CHAPTER I

INTRODUCTION

The objective of this project is to develop a furnance capacity monitor which gives the exact quantity the furnance hold and release while operating it. This project is aimed to work on a industrial purpose .the old product containt a conveyer system which carry a person and that person see the capacity of the funance in the middle of the furnance mouth . This is so dangerous and the heat from the furnance can kill a person. To avoid a chance of this we develop a monitoring tool using Internet of things

1.1 WORKFLOW

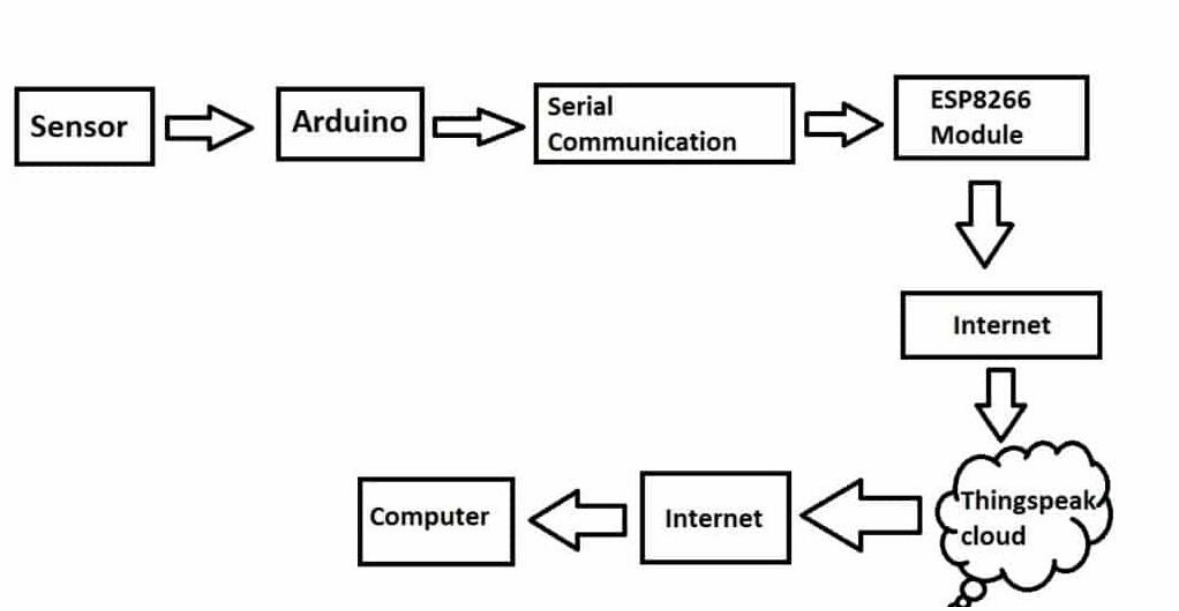


Fig.1.1 Workflow

1.1.1 WORKFLOW DESCRIPTION

The basic block diagram of the Sanitizing Bot is shown in the above figure. Mainly this block diagram consists of the following essential blocks.

(i) Sensor

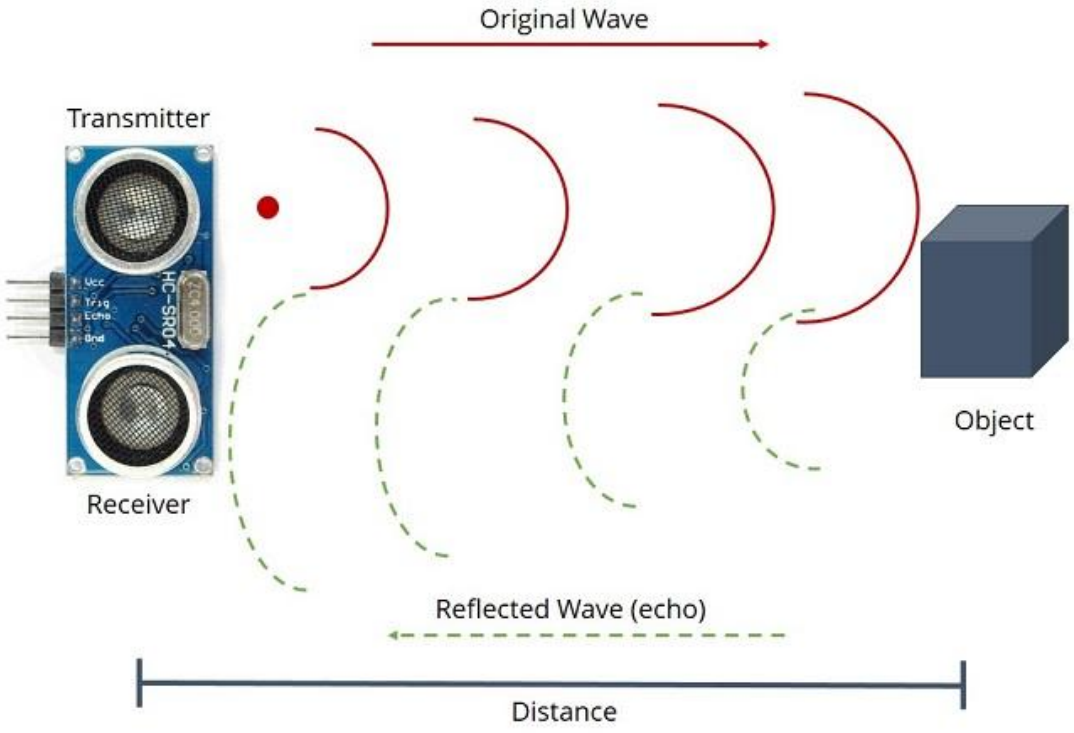
(ii) ESP 8266

(iii) Internet

(iv) Think speak cloud

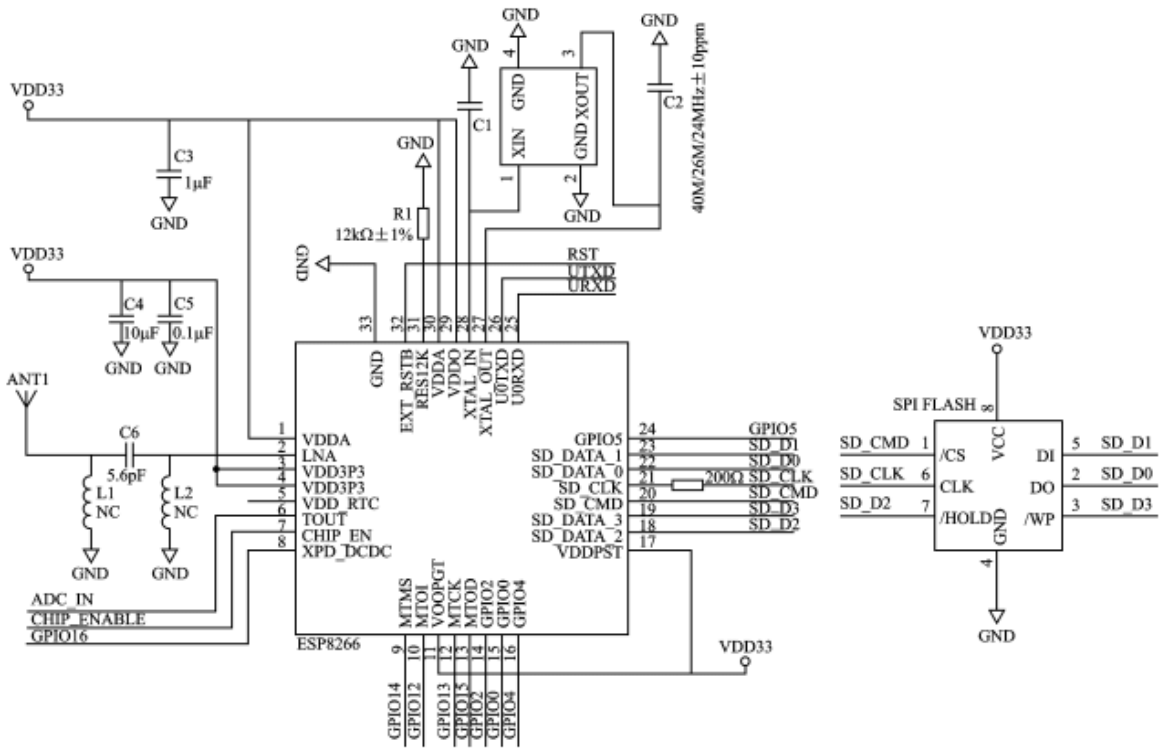
Sensor

Ultrasonic sensors work by sending out a sound wave at a frequency above the range of human hearing. The transducer of the sensor acts as a microphone to receive and send the ultrasonic sound. Our ultrasonic sensors, like many others, use a single transducer to send a pulse and to receive the echo. The sensor determines the distance to a target by measuring time lapses between the sending and receiving of the ultrasonic pulse.



ESP 8266

The ESP8266 is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif Systems in Shanghai, China. The chip was popularized in the English-speaking maker community in August 2014 via the ESP-01 module, made by a third-party manufacturer Ai-Thinker.



Internet

a global computer network providing a variety of information and communication facilities, consisting of interconnected networks using standardized communication protocols. "the guide is also available on the internet"

Think speak cloud

ThingSpeak is IoT Cloud platform where you can send sensor data to the cloud. You can also analyze and visualize your data with MATLAB or other software, including making your own applications. The ThingSpeak service is operated by MathWorks. In order to sign up for ThingSpeak, you must create a new MathWorks Account or log in to your existing MathWorks Account.

ThingSpeak is free for small non-commercial projects. ThingSpeak includes a Web Service (REST API) that lets you collect and store sensor data in the cloud and develop Internet of Things applications. It works with Arduino, Raspberry Pi and MATLAB (premade libraries and APIs exists) But it should work with all kind of Programming Languages, since it uses a REST API and HTTP.

Universal connection Protocol

Separate terminal indicate the working of the module o/p is provided which are used as clocks for a subsequent stages without extra logic, thus simplifying capture designs. Individual preset inputs allow the circuit to be used as programmable shots. Both the parallel load and the master reset inputs asynchronously override clocks.

