefits	
	2.4 Methodology	
	2.5 Functions	
3	Purpose and Requirement Specification	10-14
	3.1 Purpose Specification	
	3.1.1 Measurement of Heart Rate in the Proposed System	
	3.1.2 Measurement of data the proposed system	
	3.1.3 Implementation of IoT in proposed System	
	3.2 Requirement Specification	
	3.2.1 Sensors	
	3.2.2 Components	

4	Design Specification	15-18
	4.1 Process Model Specification	
	4.2 Domain Specification	
	4.3 Service Specification	
	4.4 IoT Level Specification	
5	Methodology	19-23
	5.1 Functional View Specification	
	5.2 Operational View Specification	
	5.3 Gantt Chart	
6	Application Development	24-34
	6.1 Program code	
	6.2 Input/output	
7	Testing	35-37
	7.1 Test Case	
	Limitations	38
	Conclusion	39
	Appendix A- Setting Up Arduino Uno	40-43
	Appendix B – Setting Up ThingSpeak	44-48
	Appendix C – Setting Up Wi-Fi Module	49-69
	Bibliography	70

PREFACE

The Heart Rate Monitoring system is developed using IOT technology with an objective of detecting the heartbeat of the patient in order to monitor the risk of heart attack and also the regular checkup. Body health monitoring is very important to us to make sure our health is in excellent condition. One of the vital parameter for this device under consideration is the heart rate . In this project we describe the design of low cost heart rate monitoring device from fingertips based on the Bluetooth technology. The entire system is comprised of several parts such as Heart Rate module and ThinkSpeak. The Heart Rate module picks up heart rate signal by a non- invasive technique from the subject and sends it wirelessly to computer or android application using Wifi module. This system can be embraced and combined as a part of telemedicine constituent. The data received from heart rate module can be saved and viewed for further medical usage. The result from this device prototype can be utilized for various clinical investigations, indeed these Wifi Signal can be transmitted between 15 to 20 meters radius



CHAPERT- 1

INTRODUCTION and CONCEPTS

1.1 INTRODUCTION

The heart is one of the most important organs in the human body. It acts as a pump for circulating oxygen and blood throughout the body, thus keeping the functionality of the body intact. A heartbeat can be defined as a two-part pumping action of the heart which occurs for almost a second. It is produced due to the contraction of the heart. When blood collects in upper chambers, the SA(Sino Atrial) node sends out an electrical signal which in turn causes the atria to contract. This contraction then pushes the blood through tricuspid and the mitral valves; this phase of the pumping system is called diastole. The next phase begins when the ventricles are completely filled with blood. The electrical signals generating from SA node reach the ventricle and cause them to contract. In today's scenario, health problems related to heart are very common. Heart diseases are one of the most important causes of death among men and women. It claims approximately 1 million deaths every year. Heart rate is a critical parameter in the functioning of the heart. Therefore heart rate monitoring is crucial in the study of heart performance and thereby maintaining heart health.

This paper proposes a heart rate monitoring detection system using IoT. Nowadays treatment of most of the heart-related diseases requires continuous as well as long term monitoring. IoT is very useful in this aspect as it replaces the conventional monitoring systems with a more efficient scheme, by providing critical information regarding the condition of the patient accessible by the doctor.

1.2 DEFINITION

The Internet of Things (IoT) has been defined as:

Definition: A dynamic global network infra-structure with self-configuring capabilities based on the standards and interrogatable communication protocols where physical and virtual “things” have identities, physical attributes and, virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network, often communicate data associated with users .

1.3 CHARACTERISTICS of IoT

- 1.3.1 Dynamic & Self-Adapting
- 1.3.2 Self-Configuring
- 1.3.3 Interoperable Communication Protocol
- 1.3.4 Unique Identity
- 1.3.5 Integrated into Information Network

1.4 FUNCTIONALITIES

The Three Cs of IoT:

- Communication:

IoT communicates information to people and systems, such as state and health of equipment (e.g. It's on or off, charged, full or empty) and data from sensors that can monitor a person's vital signs. In most cases, we didn't have access to this information before or it was collected manually .

- Control and Automation:

In a connected world, a business will have visibility into a device's condition. In many cases, a business or consumer will also be able to remotely control a device. For example, a business can remotely turn on or shut down a specific piece of equipment or adjust the temperature in a climate- controlled environment. Meanwhile, a consumer can use IoT to unlock their car or start the washing machine. Once a performance baseline has been established, a process can send alerts for anomalies and possibly deliver an automated response. For example, if the brake pads on a truck are about to fail, it can prompt the company to take the vehicle out of service and automatically schedule maintenance.

- Cost Savings:

IoT refers to an Internet Of Things (IoT). Connecting any device (including everything from cell phones, vehicles, home appliances and other wearable embedded with sensors and actuators) with Internet so that these objects can exchange data with each other on a network.

- Life is easier with IoT:

A question would arise in your mind that why we are concerned about IoT? Here is the answer that why you should be concerned about IoT. Say for example you are on your way to a meeting; your car could have access to your calendar and already know the best route to take. If the traffic is heavy your car might send a text to the other party notifying them that you will be late. What if your alarm clock wakes up you at 6 a.m. and then notifies your coffee maker to start making coffee for you? Being able to turn the lights on in your house or heating before coming home using your Smartphone? Yes, all these things are possible because of IoT.

- Forecast of Gartner:

All IT spending segments are forecast to decline in 2020 (see Table 1). Enterprise software is expected to have the strongest rebound in 2021 (7.2%) due to the acceleration of digitalization efforts by enterprises supporting a remote workforce, delivering virtual services such as distance learning or telehealth, and leveraging hyperautomation to ensure pandemic-driven demands are met.

Table 1. Worldwide IT Spending Forecast (Millions of U.S. Dollars)

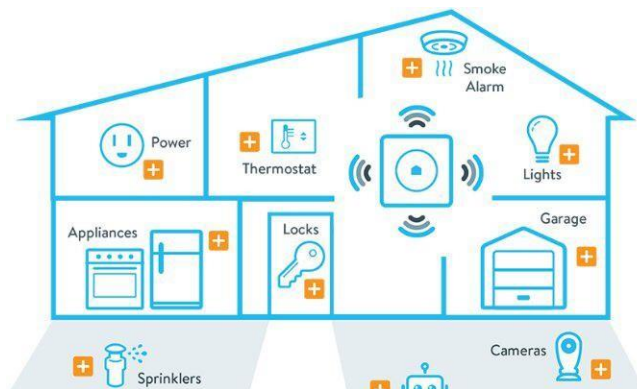
	2019 Spending	2019 Growth (%)	2020 Spending	2020 Growth (%)	2021 Spending	2021 Growth (%)
Data Center Systems	214,911	1.0	208,292	-3.1	219,086	5.2
Enterprise Software	476,686	11.7	459,297	-3.6	492,440	7.2
Devices	711,525	-0.3	616,284	-13.4	640,726	4.0
IT Services	1,040,263	4.8	992,093	-4.6	1,032,912	4.1
Communications Services	1,372,938	-0.6	1,332,795	-2.9	1,369,652	2.8
Overall IT	3,816,322	2.4	3,608,761	-5.4	3,754,816	4.0

Source: Gartner (October 2020)

1.5 APPLICATIONS

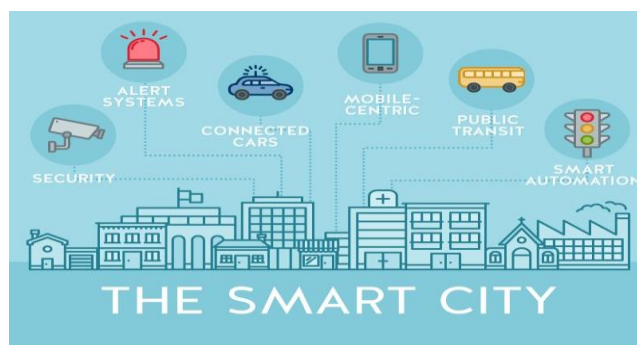
- Smart Home:

The concept of Smart Home is brought up to save time, energy and money. With the introduction of Smart Homes, we would be able to switch on air conditioning before reaching home or switch off lights even after leaving home or unlock the doors to friends for temporary access even when you are not at home.



- Smart cities:

Smart surveillance, automated transportation, smarter energy management systems, water distribution, urban security and environmental monitoring all are examples of internet of things applications for smart cities. IoT will solve major problems faced by the people living in cities like pollution, traffic congestion and shortage of energy supplies etc. By installing sensors and using web applications, citizens can find free available parking slots across the city. Also, the sensors can detect meter tampering issues, general malfunctions and any installation issues in the electricity system.



- Wearables:

Wearable devices are installed with sensors and software's which collect data and information about the users. This data is later pre-processed to extract essential insights about user. These devices broadly cover fitness, health and entertainment requirements. The pre-requisite from internet of things technology for wearable applications is to be highly energy efficient or ultra-low power and small sized.



- Healthcare:

IoT in healthcare is aimed at empowering people to live healthier life and



regular checkup by wearing connected devices. The collected data will help in personalized analysis of an individual's health and provide tailor made strategies to combat illness.

1.6 CHALLENGES in IoT

In the era of IoT, everything is connected, linked up much more than we see in and around us.

IoT is certainly opening door to a lot of opportunities but also to many challenges.

- Security Challenges—IOT can be hacked:

Security is a big issue with IoT devices. With billions of devices being connected together over Internet, how can people be sure that their information is secure? These security issues can be of the following kinds

I. Data Encryption:

IoT applications collect tons of data. Data retrieval and processing is integral part of the whole IoT environment. Most of this data is personal and needs to be protected through encryption.

Encryption is widely used on the internet to protect user information being sent between a browser and a server, including passwords, payment information and other personal information that should be considered private. Organizations and individuals use encryption to protect sensitive data stored on computers, servers and mobile devices like phones or tablets.

II. Data Authentication:

After successful encryption of data chances of device itself being hacked still exist. If there is no way to establish the authenticity of the data being communicated to and from an IoT device, security is compromised.

III. Privacy Challenges- Private information can be stolen:

Then we have the issue of privacy and data sharing. That is because these devices not only collect personal information like users' names and telephone numbers, but can also monitor user activities (e.g., when users are in their houses and what they had for lunch).

- Connectivity Challenges—Billions of devices on a centralized server

One of the biggest challenges for IoT in the future is to connect large number of devices and massive amounts of data that all of these devices are going to produce. There will be need to find out a way to store, track, analyze and make sense of the vast amounts of data that will be generated.

Presently, we rely upon centralized, server/client model to authorize, authenticate, and connect several nodes present on the network. This model is sufficient for the number of IoT devices that are currently a part of the ecosystem. However, in the future, when hundreds of billions of devices will join the network, it will be difficult to manage all the data. Moreover, the capability of current cloud servers is so less that it can breakdown if it has to handle large amounts of information.

- Compatibility and Longevity Challenges-Extra hardware and software

Different technologies like ZigBee, Z-Wave, Wi-Fi, Bluetooth and, Bluetooth Low Energy (BTLE) are all battling to become the dominant transport mechanism between devices and hubs. This becomes a major source of problems when a lot of devices have to be connected, such dense connectivity requires the deployment of extra hardware and software.

CHAPTER – 2

OBJECTIVES & FUNCTION SPECIFICATION

2.1 OBJECTIVE

These days we have an increased number of heart diseases including increased risk of heart attacks. Our proposed system uses sensors that allow to detect heart rate of a person using heartbeat sensing even if the person is at home. The sensor is then interfaced to a microcontroller that allows checking heart rate readings and transmitting them over internet. The user may set the high as well as low levels of heart beat limit.

2.2 SCOPE

The current version of the processing application displays the near-time PPG heart rate but does not record anything. Here is a lot of room for improvements. Logging heart rate measurements and PPG samples along with the time-stamp information available from the PC. Beeping sound alarm for heart rates below or above threshold. Heart rate trend over time, etc.

2.3 BENEFITS

- Cheap and easy to build
- Light weight
- Easily scalable
- Remote monitoring
- Real-time alert

2.5 METHODOLOGY

In this system we use the pulse sensor with Arduino Uno and ESP8266 module, the pulse sensor is placed on the finger and it measures the heart rate and then sends the heart rate to ThinkSpeak.

ESP8266: Early recognition of the disease is very vital in preventing more complications in the future.

There is three cases in which the heart rate is displayed:

HEART RATE MONITORING

- Low Pulse Rate: The low pulse rate is displayed when the heart rate per BPM(Beats per minute) is

>40 and <60. The low pulse rate may lead to medical complications this indicates that the patient needs the doctors help(ex: Low BP)

- Normal Pulse Rate: The normal pulse rate ranges is between >60 and <100 which indicates that the patient has the normal range pulse rate with no complication.

High Pulse Rate: The high pulse rate is between >100 and <150 which indicates the patient has the high pulse range that could result in the heart related diseases(ex: High Blood Pressure)

The above readings are displayed in the mobile based application known as ThinkSpeak via ESp8266.

2.6 FUNCTIONS

- Similar API & UI for all supported hardware & devices.
- Connection to the cloud using WiFi .
- Set of easy-to-use Widgets.
- Easy to integrate and add new functionality using ThinkSpeak
- History data monitoring via Super Chart widget.
- Device-to-Device communication using Bridge Widget, Sending emails, tweets, push notifications, etc.