a) Low Power : 95 mw
b) High Speed :40MHz

c) Synchronous counting

d) Asynchronous Master Reset and parallel Load

e) Individual preset inputs

f) Cascading Circuitry Internally

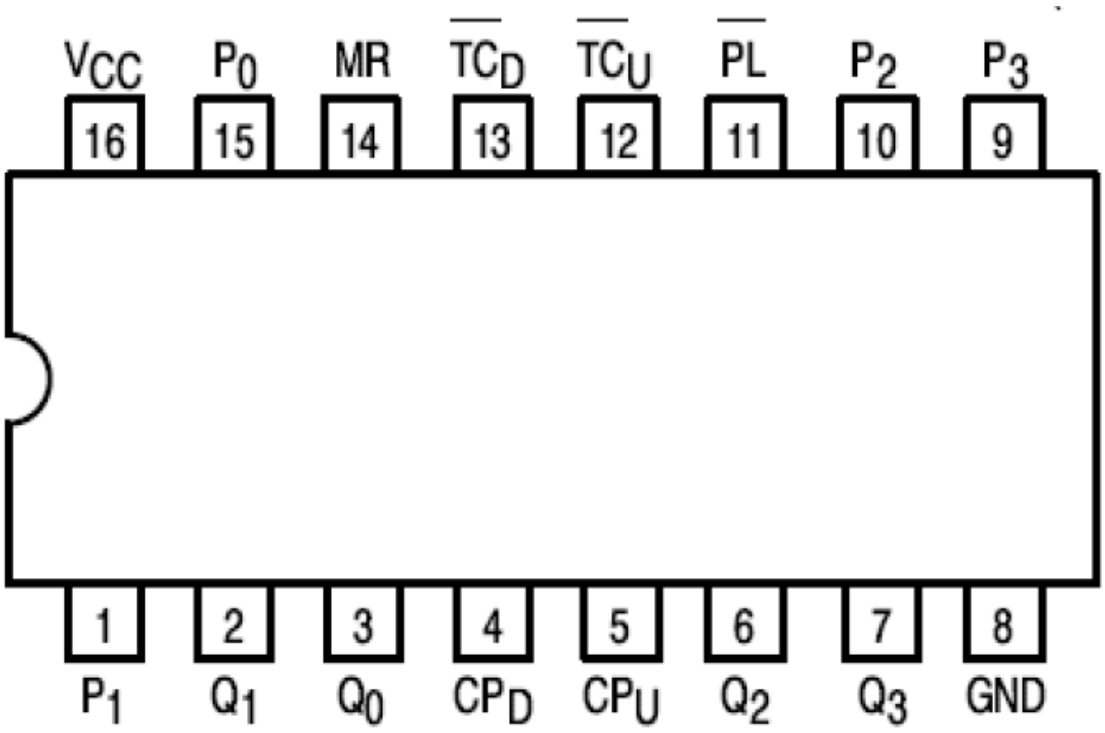


Figure 1.2 Chipset Diagram

PIN NAMES		LOADING (Note a)	
		HIGH	LOW
CP_U	Count Up Clock Pulse Input	0.5 U.L.	0.25 U.L.
CP_D	Count Down Clock Pulse Input	0.5 U.L.	0.25 U.L.
$\overline{MR}$	Asynchronous Master Reset (Clear) Input	0.5 U.L.	0.25 U.L.
PL	Asynchronous Parallel Load (Active LOW) Input	0.5 U.L.	0.25 U.L.
P_n	Parallel Data Inputs	0.5 U.L.	0.25 U.L.
$\overline{Q_n}$	Flip-Flop Outputs (Note b)	10 U.L.	5 (2.5) U.L.
$\overline{TC_D}$	Terminal Count Down (Borrow) Output (Note b)	10 U.L.	5 (2.5) U.L.
TC_U	Terminal Count Up (Carry) Output (Note b)	10 U.L.	5 (2.5) U.L.

NOTES:
a. 1 TTL Unit Load (U.L.) = 40 µA HIGH/1.6 mA LOW.
b. The Output LOW drive factor is 2.5 U.L. for Military (54) and 5 U.L. for Commercial (74) Temperature Ranges.

Figure 1.3 Chipset Explained

Interaction

The working of the project is explained below.

Working of Modules

- The kit is connected with Blynk application via wireless connection.
- The ultrasonic sensor is connected to the node mcu
- Eacha nd every reading of the sensor is send to the node mcu .
- The nodemcu send the data to the app interface of the blynk via wifi connection .

CHAPTER 2

SOFTWARE DESCRIPTION

2.1.ARDUINO IDE- (Universal Content)

2.1.1 Introduction

- Arduino IDE is open source software that is mainly used for writing and compiling the code into the Arduino Module.
- It is official Arduino software, making code compilation too easy that even a common person with no prior technical knowledge can get their feet wet with the learning process.
- It is easily available for operating systems like MAC, Windows, and Linux and runs on the Java Platform that comes with inbuilt functions and commands that play a vital role for debugging, editing and compiling the code in the environment.
- A range of Arduino modules available including ESP 8266, Arduino Mega, Arduino Leonardo, Arduino Micro and many more.
- Each of them contains a microcontroller on the board that is actually programmed and accepts the information in the form of code.
- The main code, also known as a sketch, created on the IDE platform will ultimately generate a Hex File which is then transferred and uploaded in the controller on the board.
- The IDE environment mainly contains two basic parts: Editor and Compiler where former is used for writing the required code and later is used for compiling and uploading the code into the given Arduino Module.
- This environment supports both C , C++ and Python languages.

2.2 EMBEDDED C PROGRAMMING

Embedded C Programming is the soul of the processor functioning inside each and every embedded system we come across in our daily life, such as mobile phone, washing machine, and digital camera.

Each processor is associated with embedded software. The first and foremost thing is the embedded software that decides functioning of the embedded system. Embedded C language is most frequently used to program the microcontroller.

Earlier, many embedded applications were developed using assembly level programming. However, they did not provide portability. This disadvantage was overcome by the advent of various high level languages like C, Pascal, and COBOL. However, it was the C language that got extensive acceptance for embedded systems, and it continues to do so. The C code written is more reliable, scalable, and portable; and in fact, much easier to understand.

About C Language

C language was developed by Dennis Ritchie in 1969. It is a collection of one or more functions, and every function is a collection of statement performing a special task. C language is a middle-level language as it supports high-level applications and low-level applications. Before going into the details of embedded C programming, we should know about RAM memory organization.

Salient Features Of The Language

- C language is software designed with different keywords, data types, variables, constants, etc.
- Embedded C is a generic term given to a programming language written in C, which is associated with particular hardware architecture.
- Embedded C is an extension to the C language with some additional header files. These header files may change from controller to controller.
- The microcontroller 8051 #include<reg51.h> is used.

The embedded system designers must know about the hardware architecture to write programs. These programs play prominent role in monitoring and controlling external devices.

They also directly operate and use the internal architecture of the microcontroller, such as interrupt handling, timers, serial communication and other available features.

About PYHTON Language

Python programming powers intuitive interfaces of intelligent and effective Internet of Things (IoT) systems that are paramount in remote sensor networks, big data and data analysis, automation, and machine learning. IoT applications function efficiently with the help of Python **libraries/packages** which include:

NUMPY

Numpy is a scientific computing package that helps to create datasets to test with the time series data in IoT. Numpy features are used in IoT to read sensor bulk data from the database inbuilt functions in the system

SOCKETS AND MYSQLDB

Sockets that facilitate networking in IoT devices include TCP/IP and UDP, which are compatible to work with Python packages. TCP/IP and UDP act as transport layer protocols for communication. The MySQLdb is a go-to relational format database that helps in the development of remote stores for the IoT system.

MATPLOTLIB

To get data insights, matplotlib visualizes the most paramount operations by giving a variety of graphs to represent the data.

REQUESTS, TKINTER AND TENSORFLOW

To make HTTP calls and parse responses in Python, the **request package** acts as a major protocol for data exchanges. **Tkinter GUI** puts the aspects of Python script in a controlled distribution, which enables functional testing and repeated executions in IoT Python devices. Therefore, the numerical computations of machine learning initiated into the IoT systems utilize the representation in data flow graphs dealing with huge non-linear datasets and deep learning aspects.

Salient Features Of The Language

- **Python is a high-level programming language**.
  Python is very easy to learn the language as compared to other languages like C, C#, Javascript, Java, etc. It is very easy to code in python language and anybody can learn python basics in a few hours or days. It is also a developer-friendly language.
- **Free and Open Source:**
  Python language is freely available at the official website and you can download it from the given download link below click on the Download Python keyword.Since it is open-source, this means that source code is also available to the public. So you can download it as, use it as well as share it.
- **Object-Oriented Language:**
  One of the key features of python is Object-Oriented programming. Python supports object-oriented language and concepts of classes, objects encapsulation, etc.
- **GUI Programming Support:**
  Graphical User interfaces can be made using a module such as PyQt5, PyQt4, wxPython, or Tk in python.
  PyQt5 is the most popular option for creating graphical apps with Python.
- **High-Level Language:**
  Python is a high-level language. When we write programs in python, we do not need to remember the system architecture, nor do we need to manage the memory.
- **Extensible feature:**
  Python is a Extensible language. We can write us some Python code into C or C++ language and also we can compile that code in C/C++ language.
- **Python is Portable language:**
  Python language is also a portable language. For example, if we have python code for windows and if we want to run this code on other platforms such as Linux, Unix, and Mac then we do not need to change it, we can run this code on any platform.
- **Python is Integrated language:**
  Python is also an Integrated language because we can easily integrated python with other languages like c, c++, etc.
- **Interpreted Language:**
  Python is an Interpreted Language because Python code is executed line by line at a time. like other languages C, C++, Java, etc. there is no need to compile python code this makes it easier to debug our code. The source code of python is converted into an immediate form called bytecode.
- **Large Standard Library**
  Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.
- **Dynamically Typed Language:**
  Python is a dynamically-typed language. That means the type (for example- int, double, long, etc.) for a variable is decided at run time not in advance because of this feature we don't need to specify the type of variable.

C	PYTHON
C is procedural-oriented programming language.	Python is object-oriented programming language.
It is compulsory to declare the datatype of variables.	Type declaration is not required in Python.
C has for, while, do-while loops.	Python has for, while loops only.
C has switch statements.	Python does not have switch statements.
C does not allows Exception Handling.	Python allows Exception Handling.

Fig 2.1 Difference between C and PYTHON

Additional Features of The Embedded Software

Datatypes

The data type refers to an extensive system for declaring variables of different types like integer, character, float, etc. The embedded C software uses four data types that are used to store data. The size and range of different data types on a 32-bit machine is given in the following table. The size and range may vary on machines with different word sizes.

Data Type	Size	Range
Char or signed char	1byte	-128 to +128
Unsigned char	1byte	0 to 255
Int or singed int	2byte	-32768 to 32767
Unsigned int	2byte	0 to 65535

Table 2.1Datatypes of Embedded C

Keywords

There are certain words that are reserved for doing specific tasks. These words are known as keywords. They are standard and predefined in the Embedded C. Keywords are always written in lowercase. These keywords must be defined before writing the main program. The basic keywords of embedded software are given below:

Name	Funtion
sbit	Accessing of single bit
bit	Aceessing of bit addressable memory of RAM
sfr	Accessing of sfr register by another name

Figure 2.2 Keywords used in Embedded C

SBIT: This data type is used in case of accessing a single bit of SFR register.

- Syntax: sbit variable name = SFR bit ;
- Ex: sbit a=P2^1;
- Explanation: If we assign p2.1 as 'a' variable, then we can use 'a' instead of p2.1 anywhere in the program, which reduces the complexity of the program.

BIT: This data type is used for accessing the bit addressable memory of RAM (20h-2fh).

- Syntax: bit variable name;
- Ex: bit c;
- Explanation: It is a bit sequence setting in a small data area that is used by a program to remember something.

SFR: This data type is used for accessing a SFR register by another name. All the SFR registers must be declared with capital letters.

- Syntax: SFR variable name = SFR address of SFR register;
- Ex: SFR port0=0x80;

- Explanation: If we assign 0x80 as ‘port0’, then we can use 0x80 instead of port0 anywhere in the program, which reduces the complexity of the program.

SFR REGISTER: The SFR stands for ‘Special Function Register’. Microcontroller 8051 has 256 bytes of RAM memory. This RAM is divided into two parts: the first part of 128 bytes is used for data storage, and the other of 128 bytes is used for SFR registers. All peripheral devices like I/O ports, timers and counters are stored in the SFR register, and each element has a unique address.

The Structure Of An Embedded C Program

- comments
- preprocessor directives
- global variables
- main() function

{

Comments: In embedded C programming language, we can place comments in our code which helps the reader to understand the code easily.

Preprocessor Directives

All the functions of the embedded C software are included in the preprocessor library like “#includes<reg51.h>, #defines”. These functions are executed at the time of running the program.

Global Variable

A global variable is a variable that is declared before the main function, and can be accessed on any function in the program.

```
#include<reg51.h>
sbit a=p1^5;          /*global declaration*/

void main()
{
```

Local Variable

A local variable is a variable declared within a function, and it is valid only to be used within that function.

```
void main()
{
    unsigned int k;       /*local declaration*/
    a=0x00;
    while(1)
    {
```

Main () Function

The execution of a program starts with the main function. Every program uses only one main () function.