CHAPTER – 3

PURPOSE & REQUIREMENT SPECIFICATION

3.1 PURPOSE SPECIFICATION

The 'purpose specification principle', that is, the principle that a citizen needs to be informed why the personal data is being collected and the specific purposes for which it will be processed and kept, is a central protection for a citizen in data protection law.

3.1.1 Measurement of heart rate using pulse sensor in proposed System

The Pulse Sensor is a plug-and-play heart-rate sensor for Arduino. The essence is an integrated optical amplifying circuit and noise eliminating circuit sensor. Clip the Pulse Sensor to your earlobe or fingertip and plug it into your Arduino, you can ready to read heart rate.

3.1.2 Measurement of heart rate by ESP8266 in proposed System

The ESP8266 is a very user-friendly and low-cost device to provide internet connectivity to your projects. The module can work both as an Access point and as a station, hence it can easily fetch data and upload it to the internet making the Internet of Things as easy as possible. It can also fetch data from the internet using API's hence your project could access any information that is available on the internet, thus making it smarter. Another exciting feature of this module is that it can be programmed using the Arduino IDE which makes it a lot more user friendly.

3.1.3 Implementation of IoT in proposed System

The proposed system provides improvements to the existing system design. It tries to make the existing system more efficient, convenient and user-friendly. The implementation of the proposed design of the requires following hardware components:

- Pulse Sensors
- ESP8266 Wifi module

3.2 REQRIMENTS SPECIFICATION

Hardware:

Components	Requirements
Arduino Uno	1
Pulse sensor	1
LCD	1
Resistor	10k 2, 20k 2
Bread Board	1
Jumper Wire	-
ESP8266	1

Software:

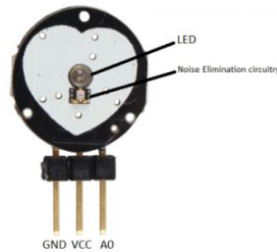
COMPONENTS	REQRIMENT
Arduino IDE	Version 1.8.2
Processor	64-Bit
Operating System	Windows 7 and above
RAM	4GB & above

3.2.2 SENSORS

The devices which converts the electrical signals into digital signals are known as sensors.

Pulse Sensor:

Clip the Pulse Sensor to your earlobe or fingertip and plug it into your Arduino, you can ready to read heart rate. The pulse sensor has three pins: VCC, GND & Analog Pin. There is also a LED in the center of this sensor module which helps in detecting the heartbeat. Below the LED, there is a noise elimination circuitry that is supposed to keep away the noise from affecting the readings.



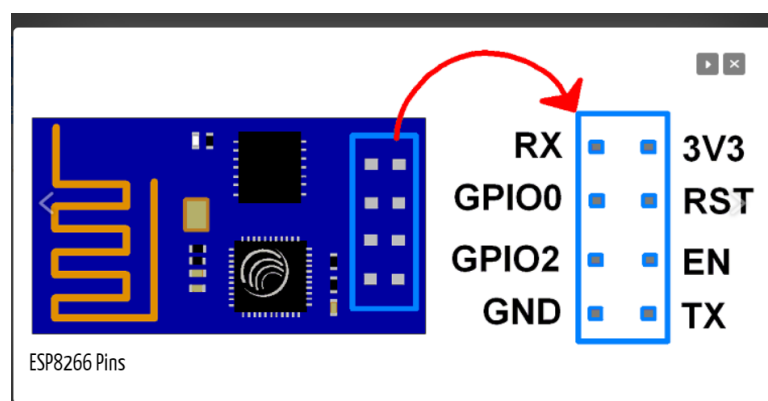
3.2.3 Components

ESP8266:

The ESP8266 is a very user-friendly and low-cost device to provide internet connectivity to your projects. The module can work both as an Access point and as a station, hence it can easily fetch data and upload it to the internet making the Internet of Things as easy as possible

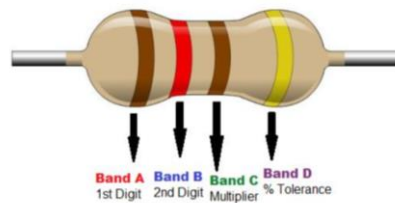
ESP8266 here is its pins description.

- Pin1:Ground
- Pin2:Tx/GPIO1
- Pin3:GPIO-2
- Pin4:CH_EN
- Pin5:Flash/GPIO-0
- Pin6:Reset
- Pin7:RX/GPIO-3
- Pin 8 : Vcc



Resistor:

A resistor is a passive two-terminal electrical component that implements electrical resistance as a circuit element. In electronic circuits, resistors are used to reduce current flow, adjust signal levels, to divide voltages, bias active elements, and terminate transmission lines, among other uses.



Bread Board:

A breadboard is a construction base for prototyping of electronics. Originally it was literally a bread board, a polished piece of wood used for slicing bread.



Arduino Uno Board:

Arduino is a single-board microcontroller meant to make the application more accessible which are interactive objects and its surroundings. The hardware features with an open-source hardware board designed around an 8-bit Atmel AVR microcontroller or a 32-bit Atmel ARM. Current models consists a USB interface, 6 analog input pins and 14 digital I/O pins that allows the user to attach various extension boards.



LCD:

An LCD screen is an electronic display module and has a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. A 16x2 LCD means it can display 16 characters per line and there are 2 such lines.



Jumper Wire:

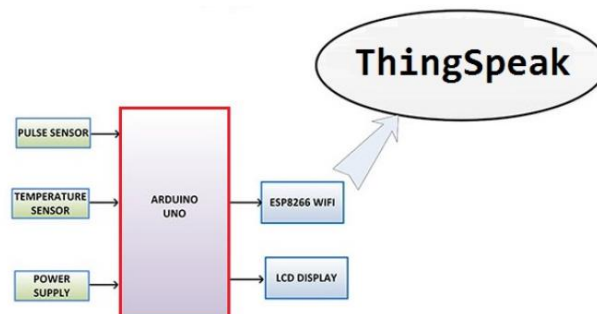
A jump wire is an electrical wire, or group of them in a cable, with a connector or pin at each end (or sometimes without them – simply "tinned"), which is normally used to interconnect the components of a breadboard or other prototype or test circuit, internally or with other equipment or components, without soldering.

CHAPTER – 4

DESIGN SPECIFICATION

4.1 PROCES MODEL SPECIFICATION

Process models are processes of the same nature that are classified together into a model. Thus, a process model is a description of a process at the type level. Since the process model is at the type level, a process is an instantiation of it. The same process model is used repeatedly for the development of many applications and thus, has many instantiations. One possible use of a process model is to prescribe how things must/should/could be done in contrast to the process itself which is really what happens. A process model is roughly an anticipation of what the process will look like. What the process shall be will be determined during actual system development. Process specification is a generic term for the specification of a process. Its context is not unique to "business activity" but can be applied to any organizational activity.



4.2 DOMAIN SPECIFICATION

Physical Entities:

Physical Entities is a discrete and identifiable entity in the physical environment. The IoT system provides information about the physical entity or performs actuation upon the physical entity. The physical Entity of Heart Rate Monitoring based Visually impaired people.

Virtual Entities:

Virtual Entity is a representation of the physical entity in the digital world for each physical entity, there is a virtual entity in the domain model.

Devices:

Device provides a medium between physical entities and virtual entities. Devices are either attached to physical entities or places near physical entities. Devices are used to gather information about the physical entities perform actuation upon physical entities or used to identity physical entities.

Resources:

Resources are the software components which can be either “on-device” or “network- resources”. On device resources are hosted on the device and include software components that either provides information on or enable actuation upon the physical entity to which the device is attached. Network resources include the software components that are available in the network.

Resources of Smart Cane for the Visually impaired :

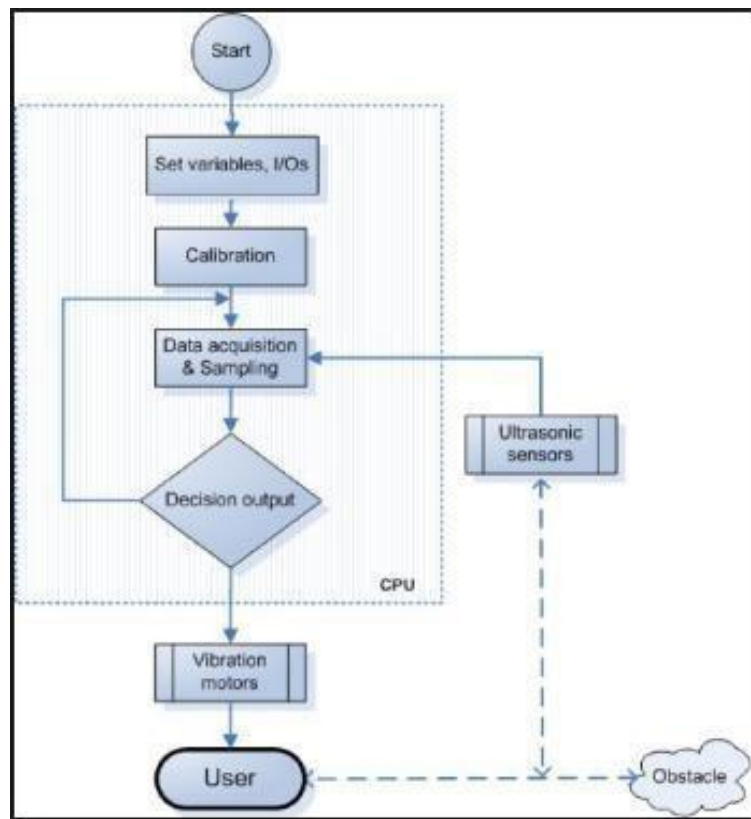
- I. Arduino IDE
- II. ThinkSpeak

Services:

Service provide an interface for interacting with the physical entity. Services access the resources hosted on the device or network resources to obtain information about the physical entity or perform actuation about the physical entity.

4.3 SERVICE SPECIFICATIONS

Service specifications define the service in the IOT system, services types, services inputs/outputs, services end points, Service schedules, service preconditions and service effects.

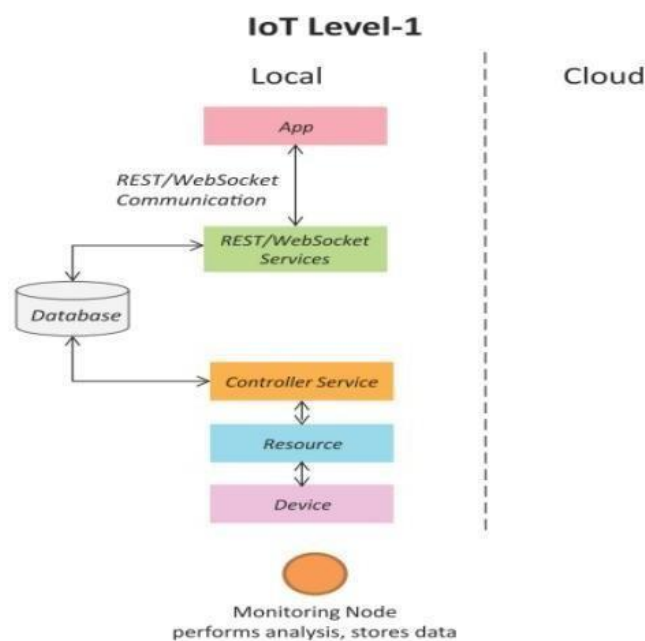


4.4 IoT LEVEL SPECIFICATION

IoT level-1 IoT system has a single node/device that perform sensing and/or actuation stores data, performs analysis and hosts the application. level-1 IoT systems are suitable for modeling low cost and low-complexity solutions where the data involved is not big and the analysis are not computationally intensive.

5

A level - 1 IoT system has a single node/device that performs sensing and/or actuation, stores data, performs analysis and hosts the application.



Level - 1 IoT systems are suitable for modelling low - cost and low - complexity solutions where the data involved is not big and the analysis requirements are not computationally intensive.

CHAPTER – 5 METHODOLOGY

5.1 FUNCTIONAL VIEW SPECIFICATION

Defines the functions of the IOT systems grouped into various functional groups. Each functional group either provides functionalities for interacting with the instances of concept defined in the domain model

Communication: It handles the communication for IOT systems. The communication includes the communication protocols that form the backbone of the IOT systems and enable network connectivity

Services: It includes various services involved in the IOT system such as services for device monitoring, device control services, data publishing services and data analysis services.

Management: Includes all the functionality that are needed to configure and manage the IOT system. It is well maintained and automated, by reducing the manual work for gardening. The water motor is automatically turned OFF and ON depending on the data collected and analyzed. As the readings are met the threshold levels the water motor is turned OFF and the speakers are turned.

The system employs CC1010 RF-transceivers [CC1010, 2006] for wireless data transmission. Non- directionality and a medium range of RF transmissions (40-50m) make it suitable for our application. While operating the device, the user does not have to worry, whether the bus is standing towards his left, right, back or in front. Both the user and the bus modules are fitted with an antenna for wireless communication, which can be folded when the device is not in use.