Advantages Of Embedded C Program

- Its takes less time to develop application program.
- It reduces complexity of the program.
- It is easy to verify and understand.
- It is portable in nature from one controller to another.

2.3 SOURCE CODE

```
#define BLYNK_PRINT Serial


#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
#define TRIGGERPIN D1
#define ECHOPIN    D2
// You should get Auth Token in the Blynk App.
// Go to the Project Settings (nut icon).
char auth[] = "Your auth token";

// Your WiFi credentials.
// Set password to "" for open networks.
char ssid[] = "your wifi SSID";
char pass[] = "your password";

WidgetLCD lcd(V1);

void setup()
{
  // Debug console
  Serial.begin(9600);
pinMode(TRIGGERPIN, OUTPUT);
  pinMode(ECHOPIN, INPUT);
  Blynk.begin(auth, ssid, pass);
  // You can also specify server:
  //Blynk.begin(auth, ssid, pass, "blynk-cloud.com", 8442);
  //Blynk.begin(auth, ssid, pass, IPAddress(192,168,1,100), 8442);

  lcd.clear(); //Use it to clear the LCD Widget
```

```
  lcd.print(0, 0, "Distance in cm"); // use: (position X: 0-15, position Y: 0-1, "Message you want to print")
  // Please use timed events when LCD printintg in void loop to avoid sending too many commands
  // It will cause a FLOOD Error, and connection will be dropped
}

void loop()
{
  lcd.clear();
  lcd.print(0, 0, "Distance in cm"); // use: (position X: 0-15, position Y: 0-1, "Message you want to print")
  long duration, distance;
  digitalWrite(TRIGGERPIN, LOW);
  delayMicroseconds(3);

  digitalWrite(TRIGGERPIN, HIGH);
  delayMicroseconds(12);

  digitalWrite(TRIGGERPIN, LOW);
  duration = pulseIn(ECHOPIN, HIGH);
  distance = (duration/2) / 29.1;
  Serial.print(distance);
  Serial.println("Cm");
  lcd.print(7, 1, distance);
  Blynk.run();

  delay(3500);

}

#define BLYNK_PRINT Serial
#include <ESP8266WiFi.h>
#include <BlynkSimpleEsp8266.h>
```

```
//Motor PINs
#define ENA D0
#define IN1 D1
#define IN2 D2
#define IN3 D3
#define IN4 D4
#define ENB D5

bool forward = 0;
bool backward = 0;
bool left = 0;
bool right = 0;
int Speed;
char auth[] = "OaGy6r4LEojCMlul0RXnqCzcIbvKW020"; //Enter your Blynk application
auth token
char ssid[] = "project"; //Enter your WIFI name
char pass[] = "demoworld"; //Enter your WIFI passowrd


void setup() {
  Serial.begin(9600);
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENB, OUTPUT);

  Blynk.begin(auth, ssid, pass);
}
```

```
BLYNK_WRITE(V0) {
  forward = param.asInt();
}

BLYNK_WRITE(V1) {
  backward = param.asInt();
}

BLYNK_WRITE(V2) {
  left = param.asInt();
}

BLYNK_WRITE(V3) {
  right = param.asInt();
}

BLYNK_WRITE(V4) {
  Speed = param.asInt();
}

void smartcar() {
  if (forward == 1) {
    carforward();
    Serial.println("carforward");
  } else if (backward == 1) {
    carbackward();
    Serial.println("carbackward");
  } else if (left == 1) {
    carturnleft();
    Serial.println("carfleft");
```

```
  } else if (right == 1) {
    carturnright();
    Serial.println("carright");
  } else if (forward == 0 && backward == 0 && left == 0 && right == 0) {
    carStop();
    Serial.println("carstop");
  }
}
void loop() {
  Blynk.run();
  smartcar();
}

void carforward() {
  analogWrite(ENA, Speed);
  analogWrite(ENB, Speed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
void carbackward() {
  analogWrite(ENA, Speed);
  analogWrite(ENB, Speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}
void carturnleft() {
  analogWrite(ENA, Speed);
```

```
  analogWrite(ENB, Speed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
}
void carturnright() {
  analogWrite(ENA, Speed);
  analogWrite(ENB, Speed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
}
void carStop() {
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, LOW);
```

CHAPTER 3

SYSTEM STUDY

In this chapter, we explain the existing and the proposed sustem in detail with a pros and cons of the proposed systems and we explained more about the danger in working in metal industry.

Working with molten metal has always been and will always be a dangerous profession. While it is impossible to remove the risk from melting metal, it is possible to make the melt shop an accident-free workplace.No matter how carefully equipment is manufactured, workers are trained or procedures are followed, the possibility of an accident can occur wherever molten metal is present.

Melt shop supervisors play a key role in assuring safe operation of equipment. Safety training needs to extend beyond melt shop workers. Management MUST make it a commitment to make safety a key corporate value.

3.1 EXISTING SYSTEMS

The Existing system of this project uses a conveyer belt to carry a person around the furnance and there is a chance of high temperature air can surround a worker it may kill him. To avoide this kind of problems we introduce this furnance monitoring system.

3.2 PROPOSED SYSTEM

In our project, we develop a furnance monitor to see the quantity of the product in the furnce to be filled and maintain a optimal quantity to avoid overflow and low quantity cases , we develop a software base architechture to enable the service over internet of things .

Our project have a app based interface which connect with the hardware interface to do things great and give the acurate data , it give the data which is not given by a human expert .. so we use this kind of service based architectures to plan for it.

CHAPTER 4

CIRCUIT DESCRIPTION

1. Start
2. The sensor senses the change and convert it into electrical reading.
3. The electrical reading is send for a ardino architecture.
4. That arduino send the signal to the seriel communication interface.
5. The serial communication signal is processed in the micro controller ESP 8266.
6. That processed output is send to the internet .
7. By travelling in the internet that data drops to Cloud system
8. The cloud system send data for a device with authorized IP Address .
9. Once identified the data is sedn via internet to the host device.
10. Finally it comes to our mobile appliation.

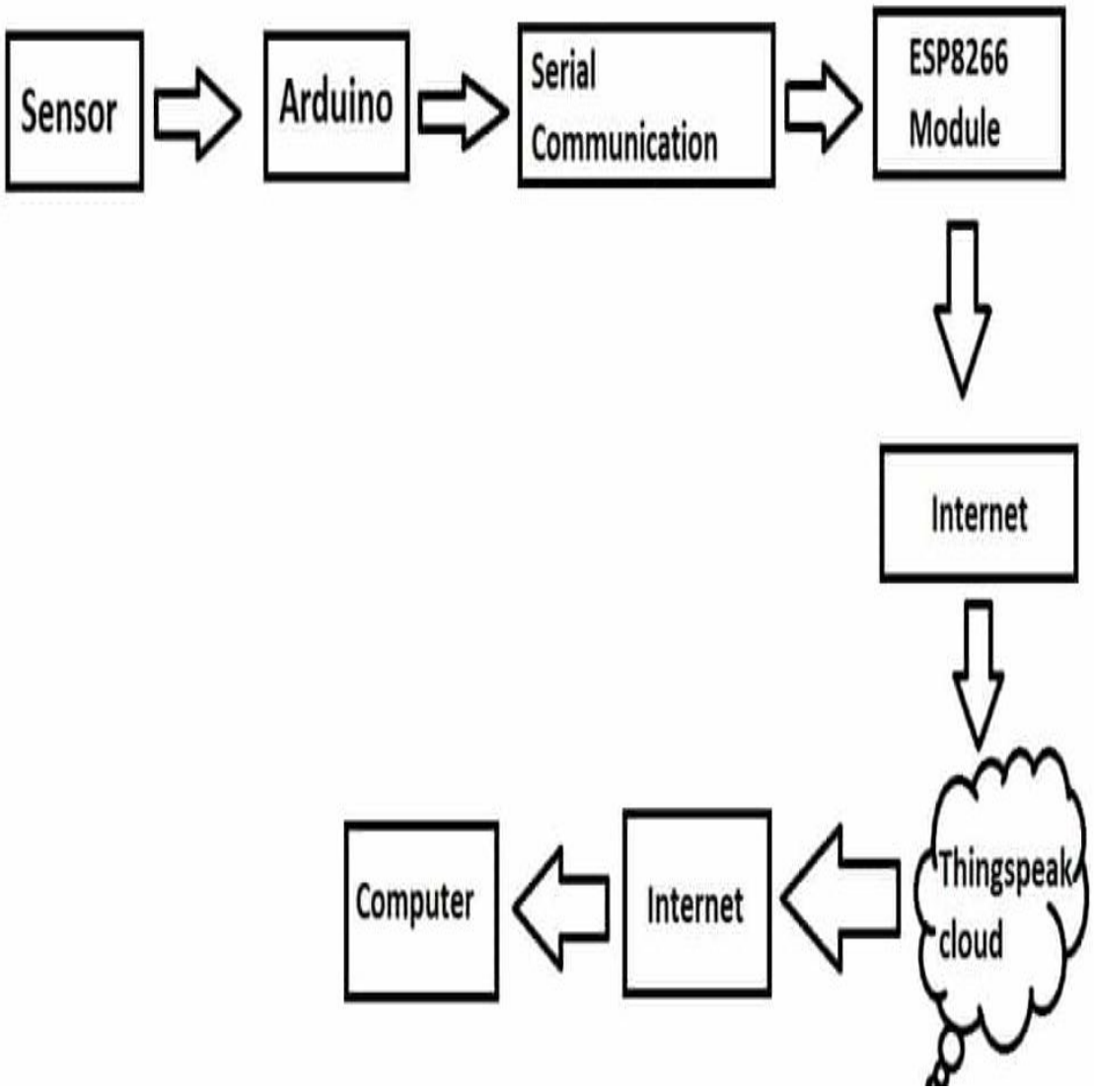


Fig. 4.1. Flow Diagram

CHAPTER 5

COMPONENTS DESCRIPTION

5.1 LIST OF COMPONENTS

- NODEMCU ESP 8266
- ULTRASONIC SENSOR
- ULTRA VOILET LED
- BREAD BOARD
- JUMPER WIRES
- APP INTERFACE

5.1.1 Microcontroller- NODEMCU ESP 8266

Embedded System- (Universal Content)

Introduction

A system is something that maintains its existence and functions as a whole through the interaction of its parts. E.g. Body, Mankind, Access Control, etc A system is a part of the world that a person or group of persons during some time interval and for some purpose choose to regard as a whole, consisting of interrelated components, each component characterized by properties that are selected as being relevant to the purpose.

- Embedded System is a combination of hardware and software used to achieve a single specific task.
- Embedded systems are computer systems that monitor, respond to, or control an external environment.
- Environment connected to systems through sensors, actuators and other I/O interfaces.
- Embedded system must meet timing & other constraints imposed on it by environment.

An embedded system is a microcontroller-based, software driven, reliable, real-time control system, autonomous, or human or network interactive, operating on diverse physical variables and in diverse environments and sold into a competitive and cost conscious market.

An embedded system is not a computer system that is used primarily for processing, not a software system on PC or UNIX, not a traditional business or scientific application. High-end embedded & lower end embedded systems. High-end embedded system - Generally 32, 64 Bit Controllers used with OS. Examples Personal Digital Assistant and Mobile phones etc .Lower end embedded systems - Generally 8,16 Bit Controllers used with an minimal operating systems and hardware layout designed for the specific purpose. Examples Small controllers and devices in our everyday life like Washing Machine, Microwave Ovens, where they are embedded in.