5.2 OPERATIONAL VIEW SPECIFICATION

In this step, various options pertaining to the IOT system deployment and operation are defined. Operational view specification for water level monitoring system are as follows:

- Devices: Computing device (Arduino Uno board), pulse sensor, Wi-Fi Module.
- Communication Protocols: link layer-802.11, Network Layer-IPv4/IPv6, Transport- TCP, Application – HTTP.
- Services:
 1. Controller Service – Hosted on device, implemented in C programming environment, runs as a native service.
 2. Mode service
 3. Application
 4. Management

Device management – Arduino UNO management.

Communication Options:

The communication functional group handles the communication for the IOT system. It includes the communication protocols that forms the backbone of IOT systems and enable network connectivity. The System is connected to the web-page through the Wi-Fi module using the TCP protocols.

TCP (Transmission Control Protocol) is a standard that defines how to establish and maintain a network conversation via which application programs can exchange data. TCP works with the Internet Protocol (IP), which defines how computers send packets of data to each other.

It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and—because it is meant to provide error-free data transmission—handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive. In the Open Systems

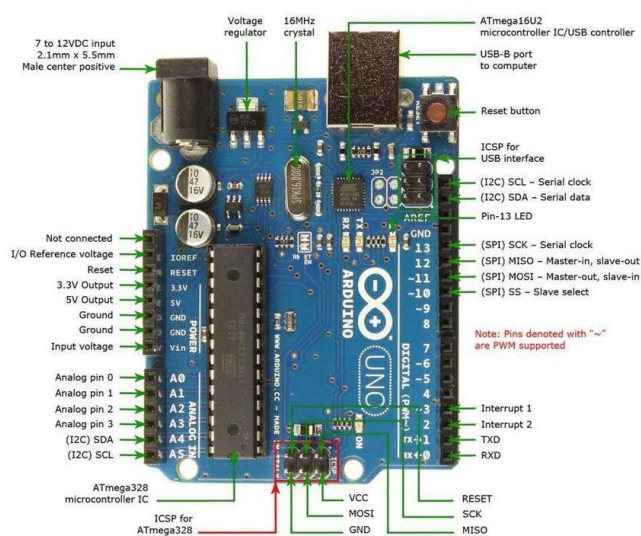
Interconnection (OSI) communication model, TCP covers parts of Layer 4, the Transport Layer, and parts of Layer 5, the Session Layer.

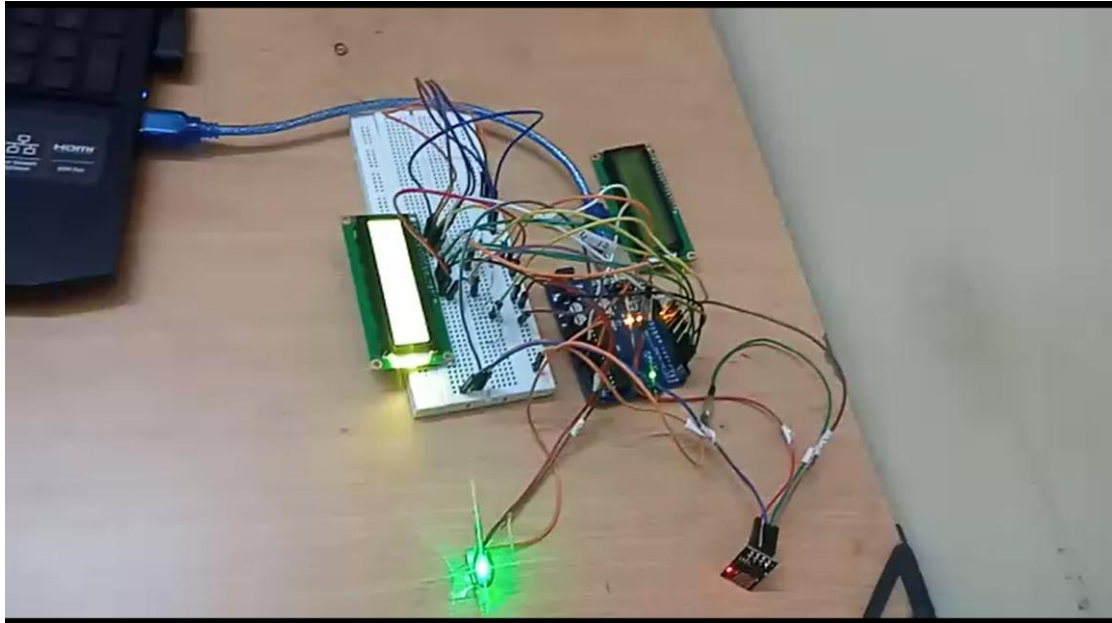
For example, when a Web server sends an HTML file to a client, it uses the HTTP protocol to do so. The HTTP program layer asks the TCP layer to set up the connection and send the file. The TCP stack divides the file into packets, numbers them and then forwards them individually to the IP layer for delivery. Although each packet in the transmission will have the same source and destination IP addresses, packets may be sent along multiple routes. The TCP program layer in the client computer waits until all of the packets have arrived, then acknowledges those it receives and asks for the retransmission on any it does not (based on missing packet numbers), then assembles them into a file and delivers the file to the receiving application.

Device and Component Integration:

The devices and components used in this system are Arduino Uno Board, Pulse Sensor, LCD and Wi-Fi Module.

The Arduino is then externally connected to the Web-Based Cloud, the ThingSpeak.com website i.e. the MATLAB server through the Wi-Fi module ESP8266. The collected data from the Pulse sensor and the Humidity & LCD is then sent to the MATLAB server. These data is updated at every 1000 seconds.





Circuit Diagram:

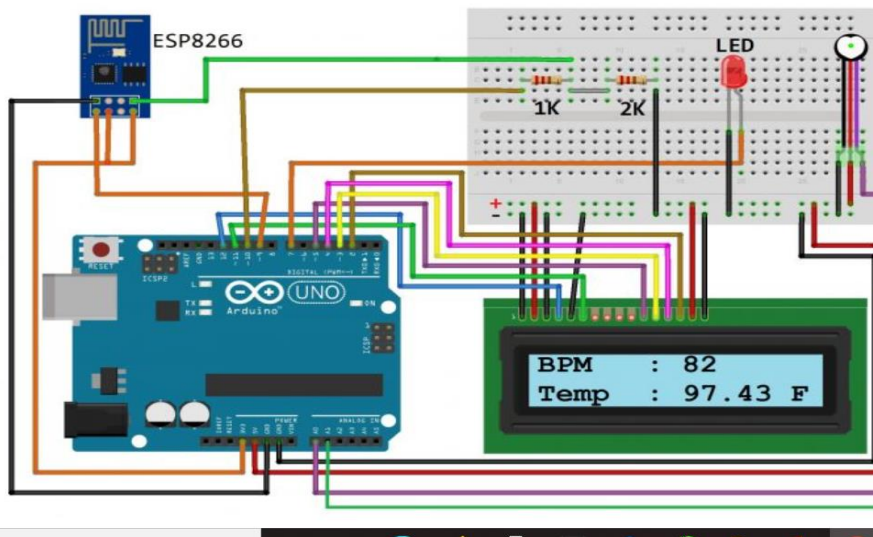
A circuit diagram is a graphical representation of an electrical circuit. A pictorial circuit diagram uses simple images of components, while a schematic diagram shows the components and interconnections of the circuit using standardized symbolic representations.

This Circuit diagram represents the electrical circuit of the “Heart Rate Monitoring” System.

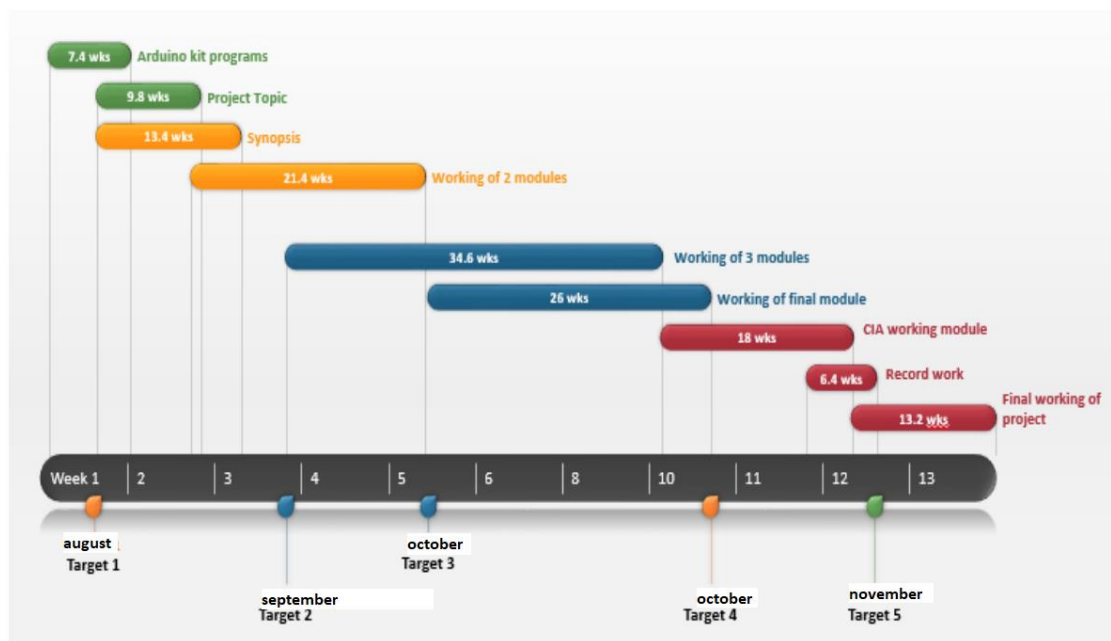
HEART RATE MONITORING

Circuit Diagram & Connections:

For designing IoT Based Patient Health Monitoring System using ESP8266 & Arduino, assemble the circuit in the figure below.



5.3 GANTT CHART



CHAPTER – 6**APPLICATION DEVELOPMENT****6.1 PROGRAM CODE**

```
#include<SoftwareSerial.h>

#define DEBUG true

SoftwareSerial esp8266(9,10);

#include <LiquidCrystal.h>

#include<stdlib.h>

LiquidCrystal lcd(12,11,5,4,3,2);

#define SSID "Airtel-MyWiFi-AMF-311WW-95B4"

#define PASS "110391b4"

#define IP "184.106.153.149"// thingspeak.com ip

String msg = "GET /update?key=V7X84JDKYS3Z95Z0"; //change it with your api
key

//Variables

float temp;

int hum;

String tempC;

int error;

int pulsePin = 0; // Pulse Sensor connected to analog pin

int blinkPin = 13; // pin to blink led at each beat

int fadePin = 5;
```

```
int fadeRate = 0;

// Volatile Variables, used in the interrupt service routine!

volatile int BPM; // int that holds raw Analog in 0. updated every 2mS

volatile int Signal; // holds the incoming raw data

volatile int IBI = 600; // int that holds the time interval between beats! Must be
seeded!

volatile boolean Pulse = false; // "True" when heartbeat is detected. "False" when not
a "live beat".

volatile boolean QS = false; // becomes true when Arduino finds a beat.

// Regards Serial OutPut -- Set This Up to your needs

static boolean serialVisual = true; // Set to 'false' by Default.

volatile int rate[10]; // array to hold last ten IBI values

volatile unsigned long sampleCounter = 0; // used to determine pulse timing

volatile unsigned long lastBeatTime = 0; // used to find IBI

volatile int P = 512; // used to find peak in pulse wave

volatile int T = 512; // used to find trough in pulse wave

volatile int thresh = 525; // used to find instant moment of heart beat

volatile int amp = 100; // used to hold amplitude of pulse waveform

volatile boolean firstBeat = true; // used to seed rate array

volatile boolean secondBeat = false; // used to seed rate array
```



```
void setup()

{

    lcd.begin(16, 2);

    lcd.print("Connecting...");

    Serial.begin(9600);

    Esp8266.begin(9600);

    Serial.println("AT");

    esp8266.println("AT");

    delay(5000);

    if(esp8266.find("OK")){

        connectWiFi();

    }

    interruptSetup();

}

void loop()

{

    lcd.clear();

    start:

    error=0;

    lcd.setCursor(0, 0);
```

```
    lcd.print("BPM = ");

    lcd.print(BPM);

    delay (100);

    lcd.setCursor(0, 1); // set the cursor to column 0, line 2

    delay(1000);

    updatebeat();

    if (error==1){

        goto start; //go to label "start"

    }

    delay(1000);

}

void updatebeat()

{

    String cmd = "AT+CIPSTART=\"TCP\", \"";

    cmd += IP;

    cmd += "\",80";

    Serial.println(cmd);

    esp8266.println(cmd);

    delay(2000);

    if(esp8266.find("Error")){

        return;

    }

}
```

```
}  
  
cmd = msg;  
  
cmd += "&field1=";  
  
cmd += "\r\n";  
  
Serial.print("AT+CIPSEND=");  
  
esp8266.print("AT+CIPSEND=");  
  
Serial.println(cmd.length());  
  
esp8266.println(cmd.length());  
  
if(esp8266.find(">"))  
{  
  
    Serial.print(cmd);  
  
    esp8266.print(cmd);  
  
}  
  
else  
  
{  
  
    Serial.println("AT+CIPCLOSE");  
  
    esp8266.println("AT+CIPCLOSE");  
  
    error=1;  
  
}  
  
}
```

```
boolean connectWiFi(){  
  
    Serial.println("AT+CWMODE=1");  
  
    esp8266.println("AT+CWMODE=1");  
  
    delay(2000);  
  
    String cmd="AT+CWJAP=\"";  
  
    cmd+=SSID;  
  
    cmd+="\", \"\"";  
  
    cmd+=PASS;  
  
    cmd+="\"\"";  
  
    Serial.println(cmd);  
  
    esp8266.println(cmd);  
  
    delay(5000);  
  
    if(esp8266.find("OK")){  
  
        Serial.println("OK");  
  
        return true;  
  
    }else{  
  
        return false;  
  
    }  
  
}
```

```
void interruptSetup()

{

    TCCR2A = 0x02; // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO
    INTO CTC MODE

    TCCR2B = 0x06; // DON'T FORCE COMPARE, 256 PRESCALER

    OCR2A = 0x7C; // SET THE TOP OF THE COUNT TO 124 FOR 500Hz
    SAMPLE RATE

    TIMSK2 = 0x02; // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2
    AND OCR2A

    sei(); // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED

}

ISR(TIMER2_COMPA_vect)

{ // triggered when Timer2 counts to 124

    cli();          // disable interrupts while we do this

    Signal = analogRead(pulsePin); // read the Pulse Sensor

    sampleCounter += 2; // keep track of the time in mS

    int N = sampleCounter - lastBeatTime;

    // find the peak and trough of the pulse wave

    if(Signal < thresh && N > (IBI/5)*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 && Signal > P)

{
    // thresh condition helps avoid noise

    P = Signal; // P is the peak

}

if (N > 250)

{
    // avoid high frequency noise

    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) )

    {

        Pulse = true;           // set the Pulse flag when there is a pulse

        digitalWrite(blinkPin,HIGH);           // turn on pin 13 LED

        IBI = sampleCounter - lastBeatTime;           // time between beats in mS

        lastBeatTime = sampleCounter;           // keep track of time for next pulse


        if(secondBeat){           // if this is the second beat

            secondBeat = false;           // clear secondBeat flag

            for(int i=0; i<=9; i++){ // seed the running total to get a realistic BPM at
startup
                rate[i] = IBI;

            }

        }

    }

}
```

```
}

    if(firstBeat){                // if it's the first time beat is found

        firstBeat = false;        // clear firstBeat flag

        secondBeat = true;        // set the second beat flag

        sei();                    // enable interrupts again

        return;                  // IBI value is unreliable so discard it

    }

    word runningTotal = 0; // clear the runningTotal variable

    for(int i=0; i<=8; i++){      // shift data in the rate array

        rate[i] = rate[i+1];      // and drop the oldest IBI value

        runningTotal += rate[i];  // add up the 9 oldest IBI values

    }

    rate[9] = IBI; // add the latest IBI to the rate array

    runningTotal += rate[9]; // add the latest IBI to runningTotal

    runningTotal /= 10; // average the last 10 IBI values

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

    QS = true; // set Quantified Self flag

}

}

}
```

6.2 INPUT/OUTPUT

The ESP8266 module works with 3.3V only, anything more than 3.7V would kill the module hence be cautious with your circuits. Here is its pins description.

Pin 1: Ground: Connected to the ground of the circuit

Pin 2: Tx/GPIO – 1: Connected to Rx pin of programmer/uC to upload program

Pin 3: GPIO – 2: General purpose Input/output pin

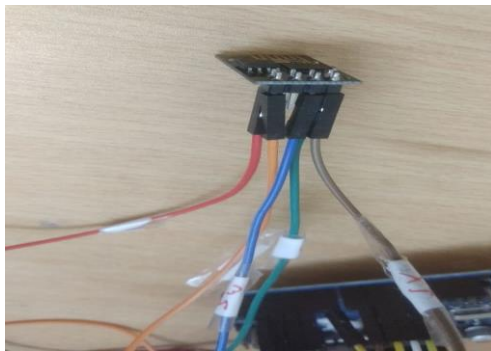
Pin 4 : CH_EN: Chip Enable/Active high

Pin 5: Flash/GPIO – 0: General purpose Input/output pin

Pin 6 : Reset: Resets the module

Pin 7: RX/GPIO – 3: General purpose Input/output pin

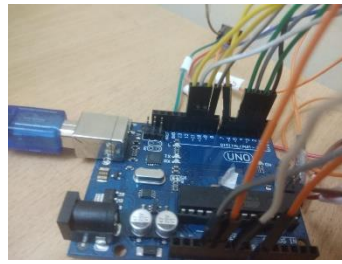
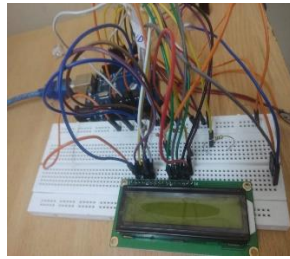
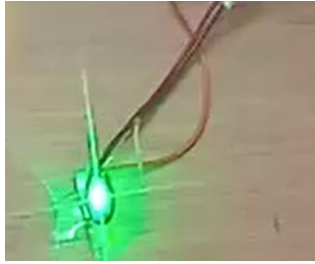
Pin 8: Vcc: Connect to +3.3V only



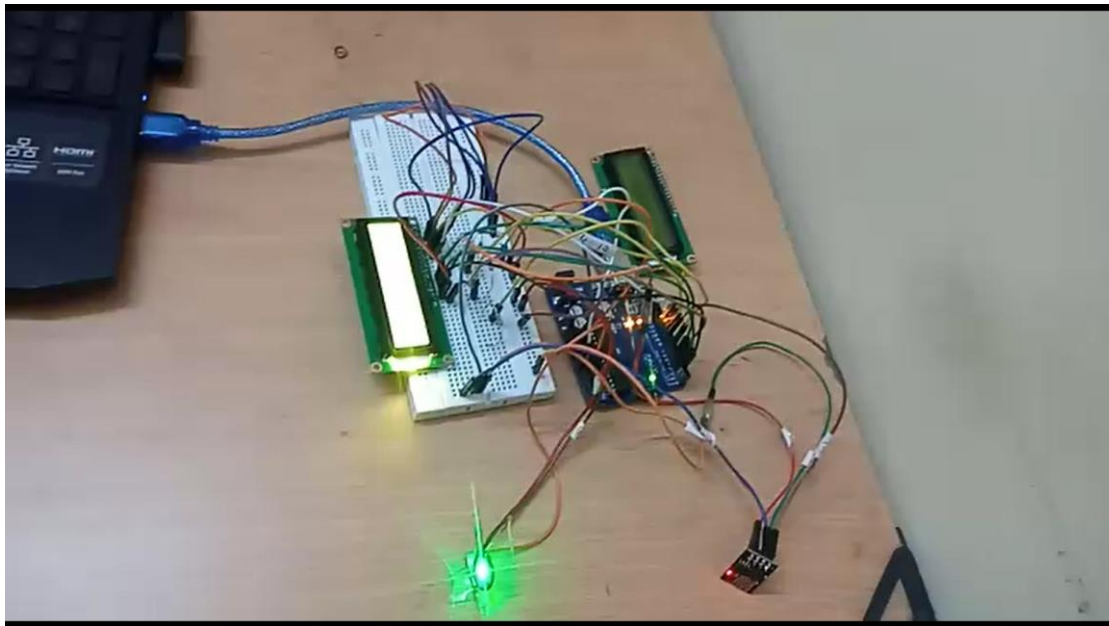
Pulse Sensor and LCD

1. Connect Pulse Sensor output pin to A0 of Arduino and other two pins to VCC & GND.
2. Connect LM35 Temperature Sensor output pin to A1 of Arduino and other two pins to VCC & GND.
3. Connect the LED to Digital Pin 7 of Arduino via a 220-ohm resistor.
4. Connect Pin 1,3,5,16 of LCD to GND.
5. Connect Pin 2,15 of LCD to VCC.
6. Connect Pin 4,6,11,12,13,14 of LCD to Digital Pin 12,11,5,4,3,2 of Arduino.
7. The RX pin of ESP8266 works on 3.3V and it will not communicate with the Arduino when we will connect it directly to the Arduino. So, we will have to make a voltage divider for it which will convert the 5V into 3.3V. This can be done by connecting the 2.2K & 1K resistor. Thus the RX pin of the ESP8266 is connected to pin 10 of Arduino through the resistors.

8. Connect the TX pin of the ESP8266 to pin 9 of the Arduino.



Screenshots



CHAPTER – 7

TESTING

Testing defines the status of the working functionalities of any particular system. Through testing particular software, one can't identify the defects in it but can analyse the performance of software and its working behavior. By testing the software, we can find the limitations that become the conditions on which the performance is measured on that particular level. In order to start the testing, process the primary thing is requirements of software development cycle. Using this phase, the testing phase will be easier for testers.

The capacity of the software can be calculated by executing the code and inspecting the code in different conditions such as testing the software by subjecting it to different sources as input and examining the results with respect to the inputs.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product.

The different types of testing performed on this project are:

Black Box Testing

Internal system design is not considered in this type of testing. Tests are based on the requirements and functionality.

End-to-End Testing

Similar to system testing, End-to-end testing involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

Integration Testing

Testing of all integrated modules to verify the combined functionality after integration is termed as Integration Testing. Modules are typically code modules, individual applications, client and server applications on a network,

etc. This type of testing is especially relevant to client/server and distributed systems.

Unit Testing

Testing of an individual software component or module is termed as Unit Testing. It is typically done by the programmer and not by testers, as it requires a detailed knowledge of the internal program design and code. It may also require developing

White Box Testing

White Box testing is based on the knowledge about the internal logic of an application's code. It is also known as Glass box Testing. Internal software and code working should be known for performing this type of testing. Under this test are based on the coverage of code statements, branches, paths, conditions etc.

7.1 Test Case

Test Case No.: 01

Purpose	Sensor Readings
Input	Collect and update the reading
Expected Output	Data to be updated to the cloud through Arduino Uno board.
Pass/Fail	PASS

Test Case No.: 02

Purpose	Pulse Sensor
Input	Depending on the readings the heart rate turns ON/OFF
Expected Output	The led turns ON/OFF automatically.
Pass/Fail	PASS

Test Case No.: 03

Purpose	LCD
Input	The read the heart rate value
Expected Output	Between 60 to 100 normal

HEART RATE MONITORING

Pass/Fail	PASS
-----------	------

Test Case No.: 04

Purpose	ThinkSpeak
Input	Depending on the readings the heart rate value
Expected Output	The display the heart rate vaule in graph format
Pass/Fail	PASS

LIMITATIONS

1. **Data:** The more the information, the easier it is to make the right decision.
Knowing what to get from the grocery while you are out, without having to check on your own, not only saves time but is convenient as well.
2. **Tracking:** The computers keep a track both on the quality and the viability of things at home. Knowing the expiration date of products before one consumes them improves safety and quality of life. Also, you will never run out of anything when you need it at the last moment.
3. **Time:** The amount of time saved in monitoring and the number of trips done otherwise would be tremendous.
4. **Money:** The financial aspect is the best advantage. This technology could replace humans who are in charge of monitoring and maintaining supplies.

Conclusion - Future Application of the Project

An IoT-based human heartbeat rate monitoring and control system is developed. This system uses the capability of a heart pulse sensor for data acquisition. A human's heartbeat is captured as data signals and processed by the microcontroller. The processed data are transmitted to the IoT platform for further analytics and visualization. Experimental results obtained were found to be accurate as the system was able to sense and read the heartbeat rate of its user and transmits the sensed data via Wifi to the ThinkSpeak. From the results obtained, it was found that the heartbeat rate of low if >40 and <60 , medium if >60 and <100 , high if >100 and <150 . Furthermore, this research paper presents an approach that is flexible, reliable, and confidential for a heartbeat rate monitoring and control system using sensor network.

IoT technology. The implemented device can be deployed to the medical field to assist the medical practitioners to efficiently and reliably do their work without difficulties.

The current version of the processing application displays the near-time PPG heart rate but does not record anything. Here is a lot of room for improvements. Logging heart rate measurements and PPG samples along with the time-stamp information available from the PC. Beeping sound alarm for heart rates below or above threshold. Heart rate trend over time, etc.

APPENDIX – A**SETTING UP ARDUINO UNO**

If you want to program your Arduino/Genuino Uno while offline you need to install the Arduino Desktop IDE The Uno is programmed using the Arduino Software (IDE), our Integrated Development Environment common to all our boards. Before you can move on, you must have installed the Arduino Software (IDE) on your PC, as explained in the home page of our Getting Started.

Connect your Uno board with an A B USB cable; sometimes this cable is called a USB printer cable.



The USB connection with the PC is necessary to program the board and not just to power it up. The Uno automatically draw power from either the USB or an external power supply. Connect the board to your computer using the USB

cable. The green power LED (labelled PWR) should go on.