Block Diagram for Embedded System

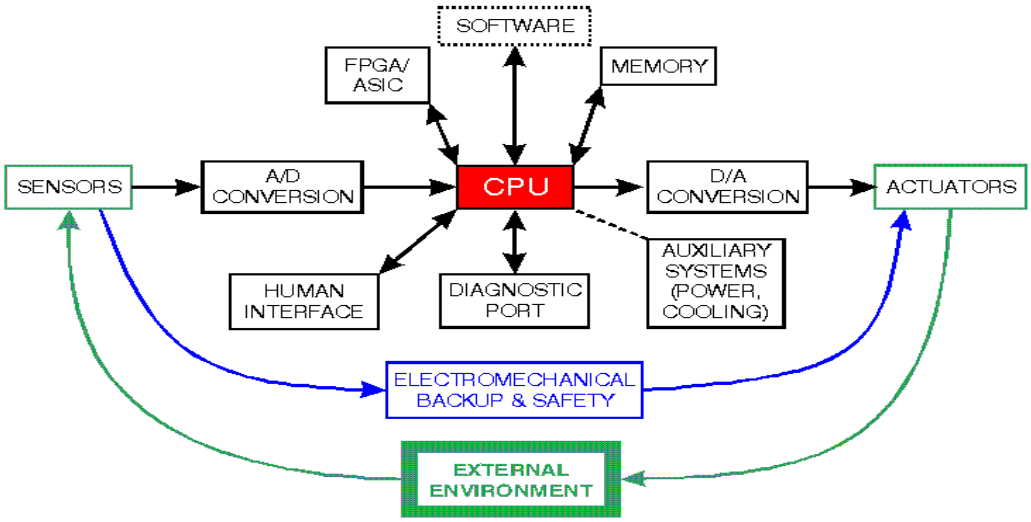


Figure 5.1 Block Diagram of Embedded Systems

Design of Embedded System

Like every other system development design cycle embedded system to have a design cycle. The flow of the system will be like as given below. For any design cycle these will be the implementation steps. From the initial state of the project to the final fabrication the design

considerations will be taken like the software consideration and the hardware components, sensor, input and output. The electronics usually uses either a microprocessor or a microcontroller. Some large or old systems use general-purpose mainframe computers or minicomputers.

Embedded Product Development Life Cycle

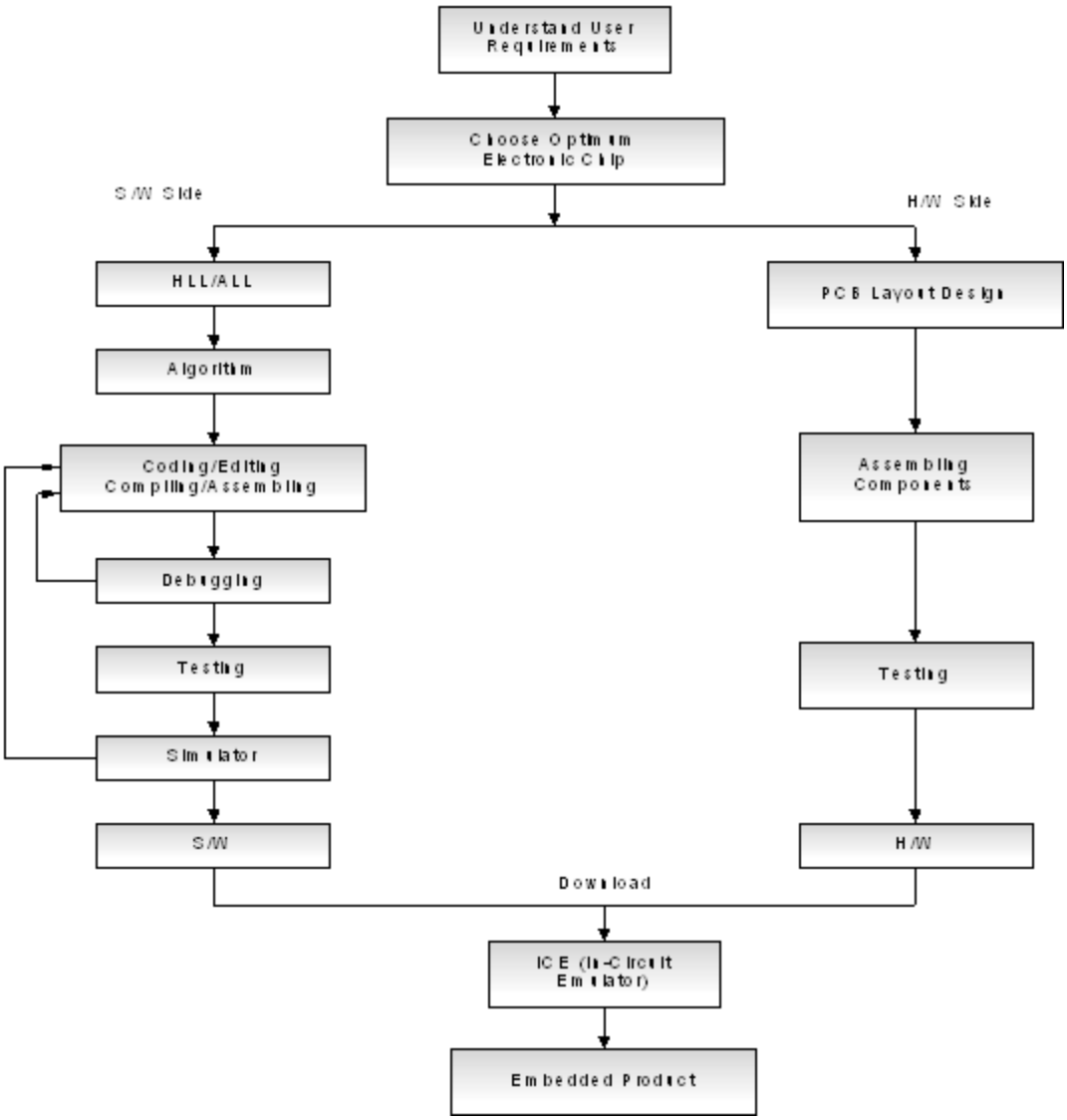


Fig.5.2 Embedded Product Development Life Cycle

Classification

- Real Time Systems.
- RTS is one which has to respond to events within a specified deadline.
- A right answer after the dead line is a wrong answer
- RTS classification

Hard Real Time Systems

"Hard" real-time systems have very narrow response time.

Example: Nuclear power system, Cardiac pacemaker.

Soft Real Time Systems

"Soft" real-time systems have reduced constrains on "lateness" but still must operate very quickly and repeatable

Example: Railway reservation system – takes a few extra seconds the data remains valid.

ESP 8266 Introduction

The ESP 8266 is a microcontroller board based on a removable, dual-inline-package. The board is equipped with sets of digital and analog input/output (I/O) pins that may be interfaced to various expansion boards(shields) and other circuits. The board has 14 digital I/O pins (six capable of PWM output), 6 analog I/O pins, and is programmable with the Arduino IDE (Integrated Development Environment), via a type B USB cable. it has one 5v output pin and one 3.3v output pin. it also contains three ground pins.

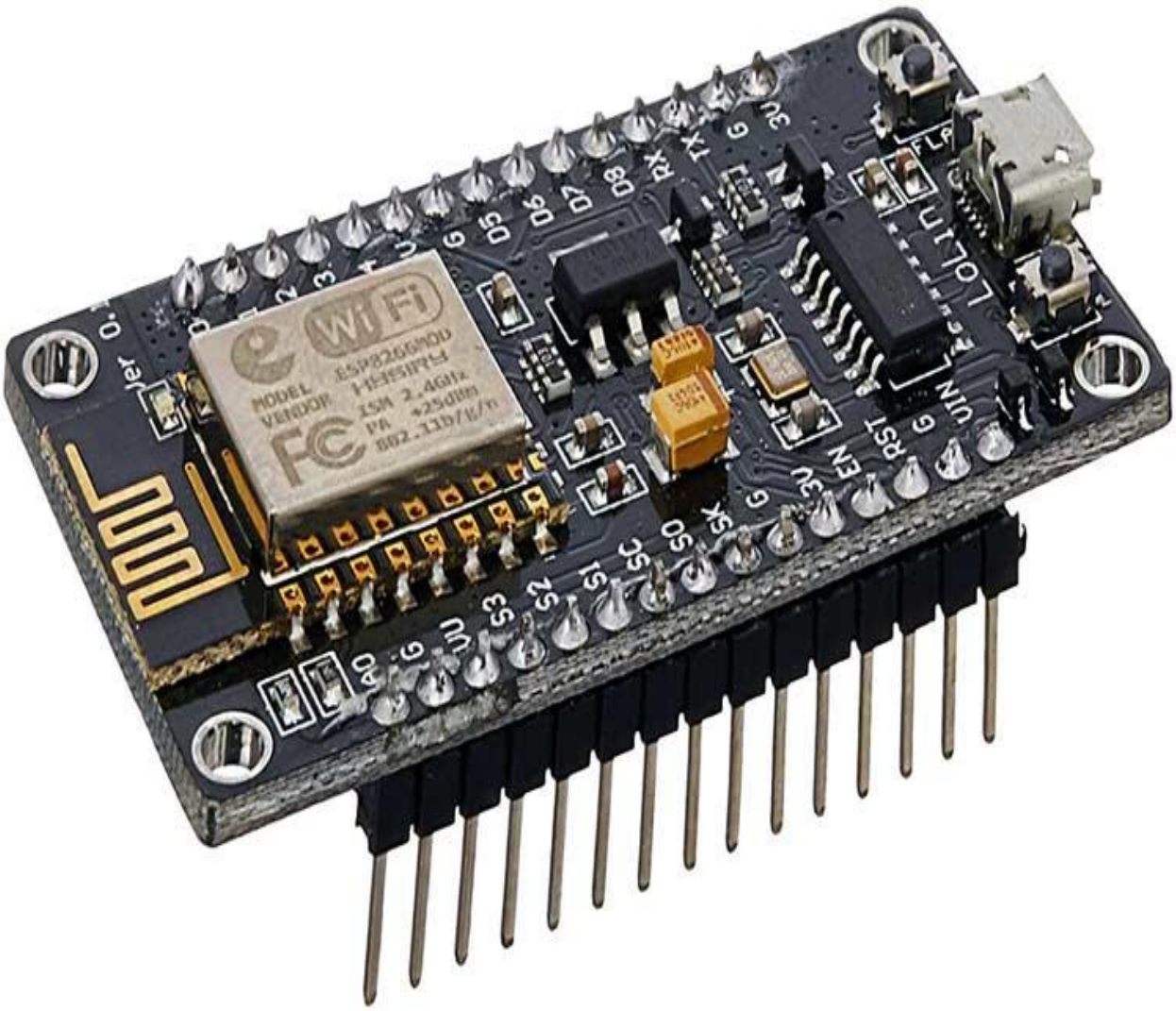


Figure 5.3 ESP 8266

Specifications

- **Voltage:**3.3V.
- **Wi-Fi Direct (P2P):** soft-AP.
- **Current consumption:** 10uA~170mA.
- **Flash memory attachable:** 16MB max (512K normal).

- Integrated TCP/IP protocol stack.
- **Processor:** Tensilica L106 32-bit.
- **Processor speed:** 80~160MHz.
- **RAM:** 32K + 80K. • GPIOs: 17 (multiplexed with other functions).
- **Analog to Digital:** 1 input with 1024 step resolution.
- +19.5dBm output power in 802.11b mode
- **802.11 support:** b/g/n.
- **Maximum concurrent TCP connections:** 5.