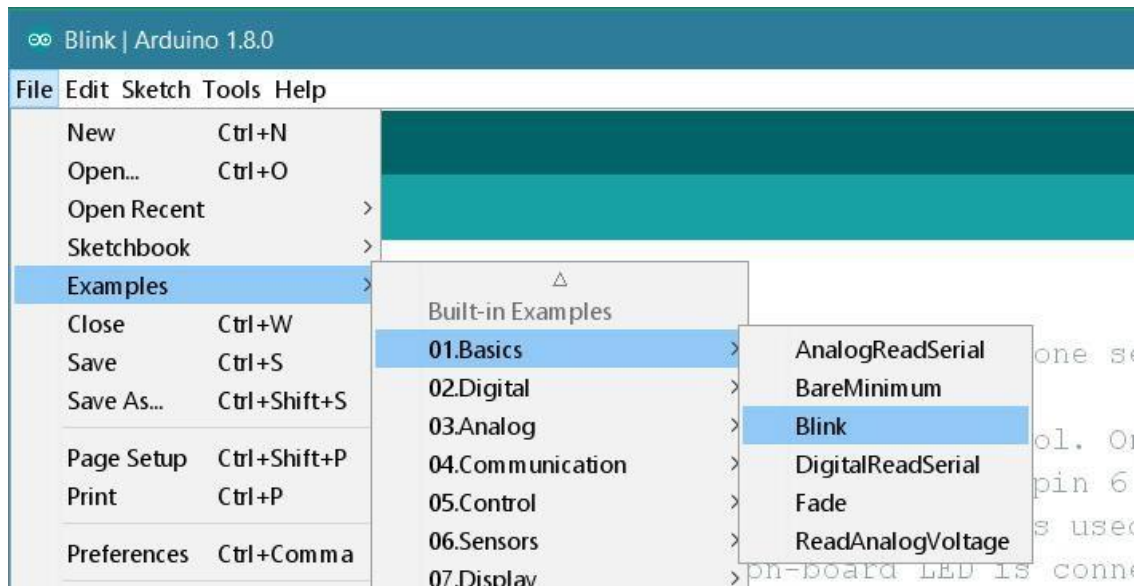
Install the board drivers

- If you used the Installer, Windows - from XP up to 10 - will install drivers automatically as soon as you connect your board.
- If you downloaded and expanded the Zip package or, for some reason, the board wasn't properly recognized, please follow the procedure below.
- Click on the Start Menu, and open up the Control Panel.
- While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
- Look under Ports (COM & LPT). You should see an open port named "Arduino UNO (COMxx)". If there is no COM & LPT section, look under "Other Devices" for "Unknown Device".
- Right click on the "Arduino UNO (COMxx)" port and choose the "Update Driver Software" option.
- Next, choose the "Browse my computer for Driver software" option.
- Finally, navigate to and select the driver file named "arduino.inf", located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub- directory). If you are using an old version of the IDE (1.0.3 or older), choose the Uno driver file named "Arduino UNO.inf"

Windows will finish up the driver installation from there.

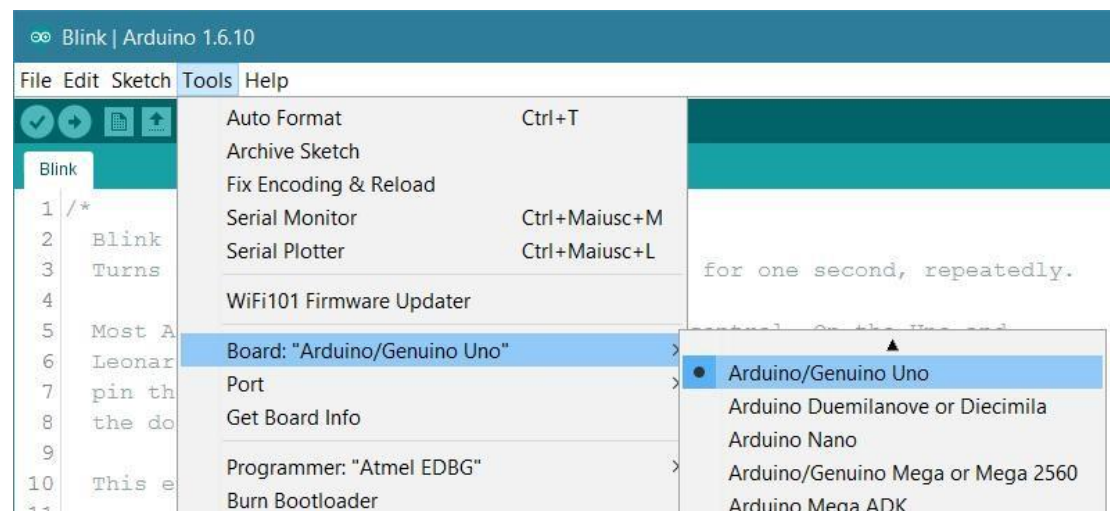
Open your first sketch

Open the LED blink example sketch: File > Examples > 01.Basics > Blink.

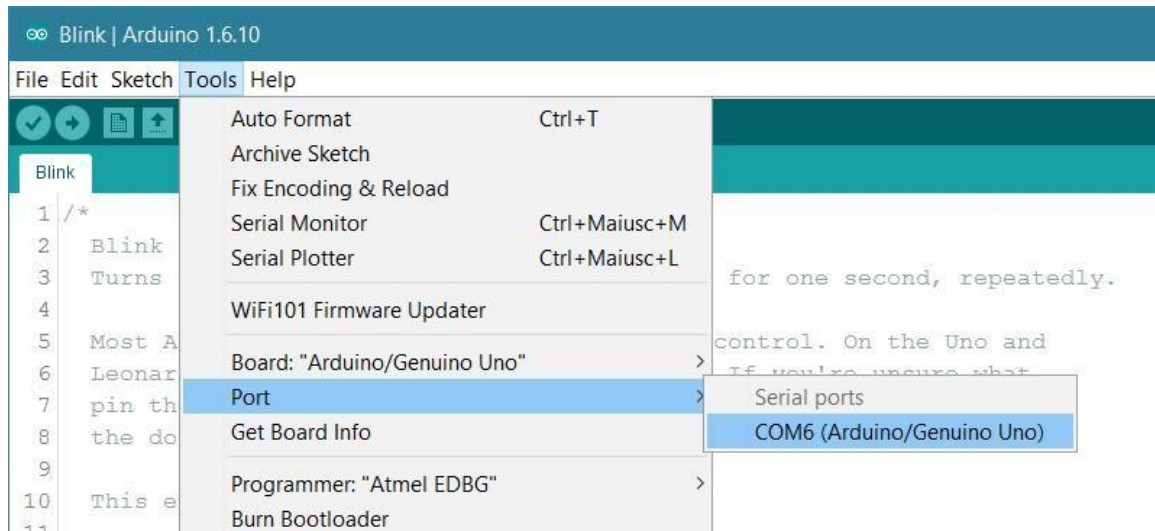


Select your board type and port

You'll need to select the entry in the Tools > Board menu that corresponds to your Arduino or Genuino board.

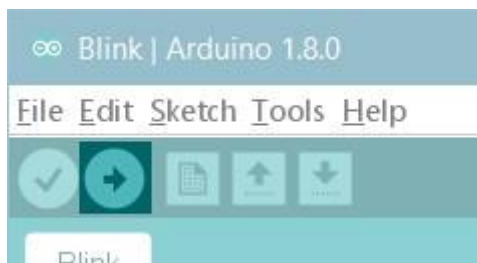


Select the serial device of the board from the Tools | Serial Port menu. This is likely to be COM3 or higher (COM1 and COM2 are usually reserved for hardware serial ports). To find out, you can disconnect your board and re-open the menu; the entry that disappears should be the Arduino or Genuino board. Reconnect the board and select that serial port.



Upload the program

Now, simply click the "Upload" button in the environment. Wait a few seconds - you should see the RX and TX leds on the board flashing. If the upload is successful, the message "Done uploading." will appear in the status bar.



A few seconds after the upload finishes, you should see the pin 13 (L) LED on the board start to blink (in orange). If it does, congratulations! You've gotten Arduino or Genuino up-and-running. If you have problems, please see the troubleshooting suggestions

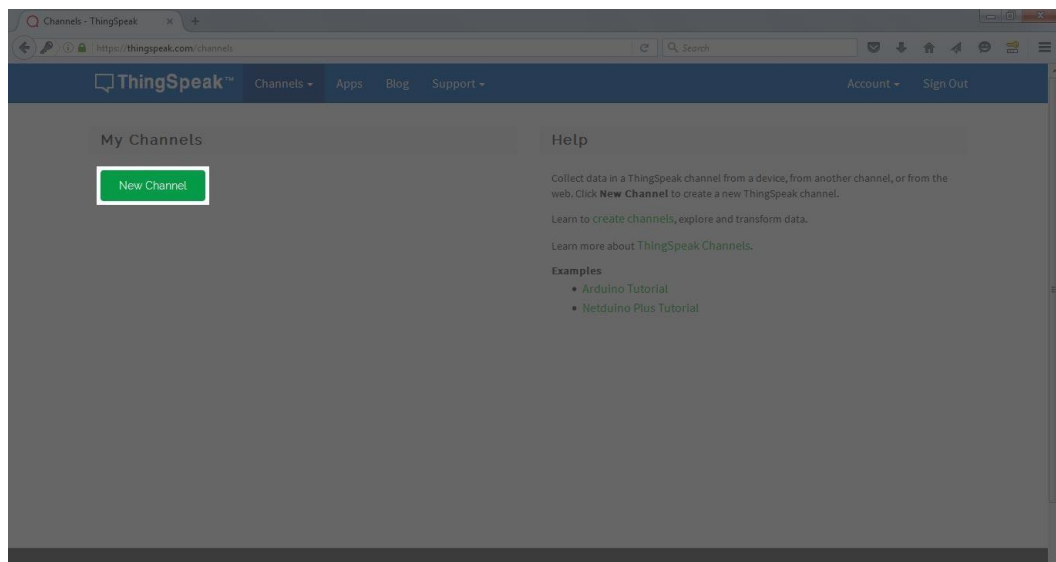
APPENDIX – B

SETTING UP THINGSPEAK

1. Introduction:

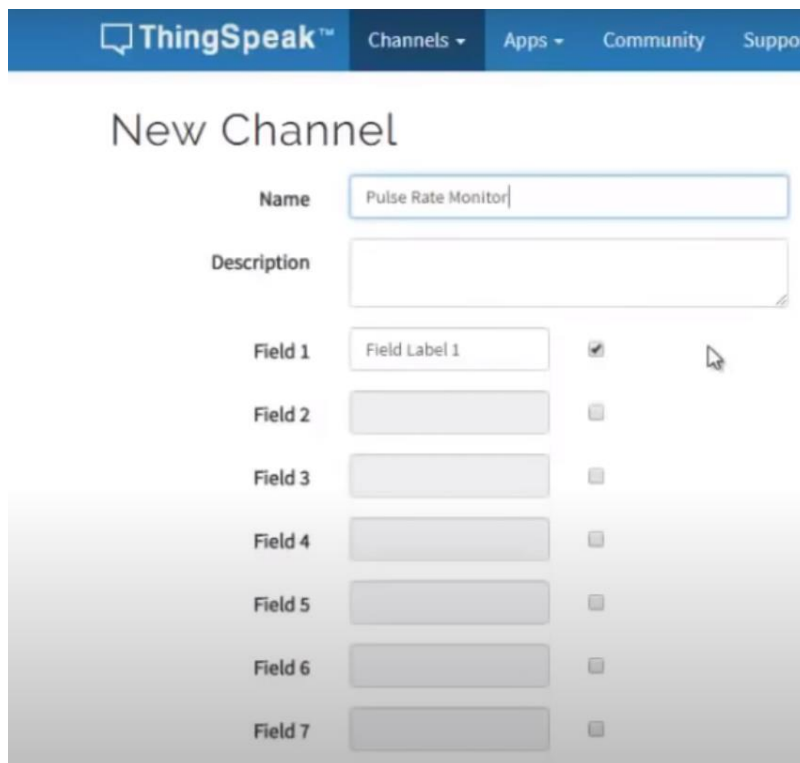
This tutorial explains how to setup account with ThingSpeak and basics. ThingSpeak is an open cloud data platform where you can store and retrieve data.

1. ThingSpeak If you do not have a ThingSpeak account create one. Once you have a ThingSpeak account login to your account. Create a new channel by clicking on the button as shown in below image - A channel is the source for your data. Where you can store and retrieve data. A channel can have maximum 8 fields. It means you can store 8 different data to a channel.