Pin Description

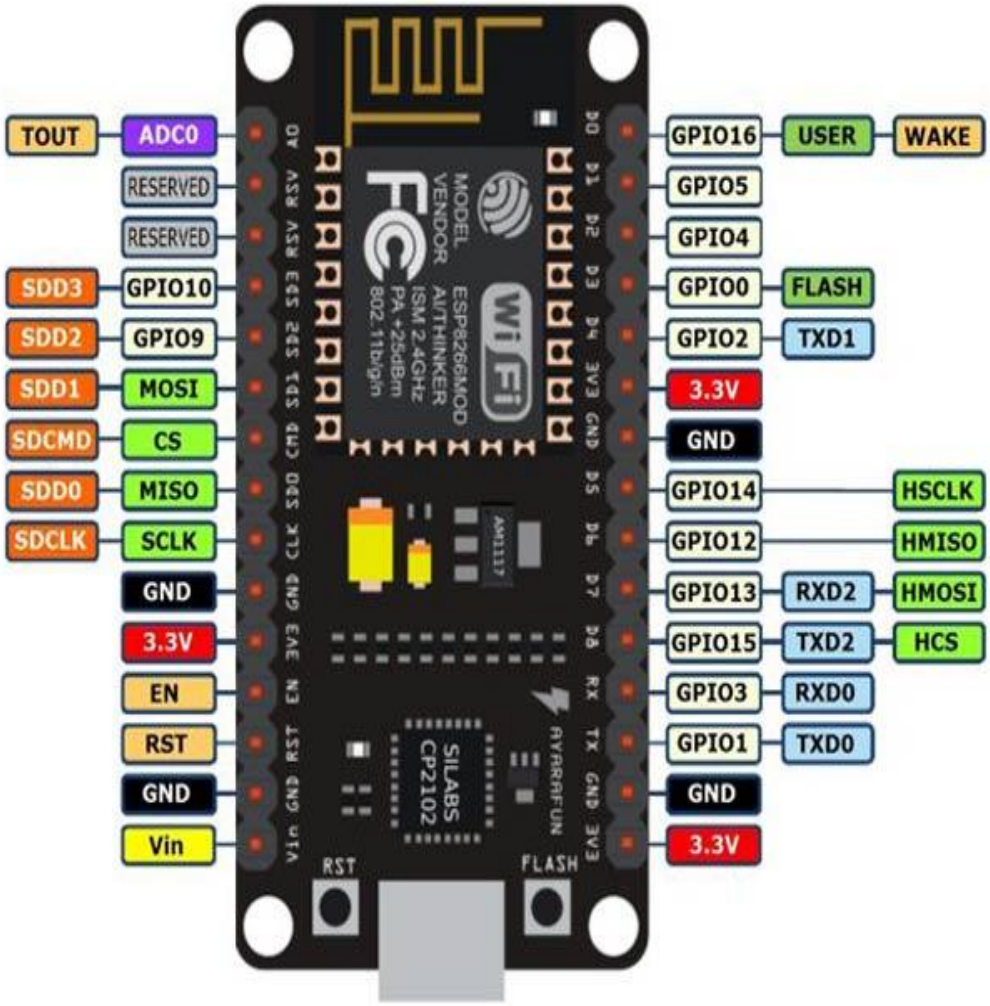


Fig 5.4 Pin Description

USING ARDUINO IDE

The most basic way to use the ESP 8266module is to use serial commands, as the chip is basically a Wi-Fi/Serial transceiver. However, this is not convenient. What we recommend is using the very cool ESP 8266 project, which is a modified version of the Arduino IDE that you need to install on your computer. This makes it very convenient to use the Arduino chip as we will be using the well-known Arduino IDE. Following the below step to install Arduino library to work in Arduino IDE environment.

ESP 8266 And Arduino IDE

ESP 8266 is an **eLua** based firmware for the **ESP 8266 Wi-Fi SOC from Espressif**. The firmware is based on the **Espressif NON-OS SDK** and uses a file system based on **spiffs**. The code repository consists of 98.1% C-code.

The ESP 8266 firmware is a companion project to the popular ESP 8266 dev kits, ready-made open source development boards with ESP 8266 chips.

ESP 8266 Arduino Core

As Arduino.cc began developing new MCU boards based on non-AVR processors like the ARM/SAM MCU and used in the Arduino Due, they needed to modify the Arduino IDE so that it would be relatively easy to change the IDE to support alternate tool chains to allow Arduino C/C++ to be compiled down to these new processors.

Some creative ESP 8266 enthusiasts have developed an Arduino core for the ESP 8266 Wi-Fi SoC that is available at the **GitHub ESP 8266 Core webpage**. This is what is popularly called the "ESP 8266 Core for the Arduino IDE" and it has become one of the leading software development platforms for the various Arduino based modules and development boards, including ESP 8266.

Advantages Of ESP 8266 Platform Relative To Arduino

- Low cost
- Integrated support for WIFI network
- Reduced size of the board
- Low energy consumption

Disadvantages

- Need to learn a new language and IDE
- Reduced pin out
- Scarce documentation

Getting Started With ESP 8266

Before using the IDE, install the USB driver in the operating system, which may be the CH340 or CP2102, depending on the board version.

Arduino IDE

- Before using the IDE, install the USB driver in the operating system, which may be the CH340 or CP2102, depending on the board version.
- Run the Arduino IDE. If it has not been installed, do so here: https://www.arduino.cc/en/Main/Software
- Open the Preferences window and type in the "Additional Board Manager URLs" the following address:
- http://arduino.ESP 8266.com/stable/package_Arduino_ESP 8266_index.json
- Open Board Manager menu and type ESP 8266 and install.
- On the Tools menu, configure your board according to the model you are using. The most common options are:

Board: ESP 8266, according to its model

CPU Frequency: 80 MHz

Upload Speed: 115200

Note: When you use the Arduino board with the Arduino IDE, the FCK firmware will be deleted and replaced by the sketch. If you want to use the FCK SDK again, will be necessary "flashing" the firmware again.

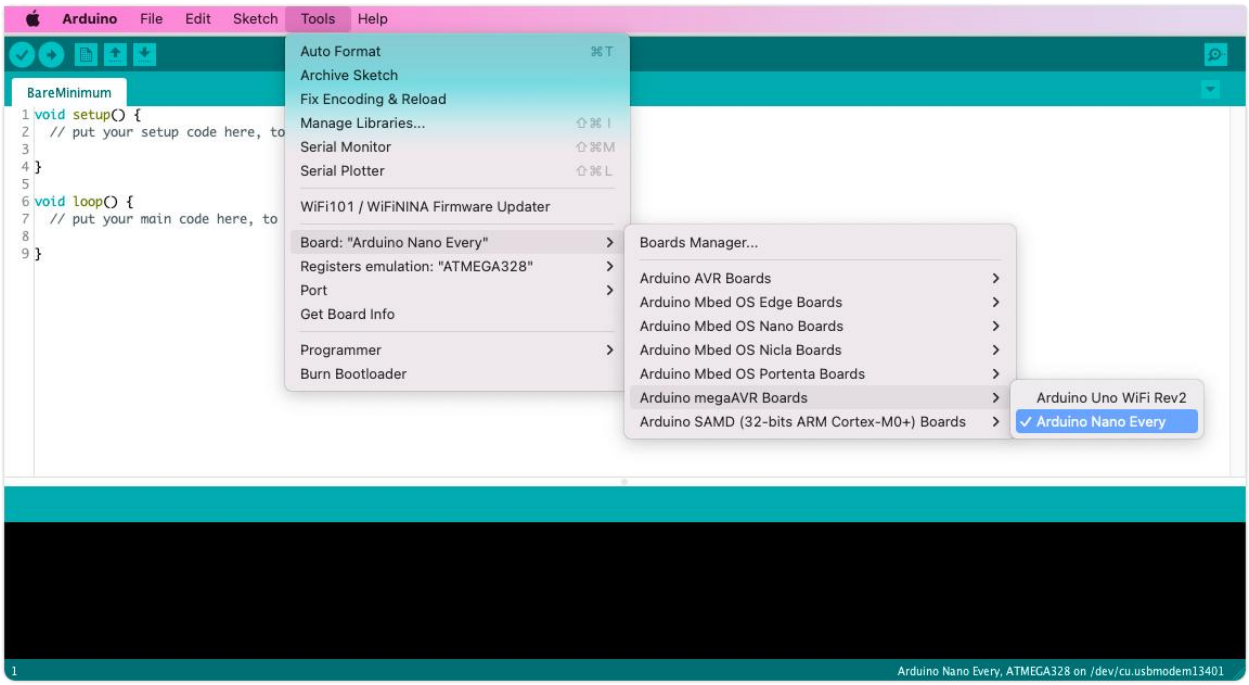


Fig 5.5 Arduino IDE

ESP 8266 Spiffs File System

Arduino plug-in which packs sketch data folder into SPIFFS file system image, and uploads the image to ESP 8266 flash memory.

Installation

- Make sure you use one of the supported versions of Arduino IDE and have ESP32 core installed.
- Download the tool archive from **releases page**.
- In your Arduino sketchbook directory, create tools directory if it doesn't exist yet.
- Unpack the tool into tools directory (the path will look like<home_dir>/Arduino/tools/ESP 8266FS/tool/esp32fs.jar).

Usage

- Open a sketch (or create a new one and save it).

- Go to sketch directory (choose Sketch > Show Sketch Folder).
- Create a directory named data and any files you want in the file system there.
- Make sure you have selected a board, port, and closed Serial Monitor.
- Select Tools > Arduino Sketch Data Uploadmenu item. This should start uploading the files into Arduino flash file system. When done, IDE status bar will display SPIFFS Image Uploaded message. Might take a few minutes for large file system sizes.

Example Code

Arduino Like Io Access

```
Pin = 1
gpio.mode(pin,gpio.OUTPUT)
gpio.write(pin,gpio.HIGH)
gpio.mode(pin,gpio.INPUT)
print(gpio.read(pin))
</pre>
```

Blinking LED

<pre class=”lang:default decode:true” title=”BootReceiver.class”> lighton=0 tmr.alarm(0,1000,1,function() if lighton==0 then lighton=1 led(512,512,512) — 512/1024, 50% duty cycle else lighton=0 led(0,0,0) end end)

Connect To Mqtt Broker

```
-- init mqtt client with keep alive timer 120sec
m = mqtt.Client("clientid", 120, "user", "password")
-- setup Last Will and Testament (optional)
-- Broker will publish a message with qos = 0, retain = 0, data = "offline"
-- to topic "/lwt" if client don't send keep alive packet
m:lwt("/lwt", "offline", 0, 0)
m:on("connect", function(con) print ("connected") end)
m: on("offline", function(con) print ("offline") end)
-- on publish message receive event
```

```
m: on ("message", function (conn, topic, data)
print (topic .. ":" )
if data ~= nil then
print (data)
end
end)
-- for secure: m: connect ("192.168.11.118", 1880, 1)
m: connect ("192.168.11.118", 1880, 0, function (conn) print ("connected") end)
-- subscribe topic with qos = 0
m: subscribe ("/topic", 0, function (conn) print ("subscribe success") end)
-- or subscribe multiple topic (topic/0, qos = 0; topic/1, qos = 1; topic2, qos = 2)
-- m: subscribe ({["topic/0"] =0, ["topic/1"] =1, topic2=2}, function (conn) print ("subscribe success") end)
-- publish a message with data = hello, Qos = 0, retain = 0
m: publish ("/topic","hello", 0, 0, function (conn) print ("sent") end)
m: close ();
-- you can call m: connect again
```