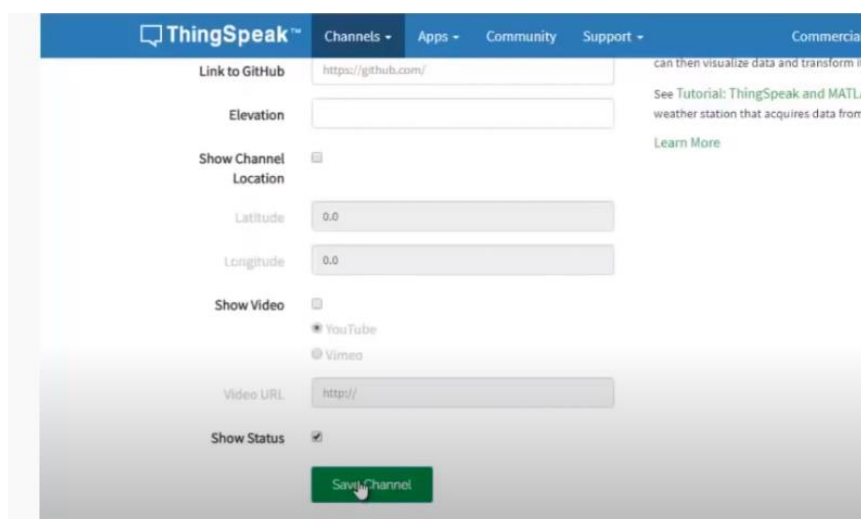
Enter basic details of the channel

Here we are creating channel as Pulse rate monitor and save it.



The screenshot shows the 'New Channel' form on the ThingSpeak website. The form includes a 'Name' field with the text 'Pulse Rate Monitor', a 'Description' field, and seven 'Field' entries. The first field is labeled 'Field 1' and has a value of 'Field Label 1'. The other fields are labeled 'Field 2' through 'Field 7' and are currently empty. Each field has a checkbox to its right, which is checked for Field 1 and unchecked for the others.

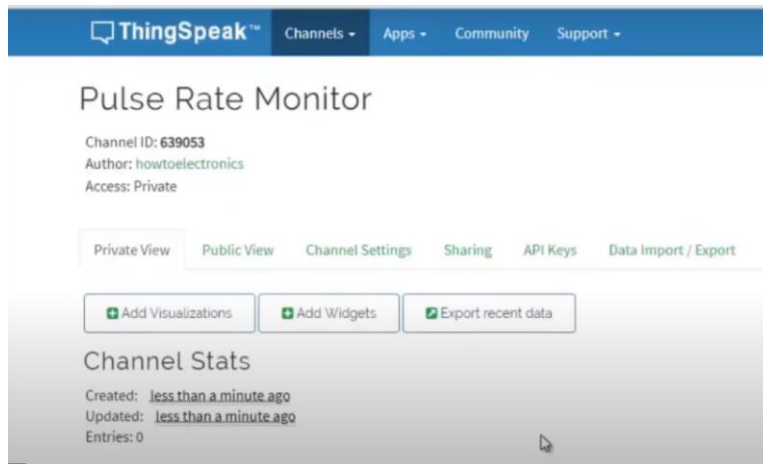
Scroll down and save the channel.



The screenshot shows the bottom portion of the 'New Channel' form. It includes a 'Link to GitHub' field with the URL 'https://github.com/', an 'Elevation' field, a 'Show Channel Location' checkbox, 'Latitude' and 'Longitude' fields, a 'Show Video' checkbox, a 'Video URL' field, and a 'Show Status' checkbox. A green 'Save Channel' button is visible at the bottom. On the right side, there is a link to 'See Tutorial: ThingSpeak and MATLAB' and a 'Learn More' link.

Channel ID

Channel Id is the identity of your channel. Note down this.

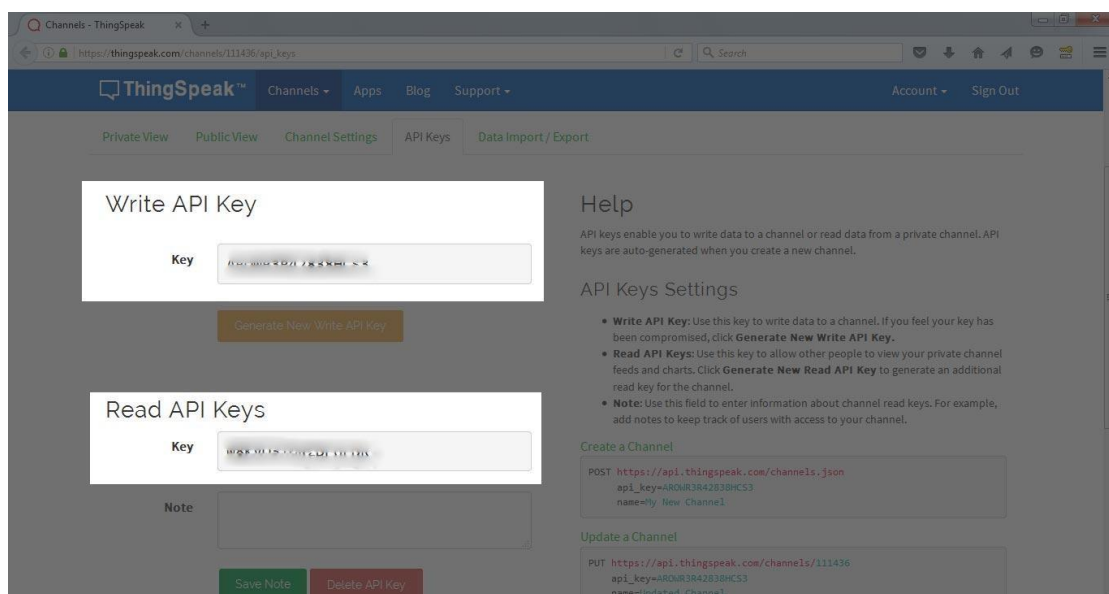


API Keys

API (Application Programming Interface) keys are the keys to access to your channel. In simple language you can understand that these are password to access your channel.

1. To update channel / data logging: API Write Key will be used to access in this mode.
2. To retrieve data: API Read Key will be used to access in this mode.

Click on the API tab to know your API keys. We have blurred our API Keys for security reasons.



3. Accessing Control

You may use following URLs to access your channel –

3.1 To Update channel / data uploading / data logging

URL:http://api.thingspeak.com/update?api_key=YOUR-API&field1=VAR-1&field2=VAR-1

1. YOUR-API: Your API Write Key

2.VAR-1: Pulse Rate Data

3.2 Retrieve channel / data reading

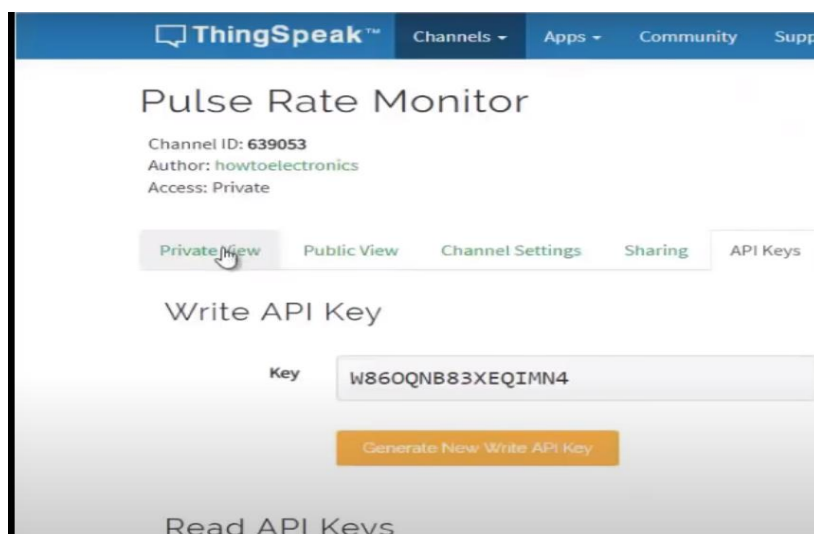
URL:http://api.thingspeak.com/channels/YOUR-CHANNEL-ID/fields/FIELD.json?results=NOS-OF-RESULTS&api_key=YOUR-API
make the following replacements in the above-mentioned URL-

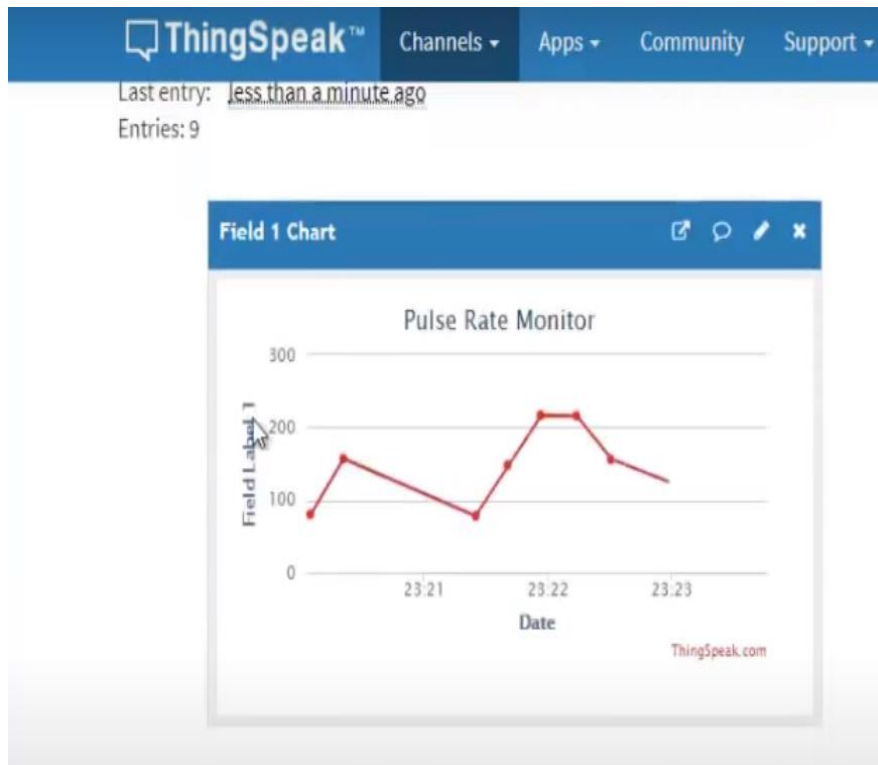
1. YOUR-CHANNEL-ID - Your channel ID
2. FIELD - Field you want to retrieve. Write 1 for Field1, 2 for Field2
3. NOS-OF-RESULTS = The number of rows you want to retrieve.
4. YOUR-API: Your API Read Key

Response: You will get data as per your specifications in JSON format.

4.Reading data through ThingSpeak website.

Login to your account. Select your channel and click on the view as shown in the following image.





APPENDIX-C

SETTING UP WiFi MODULE

Technology goes ahead exponentially with each year whether we do something or not. With the same speed engineers work hard to reduce the size of every electronic device or component and loose most of the wiring. If you look back 20 years ago a device was called portable if you could rise it in your arms and carry up. Today nothing is portable if it can't fit in your pocket.

Nowadays devices, either they are made for regular or industrial use, are based on wireless communication technology and the main reason is not to get rid of wires, but to be able to interconnect between them. Meanwhile buying a wireless device became natural for everyone and price for Wi-Fi Ready equipment lowered with time passing.



The ESP8266 module works with 3.3V only, anything more than 3.7V would kill the module hence be cautions with your circuits. Here is its pins description.

Pin 1: Ground: Connected to the ground of the circuit

Pin 2: Tx/GPIO – 1: Connected to Rx pin of programmer/uC to upload program

Pin 3: GPIO – 2: General purpose Input/output pin

Pin 4 : CH_EN: Chip Enable/Active high

Pin 5: Flash/GPIO – 0: General purpose Input/output pin

Pin 6 : Reset: Resets the module

Pin 7: RX/GPIO – 3: General purpose Input/output pin

Pin 8: Vcc: Connect to +3.3V only

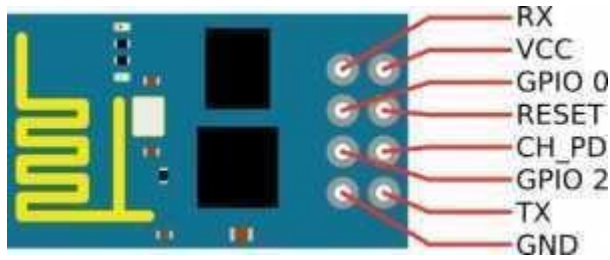
ESP8266 Arduino module comes with PCB trace antenna which seems to have a very good coverage. Other version can have on-board ceramic antenna or *an* external connector which allows you to attach external Wi-Fi antennas modules. ESP-01 has only 6 active pins, although the MCU can support up to 16 I/O. Board dimensions are 14.3 x 24.8 mm.



Over the internet I found that ESP8266 Arduino module, version 01, is sold in two or more versions, which at first glance seem quite the same. After buying both of them i saw that there is a difference in size of the flash memory. You may encounter issues while flashing if you don't make the proper settings according to board specifications.

Although the board default has 2 available GPIOs, you can do some workarounds and use other MCU available pins if you have the proper soldering tools. I managed to use GPIO 16 in order to wake up the device after DEEP SLEEP mode (*explained later in SLEEP MODES*).

Module pin description (pinout)



Pins are arranged in two rows, having 4 on each row. Some models have pin description on the PCB, which make it simple. On the top row you can find following pins from the left to the right:

GND (Ground from power supply)

GPIO2 (Digital I/O programmable)

GPIO0 (Digital I/O programmable, also used for BOOT modes)

RX – UART Receiving channel

On the bottom (second row) you can find:

TX – UART Transmitting channel

CH_PD (enable/power down, must be pulled to 3.3v directly or via resistor)

RESET – reset, must be pulled to 3.3v)

VCC -3.3v power supply

All esp8266 Arduino compatible modules must be powered with DC current from any kind of source that can deliver stable 3.3V and at least 250mA. Also, logic signal is rated at 3.3v and the RX channel should be protected by a 3.3v divisor step-down. You should be careful when using this module with Arduino or other boards which supplies 5v, because this module usually **does** not come with overpower protection and can be easily destroyed.

ESP8266 ARDUINO TUTORIAL

Technology goes ahead exponentially with each year whether we do something or not. With the same speed engineers work hard to reduce the size of every electronic device or component and loose most of the wiring. If you look back 20 years ago a device was called portable if you could rise it in your arms and carry up. Today nothing is portable if it can't fit in your pocket.

Nowadays devices, either they are made for regular or industrial use, are based on wireless communication technology and the main reason is not to get rid of wires, but to be able to interconnect between them. Meanwhile buying a wireless device became natural for everyone and price for Wi-Fi Ready equipment lowered with time passing.

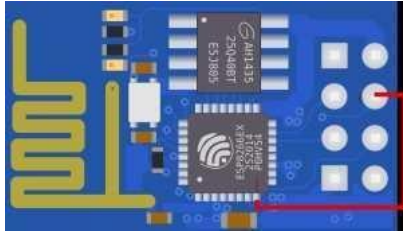
While you read this, a Wi-Fi microchip has no more than 5mm length and can be powered with as low as 10 micro Amps during sleep period. In this article we are going to test one of the cheapest and easy to use Wi-Fi development platform, the ESP8266 Arduino compatible device.

Cheap Wi-Fi solution for your first IoT project – ESP8266 Arduino compatible Wi-Fi module [with 1Mb flash upgrade](#)

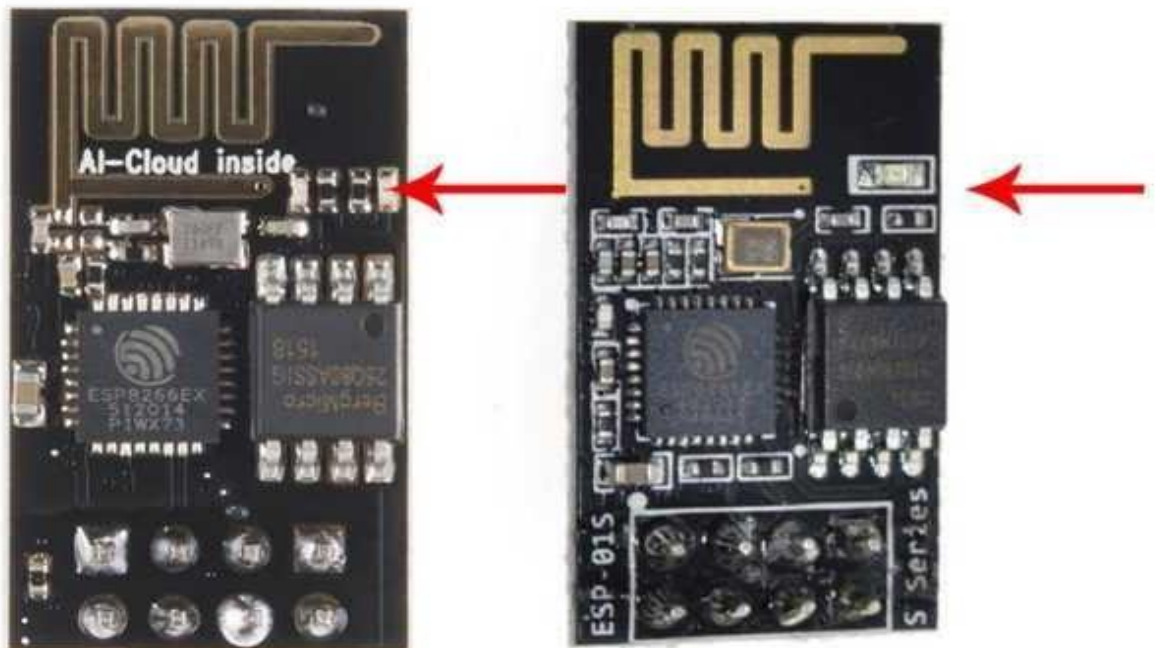
If you are going to use [ESP-01](#) in a project that is powered by batteries or by solar power it is mandatory to know everything about ESP8266 arduino Sleep modes. Current version offers 3 different sleep modes which can be triggered programmatically. ESP8266WiFi library offers specific functions to call sleep modes which can take settings parameters that change the callback jobs after wake-up like waking up with RF module powered off or on.

The most important mode is DEEP_SLEEP because of the very low power consumption rates during sleep. Deep sleep mode is very common in projects that do data-logging at specific intervals and idle between measurements.

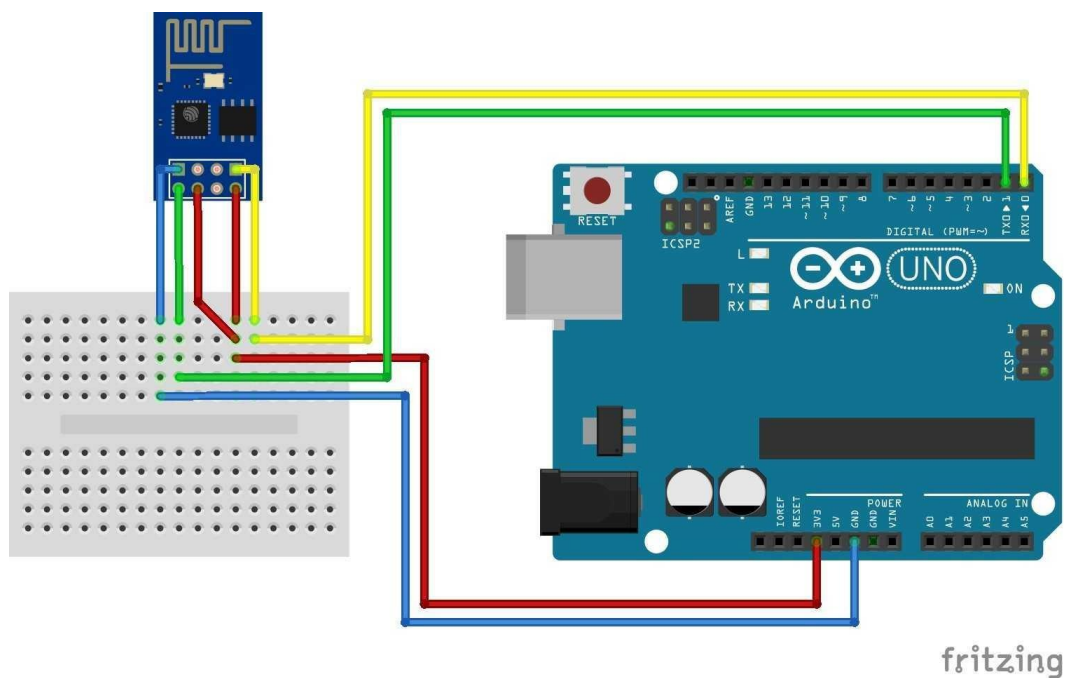
In order to take advantage of this mode when using esp8266 Arduino compatible module, ESP-01 standard, you need to make a little workaround and connect REST pin with the GPIO16 pin.



The differences can be seen in the following pictures:



Arduino Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the *Atmega16U2* (Atmega8U2 up to version R2) programmed as a USB-to-serial converter. In order to use Arduino as a bridge, first you need to load an empty program on it. After doing that, you need to make the following connections in order to work:



After that you should be able to see data and send AT commands in Serial Monitor by selecting Arduino's COM port, setting a proper baudrate, default should be 115200, and make the additional settings to read "Both NL & CR".

Firmware Over The Air (FOTA) solution in every embedded DIY or commercial project is a highly desirable if not a required feature today, when every project core needs to be scalable. So the possibility to upload your code from a remote computer via Wi-Fi connection rather than Serial is a high advantage in every project. First you need FOTA needs prerequisites. First firmware upload needs to be done via Serial, and if the OTA routines are correctly implemented in the program next uploads can be done over the air.

Because the module needs to be exposed wirelessly, the chance of being hacked and loaded with malevolent code exists. You can improve your security by setting a custom port and password. Check functionalities from the [ArduinoOTA library](#) that may help you to boost security. Because of the complexity of these procedures we will cover the full story in a future article, but for now be aware that this option exists and it works pretty well.

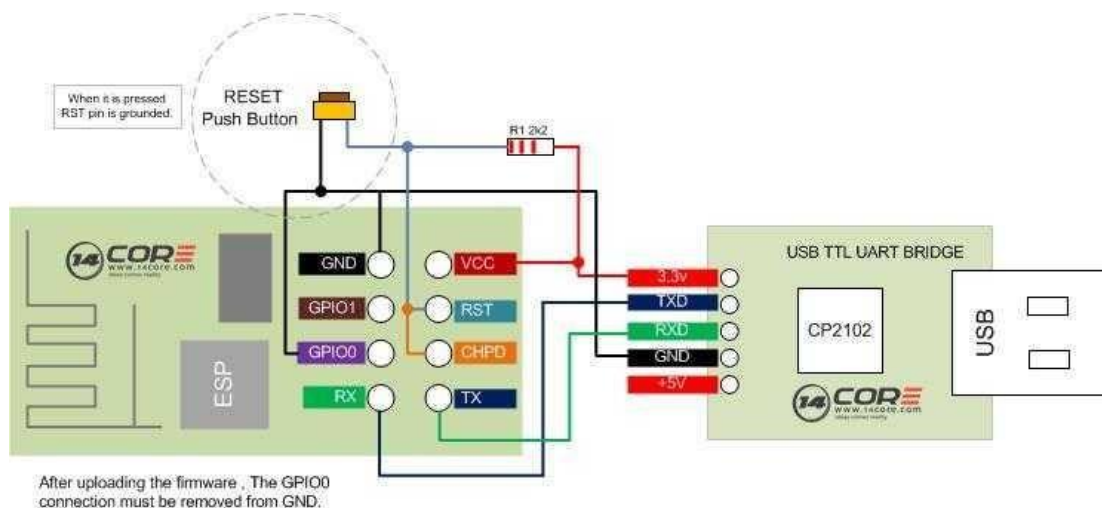
Another way to connect the esp8266 Arduino module to a computer is to use a [TTL or FTDI USB- to-serial](#) dedicated module. There are plenty of them on the market and they are quite cheap, but make no mistake, here quality does matter. You may encounter problems when working with it if ending up with a cheap one because of the differences in connections and also driver compatibility.

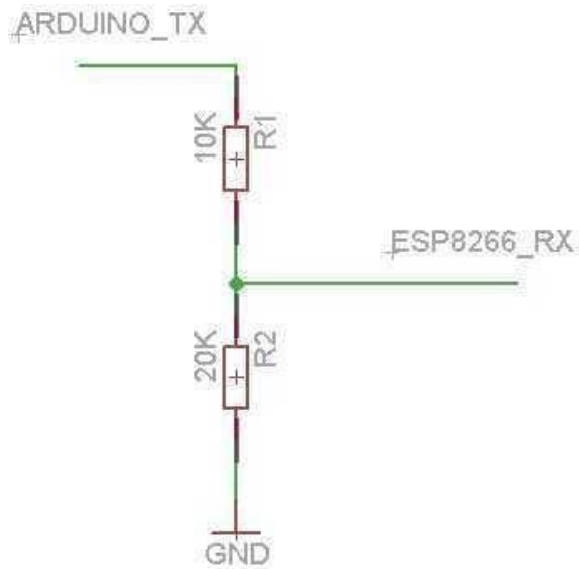
Most used TTL / FTDI converter chips are CH340G, CP2102 and FT232RL. I personally used the first two ones and I have no problem when loading programs. Following connections need to be done:

ESP-01	TTL / FTDI
RX	TX
TX	RX
VCC	3.3v
GND	GND
RST	3.3v / float
CH_PD	3.3v

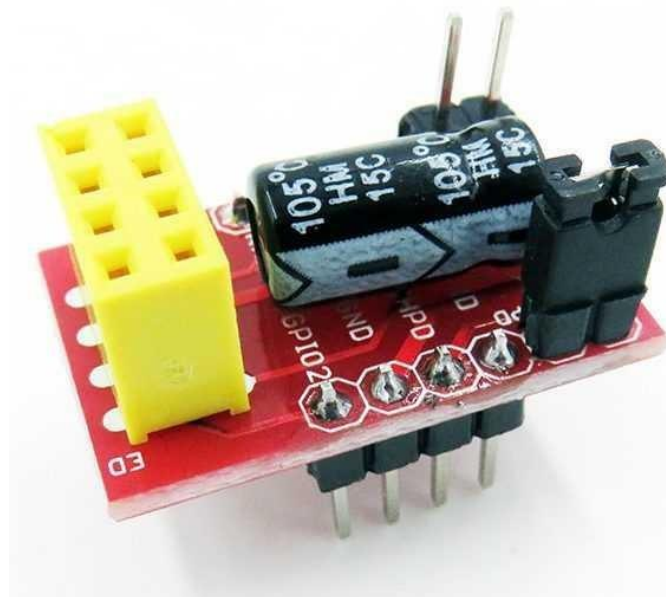
I highly recommend you not to use the TTL 3.3v power supply because most of them are not able to provide enough power to handle the esp8266 Arduino compatible device. The embedded voltage regulator used on this module are not the happiest choice and you may get in trouble if it cannot support ESP peaks. If you choose to use an external power supply don't forget to setup a common ground in order to have a working circuit.