Spiffs File System

```
#include"FS.h"
void setup () {
Serial. begin (115200);
bool ok = SPIFFS.begin ();
if (ok) {
Serial.println ("ok");
boo exist = SPIFFS. Exists ("/index.html");
if (exist) {
Serial.println ("The file exists!");
File f = SPIFFS. Open ("/index.html", "r");
if (! f) {
Serial.println ("Some thing went wrong trying to open the file...");
```

```
}
else {
int s = f.size ();
Serial.printf ("Size=%d\r\n", s);
// USE THIS DATA VARIABLE
String data = f.readString ();
Serial.println (data);
f.close ();
}
}
else {
Serial.println ("No such file found.");
}
}
}
void loop () {
// put your main code here, to run repeatedly :}
```

Functional Diagram Of ESP 8266

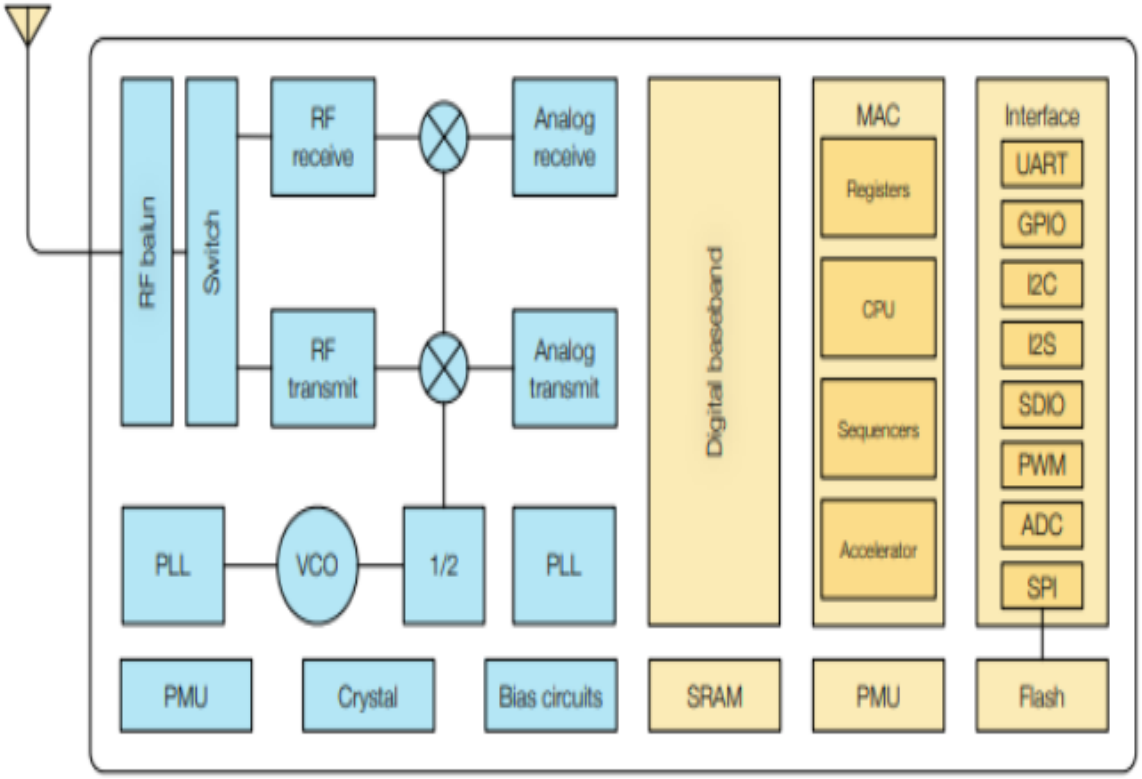


Figure 5.6 Functional Diagram Of ESP 8266

Pinout Description

Figure 5.7 ESP 8266 Pin

Pinout Definition

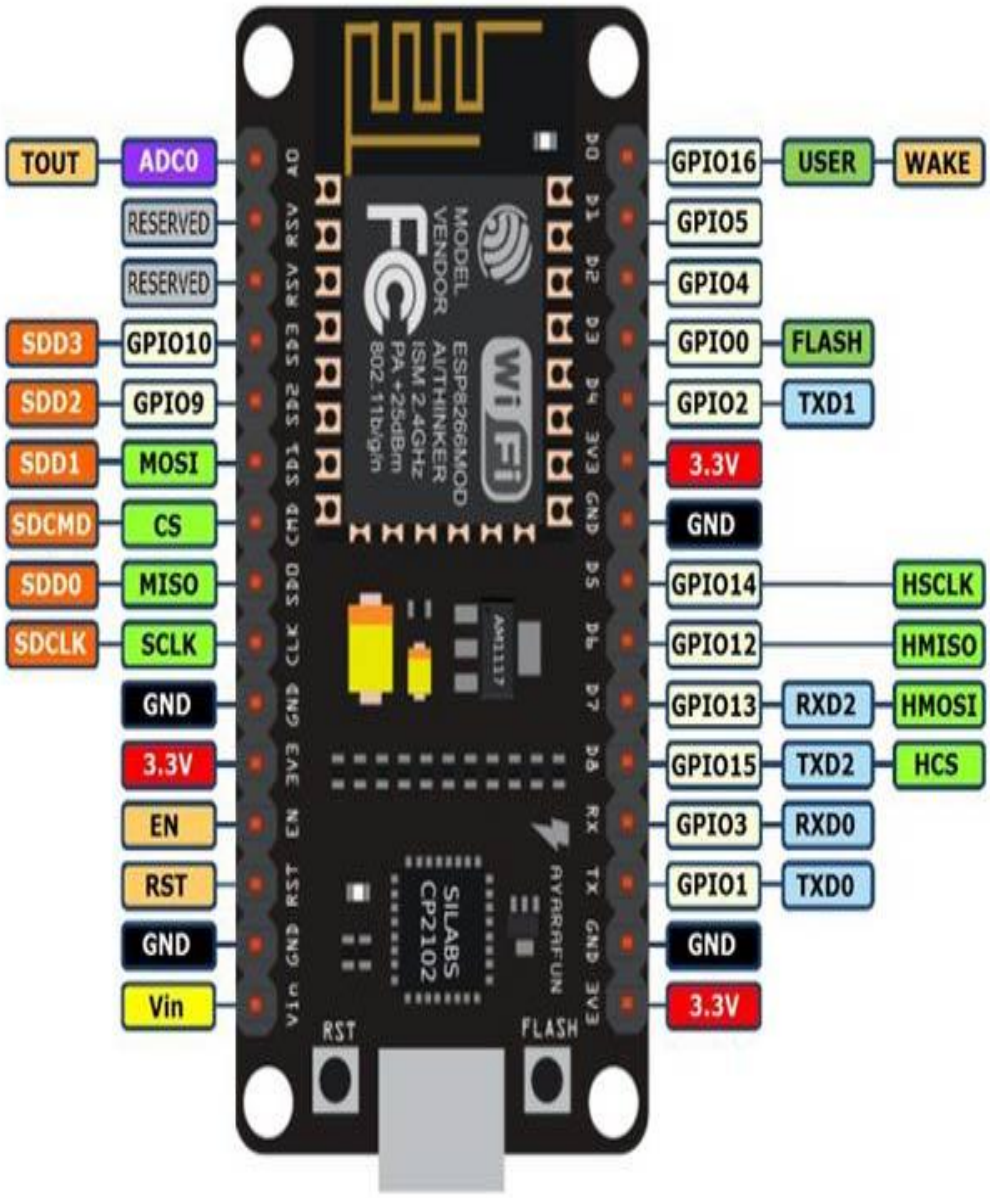


Fig.5.8 Pinout Definition

Pin No	Symbol	Level	Description
1	Vss	0 V	Ground
2	V_{DD}	5.0V	Supply voltage for logic
3	V o	variable	Operating voltagefor LCD
4	RS	H/L	H data/L instruction code
5	R/W	H/L	H/Read (MPU-module) L/write (MPU-module)
6	E	H, H-L	Chip enable signal
7	DB0	H/L	Data bit 0
8	DB1	H/L	Data bit 1
9	DB2	H/L	Data bit 2
10	DB3	H/L	Data bit 3
11	DB4	H/L	Data bit 4
12	DB5	H/L	Data bit 5
13	DB6	H/L	Data bit 6
14	DB 7	H/L	Data bit 7
15	A		Power supply for LED backlight (+)
16	K		Power supply for LED backlight (-)

Table 5.1 Pin out Definition

ESP 8366 Core

As Arduino.cc began developing new MCU boards based on non-AVR processors like it would be relatively easy to change the IDE to support alternate tool chains to allow Arduino C/C++ to be compiled for these new processors. They did this with the introduction of the Board Manager and the SAM Core. A "core" is the collection of software components required by the Board Manager and the Arduino IDE to compile an Arduino C/C++ source file for the target MCU's machine language. Some ESP 8266 enthusiasts developed an Arduino core for the ARM/SAM MCU and used in the Arduino Due, they needed to modify the Arduino IDE so that ESP 8266 Wi-Fi SoC, popularly called the "ESP 8266 Core for the Arduino IDE".[16] This has become a leading software development platform for the various ESP 8266-based modules and development boards, including NANO.

5.1.2 ULTRASONIC SENSOR

Fig 5.9 Ultrasonic sensor

Description

Ultrasonic sensing is one of the best ways to sense proximity and detect levels with high reliability.

Our technical support gets emails all of the time about how our sensors work and what environments our sensors work (or don't work) in.

This guide was created as an introduction to ultrasonic sensing, it's principles, and **how ultrasonic sensors work** in your applications.

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves.

An ultrasonic sensor uses a transducer to send and receive ultrasonic pulses that relay back information about an object's proximity.

High-frequency sound waves reflect from boundaries to produce distinct echo patterns.

Features

When using multiple sensors in an application, it's important to connect them in a way that will allow you to avoid issues like crosstalk or any other interference.

To prevent the disruption of the ultrasonic signals coming from your sensor, it's important to keep the face of the ultrasonic transducer clear of any obstructions.

Common obstructions include:

- Dirt
- Snow
- Ice
- Other Condensation

For this particular use case, we offer our Self Cleaning sensors.

They are intended specifically for applications requiring the resistance of condensation in high moisture environments, our self-cleaning function is designed to run continuously in order for the self-cleaning feature to be active.

Block Diagram

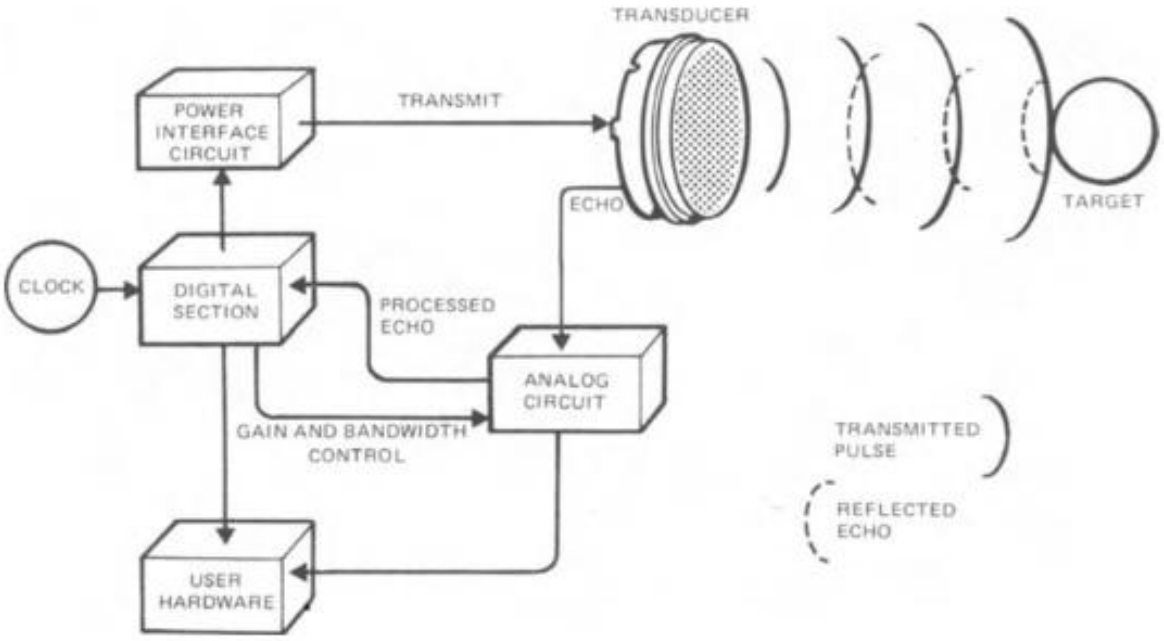


Figure 5.10 Block diagram of UV

5.1.3.Ultra voilet Led

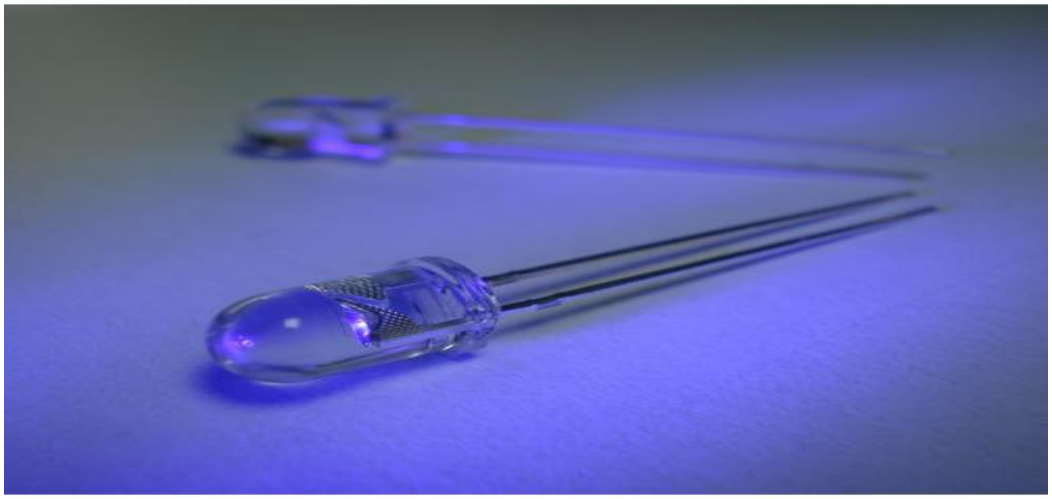


Fig 5.12 LED UV

Description

UV LEDs are more environmentally friendly as they do not contain harmful mercury, do not produce ozone, and consume less energy. Use of UV-C LEDs is rapidly growing in applications such as germicidal (UVGI) for air, surface and water purification.

5.1.4. Bread Board

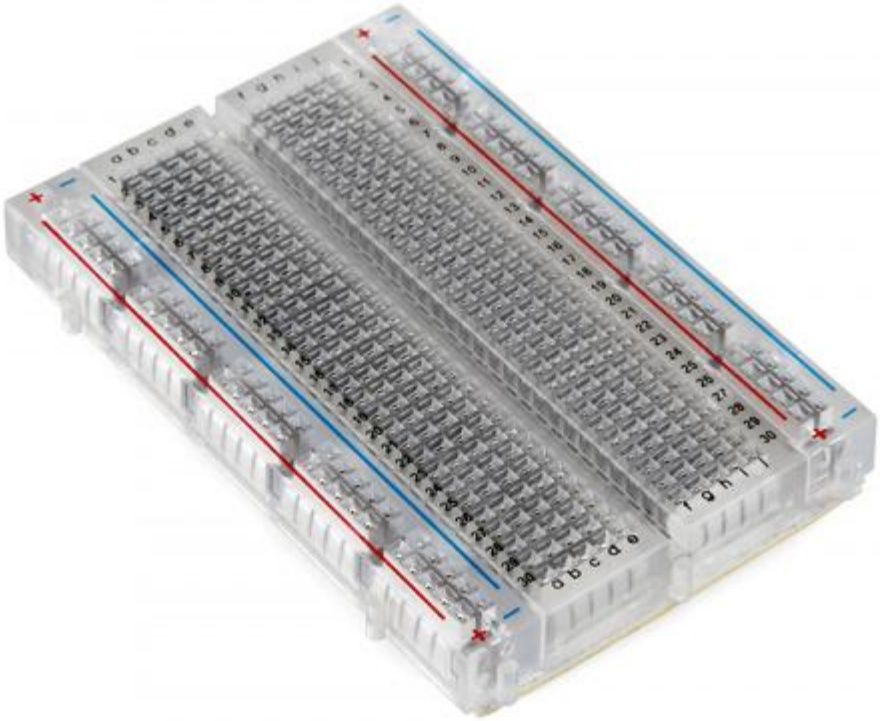


Fig 5.13 Bread Board

Breadboards are one of the most fundamental pieces when learning how to build circuits. In this tutorial, you will learn a little bit about what breadboards are, why they are called breadboards, and how to use one. Once you are done you should have a basic understanding of how breadboards work and be able to build a basic circuit on a breadboard.

5.1.5.Jumper Wire

Fig 5.14 Jumper wire

A jump wire (also known as jumper, jumper wire, DuPont 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.[1]

Individual jump wires are fitted by inserting their "end connectors" into the slots provided in a breadboard, the header connector of a circuit board, or a piece of test equipment.

5.1.7. App Interface

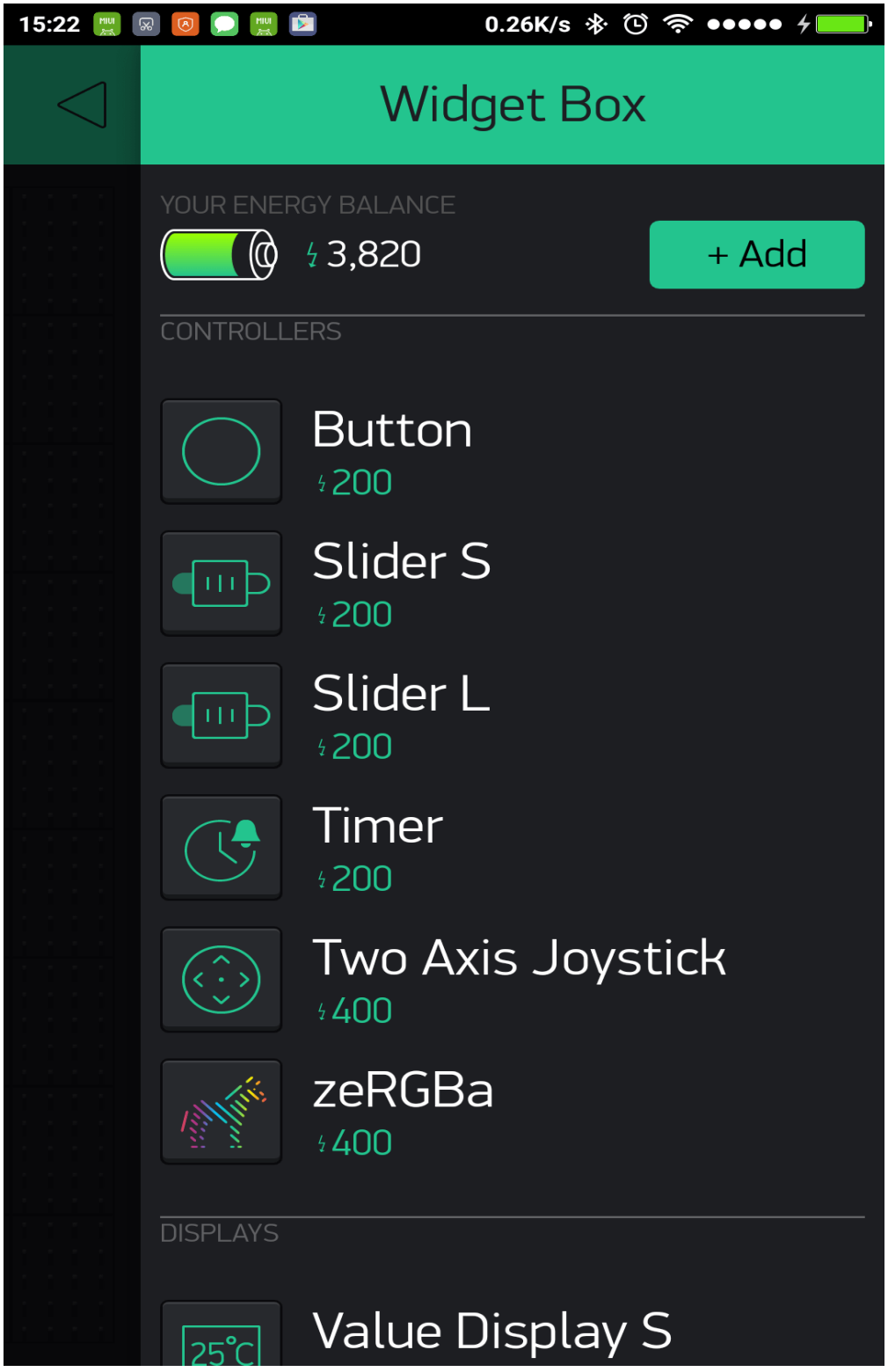


Fig 5.16 Blynk Interface

Blynk was designed for the Internet of Things. It can control hardware remotely, it can display sensor data, it can store data, vizualize it and do many other cool things.

There are three major components in the platform:

- **Blynk App** - allows to you create amazing interfaces for your projects using various widgets we provide.
- **Blynk Server** - responsible for all the communications between the smartphone and hardware. You can use our Blynk Cloud or run your private Blynk server locally. It's open-source, could easily handle thousands of devices and can even be launched on a Raspberry Pi.
- **Blynk Libraries** - for all the popular hardware platforms - enable communication with the server and process all the incoming and outcoming commands Now imagine: every time you press a Button in the Blynk app, the message travels to the Blynk Cloud, where it magically finds its way to your hardware. It works the same in the opposite direction and everything happens in a blynk of an eye.