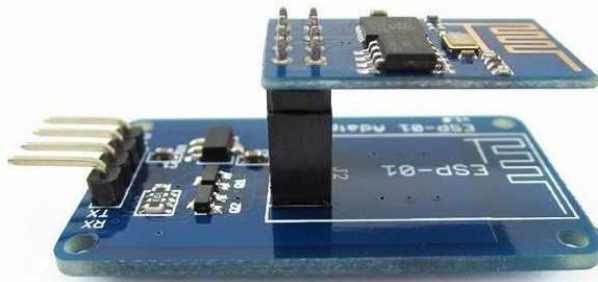
You can find TTL modules that have TX rated at 3.3v, if not, you should step-down the TX channel to protect your ESP-01 module. You can see below a wiring scheme between ESP-01 and CP2102 which includes a reset button connected to ground, and also GPIO0 for boot switch.



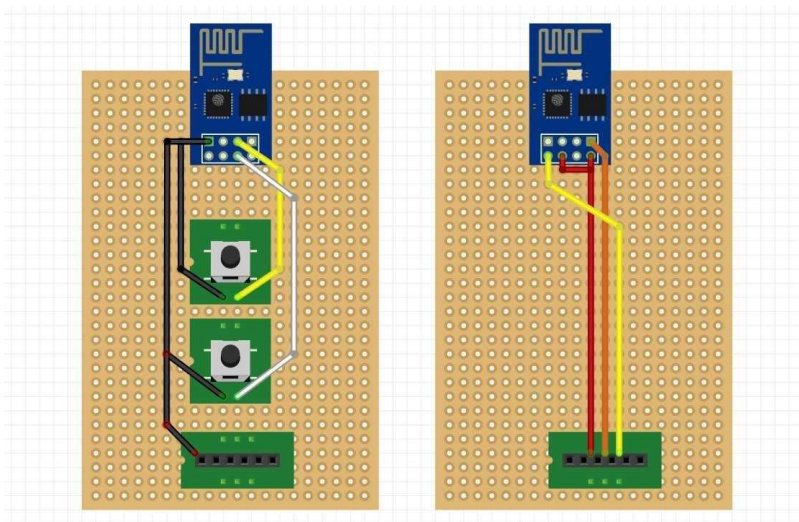


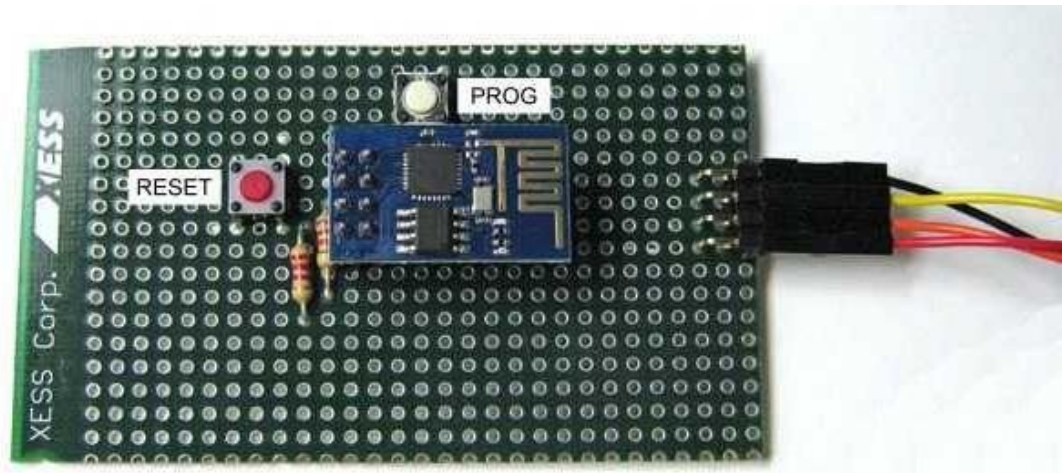
The most comfortable way to communicate with the ESP-01 is by using a dedicated ESP01 programmer module, which can be bought also from the same producers and the costs are very low. This kind of devices can have buttons integrated for RESET and BOOT switching and also come with a dedicated slot. All you need to do is to plug your esp and you're done.





You can also build your own programmer if you have a bit of skills, you don't need to be an expert. Here are few homemade tryouts:





In order to setup your Arduino IDE to work with your esp8266 arduino compatible module you need to make the following steps:

Connect your

ESP8266-01 Module

to PC Open your

Arduino IDE

Go to File -> Preferences

Add this link to

Additional Board

Manager Go to Tools -

> Board Manager

Find ESP8266 board set and activate it

Select Generic ESP8266 board

from Tools->Boards Choose

your programmer COM port

You are ready to go!



Now, to be able to download the program to your ESP-01 module, you first need to put your device in the proper BOOT mode (*Download code from UART*). ESP8266-01 have the following boot modes:

MTDO / GPIO15	GPIO0	GPIO2	Mode	Description
L	L	H	UART	Download code from UART
L	H	H	Flash	Boot from SPI Flash
H	X	X	SDIO	Boot from SD-card

Note that L = LOW (putting to Ground / -3.3v) and H = HIGH (putting to 3.3v)

After resetting the module in *Download code from UART* you should see a message containing “*boot mode:[1,6]*” in the serial monitor, if you are on the correct baudrate. A wrong baudrate setting will display garbage text / characters or nothing at all. After that you should be able to upload your sketch to ESP8266. When upload is done, module should reset itself. Don’t forget to pull HIGH the GPIO0 or the module will get in Download mode again and you will not be able to see it working. The module can be rebooted at anytime by pulling REST pin to LOW. After each reset it will follow the boot sequence and program loading.

Once the ESP8266 board is installed and activated in Arduino IDE, you will be able to include all ESP WiFi libraries and examples that comes with the package. The most used library is ESP8266WiFi which offers many implementation examples like WiFiClient, WiFiServer, WiFiAccessPointetc.

You can find a lot of project examples over the internet, I for example, found:

```
#include<SoftwareSerial.h>

#define DEBUG true

SoftwareSerial esp8266(9,10);

#include <LiquidCrystal.h>

#include<stdlib.h>

LiquidCrystal lcd(12,11,5,4,3,2);

#define SSID "Airtel-MyWiFi-AMF-311WW-95B4"

#define PASS "110391b4"

#define IP "184.106.153.149"// thingspeak.com ip

String msg = "GET /update?key=V7X84JDKYS3Z95Z0"; //change it with your api
key

//Variables

float temp;

int hum;

String tempC;

int error;

int pulsePin = 0; // Pulse Sensor connected to analog pin

int blinkPin = 13; // pin to blink led at each beat

int fadePin = 5;

int fadeRate = 0;

// Volatile Variables, used in the interrupt service routine!
```

```
volatile int BPM; // int that holds raw Analog in 0. updated every 2mS

volatile int Signal; // holds the incoming raw data

volatile int IBI = 600; // int that holds the time interval between beats! Must be
seeded!

volatile boolean Pulse = false; // "True" when heartbeat is detected. "False" when not
a "live beat".

volatile boolean QS = false; // becomes true when Arduino finds a beat.

// Regards Serial OutPut -- Set This Up to your needs

static boolean serialVisual = true; // Set to 'false' by Default.

volatile int rate[10]; // array to hold last ten IBI values

volatile unsigned long sampleCounter = 0; // used to determine pulse timing

volatile unsigned long lastBeatTime = 0; // used to find IBI

volatile int P = 512; // used to find peak in pulse wave

volatile int T = 512; // used to find trough in pulse wave

volatile int thresh = 525; // used to find instant moment of heart beat

volatile int amp = 100; // used to hold amplitude of pulse waveform

volatile boolean firstBeat = true; // used to seed rate array

volatile boolean secondBeat = false; // used to seed rate array
```

```
void setup()

{

    lcd.begin(16, 2);

    lcd.print("Connecting...");

    Serial.begin(9600);

    Esp8266.begin(9600);

    Serial.println("AT");

    esp8266.println("AT");

    delay(5000);

    if(esp8266.find("OK")){

        connectWiFi();

    }

    interruptSetup();

}

void loop()

{

    lcd.clear();

    start:

    error=0;

    lcd.setCursor(0, 0);
```

```
    lcd.print("BPM = ");

    lcd.print(BPM);

    delay (100);

    lcd.setCursor(0, 1); // set the cursor to column 0, line 2

    delay(1000);

    updatebeat();

    if (error==1){

        goto start; //go to label "start"

    }

    delay(1000);

}

void updatebeat()

{

    String cmd = "AT+CIPSTART=\"TCP\", \"";

    cmd += IP;

    cmd += "\",80";

    Serial.println(cmd);

    esp8266.println(cmd);

    delay(2000);

    if(esp8266.find("Error")){

        return;

    }

}
```

```
s}

cmd = msg;

cmd += "&field1=";

cmd += "\r\n";

Serial.print("AT+CIPSEND=");

esp8266.print("AT+CIPSEND=");

Serial.println(cmd.length());

esp8266.println(cmd.length());

if(esp8266.find(">"))

{

    Serial.print(cmd);

    esp8266.print(cmd);

}

else

{

    Serial.println("AT+CIPCLOSE");

    esp8266.println("AT+CIPCLOSE");

    error=1;

}

}
```



```
boolean connectWiFi(){  
  
    Serial.println("AT+CWMODE=1");  
  
    esp8266.println("AT+CWMODE=1");  
  
    delay(2000);  
  
    String cmd="AT+CWJAP=\"";  
  
    cmd+=SSID;  
  
    cmd+="\", \"\"";  
  
    cmd+=PASS;  
  
    cmd+="\"\"";  
  
    Serial.println(cmd);  
  
    esp8266.println(cmd);  
  
    delay(5000);  
  
    if(esp8266.find("OK")){  
  
        Serial.println("OK");  
  
        return true;  
  
    }else{  
  
        return false;  
  
    }  
  
}
```

```
void interruptSetup()

{

    TCCR2A = 0x02; // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO
    INTO CTC MODE

    TCCR2B = 0x06; // DON'T FORCE COMPARE, 256 PRESCALER

    OCR2A = 0x7C; // SET THE TOP OF THE COUNT TO 124 FOR 500Hz
    SAMPLE RATE

    TIMSK2 = 0x02; // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2
    AND OCR2A

    sei(); // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED

}

ISR(TIMER2_COMPA_vect)

{ // triggered when Timer2 counts to 124

    cli();          // disable interrupts while we do this

    Signal = analogRead(pulsePin); // read the Pulse Sensor

    sampleCounter += 2; // keep track of the time in mS

    int N = sampleCounter - lastBeatTime;

    // find the peak and trough of the pulse wave

    if(Signal < thresh && N > (IBI/5)*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 && Signal > P)

{
    // thresh condition helps avoid noise

    P = Signal; // P is the peak

}

if (N > 250)

{
    // avoid high frequency noise

    if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) )

    {

        Pulse = true;           // set the Pulse flag when there is a pulse

        digitalWrite(blinkPin,HIGH);      // turn on pin 13 LED

        IBI = sampleCounter - lastBeatTime;    // time between beats in mS

        lastBeatTime = sampleCounter;    // keep track of time for next pulse


        if(secondBeat){           // if this is the second beat

            secondBeat = false;    // clear secondBeat flag

            for(int i=0; i<=9; i++){ // seed the running total to get a realistic BPM at
startup
                rate[i] = IBI;

            }

        }

    }

}
```

```
}

    if(firstBeat){                // if it's the first time beat is found

        firstBeat = false;        // clear firstBeat flag

        secondBeat = true;        // set the second beat flag

        sei();                    // enable interrupts again

        return;                  // IBI value is unreliable so discard it

    }

    word runningTotal = 0; // clear the runningTotal variable

    for(int i=0; i<=8; i++){      // shift data in the rate array

        rate[i] = rate[i+1];      // and drop the oldest IBI value

        runningTotal += rate[i];  // add up the 9 oldest IBI values

    }

    rate[9] = IBI; // add the latest IBI to the rate array

    runningTotal += rate[9]; // add the latest IBI to runningTotal

    runningTotal /= 10; // average the last 10 IBI values

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

    QS = true; // set Quantified Self flag

}

}
```

BIBLIOGRAPHY

- <https://www.google.com/search?q=how2electronics&oq=&aqs=chrome.4.35i39i362l8...8.1156852762j0j15&sourceid=chrome&ie=UTF-8>
- <https://www.irjet.net/archives/V5/i3/IRJET-V5I3586.pdf>
- <https://circuitdigest.com/microcontroller-projects/arduino-smart-blind-stick>
- www.enggjournals.com/ijet/docs/IJET17-09-05-302.pdf
- <https://www.google.com/search?q=thingspeak&oq=th&aqs=chrome.1.69i57j69i59l3j0i67i457j0l2j46i433.1803j0j15&sourceid=chrome&ie=UTF-8>