5.2 FLOW CHART

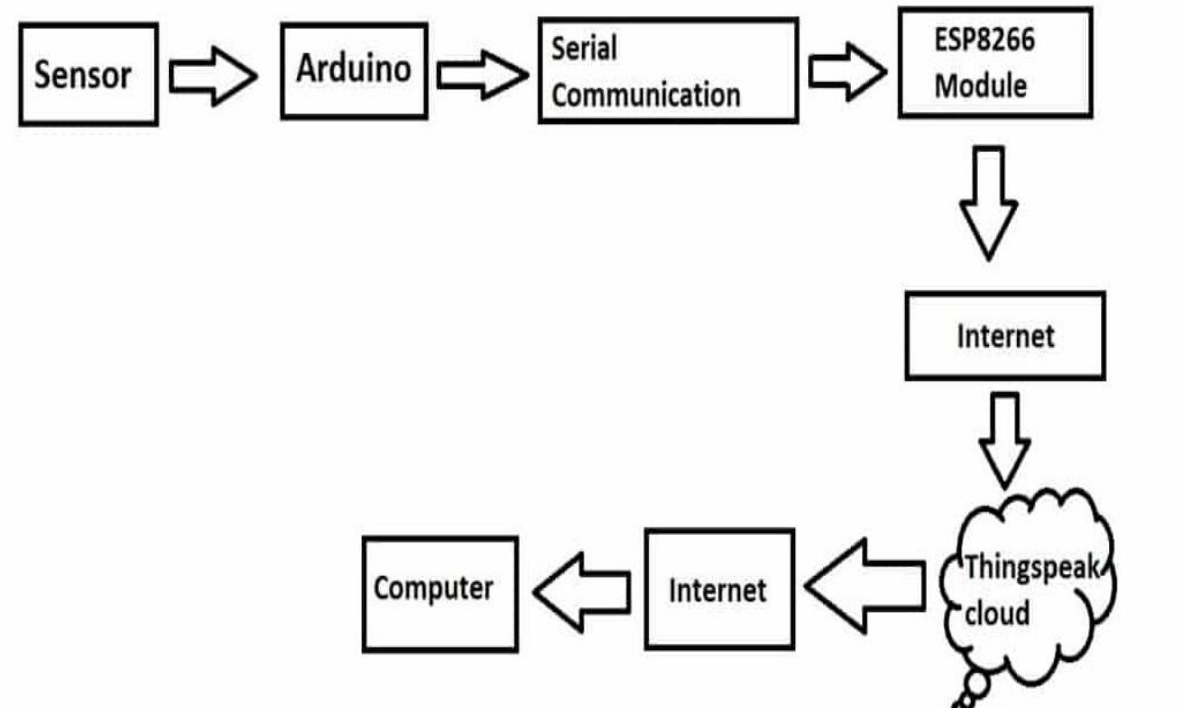


Fig 5.17 Flowchart of the project

Algorithm

- Start
- The sensor senses the change and convert it into electrical reading.
- The electrical reading is send for a ardino architecture.
- That arduino send the signal to the seriel communication interface.
- The serial communication signal is processed in the micro controller ESP 8266.
- That processed output is send to the internet .
- By travelling in the internet that data drops to Cloud system
- The cloud system send data for a device with authorized IP Address .
- Once identified the data is sedn via internet to the host device.
- Finally it comes to our mobile appliation.

CHAPTER 6

APPENDICES

6.1 SAMPLE SCREEN

Fig 6.1 App Interface

6.1.1 SAMPLE LOADING

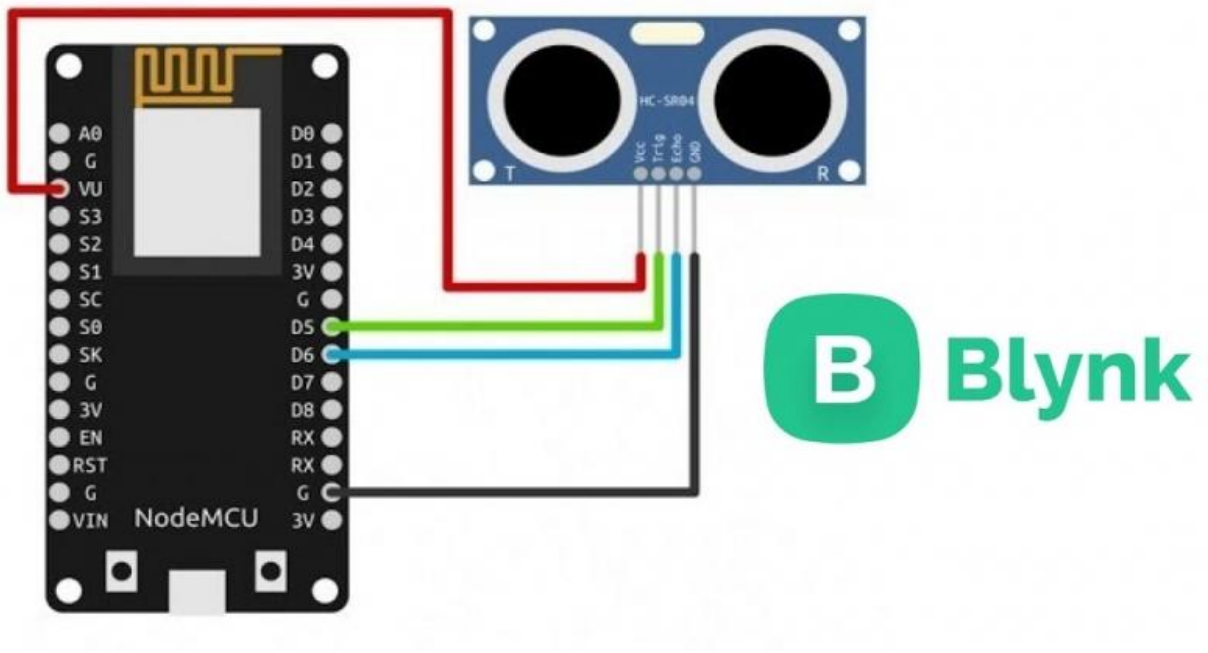


Fig 6.2 Kit Diagram

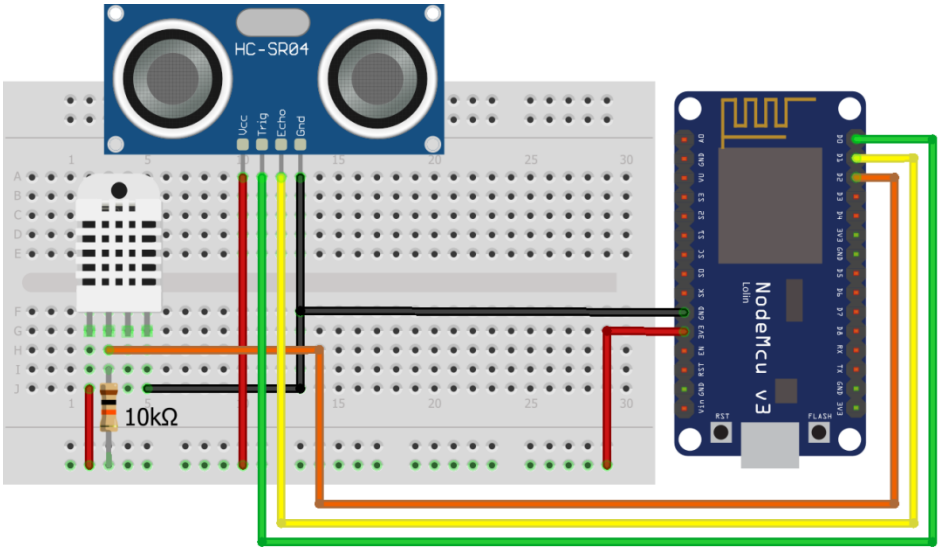


Fig 6.3 KIT Sketch

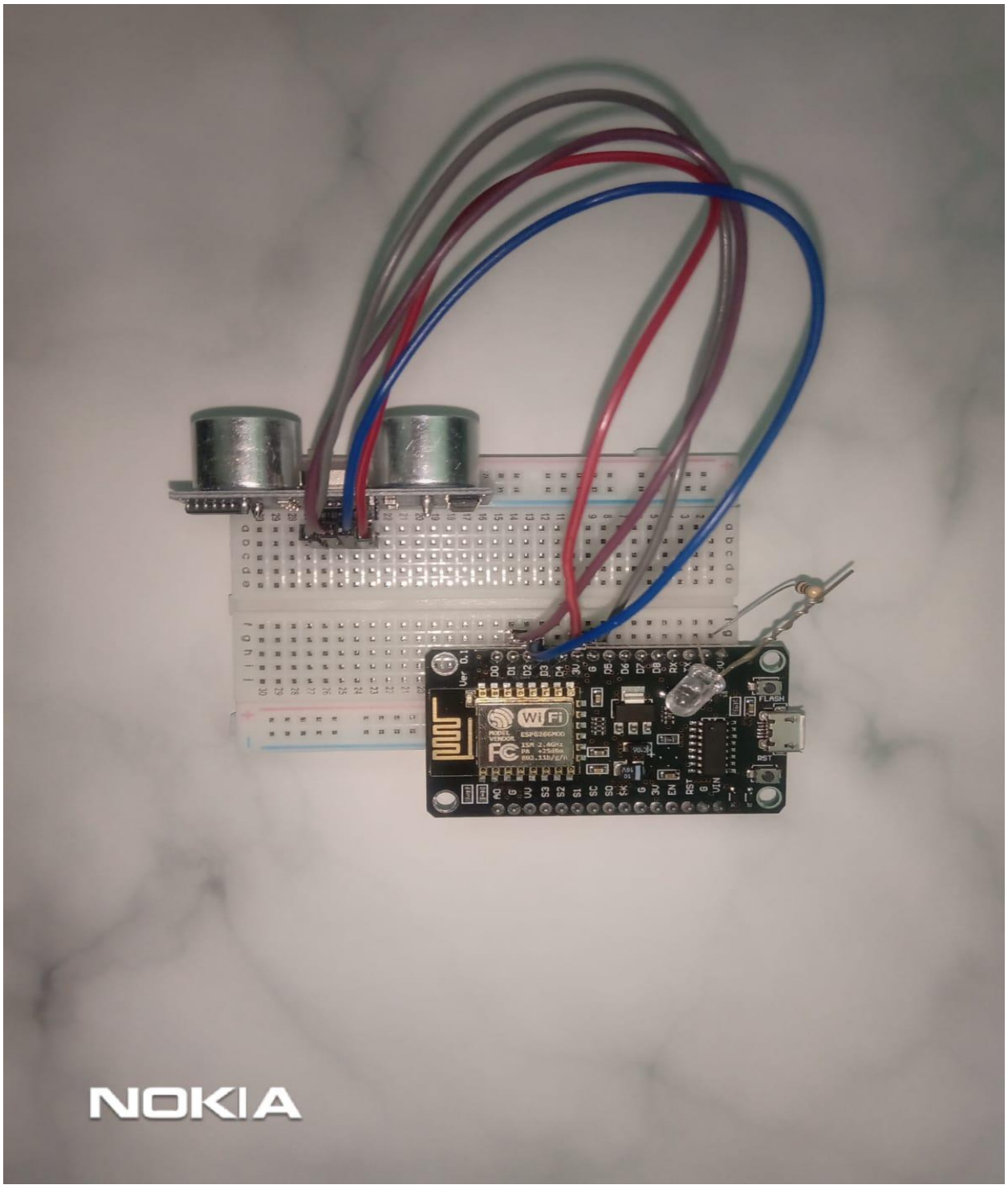


Fig 6.4 Kit Image

6.2CONCLUSION

Hereby we come to the end of our project — Furnance Monitoring System,

- This IOT enabled furnance monitoring system is a detication to th industrial workers who work in smelting indystry .

6.2.1 FUTURE EXPANSION

- Keep it in a best case help to prevent damage of the kit
- Low cost build for future.
- Easy to use proceduce with smart watch compact
- Eliminate life loss procedure endurance.

6.3.BIBLIOGRAPHY

- Programming in ANSI C: E BALAGURUSAMY
- Arduino book for begginers
- https://developer.arduino_librsary.com
- https://developer.nodemcu.com
- Programming in ANSI C: E BALAGURUSAMY
- ESP32: Programming NodeMCU Using Arduino IDE:UpSkill Learning
- “The Internet of Things” by Samuel Greengard
- “The Fourth Industrial Revolution” by Klaus Schwab
- “Getting started with Internet of Things” by Cuno Pfister
- “Learning Internet of Things” by Peter Waher
- “Precision: Principles, Practices and Solutions for the Internet of Things” by Timothy Chou
- “The Second Machine Age: Work, Progress and Prosperity in a Time of Brilliant Technologies” by Erik Brynjolfsson and Andrew McAfee
- “The Silent Intelligence” by Daniel Kellmereit and Daniel Obodovsk